Dissertation

Postures Detection with
OpenMovement Inertial Measurement Units

Student: Danh Thanh Nguyen
Supervisor: Patrick Olivier

School of Computing Science
Newcastle University, United Kingdom
May 2016

Abstract

Activity recognition is an important branch of pervasive computing. The activity recognition aims to recognise actions of agents through the learning of series of signals obtained from them. Human acceleration has been used extensively as a means to express actions' characteristics. Developed at Newcastle University, the OpenMovement Inertial Measurement Units (IMU) are the ideal platform for capturing human activities' properties. The project aims to use the data captured by the WAX9, which is a product of the OpenMovement IMU family to develop an algorithm that is able to detect a number of common postures.

Declaration

"I declare that this document is my own work except where otherwise stated."

Danh Thanh Nguyen

Acknowledgements

First and foremost, I would like to shown my endless gratitude to my parents for their eternal love, support and encouragement.

I also would like to show my gratitude to Professor Patrick Olivier at Open Lab, Newcastle University, who offered me the opportunity to conduct this project, for his guidance throughout the project. Also to Nils Hammerla, Thomas Ploetz and Daniel Jackson who gave me valuable advices and experiences about machine learning and embedded systems.

Finally, I would like to thank Ross Quinlan whose works have given me a lot of inspirations.

Table of contents

1. Introduction

1.1. Context

At the heart of human social development lies the ability to exchange information. In daily life, people communicate by using verbal and non-verbal expression. Non-verbal expressions are demonstrated through facial expressions, the gaze, hand gestures, body movements, and so on. For example, a person could greet someone by waving a hand, smiling etc. Therefore, it is important to study recognition of non-verbal expressions, or activity recognition in order to establish a natural interface.

Activity recognition is an important research branch of pervasive computing. The goal of activity recognition is to recognise the actions and goals performed by agents based on a series of signal captured from the agents as well as environmental conditions [1].

Research so far mainly focuses on recognizing "which" activity is being performed at a specific point in time [1]. Successful researches to date have mainly focused on recognising simple activities. On the other hand, complex human activities have still been remaining an active and challenging area of research [2].

Over the years many applications have been studied and developed in order to tackle numerous problems on a wide range of areas such as healthcare, security, entertainment etc. Specifically, a number of end-to-end applications, especially in health monitoring and navigation for people with cognitive disability. Furthermore, dedicated hardware products such as the Multi-modal Sensor board built at Intel Research Seattle and University of Washington have facilitated the exploration and evaluation as to how activity data could be recorded and inferred. Researchers have also shown the possibility of recognising many daily physical activities such as sitting, walking. Various specialised techniques have been employed and will be discussed in the next sections [3].

Recognising daily activities plays an important role in evaluating the quality of life both in the elderly and patients with mobility disabilities. This includes detecting postural transitions e.g. sitting to standing and vice versa, standing to walking as well as detecting activities such as walking, running.

It is important to understand this "important role" via an example.

Let us consider the following scenario: An elderly man who stays alone wakes up at dawn in his small apartment. He turns on the stove to make some tea, switches on the toaster oven and gets some bread and cereal from the cupboard. He then takes his morning medication after which a machine-generated voice alerts him to turn off the oven. Later that day, his daughter accesses a website where she scans a check-list, which was created by a sensor network in her father's apartment. She finds that her father is doing very well, taking his medicine on schedule, and continuing to manage his daily life on his own. That information puts her mind at ease [4].

There have been a number of ways as to how to obtain the input data for

the activity recognition process. Likewise, there are also different kinds of data which can be used for this purpose. Specifically, they can be divided into two categories: visual and digital data. Visual input focuses on the ability of perceiving and recognising activities using the visualisation of the agents which is quite similar to the way humans interpret activities. These include images, videos input etc. The other category of input could normally be considered digital input since they contain values for partially invisible digital features which are sometimes omitted or impossible for humans to perceive such as acceleration, velocity, temperature.

Many past works have demonstrated 85% to 95% rates of recognition for ambulation, postures as well as other activities using accelerometer's data [5]. In this project, digital data is collected. In order to obtain and observe agents' actions, some data logging device is required. Over the years, it has been shown that acceleration data has remained one of the most useful and commonly used approach. On the other hand, the selection of device must take many factors into account. For example, apart from being able to guarantee the accuracy of the data captured, the device must ensure users' satisfaction. It is clear that many users would not want to wear a bulky set of movement capturing system that could potentially affect the movements as well as their "natural" characteristics. Therefore, there is a need for neat, compact embedded devices which users can wear comfortably at some common locations on the body. A body worn Inertial Measurement Unit that is capable of capturing three dimensional acceleration data such as AX3, WAX9 and AX9, developed under OpenMovement hardware platform by Axivity could be considered an ideal candidate for this problem.

1.2. Aims and objectives

1.2.1. Aims

The device location on the body also plays an important role in the accuracy of an activity recognising algorithm. It is shown that one of the most commonly used location on the body is the waist since it is very close to the centre of mass of human body thus displaying the most representative part of human movement [6].

The project aims to perform evaluation on various candidate features of which the most appropriate ones are to be chosen to be the target for an algorithm that is optimised for devices worn at waist positions for the Inertial Measurement Units developed under OpenMovement platform that is capable of detecting the current status of the user and potentially simple activities such as walking, standing and so on using accelerometer tri-axis data.

By using this algorithm, it is another indirect goal that devices will be able to perform contextual triggering, that is, the devices will be able to realise when the user is in a "rest" status such as sitting, lying and when he/she is in an active status such as walking, running. Being able to beware of this fact, the devices can then decide when would be a good time to turn off some unnecessary sensor components such as gyroscope whose use would require a significant part of battery, practically saving some energy and prolonging the battery's lifespan. The project therefore aims to provide a discussion on the feasibility of this

algorithm on deployment.

1.2.2. Objectives

Objective 1:

Produce an algorithm optimised for the lower back position that is capable of detecting basic human postures: standing, walking, lying given a three dimensional acceleration data as the primary input.

Objective 2:

Demonstrate the accuracy, efficiency and feasibility of the produced algorithm by:

1. Produce a semi-automated implementation on a simulated environment in which the targeted device's specifications are taken into consideration.

2. Perform experiments and evaluation on the produced algorithm and implementation.

3. Produce a discussion on the feasibility of the algorithm's implementation in the deployment to the actual sensor devices.

2. Relevant work and research

In the following section, targeted activity classes imply activities are targeted for classification.

Note: *The terms classifier, classification process and classification are used interchangeably, all of which imply the process in which attempts are made in classifying some given data samples. The terms attribute and feature are used interchangeably to imply the information units extracted from the signal. These terms will be explained in greater details in the next section.*

2.1. Challenges

2.1.1. Research challenges

Human activity data bears some very unique characteristics. This uniqueness often leads to various challenges on which researches have been extensively conducted.

Since activity data could potentially provide information about the identity of a person, their personality as well psychological state, it is therefore difficult to extract [7]. There exists no concrete taxonomy for any kind of activities even though some general ones do exist [1][3]. This non-stand problem also presents other two problems: intra-class variability and interclass similarity.

The problem of intra-class variability can be understood that an activity class when performed at different times could produce different signals in terms of magnitude and frequency. Such difference exists because the activity class might be performed by different individuals. The variability nevertheless could also occur within the context of the same individual since there are many factors that can affect the performance of the activity such as stress, mood, emotional state. For example, the signal obtained when a person performs a walk could vary depending on the times of the day e.g. the walking style could be more active in the morning compared to the evening. [1]

On the other hand, the problem of interclass similarity implies that various distinct activities e.g. raising hand, playing badminton could produce similar signals which could potentially confuse the classification system. For example, signal data produced by the action of waving a hand could prove to be quite similar to that of playing badminton or making a cup of coffee.

The problem of interclass similarity could further lead to another challenge called NULL class. Given that the signal is often recorded continuously in long term behavioural monitoring, it is likely that only a small fraction of the whole signal is relevant. Other activities can nevertheless produce the similar patterns which can easily be confused with the relevant data. The imbalance between the amount of relevant signals and irrelevant signals could sometimes be referred as the class imbalance problem.

Apart from the challenges originated from the nature of human activity data itself, recording activity data nevertheless is another challenging task.

The process of collecting and classifying training data, often called ground truth

annotation could be a tedious and expensive in terms of labour as the annotator needs to either perform the annotation in real time or scan through the data and manually label all of the relevant activities. This could furthermore be challenging when it comes to digital movement data such as acceleration, temperature [1].

Lastly, it requires a lot of planning in order to conduct a proper human activity recognition experiment. There exists a trade-off between unobtrusiveness and the efficiency of the sensors; the time required to prepare, conduct, and maintain the experiment as well as the logistics and expenses for participants and equipment.

2.1.2 Application challenges

With all the problems that come from the nature of human activity data set aside, it is also very important to learn about the sensing equipment, which is the entry to the activity recognition experiments.

Sensors might cause a significant variability to the output signal data due to internal and external causes. Examples of internal causes include hardware failures, clock drifting and so on. On the other hand, the displacement of sensors from their intended positions i.e. sudden high level movements could provide a false illusion to the class of the activity as a result of drifted signal data.

The captured data is also likely to be affected by noise, therefore it is advantageous to have a reliable noise filtering process.

It is important to point out that there also exists another trade-off between the granularity of data collected and the processing power of the conducting devices. This trade-off is proved in the context of real time behavioural monitoring: Lightweight devices help make the process much faster by reducing the granularity of data, thus improving the system latency. On the contrary, more powerful sensors could provide a better level of granularity, allowing activities to be further analysed and inferred. However, the cost of that could be a growing dataset which could potentially make it harder for the system to keep up with the latest user's behaviours

2.1.3. Design and implementation challenges

Another problem which could be subject to the trade-off mentioned above is the design of the system. In a real-time behavioural monitoring environment in which time is about important as the classification performance of the algorithm, it is a challenging task to select the appropriate classification techniques. Many techniques could offer a very good performance in terms of accuracy, however, it is undeniable that with a growingly complex algorithm comes the cost of calculation and processing power. It is therefore crucial to evaluate these techniques in great depth.

Next, let us introduce features, an important concept of activity recognition. Features, in the context of activity recognition are a set of extracted information from the input signal which carries different characteristics of the signal and are used to classify activity classes. There is a wide range of options for activity

feature, each of which has their own advantages and disadvantages. Knowing which feature would potentially yield better results is essential in the making of the algorithm. The feature selection process therefore requires a well-planned and well-analysed implementation. Furthermore, the accelerometer data is not distributed independently and identically which could potentially hinder the analysis process [8].

In order to obtain data from human bodies, appropriate sensor devices are required. As mentioned earlier, IMUs are the ideal platform for this task. Let us now get more specific on the target devices. IMUs operates continually until it runs out of charge. This means the devices work even when the user is not performing any "dynamic" activity which would eventually lead to the battery running out faster. Understanding when to trigger the device's other sensors i.e. gyroscope and when to turn on stand-by is essential in keeping the device working for longer.

2.2. Human Activity Recognition Chain

Activity recognition, particularly using body-worn sensors is a multi-stage process. There have been several works conducted that provide an overall scheme of design and sometimes implementation on this area. However, these works are conducted towards different aspects of the problem of activity recognition. For example, in 2004 Bao and Intille proposed a solution that suggests the use of unsupervised data recording, or user-annotated data [5]. On another hand, in 2002 Laerhoven proposed a solution using multi-sensor networks as a means to experiment the changes they make on performance [9]. The problem with these works is that, despite giving a very promising results, they don't have a unified perspective. Therefore, it is crucial to have a collectively common rules that govern how body-worn sensors based activity recognition should be laid out and deployed. In 2006 Bulling proposed a solution to this by introducing a process called Activity Recognition Chain which describes how activity recognition could be conducted [1]. The chain, focusing on supervised learning, consists of a number of steps that are to be carried out in chronological order. It also forms the foundation for the design of the project. Let us make a brief description about the process.

Step 1. Collecting data

In the first step of the process, raw data is collected from sensors which may be located at different locations on the body of the bearer. In advanced human activity recognition environments, it is possible to have sensors set up in the experimenting environment in order to collect additional data which could be useful in determining activity classes better or in more sophisticated levels of activity class. For examples, the Ambient Kitchen project, conducted in 2007-2010 by Digital Interaction (now Open Lab), Newcastle University provided several environment-based sensors e.g. pressure-sensitive floor, wireless accelerometers that are embedded into specially designed kitchen accessories through which users' behaviour in the kitchen can be tracked and analysed in great depth [10].

It is also important to beware of the problems that might be encountered while

working with sensors. For example, it is shown in practice that sensors' sampling frequency is not always as indicated by configuration i.e. a sensor whose sampling rate is configured at 100Hz might produce 98Hz output data due to a number of problems usually encountered in real-time environment i.e. hardware failure, clock drifting. On another hand, raw data could also be affected by noise coming from physical activities, device displacement etc.

At the end of this phase, the raw data obtained from sensors would be expected.

Step 2. Pre-process data

In this phase, data is fed forwards filters from which potentially relevant data i.e. data that probably contains useful information about activity classes targeted is to be isolated. There are a number of scenarios where this pre-processing phase comes in very handy. For example, as signal is acquired through sensors, it could be vulnerable against noise, which adds an extra level of undesirable redundancy to the output data. It is therefore often desirable to have these noises filtered off the original data, leaving more relevant samples in the data.

At the end of this phase, the signal data's noise level would be expected to be reduced.

Step 3. Data segmentation

In long-term behaviour monitoring, it is sometimes desirable that users simulate daily life routines as closely as possible. Therefore, it is often the case that users don't and are not able to move during the whole course of the monitoring. This is especially true with the case of elderly people, patients whose amount of time spent for dynamic activities e.g. walking, running is significantly less than that of static postures e.g. sitting, lying. Put in other words, for a significant part of the time domain, the raw data collected is irrelevant to targeted activity classes whereas potentially relevant data arises only at some very specific points. Furthermore, the boundaries between actions performed continuously could also get very uncertain given the non-standard characteristic of human activities [1].

Given the problems stated above, it is therefore necessary for data to be further refined in order to produce bits of data which are informational in terms of recognising targeted activity classes.

Having acquired pre-processed data which offers a better level of credibility than original raw data, the signal is now tokenised into different segments in an attempt to confine the characteristics of the activity class into these segments.

There have been a number of proposals as to how data can be segmented. Amongst the most commonly known works are:

Windowing.

Data is segmented into windows which are normally of some fixed size. The choice of the window size as well as distance between adjacent windows can directly influence the performance of the recognition system. Many activity recognition system proposals e.g. have used this approach [1][6].

Energy based.

This approach is based on the fact that different activity classes carries different levels of energy intensity which can be translated directly to the energy use of the signal.

At the end of this phase, the signal data would be expected to consist of optionally continuous segments that are likely to be relevant to the targeted activity classes.

Step 4. Feature Selection and Extraction

At this stage qualified segments will get reduced to a discriminative unit [1] which often consists of a collection of features. They are often extracted directly from data contained in each of the segment. A good selection of features, collectively called feature space, could potentially improve the classification accuracy, separating targeted activity classes very well from each other on the feature space. However, this selection sometimes might not be performed efficiently by automation and thus requires expertise and experience from system designers. Nevertheless, a number of works on automating the feature selecting process in a non-biased manner have been proposed. For example, Huynh et al. (2005)'s "Analysing Features for Activity Recognition" work proposed a method for analysing the competence of features using cluster precision.

The selection of features also has a significant impact on the performance of the recognition system. The more features there are in the feature space, the more computation the recognition system has to perform which also influence the latency of the system. Similarly, rich but complex features could significantly improve the classification accuracy i.e. how well accurate the system is able to differentiate between activity classes given some test data set but could also bring down the latency. On the contrary, lower feature spaces and simpler features are likely to improve the system latency while placing a compromise on the accuracy of the classifier.

At the end of this phase, the system would have a data mapping which maps the pre-processed data to its corresponding feature extracts (which can consist of tens to hundreds of feature).

Step 5. Training (Learning)

Having the extraction of features on signals at hand, it is required that the system be trained on the produced feature extracts so that it is able to establish a unified inference capability to any set of test data based on the data learned, provided that at least a subset of the features on feature space are present on the test data's attribute set.

There are several methods as to how data is trained. The choice as to which method to be used, as mentioned above in step 4 is related to a trade-off between the latency and classification accuracy of the system.

The table below gives an overview about a number of commonly used training methods in practice.

Name	Description
Decision Tree Learning	Categorise discrete-valued targeted classes based on a decision tree which is built from analysing the effect of learned features on input data's distribution. The root of the decision tree represents the categorisation on the learned feature which has the most potential on classifying the input data based on e.g. information gain, gain ratio [11].
Bayesian Learning	A probabilistic inference method that classifies candidate hypotheses based on the probability distribution on the observed data.
Artificial Neural Networks	Inspired by the biological neuron network. Artificial Neural Networks can be described as a system of interconnected neurons whose connections are weighted. In order to for an Artificial Neural Network to operate, it needs to know: 1. How neurons are connected. 2. How to update the weight of connections on learning. 3. How to convert a neuron's weighted input into its output activation.

Table 1. Common learning methods.

An independent train model is then constructed for each activity class.

At the end of this phase, the system would have a complete logical reasoning foundation based on the features learned. It will now be able to classify any given input data provided that not all features learned are missing.

Step 6. Classification

Classification is one of the final steps in the Human Activity Recognition chain in which the system attempts to classify samples from test datasets. It is also the main criterion that gives a very definitive look on the success of the decisions that are made in the previous steps e.g. window size, window length, feature space.

The classification is usually carried out by interpreting the reasoning foundation constructed in the training stage.

When an input instance is ready to be classified, the system will attempt to feed the input instance into each of the train models built in the training stage, expecting a probabilistic output activity class from them. The output set is then analysed in order to find out which of those output classes should be trusted i.e. having the highest confidence score. One of the most common ways to decide the final output class is to use the probability of each output itself as the measure of confidence score.

2.3. Programming languages

It is also important to learn about the tools available for the simulation stage in order to evaluate and decide which is the most appropriate language for the implementation. Following is a list of most suitable candidates based on personal opinion.

Note: *The good and The bad sections describe the language's advantages and disadvantages in the context of the project's simulation implementation.*

Modelling language: Vienna Development Method (VDM)

The Vienna Development Method (VDM) is a commonly known model-oriented formal methods for the development of cyber systems and software. It offers a rich set mathematically well-founded tools for expressing and analysing system models during early design stages, which helps identify any logical problems with the design before the expensive implementation is started [12].

The standard language of the VDM, created in 1970s at IBM labs is the VDM-SL (Vienna Development Method – Specification Language) which has been improved through a number of extensions in order to match the trend of computer-based system market.

VDM-RT is an extension for the VDM-SL which offers many real-time specific features such as internal clock and is widely-used for co-modelling and co-simulation of embedded system.

The good:

- Offers object-oriented programming capability.
- Capture real-time systems very well with a complete set of tools available to model and simulate an embedded system.

The bad:

- Limited built-in libraries which makes the simulation implantation that involves a lot of data processing and memory management very challenging.
- File I/O is limited which is a significant disadvantage.

The Java programming language

Java is a general purpose high level programming language, invented by James Gosling at Sun Microsystem © (acquired by Oracle©) in 1990s. The language offers a rich set of libraries and is amongst the first pure object oriented programming languages.

The good:

- Object-oriented capability makes it easier to implement the simulation.
- Rich SDK with many standardised helpful libraries e.g. the collection framework.
- Real time software enabled.

The bad:

- Memory management not possible which potentially hinders the optimisation of the algorithm.
- Limited low-level facilities

The C programming language

C, developed by Dennis Ritchie in 1970s to re-implement the Unix operating system, is a general-purpose programming language that is widely used for system programming i.e. developing operating systems, embedded systems thanks to its code portability, on demand system resource access.

The good:

- Rich set of low-level facilities.
- Direct access to system resource addresses.
- Low overhead on abstraction making algorithms' implementation potentially faster and more efficient.
- The practical language for embedded application development.

The bad:

- Object orientation is not supported. However, the `structure` structures could provide a solution to equivalent capability.

2.4. Sensing devices

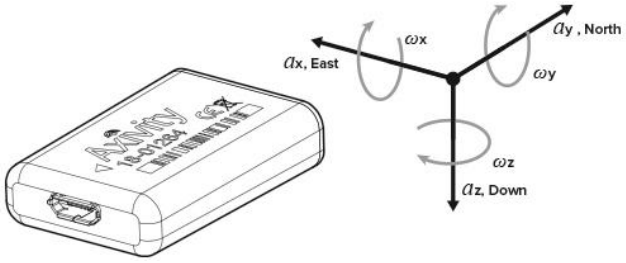


Figure 1. The WAX9 sensor. (Source: Axivity.com)

The sensing device that is to be used in this project is WAX9. The WAX9 is a wireless-enabled inertial measurement unit (IMU). The sensor consists of a MEMS tri-axis accelerometer, gyro and magnetometer that is able to stream data wirelessly thanks to the Bluetooth Low Energy radio. The device is powered by OpenMovement © platform that offers an open source API for developing software applications with the device family [13].

The WAX9 sensor is an ideal candidate for wirelessly gathering real time movement data. Additionally, the WAX9 also features a barometric pressure sensor and temperature sensor.

The sensor is powered by the Microchip's PIC24FJ256GB110 Family, offering a

16-bit architecture with linear addressing of up to 12 MB (program space) and 64 KB (data) [14]. The performance of the process can get up to 16MIPS with the instruction set being able to support different addressing modes and optimised for high level languages such as C [14].

The sensor offers a wide range of sensitivity: +/- 2 g, +/- 4 g, +/- 8 g that can be used for a wide range of applications. This sensitivity level can be fully configured through the build.

Sensitivity	**Example use case**
±2g	Fine movements such as hand writing or painting.
±4g	Mild activities such as walking.
±8g	Moderate activities such as sprinting & jumping.

Table 2. Suggested use for different sensitivity levels. (Source: Axivity.com)

The WAX9 sensor is also able to output data at different sampling rates with the default sampling rate being 50Hz. [13]. At 50Hz, the sensor can operate for up to 6 hours. However, when the device is put on hibernate mode, the battery can stay for up to 56 days. [13]

The output data from the sensor can be of two different formats: Binary format and text format.

The binary format which is machine readable is the recommended operating mode for high performance systems. The sensor data is transmitted in raw binary using slip encoding (RFC 1055) allowing the framing of the data to be extracted. The binary stream mode offers greatly reduced channel usage and hence higher sample rates.

On the other hand, under text formatting mode, the sensor produces data file of csv extension with characters being UTF-8 encoded. The sample number that is the first variable in the list (which represents a sample) can be used to identify if samples are missing due to e.g. the sensor experiencing out-of-range sensitivity. Additional information such as battery, temperature, pressure can be sampled at a lower rate i.e. twice per second in order to save battery. This is the default mode of sampling data.

The formal default format of the data is:

```
N, Ax, Ay, Az, Gx, Gy, Gz, Mx, My, -Mz, Batmv, Temp, Pres, Ia
```

Since the floating point operation is not allowed in the sensor's environment, all raw sensor's data is expressed in terms of signed integers (16 or 32 bits). Conversions are required in order to obtain correct values for designated units.

However, when the device is used in conjunction with the software provided, it is possible to have all data ready for use as the conversion is carried out on the fly.

Where:

Symbol	**Description**	**Unit**
N	The sample number	Integer
Ax	Accelerometer's reading of x dimension	g (the acceleration due to gravity which is approximately 9.81 ms^-2)
Ay	Accelerometer's reading of y dimension	g
Az	Accelerometer's reading of z dimension	g
Gx	Gyroscope's reading of x dimension	Proportional to angular velocity. The unit is degree per second (dps)
Gy	Gyroscope's reading of y dimension	Proportional to angular velocity
Gz	Gyroscope's reading of z dimension	Proportional to angular velocity
Mx	Magnetometer's reading of x dimen-sion	Pa (Pascal – SI derived unit). Conversion to altitude can be made reasonably correctly with assumption about typical conditions
My	Magnetometer's reading of y dimension	Pa
Mz	Magnetometer's reading of z dimension	Pa
Batmv	Battery level	Volt.
Temp	Temperature sensor's reading	Degree Celsius

`Inactive count`	Used to hibernate the sensor in due course.	Integer

Table 3. Variables in the output data files of WAX9.

Tools

The WAX 9 sensor comes with a Windows GUI utility client developed at Newcastle University by Daniel Jackson [15] that is able to start and stop the streaming of data wirelessly from the sensor. In order to do this, the WAX9 sensor must first be discoverable and then connected to the computer from which the utility software is running. It is also able to record the streamed data in csv format. The software also provides a basic tool to visualise the currently streaming of data.

The format of data is very similar to that of text format output discussed above.

```
$WAX9, receivedTime, sampleNumber, sampleTime, accelX, accelY, ac-
celZ, gyroX, gyroY, gyroZ, magX, magX, magZ
```

Where `$WAX9` denotes the data line received at `receivedTime` in hh:dd:mm.000 format and `sampleTime` denotes the time of the device's internal clock in seconds (0 – 65535, wrap-around applied). The other variables are already converted values which have been discussed above.

3. Design

This section will describe the detailed design of the algorithm. Where possible, it also discusses about the reasons made behind the design decisions.

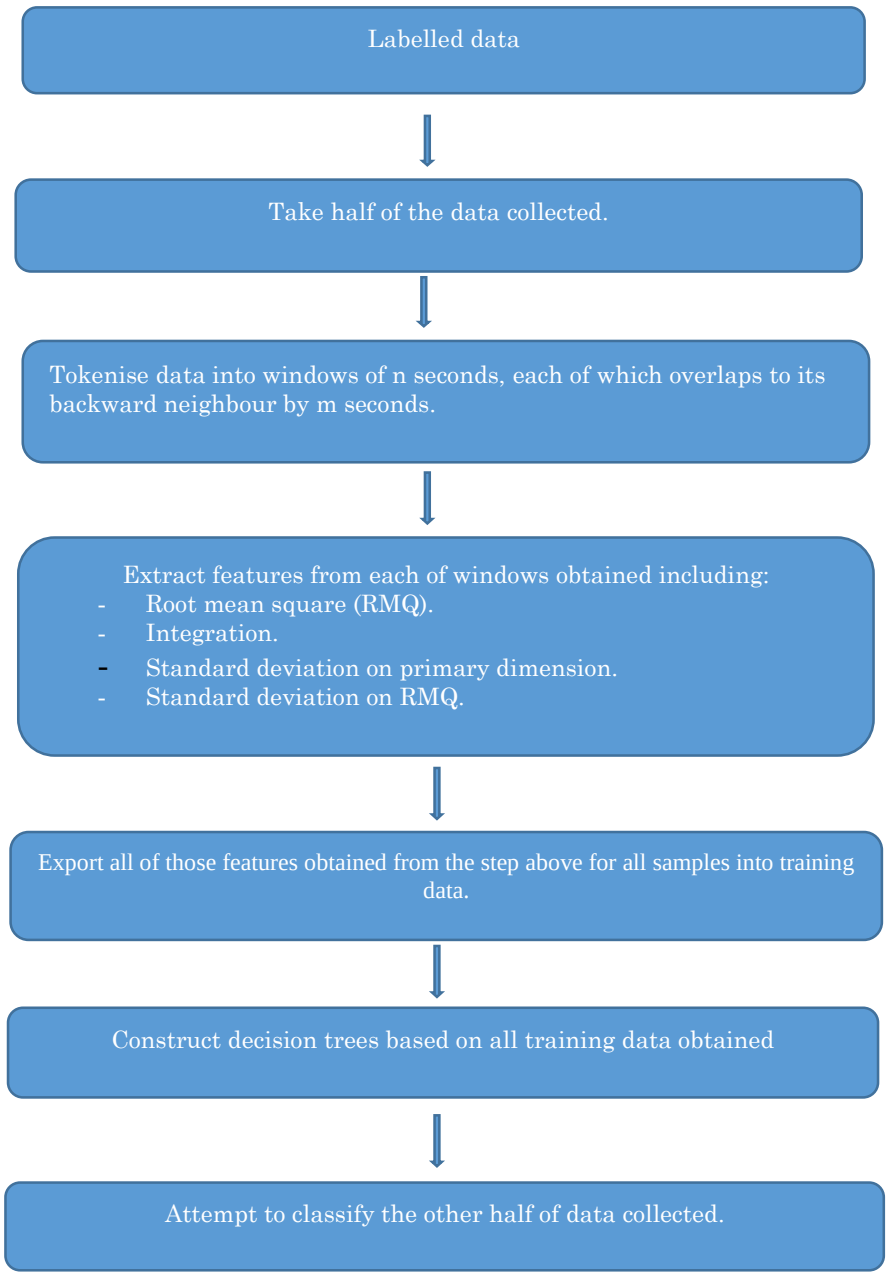


Figure 2. Design overview.

3.1. Collecting user data through the WAX9 sensor

3.1.1. Sensor use

Using multiple sensors often offers a very good performance in terms of classification accuracy. However, multi-sensor use is challenging as it requires a careful approach to synchronising across sensors which themselves could nevertheless needs synchronisation between internal sensors. The synchronisation problem in essence is due to the drifting of internal clocks used.

Furthermore, the size and complexity of the data set increases as more multiple sensors are involved. There is also a problem as to how to deal with a growing set of data effectively to maximise the potential of data collected.

Given all the challenges above and the scope of the project, it is therefore more desirable to have only one sensor for the task of collecting data. Not only does this simplify the data set and rule out the problem of inter-device synchronisation, it also helps in keeping the algorithm complexity at a reasonable level making it practical for implementation.

Nevertheless, it is also shown in a lot of works in which a single sensor was used that the classification performance is very promising e.g. Godfrey's Vespa algorithm (2014) on recognising postural transitions yields an excellent classification result of over 86% [16], Rodriguez-Martin et al. AL algorithm (2014) on the same problem also produces a similar result [6].

The decision therefore was made towards using a single sensor.

3.1.2 Sensor location

The location of the sensor plays an important role on the output signal of the device. The recorded data could complete change given the same activity script for different locations e.g. the sensor readings for a wrist worn sensor and those for a back-worn sensor might be completely unrelated for the walking activity. This happens because sensors mounted at different locations on the body are likely to record movements of different parts of the body. The difference in output data could therefore change the algorithm design at many stages i.e. feature selection. Furthermore, depending on where the sensor is placed, the noise level might as well go up or down which also affects the credibility of the signal.

Recent researches on body-worn sensors tend to mount the sensors at a number of particular locations that, while it may partially depend on the specific goal of the problem, are believed to help sensors generate better signals. Amongst the most popular locations are: wrist, ankle, arm and waist locations. The lateral-waist location was chosen from this list because it enables accurate movement monitoring since the waist is close to the centre of mass of the body and therefore could be considered one of the most representative parts of human movement [6]. The lateral-waist location is also believed to be affected less severely than other waist locations such as back location i.e. the orientation of the sensor when mounted in the lateral-waist is unlikely to get affected by the non-linear structure of the vertebral column.

The decision for the sensor location therefore was made towards the lateral-waist sensor placement.

3.1.3. Sensor configuration

The utility software is configured to instruct the sensor to produce samples at 10Hz sampling rate. The first reason for this is that it offers a good level of information in the context of this project which is sufficient for detecting targeted postures while offering a lower-cost computation and memory footprint

in the algorithm compared to the default sampling rate of 50Hz [13]. It is shown in practice that the maximum frequency of the walking activity is approximately 10Hz [17]. Therefore, a sampling rate of 10Hz is considered ideal in the context of this project.

Give the target activity classes of the project, the sensitivity of the sensor is configured for +/- 8g so that more intensive dynamic activities could be captured normally [8].

The decision made for the sensor's configuration properties is 10Hz sampling rate, +/- 8g sensitivity.

3.1.4. Sensor mounting orientation

The mounting orientation of the sensor is described as follows.

Figure 3. Sensor mounting orientation. (Source: Axivity.com)

The information about the specific axes is as follows.

Axis	**Setting**
x	Alongside the lateral-waist
y	Alongside the body
z	Alongside the anterior-waist

Table 4. Axes settings.

3.1.5. Activity Script

It is discussed in the Human Activity Recognition chain overview that when activities performed continuously, it is possible the boundaries between actions

performed continuously could also get very "blurry" given the non-standard characteristic of human activities. This is not a desirable state for input signal.

In order to separate signal of different activities and automate the process of labelling train data that is to be performed at later stages, a predefined script that declares the order in which activities are to be performed by bearers. The script also tells the users about how long each of the activity should be performed. The script is illustrated in the following diagram.

Action	CAL	STAND	CAL	WALK	CAL	LIE	CAL	Total
Duration (s)	5	14	5	14	5	14	5	62

Figure 4. The activity script.

Where:

- **CAL**: Calibration time, the user if is moving then he should come to the "rest" status i.e. standing as soon as he enters the calibration time. Else if the user is not moving i.e. standing, sitting, lying then he should stay still at that posture on entering the calibration phase. The calibration phase is aimed to re-stabilise the sensor, getting it ready for the next activity to be performed.
- **STAND:** The user is required to stand comfortably for designated duration. This would bring a realistic level of noise into the output signal which could be helpful in overfitting prevention.
- **WALK:** The user is required to perform a casual walk at different paces.
- **LIE**: The user is required to lie down comfortably.

The script is expected to produce two final signals for each user. The first signal is then used in the upcoming stages of the learning phase in order to train the system whereas the other signal is used for testing and validation.

3.2. Signal pre-processing

At the beginning of this stage, the signal is in a raw form, containing a lot of noise and irrelevant data. This sort of data is not useful for the learning process and therefore is to be removed from the signal.

It is important that the raw signal collected from the first stage follows a general pattern so that the signal can be pre-analysed in an automated manner. The aim of this process is to identify fragments of signal that contain relevant data about performed activity classes. The solution for this problem, which is mentioned above is to ask the user to perform the activity classes at specific points in the

signal time domain which can be deduced automatically by the system.

The simple formula, implemented in the system for figuring out the interval's ends of some activity class is as follows:

Action's interval commences:

```
(ACTION ORDER) - 1 * ACTION_DURATION + CALIBRATION_TIME + NOISE
```

Action's interval ends:

```
(ACTION_ORDER) * ACTION_DURATION - NOISE
```

Where:

- `ACTION ORDER`: The order of the activity class as per execution order.
- `ACTION DURATION`: The time allocated for user to perform a particular activity (in this project it applies for all activities) including noise and allocated calibration time.
- `NOISE`: The time allocated for noise filtering at the beginning and the end of the activity performance.
- `CALIBRATION TIME`: As described in the activity script description.

Using the formulas above with the parameters declared, if the user performs all the activity classes as specified by the script in terms of both order and duration then the following intervals are expected to contain information about activity classes:

Activity class	**Interval (second)**
Standing	7 – 17
Walking	26 – 36
Sitting	45 – 55

Table 5. Expected activity classes intervals.

After being filtered, the signal now contains only relevant data to activity classes.

3.3. Signal segmentation

In this step, the filtered data is now segmented into continuously overlapped windows. There are two parameters that need to be chosen and tuned in this process. They are window size and window offset.

3.3.1. Window size

The window size defines how large a window should be. There exists a trade-off in choosing the size for the windows. The window size should be large enough in order to fully preserve activity's statistical characteristics. However, as the window size gets larger, it is possible that the latency of the system can decrease

as the system always needs to gather a full window before it can perform any analysis. Smaller window sizes, however, could potentially increase the risk of intra-class variability, potentially reducing the classification performance of the system.

The window size also depends on the context of the problem. Activity frequency is also taken into consideration when analysing the window size. It is shown in experiments that knee join ankle frequency can get up to 3Hz while a front-to-back posture sway could get to 1Hz [17].

Amongst the most commonly used window size are: one second, two seconds for postures [18], three seconds for postural transitions [6]. Given the context of the problem (human daily activities) with the aid of experimenting and researches proposed above, the window size choice tends to suggest the use of the two-second windows.

3.3.2. Window offset

Window offset concerns the offset between two adjacent windows in the signal. The offset from window `w2` to window `w1` starts from `w1` i.e. two windows can overlap each other if the offset is less than that of the window size. It effectivity specifies the density of windows distributed. A trade-off also exists regarding the window offset. In order to achieve a balance between memory footprint and activity characteristic preservation, the offset chosen to be implemented is ½ of the window size.

3.4. Feature extraction

One of the most important decisions needs to be made during the design stage is the selection of features. Through the years, researches and works have proposed a wide of range of features. While high level features such as Fourier Transform related coefficients (which works based on the observation that every continuous signal could be constructed by combining several waves of different frequencies and amplitudes, Empirical Cumulative Distribution (which works by obtaining the integral the quantile function from the histogram of the signal) etc. have been proven to yield a very good result in classifying performance [8][19]. However, some of them require a significant amount of computation i.e. the Fourier Transform coefficients computation mainly relies on operations on floating point numbers (either the natural logarithm or trigonometric functions) which might include the use of power operations very often.

On the other hand, hand-picked statistical features have long been used in pervasive and wearable computing thanks to their low-cost computation requirements as well as their good performance across activity recognition problems [1][5].

Let us now describe briefly the mathematical functions that are to be used to form the feature space of the algorithm.

3.4.1 Mean

In the context of this project, mean or sometimes called arithmetic mean, mathematical expectation, average refer to a "central" value of a discrete set of numerical values. Essentially, it can be calculated by dividing the sum of values in the dataset by the number of values in that dataset.

$$mean = \frac{x1 + x2 + \cdots + xn}{n}$$

Where:

- $x1, x2, \ldots, xn$: The dataset consisting n values from $x1$ to xn.
- $mean$: The mean of the dataset.

3.4.2. Standard deviation

The standard deviation of a dataset can be said to be the expectation of the deviation of a random variable in the dataset from its mean value. In other words, it describes how far apart values in the dataset are spread out from the mean value. The formal formula for calculating the standard deviation given a data set $x_1 \ldots x_n$ with the calculated mean $mean$ is:

$$Standard\ deviation = \sqrt{\frac{1}{N}\sum_{1}^{N}(x_i - mean)^2}$$

3.4.3. Integration

The integration (integral) is often referred as the reverse process of calculating differentiation (which could be understood as a way to measure the rate of change of the function at a specific point) and has a wide range of applications e.g. calculating volume, velocity, area.

There are several ways to calculate the definite integral of the function, some of which are able to offer flexible granularity. The detailed implementation of the function is described in the Implementation section.

The design decision made is to use a wide range of statistical features as the foundation of the feature space. The features selected are described briefly as follows. The detailed implementation is discussed in the Implementation section.

Note:

- *It is however necessary to recall that all of these features are extracted from windows resulting from the segmentation and filtering process and they don't apply to single samples.*
- *The features are considered in a time-domain two dimensional function whose range is the corresponding set of values e.g. Ax, Ay, Az, Root mean square.*

Feature	**Relevance**
Mean over root mean square (RMQ) domain – See note at the end of this section.	Offers an overview about the level of magnitude of acceleration over all axes. This feature could give a hint as to whether the user's corresponding posture is static or dynamic.
Standard deviation over the RMQ domain	Offers a view about the level of fluctuation of acceleration over all axes thus offering a look about the dynamicity of the user's posture.
Mean over the y domain	Offers an overview about the level of magnitude of acceleration over the y axis. This feature could give a hint as to whether the user's corresponding posture is static or dynamic upright.
Standard deviation over the y domain	Offers a view about the level of fluctuation of acceleration over the y axis thus offering a look about the dynamicity of the user's posture upright.
Integration over the root mean square domain.	Offers the "volume" of overall acceleration over all axes gained during the course of the window. More importantly, this could further be understood as the change in the velocity (scalar) of the user during the window. See figure 5.
Integration over the y domain.	Offers the "volume" of overall acceleration over the y axis gained during the course of the window. The further implication is similar to that of the integration over RMQ.

Table 6. Discussion on various statistical features.

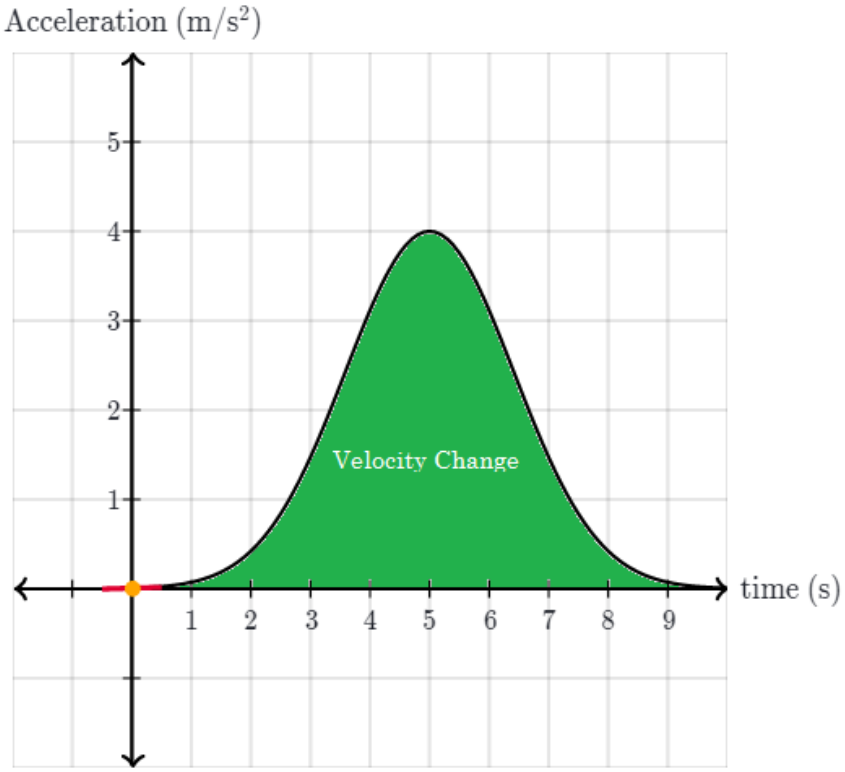


Figure 5. An example of time domain graph. (Original source: khanacademy)

Where:

- Root mean square domain: The domain of root mean square values calculated from samples of the window according to the following formula:

$$RMQ = (Ax^2 + Ay^2 + Az^2)^{\frac{1}{2}}$$

- Y domain: The domain of Ay values i.e. the (converted) accelerometer's reading in y axis.

Note: *It is important to note that this is different from the formal definition of the root mean square in that the sum of the squares are not divided by the number of variables as the current definition makes use of the relation with the gravity force g better in recognising static postures. [6]. Therefore, the abbreviation used (RMQ) is slightly different from the commonly used abbreviation (RMS).*

3.5. Learning method

Several supervised learning methods have been in use in recent works. Some of them, if used against the appropriate problem, can produce excellent classification results. As mentioned above in the Relevant work section, some commonly used learning methods are: Bayes algorithms, decision tree algorithms, artificial neural networks. It is important to be aware of the advantages and disadvantages of each algorithm and the context of the problem.

In the context of the project, it is desirable to have a learning method that offers low cost computation and memory footprint, is straight forward to implement on

plain C programming language and provides support for real-valued functions. The decision tree learning method is therefore could be considered an ideal learning platform for the feature set. A number of decision tree algorithms developed have also been proven to offer a completely expressive hypothesis space, avoiding the difficulty of restricted hypothesis space found in a number of algorithms such as Candidate Elimination algorithm [11].

Let us briefly describe the idea behind decision tree learning method. More information can be found in Mitchell (1997).

The decision tree learning method attempts to provide a means for classifying the discrete-valued target category of input vectors after it has completed the learning phase with train data.

Towards the end of the learning phase, the decision tree learning method attempts to construct a tree built from the highly analysed grouping of actual train data. The tree contains a root node, potentially several non-leaf nodes and leaf nodes. Each non-leaf node explores a new part of the hypothesis space. The leaf nodes are the final decisions to be produced based on the input vector.

The structure of the decision tree i.e. how the tree is built depends on the potential information gain correlation between features i.e. how well a feature can classify the current data set. Therefore, the root node will refer to the test on the feature that offers that best information gain with subsequent non-leaf nodes having lower information gains.

Consider an input vector i.e. a test sample. In order to obtain a classification result i.e. getting to the leaf node, the input vector must go through the branches of the tree i.e. the hypothesis space offered by the tree. At each branch a particular feature of the input vector will be tested which would decide which path the input vector needs to take. The process recurs until the input vector reaches some leaf node. It is also important to beware that as the input vector progresses further towards the leaf nodes, the hypothesis space get smaller as a result of growing search condition.

The order of features to be tested i.e. the path the input vector takes depends on the value of the input vector itself.

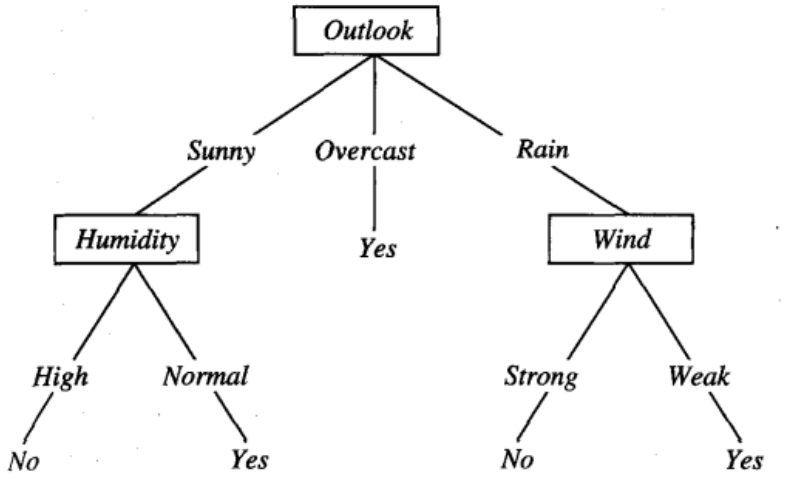


Figure 6. An example of decision tree training with discrete-value attributes.

(Source: Mitchell)

The decision tree initially was designed for discrete-valued functions i.e. functions that return a finite set of values. For example: The range of a Temperature function could be {hot, cold, cool, warm}. In order to provide support for real-valued attributes, several works have been proposed which tends towards the construction of a threshold which essentially turns a real-valued function into a discrete-valued function [11]. Multiple variances of this proposal exists e.g. multiple thresholds.

A generic basic algorithm of building decision tree from trained data based on various works e.g. ID3 algorithm by Ross Quinlan [11], LearnUnprunedTree algorithm by Andrew Moore [20] is expressed in pseudo code as follows:

```
Tree BuildTree (Feature Vector, Train Data, Labels)

Parameter:

     Feature Vector: the feature vector [F1…Fm] representing the
learning features

     Labels: The label vector [L1… Ln] representing the
classification result for n samples.

     Train data: An n-by-m matrix D that represent the train data
that consists of n samples (featuring a feature vector of m).

Return:

     Tree: The decision tree.


Begin

// Tree: the tree to be built

// bestThreshold: the best threshold for a real-valued feature

// f: the feature with the highest information gain
```

```
// InfoGainVec: The vector of information gains from every learning
feature.
Declare
     tree : Tree = empty,
     bestThreshold : numeric = 0,
     f : Feature = Null;
     InfoGainVec : Vector = empty vector


// special cases
If D1 = D2 = … = Dn then
     Return a tree with a single root node whose label is the
majority in the vector L;
Else if L1 = L2 = … = Ln then
     Return a tree with a single root node whose label is L1;


/****************************** FEATURE ANALYSIS OPERATIONS
**********************
// normal case
Else then
// Find the highest information gain as well as the feature which
yields it.
For i = 1 to m // traverse through the feature vector
     // traditional categorical case
     If Fi is discrete-valued then
               // record gain
          InfoGainVec += Gain(Fi);
// if Fi is continuous
     Else then
      // threshold set is the set of values in which label changes
(Mitchell)
          Let bestThreshold = argmax(Gain(Fi | threshold)),
                g = max(Gain(Fi | bestThreshold) in
         InfoGainVec += g;
End for
```

```
/******************************** TREE OPERATIONS
**********************

// now build the subtree
// identify the feature with highest information gain
Let f = Fk such that f = max(InfoGainVec) in
     //Tree Operation: Add a node specifying f to the tree
     tree += Node(f);

     /*************** If best feature f is discrete-valued
// if f is discrete-valued then build the subtrees from possible
values for f
If f is discrete-valued then
     // duplication is not allowed
     For each possible value of f : Djk
          // Tree operation: Add a node specifying f = Djk.
          Node(f) += Nodej : Node(f = Djk);
     // Subset of examples and features for this node
     // Add subtrees
          Let Dsubset = Train Data such that Vk = Djk,
              Lsubset = Labels for Dsubset,
              Fsubset = F - {f}
          in
              Nodej += BuildTree(Fsubset, Dsubset, Lsubset);
     End for

     /*************** Else best feature f is continuous
// else if f is continuous then build two subtrees.
//The first subtree has the data set with D[]k <= threshold, the
other one has D[][k] > threshold
Else if f is continuous then
     Node(f) += Node1 = Node(f <= bestThreshold);
     Node(f) += Node2 = Node (f > bestThreshold);
```

```
Let
         Dsubset1 = Train Data from D such that D[][k] <=
bestThreshold,
         Dsubset2 = Train Data from D such that D[][k] >
bestThreshold,
         Lsubset1 = Labels for Dsubset1,
         Lsubset2 = Labels for Dsubset2,
         Fsubset = F – {f}
     in
         // add two subtrees split by the threshold
        Node1 += BuildTree(Fsubset, Dsubset1, Lsubset1),
        Node2 += BuildTree(Fsubset, Dsubset2, Lsubset2);
End if
Return Tree;
End
```

Figure 7. Proposal on decision tree algorithm.

It is also important to beware of problems concerning decision tree learning including overfitting i.e. tree is vulnerable against noise due to the tree construction or lacks train data which leads to a poor test result despite the perfect classification of the trained data [11]. While the former problem could be remedied by re-analysing the tree, it would be very challenging as the classification performance could be reduced significantly if the latter problem were to occur.

3.6. Classification.

Two approaches for classification using decision trees were considered. The first approach was to follow the procedure proposed in the initial Human Activity Recognition chain [1] by developing an independent decision tree for each of the activity class. These independent trees are then binary: they either predict yes or no regarding the corresponding class. The results are then attached with a confident score each from which the result with the highest score is chosen as the final result. While this approach works well and promises to give a good result by minimising the number of output values, effectively reducing the risk of misclassification, it may not be the most appropriate approach in the context of this project given the following reasons:

- The set of trained data as well as test data is limited to a few number of users. Therefore, it is possible for overfitting to occur.
- Given the limited set of data and features and the likelihood of overfitting, it is likely that classifier could produce results with absolute confident scores since there are only two possible outputs yes

or no corresponding to whether the decision on the activity class is either match or not match. It is therefore could make the final decision become harder since there might be more than one result having absolute confident score.

It is therefore more appropriate in the context of this project to have a single decision tree whose output labels consist of all possible classes. This is also the decision made as to how classification is conducted.

4. Implementation

In this section, the implementation of the design was described in great details. It describes the most important matters regarding the implementation aspect while attempting to follow the structure of the design as closely as possible. It should be noted however that this section does not attempt to describe every function and libraries used. When describing algorithms used, general procedures will be used if the implementation code is complicated and contains many intermediary values that are not of interest. When describing data structures and fundamental operations, the source code which shows declarations with complete documentation are used.

Note: *From this point onwards, the terms program and system will be used interchangeably, both of which imply the executable built from the source code of the program.*

4.1. Overall source code structure

The source code structure of the program is very straight forward.

Specifically, the program only contains one single headers which contains all function declarations, type definitions and macros developed in the system. All of the system or third party header files are also included in this header file. The name of this header file is "cawax.h".

The implementation code, however is broken into different source files, each of which implements a different part of the system.

The complete source code structure and description can be found in Appendix 1.

4.2. Collecting and migrating data

4.2.1. Collecting data

The sensor data is obtained through the WAX9 software utility [15]. The output file format, as mentioned in the Devices section earlier, is Comma Separate Values – .csv format. The format of the data file follows the manual guide provided with the software.

The main function that was implemented to read data from the data file has the following signature:

```
/*
Read {count} lines from the csv file (starting from the first line)
into a LinkedList struct.
     fileName: the name of the file to be read
     count: how many lines to read
     linesRead: how many lines read
 - if {count} = {DEFAULT_PARAMETER_VALUE} then read the whole file.
*/

LinkedList * readFile(const char * fileName, int count, int *
linesRead);
```

The implementation for reading a file was implemented so that changes in the

format of the data file could be easily adapted into the program in a hassle-free way. This includes updating the index of the relevant information e.g. `Ax`, `Ay`, `Az` only.

The function will stop reading data and report the number of samples read if either the `EOF` is reached or the first required token is found missing. This is at the moment is for safety measure only. However, given that one of the abilities of the decision tree is to handle missing attributes, this safety measure is to be removed in the future development in order to improve the robustness of the system.

The csv file name should be renamed to conform to following format:

```
user_name_[purpose].csv
```

Where:

- `[purpose]`: specifies whether the data file is a training data set or a test data test which is either "`test`" or "`train`".

4.2.2. Data structures

4.2.2.1. Time format

A special time format which is used to represent the internal clock reading of the sensor obtained from the data samples is implemented that is aimed to be platform independent and robust.

The whole internal clock reading is packed in one single unsigned long value.

```
/*
TIME def (the time the sample is received) (22 bits) of a sample ex-
tracted from the output file including:
- Minute
- Second
- Milisecond
*/
typedef long CAWAX_TIME_MSM;
#define CAWAX_TIME_FROM_MSM(minute, second, millisec) \
          (((CAWAX_TIME_MSM)(minute)   << 16)) \
         |(((CAWAX_TIME_MSM)(second)   << 10)) \
         |(((CAWAX_TIME_MSM)(millisec)       ))
```

All extracting and assembling operations are performed by bitwise operations and stored partly in appropriate macro definitions in order to save time in linking phase.

4.2.2.2. Sample structure

The sample structure is one of the most important and commonly used data structures throughout the system since it represents signal samples. It contains respectively: the internal clock reading of the sensor, the order of the sample which is used to detect lost samples, tri-axial acceleration values which are all in floating point format and the RMQ value of those values.

The sample of the structure is as follows:

```
/**
 * Represent a sample obtained from the device
 */
typedef struct sample {
     INTERNAL_TIME time; // internal time of the device
     sample_th order; // order as in time domain
     acc x;
     acc y;
     acc z;
     acc rmq; // rmq calculated on the fly
} Sample;
```

4.2.2.3. Signal structure

The structure of a signal is represented in terms of a double linked list that is NULL terminated.

The double linked list was chosen as the representative structure for the signal because it performs crucial operations such as node deletion and addition very efficiently in $O(1)$ bound which is very important because the design involves pre-processing and segmenting data.

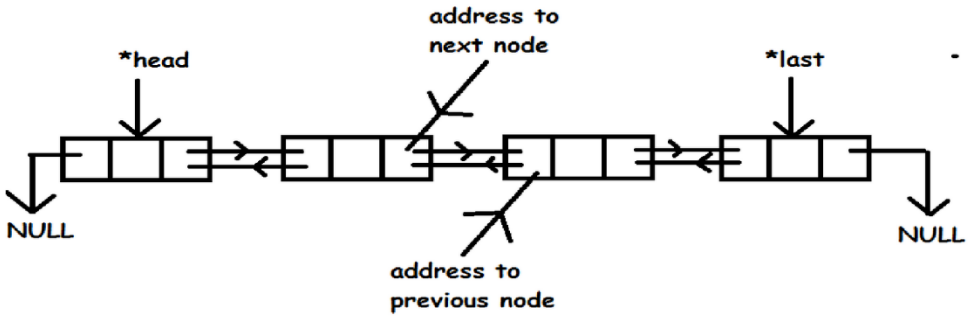


Figure 8. Linked list structure illustration. (Source: cprogramto.com)

```
typedef struct node {
    Sample sample;
    struct node *next;
    struct node *prev;
} Node;

typedef struct linkedList {
    size_t count;
    Node *head;
} LinkedList;
```

The system provided a rich set of operations usually supported and required on a vector-like data structure. This includes adding, removing nodes, jumping to specific locations on the lists, getting sub-lists. Furthermore, in order to support the processing and analysis of signal data, additional operations were also implemented to support the retrieval of any time domain real-valued series e.g. `Ax`, `Ay`, `Az`, `RMQ`.

Below is the signature and the documented exact behaviours of implemented operations.

```
typedef LinkedList Signal;

/*
Jump from one node to another node.

EXAMPLE:
 - A step of 1 will return the next node or NULL if there's no next
node (tail).
 - A step of -1 will return the prev node or NULL if there's no prev
node (head).

*/
Node * jump(const Node * depart, long step);

/*
 * Get the tail of the signal
 */
Node * tail(LinkedList* list);

/*
Make a new Node for a sample.
RETURN: Pointer to the newly created node
*/
Node * makeNode(Sample sample);

/*
Make a new list with NULL head and 0 count
Return: Pointer to the newly created list
*/
LinkedList * makeList();

/**
 * Get the node pointer at the specified index in the list
 *  - Return a null pointer if the index arg exceeds the size of the
list.
 */
Node *get(node_index index, LinkedList * list);

/**
 * Add a sample to the end a signal list
```

```
 *  - Update pre tail node
 *  - Create new node for the new sample
 *  - Link new node to the list
 */
LinkedList *add(Sample sample, LinkedList *list);

/**
 * Add a sample to the specified index of the list
 *  - Create the node for the new sample
 *  - Update the neighbour nodes as to the newly created node
 */
LinkedList *addI(node_index index, Sample sample, LinkedList *list);

/**
 * Remove (unlink) a sample from the end of the signal list based on
the time and the sample order
 *  - Update the node before the last node
 *  - Free the last node
 */
LinkedList * removeI(LinkedList *list);

/*
Get a subList from the original list
This is used for obtaining continuous windows from the signal with
0.5second overlapping.

*/
LinkedList * subList(sample_th start, sample_th end, LinkedList *
original, LinkedList * sub);

/**
 * Get the time domain series for a specific dataDimension from
{start} (inclusive) to (inclusive) {end}
 *  - If {start} = 0 then we start from the beginning
 *  - If {end} out of bound (>= list count) then {end} is tail
(count - 1)
Return a sub series array terminated by 100
 *
 */
acc *getSubSeries(dataDimension d, node_index start, node_index end,
LinkedList *list);

/*
Free the linked list: free all memory blocks allocated for the list
including:
 - All the nodes.
 - The list.
*/
```

```
void freeLinkedList(LinkedList * list);
```

While freeing signal to preserve memory, it is important that the list is freed from the tail so that the consistency the signal could be preserved in the event of failure.

4.3. Signal processing

The order of segmentation and pre-processing is swapped in the implementation compared to the design. The reason behind this is on the boundaries of signal sections. If signal is to be pre-processed first, then the resulting signal will contain segments with discontinuous sample values. This makes it a bit harder to deal with windowing later since every segmentation execution round will have to check if the window is continuous or not.

Therefore, the order of execution of the procedure was adjusted as follows:

- Tokenise raw data into windows based on the parameter specified in the design stage.
- Build features.
- Remove irrelevant data.

4.3.1. Signal segmentation

The window is represented as a structure called `FeaturedWindow` (since the features are extracted at the same time that the window is constructed) which contains variable fields that collectively make up the feature space of the algorithm. A linked list structure then was implemented to represent the continuous target (to be labelled) windows.

```
typedef struct _feature {
	LinkedList * samples;
	node_index windowStart;
	node_index windowEnd;
	// features
	acc mean_Y;
	acc mean_RMQ;
	acc stdDev_Y;
	acc stdDev_RMQ_XYZ;
	acc integral_Y;
	acc integral_RMQ_XYZ;

	int class;

	struct _feature * next;

	//todo expand (make use of secondary-dataDimension data - Y,
Z)
} FeaturedWindow;
```

Since the main header file is where all design parameters were specified, it made

it quite easy to experiment with different design parameters e.g. sampling rate, window offset.

The procedure for segmenting data into windows is as follows:

- Check if data is sufficient to learn all targeted classes (using the pre-define script).
- Traverse through the signal:
    - Update window head and window tail appropriately with the specified window size and window step size.
    - Construct the window, extract the features.
    - Add the newly created window to the output list.
    - Or terminate the window making process if the next window does not have enough window samples.

4.3.2. Feature extraction

4.3.2.1. Mean function's implementation

The mean feature was implemented in accordance to the formula stated in the design stage. The implementation for obtaining the mean value from some time-domain real-valued series is as follows:

```
/**
Calculate the mean of a sub series from the signal.
*/
acc mean(acc * input, node_index count)
{
	//printf("Calling mean at input: %p, count: %d\n", input,
count);
	long i = 0;
	acc sum = 0;
	acc current = 0;
	while ((current = input[i]) != TERMINATION_VALUE && i !=
count) {
		sum += current;
		++i;
	}
	//printf("Calling mean at input: %p, count: %d, final i: %d,
final sum: %.3f\n", input, count, i, sum);
	return sum / i;
}
```

4.3.2.2. Standard deviation function's implementation

The standard deviation feature was implemented in accordance to the formula stated in the design stage. The implementation for obtaining the standard deviation from some time-domain real-valued series is as follows:

```
/**
Calculate the SD of a sub series from the signal.
This would help figure out how acceleration values fluctuated over
time thus
potentially giving us a useful view on the properties of the move-
ment.
*/
//TODO free
acc standardDeviation(acc * input, size_t count)
{
	//printf("Calling StdDev at input: %p, count: %d\n", input,
count);
	acc m = mean(input, count);

	long i = 0;
	acc sqrSum = 0;
	acc current = 0;
	while ((current = input[i]) != TERMINATION_VALUE && i !=
count) {
		sqrSum += pow(current - m, 2);
		++i;
	}
	return sqrt(sqrSum / i);
}
```

4.3.2.3. Integral function's implementation

The integral feature was implemented based on the Trapezoidal rule on step size 1 with some minor modifications to suit the context of the project.

The trapezoid rule works by segmenting the whole series into trapezoid whose width is a free choice. Smaller choices of the trapezoid width offer a higher level of granularity in the segmentation and therefore more accurate values. This comes at the cost of more trapezoids created making the computation bigger. The width might as well be dynamic.

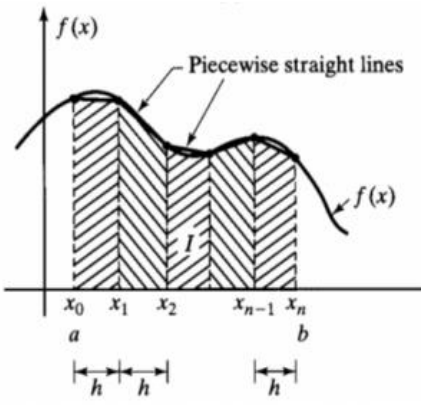


Figure 9. Trapezoidal rule on fixed width.

(Source: http://www.unistudyguides.com)

The final integral values can then be calculated by assembling i.e. adding all trapezoidal blocks together. This also includes negative areas in which the data point is negative as well since the aim of the integral in the context of the project is to work out the scalar gain in velocity.

The shape of the trapezoids can vary over the course of the signal and therefore it is required that different formulas are used.

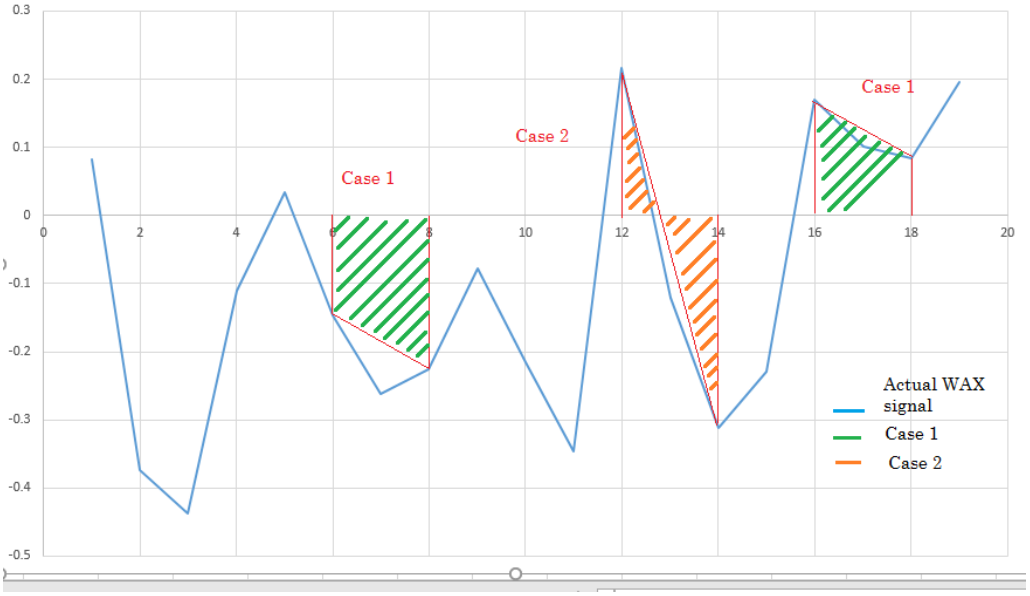


Figure 10. Different cases encountered in the signal.

Case 1. *All window sample values are either positive or negative.*

$$Trapezoid\ area = abs\ (\frac{1}{2}.(b-a).(f(a)+f(b)))$$

Where:

- abs: The absolute function (for "under the x axis" case)
- a, b: the time points in the series (in the context of the project)
- $f(a), f(b)$: the corresponding data points to a and b. (in the context of the project)

Case 2. *Given the same parameter variable names, in this case however, window sample values contain both negative and positive values.*

$$Trapezoid\ area = abs\ (\frac{1}{2(abs(f(a)+abs(f(b))}.(f(a)^2+f(b)^2).abs(b-a)$$

[https://github.com/danhng/cawax/blob/master/support/trapezoid_diff_signs.JPG]

The trapezoid functions were implemented in exact accordance to the formulas discussed.

Below is the method signature and the pseudo code of the integral calculating function for one single series. Note that the name of the method is from the Simpson rule which is a practical method for calculating integrals based on trapezoidal rules with dynamic step size.

```
METHOD SIGNATURE

Parameters:

- LinkedList * signal: The signal containing samples to be analysed.

- size_t step: The step size (default is 1) drives how accurate the
results would be.

- base: Optional positive base against which the integration is
calculated. (Default is 0).

- int inputTargets: which component(s) of the samples for which the
integration is to be calculated.

- vel_g * buf: buffer for result(s)

+ result for each input inputTarget is put in the order as per the
input targets (X -> Y -> Z -> RMQ -> M -> TIME -> ORDER)

Return:

- vel_g * : the integral buf specified in the parameter list;


PSEUDO CODE

if base is negative

     return;

//reprocess signal based on base

while (loop through all nodes)

     sample->target_value += base;

simpson = 0; // simpson ~ integral

current_sample = head of signal;

next_sample = head of signal + step;

tail;

while (current sample <> tail)

     tmp = current sample;

     while (next.time = current.time) // handle time duplicated
samples

          tmp = (current_sample + next) / 2;

          next_sample = next_sample + step;

     calculate the trapezoid (tmp, next_sample, base);

     add the trapezoid to the simpson; // accumulate trapezoid to
integral

     current_sample = next_sample;
```

```
     if (tail – current_sample) < step
          next_sample= tail; // last trapezoid
     else
          next_sample = current_sample + step;
return simpson;
*/
vel_g * simpsonSingle(LinkedList * signal, sample_th start, sam-
ple_th count, size_t step, acc base, char recoverSignal, int gOr-
Meter, int unitToMicro, int inputTarget, vel_g * buf)
```

A wrapper method for calculating integration of many time domain series at one was also implemented in order to provide better code utilisation.

4.3.3. Filtering & labelling data.

The system was implemented in an attempt to automatically label all relevant data samples as described in the design stage.

4.3.3.1. Filtering: The script implementation

All properties that are necessary to define the activity script were stored in the implementation. They include: action orders, calibration time, action time, noise time. Activity class orders are defined as per the order in which they are performed in the script. All the relevant durations are specified in seconds.

```
#define ORDER_STAND 1
#define ORDER_WALK 2
#define ORDER_LIE 3
#define ORDER_N_A 4

#define CALIBRATION_TIME_SECOND 5
#define ACTION_TIME_SECOND 14
#define NOISE_SECOND 2


#define TOTAL_TIME_FOR_ACTION 19
```

Last but not least, the implementation also provided operations for working out the relevant intervals (the labelling interval) given an activity class. This is important since it drives the process of filtering out irrelevant data which were produced during calibrations.

```
/*
Deduce the sample number that marks the beginning of the action
segment
*/
long action_start_sample(int action_order);
/*
```

```
Deduce the sample number that marks the end of the action segment
*/
long action_end_sample(int action_order);
```

4.3.3.2. Labelling data

Windows, if found in the "labelling interval", will be labelled with appropriate activity class which share the ORDER_ prefixed definitions with class order and be set to become training data. Unqualified windows that get labelled "ORDER_N_A" will be discarded.

A window is considered qualified for class labelling if and only if both ends `windowStart` and `windowEnd` variable fields are within the corresponding class's labelling interval.

A utility function that converts numerical class labels into character strings for improved readability was also implemented as a macro definition.

4.4. Learning and Classification

The decision tree implementation used in the system is C4.5 which was developed by Ross Quinlan [21] that brings numerous improvements over the old ID3 algorithm [11] including:

- Better memory allocation management.
- Support for real valued attributes was provided with description earlier which is one of the main reasons why the C4.5 was consulted and selected.
- Support for handling missing attributes which could create a more robust and flexible learner.
- Support post-creation tree re-analysis that allow the tree to be pruned. This sticks to the Occam's razor principle [11]:

  Occam's razor: Prefer the simplest hypothesis that fits the data.

C4.5 offers learner and classifier in two flavours [21]: decision trees and rules. The aim of the rules is to create a more human-readable format by using conjunction and disjunction notations for expressing the hypothesis space. However, in this project, the decision tree format was selected as the target learner and classifier [1].

[1] Note: Permission for using C4.5 for research purposes has been granted by Ross Quinlan. Please see the Appendix 5. Source code for the algorithm is under the copyright restriction declared by Ross Quinlan.

Turning back to the program itself, after the features have been extracted, the system now has the features ready. In order to incorporate with C4.5, the program now produces the data file required by C4.5 tree generator [2]. Namely, if A was the input subject e.g. user name then the following files would be produced:

File	Description
`A.names`	The. names file provides declarations for learning attributes i.e. the name of the attribute, whether it is categorical or continuous. The. names file also provide declaration for the target attribute which must be categorical.
`A.data`	The .data file contain training data that will be used by c4.5 program to generate decision tree. The training data should have been already labelled by the system at earlier stages.
`A_test.data`	The _test.data file contains the feature ready data files name which is used in testing and verification.

Figure 11. Description of C4.5. recipe files.

4.5. Putting it together

A typical execution of the program is described below, starting with the data files containing the signals of two script's executions (corresponding with the training set and the testing set) and ending up with the classification tree produced and the test data processed.

[2] For more information about C4.5, consult the manual of the program available at http://www.rulequest.com/Personal/.

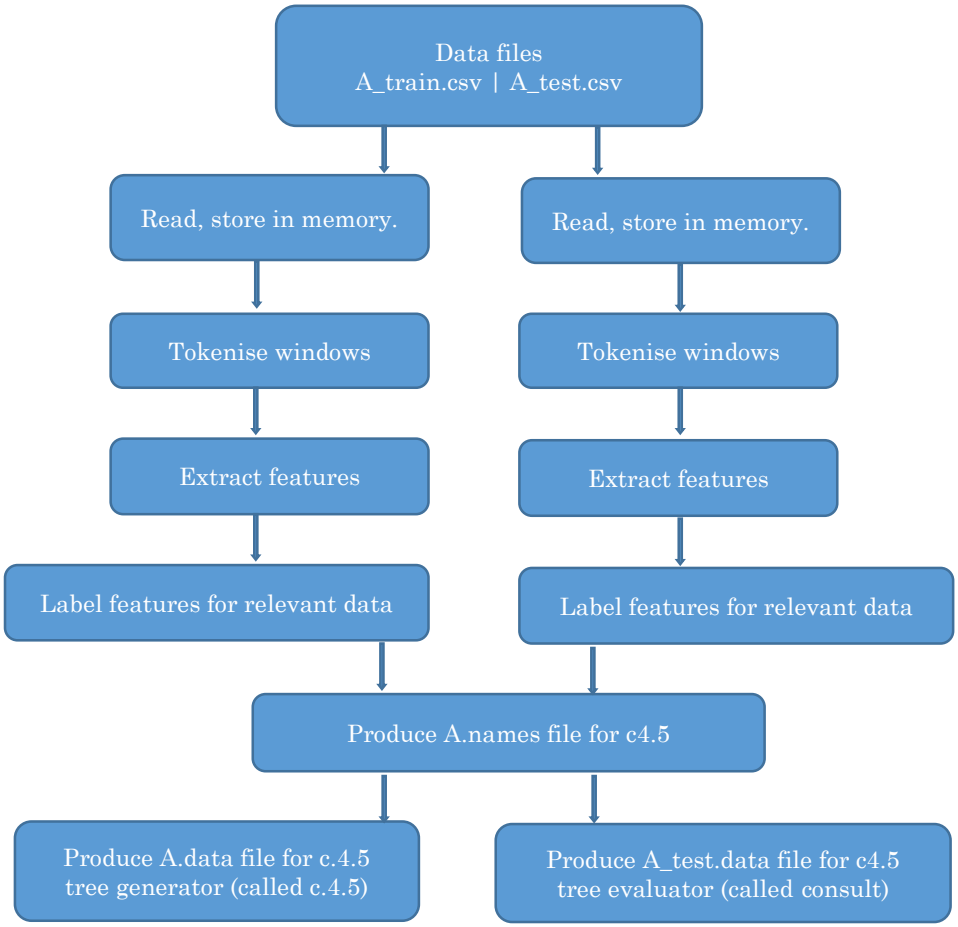


Figure 12. Program execution flow.

4.6. Implementation notes.

The program was implemented using ISO C11 standard. It was also designed to be able to be cross-platform with targeted environments including Linux and Microsoft Windows. Specifically, tested environments include Ubuntu 15.10 LTS, Microsoft Windows 10 64 bits.

The program was initially developed using Microsoft Visual Studio 15 Integrated Development Environment. After the main skeleton of the program was completed, the program was ported to Linux. Build facility was then also provided through make and Makefile.

It is important that the executable on Microsoft Windows requires Linux-based environments such as Cygwin for c4.5. to compile. Native Windows supports are yet to be produced.

5. Testing

5.1. Testing procedure

Two users were invited to participate in the experiment. They include Michael (A), male, 25 years old and Ramona (B), female, 45 years old.

The two users were asked to perform the activity script five times, resulting in two set of signals $\{A1, A2, A3, A4, A5\}$ and $\{B1, B2, B3, B4, B5\}$. These set of signals were then inspected to find out the two most suitable signals for each set of signal $A = \{Atrain, Atest\}$ and $B = \{Btrain, Btest\}$. A signal S is considered better than another signal $S1$ if it matches the script better than other one from the point of view of the inspector.

From each signal set A and B, the first signal i.e. $Atrain$ was chosen to be the training data while the other signal i.e. $Atest$ would be used as the test data.

The signal files were then renamed so that the system would be able to recognise them automatically upon execution e.g. `A_train.csv`, `A_test.csv`.

The program then can be started up and execute as normal.

5.2. Results

The classification results on two participants' test signals are shown below.

Figure 13. Classification results on test data sets.

6. Evaluation

6.1. Classification result evaluation

Compared to a number of works consulted, which yield a +85% classification performance such as [8], [16], it can be seen that the performance of the algorithm is lower. It was observed that the program got confused between standing and lying postures in about 30% of corresponding test cases. This could be explained as the decision tree's hypothesis spaces for standing and lying are not really separated. A possible cause for this problem is that the learning datasets obtained were not large enough to capture users' statistical characteristics.

On the other hand, classification results for walking activity is significantly better, offering a +85% accuracy.

Overall, the classification performance is about 70% which could be considered positive in the context of the project but it is still below initial expectation.

Let us summarise the points learned from the results and the trees:

- Statistical features such as standard deviation, RMQ can still be useful.
- The learning datasets were not large enough.
- The activity script therefore may need to be adjusted to allow larger datasets to be collected.

6.2. Feasibility analysis on being deployed in a real-time environment

This section will analyse the algorithms regarding various aspects e.g. time complexity, space complexity i.e. memory footprint. It then discusses about the feasibility of the program being deployed and integrated into the WAX9 embedded environment as one of the initial goals set up for this project.

6.2.1. Floating point operations

The program makes a heavy use of floating point operation due to the data in the csv files having been already converted to the correct unit. Using floating points is considered expensive as the number of digits to be processed and computed gets bigger. The WAX9 internal environment supports 16-bit and 32-bit unsigned integer operations only [13]. Therefore, given the current implementation the program cannot be deployed in the targeted environment. However, fortunately, it should be noted that most of the floating point operations occur as a result of the original acceleration data `Ax`, `Ay` and `Az` being floating point numbers. Therefore, it is possible to change the format of these variables into integers using their original readings. If the conversion is to be made, most of the features will unlikely be subject to any significant loss since the dominant types of operation used in extracting features are addition and multiplication as can be seen in calculating the integration and other statistical features.

However, the root mean square feature could be potentially expensive as it involves square root operations. It is, however, not a fatal problem as it is proven in practice with a device implementation called Prompter, which makes use of the similar feature. [22].

On another hand, it is also important that floating point constants used i.e. g (9.80665) might be required to get scaled up or rounded up in order to adapt the embedded environment.

6.2.2. Memory footprint

Memory allocation is also an important aspect. According to Microchip, the processor line of the device has 16Kilobytes of main memory (SRAM) which is equivalent to 8000 16-bit integers or 4000 32-bit integers. For a 10Hz sampling rate which produces 10 samples per second, each of which containing up to 10 variables in most cases (additional information such as pressure and battery level are output in 2Hz [13]) which might take up to about 100 32-bit integers for one second added to the window size. This has a significant impact on the design parameters as to the size of the windows as one second added in the size of the window will result in approximately 2.5% memory overhead. Furthermore, memory is also required for storing intermediate results when calculating features, which needs to be reduced.

6.2.3. Implementation strategy

As of the current implementation strategy, it can be seen that the order of stages in the program execution will result in a notable impact on memory as well as device throughput as features are extracted for the whole unprocessed signal. If the implementation was to be ported to the real device's environment, the order needs to be adjusted in favour of memory saving and throughput. If the device were to perform context switching i.e. turning off unnecessary sensors, it needs to do it within a short interval so that the users should not already be performing another activity class which contradicts to the goal of the current switch. In other words, it is desirable that the device keeps the latency at a good level, avoiding stale switching. In order to do so, design parameters need to be experimented to balance between classification performance and latency.

The last matter with the current implementation is the learner and classifier. The C4.5 is not suitable for being deployed due to copyright restrictions and some of its limitations found in practice [21]. It is therefore necessary for a learner to be developed if the program is to be deployed.

To sum up, it is obvious that at this step, there are a number of problems that are yet to be addressed in order to turn the program to a practically deployable platform. Further work is required, some of which are discussed in the next section: Further developments.

6.3. Objectives evaluation

There exists a number of problems that are yet to be addressed in terms of deployment and classification performance. The current implementation has also yet to make the final targeted stage, which is the deployment to the real devices.

The classification accuracy is at an acceptable level of approximately 70% which however is notably lower than recent works e.g. over 85% [6].

6.3.1. Objective 1

Produce an algorithm optimised for the lower back position that is capable of detecting basic human postures: standing, walking, lying given a three dimensional acceleration data as the primary input.

The project managed to provide a complete design with detailed reasons for important decisions.

However, at some points e.g. window size, window offset, the decisions made were not convincing and requires further proof.

6.3.2. Objective 2

Demonstrate the accuracy, efficiency and feasibility of the produced algorithm by:

1. Produce a semi-automated implementation on a simulated environment in which the targeted device's specifications are taken into consideration.

The project produced a complete implementation for the design proposed in the chosen programming languages.

However, the implementation was still biased towards the "manual" side compared to the "automation" side. The classification process is the clearest problem in which an automated approach failed to be produced which then led to the use of a manual approach. Another problem is that the implementation did not concern the "embedded" side, which is one of the initial goals discussed in the project proposal.

2. *Perform experiments and evaluation on the produced algorithm and implementation.*

The project did provide experiments on users.

However, the number of users participating is very limited which was proven to be the root of many problems including inefficient decision tree, low classification performance on some specific activity classes i.e. standing and lying.

3. Produce a discussion on the feasibility of the algorithm's implementation in the deployment to the actual sensor devices.

The project provides an overview about the feasibility of the algorithm which discusses many practical problems found in embedded environment e.g. floating point operations, memory usage.

However, the analysis could have been better if a more detailed analysis on the time and space complexity of algorithms and its impact on the actual working of the sensor was provided. This is due to the lack of experience and knowledge on the embedded programming environment.

7. Future work

It is undeniable that the design and implementation of the current program needs a lot of improvement before it can be deployed to the targeted environment. The section below proposes a number of ideas which can be considered for future developments. These suggestions were produced after the implementation and the testing phases.

Awareness of the time of the day when learning and classifying data

If the device was able to work out the approximate time of the day it could potentially produce a lot of benefits in terms of memory usage and latency. For example, for long term behavioural monitoring, the device can be instructed to be aware of the sleeping schedule of the users i.e. night time so that it can then use the information about the current time to deliberately turn off most of the sensors if it was, for example 2h30am.

It is undeniably a challenging problem to provide time-awareness to the devices since they only have internal clocks that count the time elapsed since start-up. The implementation may involve providing a periodic synchronisation paradigm with surrounding Bluetooth-enabled devices.

Piping waxrec to provide real-time simulation and data collection

`waxrec` is the underlying program that provides real time data streaming feature for the WAC9 GUI utility software. [13]

At the moment activity data must be collected in advance for the program to start. It is more desirable to have the program itself invoke the `waxrec` process which then streams the data in real time and supply it to the parent process (the program).

Replacement of C4.5 by C5.0 in the simulation

C4.5 has been superseded by C5.0, which has a lot of improvements over its predecessor including:

- Better memory management.
- Reduce decision trees' size which effectively reduces computation time and memory footprint.
- Boosting feature which is a new feature that introduces the use of networked decision trees to improve classification performance.

Auto-detection of wrong sampling rates in input files

The program at the moment is not aware of the correctness of the sampling rate of the input data files. It implicitly assumes that the data file has the sampling rate of a pre-configured value i.e. 10Hz. If the data file had a different sampling rate and this problem was not spotted during and after the execution, the learning and classifying phases might be influenced severely as a result of incorrect window segmentations.

Therefore, it is more desirable to have the system automatically detecting

incorrect sampling rate. The implementation of this feature might include the use of the internal clock readings of subsequent samples.

Script adjustment

The pre-defined script, while being succinct might not work well for the elderly people given that it only provides 5 seconds of calibration time and 14 seconds for performing activities. Therefore, it is more desirable to have a script that could be applied to a wide range of ages. This requires significant adjustments to the current script.

Configure design parameters through configuration files

As of the current implementation, all of the design parameters e.g. window size, indices of variables in the input data files are stored in the main header file of the program. This can be improved by placing them in configuration files as it makes it easier to compile the program for different sensors which outputs data files with different indices of variables.

8. Conclusions

Activity recognition is an important field in pervasive computing. Its goal is to recognise actions performed by agents through the learning of a series of signal captured from the agents as well as environmental conditions. The activity recognition problem is important because it influences a wide range of human-centred problems such as entertainment, security, and eldercare. Researches so far has focused on the recognition of simple activities while recognising concurrent activities or behaviour prediction remains a challenging problem.

The project set out with two primary goals. The first goal is to design an algorithm for detecting three different postures including standing, walking and lying which, if deployed on sensor devices, could help them perform contextual switching in accordance to the user's action. The next goal is to provide a simulation implementation and demonstrate the classification performance of the algorithm through a number of tests on real users. Initially, the project also aimed to provide a deployment of the algorithm to the sensor devices used in the project. However, due to a number of unforeseen problems and lack of experience, this goal has been cancelled and would be a subject for further development in the future.

The project managed to provide a complete design for the algorithm and an implementation for it. A number of tests have been conducted on participating users which yields an overall classification performance of 70% which is notably lower than researches and works consulted while carrying out this project.

A number of suggestions for future developments have been proposed, suggesting the use of an improved learning method along with a number of features which could help in automating the program's execution.

9. References

1. Bulling, A., Blanke, A., & Schiele, B., (2014). “A Tutorial on Human Activity Recognition Using Body-Worn Inertial Sensors”. ACM Computing Surveys, Vol. 46, No. 3, Article 33.
2. Kim, E., Helal, S., & Cook, D. (2010). "Human Activity Recognition and Pattern Discovery". *IEEE Pervasive Computing*, vol. 9, no. 1, pp. 48-53.
3. Brush, J., Krumm, J., & Scott, J., (2010), “Activity Recognition Research: The Good, the Bad, and the Future”, Pervasive 2010 Workshop: How to do good research in activity recognition.
4. Wikipedia (2015). “Activity Recognition”. [Online]. Available at: https://en.wikipedia.org/wiki/Activity_recognition. Last accessed on 05/11/2015. Last accessed on 04-May-2016.
5. Bao, L., & Intille, S., (2004). Activity Recognition from User-Annotated Acceleration Data. Massachusetts Institute of Technology, Cambridge, MA 02142, United States of America.
6. Rodriguez-Martin, D., Samà, A., Perez-Lopez C., Cabestany, J., Català, A., & Rodriguez-Molinero, A., (2014). “Posture Transition Identification on PD patients through a SVM-based technique and a single waist-worn accelerometer”. Neurocomputing. [Online]. Available at: http://dx.doi.org/10.1016/j.neucom.2014.09.084. Last accessed on 07-Dec-2015.
7. Michalis, V., Christophoros, N., & Ioannis, K., (2015). “A Review of Human Activity Recognition Methods”. Frontiers in Robotics and AI, vol. 2, no. 00028.
8. Hammerla, N.Y., Kirkham, R., Andras, P., & Ploetz, T., (2013). “On preserving statistical characteristics of accelerometry data using their empirical cumulative distribution”. Proceedings of the 2013 International Symposium.
9. Laerhoven, V.K., Schmidt, A., & Gellersen, H.W., (2002). “Multi-Sensor Context Aware Clothing”. Proceedings of the 6th International Symposium on Wearable Computers. IEEE.
10. Khan, A., Jackson, D., Comber, R., Pham, C., Ploetz, T., & Olivier, P., (2010). “Ambient Kitchen”. Open Lab, Newcastle University, United Kingdom. [Online]. Available at: https://openlab.ncl.ac.uk/things/ambient-kitchen/. Last accessed on 04-May-2016.
11. Mitchell, T., (1997). “Machine learning”. The McGraw-Hill Companies, Inc., United States of America.
12. Overture Tool. (n/a). “The Vienna Development Method”. [Online]. Available at: http://overturetool.org/method/. Last accessed on 02-May-2016.
13. Axivity©. (2016). Axivity© Technical Guide. [Online]. Available at: http://axivity.com/userguides/wax9/technical/. Last accessed on 28-Apr-2016.
14. Microchip Technology Inc. (2009). “PIC24FJ256GB110 Family Data Sheet”. [Online]. Available at: http://ww1.microchip.com/downloads/en/DeviceDoc/39897b.pdf. Last accessed on 06-May-2016.
15. Digital Interaction. (2016). “OpenMovement”. [Online]. Available at: https://github.com/digitalinteraction/openmovement. Last accessed on 08-May-2016.
16. Godfrey, A., Barry, G., Mathers, J.C., & Rochester, L., (2014). "A comparison

of methods to detect postural transitions using a single tri-axial accelerometer,". Engineering in Medicine and Biology Society (EMBC). 36th Annual International Conference of the IEEE, pp.6234-6237.
17. McCamley, J., & Harrison, J.S., (2016). "Nonlinear Analysis for Human Movement Variability". CRC Press, Taylor & Francis Group, NW.
18. Randell, C., & Muller, Henk., (2000). "Context Awareness by Analysing Accelerometer Data". *Wearable Computers, The Fourth International Symposium on*, Atlanta, GA, USA, pp. 175-176.
19. Huynh, T., Schiele, B., (2005). "Analyzing Features for Activity Recognition". Multimodal Interactive Systems, Germany. [Online]. Available at: http://www.mobvis.org/publications/huynh2005eusai.pdf. Last accessed on 02-May-2016.
20. Moore, A., (n/a). "Decision Trees". Carnegie Mellon University, United States of America. [Online]. Available at: http://www.autonlab.org/tutorials/dtree18.pdf. Last accessed on 29-Apr-2016.
21. Quinlan, R., (n/a). [Online]. Available at: http://www.rulequest.com/Personal/. Last access of 07-May-2016.
22. Jackson, D., (2015). "Prompter". Open Lab, Newcastle University, United Kingdom.
23. Casale, P., Pujol, O., Radeva, P., (2011). "Human Activity Recognition from Accelerometer Data Using a Wearable Device". Germany. Berlin. Springer-Verlag Berlin Heidelberg.
24. Abu-Mostafa, Y., Magdon-Ismail, M., Lin, & H.T., (2012). "Learning from data". AMLBook. United States of America.
25. Casale, P., Pujol, O., & Radeva, P., (n/a). "Human Activity Recognition from Accelerometer Data Using a Wearable Device". Computer Vision Center, Bellaterra, Barcelona, Spain, Dept. of Applied Mathematics and Analysis, University of Barcelona, Spain.
26. Bidargaddi, N., Sarela, A.; Boyle, J.; Cheung, V., Karunanithi, M., Klingbei, L.; Yelland, C., & Gray, L., (2007). "Wavelet based approach for posture transition estimation using a waist worn accelerometer". in Engineering in Medicine and Biology Society. EMBS 2007. 29th Annual International Conference of the IEEE, pp.1884-1887.

Appendices

Appendix 1. Source code structures

File name	Description
cawax.h	Main header of the program.
cawax_analysis.c	Contains signal segmentation procedures.
cawax_datareader.c	Contains I/O procedures.
cawax_datastruct.c	Contains basic data structure operations of the program e.g. sample, signal.
cawax_debug.c	Contains a debugging facility for the program.
cawax_debug.h	Header of the debugging facility.
cawax_main.c	The main entry point of the program.
cawax_maths.c	Contains feature definitions.
cawax_test.c	Contains function tests.
cawax_time.c	Contains the internal time related definition.
cawax_utils.h	Contains utility functions.
run.sh	The bash script that perform a full cycle including compilation, build and execution of the program.
Makefile	Makefile for the program.
Simulated.vc*	Microsoft Visual Studio 15 related files.

Figure 14. Source code descriptions.

Appendix 2. Documented source code

Complete source code, documentation and manual guides are available through GitHub with development progress (commits etc.) made publicly available. Please head to the home repository of the project for more information.

https://github.com/danhng/cawax/simulated

Appendix 3. Test data

A limited data sets recorded are also available through GitHub. Please head to the home repository of the project for more information.

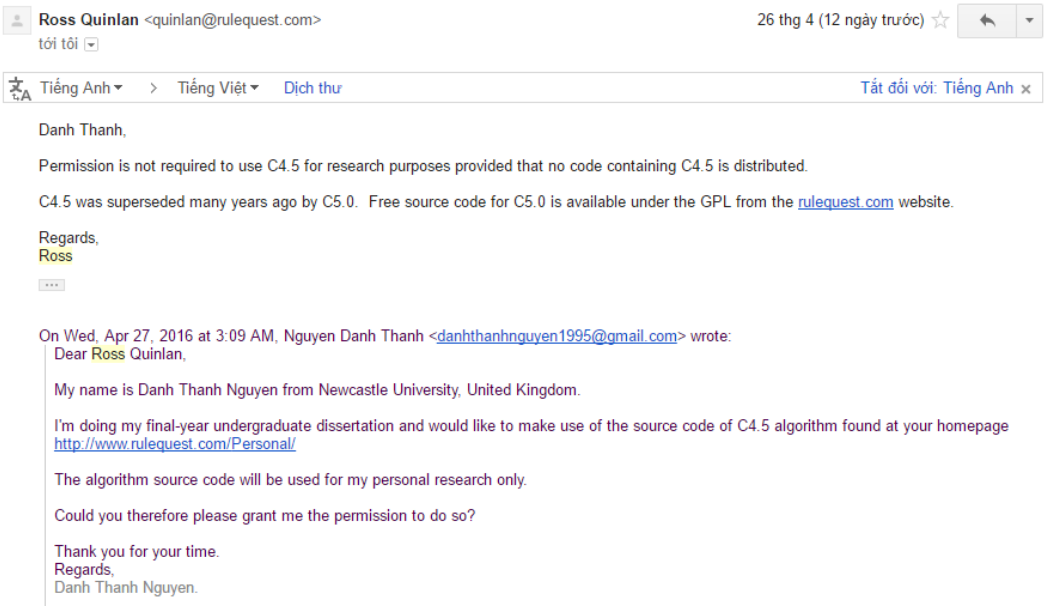
https://github.com/danhng/cawax/simulated

Appendix 4. Trapezoid case 2 proof

Hand-written proof for trapezoid case 2 is available through GitHub.

https://github.com/danhng/cawax/blob/master/support/trapezoid_diff_signs.JPG

Appendix 5. C4.5 use permission



Ross Quinlan <quinlan@rulequest.com> 26 thg 4 (12 ngày trước)
tới tôi

Tiếng Anh › Tiếng Việt Dịch thư Tắt đối với: Tiếng Anh

Danh Thanh,

Permission is not required to use C4.5 for research purposes provided that no code containing C4.5 is distributed.

C4.5 was superseded many years ago by C5.0. Free source code for C5.0 is available under the GPL from the rulequest.com website.

Regards,
Ross

On Wed, Apr 27, 2016 at 3:09 AM, Nguyen Danh Thanh <danhthanhnguyen1995@gmail.com> wrote:

Dear Ross Quinlan,

My name is Danh Thanh Nguyen from Newcastle University, United Kingdom.

I'm doing my final-year undergraduate dissertation and would like to make use of the source code of C4.5 algorithm found at your homepage http://www.rulequest.com/Personal/

The algorithm source code will be used for my personal research only.

Could you therefore please grant me the permission to do so?

Thank you for your time.
Regards,
Danh Thanh Nguyen.

Figure 15. C4.5 permission request.

Figures and tables

End