WEARABLE SENSOR BASED ACTIVITY CLASSIFICATION DURING FAST BOWLING IN CRICKET

Jayamini Susankalpana Ranaweera

(148465E)

Degree of Master of Science

Department of Electronic and Telecommunication Engineering

University of Moratuwa

Sri Lanka

December 2018

WEARABLE SENSOR BASED ACTIVITY CLASSIFICATION DURING FAST BOWLING IN CRICKET

Jayamini Susankalpana Ranaweera

(148465E)

Thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Electronics and Automation

Department of Electronic and Telecommunication Engineering

University of Moratuwa
Sri Lanka

December 2018

DECLARATION, COPYRIGHT STATEMENT AND STATEMENT OF SUPERVISOR

"I declare that this is my own work and this thesis does not incorporate without acknowledgment any material previously submitted for a Degree or Diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgment is made in the text."

"I also grant University of Moratuwa, Sri Lanka the non-executive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books)."

……. ……..

Signature Date

The above candidate has carried out research for the Master's Thesis under my supervision.

Name of Supervisor: Dr. Pujitha Silva

…….. ……

Signature of Supervisor Date

ABSTRACT

Inertial Measurement Unit (IMU) data can depict three dimensional rotational angles specific to a motion. However, either to prevent injuries or to enhance performance based on IMU data, a specific segment of the total movement cycle needs to be analysed. This requires a process to segment the total motion into key phases during the complete movement cycle. The proposed method focuses on the major research question of developing a pattern recognition model to classify the three main phases (Run Up, Delivery Stride and Follow Through) of fast bowling action in cricket.

The research focuses on seven fast bowlers delivering a minimum of four deliveries in a training environment with IMU's to capture motion. Nine-axis IMU's are selected and quaternion based three-dimensional motion data are captured and stored. The research initially focuses on finding the most appropriate sensor position on body among calf, thigh, trunk and forearm to collect data for activity classification in fast bowling. The classification performance obtained by Support Vector Machines (SVM) indicate that overall, second and fourth quaternion on Forearm is the most suitable combination of quaternion and position for data collection.

Data collected from IMU's on forearm are used to develop a machine learning model to segment the three key phases of the fast bowling action. Video feedback is also obtained when defining initial classes for classification. A moving window collects time domain statistical features, Least Absolute Shrinkage and Selection Operator (LASSO) is used for feature selection and Principle Component Analysis (PCA) for dimensionality reduction. Synthetic Minority Over-Sampling Technique (SMOTE) is implemented to overcome class imbalances. K-Nearest Neighbour (k-NN), Random Forest (RF), Naïve Bayes (NB) and Support Vector Machines (SVM) are tested as supervised classification methods for activity classification. Cross validation determines classification model performance based on accuracy, precision, recall and F-measure values. The results indicate that k-Nearest Neighbour produces best overall classification accuracy of 82% among the tested supervised classifiers. Finally, the model is verified against a test sample from one of the bowlers.

ACKNOWLEDGMENT

First I would like to thank my supervisor Dr.Pujitha Silva for mentoring me and teaching me throughout the course of the research and guiding me in moments of confusion by showing the correct path towards successful research completion. I also owe my gratitude to Dr.Upeka Premaratne and Dr.Amal Shehan Perera for all the advice, guidance and supervision given to me in machine learning aspects of my research. I am also thankful to course coordinator of the MSc in Electronics and Automation Prof. Rohan Munasinghe for all the guidance provided throughout the course of study. I would also like to thank Mr. Damith Kandage for all the timely assistant on various matters during the study.

Next, I owe my gratitude to Mr. Siva Gawsalyan, Mr. Shehan Deshapriya, Mr. Rasika Manjujeewa and Mr. Udith Shan for all the timely assistance provided in manipulating Kairos motion analysis system and obtaining video feedback during the data gathering phase. I would also like to thank Mr. Samith Danushka from Cric Sri Lanka for providing testing facilities for data collection including bowlers from the academy during the data gathering. I also owe my gratitude to all cricketers who participated in the data gathering.

I would also like to thank General Manager – Autonomation at MAS Intimates (Pvt) Ltd Dr. Chandika Wickramatillake for providing me necessary leave from work to participate in matters pertaining to the research. Finally, I would like to thank my family, friends and colleagues for all encouragement provided during the study.

Jayamini Susankalapana Ranaweera

B.Eng (Hons) (SHU-UK), B.Sc (USJP - SL), MIET

Assistant Manager – Research & Innovation

MAS Intimates (Pvt) Ltd

TABLE OF CONTENTS

CHAPTER 6

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
IMU	Inertial Measurement Unit
ISB	International Society of Biomechanics
BFC	Back Foot Contact
FFC	Front Foot Contact
DWT	Discrete Wavelet Transform
DFT	Discrete Fourier Transform
PSD	Power Spectral Density
DC	Direct Current
PCA	Principal Component Analysis
k-NN	k Nearest Neighbour
SVM	Support Vector Machine
RF	Random Forest
NB	Naïve Bayes
SLGMM	Supervised Learning Gaussian Mixture Model
A-NN	Artificial Neural Network
HMM	Hidden Markov Model
RMS	Root Mean Square
MAD	Median Absolute Deviation
IQR	Inter Quartile Range
PC1	Principal Component 1
PC2	Principal Component 2
TCP	Transmission Control Protocol
UDP	User Diagram Protocol
DSLR	Digital Single Lens Reflex
SMOTE	Synthetic Minority Over-Sampling Technique
LASSO	Least Absolute Shrinkage and Selection Operator

CHAPTER 1

1. INTRODUCTION

Following research is focused around developing a machine learning model to classify human activity, specifically on classifying key phases during fast bowling in cricket. It is a continuation of the work conducted for Design project module for the Postgraduate Diploma in Electronics and Automation.

1.1 Background and Motivation

Cricket has become one of the key sports in Sri Lanka. Modern cricket is transforming into a sport embedded with key factors of technology. Developed countries are relishing upon the usage of technology and biomechanics in cricket. They have gained a competitive edge over countries like Sri Lanka in most sports by the usage of modern engineering technologies. This paved way towards exploring the capabilities of fusing engineering principles into cricket to assist Sri Lankan cricketers compete more comprehensively with other high-profile cricket playing countries.

Most modern biomechanics analysis centres rely on the usage of high speed cameras like VICON Motion Capturing System for motion detection. These systems have been extensively used by many researchers for cricket related motion analysis. However, these systems have the following disadvantages,

- Highly expensive to purchase.
- Requires specific laboratory facilities.
- Requires expertise assistance for application and analysis.

These key disadvantages have paved way towards the importance of developing low cost wearable motion analysis systems which can be easily used by Sri Lankan cricketers to help enhance their performance levels. And the lack of technological availability and continuous demonstration of poor performances of Sri Lankan sportsmen paved way towards the motivation for me to develop a three-dimensional motion analysis system to Sri Lanka. To accomplish this quest, I needed to solve the major research question addressed through this research.

My previous Design Project work on the Postgraduate Diploma in Electronics & Automation concentrated on developing a wearable sensor based system for such applications. Inertial Measurement Units (IMU's) were used as wearable sensors to detect the motion of cricketers. However, there were few parameters which needed to be addressed, for this system to be used as a performance analysis or injury prevention tool.

One such important parameter is 'activity classification'. Even with the implementation of wearable sensors for motion detection it creates a difficulty in segmenting the different movement phases of the activity for analysis. With the usage of wearable sensors, it requires other video processing methods to segment the different phases of the activity. This exact requirement paved way as the main background to this research and to explore the possibility of applying pattern recognition and machine learning methods to classify different phases during fast bowling. The basis is built around fast bowling in cricket, to research the possibility of applying pattern recognition techniques to IMU data with the objective of segmenting and understanding the three different phases during fast bowling in cricket.

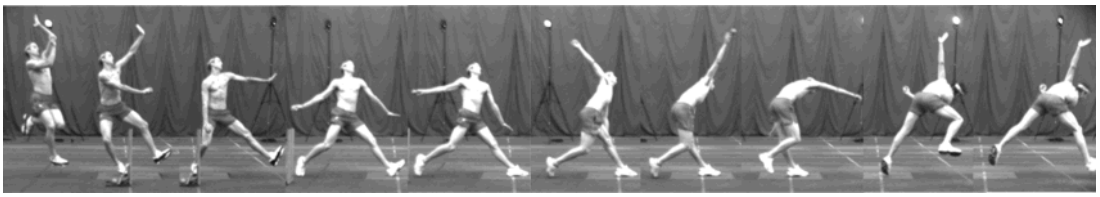


Figure 1: Phases in fast bowling action [2]

- Run up
- Delivery stride (Back Foot and Front Foot Contact)
- Follow through

Previous research [1] published on identifying key factors contributing to increasing bowling speeds in cricket also contributed as a motivating factor to develop a system capable of identifying these key phases during fast bowling to help Sri Lankan fast bowlers increase their bowling speeds.

1.2 Problem Definition

Nine (9) axis Inertial Measurement Unit based three-dimensional motion capturing system provided continuous rotational angles during fast bowling. However, this data alone did not provide enough information to assist in a biomechanical analysis of the fast bowler. The continuous data once plotted would appear as depicted below,

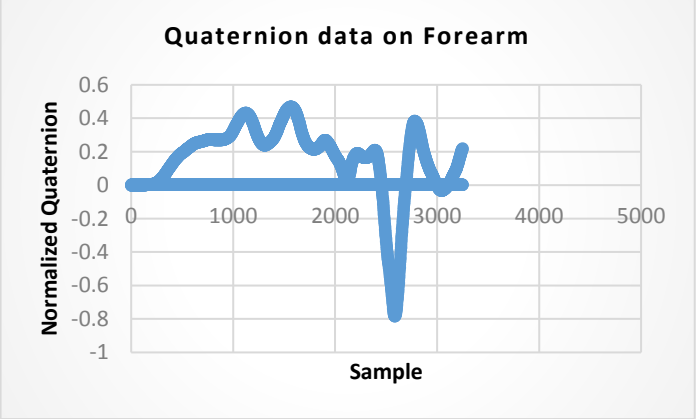


Figure 2: Quaternion data from an IMU on forearm

The above graph depicts data received in the form of a quaternion collected from an IMU on the forearm of the bowler. However, it is difficult to determine, by analysing the above graph if the bowler has an accurate release point during delivery. To achieve this requirement, the bowling window needs to be segmented from the continuous dataset. This requirement to segment the key elements of the technique during fast bowling acted as the main problem identified for this research.

Previous research on applying classification techniques to human movement mainly concentrated on classifying a complete movement like a jump, walk etc. rather than a segment of the complete activity as illustrated by following research. Hence the research problem defined and resolved during this thesis is a unique and novel topic.

1.2.1 Thesis Definition and Objectives

The fundamental objective of the current Master's research is to develop a machine learning model based on statistical parameters, derived by data collected from Inertial Measurement Units (IMU) to classify and segment the three key phases of

fast bowling, which can eventually be used as an automated model for activity segmentation during fast bowling.

1.2.2 Goals

The research work is intended to achieve following goals when presenting an appropriate solution.

Main Goal

I. Develop a machine learning model based on statistical parameters derived from data received by Inertial Measurement Units (IMU) placed on body during fast bowling to classify and segment the three key phases; Run Up, Delivery Stride and Follow Through during fast bowling in cricket.

Sub Goals

I. Determination of the most appropriate Inertial Measurement Unit (IMU) placement position on body providing greatest amount of deviation during fast bowling to assist classification.

II. Analytically identify the best classification method among supervised classification methods to suit human movement classification in fast bowling.

1.3 Limitations

- Accuracy of Inertial Measurement Unit based three-dimensional motion capturing needs to be verified in relation to a high-speed camera based motion capturing system. However, the unavailability of such a system in Sri Lanka is a limitation during for the verification phase of the research.
- The overall performance of machine learning model can be increased by including a large pool of data to the model. However, collection of large volume of data is a challenge due the requirement of testing multiple bowlers.
- Kairos motion analysis system provides a set of processed data in the form of quaternions. This creates a limitation to study the behaviour of raw data and its appropriateness to be used for human movement classification during fast bowling.

- The data undergoes multiple stages of processing prior to being sent to a classifier. This acts as a limitation when reverting to the original dataset to depict the boundaries of each phase during fast bowling.

1.4 Contributions to Society

This thesis aims at addressing the problem of limited usage of technology into sports in Sri Lanka. With current trends and technological enhancements, the world is continuously edging towards further improvements in sports. This has led Sri Lanka to lag other sporting powerhouses. This thesis aims at acting as a spark to ignite the usage of modern technology into Sri Lankan sports. Also, it would create an interest towards more researchers to contribute towards this research area. Further, producing good fast bowlers has been a challenge for Sri Lankan Cricket. And the latter has always relied on natural talent to produce good fast bowlers. However, the current research will act as a catalyst to develop good fast bowlers based on a scientific approach and help eradicate current injury worries which cloud over Sri Lankan fast bowlers. Further, the machine learning techniques elaborated through this thesis can be used as a foundation to be used for other sports like javelin throw, long jump, etc. This in turn will help to develop a new training culture based around technology in Sri Lanka.

Apart from competitive sports this thesis can also be used to assist our communities in health and physical fitness. Many individuals in the modern era have understood the importance of physical activities and exercises to live a healthy life. Thus, many new electronic equipment is being developed to assist individuals to stay healthy. The proposed model can act as a foundation for similar applications in general health physical fitness equipment.

All these factors would eventually contribute to the society by helping Sri Lankan sportsmen to complete better in world competitions and eventually in developing a healthy nation in the long run.

1.5 Publication based on the Research

An abstract based on the proposed work was published at the 26th International Society of Biomechanics (ISB) Congress, 2017 held in Brisbane, Australia from July 23rd to 27th. The abstract was titled 'INERTIAL MEASUREMENT UNIT BASED ACTIVITY SEGMENTATION DURING FAST BOWLING IN CRICKET' and it was included on page 1056 of the full abstract book.

CHAPTER 2

2. LITERATURE REVIEW

2.1 Fast Bowling in cricket

2.1.1 Fast Bowling Action Types and Phases

Fast bowling action has been segmented into three key phases [3] with key activities occurring within those key phases. Below diagram illustrates the key phases including key activities.

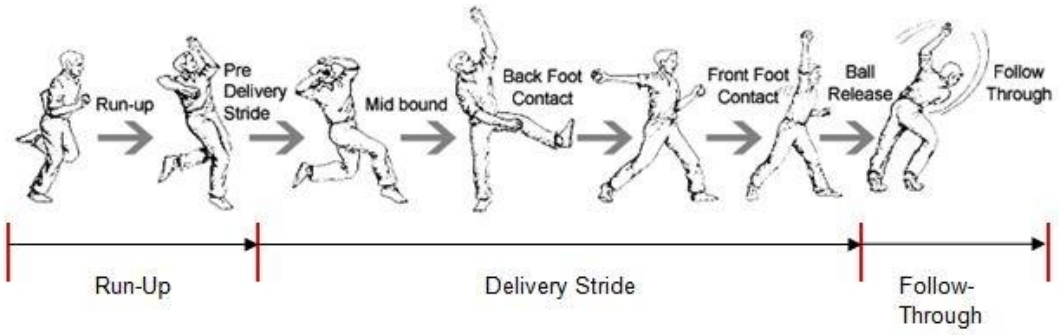


Figure 3: Three key phases in fast bowling action

Previous research has focused around these key phases to understand the contributing factors towards injury about the phases and how each segment contributes to speed of delivery.

Fast bowling consists of three main bowling techniques classified by the alignment of hips and shoulders at either the moment of Back Foot Contact (BFC), Front Foot Contact (FFC) or Ball Release. Following are the three key techniques with the largest contribution from Mixed technique.

- Front on
- Side on
- Mixed

Extensive research has been conducted to understand the biomechanics of fast bowling. A common area of fast bowling analysis is to determine the contributing factors towards increasing speeds in fast bowling. Thus, the different techniques

contributing to this factor has been extensively researched by scholars. Hence research has highlighted following key factors contributing to increasing bowling speeds in fast bowlers [1].

- Quicker run up. Bowlers having faster run ups tend to demonstrate greater bowling speeds.
- Maintaining a straighter knee during front foot contact phase.
- Exhibiting larger amounts of upper trunk flexion up to ball release point.
- Delaying onset of arm circumduction.

These key areas assist coaches in talent identification process and in player performance development as well.

2.1.2 Fast Bowling Injuries

Another aspect of fast bowling analysis helps in injury prevention. Research has demonstrated the different types of injuries occurring in each of the key phases during fast bowling. Run Up and Follow Through have less potential of contributing to injuries. Most injuries occurring during these phases are external injuries rather than internal ones [3]. Common injury threats which occur due to running can be considered in these phases. However, most injuries related to fast bowling occur during the Delivery Stride. The impact due to Front and Back Foot landing creates a large injury risk. Research shows that landing creates a ground reaction force up to six times the weight of the bowler. Most severe injuries created during Delivery Stride are caused due to excessive loading (which creates spinal column compression), arching to the spine and the forceful twisting of the trunk around the spinal column [5]. Another key injury type for fast bowlers is side strain injuries. Research shows that most side strain injuries effect internal oblique rather than external oblique [4]. All these factors highlight the importance of segmenting these three key phases during fast bowling to assist in injury prevention.

2.2 Inertial Measurement Unit selection and Processing

2.2.1 Inertial and Magnetic Sensor Specification

Research on activity classification with wearable sensors has focused on using Inertial Measurement Units (IMU's) which comprises of three axis accelerometers, three axis gyroscopes and three axis magnetometers. The ranges of accelerometer, gyroscope, magnetometer values and resolution depend on the specific application. IMU's used for trick classifications [6] during snowboarding uses +/- 16g accelerometer range, +/- 2000^{o}C gyroscope with 16-bit resolution. When the movement speed increases accelerometer range needs to increase accordingly. But a major constraint at present is locating IMU's with greater accelerometer ranges. In most IMU based applications magnetometer is also included to help eradicate drifting errors which are caused due to gyroscope drifting. Magnetometer assists to provide the horizontal earth's magnetic field and accelerometer provides the vertical acceleration due to gravity which acts as the base for drift compensation [7]. Another key parameter for IMU selection is its physical size. Since most of the IMU based applications are wearable, most studies have focused on physically smaller IMU's to support these applications. IMU developers have managed to reduce the size of the component while also increasing their performance parameters. Previous research [8] on classification of legality of bowling actions uses I^2C (Inter-Integrated Circuit) for data communication between IMU's and microcontroller and Bluetooth to transmit sensor data to a computer for storage for post processing.

2.2.2 Sampling Rates

For classification of legality of bowling actions [8] sensor sampling rate of 150Hz has been used. Sampling rate ideally depends on the application. Activity classification for high speed movement patterns like fast bowling requires greater sampling rates. A study on [9] Bowler analysis in cricket using centre of mass inertial monitoring, uses a sampling rate of 200Hz. Such sampling rates may be suitable for spin bowling analysis. Greater sampling rates will be required for fast bowling analysis when using IMU based systems. Vision based motion analysis

systems (ex: Vicon) have been used in research for fast bowler analysis operating at 300Hz [1]. Typical range of sampling rate has varied from around 200Hz to 500Hz.

2.2.3 Orientation Estimation

Orientation estimation has been used throughout literature on motion analysis system developments based on IMU's. Kalman Filter based orientation estimation centred on inertial and magnetic sensors is one of the most common methods [7]. However, quaternion based orientation estimation algorithm proposed by Sebastian Madgwick is used extensively in modern research for orientation estimation based on IMU's [10].

2.3 On Body Sensor Position for Classification

Sensor placement for IMU based bowling action legality classification used three IMU sensors placed as depicted below.

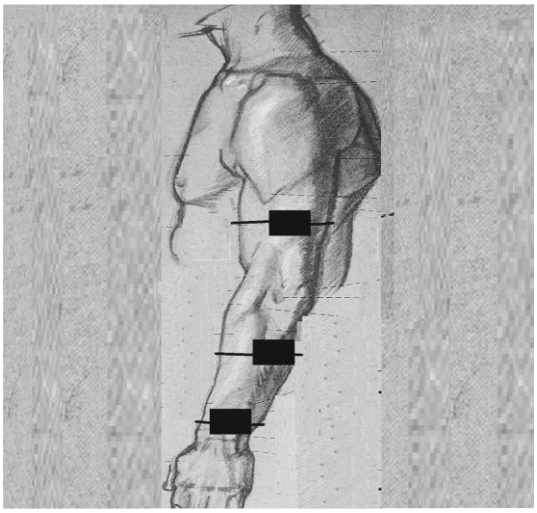


Figure 4: Wearable sensor placement positions to detect throwing in cricket [8]

Most researches have examined the correlation between sensors placed at different positions on the body for activity classification. A specific position on body has the possibility of better supporting a certain activity classification [11]. Below Table 1 illustrates the summary [11] of classification performance of different human activity classification tasks by using accelerometers. It illustrates performance created by individual and multiple sensors on different body positions.

Table1: Review of studies on accelerometer placement for activity recognition [11]

Sensor Placement Position on Body													Classified Activity	Average Classification Accuracy %
Waist	Wrist	Thigh	Side	Necklace	Chest	Hip	Lower Back	Trunk	Shanks	Ankle	Pocket	Hand		
✓													Walking, Falling	90.8
✓													Falling, walking, sitting, standing, lying	98.9
	✓												Walking, running, scrubbing, etc	95
	✓	✓		✓									Typing, watching TV, drinking, etc	91.5
	✓				✓								Lying, sitting, walking, rowing, etc	83.3
	✓				✓	✓							Sitting, running, walking, etc	92.13
							✓						Lying, sitting, workingon a computer, etc	93
✓		✓											Sitting, lying, standing, walking speed	100
		✓						✓					Siting, lying, standing, moving	92.25
								✓	✓				14 daily living activities	-
✓		✓			✓					✓			Lying, sitting, standing, all fours, etc	91
		✓			✓					✓			Stairs ascend, descend, walking, etc	90.3
		✓			✓						✓	✓	Slow walking, fast walking, running, etc	91.15
										✓			16 daily living activities	89.08
✓													Walking, running, sit-to-stand, stand-to-sit, etc	98
	✓												Long term activities	98
✓		✓	✓		✓								Standing, sitting, lying, walking, transition	96.4
								✓					Sitting, standing, walking, lying	90.4

2.4 Pattern Recognition and Machine Learning Techniques

Accelerometer and IMU based systems have been greatly used for applications and research around automatic activity classification during human movement. Wearable sensor based systems have been preferred in pattern recognition applications on humans due to following reasons [12].

- Low cost
- Immune to occlusions & interference
- Self-contained

2.4.1 Activity Classification for Non-Cricket Activities based on IMU's

Human activity classification based on wearable sensors have been used around many aspects of research. However, it is unclear to determine if classification models

developed around raw data or extracted features yield better results. Pattern recognition models developed by data obtained from IMU's have been used to classify human movements like walking, sitting, standing, etc. [11]. These models have been extensively used in sports related movement analysis with IMU sensors. One such application is to understand the patterns generated during golf swing [13]. Sensors mounted on specific body parts generate specific patterns during repetitive golf swings. Pattern recognition techniques based on IMU data are also used to classify strong, weak and sideways movements during drumming [14]. IMU sensors mounted on skateboards have been used to classify different tricks performed [6]. The classification model classifies tricks such as Ollie, Nollie, Kick flip, etc. during skateboarding. Feature extraction and classification models have also been used to classify jumps during skiing and skateboarding based on head mounted IMU sensors [16].

2.4.2 Activity Classification for Cricket Related Activities based on IMU's

Most common classification related problem for bowling is centred on determining if a certain bowling action is legal or not. Most research uses vision based systems to segment the bowling window to analyse if the action is legal or not. However, modern research has also used wearable sensors to collect three-dimensional rotational data and used classification techniques such as k-Nearest Neighbour, Naïve Bayes, Random Forest, etc. to classify the legality of bowling actions [8]. Initial research on usage of wearable sensors in cricket has used inertial sensors placed at the centre of mass of a 'Front On' fast bowler to determine Run Up speed, Pre-Delivery Stride length and hip rotational angle [9]. Unsupervised classification methods such as Hidden Markov Models have been used to classify arm rotation during bowling based on statistical features [16].

For pattern recognition using wearable sensors, the process throughout literature can be classified into four key areas [12] [6] [17] as depicted below.

2.4.3 Event Detection

In previous studies [6], event detection was used to determine the exact time intervals that included the required activities. This had reduced the amount of data

that was to be processed in subsequent classification. Event detection has been used by developing data frames around the input sensor data [6] [12]. The data from the sensors were segmented into windows based on possible timeline of specific activities. For example, [6] 1s windows with 0.5s overlap was chosen for trick classification in snowboarding considering length of a trick and its duration. A certain threshold energy level was also defined allowing it to determine if a trick was present upon exceeding the threshold level. This has a similar impact on the current research as the exact starting and ending point of fast bowling will need to be determined for activity classification during bowling.

2.4.4 Feature Selection

Feature selection is the next key part in the activity classification algorithm. Different researchers have adopted different strategies for feature extraction. In [6] feature vectors are calculated for different trick activities. Statistical parameters such as mean, variance and skewness were included. In other examples [17], discrete methods were used for feature extraction. The Discrete Wavelet Transform (DWT) was used to extract discriminative features from accelerometer data. This was achieved by decomposing the original sensor signal into several scaled and time shifted versions of a selected mother signal. In [18] Daubechies four wavelet (db4) wavelet was used as the mother wavelet for the decomposition. But in terms of relevance to current research features will need to be computed for every window in the input data.

Research segments features into two main categories [11],

- Time domain features – Statistical features such as mean, mode, variance, skewness, etc. These are used extensively in human activity recognition.
- Frequency domain features – The signal or data converted into frequency domain, mainly by Discrete Fourier Transform (DFT) representation has triggered a specific set of features. Power Spectral Density (PSD), Peak frequency, Entropy, DC component, etc. are some of the frequency domain features used for human activity classification.

2.4.5 Feature Extraction

The feature extraction approach consists of detecting and discarding the features that are demonstrated to minimally help to cause a correct response by the classifier [12]. Usually, the feature extraction step is implemented via sub-optimal search algorithms, such as, for instance, the branch-and-bound search, the Sequential Forward-Backward Selection (SFS-SBS) [20]. Including features providing minimum effect towards classification tends to increase computational time of the classifier. Hence throughout literature feature extraction methods are used to identify the most suitable features to be input into the algorithm [11]. There are three main methods for feature extraction.

- Filter methods
- Wrapper methods
- Hybrid methods

Increase in the number of features corresponds to the 'curse of dimensionality'. This creates difficulty in visualization of output from classification. Hence, techniques have been used by researchers to reduce dimensionality of features. One such method is Principal Component Analysis (PCA). A study [21] on classifying activities based on data obtained from a mobile phone accelerometer and gyroscope reveals that PCA was used to reduce 561 features to 70 principal components. This reduced computational time of classifier from 658.53s to 128s.

Since many features are required for the classification, the data set in most studies is divided into two parts. Namely,

- Training Set
- Test Set

Previous research [12] states that ratio between the number of instances available in the training set and the dimension of the feature-space must be at least ten.

2.4.6 Classification

The extracted features act as input to different classification techniques. These classifiers are mainly divided into two segments [11],

- Supervised classification approaches
    - *k-Nearest Neighbour (k-NN)* – k-NN classifier has been used extensively for human activity classification. In research [11] to classify different physical activities such as walking, standing, running, etc. k-NN algorithm provides the highest accuracy among all classifiers with an accuracy of 99.25%. For IMU based trick classification [6] k-NN provided the fastest response time of 5.2s at an accuracy of 96%. To develop a machine learning model based on accelerometers on body [12] has used k-NN as a single frame classifier with a single frame. However, the exact value for 'k' as the number of neighbours has varied from one research to another.
    - *Support Vector Machines (SVM)* – Usage of SVM's for human activity classification has demonstrated variable accuracy percentages. But in most cases, it has reached above 90% accuracy levels [11] [6]. However, in skateboarding trick classification [6] SVM is the slowest among the classifiers with a classification time of 37.2s.
    - *Random Forest (RF)* – An algorithm based around the combination of multiple decision trees is also used in many research topics related to human activity recognition. In a research [22] on developing a machine learning model to classify hand movement during dumbbell based exercise uses a RF with 99.97% accuracy of classification. Also, a research [11] on human movement classification uses a RF at 98.95% accuracy to classify movements such as walking, standing, running, etc.
    - *Naïve Bayes (NB)* – NB is also used [12] as a single frame classifier to classify common human movements such as walking, running etc. In skateboarding trick classification [6] NB provides 97.8% accurate

classification at a computational time 6.2s. This shows that NB is a very effective classifier.

Other supervised classification techniques such as Supervised Learning Gaussian Mixture Models (SLGMM), A-NN etc. are also used for human activity classification. However, the general approach is to use few different classifiers for the same task and compare their accuracy. But the approaches to develop specific classes for the training set used for supervised classification are not clearly documented in literature.

- Unsupervised classification approaches
    - k-Means – In comparison to supervised methods k-Means demonstrates low accuracy rates for human activity classification. It reaches around 72.95% accuracy in classification [11]. But research doesn't demonstrate enough details about setting the cluster centre points.
    - Hidden Markov Models (HMM) – HMM's demonstrate the best classification accuracy among unsupervised classifiers [11]. This is demonstrated throughout literature.
    - Gaussian Mixture Models (GMM)

Unsupervised classification approaches have been mainly used in situations where development of classes for training set becomes difficult in human activity classification.

2.4.7 Classification Evaluation

Evaluation of activity classification algorithms have been carried out in few different ways in research. One key aspect is to evaluate the classifiers. In [6] sensitivity and specificity for the detection of trick events were calculated in relation to the number of all segmented windows. In this early stage of the project, the evaluation of the classification was only based on correctly detected trick events. For the above classification evaluation was based upon leave-one-subject-out cross-validation. Evaluation of classifiers is a key activity in most human movement activity

classification research. Below table depicts the response time of four classifiers used in automatic activity classification [18].

Classifiers	**RF**	**NB**	**Lazy IBK**	**MP**
Testing Time (s)	0	0.15	0.38	0.08
Training Time (s)	0.38	0.1	0	538.9

Figure 5: Random Forest, Naive Bayes, Lazy IBK, Multilayer Perceptron response times [18]

The key method for evaluation in most research based on human activity classification is to develop cross validation methods. For human activity classification, such as walking, running, etc. 10-fold cross validation is used [11]. But throughout literature validation of human activity classification is developed around below statistical verification parameters.

- Accuracy
- F-Measure
- Precision
- Recall
- Specificity

Table 2: Classifier performance evaluation [11]

Performance of Supervised Algorithms					
Classifier	*Accuracy +/- std (%)*	*F-Measure (%)*	*Recall (%)*	*Precision (%)*	*Specificity (%)*
k-NN	96.53 +/- 0.20	94.6	94.57	94.62	99.67
RF	94.89 +/- 0.57	82.87	82.28	83.46	99.43
SVM	94.22 +/- 0.28	90.66	90.98	90.33	99.56
SLGMM	94.22 +/- 0.28	69.94	69.99	69.88	98.39
Performance of Unsupervised Algorithms					
Classifier	*Accuracy +/- std (%)*	*F-Measure (%)*	*Recall (%)*	*Precision (%)*	*Specificity (%)*
HMM	80 +/- 2.10	67.67	65.02	66.15	97.68
K-means	68.42 +/- 5.05	49.89	48.67	48.55	93.21
GMM	73.6 +/- 2.32	57.68	57.54	58.82	96.45

Another evaluation parameter on similar research is the [6] correlation coefficients between each pair of accelerometer signals. They are obtained by computing the dot

product of pairs of frame vectors, normalised to their length, and are highly helpful in discriminating activities that involve motions of several body parts.

However, research has concentrated on other validation methods like cross checking the results with a secondary system such as an optical system. The activity classifications obtained by the algorithms were checked against results obtained by a system like a ‘VICON system’. This has helped to improve the overall effectiveness of those researches.

CHAPTER 3

3. ON BODY SENSOR POSITION SELECTION METHODOLOGY

This thesis uses Kairos 3-D motion analysis system for data collection. Kairos uses Inertial Measurement Units to provide quaternion based 3-dimensional motion data during movement. The first step in the research was to determine the most appropriate sensor position on body that would provide the best accuracy for classification of the three key phases during fast bowling.

The methodology adopted to determine the best body location and quaternion number for classification was to fit the data into a classifier and determine its accuracy for each position and quaternion. The position which has the highest accuracy factor can be considered as the suitable location for final data collection for segmentation of key three phases in fast bowling by using an IMU sensor. Therefore, a pattern recognition algorithm is developed initially to determine the best sensor on body sensor position for the study.

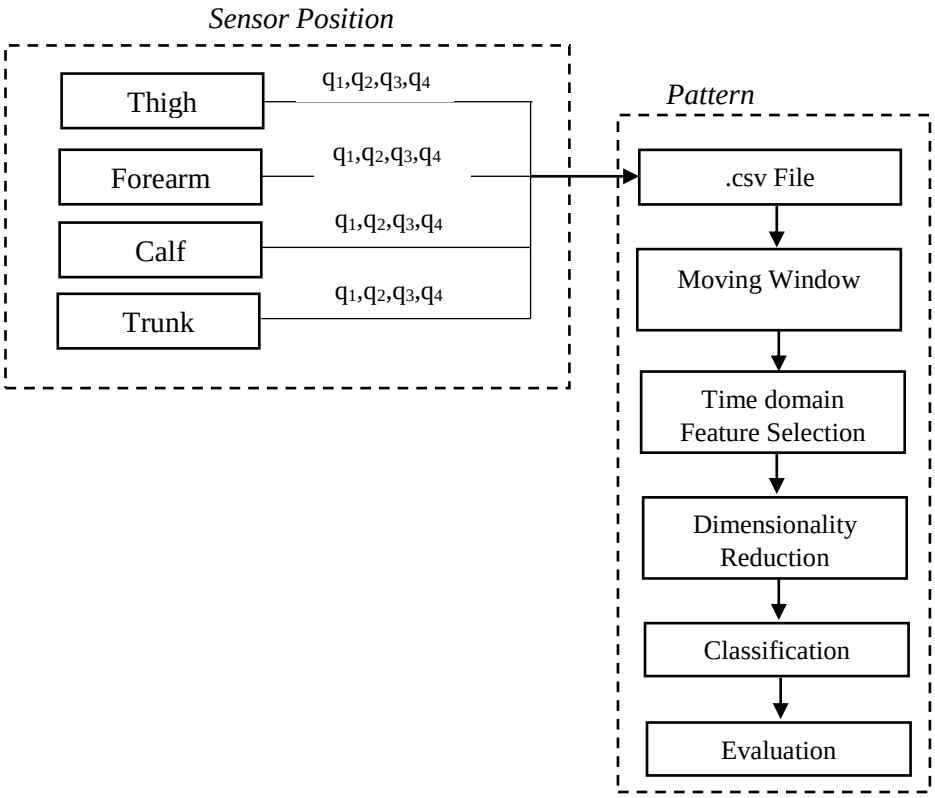


Figure 6: Proposed system flow chart

3.1 Sensor Positions

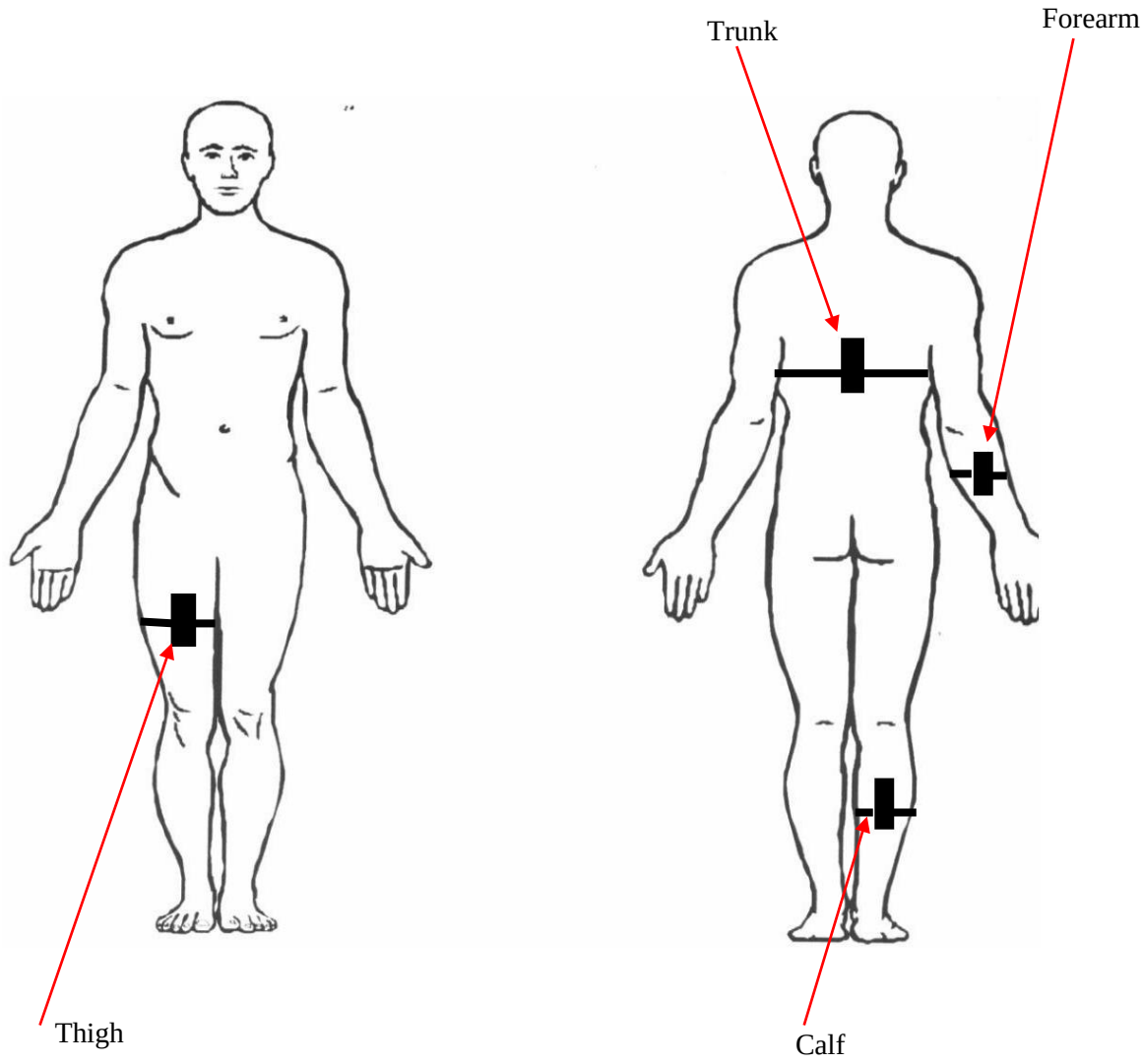


Figure 7: IMU placement positions on body for data collection

Four positions on the body were considered as potential sensor placement areas.

- Thigh – Sensors were placed on the front leg (left leg for right arm bowlers and vice versa).
- Forearm – Sensors were placed on the bowling arm (right arm for bowlers delivering with right arm and vice versa)
- Trunk – Sensors were placed on the upper trunk.
- Calf – Sensors were placed on the front leg (same as the thigh).

3.2 Feature Selection

As discussed previously in literature, various feature selection methodologies have been used previously to determine best features for similar applications. Therefore, as used in most cases a moving window is used for obtaining the features. Each window comprises of 20 samples and a window overlap of 50%. This was done independently for every quaternion on each body sensor position. Following were the calculated features,

Figure 8: Feature selection moving window

Among the three types of features, time domain features were used for this analysis. Hence, eight-time domain features were calculated for each sliding window.

$$\text{Mean } (Y) = \sum_{i=1}^{N} Y_i / N$$

$$\text{Median } (Y) = Y_{(N+1)/2} \quad \text{if N is odd}$$

$$Y = \frac{(Y_{(N/2)} + Y_{(N/2)+1})}{2} \quad \text{if N is even}$$

$$\text{Variance } (S^2) = \frac{\sum_{i=1}^{n} (Y_i - Y)^2}{(N-1)}$$

$$\text{Skewness} = \frac{\sum_{i=1}^{n} (Y_i - Y)^3}{(n-1) \times S^3} \qquad S \text{ is Standard Deviation}$$

$$\text{Kurtosis} = \frac{\sum_{i=1}^{n} (Y_i - Y)^4}{(n-1) \times S^4} \qquad S \text{ is Standard Deviation}$$

Apart from above features Root Mean Square (RMS), Median Absolute Deviation (MAD) and Inter Quartile Range (IQR) were also used as features for the study.

3.2.1 Feature Scaling

Once the features were selected it was observed that certain features were out of scale. Hence a standardization step was required prior to dimensionality reduction. In this approach the mean and standard deviation of entire feature vector was calculated. The dataset was scaled by subtracting every element by the mean and dividing by the standard deviation.

3.3 Feature Extraction (Dimensionality Reduction)

The next step was to reduce dimensionality of the features. Various techniques have been used previously for the task of dimensionality reduction. Backward Elimination was used as the first method for feature extraction. However, through this method certain datasets reduced to more than three features. This method produced a visualization difficulty of the data since some cases represented more than two or three features. Hence, Principal Component Analysis (PCA) was used as a dimensionality reduction technique.

3.3.1 Principal Component Analysis (PCA) for Dimensionality Reduction

To minimize over fitting and for visualization purposes PCA was used for dimensionality reduction. PCA transforms the original variables into a new set of small variables without losing the most important information of the original data. Owing to requirements of visualization in this study the original dataset was transformed into two principal components. This is achieved by assuming directions with largest variances as the most important. In this instance PC1 (First Principal Component) and PC2 (Second Principal Component) are generated and they are orthogonal to each other with PC1 acting as the most important direction.

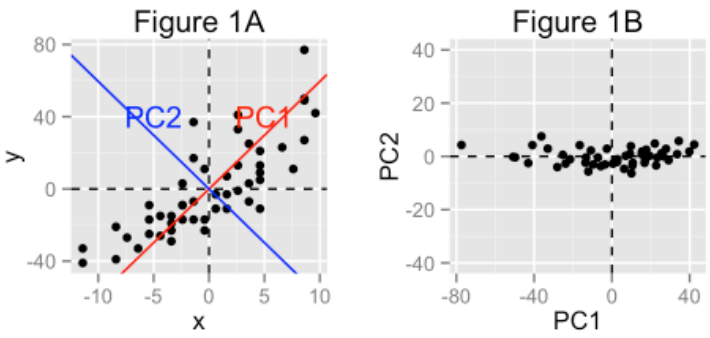


Figure 9: PC1 and PC2 orthogonality interpretation [23]

3.4 Classification

Support Vector Machines (SVM) have been used for both classification and regression tasks. Throughout literature SVM's have been used for human movement classification as a supervised classifier. However, in most instances k-NN have performed better in human movement classification compared to SVM's. But in this scenario, it required one classifier to compare different sets of data. In k-NN selecting correct 'k' number across all datasets was challenging. Hence a SVM was more suitable in this instance. Following characteristics in SVM were also considered for its selection.

- Suitable for instances with less number of classes. In this instance, there were three classes (ideally two classes).
- Suits classification with higher number of features. Current classification consisted of eight features.
- When there is non-uniform weighing among features.

In SVM's features are mapped into high dimensions and a corresponding hyper plane is selected to best classify the results. However, it is worthwhile to that application of PCA reduces dimensionality prior to classification. Therefore, a linear 'kernel' was used for the SVM for classification.

3.5 Evaluation

10-fold Cross Validation was used to evaluate every model. The dataset was divided into ten folds where one sample acted as the test set and the others as training set. For each fold Accuracy, Precision and Recall were calculated. This was repeated 10 times and the average of each parameter was considered as the final value. And finally, F-measure was calculated from the averages of Precision and Recall. On-body sensor position and quaternion providing the best values among the evaluation parameters were selected as the suitable quaternion and on body sensor position for final data gathering for fast bowling phases classification.

$$\text{Accuracy} = \frac{T_p + T_n}{T_p + T_n + F_p + F_n} \qquad \text{Precision} = \frac{T_p}{T_p + F_p} \qquad \text{Recall} = \frac{T_p}{T_p + F_n}$$

$$\text{F – Measure} = 2 \times \frac{(\text{Average Precision} \times \text{Average Recall})}{(\text{Average Precision} + \text{Average Recall})}$$

Where, Tp= True Positive F_p= False Positive

F_n= False Negative T_n= True Negative

These parameters were derived based on the confusion matrix generated for each classification. Below is an example 3x3 confusion matrix. Where, Accuracy would be indicated by sum of number diagonal items divided by total instances.

y_pred:	**1**	**2**	**3**
1	13	0	1
2	6	31	1
3	9	0	27

Precision would be defined from the confusion matrix as the ratio of number of correctly classified instance per class to the number of predictions per class. Whereas Recall would be ratio of number of correctly classified instance per class to the number of instances per class.

3.6 Participants

Three participants were selected for the initial data gathering to determine sensor position on body that would provide best accuracy results for classification. All

participants belonged to ‘Mixed type’ fast bowling action type. All three participants were active cricketers. Official consent was obtained from each participant to participate in the data gathering and to take photos and videos during the session.

Table 3: Bowlers age, height and weight

Bowler Number	Age	Height (cm)	Weight (Kg)
1	27	164	63
2	17	172	60
3	17	170	65

3.7 Data Gathering Methodology

Initial data gathering was conducted at Cric Sri Lanka indoor cricket nets. Sensors mounted using Velcro straps were placed on specific positions on body and the subjects were requested to bowl with the sensors.

Each subject delivered five deliveries. One critical parameter for the classification model was to derive the separate classes for Run up, Delivery Stride and Follow through. Therefore, data gathering was conducted separately for each class.

Table 4: Data sample generation per bowler

Class	Number of Iterations
Full bowling action	5
Run up	4
Delivery stride	4
Follow through	4

Figure 10: Second bowler

Figure 11: Third bowler

3.7.1 Data Types

To reduce the number of repetitions, data were gathered in combination of sensor positions. Prior to initiating each delivery, the bowler maintained a stationary position to assist initial sensor calibration. A clapper was used as a benchmark to initiate movement and data storage. When data were collected for the classes, bowler conducted Run Up, Delivery Stride and Follow Through separately. These acted as the data for each class of classification.

The orientation estimation was based around Madgwick's orientation estimation filter [10]. This filer is a quaternion based filter. Hence each data sample provided four quaternion values. Each of these four quaternion values were normalized and stored as data, where each of the quaternions was examined to determine accuracy of classification. A quaternion has four parts. It is like a complex number with one real component (q_0) and three complex components (q_1, q_2, q_3) which can be used to represent a 3D rotation. A quaternion can be considered as a hypercomplex number.

$$q = q_0 + q_1 i + q_2 j + q_3 k = [q_0, q_1, q_2, q_3]$$

Where,

q_0 = quaternion real component

q_1, q_2, q_3 = quaternion imaginary components

i,j,k = imaginary basis vectors with $i^2 = j^2 = k^2 = -1$

By visualization it will be easier to understand a quaternion in reference to the rotation created by rotating frame B to A as illustrated below. In a nutshell it represents the orientation of frame B in reference to frame A.

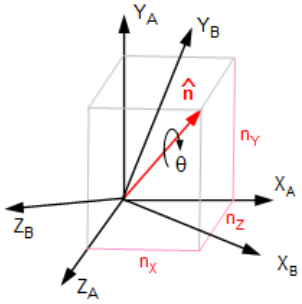


Figure 12: Quaternion generation from IMU for movements [10]

$q_0 = \cos(\theta/2)$

$q_1 = n_x \sin(\theta/2)$

$q_2 = n_y \sin(\theta/2)$

$q_3 = n_z \sin(\theta/2)$

Where,

q_0 = quaternion real component
q_1, q_2, q_3 = quaternion imaginary components
θ = rotation angle
n_x, n_y, n_z = rotation axis components

In the quaternion number, scalar component represents rotation angle and others represent direction of rotation. All these values were stored in .csv file for post processing. Each .csv file contained Run Up, Delivery Stride and Follow Through per quaternion. And the class number was stored per each class along with the data sample.

3.8 Madgwick Filter

- It is specially designed to be used for inertial based sensors.
- It has the capability of reaching the accuracy obtained by a Kalman Filter with less mathematical complexity.
- Suitable for operations with high sampling rates.

Due to the above reasons this filter was selected as the orientation estimation filter to be used in this system. Below are the generalized operational points of the filter.

- Accelerometer, Gyroscope and Magnetometer Normalization.
- Finding reference direction of earth's magnetic field.
- Gradient descent algorithm as a corrective step.
- Rate of change of quaternion calculation.
- Integrate to yield quaternion.
- Quaternion normalization.

3.9 Drift Compensation

In the current design an initial magnetometer calibration was conducted. This was performed to compensate for the gyroscope drifting by considering magnetometer reading as a reference.

CHAPTER 4

4 ON BODY SENSOR POSITION SELECTION DATA ANALYSIS AND RESULTS

Initial step in the analysis was to visualize the output obtained from each sensor position. Below diagrams represent data variation during full bowling action for subject 1.

4.1 Original Data Plots on Sensor Positions

All data plots pertaining to subject 1 has been plotted.

Subject 1

Calf

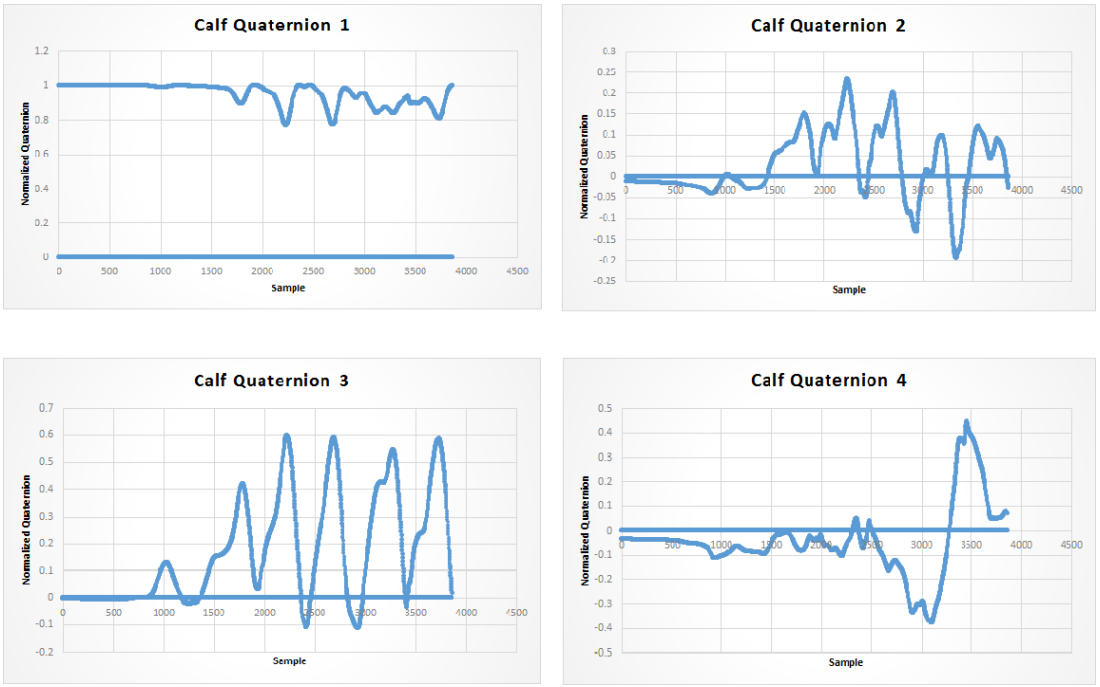


Figure 13: Quaternion data for full bowling action of first bowler from IMU on Calf

The initial graphs developed from the sensor on the Calf demonstrated consistent fluctuations among all quaternions. However, q_1, q_2 and q_3 showed consistent deviations and q_4 showed variations through the graph, which may indicate boundaries for different classes of movement.

Forearm

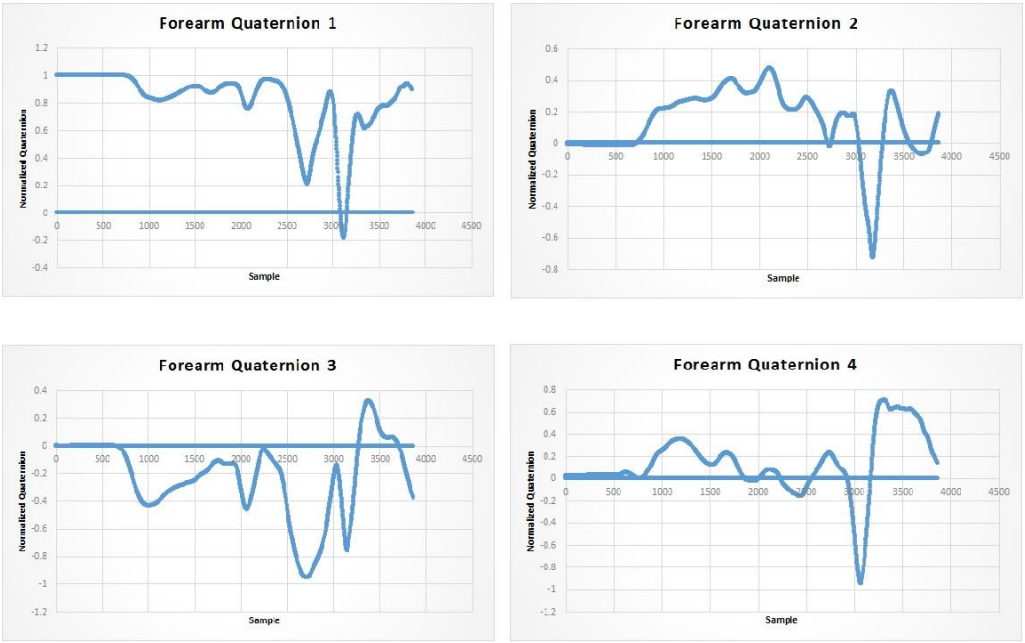
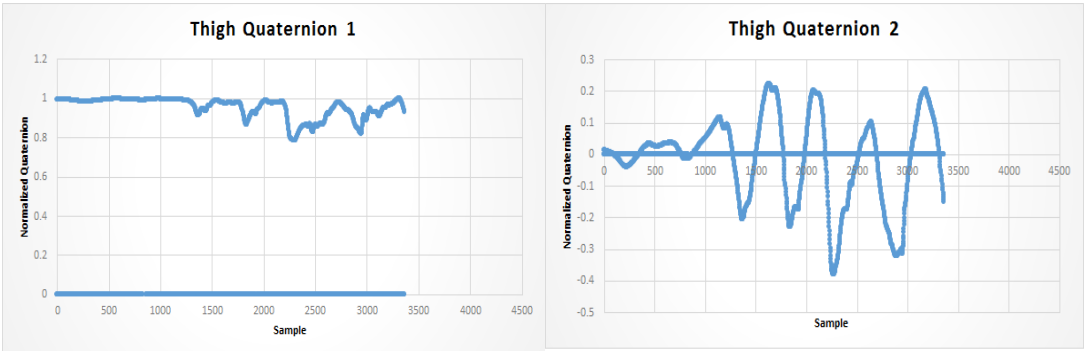


Figure 14: Quaternion data for full bowling action of first bowler from IMU on Forearm

In comparison to the data plot from Calf all quaternion data from Forearm demonstrated higher fluctuations/variations throughout the plot. Third quaternion depicts least inter class deviations.

Thigh



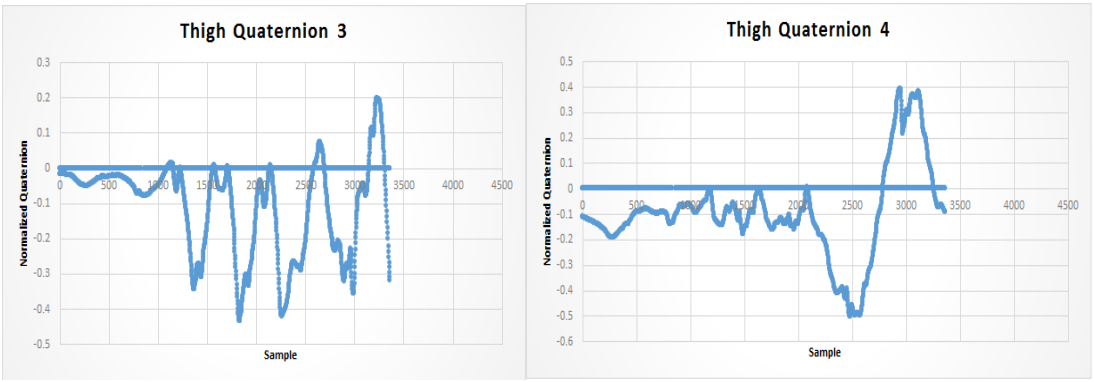


Figure 15: Quaternion data for full bowling action of first bowler from IMU on Thigh

Graphed data plot from Thigh demonstrated similarities to the data from the Calf. Only q_4 demonstrates higher variations in the plot which can indicate possibility of the existence of class boundaries.

Trunk

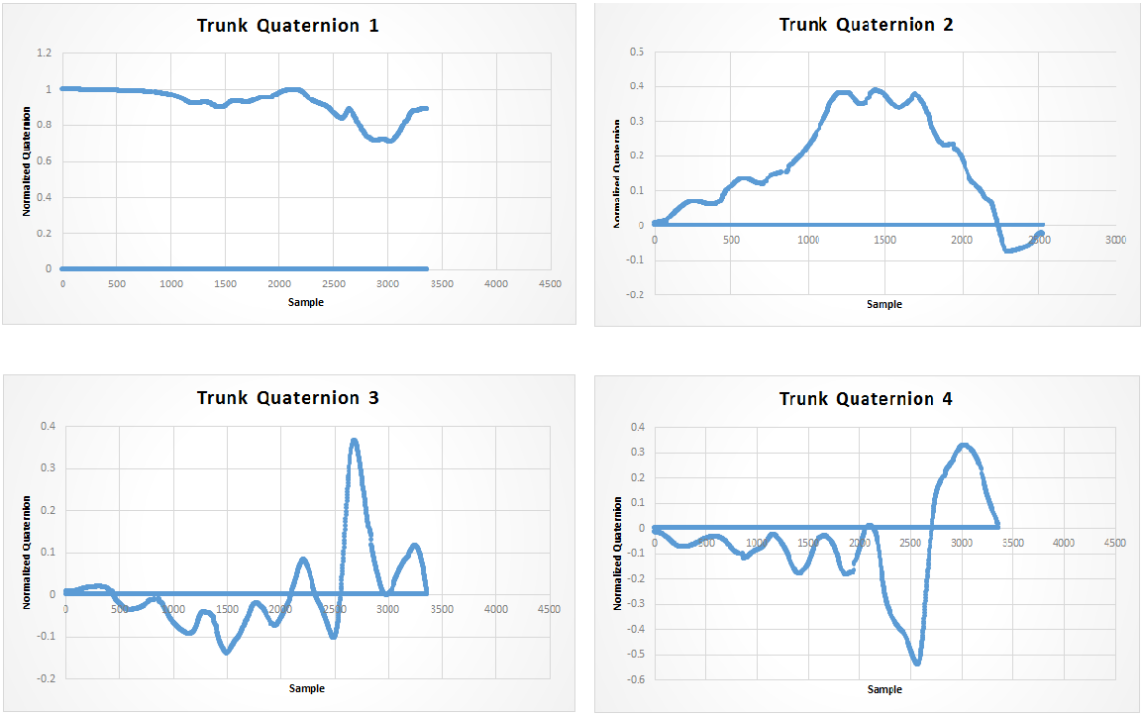


Figure 16: Quaternion data for full bowling action of first bowler from IMU on Trunk

From the sets of data received (plot) from the Trunk q_3 and q_4 demonstrated higher variations. This suggested the existence of observable boundaries for the classes, whereas q_1 and q_2 represented less likelihood of clear boundaries for the classes.

Note

Overall, in all data sets q_2 and q_4 represented best observable boundaries for classes. And among the datasets, data from the forearm showed best suitability for classification.

4.2 Definition of Classes

To define classes, all subjects performed deliveries in below sequence. Data collection was initiated and ended visually at below specified positions for each class.

- Full delivery
- Run Up
- Delivery Stride
- Follow Through

Table 5: Definition of classes for classification

Segment	Beginning	End
Run Up	First Clap	Pre-delivery stride end
Delivery Stride	Mid Bound Beginning	Ball Release
Follow Through	Ball Release	Final Clap

Delivery Stride Class – Subject 2

Figure 17: Delivery Stride – Subject 2

4.3 Feature Selection

As discussed previously eight-time domain features were calculated for every sliding window. R Studio was used as the machine learning software tool for the analysis.

Table 6: Feature data plot with classes

	V1	V2	V3	V4	V5	V6	V7	V8	V9
1	-1.533190e-03	-0.001530247	6.574971e-09	0.0001501767	0.059495653	-1.7923856	0.0015352255	1.337224e-04	1
2	-1.443703e-03	-0.001430531	1.879499e-09	0.0000529790	-0.468165384	0.4261740	0.0014443226	0.000000e+00	1
3	-1.376817e-03	-0.001363160	2.575945e-09	0.0001130190	0.030593297	-1.8397010	0.0013777079	6.767772e-05	1
4	-1.331901e-03	-0.001317512	3.423324e-10	0.0000239140	-0.734103013	-1.2223746	0.0013320231	0.000000e+00	1
5	-1.334593e-03	-0.001341426	1.225456e-10	0.0000239140	0.948683298	-1.2766440	0.0013346372	0.000000e+00	1
6	-1.305935e-03	-0.001341426	2.734419e-09	0.0001129910	0.788078737	-1.4734663	0.0013069314	0.000000e+00	1
7	-1.249956e-03	-0.001228435	2.275339e-09	0.0000000000	-1.238067384	-0.2045444	0.0012508223	0.000000e+00	1
8	-1.214773e-03	-0.001206701	2.249563e-10	0.0000217340	0.678521701	-0.6662755	0.0012148612	3.222283e-05	1
9	-1.173169e-03	-0.001182787	1.581131e-09	0.0000673710	0.842002168	-0.9249787	0.0011738109	3.545490e-05	1
10	-1.145419e-03	-0.001134429	1.278381e-09	0.0000578640	-0.132931675	-1.5161145	0.0011459500	4.268405e-05	1
11	-1.066226e-03	-0.001105639	1.102321e-08	0.0001634290	0.606732945	-1.1282791	0.0010711378	1.071356e-04	1
12	-7.554317e-04	-0.000854000	8.555469e-08	0.0004320000	0.575146545	-1.0975187	0.0008075628	3.090376e-04	1
13	-5.891438e-05	-0.000156000	3.451582e-07	0.0009160000	0.397735655	-1.2767132	0.0005763619	6.567918e-04	1
14	1.051305e-03	0.001007098	6.411030e-07	0.0013369620	0.074229457	-1.4087802	0.0013098920	1.047997e-03	1
15	2.441404e-03	0.002353466	8.451884e-07	0.0014905870	0.081659181	-1.3756101	0.0026010373	1.183352e-03	1

When time domain features were obtained, it was observed that few features were out of scale. Hence a feature scaling step was necessary prior to classification.

Comparison between multiple features

Calf

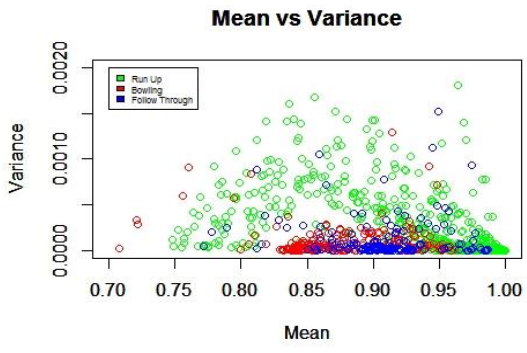


Figure 18: Feature plot for q_1 on Calf

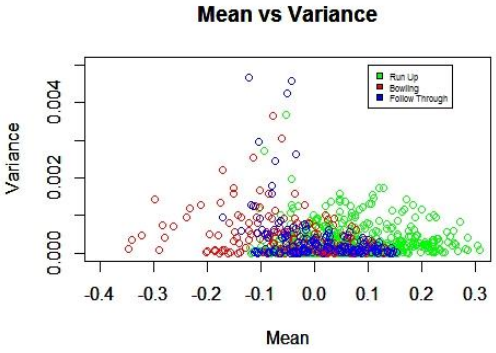


Figure 19: Feature plot for q_2 on Calf

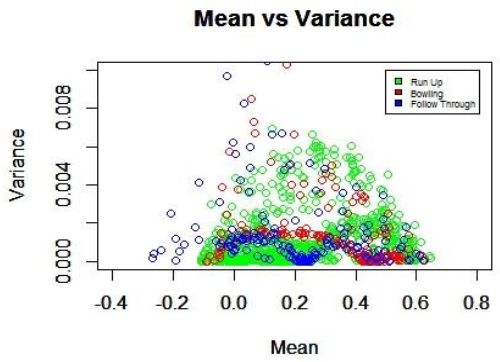


Figure 20: Feature plot for q_3 on Calf

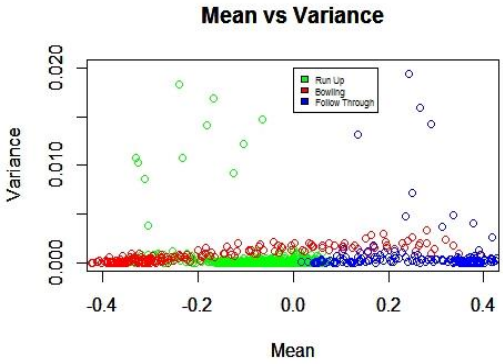


Figure 21: Feature plot for q_4 on Calf

Forearm

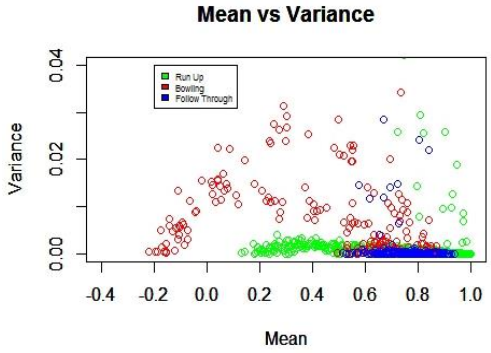


Figure 22: Feature plot for q_1 on Forearm

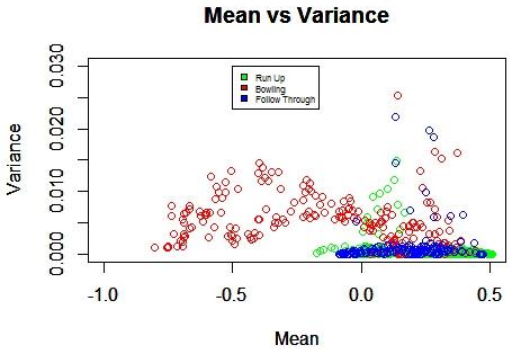


Figure 23: Feature plot for q_2 Forearm

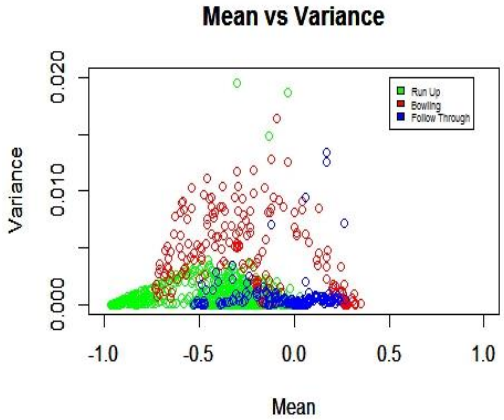


Figure 24: Feature plot for q_3 on Forearm

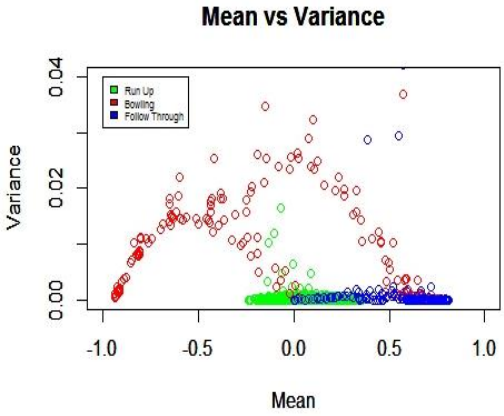


Figure 25: Feature plot for q_4 Forearm

Thigh

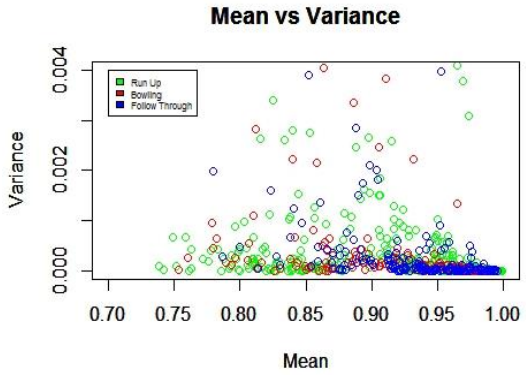


Figure 26: Feature plot for q_1 on Thigh

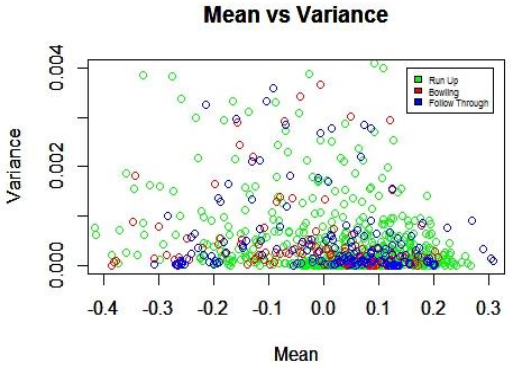


Figure 27: Feature plot for q_2 on Thigh

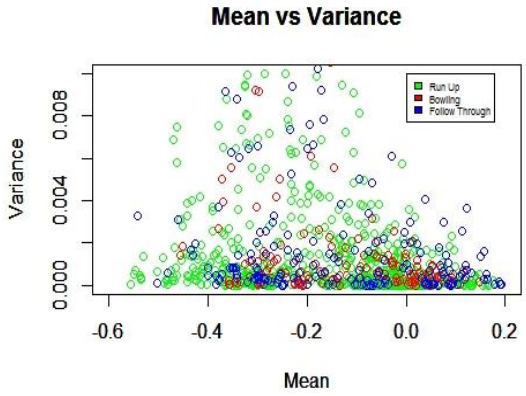


Figure 28: Feature plot for q_3 on Thigh

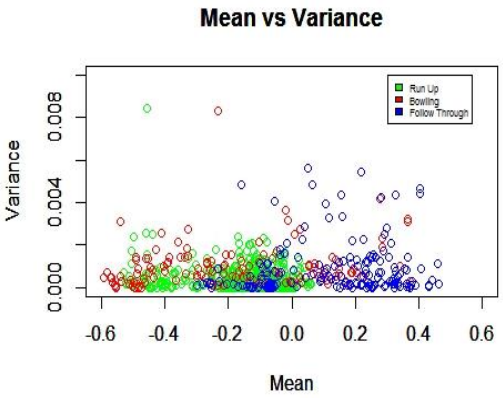


Figure 29: Feature plot for q_4 on Thigh

Trunk

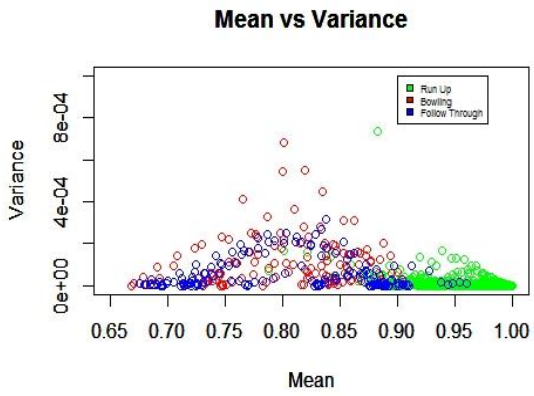


Figure 30: Feature plot for q_1 on Trunk

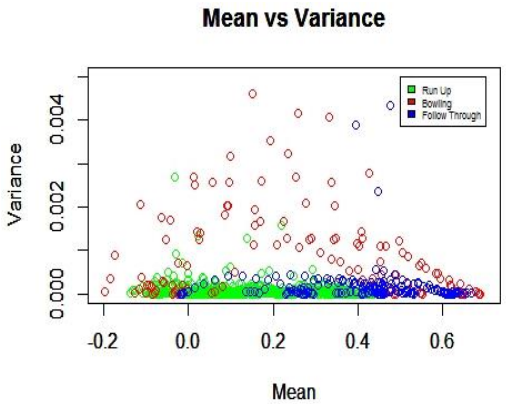


Figure 31: Feature plot for q_2 on Trunk

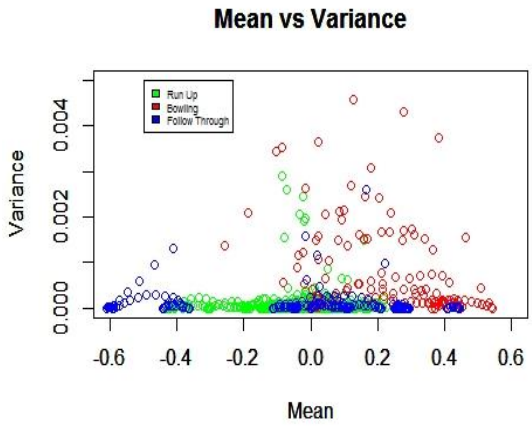


Figure 32: Feature plot for q_3 on Trunk

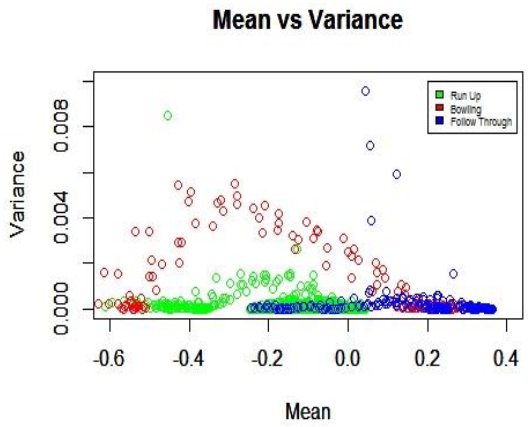


Figure 33: Feature plot for q_4 on Trunk

4.3.1 Feature Scaling

Features which were not scaled, tend to have less significance in the classification model. It was observed in the original feature set that some of the features were not scaled. For example, variance and median average deviation observe to be non-scaled. Hence a feature scaling step was conducted to scale all the features.

4.4 Dimensionality Reduction

Dimensionality reduction was conducted using Principal Component Analysis. The dataset was transformed into two main components (PC1 and PC2). Following data were generated for data from q_1 of forearm sensor.

- Generating correlation matrix

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
[1,]  1.00  1.00 -0.44 -0.60 -0.29  0.09  0.99 -0.67 -0.33
[2,]  1.00  1.00 -0.44 -0.60 -0.31  0.09  0.99 -0.66 -0.33
[3,] -0.44 -0.44  1.00  0.89  0.02 -0.05 -0.44  0.76  0.23
[4,] -0.60 -0.60  0.89  1.00  0.05 -0.20 -0.60  0.87  0.25
[5,] -0.29 -0.31  0.02  0.05  1.00 -0.13 -0.28  0.06  0.00
[6,]  0.09  0.09 -0.05 -0.20 -0.13  1.00  0.10 -0.18  0.02
[7,]  0.99  0.99 -0.44 -0.60 -0.28  0.10  1.00 -0.68 -0.34
[8,] -0.67 -0.66  0.76  0.87  0.06 -0.18 -0.68  1.00  0.23
[9,] -0.33 -0.33  0.23  0.25  0.00  0.02 -0.34  0.23  1.00
```

Figure 34: Correlation matrix for feature set

- Generating Eigenvalues and Eigenvectors

```
eigen() decomposition
$values
[1] 4.7670068605 1.3529833654 1.0932245304 0.8388659543 0.6925513186 0.1718398387 0.0695540524
[8] 0.0130114517 0.0009626281
```

Figure 35: Eigenvalues of correlation matrix

```
$vectors
             [,1]         [,2]        [,3]        [,4]        [,5]        [,6]         [,7]         [,8]
 [1,] -0.41949755 -0.278141238  0.09720310  0.02651993  0.24161405  0.13008565  0.0413487086  0.344534537
 [2,] -0.41911712 -0.285124969  0.08981442  0.03295185  0.22832967  0.14201801  0.0612779453  0.450339188
 [3,]  0.33245720 -0.477348302  0.04415705 -0.17954851  0.33415118 -0.46993254 -0.5407363647  0.049402555
 [4,]  0.38815198 -0.383090382  0.12696607 -0.07381009  0.13962867 -0.12437904  0.8031512679 -0.030562553
 [5,]  0.11439892  0.563163467  0.32414122 -0.23413989  0.70686904  0.09473932  0.0327299263  0.007242526
 [6,] -0.07829564 -0.001772212 -0.77950577 -0.58427325  0.16002746  0.10159124  0.0944747707  0.001501826
 [7,] -0.42008982 -0.269944046  0.08922036  0.01494269  0.24522836  0.07007507 -0.0008788577 -0.820681334
 [8,]  0.39784352 -0.276380071  0.10489283 -0.08872924 -0.02313966  0.83508294 -0.2169721428 -0.038469596
 [9,]  0.18221049  0.017595417 -0.48244704  0.74580989  0.41519876  0.07093326  0.0036206195 -0.007963368
              [,9]
 [1,]  0.735387052
 [2,] -0.674557793
 [3,] -0.009657680
 [4,]  0.007690888
 [5,] -0.010377491
 [6,]  0.002207306
 [7,] -0.062490867
 [8,]  0.001853515
 [9,]  0.002034414
```

Figure 36: Eigenvectors of correlation matrix

- Compute new data set

All above steps were performed in few lines of code in R Studio. Hence PCA transformation yielded below specified new dataset.

Table 7: PC1 and PC2 data after PCA

	PC1	PC2	V3
1	-1.43253008	-0.63198232	1
2	-1.54904317	0.12266446	1
3	-1.44227455	-0.17911189	1
4	-1.49674860	0.35936540	1
5	-1.37435306	-0.49889213	1
6	-1.53585177	-0.51116304	1
7	-1.44411273	-0.18765079	1
8	-1.37755229	-0.48452476	1
9	-1.39506475	-0.83372530	1
10	-1.48510302	-0.22356340	1
11	-1.82371429	1.30933013	1
12	-1.93576169	1.86172176	1
13	-1.84531864	1.44988629	1
14	-1.49410043	0.01559626	1
15	-1.50908387	-0.13398692	1
16	-1.84396508	1.75326920	1

4.5 Classification

A linear kernel based Support Vector Machine (SVM) classifier was used to classify each quaternion on every different on body position. For every instance the training set and test set were plotted and eventually a matrix comprising of evaluation parameters was developed to evaluate the model.

4.5.1 Training Set Vs Test Set Plot

Datasets classified using SVM, were evaluated using Ten-fold Cross Validation and they were plotted for visualization and analysis. The fold providing best accuracy was plotted for visualization. In every instance the Training Set and its corresponding Test Set was plotted.

Calf

Quaternion 1

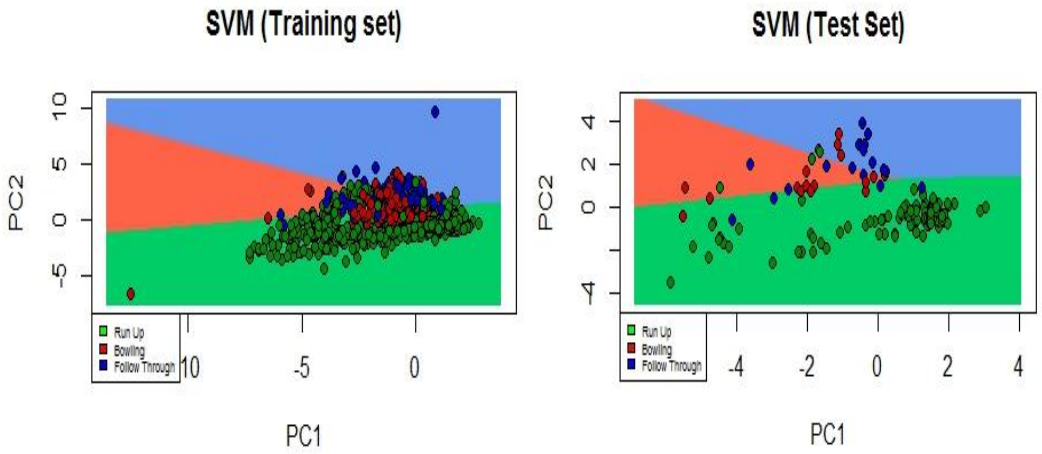


Figure 37: Training set Vs Test set SVM classification data plot for q_1 on Calf

Quaternion 2

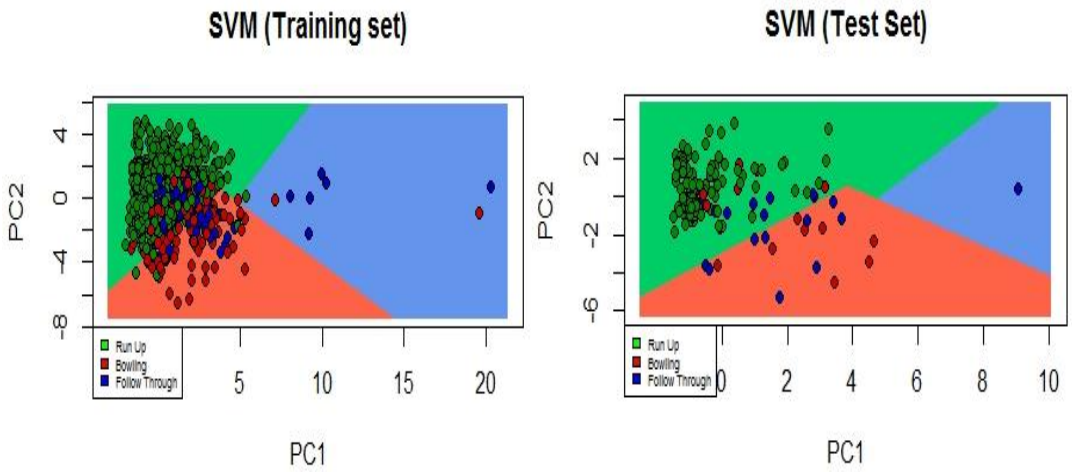


Figure 38: Training set Vs Test set SVM classification data plot for q_2 on Calf

Quaternion 4

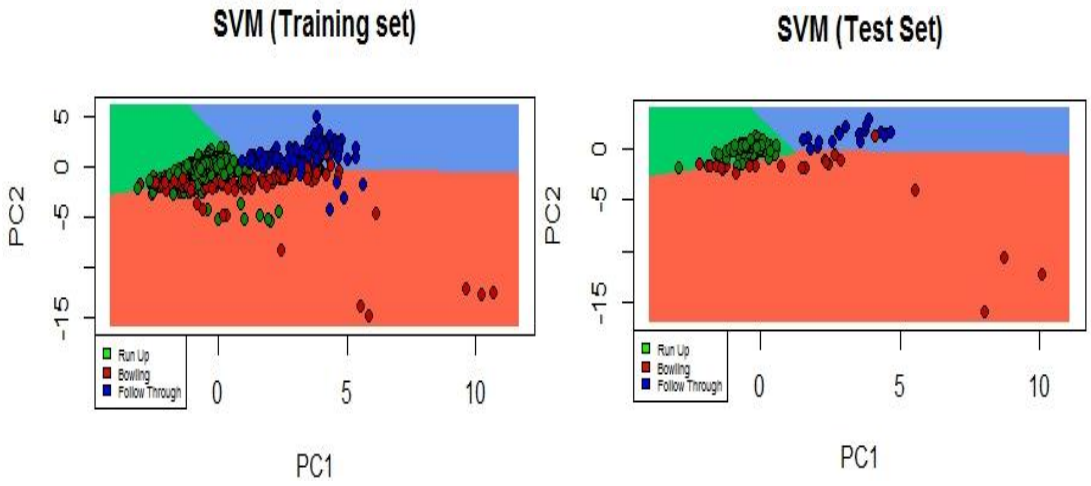


Figure 39: Training set Vs Test set SVM classification data plot for q_4 on Calf

The regions for all three classes have been defined in all the quaternions. However, in the training sets all data points depict to be clustered together. In the test sets, majority of Run Up data were correctly classified, however Delivery Stride and Follow Through data showed incorrect classifications. q_4 showed the best classification results from the plots and q_3 provided correct decision regions only for Run Up data, hence q_3 plot wasn't included.

Forearm

Quaternion 1

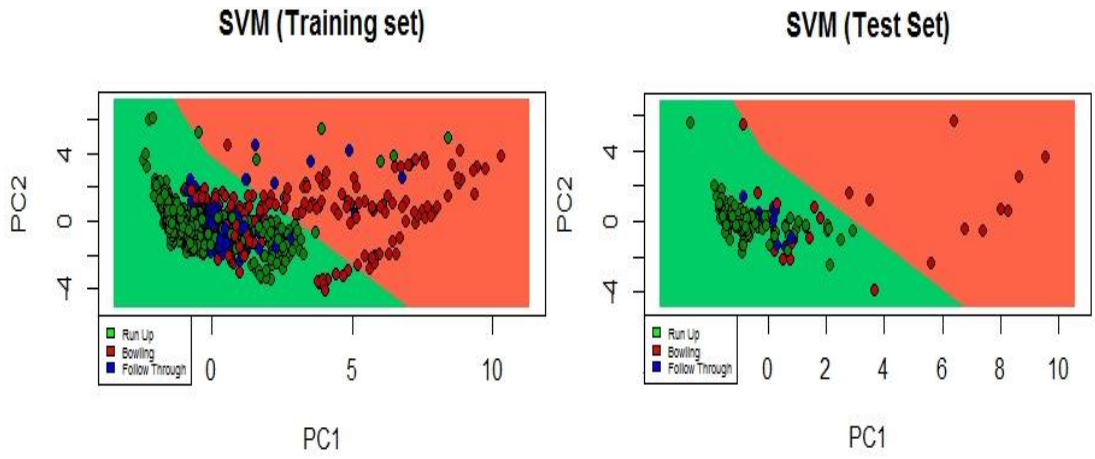


Figure 40: Training set Vs Test set SVM classification data plot for q_1 on Forearm

Quaternion 2

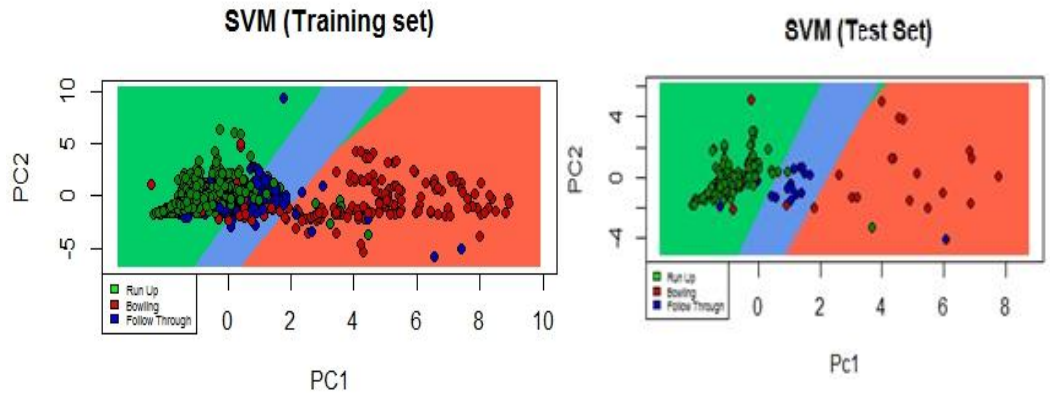


Figure 41: Training set Vs Test set SVM classification data plot for q_2 on Forearm

Quaternion 3

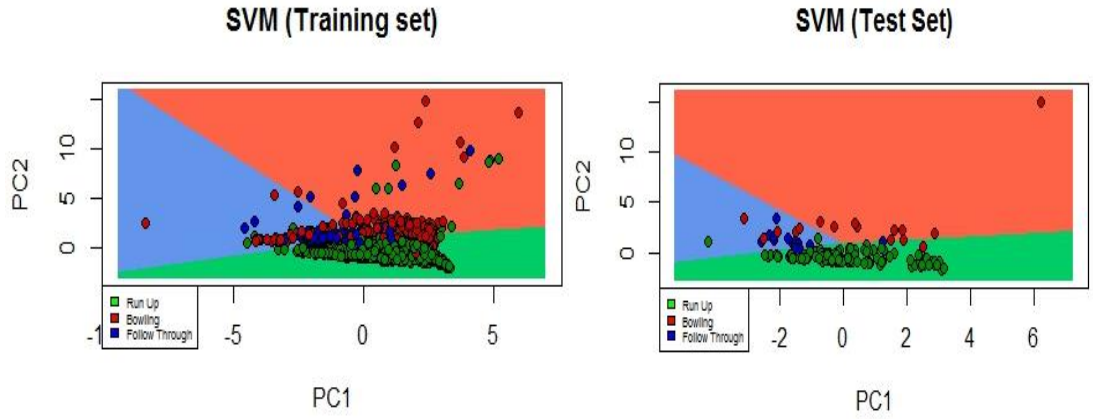


Figure 42: Training set Vs Test set SVM classification data plot for q_3 on Forearm

Quaternion 4

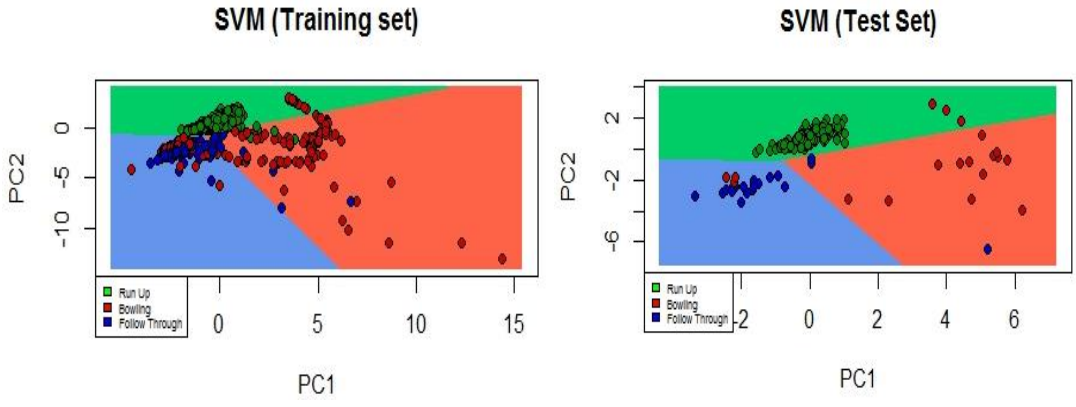


Figure 43: Training set Vs Test set SVM classification data plot for q_4 on Forearm

Data received from the forearm, once plotted demonstrated improved results in comparison to the results obtained from Calf. However, in Quaternion 1 Run Up and Follow Through decision regions demonstrated to be overlapped. Hence, a clear decision region was not defined for Follow Through. But all other Quaternion plots show improved results.

Thigh

Quaternion 4

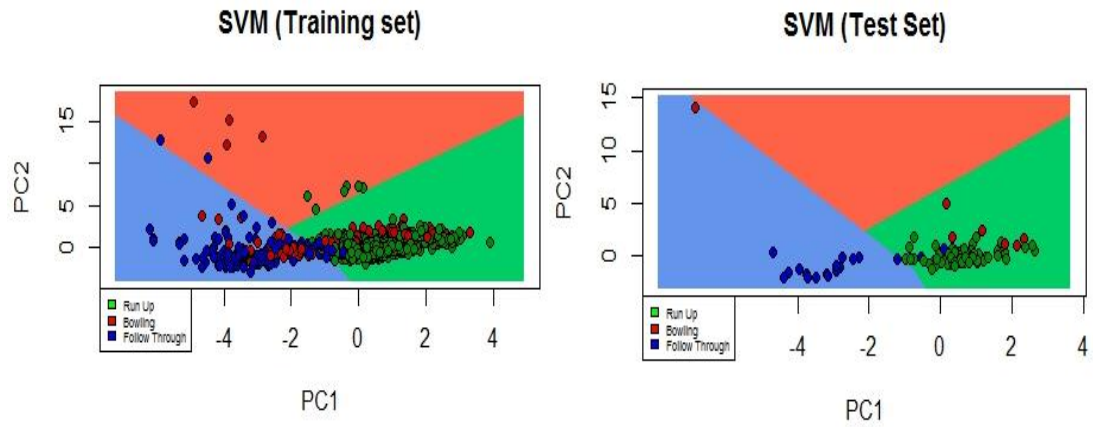


Figure 44: Training set Vs Test set SVM classification data plot for q_4 on Thigh

Among all quaternions for data received from the Thigh only fourth quaternion showed correctly defined decision boundaries. However, the fourth quaternion plot from Thigh showed less performance in comparison to previous plots.

Trunk

Quaternion 1

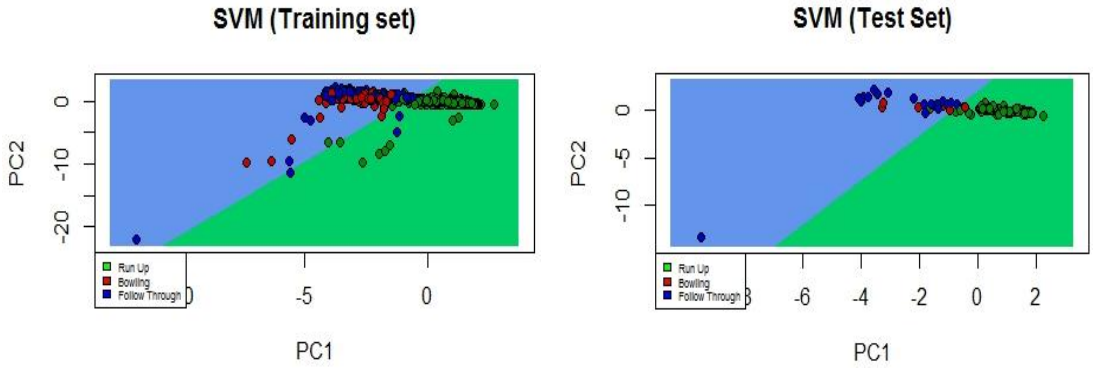


Figure 45: Training set Vs Test set SVM classification data plot for q_1 on Trunk

Quaternion 2

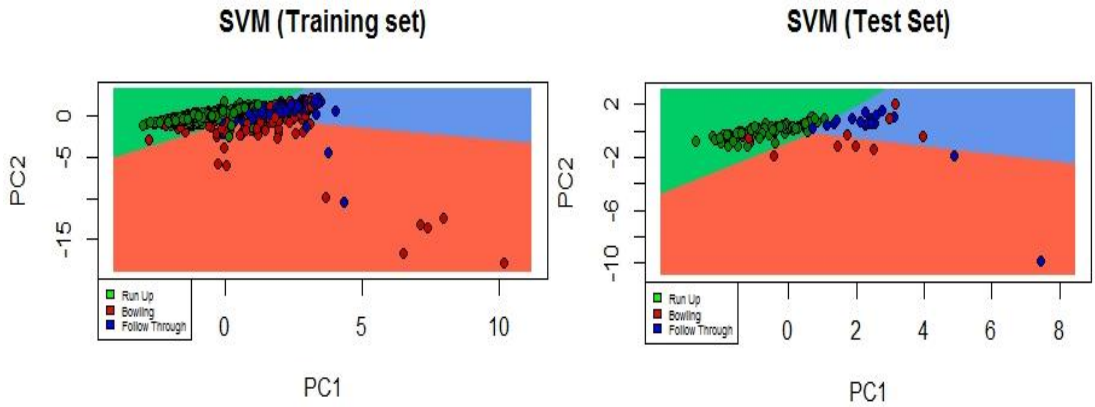


Figure 46: Training set Vs Test set SVM classification data plot for q_2 on Trunk

Quaternion 3

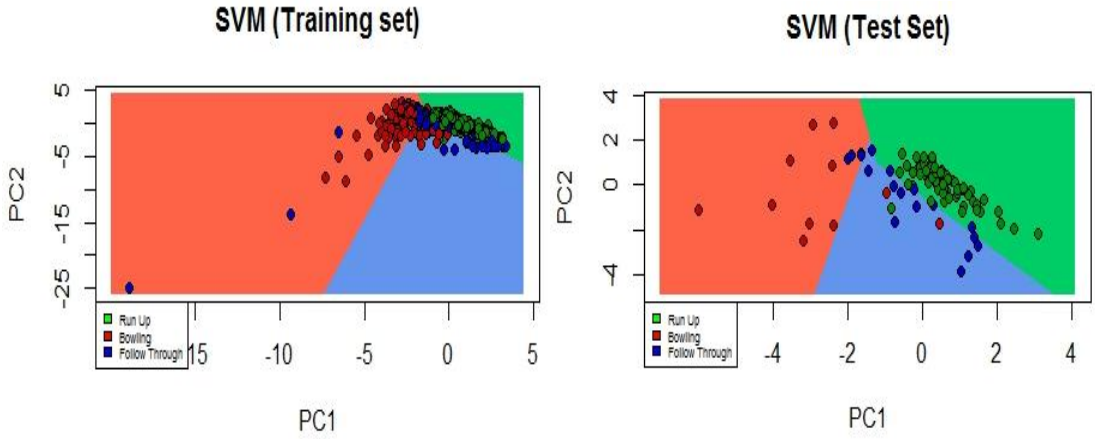


Figure 47: Training set Vs Test set SVM classification data plot for q_3 on Trunk

Quaternion 4

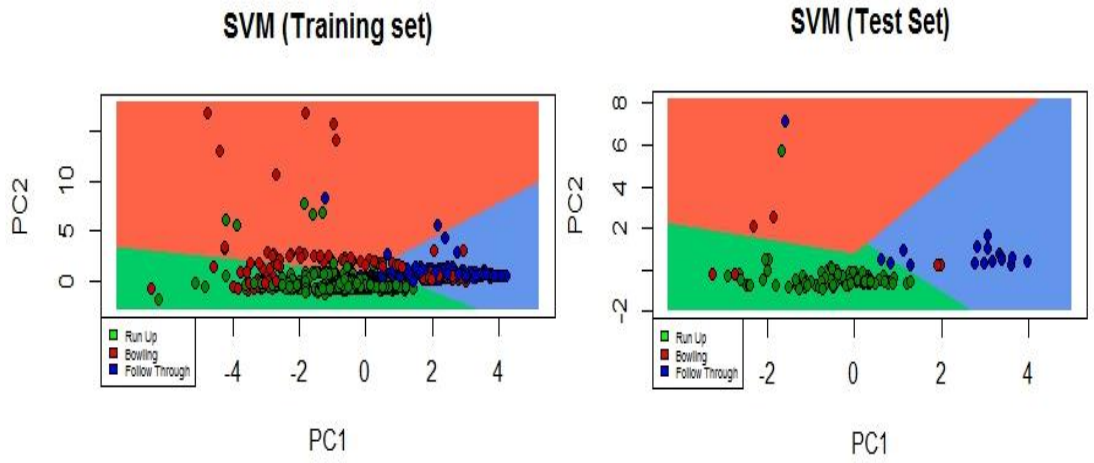


Figure 48: Training set Vs Test set SVM classification data plot for q_4 on Trunk

Graphs plotted for all instances of data for datasets from Trunk showed similar illustrations to plots from Forearm. In first quaternion Delivery Stride (Bowling) and Follow Through decision regions demonstrated to be overlapped. Hence, Delivery Stride decision region was not clearly depicted in the plot. But all other quaternion plots illustrated clear classification results, with third quaternion illustrating best visual results.

In all plots second and fourth quaternion demonstrated best defined decision regions and corresponding classification points with data from Forearm and Trunk showing overall best classification visualization results.

4.6 Classification Evaluation

Table 8: Performance parameters of classification

Parameter	**Calf**				**Forearm**			
	SVM				**SVM**			
	Q1 (%)	**Q2 (%)**	**Q3 (%)**	**Q4 (%)**	**Q1 (%)**	**Q2 (%)**	**Q3 (%)**	**Q4 (%)**
Accuracy	83	81	78	92	82	89	89	90
Precision	94	99	99	97	99	97	99	99
Recall	92	82	78	95	82	93	95	98
F-Measure	93	90	87	96	90	95	97	99
Parameter	**Trunk**				**Thigh**			
	SVM				**SVM**			
	Q1 (%)	**Q2 (%)**	**Q3 (%)**	**Q4 (%)**	**Q1 (%)**	**Q2 (%)**	**Q3 (%)**	**Q4 (%)**
Accuracy	86	89	86	87	74	74	75	86
Precision	98	99	96	99	99	99	99	99
Recall	97	94	93	92	74	74	75	89
F-Measure	98	97	94	95	85	85	85	94

The above tables demonstrate the performance of each classification. As discussed previously Accuracy, Precision, Recall and F-measure were considered as classifier performance evaluation parameters. It was observed that all quaternions, on every considered body positions demonstrated accuracy levels above 74%. The best accuracy percentage of 92% was achieved for fourth quaternion dataset classification on Calf. It was also evident that fourth quaternion for each position yielded best accuracy of classification except for data collected on Trunk.

4.7 Discussion

It was observed that classification conducted by data received from Forearm yielded the best overall Accuracy percentage. Best Precision was obtained for data classified by sensors on Thigh followed by Forearm. Best percentages for Recall and F-Measure were obtained by data classified from sensors on Trunk and Forearm.

Due to high Accuracy percentage (90%) and high-performance values scored for Precision, Recall and F-measure sensor data received from Forearm was considered best for classification. Therefore, Forearm was selected as the best position to collect data for classification of different phases of fast bowling action. Further, sensors placed on Forearm provided least disturbances during data collection and developing a mechanism to hold sensors on Forearm was easier than placing sensors on other locations on the body.

Among the four quaternions of data on Forearm the fourth quaternion provided the best results for all performance parameters of evaluation (90%). This feature was observed for evaluation parameters on Thigh (86%), Trunk (87%) and Calf (92%) except for the Accuracy parameter on Trunk. This decision also emphasises the previous observations on Training Set Vs Test Set plots and Mean vs Variance feature plots. The second quaternion also yielded good results (89% Accuracy on Forearm). Hence second and fourth quaternion data obtained by sensors on Forearm was selected as the quaternion number and body position for next data collection.

It should also be noted that Run Up class had greater weight (more data points) in comparison to other two classes. Hence the classification performance parameters tend to be biased towards the performance generated from Run Up class. As a result, high performance parameter percentages were observed in the results.

Above results, when compared with those obtained for human activity classification by sensors on different locations [26] indicate that hand movements are best classified by sensors on Wrist. Results from current research indicate similar results where Delivery Stride, which includes considerable amount of hand movements is best classified when sensor is placed on the Forearm. However, as in former study the effect of using combination of sensors is not analysed in current research.

CHAPTER 5

5 ACTIVITY CLASSIFICATION DURING FAST BOWLING IN CRICKET

From the previous data gathering and analysis it was concluded that best on body position for activity classification during fast bowling was the Forearm. Further, second and fourth quaternion values yielded best accuracy for classification. Hence the second set of data collection was conducted with an IMU sensor placed on the forearm with second quaternion acting as the measurement to be analysed.

5.1 Data Collection Methodology

The same 9 axis inertial measurement unit which was used in the previous step was used to gather relevant data. A special strap on unit was designed using leather and Velcro straps to hold the sensor during bowling.

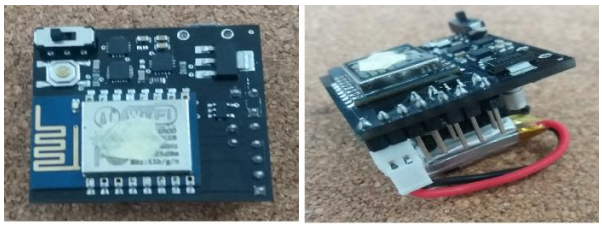


Figure 49: MPU9250 integrated ESP 8266 Wi-Fi module

Figure 50: Wearable strap on forearm

5.1.1 Battery Selection for Sensor

The IMU sensor (MPU9250) was incorporated with an ESP-8266MOD module for wireless data transmission. As a result, the sensor was relatively small. It was observed that the setup consumed a maximum of 80mA to 100mA during data

transmission through Wi-Fi. Hence a Lithium Polymer battery of 180mAh was selected for the operation.

5.1.2 Wireless Data Transmission

With the use of an ESP-8266 module it was possible to transmit data wirelessly during motion. A Python socket programming script was written to collect the data transmitted through Wi-Fi.

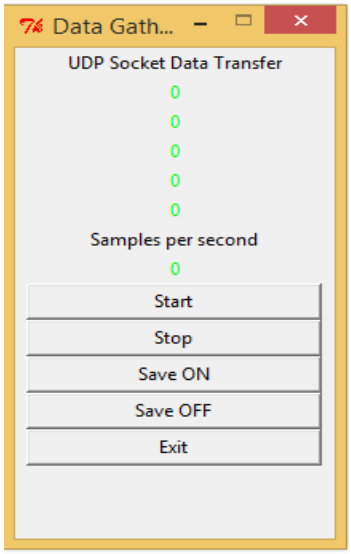


Figure 51: UDP data collection interface

At the first step four quaternion values and corresponding timestamps were sent using Transmission Control Protocol (TCP). However, it was not possible to reach sampling rates beyond 50 Hz using TCP. Fast sampling rates of more than 300Hz was achieved when data were buffered and transmitted. However, there was a data loss (100ms) during data transmission of the buffered data packets.

This issue in data rate loss was eradicated using User Diagram Protocol (UDP). With UDP, sampling rates beyond 300Hz was achieved during data transmission. However, there was a general tendency to lose certain data packets when transmitting data using UDP (Please refer to section 6.2.4). Data transmission began once a character sent by Python script was received by ESP module. The received data were stored in .csv file for processing.

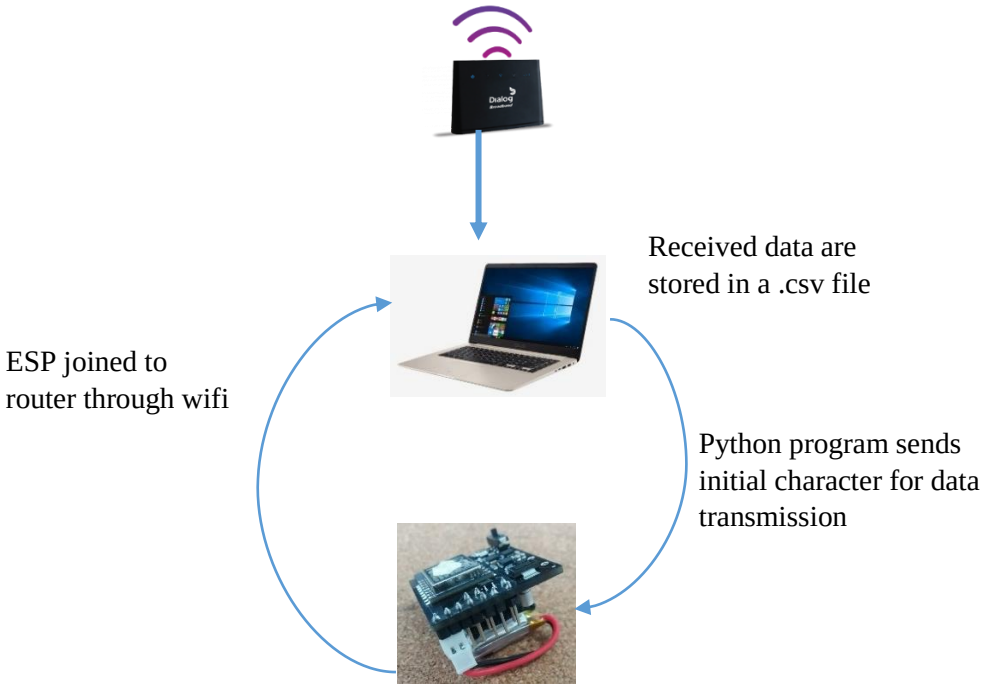


Figure 52: Data transmission connectivity

5.1.3 Definition of Classes for Classification

Definition of classes for classification was one of the key requirements for a supervised classification task. Defining these classes correctly was one of the key challenges for this study. Three key phases in fast bowling were defined as the three main classes for this activity classification.

- Class 1 – Run up
- Class 2 – Delivery Stride (Bowling)
- Class 3 – Follow Through

Video feedback received from a camera synchronized with sensor data was used as the method for definition of classes. Canon EOS 1300D DSLR camera was used to obtain video feedback to define the classes. Movie size was set to 1280 x 720 to obtain a frame rate of 50 frames per second. All participants were advised to perform each delivery in below routine.

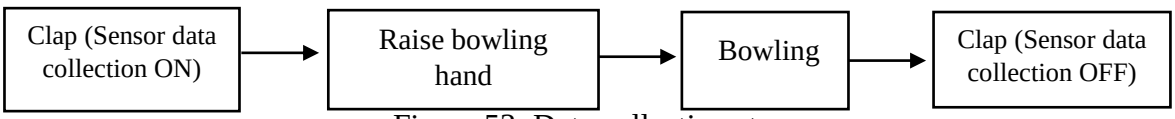


Figure 53: Data collection steps

Sensor initiates data collection at the first clap and ends data at final clap. The audio peaks generated during the claps were used to sync video with sensor data. Raising of hand generated a secondary point to determine accuracy of data syncing. A factor to be noted was the difference in sensor and video sampling rates.

Once both sensor and video were synced the three classes were segmented using video feedback for beginning and ending of each class. Relevant data samples were divided into each class accordingly.

Class 1 – Run Up

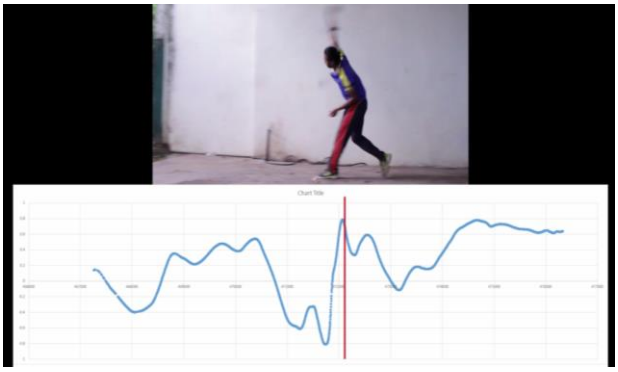
Run up beginning position

Run up end position

Figure 54: Run Up class visualization for subject 4

Class 2 – Delivery Stride

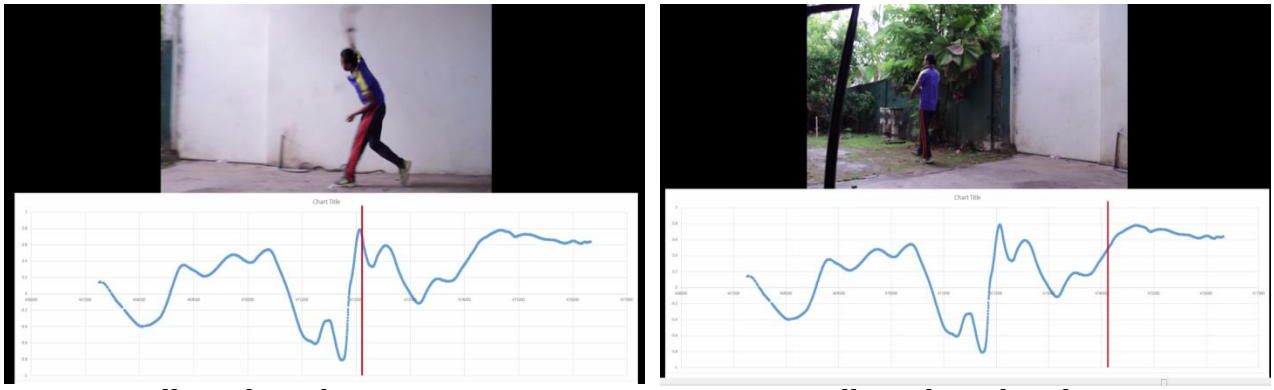
Delivery Stride start position



Delivery Stride end position

Figure 55: Delivery Stride class visualization for subject 4

Class 3 – Follow Through



Follow Through start position Follow Through end position

Figure 56: Follow Through class visualization for subject 4

5.1.4 Data Gathering Participants

Four participants were selected for the data gathering. The consent of each participant was obtained to video their relevant bowling actions. Below table depict the details of relevant samples.

Table 9: Data gathering sample set

Bowler Number	Age	Height (cm)	Weight (Kg)	Number of samples
1	28	164	63	8
2	28	173	83	8
3	28	175	65	8
4	36	173	70	8

5.2 Classification methods

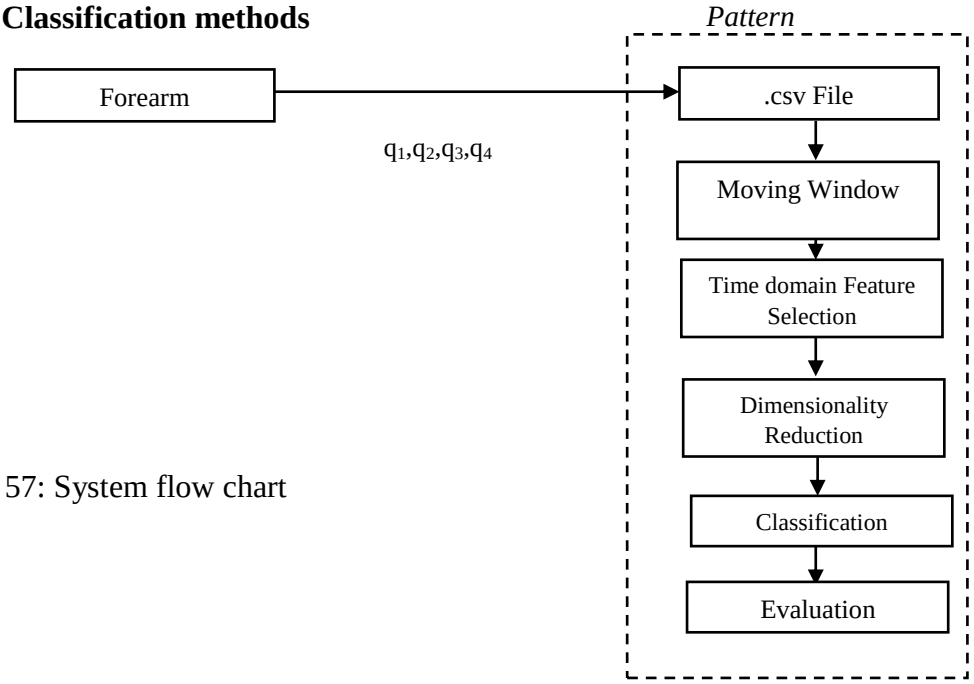


Figure 57: System flow chart

5.2.1 Original Data Plots

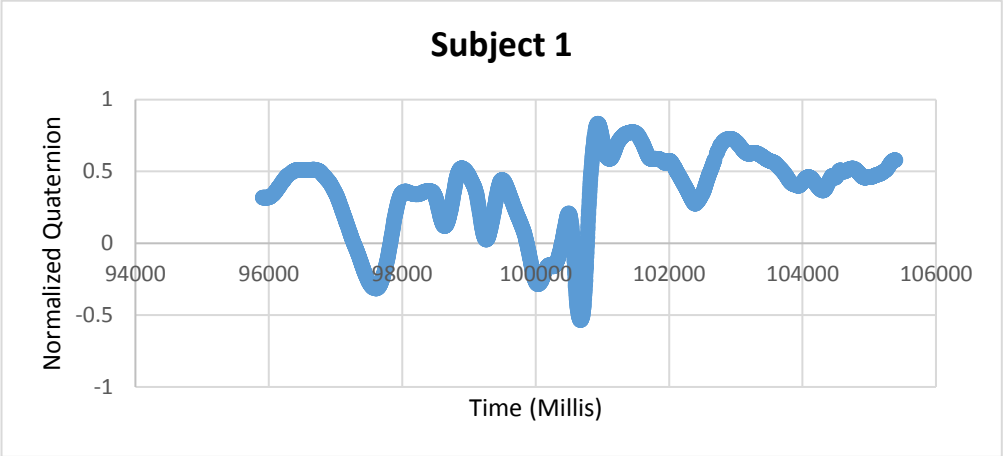


Figure 58: Subject 1 original data plot – Quaternion 2

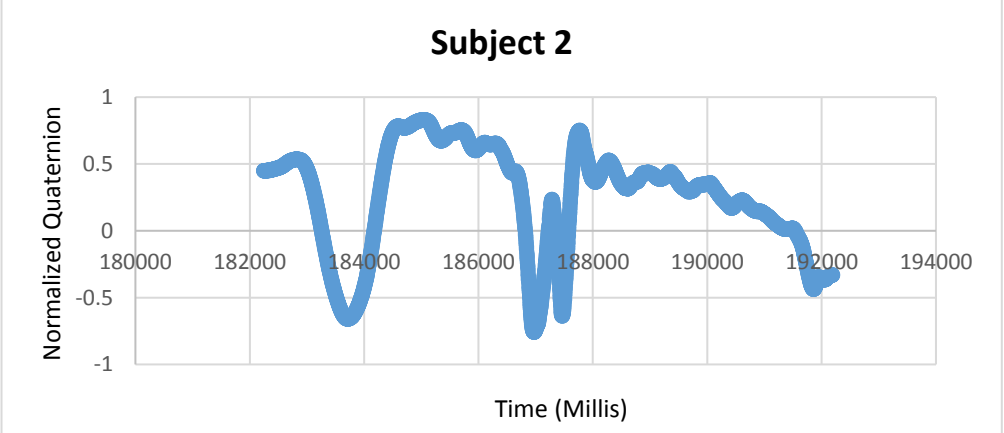


Figure 59: Subject 2 original data plot – Quaternion 2

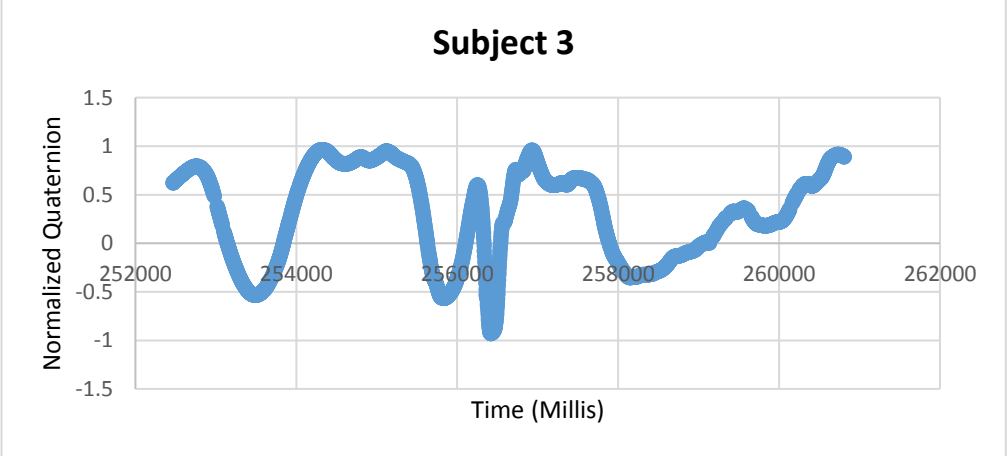


Figure 60: Subject 3 original data plot – Quaternion 2

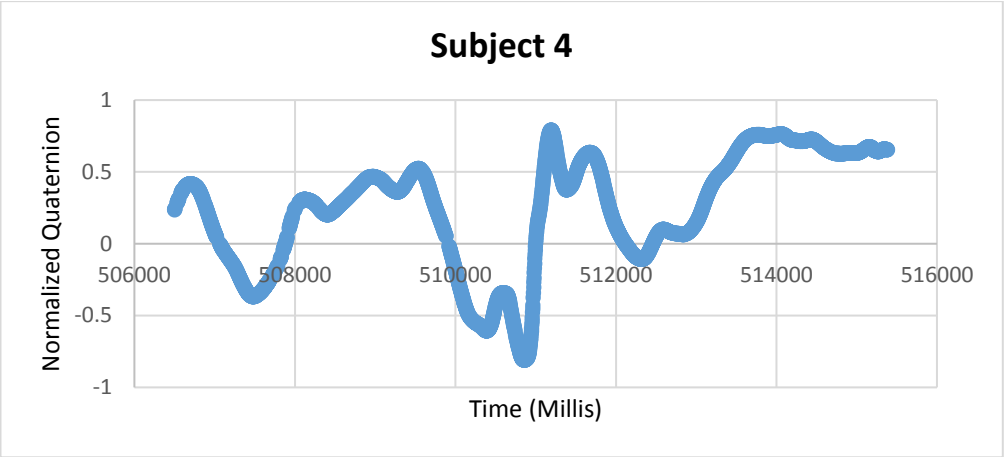


Figure 61: Subject 4 original data plot – Quaternion 2

5.2.2 Data Storage

As in the previous scenario data received from the sensor during motion were stored in a .csv file for processing.

	A	B	C	D	E	F	G
1	Time	Millis	Quaternion 1	Quaternion 2	Quaternion 3	Quaternion 4	
2	12:43.5	92079	-0.7506	0.4157	-0.1976	-0.4741	
3	12:43.5	92082	-0.7508	0.4158	-0.1974	-0.4737	
4	12:43.5	92085	-0.751	0.416	-0.1973	-0.4734	
5	12:43.5	92088	-0.751	0.4163	-0.1973	-0.473	
6	12:43.5	92091	-0.7511	0.4165	-0.1973	-0.4727	
7	12:43.5	92094	-0.751	0.4168	-0.1975	-0.4725	
8	12:43.5	92096	-0.751	0.4171	-0.1976	-0.4723	
9	12:43.5	92098	-0.7509	0.4174	-0.1978	-0.472	
10	12:43.5	92101	-0.7508	0.4177	-0.1981	-0.4717	
11	12:43.5	92104	-0.7507	0.4181	-0.1985	-0.4715	
12	12:43.5	92106	-0.7505	0.4184	-0.1988	-0.4713	
13	12:43.5	92108	-0.7504	0.4188	-0.1992	-0.4711	
14	12:43.5	92111	-0.7501	0.4192	-0.1997	-0.4708	
15	12:43.5	92114	-0.7499	0.4196	-0.2001	-0.4706	
16	12:43.5	92116	-0.7498	0.42	-0.2005	-0.4704	
17	12:43.5	92118	-0.7495	0.4204	-0.2011	-0.4702	
18	12:43.5	92121	-0.7493	0.4208	-0.2017	-0.4699	
19	12:43.5	92124	-0.749	0.4213	-0.2023	-0.4696	
20	12:43.5	92127	-0.7488	0.4217	-0.2029	-0.4693	
21	12:43.5	92129	-0.7486	0.4221	-0.2035	-0.4691	
22	12:43.5	92132	-0.7483	0.4226	-0.2041	-0.4688	
23	12:43.5	92135	-0.7482	0.423	-0.2047	-0.4685	

100-0051

Ready

Figure 62: Data stored .csv file

5.2.3 Feature Selection

As in the previous scenario, a moving window was used to obtain relevant time domain statistical features. A moving window of 400 samples per window with 50% overlap was selected. This sample range was selected to overcome similarities created between classes due to small window sizes.

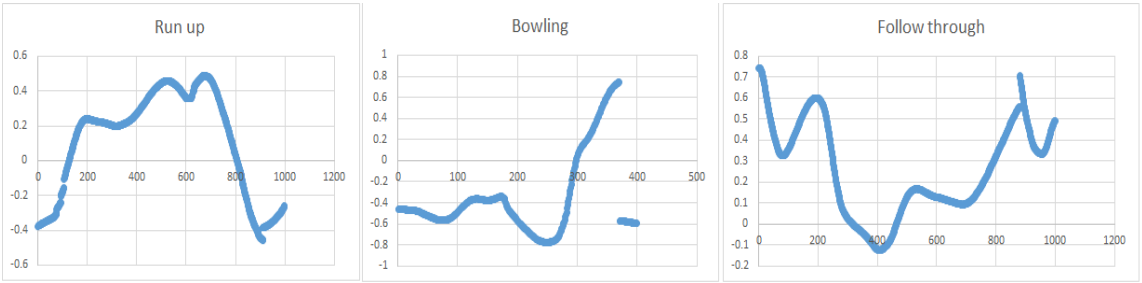


Figure 63: Run Up, Delivery Stride and Follow Through windows – Subject 2

Figure 64: Moving window with 50% overlap

Eight-time domain features were selected as relevant features for the analysis.

- Mean
- Median
- Variance
- Skewness
- Kurtosis
- Root Mean Square (RMS)
- Inter-Quartile Range (IQR)
- Median Absolute Deviation (MAD)

Table 10: Feature Set

Filter

	Mean	Median	Variance	IQR	Skewness	Kurtois	RMS	MAD	Class
1	-0.87358645	-0.27184265	-0.276005492	0.34819077	-0.759324382	-0.45837995	-1.41825610	-0.84152640	1
2	-0.01681349	-0.16530855	-0.799896552	-0.81466129	1.319212108	-0.08962895	-1.08547398	-0.91896267	1
3	0.34575413	0.33937293	-0.809722300	-0.86028328	-0.787020368	-0.10516393	-0.46556630	-0.75649833	1
4	-0.10194357	0.23767463	-0.187832304	-0.04519293	-1.254355296	0.16547364	-0.67835105	-0.44827161	1
5	-1.34314911	-1.42290916	-0.073049961	0.24685567	0.955919435	-0.21268313	-1.07861226	0.17045938	1
6	-1.05732844	-0.89478138	0.050883038	0.83129207	0.037725501	-0.68559467	-1.07389846	1.15055962	1
7	-0.06080954	-0.05716252	-0.801977581	-0.89725575	-1.088646801	1.05796724	-1.16258844	-0.76940438	1
8	0.26582997	0.28222259	-0.785343071	-0.60724233	-0.319324791	-0.52299626	-0.58669120	-0.55000161	1
9	0.60131709	0.50291083	-0.854996980	-1.00359401	0.201871192	-0.30165275	-0.04445854	-0.86202423	1
10	-0.16243851	0.27870564	0.143389165	0.37354574	-0.973019900	-0.08435300	-0.49857832	-0.04438832	1
11	-1.34388585	-1.25263045	0.089368416	0.45732740	0.179754700	-0.47036844	-0.92785516	0.84157372	1
12	-0.90826372	-0.24385364	0.197127092	0.65457044	-0.857736227	-0.32766604	-0.94044414	-0.43384721	1
13	0.14316365	0.04131191	-0.794811133	-0.68356165	0.480307384	-0.45096981	-0.80618137	-0.57733206	1
14	0.43477962	0.36076767	-0.843960595	-0.93100585	-0.129698586	-0.08856758	-0.33281761	-0.79673483	1
15	0.06597369	0.26668942	-0.376593702	-0.45256851	-1.528218839	0.60698400	-0.59531345	-0.49078564	1
16	-1.18040460	-1.17437844	0.258265229	0.63150505	0.508029480	-0.37744312	-0.85609805	0.74667633	1

wing 1 to 13 of 385 entries

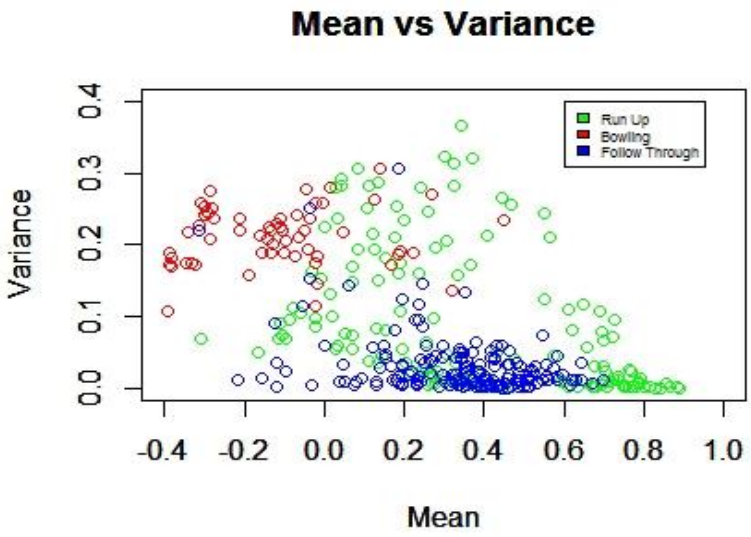


Figure 65: Mean Vs Variance feature plot

5.2.4 Feature Scaling

It was observed that selected features were out of scale. This situation had the risk of certain features depicting to contribute more towards the classifiaction model than their actual relevance. Hence, a feature scaling step was conducted prior to dimentionality reduction.

5.2.5 Feature/Dimensionality Reduction

5.2.5.1 Least Absolute Shrinkage and Selection Operator (LASSO)

In machine learning LASSO [25] is used mainly as a regularization and feature selection tool. Hence, in this research LASSO was used to reduce the feature set before classification. It was used to reduce the number of features to decrease operational complexity and eradicate overfitting of the classifier. LASSO would eradicate features whose coefficients become zero when the optimization problem is minimized.

- As an initial step the behaviour of each feature on the model was plotted. This would illustrate the significance of each feature to the model based on coefficient values.

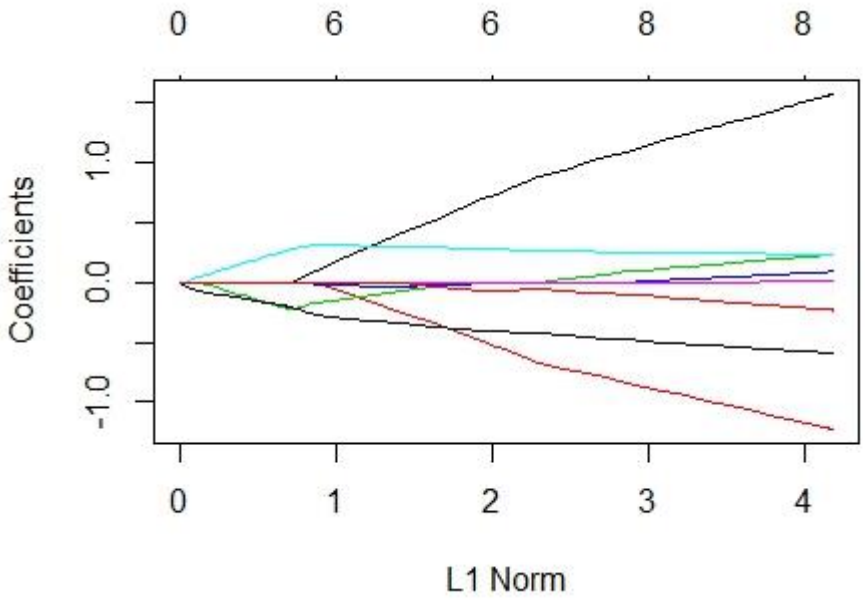


Figure 66: Illustration of coefficients of features

- The next challenge was to select relevant λ value that best selects relevant features. It was important to choose correct value since too high or low value would result in inaccuracies to the model. Cross validation was used for this purpose and corresponding features within one standard error of minimum mean cross validation error were selected to be included into the model.

- Coefficients which become zero at minimum mean cross validation error (λ_{min}) were calculated. It was observed that none of the coefficients of features reached zero. Relevant console output is illustrated below.

```
> as.matrix(coef(c, c$lambda.min))
                    1
(Intercept)  2.023809524
Mean         1.531942043
Median      -1.193537459
Variance     0.218492424
IQR          0.081845677
Skewness     0.233842144
Kurtois      0.005370676
RMS         -0.581196732
MAD         -0.220137127
```

- Finally, coefficients of features which become zero at largest value of lambda such that error is within one standard error of the minimum were calculated (λ_{1se}). It was observed that coefficients of Kurtosis and Median Absolute Deviation (MAD) reached zero. Hence, they were not included into the classifier.

```
> as.matrix(coef(c, c$lambda.1se))
                    1
(Intercept)  2.02380952
Mean         0.20757431
Median      -0.09079894
Variance    -0.13095438
IQR         -0.02874632
Skewness     0.30759359
Kurtosis     0.00000000
RMS         -0.29955691
MAD          0.00000000
```

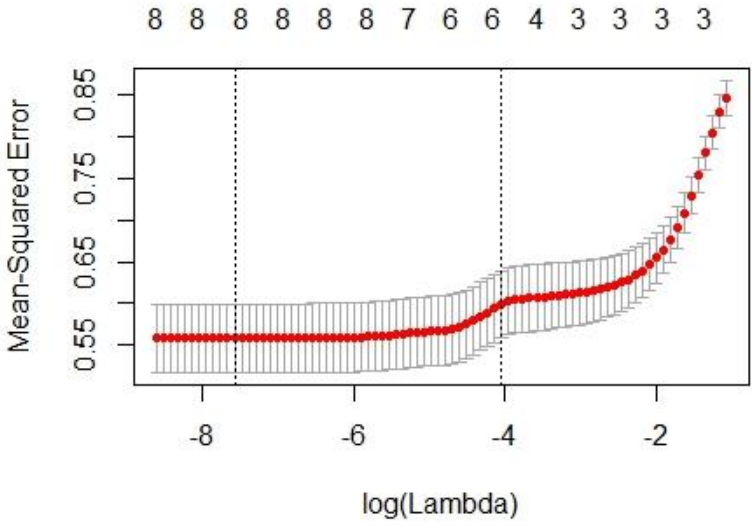


Figure 67: Lambda values at λ_{min} and λ_{1se}

5.2.5.2 Dimensionality Reduction with PCA

To reduce calculation complexity to the classification algorithm and for visualization purposes the feature set (six features) was reduced to two components using Principal Component Analysis. Principal Component 1 (PC1) and Principal Component 2 (PC2) were calculated.

Table 11: Dimensionally reduced feature data via PCA

	PC1	PC2	V3
1	-3.392712697	1.21260758	1
2	-3.482642276	1.70417703	1
3	-1.265677104	-0.44188349	3
4	-1.003056193	-1.06861158	3
5	0.278302794	-1.56386946	3
6	0.059371104	-0.72616886	3
7	-0.408095829	-0.24320428	3
8	-0.654735659	-0.16214481	3
9	1.207815733	2.12473342	2
10	-1.011952374	-0.25482455	1
11	-1.047111935	-1.16288615	3
12	1.741259943	2.15311068	1
13	-0.631505124	-1.15635192	1
14	3.203201497	0.63871162	2
15	1.982726889	1.44081651	1
16	0.602838554	-0.35606498	3
17	-2.722824233	0.64244062	1

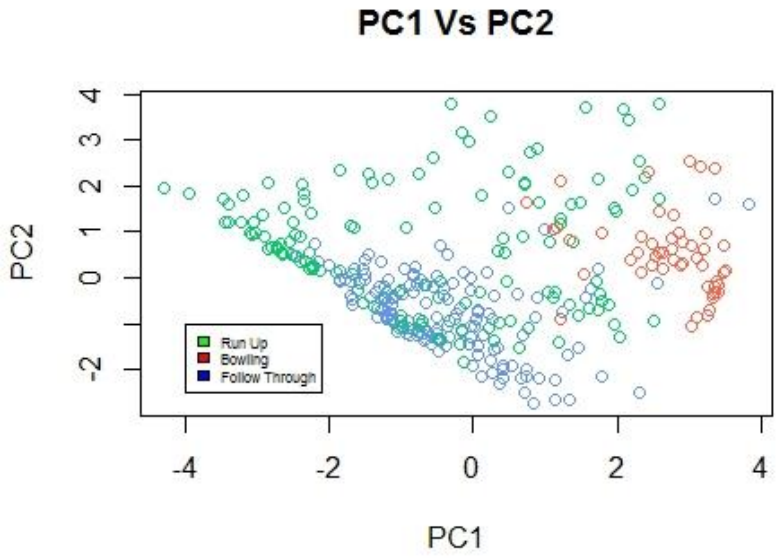


Figure 68: Principal Component 1 (PC1) Vs Principal Component 2 (PC2) plot

5.2.6 Classification

The principal classification methods used were supervised classification models. Four models were used to compare performance among them and to determine best classifier for such applications.

5.2.6.1 k- Nearest Neighbour (k-NN)

k-NN classifier was used as the first classifier. The first step was to determine best k number for classification. For evaluation purposes 5-fold cross validation was used. Hence, for every fold k number was varied from 1 to 100 and the corresponding accuracy was calculated for every k number. The k number providing best accuracy for corresponding fold was selected as relevant k number for classification.

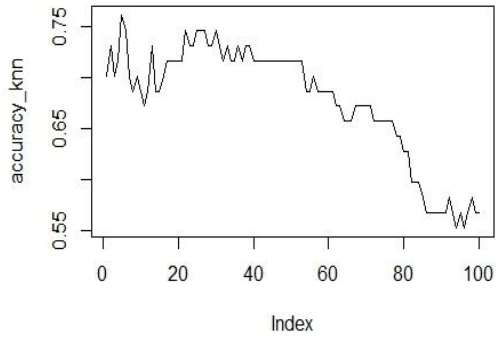


Figure 69: Accuracy Vs k number – Fold 1

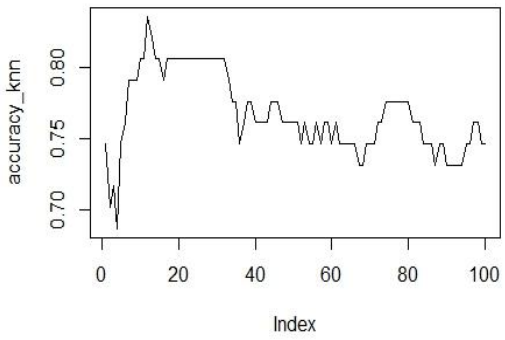


Figure 70: Accuracy Vs k number – Fold 2

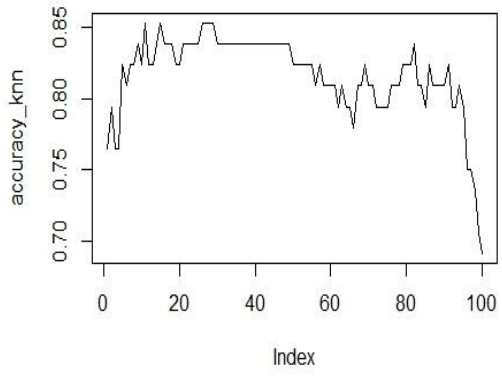


Figure 71: Accuracy Vs k number – Fold 3

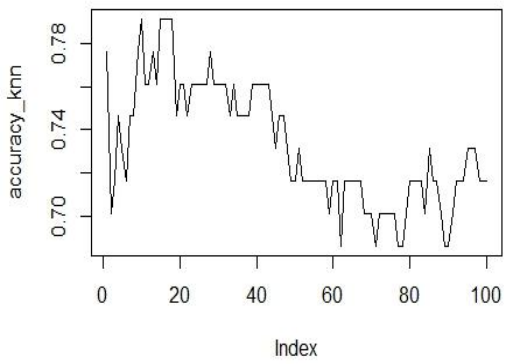
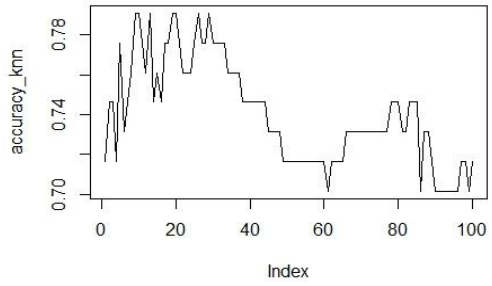


Figure 72: Accuracy Vs k number – Fold 4



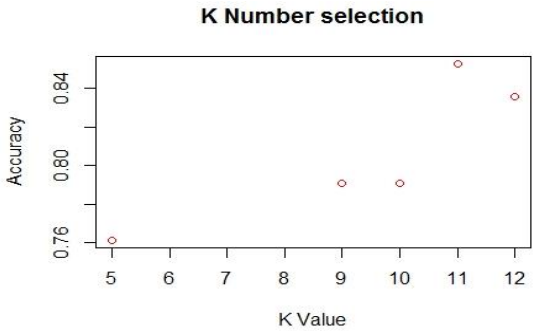


Figure 73: Accuracy Vs k number – Fold 5 Figure 74: Maximum k numbers

Hence 11 was selected as the k number for classification using k-NN classifier.

Training Set

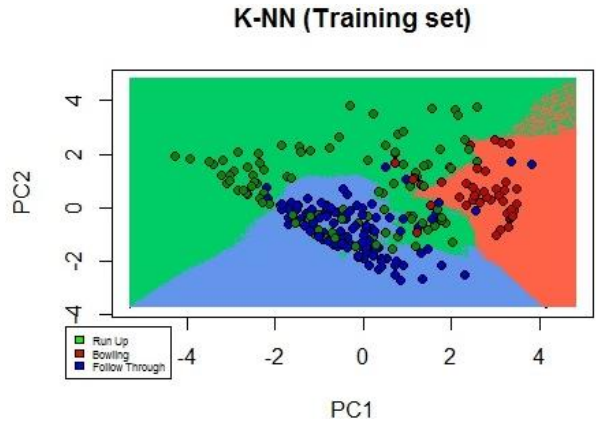


Figure 75: k-NN Training Set plot

Training model was plotted depicting results from the fold which provided best average accuracy. It was observable that Run Up and Follow Through classes show an overlap when fitting into their specific regions.

Test Set

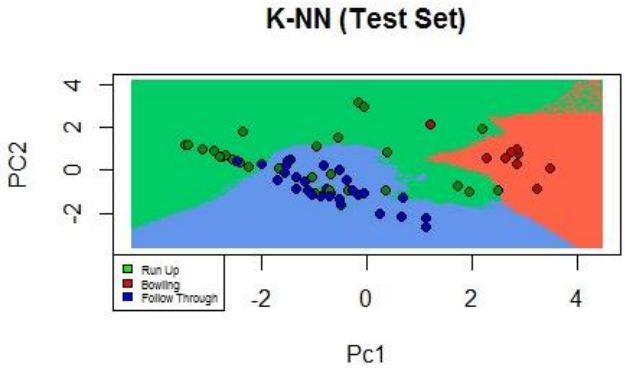


Figure 76: k-NN Test Set plot

The Test set visualization demonstrated that the major inaccuracies were caused due to incorrect classification of Run Up and Follow Through classes. The corresponding Test set of the fold that was used for the Training set was used for classification of the Test set.

5.2.6.2 Support Vector Machine (SVM)

A linear kernel based SVM was used as the second classifier for classification. 5-fold cross validation was used to create folds and corresponding Training and Test sets. Fold with best accuracy was selected for visualization and analysis.

Training Set

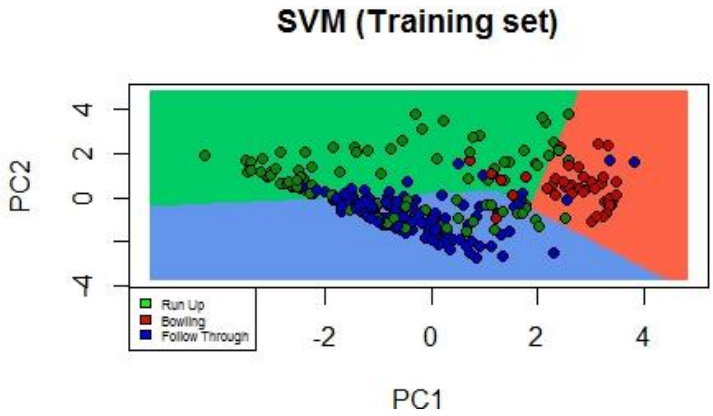


Figure 77: SVM Training Set plot

In the Training set it was evident that certain amounts of Run Up and Follow Through classes were classified incorrectly.

Test Set

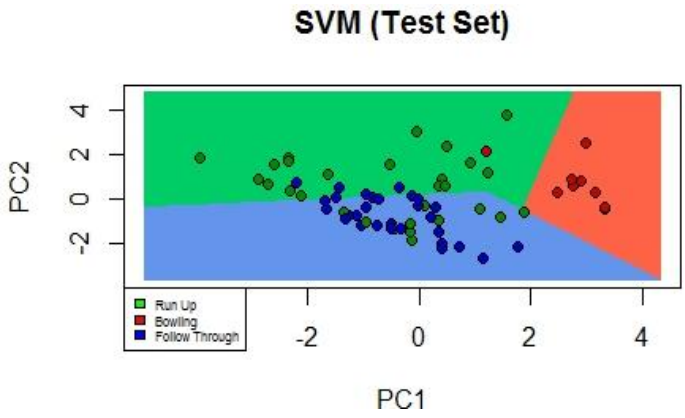


Figure 78: SVM Test Set plot

In the Test set classification, it was clear that majority of Delivery Stride class was correctly classified. However, a certain portion of Run Up and Follow Through classes were incorrectly classified.

5.2.6.3 Naïve Bayes

Naïve Bayes was used as the third supervised classification model. It has been used throughout literature for similar instances for human activity classification.

Training Set

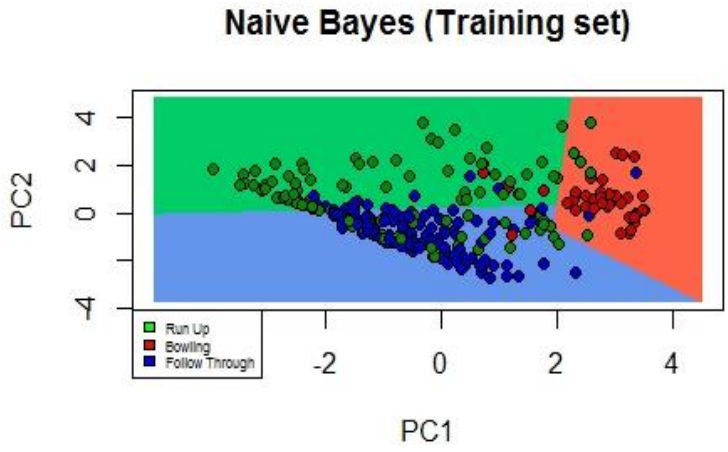


Figure 79: Naïve Bayes Training Set plot

Test Set

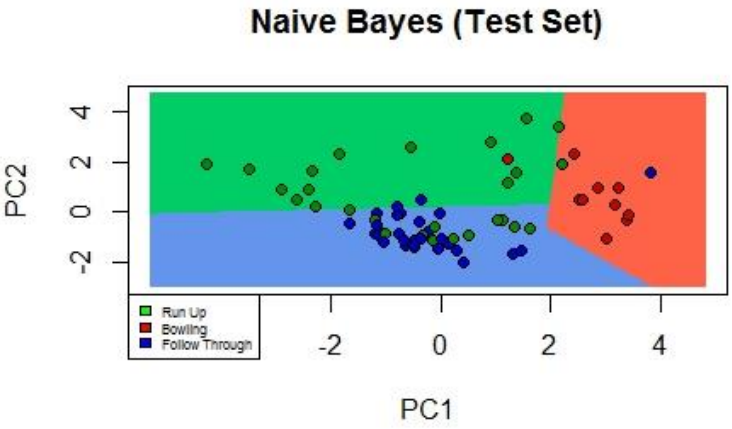


Figure 80: Naïve Bayes Test Set plot

5.2.6.4 Random Forest

The final classification model compared was Random Forest classifier. It was also used as a supervised classification model. It is based around developing multiple decision trees at training instance. The number of trees were selected at 5000 after multiple cycles with comparison against accuracy.

Training Set

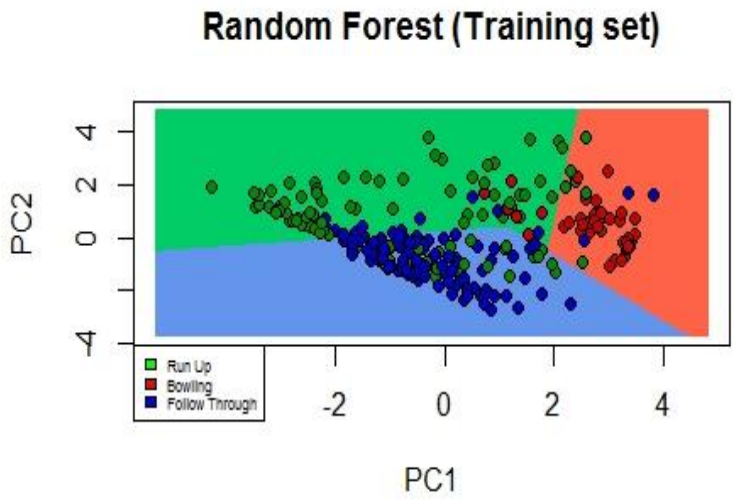


Figure 81: Random Forest Training Set plot

Test Set

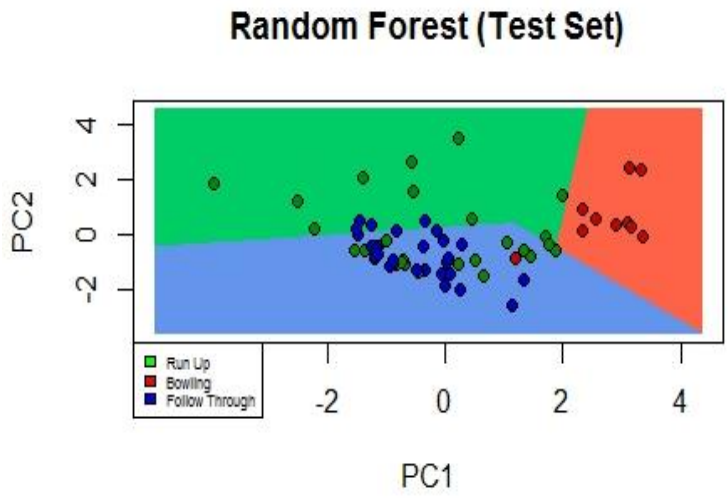


Figure 82: Random Forest Test Set plot

5.3 Classifier Evaluation

Five-Fold Cross Validation was used as the primary evaluation method for above classification. Five folds were selected due to reduced number of instances in the data set. As used throughout literature for similar research, below evaluation parameters were used for evaluation.

- Accuracy
- Precision
- Recall
- F Measure

Accuracy, Precision and Recall were calculated for each fold and the mean value across all folds was calculated as the final value. F-Measure was calculated from mean values of Precision and Recall.

Table 12: Summary of classifier performance

Parameter	SVM (%)	k-NN (%)	Naïve Bayes (%)	Random Forest (%)
Accuracy	73	77	74	75
Deviation	+/- 1	+/- 2	+/- 3	+/- 4
Precision	74	79	77	73
Recall	58	65	57	70
F - measure	65	71	66	71

5.4 Synthetic Minority Over-Sampling Technique (SMOTE)

It was observed that there was an imbalance in between the classes. Run Up and Follow Through classes exhibited similar size of samples represented in the classes. However, the sample size of Delivery Stride class was very less in comparison to other two classes. As a result, the weight of classification accuracy was governed by Run Up and Follow Through classes. This effect was overcome using SMOTE algorithm. SMOTE creates more samples around the minority class and reduces certain samples from majority class to balance out the sample distribution among the classes.

5.4.1 k-NN Classifier Comparison with SMOTE

As specified in the original SMOTE [24] documentation. SMOTE needs to be applied with a feature selection algorithm. In below instance, SMOTE was applied on the data obtained after dimensionality reduction via PCA to a k-NN classifier since it produced best results among the tested classifiers. The Delivery Stride (Bowling) class was over sampled in relation to Run Up class.

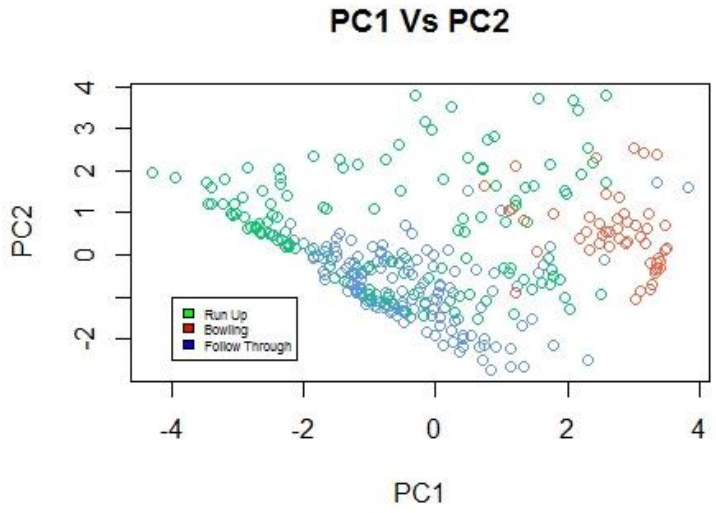


Figure 83: PC1 Vs PC2 data points plot before applying SMOTE

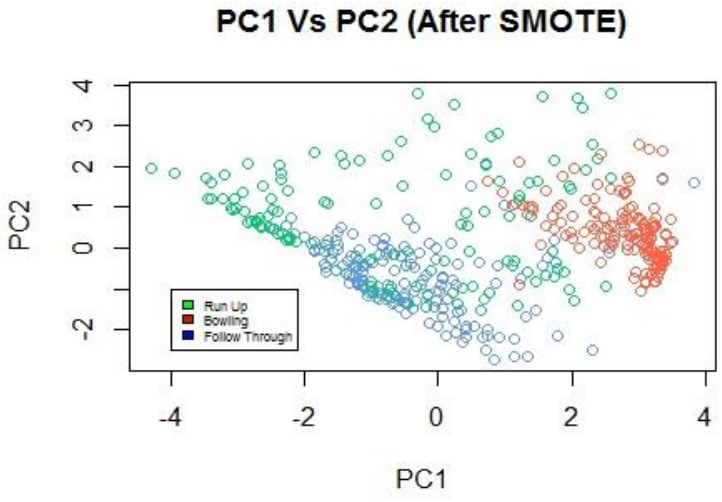


Figure 84: PC1 Vs PC2 data points plot after applying SMOTE

As illustrated in the diagrams, Delivery Stride class was populated with more data points without affecting the nature of original data points distribution.

k – Number Selection

5 – Fold Cross Validation was used to evaluate the effect of applying SMOTE prior to classification. As discussed previously, k number for k-NN algorithm was selected by varying k from 1 to 100 among each fold and by detecting fold and k number providing best accuracy.

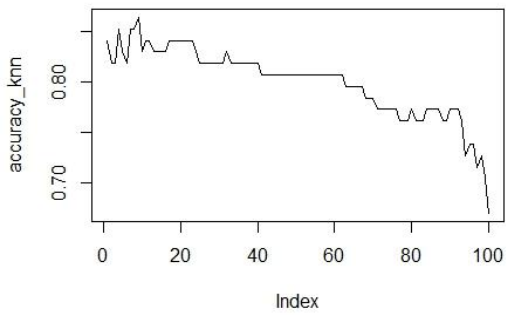


Figure 85: Accuracy Vs k number – Fold 1

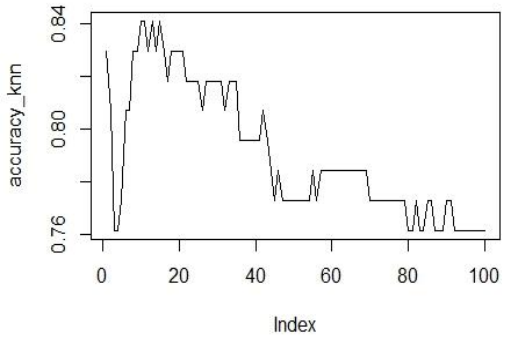


Figure 86: Accuracy Vs k number – Fold 2

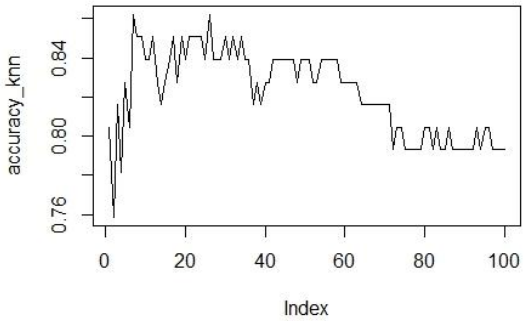


Figure 87: Accuracy Vs k number – Fold 3

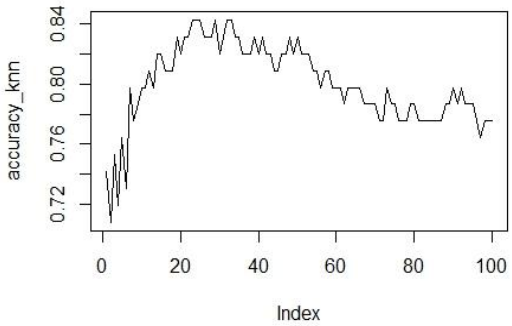


Figure 88: Accuracy Vs k number – Fold 4

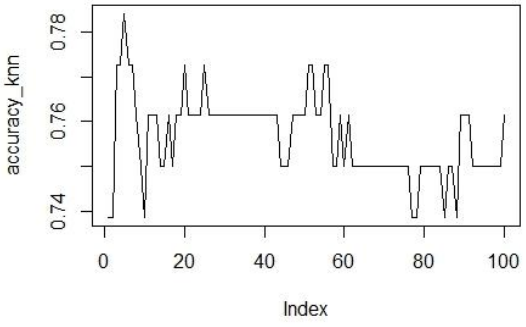


Figure 89: Accuracy Vs k number – Fold 5

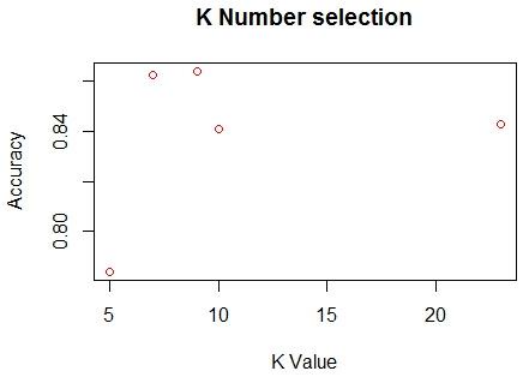


Figure 90: Maximum k numbers across folds

Hence, k = 9 was selected as the suitable k number to be used in k-NN algorithm.

Training Set

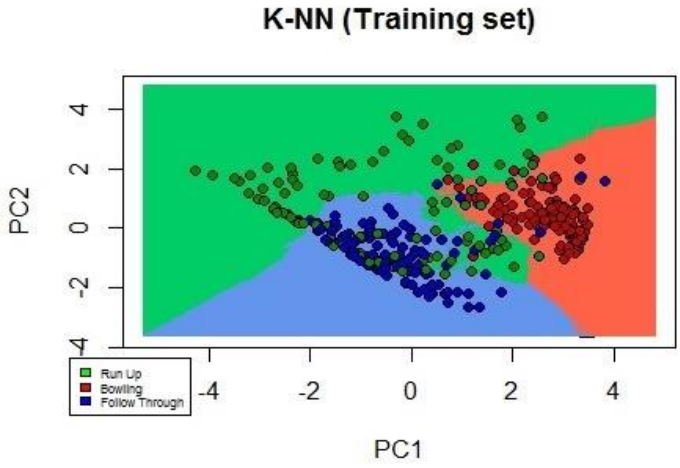


Figure 91: k-NN Training Set plot after SMOTE

Test Set

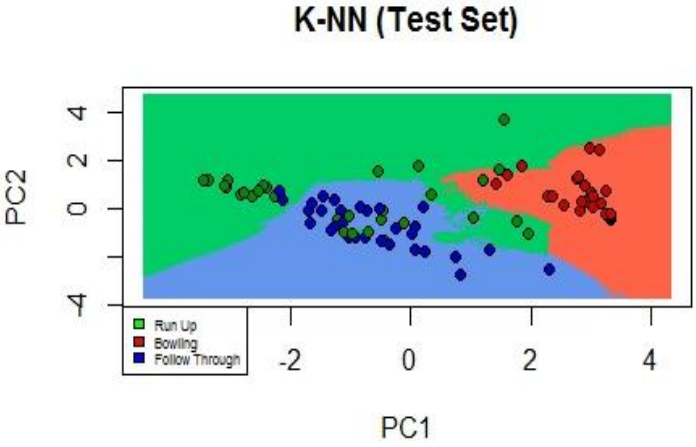


Figure 92: k-NN Test Set plot after SMOTE

Classifier Performance

Table 13: Classifier evaluation parameters

Parameter	**k-NN (%)**
Accuracy	82
Standard Deviation	+/- 4
Precision	80
Recall	60
F-Measure	68

5.5 Model Testing on Sample Dataset

The final step was to test the classification model on a sample dataset. For this requirement 5th dataset of second subject was selected.

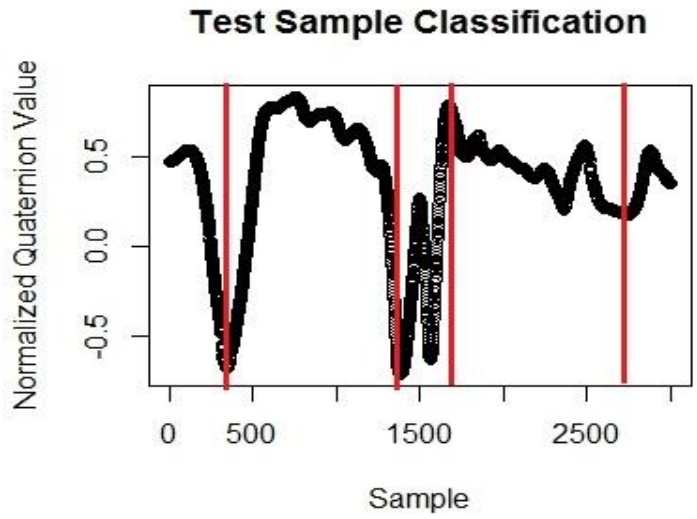


Figure 93: Test dataset plot with marked class boundaries

Above sample set was inserted into the model as a Test set and coloured vertical dotted lines were plotted on the graph to mark the different phases classified by the model. k-NN classifier demonstrated best accuracy results. Hence, it was used for this classification.

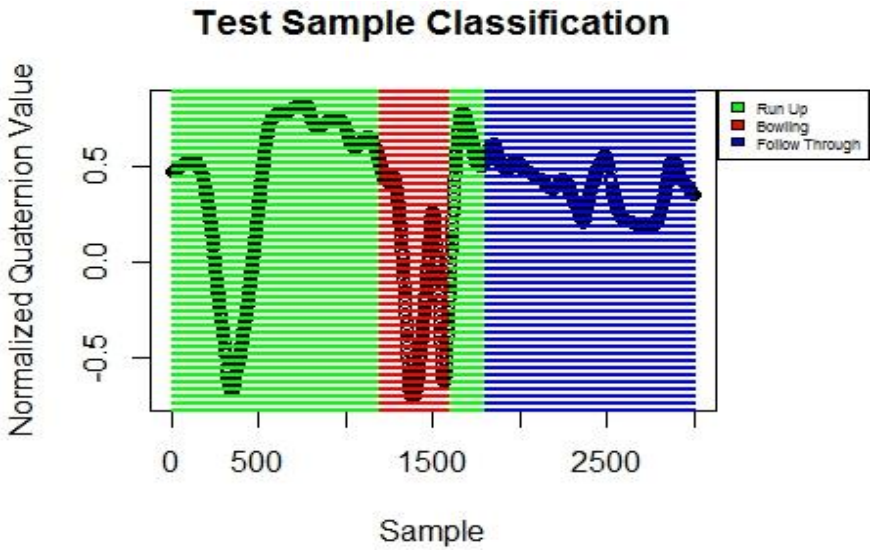


Figure 94: Test dataset plot with marked specific class regions

The main challenge for above plot was to revert to original data from the extracted feature set. However, a pattern was demonstrated in the moving window which could help to revert to original dataset.

Window 1 = Sample [0:400]

Window 2 = Sample [200:600]

Window 3 = Sample [400:800]

Window 4 = Sample [600:1000]

Window 5 = Sample [800:1200]

…….

Hence lower data point of a specific window can be linked to original dataset as,

Window_Lower [n] = (n x 200) – 200 for n=1,2, 3...

Upper data point of a specific window can be linked to original dataset as,

Window_Upper [n] = (n x 200) + 200 for n=1,2, 3...

5.6 Discussion

The classification indicates that Support Vector Machines (SVM) and Naïve Bayes Test set plots showed similar classification regions. Overall, k-NN provided best classification accuracy of 77% with a k number of 11. But it was improved to 82% with k number as 9 when SMOTE was used. All supervised classifiers used, demonstrated standard deviations of 1% to 4%. When other classification parameters were considered, best Precision rate of 79% and F-Measure of 71% was obtained by k-NN algorithm. However, best Recall measure of 70% was achieved by Random Forest algorithm. Hence, it was evident that k-Nearest Neighbour could be regarded as the best supervised classifier among the ones used of this human movement classification problem. But overall higher accuracies haven't been reached due to less number of data samples.

With the application of SMOTE to balance the sizes of three classes it was observed that k-NN classifier increased classification accuracy from 77% to 82% and Precision was improved from 79% to 80%. However, the results indicated a reduction in the performance parameters of Recall and F-Measure.

When the model was tested against a sample dataset it was observed that most of the data windows were represented to be classified accurately. However, few dataset windows were incorrectly classified.

The final accuracy of 82% obtained from k-NN classification (after SMOTE) indicates similarities to the results obtained for human activity classification by having accelerometers on the wrist [26]. In latter [26] study, accelerometers on the wrist classifies running at 80% accuracy by using a Hidden Markov Model. However, the results from the study [26] indicate a rapid increase (18%) in accuracy when a second sensor is added on the hip for classification. This trend of increase in classification accuracy continues when all three sensors are used for classification. Although the latter study uses an unsupervised classification method, there is a definite case to add a secondary sensor to increase classification accuracy of the discussed model in the current research. The results summarised on Table 8 illustrates acceptable accuracies for sensors placed on Trunk and Calf. Therefore, using multiple secondary sensors could improve classification accuracy of the model proposed from current research.

Research [27] further indicates that bowling technique is a major contributing factor towards back injuries in fast bowlers. Mixed bowling action type is considered to lead higher incidence of back injuries due to excessive lumbar spine extension and rotation during ball release and resulting in increased counter rotation during Delivery Stride. These effects have been identified to be minimal on Side-on technique of fast bowlers. Therefore, the proposed method from current research can be used to classify the back rotation and extension during Delivery Stride to correct a Mixed type fast bowling technique in a bowler towards a Side-on action type without affecting key performances of the bowler based on wearable sensors.

Further, the proposed model can be used in other similar throwing sports such as javelin throw. Research [28] indicates that Run Up speed has a direct correlation to the success of javelin throw. Hence, the proposed method can be used to classify the Run Up and other key activities in javelin throw to assist performance enhancement by using wearable sensors.

CHAPTER 6

6 CONCLUSIONS AND RECOMMENDATIONS

6.1 Key Findings

- The results of the initial experiment conducted to determine best on body sensor position for activity classification during fast bowling in cricket suggested that 'Forearm' is the best position to place Inertial Measurement Units (IMU's) to gather data for such classification problems.
- Second and Fourth Quaternions provided best overall performance for the classifiers during the study to determine best on body sensor position for data collection.
- Accuracy measures obtained by Support Vector Machine (SVM) during the study to determine best on body sensor position suggests that comparatively high accuracy rates were achieved due to large variation in sample size among the classes. Hence, the results of confusion matrices were governed by the two large classes of Run Up and Follow Through. This issue was eradicated by the usage of Synthetic Minority Over-Sampling Technique to data processed after LASSO and PCA to increase the number of data points on minority classes.
- It is important to use a secondary camera to assist in defining the boundary points of the classes. It will be best if the frame rate of camera can reach the sampling rate of sensor to increase accuracy of boundary points.
- User Diagram Protocol (UDP) could transmit data beyond the expected rate of 350Hz during data collection. However, it was observable that when UDP was used, small amounts of data packets were lost during transmission.
- k-Nearest Neighbour classifier provided the best results for activity classification in fast bowling in cricket among other supervised classifiers like Naïve Bayes, Random Forest and Support Vector Machine.
- The classification model can be improved further by adding more data and it was demonstrated that it can now be used to analyse a fast bowler for

performance enhancement or injury prevention related parameter in either Run Up, Delivery Stride or Follow Through regions.

6.2 Detailed Findings and Suggestions

6.2.1 On Body Sensor Position

The results have suggested that using an IMU on the forearm is the best position when gathering data to classify different phases in fast bowling. However, it will be important to determine the effect on the model if sensors were used in multiples to determine classification accuracy. For example, the effect of sensors on Forearm and Calf at the same instance of data gathering could be analysed as next steps. Further, to increase model accuracy more samples need to be added into the model.

6.2.2 Quaternions

Quaternions were used as the main data for classification in the research. Second and Fourth quaternions demonstrated better results in comparison to other two Quaternions. However, there is an opportunity to investigate the effect of using raw three axis accelerometer, gyroscope and magnetometer data in the model. It will also be important to determine the effect of using a more derived data from the Quaternions (Ex: Rotational angle, yaw, pitch, etc) on the classification models.

6.2.3 Inertial Measurement Units (IMU's) and Microcontroller

Since fast bowling in cricket consists of fast movements, a minimum sampling rate of 300Hz was required. However, it was observed that on certain cycles the maximum accelerometer range of 16g in MPU 9250 was not sufficient to absorb these movements. Data inversions were observed. This had a minimum effect on the research since the effect was translated to all data collection cycles. However, it is best to move towards greater accelerometer ranges (multiple ranges) for future similar fast movement activity classifications.

ESP 8266 Wi-Fi module was used in this research. When TCP was used as the data transmission protocol, it was not possible to send data at rates beyond 50Hz. However, when buffered data were sent the microcontroller did not have the

capability of processing new data. Hence, User Diagram Protocol (UDP) with the risk of losing data packets was used as the data transmission protocol. However, if a microcontroller with multiple cores was used there is a possibility of using TCP to send data beyond 300Hz.

6.2.4 Transmission Control Protocol (TCP) Vs User Diagram Protocol

As discussed previously, due to limited data rate achieved based on limitations in the Wi-Fi module when using TCP as data transmission protocol, UDP was introduced as corresponding data transmission protocol. However, when using UDP data losses were observed on certain instances.

Figure 95: UDP data losses

6.2.5 Classification of Phases in Bowling

It was more appropriate to use supervised classification models over unsupervised classifiers to the existence of limited number of sample sets (32 sample sets). However, when the sample set is increased further unsupervised methods can also be analysed for performance.

6.2.5.1 Definition of Classes

The first method to define classes by doing data collection of different phases separately demonstrated overlapping of data in the specific regions. However, this was eradicated when a reference video feedback synced with sensor data was used to

define the classes in the second method. Even this method created an error since the frame rate of camera (50 fps) was less in comparison to the sensor sampling rate (300Hz). However, it can be improved further by using a camera with the same frame rate as sensor sampling rate to define the classes.

6.2.5.2 Feature Selection

In this research eight-time domain features were considered as corresponding features for classification algorithm. There is a possibility of increasing the number of features and analysing classification accuracy. Further, the effect of using frequency domain and derived features can also be analysed in future.

6.2.5.3 Feature Extraction

LASSO and Principal Component Analysis (PCA) were used as the methods to reduce dimensionality of the feature set. However, other methods such as Backward Elimination and Ridge Regression can be tested in future.

6.2.5.4 Classification and Evaluation

As specified previously, supervised classifiers were used for classification. Best overall accuracy of 82% was achieved by k-NN classifier. All other evaluation parameters (Precision, Recall and F-Measure) have scored more than 50% rates. Hence, the model can be accepted. However, none of the evaluation parameters have reached levels beyond 82%. The model performance can be improved further by increasing number of samples. Overall, k-NN algorithm was the most suitable classifier for classification of different phases in fast bowling. However, the key challenge was to determine the best k number for the classifier.

Five-Fold Cross Validation was used as the evaluation method in the study. However, Run-Up and Follow Through classes illustrated more data points in comparison to Delivery Stride class. Hence, the results are dominated by former two classes. This effect and the effectiveness of cross validation can be improved further by increasing number of data sets for classification and by using SMOTE to balance the classes.

6.3 Future Work

This study concentrated on the classification of the three key phases in fast bowling. However, there are other phases within the main phases that need to be classified in future.

- Run Up
    - Pre-Delivery Stride
- Delivery Stride
    - Mid-bound
    - Back Foot Contact (BFC)
    - Front Foot Contact (FFC)
    - Ball Release
- Follow Through

Based on classification of phases in fast bowling, the action of a bowler can be adjusted (about a base line) in future to increase his or her bowling speed or prevent injuries due to incorrect postures during bowling.

Research [1] illustrates four key parameters that can be considered to assist bowl faster in cricket.

- Run Up speed
- Knee angle at Ball Release
- Upper trunk flexion (First Foot Contact to Ball Release)
- Shoulder angle at First Foot Contact

The study indicates that fastest bowlers have a quicker Run Up. With the addition of another relevant sensor to capture running speed, the results of current study can be used to classify the Run Up of a bowler and study his or her Run Up speed to be adjusted to help greater bowling speeds during delivery.

Further, the study can be refined more by allocating all the subjects to deliver a similar kind of delivery and determine the effects during each phase to bowling speed. This experiment cannot be conducted at this phase since delivery speed is not captured. However, it can be included for future work.

It will also be important to study the variability within classes for different bowlers when the same type of delivery is delivered. Hence, the study can be improved further by allocating all subjects the same type of delivery and studying the variability within each class on an outcome such as speed.

REFERENCES

[1] Worthington, P.J., King, M.A. and Ranson, C.A., Relationships between fast bowling technique and ball release speed in cricket. *Journal of Applied Biomechanics*, 29 (1), pp. 78 –84, 2013.

[2] Worthington, P.J., A biomechanical analysis of fast bowling in cricket. PhD Thesis, *Loughborough University*, UK, 2010.

[3] Craig. The Bowler's Back. Internet: https://biomechanics101.wordpress.com/2013/12/06/the-bowlers-back/, Dec.25, 2013

[4] Kathleen Shorter, Andrew Nealon, Neal Smith and Mike Lauder. CRICKET SIDE STRAIN INJURIES: A DESCRIPTION OF TRUNK MUSCLEACTIVITY AND THE POTENTIAL INFLUENCE OF BOWLING TECHNIQUE, *29th International Conference on Biomechanics in Sports*, Porto, Portugal, 2011.

[5] Burnett A.F., Barrett C.J., Marshall R.N., Elliott B.C. and Day R.E..Three-dimensional measurement of lumbar spine kinematics for fast bowlers in cricket. *Clinical Biomechanics*, vol. 18, issue 8, pp. 574-583, Dec. 1998.

[6] Benjamin H. Groh, Thomas Kautz, Dominik Schuldhaus and Bjoern M. Eskofier. IMU-based Trick Classification in Skateboarding. *KDD Workshop on Large-Scale Sports Analytics*, Sydney, Australia, 2015.

[7] Daniel Roetenberg. Inertial and Magnetic Sensing of Human Motion. PhD Thesis, *University of Twente*, Netherlands, Page 15, 2006.

[8] Muhammad Salman, Saad QaisarAli and Mustafa Qamar. Classification and legality analysis of bowling action in the game of cricket, *Data Mining and Knowledge Discovery*, vol. 31, issue 6, pp. 1706-1734, Nov. 2017.

[9] David Rowlands, Daniel Arthur James and David Thiel. Bowler analysis in cricket using centre of mass inertial monitoring. *Sports Technology*, Volume 2, pp. 39-42, 2009.

[10] Sebastian O.H. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. April 2010.

[11] Ferhat Attal,Samer Mohammed, Mariam Dedabrishvili, Faicel Chamroukhi, Latifa Oukhellou andYacine Amirat.Physical Human Activity Recognition Using Wearable Sensors, *Sensors*, vol. 15, issue 12, 31314–31338, 2015.

[12] Andrea Mannini and Angelo Maria Sabatini. Machine Learning Methods for Classifying Human Physical Activity from On-Body Accelerometers. Instrumentation, *Signal Treatment and Uncertainty Estimation in Sensors, Sensors*, vol. 10, issue 2, 1154-75, 2010.

[13] Warangkhana Kimpan1, Natee Rientrakulchai and Wisan Tangwongcharoen. Pattern Analysis of Golf Swing using Motion Sensors, in *Proc. ICCEB*, 2013, pp. 44-48.

[14] Takashi Aoki, Gentiane Venture, Dana Kulic. Segmentation of Human Body Movement using Inertial Measurement Unit, in *Proc. 2013 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 1181 – 1186.

[15] Worthington, P.J., King, M.A. And Ranson, C.A. Relationships between fast bowling technique and ball release speed in cricket. *Journal of Applied Biomechanics*, vol. 29, issue 1, pp. 78 – 84, 2013

[16] Saad Qaisar , Sahar Imtiaz, Fatima Farooq, Sungyoung Lee. A Hidden Markov Model For Detection And Classification Of Arm Action In Cricket Using Wearable Sensors,*Journal of Mobile Multimedia*, vol. 9, pp. 128-144, 2013.
[17] Amin Ahmadi, Edmond Mitchell, Chris Richter, Francois Destelle, Marc Gowing, Noel E. O'Connor and Kieran Moran. Towards automatic activity classification and movement assessment during a sports training session, *IEEE Internet of Things*, vol. 2, issue 1, pp. 23-32, 2014.
[18] Edmond Mitchell, David Monaghan, and Noel E. O'Connor. Classification of Sporting Activities Using Smartphone Accelerometers, *Sensors*, vol. 13, issue 4, 5317–5337, 2013.
[19] Ludovic Seifert, et.al. Pattern recognition in cyclic and discrete skills performance from inertial measurement units, *Procedia Engineering*, vol. 72, pp. 196-201, 2014.
[20] Pasi Saari. Feature Selection for Classification of Music According to Expressed Emotion, Master's Thesis, *University of Jyväskylä*, Finland, 2009.
[21] Kishor Walse, Rajiv Vasantrao Dharaskar, V. M. Thakare. PCA Based Optimal ANN Classifiers for Human Activity Recognition Using Mobile Sensors Data, in *Proc. First International Conference on Information and Communication Technology for Intelligent Systems*, 2015, vol.1, pp. 429-436.
[22] Prasnthi Mandha, G.LavanyaDevi, S. Vizianand Row. A Random Forest based Classification Model for Human Activity Recognition, *International Journal of Advanced Scientific Technologies, Engineering and Management Sciences*, vol.3, special issue 1, 2017.
[23] Statistical Tools for High-throughput Data Analytics, Internet: http://www.sthda.com/english/wiki/print.php?id=206, 2013.
[24] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, W. Phillip Kegelmeyer. SMOTE: Synthetic Minority Over – sampling Technique, *Journal of Artificial Intelligence Research*, vol 16, pp. 321-357, 2002.
[25] Valeria Fonti, Eduard Belister, Feature Selection using LASSO, *Vrije Universiteit*, Netherlands, 2017.
[26] Daniel Olguin Olguin, Alex Pentland, Human Activity Recognition: Accuracy across Common Locations for Wearable Sensors, *IEEE 10th International Symposium on Wearable Computers*, Montreaux, Switzerland, 2006.
[27] Manit Arora, Justin A Paoloni, P. Kandwal, A. D. Diwan, Are Fast-Bowlers Prone to Back Injuries? Prevalence of Lumbar Spine Injuries in Fast-Bowlers: Review of MRI-Based Studies, *Asian Journal of Sports Medicine*, vol. 5, issue 4, 2014.
[28] Žuvela Frane, Slađana Borović, Nikola Foretić, THE CORRELATION OF MOTOR ABILITIES AND JAVELIN THROWING RESULTS DEPENDS ON THE THROWING TECHNIQUE, *Physical Education and Sport*, Vol. 9, No 3, pp. 219 – 227, 2011.

APPENDIX A

MPU 9250 Sensor Data Collection and Transmission by using ESP 8266

```
#include <ESP8266WiFi.h>
//#include <WiFiUDP.h>
//#include <WiFiUdp.h>
#include <Wire.h>
#include "MPU9250.h"
char incomingPacket[255];
IPAddress ip(192, 168, 8, 240);
IPAddress gateway(192, 168, 8, 1);
IPAddress subnet(255, 255, 255, 0);
// wifi connection variables
char *total = "";
char *q0_char = "";
char *q1_char = "";
char *q2_char = "";
char *q3_char = "";
//int countx = 0;
const char* ssid = "J-4G";
const char* password = "xxxxx";
boolean wifiConnected = false;

// UDP variables
unsigned int localPort = 12345;
WiFiUDP UDP;
boolean udpConnected = false;
char packetBuffer[UDP_TX_PACKET_MAX_SIZE]; //buffer to hold incoming packet,
char ReplyBuffer[] = "1.0,0.345,0.456,0.567"; // a string to send back
```

```
#define    MPU9250_ADDRESS           0x68
#define    MAG_ADDRESS               0x0C
//Magnetometer Registers
#define AK8963_ADDRESS   0x0C
#define WHO_AM_I_AK8963  0x00 // should return 0x48
#define INFO             0x01
#define AK8963_ST1       0x02  // data ready status bit 0
#define AK8963_XOUT_L    0x03  // data
#define AK8963_XOUT_H  0x04
#define AK8963_YOUT_L  0x05
#define AK8963_YOUT_H  0x06
#define AK8963_ZOUT_L  0x07
#define AK8963_ZOUT_H  0x08
#define AK8963_ST2       0x09  // Data overflow bit 3 and data read error status bit 2
#define AK8963_CNTL      0x0A  // Power down (0000), single-measurement (0001), self-test (1000) and Fuse ROM (1111) modes on bits 3:0
#define AK8963_ASTC      0x0C  // Self test control
#define AK8963_I2CDIS    0x0F  // I2C disable
#define AK8963_ASAX      0x10  // Fuse ROM x-axis sensitivity adjustment value
#define AK8963_ASAY      0x11  // Fuse ROM y-axis sensitivity adjustment value
#define AK8963_ASAZ      0x12  // Fuse ROM z-axis sensitivity adjustment value
#define    GYRO_FULL_SCALE_250_DPS    0x00
#define    GYRO_FULL_SCALE_500_DPS    0x08
#define    GYRO_FULL_SCALE_1000_DPS   0x10
#define    GYRO_FULL_SCALE_2000_DPS   0x18
#define    ACC_FULL_SCALE_2_G        0x00
#define    ACC_FULL_SCALE_4_G        0x08
#define    ACC_FULL_SCALE_8_G        0x10
#define    ACC_FULL_SCALE_16_G       0x18
// Set initial input parameters
enum Ascale {
  AFS_2G = 0,
```

```
  AFS_4G = 1,
  AFS_8G = 2,
  AFS_16G = 3
};
enum Gscale {
  GFS_250DPS = 0,
  GFS_500DPS = 1,
  GFS_1000DPS = 2,
  GFS_2000DPS = 3
};
enum Mscale {
  MFS_14BITS = 0, // 0.6 mG per LSB
  MFS_16BITS      // 0.15 mG per LSB
};
int magRead = 20;
// Specify sensor full scale
uint8_t Gscale = GFS_2000DPS;
uint8_t Ascale = AFS_16G;
uint8_t Mscale = MFS_14BITS; // Choose either 14-bit or 16-bit magnetometer resolution
uint8_t Mmode = 0x02;        // 2 for 8 Hz, 6 for 100 Hz continuous magnetometer data read
float aRes, gRes, mRes;      // scale resolutions per LSB for the sensors
int16_t accelCount[3];  // Stores the 16-bit signed accelerometer sensor output
int16_t gyroCount[3];   // Stores the 16-bit signed gyro sensor output
int16_t magCount[3];    // Stores the 16-bit signed magnetometer sensor output
float magCalibration[3] = {1, 1, 1}, magbias[3] = {0, 0, 0};  // Factory mag calibration and
mag bias
float gyroBias[3] = {0, 0, 0}, accelBias[3] = {0, 0, 0};      // Bias corrections for gyro and
accelerometer
int16_t tempCount;      // temperature raw count output
float   temperature;    // Stores the real internal chip temperature in degrees Celsius
float   SelfTest[6];    // holds results of gyro and accelerometer self test
// global constants for 9 DoF fusion and AHRS (Attitude and Heading Reference System)
```

```
float GyroMeasError = PI * (40.0f / 180.0f);   // gyroscope measurement error in rads/s (start at 40 deg/s)

float GyroMeasDrift = PI * (0.0f  / 180.0f);   // gyroscope measurement drift in rad/s/s (start at 0.0 deg/s/s)

float beta = sqrt(3.0f / 4.0f) * GyroMeasError;   // compute beta

float zeta = sqrt(3.0f / 4.0f) * GyroMeasDrift;   // compute zeta, the other free parameter in the Madgwick scheme usually set to a small or zero value

#define Kp 2.0f * 5.0f // these are the free parameters in the Mahony filter and fusion scheme, Kp for proportional feedback, Ki for integral

#define Ki 0.0f

uint32_t delt_t = 0; // used to control display output rate

uint32_t count = 0, sumCount = 0; // used to control display output rate

float pitch, yaw, roll;

float deltat = 0.0f, sum = 0.0f;        // integration interval for both filter schemes

uint32_t lastUpdate = 0, firstUpdate = 0; // used to calculate integration interval

uint32_t Now = 0;        // used to calculate integration interval

float ax, ay, az, gx, gy, gz, mx, my, mz; // variables to hold latest sensor data values

float cal_gx, cal_gy, cal_gz;

float q[4] = {1.0f, 0.0f, 0.0f, 0.0f};    // vector to hold quaternion

float eInt[3] = {0.0f, 0.0f, 0.0f};


// This function read Nbytes bytes from I2C device at address Address.

// Put read bytes starting at register Register in the Data array.

void I2Cread(uint8_t Address, uint8_t Register, uint8_t Nbytes, uint8_t* Data) {

  // Set register address

  Wire.beginTransmission(Address);

  Wire.write(Register);

  Wire.endTransmission()

  // Read Nbytes

  Wire.requestFrom(Address, Nbytes);

  uint8_t index = 0;

  while (Wire.available())

    Data[index++] = Wire.read();
```

```
}
// Write a byte (Data) in device (Address) at register (Register)
void I2CwriteByte(uint8_t Address, uint8_t Register, uint8_t Data) {
  // Set register address
  Wire.beginTransmission(Address);
  Wire.write(Register);
  Wire.write(Data);
  Wire.endTransmission();
}
// Initial time
long int ti;
volatile bool intFlag = false;
// Initializations
void setup()
{
  // Arduino initializations
  Wire.begin(0, 2);
  Wire.setClock(400000L);
  Serial.begin(115200);
  //mySerial.begin(115200);
 wifiConnected = connectWifi();
  // only proceed if wifi connection successful
  if (wifiConnected) {
    udpConnected = connectUDP();
    if (udpConnected) {
      Serial.println("udpConnected ...");
    }
  }
  // Set accelerometers low pass filter at 5Hz
  I2CwriteByte(MPU9250_ADDRESS, 29, 0x06);
  // Set gyroscope low pass filter at 5Hz
  I2CwriteByte(MPU9250_ADDRESS, 26, 0x06);
```

```
  // Configure gyroscope range
  I2CwriteByte(MPU9250_ADDRESS, 27, GYRO_FULL_SCALE_2000_DPS);
  // Configure accelerometers range
  I2CwriteByte(MPU9250_ADDRESS, 28, ACC_FULL_SCALE_16_G);
  // Set by pass mode for the magnetometers
  I2CwriteByte(MPU9250_ADDRESS, 0x37, 0x02);
  // Request continuous magnetometer measurements in 16 bits
  I2CwriteByte(MAG_ADDRESS, 0x0A, 0x16);
  // Get magnetometer calibration from AK8963 ROM
  //initAK8963(magCalibration); Serial.println("AK8963 initialized for active data mode...."); // Initialize device for active mode read of magnetometer
  //pinMode(13, OUTPUT);
  //Timer1.initialize(10000);         // initialize timer1, and set a 1/2 second period
  //Timer1.attachInterrupt(callback);  // attaches callback() as a timer overflow interrupt
  getGres();
  getAres();
  getMres();
  mRes = 10 * 0.6; //conversion from 1229 microTesla full scale 4096 to 12.29 Gauss full scale
  Gyro_cal();
  magbias[0] = -786;  // User environmental x-axis correction in milliGauss, should be automatically calculated
  magbias[1] = -396;  // User environmental x-axis correction in milliGauss
  magbias[2] = 1497;  // User environmental x-axis correction in milliGauss
  // Store initial time
  ti = millis();
  firstUpdate = micros();
}
// Counter
long int cpt = 0, magcpt = 0;
void Gyro_cal() {
  uint8_t Buf[14];
  for (int i = 0; i < 100 ; i++) {
```

```
   I2Cread(MPU9250_ADDRESS, 0x3B, 14, Buf);
  }
  cal_gx = 0; cal_gy = 0; cal_gz = 0;

  for (int i = 0; i < 50 ; i++) {
   I2Cread(MPU9250_ADDRESS, 0x3B, 14, Buf);
   gyroCount[0] = -(Buf[8] << 8 | Buf[9]);
   gyroCount[1] = -(Buf[10] << 8 | Buf[11]);
   gyroCount[2] = Buf[12] << 8 | Buf[13];

   // Calculate the gyro value into actual degrees per second
   gx = (float)gyroCount[0] * gRes; // - .664;  // get actual gyro value, this depends on scale
being set
   gy = (float)gyroCount[1] * gRes; // - (-0.221);
   gz = (float)gyroCount[2] * gRes;
   delay(1);
   cal_gx += gx; cal_gy += gy; cal_gz += gz;
  }
  cal_gx /= 50.; cal_gy /= 50.; cal_gz /= 50.;
}
// Main loop, read and display data
void loop() {
  if (wifiConnected) {
   if (udpConnected) {
     if (getUdpData() == 1) {
       uint8_t Buf[14];
       I2Cread(MPU9250_ADDRESS, 0x3B, 14, Buf);
       // Create 16 bits values from 8 bits data
       // Accelerometer
       accelCount[0] = -(Buf[0] << 8 | Buf[1]);
       accelCount[1] = -(Buf[2] << 8 | Buf[3]);
       accelCount[2] = (Buf[4] << 8 | Buf[5]);
```

```
ax = (float)accelCount[0] * aRes; // - accelBias[0];  // get actual g value, this depends on scale being set

ay = (float)accelCount[1] * aRes; // - accelBias[1];

az = (float)accelCount[2] * aRes; // - 0.490; // - accelBias[2];

// Gyroscope

gyroCount[0] = -(Buf[8] << 8 | Buf[9]);

gyroCount[1] = -(Buf[10] << 8 | Buf[11]);

gyroCount[2] = (Buf[12] << 8 | Buf[13]);

// Calculate the gyro value into actual degrees per second

gx = (float)gyroCount[0] * gRes - cal_gx; // - .664;  // get actual gyro value, this depends on scale being set

gy = (float)gyroCount[1] * gRes - cal_gy; // - (-0.221);

gz = (float)gyroCount[2] * gRes - cal_gz; // - 0.031;

magcpt++;

if (magcpt / magRead == 1) {

  magcpt = 0;

  // Read register Status 1 and wait for the DRDY: Data Ready

  uint8_t ST1;

  do

  {

    I2Cread(MAG_ADDRESS, 0x02, 1, &ST1);

} while (!(ST1 & 0x01));

  // Read magnetometer data

  uint8_t Mag[7];

  I2Cread(MAG_ADDRESS, 0x03, 7, Mag);

  // Create 16 bits values from 8 bits data

  // Magnetometer

  magCount[0] = -(Mag[3] << 8 | Mag[2]);

  magCount[1] = -(Mag[1] << 8 | Mag[0]);

  magCount[2] = -(Mag[5] << 8 | Mag[4]);

}

mx = (float)magCount[0] * mRes - magbias[0]; // get actual magnetometer value, this depends on scale being set
```

```
my = (float)magCount[1] * mRes - magbias[1];
mz = (float)magCount[2] * mRes - magbias[2];
Now = micros();
deltat = ((Now - lastUpdate) / 1000000.0f); // set integration time by time elapsed since last filter update
lastUpdate = Now;
if ((lastUpdate - firstUpdate) > 10000000)
{
  //beta = 0.004;
  beta = 0.04;
  zeta = 0.015;
}
sum += deltat; // sum for averaging filter update rate
sumCount++;
MadgwickQuaternionUpdate(ax, ay, az, gx * PI / 180.0f, gy * PI / 180.0f, gz * PI / 180.0f,  mx,  my, mz);
//countx++;
//UDP.beginPacket(ip, localPort);
UDP.beginPacket(UDP.remoteIP(), UDP.remotePort());
UDP.print("$");
UDP.print(String(millis()));
UDP.print(",");
dtostrf(q[0], 5, 4, q0_char);
UDP.write(q0_char);
UDP.write(",");
dtostrf(q[1], 5, 4, q1_char);
UDP.write(q1_char);
UDP.write(",");
dtostrf(q[2], 5, 4, q2_char);
UDP.write(q2_char);
UDP.write(",");
dtostrf(q[3], 5, 4, q3_char);
UDP.write(q3_char);
```

```
        UDP.println("#");
        UDP.endPacket();
        count = millis();
        sumCount = 0;
        sum = 0;
      }
    }
  }
}

void getMres() {
  switch (Mscale) {
    // Possible magnetometer scales (and their register bit settings) are:
    // 14 bit resolution (0) and 16 bit resolution (1)
    case MFS_14BITS:
      mRes = 10.*4912. / 8190.; // Proper scale to return milliGauss
      break;
    case MFS_16BITS:
      mRes = 10.*4912. / 32760.0; // Proper scale to return milliGauss
      break;
  }
}
void getGres() {
  switch (Gscale) {
    // Possible gyro scales (and their register bit settings) are:
    // 250 DPS (00), 500 DPS (01), 1000 DPS (10), and 2000 DPS  (11).
    // Here's a bit of an algorith to calculate DPS/(ADC tick) based on that 2-bit value:
    case GFS_250DPS:
      gRes = 250.0 / 32768.0;
      break;
    case GFS_500DPS:
      gRes = 500.0 / 32768.0;
```

```
      break;
    case GFS_1000DPS:
      gRes = 1000.0 / 32768.0;
      break;
    case GFS_2000DPS:
      gRes = 2000.0 / 32768.0;
      break;
  }
}
void getAres() {
  switch (Ascale) {
    // Possible accelerometer scales (and their register bit settings) are:
    // 2 Gs (00), 4 Gs (01), 8 Gs (10), and 16 Gs  (11).
    // Here's a bit of an algorith to calculate DPS/(ADC tick) based on that 2-bit value:
    case AFS_2G:
      aRes = 2.0 / 32768.0;
      break;
    case AFS_4G:
      aRes = 4.0 / 32768.0;
      break;
    case AFS_8G:
      aRes = 8.0 / 32768.0;
      break;
    case AFS_16G:
      aRes = 16.0 / 32768.0;
      break;
  }
}
void initAK8963(float * destination) {
  // First extract the factory calibration for each magnetometer axis
  uint8_t rawData[3];  // x/y/z gyro calibration data stored here
  writeByte(AK8963_ADDRESS, AK8963_CNTL, 0x00); // Power down magnetometer
```

```
  delay(10);

  writeByte(AK8963_ADDRESS, AK8963_CNTL, 0x0F); // Enter Fuse ROM access mode

  delay(10);

  readBytes(AK8963_ADDRESS, AK8963_ASAX, 3, &rawData[0]);  // Read the x-, y-, and z-axis calibration values

  destination[0] =  (float)(rawData[0] - 128) / 256. + 1.; // Return x-axis sensitivity adjustment values, etc.

  destination[1] =  (float)(rawData[1] - 128) / 256. + 1.;

  destination[2] =  (float)(rawData[2] - 128) / 256. + 1.;

  writeByte(AK8963_ADDRESS, AK8963_CNTL, 0x00); // Power down magnetometer

  delay(10);

  // Configure the magnetometer for continuous read and highest resolution

  // set Mscale bit 4 to 1 (0) to enable 16 (14) bit resolution in CNTL register,

  // and enable continuous mode data acquisition Mmode (bits [3:0]), 0010 for 8 Hz and 0110 for 100 Hz sample rates

  writeByte(AK8963_ADDRESS, AK8963_CNTL, Mscale << 4 | Mmode); // Set magnetometer data resolution and sample ODR

  delay(10);

}

// Implementation of Sebastian Madgwick's "...efficient orientation filter for... inertial/magnetic sensor arrays"

// (see http://www.x-io.co.uk/category/open-source/ for examples and more details)

// which fuses acceleration, rotation rate, and magnetic moments to produce a quaternion-based estimate of absolute

// device orientation -- which can be converted to yaw, pitch, and roll. Useful for stabilizing quadcopters, etc.

// The performance of the orientation filter is at least as good as conventional Kalman-based filtering algorithms

// but is much less computationally intensive---it can be performed on a 3.3 V Pro Mini operating at 8 MHz!

void MadgwickQuaternionUpdate(float ax, float ay, float az, float gx, float gy, float gz, float mx, float my, float mz) {

  float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];   // short name local variable for readability

  float norm;

  float hx, hy, _2bx, _2bz;

  float s1, s2, s3, s4;
```

```
float qDot1, qDot2, qDot3, qDot4;
// Auxiliary variables to avoid repeated arithmetic
float _2q1mx;
float _2q1my;
float _2q1mz;
float _2q2mx;
float _4bx;
float _4bz;
float _2q1 = 2.0f * q1;
float _2q2 = 2.0f * q2;
float _2q3 = 2.0f * q3;
float _2q4 = 2.0f * q4;
float _2q1q3 = 2.0f * q1 * q3;
float _2q3q4 = 2.0f * q3 * q4;
float q1q1 = q1 * q1;
float q1q2 = q1 * q2;
float q1q3 = q1 * q3;
float q1q4 = q1 * q4;
float q2q2 = q2 * q2;
float q2q3 = q2 * q3;
float q2q4 = q2 * q4;
float q3q3 = q3 * q3;
float q3q4 = q3 * q4;
float q4q4 = q4 * q4;
// Normalise accelerometer measurement
norm = sqrt(ax * ax + ay * ay + az * az);
if (norm == 0.0f) return; // handle NaN
norm = 1.0f / norm;
ax *= norm;
ay *= norm;
az *= norm;
// Normalise magnetometer measurement
```

```
  norm = sqrt(mx * mx + my * my + mz * mz);

  if (norm == 0.0f) return; // handle NaN

  norm = 1.0f / norm;

  mx *= norm;

  my *= norm;

  mz *= norm;


  // Reference direction of Earth's magnetic field //   FOR SRILANKA MAGNETIC FEILD CAN BE ASSUMED AS ZERO ALMOST FLAT THAT IS WHY _2bz,_4bz are made to zero

  _2q1mx = 2.0f * q1 * mx;

  _2q1my = 2.0f * q1 * my;

  _2q1mz = 2.0f * q1 * mz;

  _2q2mx = 2.0f * q2 * mx;

  hx = mx * q1q1 - _2q1my * q4 + _2q1mz * q3 + mx * q2q2 + _2q2 * my * q3 + _2q2 * mz * q4 - mx * q3q3 - mx * q4q4;

  hy = _2q1mx * q4 + my * q1q1 - _2q1mz * q2 + _2q2mx * q3 - my * q2q2 + my * q3q3 + _2q3 * mz * q4 - my * q4q4;

  _2bx = sqrt(hx * hx + hy * hy);

  _2bz = 0; // -_2q1mx * q3 + _2q1my * q2 + mz * q1q1 + _2q2mx * q4 - mz * q2q2 + _2q3 * my * q4 - mz * q3q3 + mz * q4q4;

  _4bx = 2.0f * _2bx;

  _4bz = 2.0f * _2bz;

  // Gradient decent algorithm corrective step

  s1 = -_2q3 * (2.0f * q2q4 - _2q1q3 - ax) + _2q2 * (2.0f * q1q2 + _2q3q4 - ay) - _2bz * q3 * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q4 + _2bz * q2) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q3 * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);

  s2 = _2q4 * (2.0f * q2q4 - _2q1q3 - ax) + _2q1 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q2 * (1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + _2bz * q4 * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (_2bx * q3 + _2bz * q1) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + (_2bx * q4 - _4bz * q2) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);

  s3 = -_2q1 * (2.0f * q2q4 - _2q1q3 - ax) + _2q4 * (2.0f * q1q2 + _2q3q4 - ay) - 4.0f * q3 * (1.0f - 2.0f * q2q2 - 2.0f * q3q3 - az) + (-_4bx * q3 - _2bz * q1) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (_2bx * q2 + _2bz * q4) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + (_2bx * q1 - _4bz * q3) * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
```

```
  s4 = _2q2 * (2.0f * q2q4 - _2q1q3 - ax) + _2q3 * (2.0f * q1q2 + _2q3q4 - ay) + (-_4bx * q4 + _2bz * q2) * (_2bx * (0.5f - q3q3 - q4q4) + _2bz * (q2q4 - q1q3) - mx) + (-_2bx * q1 + _2bz * q3) * (_2bx * (q2q3 - q1q4) + _2bz * (q1q2 + q3q4) - my) + _2bx * q2 * (_2bx * (q1q3 + q2q4) + _2bz * (0.5f - q2q2 - q3q3) - mz);
  norm = sqrt(s1 * s1 + s2 * s2 + s3 * s3 + s4 * s4);    // normalise step magnitude
  norm = 1.0f / norm;
  s1 *= norm;
  s2 *= norm;
  s3 *= norm;
  s4 *= norm;
  // Compute rate of change of quaternion
  qDot1 = 0.5f * (-q2 * gx - q3 * gy - q4 * gz) - beta * s1;
  qDot2 = 0.5f * (q1 * gx + q3 * gz - q4 * gy) - beta * s2;
  qDot3 = 0.5f * (q1 * gy - q2 * gz + q4 * gx) - beta * s3;
  qDot4 = 0.5f * (q1 * gz + q2 * gy - q3 * gx) - beta * s4;
  // Integrate to yield quaternion
  q1 += qDot1 * deltat;
  q2 += qDot2 * deltat;
  q3 += qDot3 * deltat;
  q4 += qDot4 * deltat;
  norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);    // normalise quaternion
  norm = 1.0f / norm;
  q[0] = q1 * norm;
  q[1] = q2 * norm;
  q[2] = q3 * norm;
  q[3] = q4 * norm;
}
// Wire.h read and write protocols
void writeByte(uint8_t address, uint8_t subAddress, uint8_t data) {
  Wire.beginTransmission(address);  // Initialize the Tx buffer
  Wire.write(subAddress);           // Put slave register address in Tx buffer
  Wire.write(data);                 // Put data in Tx buffer
  Wire.endTransmission();           // Send the Tx buffer
```

```
}
uint8_t readByte(uint8_t address, uint8_t subAddress) {
  uint8_t data; // `data` will store the register data
  Wire.beginTransmission(address);         // Initialize the Tx buffer
  Wire.write(subAddress);                  // Put slave register address in Tx buffer
  Wire.endTransmission(false);             // Send the Tx buffer, but send a restart to keep connection alive
  Wire.requestFrom(address, (uint8_t) 1);  // Read one byte from slave register address
  data = Wire.read();                      // Fill Rx buffer with result
  return data;                             // Return data read from slave register
}
void readBytes(uint8_t address, uint8_t subAddress, uint8_t count, uint8_t * dest) {
  Wire.beginTransmission(address);   // Initialize the Tx buffer
  Wire.write(subAddress);            // Put slave register address in Tx buffer
  Wire.endTransmission(false);       // Send the Tx buffer, but send a restart to keep connection alive
  uint8_t i = 0;
  Wire.requestFrom(address, count);  // Read bytes from slave register address
  while (Wire.available()) {
    dest[i++] = Wire.read();
  }         // Put read results in the Rx buffer
}
void MahonyQuaternionUpdate(float ax, float ay, float az, float gx, float gy, float gz, float mx, float my, float mz) {
  float q1 = q[0], q2 = q[1], q3 = q[2], q4 = q[3];   // short name local variable for readability
  float norm;
  float hx, hy, bx, bz;
  float vx, vy, vz, wx, wy, wz;
  float ex, ey, ez;
  float pa, pb, pc;
  // Auxiliary variables to avoid repeated arithmetic
  float q1q1 = q1 * q1;
  float q1q2 = q1 * q2;
```

```
  float q1q3 = q1 * q3;
  float q1q4 = q1 * q4;
  float q2q2 = q2 * q2;
  float q2q3 = q2 * q3;
  float q2q4 = q2 * q4;
  float q3q3 = q3 * q3;
  float q3q4 = q3 * q4;
  float q4q4 = q4 * q4;
  // Normalise accelerometer measurement
  norm = sqrt(ax * ax + ay * ay + az * az);
  if (norm == 0.0f) return; // handle NaN
  norm = 1.0f / norm;        // use reciprocal for division
  ax *= norm;
  ay *= norm;
  az *= norm;
  // Normalise magnetometer measurement
  norm = sqrt(mx * mx + my * my + mz * mz);
  if (norm == 0.0f) return; // handle NaN
  norm = 1.0f / norm;        // use reciprocal for division
  mx *= norm;
  my *= norm;
  mz *= norm;
  // Reference direction of Earth's magnetic field
  hx = 2.0f * mx * (0.5f - q3q3 - q4q4) + 2.0f * my * (q2q3 - q1q4) + 2.0f * mz * (q2q4 +
q1q3);
  hy = 2.0f * mx * (q2q3 + q1q4) + 2.0f * my * (0.5f - q2q2 - q4q4) + 2.0f * mz * (q3q4 -
q1q2);
  bx = sqrt((hx * hx) + (hy * hy));
  bz = 2.0f * mx * (q2q4 - q1q3) + 2.0f * my * (q3q4 + q1q2) + 2.0f * mz * (0.5f - q2q2 -
q3q3);
  // Estimated direction of gravity and magnetic field
  vx = 2.0f * (q2q4 - q1q3);
  vy = 2.0f * (q1q2 + q3q4);
```

```
vz = q1q1 - q2q2 - q3q3 + q4q4;
wx = 2.0f * bx * (0.5f - q3q3 - q4q4) + 2.0f * bz * (q2q4 - q1q3);
wy = 2.0f * bx * (q2q3 - q1q4) + 2.0f * bz * (q1q2 + q3q4);
wz = 2.0f * bx * (q1q3 + q2q4) + 2.0f * bz * (0.5f - q2q2 - q3q3);
// Error is cross product between estimated direction and measured direction of gravity
ex = (ay * vz - az * vy) + (my * wz - mz * wy);
ey = (az * vx - ax * vz) + (mz * wx - mx * wz);
ez = (ax * vy - ay * vx) + (mx * wy - my * wx);
if (Ki > 0.0f)
{
  eInt[0] += ex;      // accumulate integral error
  eInt[1] += ey;
  eInt[2] += ez;
}
else
{
  eInt[0] = 0.0f;     // prevent integral wind up
  eInt[1] = 0.0f;
  eInt[2] = 0.0f;
}
// Apply feedback terms
gx = gx + Kp * ex + Ki * eInt[0];
gy = gy + Kp * ey + Ki * eInt[1];
gz = gz + Kp * ez + Ki * eInt[2];
// Integrate rate of change of quaternion
pa = q2;
pb = q3;
pc = q4;
q1 = q1 + (-q2 * gx - q3 * gy - q4 * gz) * (0.5f * deltat);
q2 = pa + (q1 * gx + pb * gz - pc * gy) * (0.5f * deltat);
q3 = pb + (q1 * gy - pa * gz + pc * gx) * (0.5f * deltat);
q4 = pc + (q1 * gz + pa * gy - pb * gx) * (0.5f * deltat);
```

```
  // Normalise quaternion
  norm = sqrt(q1 * q1 + q2 * q2 + q3 * q3 + q4 * q4);
  norm = 1.0f / norm;
  q[0] = q1 * norm;
  q[1] = q2 * norm;
  q[2] = q3 * norm;
  q[3] = q4 * norm;
}
boolean connectUDP() {
  boolean state = false;
  Serial.println("");
  Serial.println("Connecting to UDP");
  if (UDP.begin(localPort) == 1) {
    Serial.println("Connection successful");
    state = true;
  }
  else {
    Serial.println("Connection failed");
  }
  return state;
}
// connect to wifi – returns true if successful or false if not
boolean connectWifi() {
  boolean state = true;
  int i = 0;
  WiFi.begin(ssid, password);
  Serial.println("");
  Serial.println("Connecting to WiFi");
  // Wait for connection
  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
```

```
    Serial.print(".");
    i++;
  }
  if (state) {
    Serial.println("");
    Serial.print("Connected to ");
    Serial.println(ssid);
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    WiFi.config(ip, gateway, subnet);
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
  Serial.print(WiFi.localIP());
  }
  else {
    Serial.println("");
    Serial.println("Connection failed.");
  }
  return state;
}
int getUdpData() {
  int packetSize = UDP.parsePacket();
  if (packetSize)
  {
    // receive incoming UDP packets
    //Serial.printf("Received %d bytes from %s, port %d\n", packetSize,
UDP.remoteIP().toString().c_str(), UDP.remotePort());
    int len = UDP.read(incomingPacket, 255);
    if (len > 0)
    {
      incomingPacket[len] = 0;
```

```
    }
    //Serial.printf("UDP packet contents: %s\n", incomingPacket);
    return 1;
  }
  return 0;
}
```

APPENDIX B

Python Socket Program to Receive Data from ESP8266 using UDP

```
import socket               # Import socket module
import sys
import datetime
import Tkinter as tk
import threading
import time
run_flag=False
save_flag=False
save_old_flag=False
run_old_flag=False
value=[0,0,0,0,0,0];
time_diff=0
old_millis=0
samples_per_second=0
sample_count=0
def net_thread():
    global run_flag
    global run_old_flag
    global save_old_flag
    global time_diff
    global old_millis
    global value
    global samples_per_second
    global sample_count
print "Waiting for connection..."
    s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)        # Create a socket object
    server_address=('192.168.8.100',8899)
    s.connect(server_address)
```

```
s.settimeout(0.05);
print "Connected."
while True:
    while run_flag:
        if save_flag==True:
            if save_old_flag==False:
                save_old_flag=True
                f = open('data_server1.csv', 'a')
        else:
            if save_old_flag==True:
                save_old_flag=False
                f.close()
        s.sendto("R",('192.168.8.100',8899))
        try:
            data = s.recvfrom(1024)
        except socket.timeout:
            print "Socket timeout."
            break
        millis = int(round(time.time() * 1000))
        time_diff=millis-old_millis
        if time_diff>1000:
            value[5]=sample_count
            sample_count=0
            old_millis=millis
        if data:
                now = datetime.datetime.now()
                records=data[0].splitlines()
                for record in records:
                    print record
                    if "#" in record and "$" in record :
                        clean_record=record.lstrip('$')
                        clean_record2=clean_record.rstrip('#')
```

```
                    if "#" not in clean_record2 and "$" not in clean_record2 :
                        value_temp = clean_record2.split(",")
                        value[0]=value_temp[0]
                        value[1]=value_temp[1]
                        value[2]=value_temp[2]
                        value[3]=value_temp[3]
 value[4]=value_temp[4]
                        sample_count=sample_count+1
                        if save_flag and save_old_flag:
                            f.write(unicode(now))
                            f.write(',')
                            f.write(clean_record2);
                            f.write('\n');
def net_start():
    global run_flag
    run_flag=True
    print "Starting..."
def net_stop():
    global run_flag
    run_flag=False
    print "Stopping..."
def save_on():
    global save_flag
    save_flag=True
    print "Save on..."
def save_off():
    global save_flag
    save_flag=False
    print "Save off..."
def exit_1():
    if(run_flag==False):
        print "Exiting"
```

```
        exit()
    else:
        print "Still Running, Stopping now..."
        net_stop()
        time.sleep(2)
        print "Exiting"
        exit()
def value_label(label,i):
  def label_update():
    global value
    label.config(text=str(value[i]))
   label.after(10,label_update)
  label_update()
t2 = threading.Thread(target=net_thread)
t2.setDaemon(True)
t2.start()
root = tk.Tk()
root.title("Data Gathering")
root.resizable(width=False, height=False)
root.geometry('{}x{}'.format(200, 350))
#top_frame = Frame(root, bg='cyan', width = 450, height=50, pady=3).grid(row=0,
columnspan=3)
w = tk.Label(root, text="UDP Socket Data Transfer")#.grid(row = 0, columnspan = 3)
w.pack()
label1 = tk.Label(root, fg="green")
label1.pack()
value_label(label1,0)
label2 = tk.Label(root, fg="green")
label2.pack()
value_label(label2,1)
label3 = tk.Label(root, fg="green")
label3.pack()
```

```
value_label(label3,2)
label4 = tk.Label(root, fg="green")
label4.pack()
value_label(label4,3)
label5 = tk.Label(root, fg="green")
label5.pack()
value_label(label5,4)
w1 = tk.Label(root, text="Samples per second")#.grid(row = 0, columnspan = 3)
w1.pack()
label6 = tk.Label(root, fg="green")
label6.pack()
value_label(label6,5)
button_start = tk.Button(root, text='Start', width=25, command=net_start)
button_start.pack()
button_stop = tk.Button(root, text='Stop', width=25, command=net_stop)
button_stop.pack()
button_save_on = tk.Button(root, text='Save ON', width=25, command=save_on)
button_save_on.pack()
button_save_off = tk.Button(root, text='Save OFF', width=25, command=save_off)
button_save_off.pack()
button_exit = tk.Button(root, text='Exit', width=25, command=exit_1)
button_exit.pack()
root.mainloop()
```

APPENDIX C

Support Vector Machine (SVM) Classification using R

```
# Load data
j=1  ## Bowling
p=1   ##runup
t=1  ##Bowling
l=1200
e = 1
k=1  ##runup
i=1
s=1 ##Folllow
w=1 ##Follow
b=1 ##Test
z=1 ##Test
q=1
m=1  ## Cross validation
a=1 ## Cross validation
knearest = 1
knearest_2 = 1
#install.packages('XLConnect')
#library (XLConnect)
dataset <- read.csv('Dataset.csv')
#Load Library
#install.packages('ElemStatLearn')
#install.packages('e1071')
#install.packages('seewave')
#install.packages('randomForest')
library(caTools)
library(e1071)
library(seewave)
library(caret)
```

```
# Get Column 4

dataset_runup <- dataset[1]

dataset_bowling <- dataset[2]

dataset_follow <- dataset[3]

#Get all non zero values

non_zero_runup <- matrix(na.omit(dataset_runup[dataset_runup!=0]))

non_zero_bowling <- matrix(na.omit(dataset_bowling[dataset_bowling!=0]))

non_zero_follow<- matrix(na.omit(dataset_follow[dataset_follow!=0]))

data_size = length(non_zero_bowling)

###Get non 'NA' values

#non_na = matrix(na.omit(data_non_zero))

##non_zero_bowling_na = matrix(na.omit(non_zero_bowling))

##non_zero_bowling_na = non_zero_bowling[625:data_size,1]

##Plottting intial data for visualization

#plot(non_na,xlim = c(0,2000),ylim = c(-0.2,1),type = "l", col= "red")

#par(new=TRUE)

plot(non_zero_runup,xlim = c(0,13000),ylim = c(-1.5,1),type = "l", col= "green", xlab =
"Sample", ylab = "Normalized Quaternion Value", main = "Initial classes plot")

par(new=TRUE)

plot(non_zero_bowling, xlim = c(0,2500),ylim = c(-1.5,1),type = "l", col= "red", xlab =
"Sample", ylab = "Normalized Quaternion Value", main = "Initial classes plot")

par(new=TRUE)

plot(non_zero_follow, xlim = c(0,2500),ylim = c(-1.5,1),type = "l", col= "blue", xlab =
"Sample", ylab = "Normalized Quaternion Value", main = "Initial classes plot")

legend(1000,-0.5,c("Run Up", "Bowling", "Follow Through"), pch = c(1,1), fill = c("green",
"red", "blue"), cex = 0.8)

################Bowling

# Remainder data for matrices

#Get run up cluster

length_runup = length(non_zero_runup)

remainder_runup= length_runup%%20

#Runup Matrices

mean_runup <- matrix(nrow = 500,ncol = 1)
```

```
median_runup <- matrix(nrow = 500,ncol = 1)
variance_runup <- matrix(nrow = 500,ncol = 1)
iqr_runup <- matrix(nrow = 500,ncol = 1)
skewness_runup<- matrix(nrow = 500,ncol = 1)
kurtosis_runup <- matrix(nrow = 500,ncol = 1)
rms_runup<- matrix(nrow = 500,ncol = 1)
mad_runup <- matrix(nrow = 500,ncol = 1)
#rms_bowling[1,1] = rms_bowling_20
#Calculate statistical parameters for runup data
for (k in seq(175,(length_runup-remainder_runup),175)){
  val1_run=(k-175)
  val2_run=(k+175)
  mean_runup[p,1] = mean(non_zero_runup[val1_run:val2_run],na.rm = TRUE)
  median_runup[p,1] = median(non_zero_runup[val1_run:val2_run],na.rm = TRUE)
  variance_runup[p,1] = var(non_zero_runup[val1_run:val2_run],na.rm = TRUE)
  iqr_runup[p,1] = IQR(non_zero_runup[val1_run:val2_run],na.rm = TRUE)
  skewness_runup[p,1] = skewness(non_zero_runup[val1_run:val2_run],na.rm = TRUE, type
= 1)
  kurtosis_runup[p,1] = kurtosis(non_zero_runup[val1_run:val2_run],na.rm = TRUE, type =
3)
  rms_runup[p,1] = rms(non_zero_runup[val1_run:val2_run],na.rm = TRUE)
  mad_runup[p,1] = mad(non_zero_runup[val1_run:val2_run],na.rm = TRUE)
  p=p+1
}
f = length(na.omit(mean_runup))
#Plot Data bowling
Runup_mat <- matrix(nrow = f, ncol = 9)
Runup_mat[,1]= c(na.omit(mean_runup[,1]))
Runup_mat[,2]= c(na.omit(median_runup[,1]))
Runup_mat[,3] = c(na.omit(variance_runup[,1]))
Runup_mat[,4] = c(na.omit(iqr_runup[,1]))
Runup_mat[,5] = c(na.omit(skewness_runup[,1]))
Runup_mat[,6] = c(na.omit(kurtosis_runup[,1]))
```

```
Runup_mat[,7] = c(na.omit(rms_runup[,1]))
Runup_mat[,8] = c(na.omit(mad_runup[,1]))
#Bowling_mat[,7] = c(na.omit(rms_bowling[,1]))
for(e in seq(1,f,1)){
  Runup_mat[e,9] = 1
}
###Bowling Window
length_bowling = length(non_zero_bowling)
remainder_bowling= length_bowling%%20
#Bowling Matrices
mean_bowling <- matrix(nrow = 500,ncol = 1)
median_bowling <- matrix(nrow = 500,ncol = 1)
variance_bowling <- matrix(nrow = 500,ncol = 1)
iqr_bowling <- matrix(nrow = 500,ncol = 1)
skewness_bowling<- matrix(nrow = 500,ncol = 1)
kurtosis_bowling <- matrix(nrow = 500,ncol = 1)
rms_bowling <- matrix(nrow = 500,ncol = 1)
mad_bowling<- matrix(nrow = 500,ncol = 1)
#rms_bowling[1,1] = rms_bowling_20
#Calculate statistical parameters for Bowling data
for (j in seq(175,(length_bowling-remainder_bowling),175)){
  val1_bowl=(j-175)
  val2_bowl=(j+175)
  mean_bowling[t,1] = mean(non_zero_bowling[val1_bowl:val2_bowl],na.rm = TRUE)
  median_bowling[t,1] = median(non_zero_bowling[val1_bowl:val2_bowl],na.rm = TRUE)
  variance_bowling[t,1] = var(non_zero_bowling[val1_bowl:val2_bowl],na.rm = TRUE)
  iqr_bowling[t,1] = IQR(non_zero_bowling[val1_bowl:val2_bowl],na.rm = TRUE)
  skewness_bowling[t,1] = skewness(non_zero_bowling[val1_bowl:val2_bowl],na.rm =
TRUE, type = 1)
  kurtosis_bowling[t,1] = kurtosis(non_zero_bowling[val1_bowl:val2_bowl],na.rm = TRUE,
type = 3)
  rms_bowling[t,1] = rms(non_zero_bowling[val1_bowl:val2_bowl],na.rm = TRUE)
  mad_bowling[t,1] = mad(non_zero_bowling[val1_bowl:val2_bowl],na.rm = TRUE)
```

```
  t=t+1
}
v = length(na.omit(mean_bowling))
#Plot Data bowling
Bowling_mat <- matrix(nrow = v, ncol = 9)
Bowling_mat[,1]= c(na.omit(mean_bowling[,1]))
Bowling_mat[,2]= c(na.omit(median_bowling[,1]))
Bowling_mat[,3] = c(na.omit(variance_bowling[,1]))
Bowling_mat[,4] = c(na.omit(iqr_bowling[,1]))
Bowling_mat[,5] = c(na.omit(skewness_bowling[,1]))
Bowling_mat[,6] = c(na.omit(kurtosis_bowling[,1]))
Bowling_mat[,7] = c(na.omit(rms_bowling[,1]))
Bowling_mat[,8] = c(na.omit(mad_bowling[,1]))
#Bowling_mat[,7] = c(na.omit(rms_bowling[,1]))
for(e in seq(1,v,1)){
  Bowling_mat[e,9] = 2
}
###Follow Through Data
length_follow = length(non_zero_follow)
remainder_follow= length_follow%%20
#Follow through Matrices
mean_follow <- matrix(nrow = 500,ncol = 1)
median_follow <- matrix(nrow = 500,ncol = 1)
variance_follow <- matrix(nrow = 500,ncol = 1)
iqr_follow <- matrix(nrow = 500,ncol = 1)
skewness_follow<- matrix(nrow = 500,ncol = 1)
kurtosis_follow <- matrix(nrow = 500,ncol = 1)
rms_follow <- matrix(nrow = 500,ncol = 1)
mad_follow<- matrix(nrow = 500,ncol = 1)
#rms_bowling[1,1] = rms_bowling_20
#Calculate statistical parameters for follow data
for (s in seq(175,(length_follow-remainder_follow),175)){
```

```
  val1_follow=(s-175)
  val2_follow=(s+175)
  mean_follow[w,1] = mean(non_zero_follow[val1_follow:val2_follow],na.rm = TRUE)
  median_follow[w,1] = median(non_zero_follow[val1_follow:val2_follow],na.rm = TRUE)
  variance_follow[w,1] = var(non_zero_follow[val1_follow:val2_follow],na.rm = TRUE)
  iqr_follow[w,1] = IQR(non_zero_follow[val1_follow:val2_follow],na.rm = TRUE)
  skewness_follow[w,1] = skewness(non_zero_follow[val1_follow:val2_follow],na.rm =
TRUE, type = 1)
  kurtosis_follow[w,1] = kurtosis(non_zero_follow[val1_follow:val2_follow],na.rm =
TRUE, type = 3)
  rms_follow[w,1] = rms(non_zero_follow[val1_follow:val2_follow],na.rm = TRUE)
  mad_follow[w,1] = mad(non_zero_follow[val1_follow:val2_follow],na.rm = TRUE)
  w=w+1
}
h = length(na.omit(mean_follow))
#Plot Data follow
Follow_mat <- matrix(nrow = h, ncol = 9)
Follow_mat[,1]= c(na.omit(mean_follow[,1]))
Follow_mat[,2]= c(na.omit(median_follow[,1]))
Follow_mat[,3] = c(na.omit(variance_follow[,1]))
Follow_mat[,4] = c(na.omit(iqr_follow[,1]))
Follow_mat[,5] = c(na.omit(skewness_follow[,1]))
Follow_mat[,6] = c(na.omit(kurtosis_follow[,1]))
Follow_mat[,7] = c(na.omit(rms_follow[,1]))
Follow_mat[,8] = c(na.omit(mad_follow[,1]))
#Bowling_mat[,7] = c(na.omit(rms_bowling[,1]))
for(e in seq(1,h,1)){
  Follow_mat[e,9] = 3
}
##png("legend.png", width = 450, height = 400)
##par(xpd = T, mar = par()$mar + c(0,0,0,7))
plot(Runup_mat[,1],Runup_mat[,3],xlim = c(-0.6,0.3), ylim = c(0,0.4),type = "p", col=
"green", xlab = "Mean", ylab = "Variance", main = "Mean vs Variance")
```

```
par(new=TRUE)

plot(Bowling_mat[,1],Bowling_mat[,3],xlim = c(-0.6,0.3), ylim = c(0,0.4),type = "p", col= "red", xlab = "Mean", ylab = "Variance", main = "Mean vs Variance")

par(new=TRUE)

plot(Follow_mat[,1],Follow_mat[,3],xlim = c(-0.6,0.3), ylim = c(0,0.4), type = "p", col= "blue", xlab = "Mean", ylab = "Variance", main = "Mean vs Variance")

legend(-0.6,0.4,c("Run Up", "Bowling", "Follow Through"), fill = c("green", "red", "blue"), cex = 0.5)

##par(mar=c(5, 4, 4, 2) + 0.1)

###Full Data Matrix

Full_data = matrix(nrow = (h+v+f),ncol = 9)

Full_data[1:f,] = Runup_mat[1:f,]

Full_data[(f+1):(f+v),] = Bowling_mat[1:v,]

Full_data[(f+v+1):(f+v+h),] = Follow_mat[1:h,]

##Feature Scaling

Full_data[,-9]=scale(Full_data[,-9])

"

# Buildig optimal model

## Backward Elimintion

regressor = lm(formula = Full_data[,7] ~ Full_data[,1] +Full_data[,2] + Full_data[,3]+ Full_data[,4] + Full_data[,5] + Full_data[,6],

environment(formula))

summary(regressor)

regressor = lm(formula = Full_data[,7] ~ Full_data[,1] +Full_data[,2] + Full_data[,3],

environment(formula))

summary(regressor)

##regressor = lm(formula = Full_data[,7] ~ Full_data[,1] +Full_data[,2] + Full_data[,4] ,

##              environment(formula))

##summary(regressor)

"

##Scaled Matrix

Scaled_final_matrix = matrix(nrow = (f+v+h), ncol=9)

Scaled_final_matrix[,1] = Full_data[,1]
```

```
Scaled_final_matrix[,2] = Full_data[,2]
Scaled_final_matrix[,3] = Full_data[,3]
Scaled_final_matrix[,4] = Full_data[,4]
Scaled_final_matrix[,5] = Full_data[,5]
Scaled_final_matrix[,6] = Full_data[,6]
Scaled_final_matrix[,7] = Full_data[,7]
Scaled_final_matrix[,8] = Full_data[,8]
Scaled_final_matrix[,9] = Full_data[,9]
##Shuffle Rows
Scaled_final_matrix = Scaled_final_matrix[sample(nrow(Scaled_final_matrix)),]
## ApplyPCA Training data
Data_matri_scale = matrix(nrow = (f+v+h), ncol = 3)
Data_matri_pca = data.frame(matrix((Scaled_final_matrix[,-9]),nrow = (f+v+h), ncol = 9))
Data_matri_pca[,9] = Scaled_final_matrix[,9]
pca = preProcess(x =Data_matri_pca[,-9], method = 'pca', pcaComp = 2)
Data_matri_scale = predict(pca,Data_matri_pca[,-9])
Data_matri_scale[,3] =  Scaled_final_matrix[,9]
plot(Data_matri_scale[,1],Data_matri_scale[,2])# Load data

#### SVM Classifier
accuracy = matrix(nrow = 5, ncol = 1)
precision = matrix(nrow = 5, ncol = 1)
recall =  matrix(nrow = 5, ncol = 1)
f1 = matrix(nrow = 5, ncol = 1)
classification = matrix(nrow = 2000,ncol = 10)
##Splitting into folds
flds <- createFolds(Data_matri_scale[,3], k = 5, list = TRUE, returnTrain = FALSE)
names(flds)[1] <- "train"
for(m in seq(1,5,1)){
  classifier = svm(formula = V3~. ,
                   data = Data_matri_scale[-flds[[m]],],
                   type = 'C-classification',
```

```
                kernel = 'linear')
 ## classification[,m] <- classifier
 # Predicting theTest set results
 y_pred = predict(classifier, newdata = Data_matri_scale[flds[[m]],-3])
 cm = table(y_pred,Data_matri_scale[flds[[m]],3])
 n = sum(cm) # number of instances
 nc = nrow(cm) # number of classes
 diag = diag(cm) # number of correctly classified instances per class
 rowsums = apply(cm, 1, sum) # number of instances per class
 colsums = apply(cm, 2, sum) # number of predictions per class
 p = rowsums / n # distribution of instances over the actual classes
 q = colsums / n # distribution of instances over the predicted classe
# accuracy = sum(diag)/n
 accuracy[m] = sum(diag)/n
 precision[m] = diag / colsums
 recall[m] = diag / rowsums
 f1[m] = 2 * precision[m] * recall[m] / (precision[m] + recall[m])
 data.frame(precision[m], recall[m], f1[m])
}
###Average performance values
accuracy_final = mean(accuracy)
precision_final = mean(precision)
recall_final = mean(recall)
##SD Deviation
SD_Deviation = sd(accuracy)
###F measure
f_measure = 2 * precision_final * recall_final / (precision_final + recall_final)
### Pot Values
Acuracy_max = which.max(accuracy)
#####Print Validation values
cat("Accuracy:",round(accuracy_final,digits = 2))
cat("Accuracy_SD Deviation:",round(SD_Deviation,digits = 2))
```

```
cat("Precision:",round(precision_final, digits = 2))

cat("Recall:",round(recall_final,digits = 2))

cat("F Measure:",round(f_measure,digits = 2))

print(cm)

print(Data_matri_scale[flds[[m]],3])

print(y_pred)

#####SVM

##library(e1071)

#classifier = svm(formula = split_up_training$Fold01[,3] ~ split_up_training$Fold01[,1] +
split_up_training$Fold01[,2],

#              data = split_up_training$Fold01,

#             type = 'C-classification',

#              kernel = 'linear')

# Predicting the Test set results

#y_pred_test = predict(classifier, newdata = split_up_test$Fold01[-3])

# Making the Confusion Matrix

#cm = table(split_up_test$Fold01[, 3], y_pred_test)

# Visualising the Training set results

#install.packages('ElemStatLearn')

training_set = matrix(nrow= nrow(Data_matri_scale[-flds[[Acuracy_max]],]), ncol= 3)

training_set = Data_matri_scale[-flds[[Acuracy_max]],]

test_set = matrix(nrow= nrow(Data_matri_scale[flds[[Acuracy_max]],]), ncol= 3)

test_set = Data_matri_scale[flds[[Acuracy_max]],]

########## SVM Plot

library(ElemStatLearn)

set = training_set

X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)

X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)

grid_set = expand.grid(X1, X2)

colnames(grid_set) = c('PC1', 'PC2')

y_grid = predict( svm(formula = V3~. ,data = training_set,type = 'C-classification',kernel =
'linear'), newdata = grid_set)

par(xpd=NA,oma=c(3,0,0,0))
```

```
plot(set[, -3],
     main = 'SVM (Training set)',
     xlab = 'PC1', ylab = 'PC2',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', ifelse(y_grid == 2,'tomato', 'cornflowerblue')))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', ifelse(set[, 3]==2,'red3','blue3')))
legend(par("usr")[1],par("usr")[3.5],c("Run Up", "Bowling", "Follow Through"), fill = c("green", "red", "blue"), cex = 0.5)



###Test Set
library(ElemStatLearn)
set = test_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('PC1', 'PC2')
y_grid = predict( svm(formula = V3~. ,data = training_set,type = 'C-classification',kernel = 'linear'), newdata = grid_set)
par(xpd=NA,oma=c(3,0,0,0))
plot(set[, -3],
     main = 'K-NN (Test Set)',
     xlab = 'Pc1', ylab = 'PC2',
     xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', ifelse(y_grid == 2,'tomato', 'cornflowerblue')))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', ifelse(set[, 3]==2,'red3','blue3')))
legend(par("usr")[1],par("usr")[3.5],c("Run Up", "Bowling", "Follow Through"), fill = c("green", "red", "blue"), cex = 0.5)
```