ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Εντοπισμός πτώσεων και αναγνώριση δραστηριοτήτων σε περιβάλλον έξυπνου ρολογιού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Πέτσας

Επιβλέπων : Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Εντοπισμός πτώσεων και αναγνώριση δραστηριοτήτων σε περιβάλλον έξυπνου ρολογιού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Κωνσταντίνος Πέτσας

Επιβλέπων : Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 24η Οκτωβρίου 2016.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Παναγιώτης Τσανάκας	Δημήτρης Κουτσούρης	Ηλίας Μαγκλογιάννης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Αναπλ. Καθηγητής Παν. Πειραιώς

Αθήνα, Οκτώβριος 2016

(Υπογραφή)

.....

ΚΩΝΣΤΑΝΤΙΝΟΣ ΠΕΤΣΑΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η σχεδίαση και η κατασκευή συστήματος παρακολούθησης της καθημερινής δραστηριότητας του χρήστη, ο εντοπισμός πτώσεων και η παροχή άμεσου τρόπου ειδοποίησης σε περίπτωση έκτακτης ανάγκης. Απευθύνεται κυρίως σε ηλικιωμένα άτομα στα οποία μια πτώση μπορεί να προκαλέσει σοβαρότατους τραυματισμούς και ενδεχομένως την κατάσταση παραμονής στο έδαφος για μεγάλο χρονικό διάστημα, κάτι το οποίο επιφέρει δραματικές σωματικές και ψυχολογικές συνέπειες. Ιδιαίτερη έμφαση δόθηκε στη χαμηλή κατανάλωση ενέργειας και στην ελαχιστοποίηση χρήσης υπολογιστικών πόρων ώστε να μπορεί να ενσωματωθεί σε συσκευές χαμηλών δυνατοτήρων όπως τα έξυπνα ρολόγια.

Προκειμένου να επιτευχθούν τα παραπάνω, διερευνήθηκαν σε βάθος οι δυνατότητες του έξυπνου ρολογιού Pebble Classic. Δημιουργήθηκε αρχικά εφαρμογή για τη συλλογή δεδομένων από το επιταχυνσιόμετρο του Pebble ώστε να γίνει δυνατή η περαιτέρω ανάλυση. Η αποθήκευση δεδομένων έγινε σε βάση δεδομένων που φιλοξενείται στις cloud υπηρεσίες του "Ωκεανού" μέσω ενός εξυπηρετητή με τον οποίο επικοινωνεί η εφαρμογή στέλνοντας τα δεδομένα. Στη συνέχεια δημιουργήθηκε εφαρμογή αναγνώρισης δραστηριοτήτων που ενσωματώνει και αλγόριθμο για τον εντοπισμό πτώσεων. Ο εξυπηρετητής αναλαμβάνει επιπλέον την αποστολή ειδοποιήσεων μέσω email σε περιπτώσεις έκτακτης ανάγκης όταν λαμβάνει από το ρολόι κατάλληλο μήνυμα και τέλος, φιλοξενεί διαδικτυακή εφαρμογή που καθιστά δυνατή την παρακολούθηση της δραστηριότητας του χρήστη και εμφανίζει χρήσιμα στατιστικά στοιχεία.

Συγκεκριμένα, οι εφαρμογές του Pebble χρησιμοποιούν τις γλώσσες προγραμματισμού C και JavaScript αξιοποιώντας τις διαφορετικές τεχνολογίες που υποστηρίζει το Pebble για εκτέλεση του κώδικα είτε επάνω στο ρολόι είτε σε smartphone με το οποίο συνδέεται μέσω Bluetooth, παρουσιάζονται τα πλεονεκτήματα και μειονεκτήματα της κάθε μιας και προτείνεται συγκεκριμένη μέθοδος ως καταλληλότερη. Ο εξυπηρετητής είναι υλοποιημένος σε γλώσσα Java στο πλαίσιο Spring και η βάση δεδομένων που χρησιμοποιήθηκε υιοθετεί το εγγραφοκεντρικό μοντέλο Mongo. Η διαδικτυακή εφαρμογή δέχεται σαν είσοδο από το χρήστη την ημερομηνία και αναλαμβάνει μέσω απλών HTTP Requests να τραβήξει από τη βάση δεδομένων τα σχετικά δεδομένα και να τα αναπαραστήσει γραφικά. Τέλος, για την αποστολή email χρησιμοποιείται η γενικής χρήσης λειτουργία δρομολόγησης διαδικτυακών μηνυμάτων sendmail.

Λέξεις Κλειδιά: Εφαρμογή, έξυπνο ρολόι, αισθητήρες, επιταχυνσιόμετρο, C, Javascript, Bluetooth, Mongo, αναγνώριση δραστηριοτήτων, εντοπισμός πτώσης.

Abstract

The scope of this thesis is the design and development of a platform able to monitor user's every day activity, detect falls and send alerts in case of emergency situations. It is intended mainly for elder people who are more susceptible to falls. Moreover, a fall can cause severe injuries to them. Additionaly the long lying state which can occur after a fall has dramatic physical and psychological concenquences. We empashized on low energy consumption and minimum use of computational sources so that the system can be embeded on low capability devices such as smartwatches.

In order to accomplish the above we explored deeply the capabilities of the Pebble smartwatch. At first we created an application in order to collect and store data from Pebble's accelerometer. For the data storage we used a data base on a server which resides on Okeanos' cloud services. Then we created an application which allows us to track user's activity and detect falls. The server is also responsible for sending notifications via email in case of emergency. Finally there is the possibility to use a web application to watch user's activity graphically and also learn several useful statistics.

Pebble apps use C and JavaScript programming languages allowing us to execute code on the smartwatch and the paired smartphone respectively. We explored both techniques as well as the pros and cons of each one of them and we suggested specific method as more appropriate for the needs of our system. The server was developed in Java programming language using Spring framework while the data base uses the Mongo JSON-like documents model. The web application receives a date as input and uses simple HTTP Requests to fetch the requested data from the data base and depict them graphically. For the email notifications we used sendmail which is a general purpose internetwork email routing facility.

Keywords: Application, smartwatch, sensors, accelerometer, C, Javascript, Bluetooth, Mongo, activity tracking, fall detection.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Τσανάκα για την εμπιστοσύνη που έδειξε στο πρόσωπό μου και για την καθοδήγησή του. Επίσης, τον μεταδιδακτορικό ερευνητή του Ε.Μ.Π κ. Μενύχτα, για την πολύτιμη βοήθειά του στα πρώτα στάδια της διπλωματικής. Ευχαριστώ επίσης την οικογένειά μου και την κοπέλα μου Εύη για τη συμπαράσταση και την υπομονή τους καθ΄ όλη τη διάρκεια φοίτησής μου στο Ε.Μ.Π.

Περιεχόμενα

Σχήματα

Πίνακες

Κεφάλαιο 1. Εισαγωγή

1.1 Σκοπός διπλωματικής

Οι πτώσεις, παρά τις εκτεταμένες προσπάθειες που γίνονται για την πρόληψή τους, αποτελούν σημαντική αιτία θνησιμότητας ηλικιωμένων ατόμων, ενώ πολλές φορές οδηγούν σε σοβαρούς τραυματισμούς που καθηλώνουν τους ασθενείς για μεγάλα χρονικά διαστήματα. Ακόμα και σε περιπτώσεις μη σοβαρού τραυματισμού, ο φόβος της πτώσης μπορεί να προκαλέσει στο ηλικιωμένο άτομο περιορισμό της κινητικότητάς του, απώλεια προσωπικής του αυτονομίας και μείωση της ποιότητας ζωής. Πολύ σοβαρές σωματικές και ψυχολογικές επιπτώσεις μπορεί να έχει επίσης η παραμονή για πολλές ώρες στο έδαφος, μετά από πτώση, λόγω αδυναμίας του ατόμου να σηκωθεί ή απώλειας των αισθήσεων. Το 47% των ηλικιωμένων δεν καταφέρνει να σηκωθεί χωρίς κάποια υποστήριξη ακόμα και σε περίπτωση μη τραυματισμού [26] ενώ σε περιπτώσεις μακροχρόνιας παραμονής στο έδαφος το 50% των ηλικιωμένων που βιώνουν τέτοια κατάσταση πεθαίνει μέσα σε διάστημα 6 μηνών [27]. Ο εντοπισμός πτώσεων σε πραγματικό χρόνο επιτρέπει την άμεση ειδοποίηση κάποιου κοντινού προσώπου ή κάποιου κέντρου παροχής φροντίδας ώστε να παρασχεθούν άμεσα οι πρώτες βοήθειες. Κάτι τέτοιο μπορεί να δώσει μια αίσθηση ασφάλειας στο ηλικιωμένο άτομο και ιδιαίτερα σε όσα μένουν μόνα τους και να μειώσει σε σημαντικό βαθμό τις αρνητικές συνέπειες μια πτώσης ή της κατάστασης παραμονής στο έδαφος.

Την τελευταία δεκαετία έχουν αναπτυχθεί πολλές μέθοδοι σχετικά με τον αυτόματο εντοπισμό πτώσεων που στηρίζονται κυρίως σε συσκευές βιντεοσκόπησης τοποθετημένες στον χώρο που ζει το ηλικιωμένο άτομο, αισθητήρες ήχου, αισθητήρες επιτάχυνσης τοποθετημένους σε διάφορα μέρη του σώματος (π.χ. στη μέση) και κινητά τηλέφωνα. Κάθε μια από αυτές τις μεθόδους έχει διαφορετικά πλεονεκτήματα και μειονεκτήματα. Σημαντικοί παράγοντες που λαμβάνονται πάντα υπόψιν είναι το κόστος των συσκευών, το μέγεθος, η αυτονομία, η ευκολία στη χρήση και η φορητότητα. Όσον αφορά τον αλγόριθμο που χρησιμοποιείται μπορούμε να διακρίνουμε δύο βασικές κατηγορίες. Αυτές που στηρίζονται σε αλγορίθμους μηχανικής μάθησης και αυτές που στηρίζονται σε αλγορίθμους με thresholds. Οι πρώτοι είναι συνήθως πιο πολύπλοκοι καθώς η μηχανική μάθηση απαιτεί τη συλλογή μεγάλου όγκου δεδομένων και σωστή επιλογή αντιπροσωπευτικών χαρακτηριστικών. Συνήθως συνδυάζουν την αναγνώριση δραστηριοτήτων την οποία χρησιμοποιούν για να διαχωρίσουν πτώσεις από καθημερινές δραστηριότητες. Οι δεύτεροι είναι σχετικά απλούστεροι στην υλοποίηση. Και στις δύο περιπτώσεις όμως η δυσκολία που υπάρχει είναι η απουσία δεδομένων στο διαδίκτυο από πτώσεις που συμβαίνουν σε πραγματικές συνθήκες, επομένως όλες οι μελέτες έχουν γίνει βασισμένες σε προσομοιώσεις. Επιπλέον το γεγονός ότι οι προσομοιώσεις πραγματοποιούνται απο υγιή νεαρά άτομα και όχι από ηλικιωμένους για ευνόητους λόγους έχει σαν αποτέλεσμα τη δυσκολία δημιουργίας αξιόπιστου αλγόριθμου.

Στην παρούσα διπλωματική έχει επιλεχθεί το έξυπνο ρολόι Pebble classic το οποίο πληροί τις προϋποθέσεις που αναφέρθηκαν νωρίτερα σαν συσκευή. Η τιμή του σήμερα είναι χαμηλότερη των 100 ευρώ, παρουσιάζει αυτονομία μπαταρίας διάρκειας 7 ημερών και έχει το σημαντικό πλεονέκτημα ότι ο χρήστης το φοράει επάνω του συνεχώς σαν ένα απλό ρολόι χωρίς να νιώθει περιορισμένος από αυτό. Αξιοποιώντας τις δυνατότητές του, δημιουργήθηκε ένα σύστημα εντοπισμού πτώσεων το οποίο συνδυάζει και την παρακολούθηση της δραστηριότητας του χρήστη βασιζόμενο στις μετρήσεις που προέρχονται από το επιταχυνσιόμετρο που διαθέτει. Για την υλοποίηση του παραπάνω συστήματος αξιοποιήθηκαν αποτελέσματα δημοσιεύσεων και ερευνών που έχουν γίνει σε αυτόν τον τομέα.

Για τον εντοπισμό πτώσεων έχει γίνει μελέτη και έχουν τεθεί βάσεις για μελλοντικές εργασίες και

προς τις δύο κατευθύνσεις (αλγόριθμοι thresholds και αλγόριθμοι μηχανικής μάθησης) ενώ τελικά επιλέχθηκε η υλοποίηση αλγορίθμου με thresholds έχοντας σαν βασικό γνώμονα την απλότητα και τη χαμηλή κατανάλωση μπαταρίας. Για την αναγνώριση δραστηριοτήτων έχει υλοποιηθεί αλγόριθμος που προέρχεται από την ανάλυση της μηχανικής μάθησης αλλά και ένας δεύτερος που χαρακτηρίζεται από ιδιαίτερη απλότητα λόγω της οποίας επιλέχθηκε να ενσωματωθεί στην τελική εφαρμογή. Τέλος παρέχεται η δυνατότητα παρακολούθησης της καθημερινής δραστηριότητας του χρήστη μέσω διαδικτυακής εφαρμογής το οποίο αφήνει περιθώρια για περαιτέρω μελέτη και συσχέτιση των στατιστικών στοιχείων που προκύπτουν ανα ημέρα με την φυσική κατάσταση και την υγεία του χρήστη ελέγχοντας για τυχόν παρεκκλίσεις από τη συνήθη δραστηριότητά του.

1.2 Οργάνωση κειμένου

Έπειτα από αυτή τη σύντομη εισαγωγή η διπλωματική ακολουθεί την εξής οργάνωση:

Στο **Κεφάλαιο 2** παρουσιάζονται οι σημαντικότερες έννοιες, θεωρητικές και μη, η κατανόηση των οποίων θα βοηθήσει τον αναγνώστη να κατανοήσει καλύτερα τις τεχνικές που χρησιμοποιούνται και να έχει μια πιο σφαιρική εικόνα για την πορεία που ακολουθείται.

Στο **Κεφάλαιο 3** παρουσιάζεται η γενική πορεία, τα διάφορα συστήματα που χρησιμοποιούνται καθώς και η μεταξύ τους επικοινωνία.

Στο **Κεφάλαιο 4** αναλύονται λεπτομέρειες υλοποίησης και γίνεται επεξήγηση των σημαντικότερων σημείων του κώδικα.

Το **Κεφάλαιο 5** περιλαμβάνει τα αποτελέσματα της διπλωματικής σε μορφή γραφικών παραστάσεων και στατιστικών στοιχείων.

Στο **Κεφάλαιο 6** καταγράφονται τα συμπεράσματα και πιθανές επεκτάσεις της διπλωματικής.

Κεφάλαιο 2. Υπόβαθρο – Βασικές έννοιες

2.1 Θεωρητικά

2.1.1 Smartwatch

Τα έξυπνα ρολόγια είναι ψηφιακά ρολόγια χειρός των οποίων οι δυνατότητες δεν περιορίζονται απλά στην εμφάνιση ώρας. Ενώ τα πρώτα μοντέλα είχαν πολύ λίγες δυνατότητες, τα σύγχρονα αποτελούν ουσιαστικά υπολογιστές χειρός καθώς ενσωματώνουν λειτουργικά συστήματα φορητών συσκευών (Mobile Operating Systems) και μπορούν να τρέξουν διάφορες εφαρμογές. Τα γνωστότερα λειτουργικά συστήματα που τρέχουν σε έξυπνα ρολόγια είναι το Android Wear, Watch OS, Pebble OS, WebOs και Tizen for Wearables. Τα περισσότερα είναι σχεδιασμένα με τέτοιο τρόπο ώστε να επιτρέπουν τη σύζευξη μέσω bluetooth με κάποιο smartphone ή tablet. Αυτό επιτρέπει την ανταλλαγή μηνυμάτων μεταξύ των δύο συσκευών κι έτσι δίνεται η δυνατότητα στα έξυπνα ρολόγια να λειτουργούν σαν φορητές συσκευές αναπαραγωγής πολυμέσων, ραδιοφώνου, επιτρέπουν στο χρήστη να απαντήσει σε τηλεφωνικές κλήσεις αλλά και σε μηνύματα. Οι εφαρμογές που τρέχουν στα έξυπνα ρολόγια, ανάλογα με το λειτουργικό σύστημα, μπορεί να απαιτούν σύνδεση με κάποια host εφαρμογή που τρέχει στη συσκευή με την οποία έχει γίνει η ζεύξη ή να είναι αυτοδύναμες (standalone).

Από πλευράς υλικού (hardware) υπάρχει μεγάλη ποικιλία. Τα περισσότερα διαθέτουν επαναφορτιζόμενη μπαταρία η οποία μπορεί να διαρκέσει από μερικές ώρες έως και μια ολόκληρη εβδομάδα με μια φόρτιση. Η οθόνη μπορεί να είναι είτε τετράγωνη είτε κυκλική, έγχρωμη, ασπρόμαυρη ενώ κάποια διαθέτουν ακόμα και οθόνη αφής. Τα έξυπνα ρολόγια επίσης ενσωματώνουν διαφόρων ειδών αισθητήρες όπως θερμόμετρο, επιταχυνσιόμετρο, αλτίμετρο, βαρόμετρο, πυξίδα, γυροσκόπιο, GPS και μετρητή καρδιακών παλμών. Κάποια διαθέτουν επίσης ενσωματωμένη κάμερα και μικρόφωνο.

Από άποψη λογισμικού (software), συνήθεις προεγκατεστημένες εφαρμογές περιλαμβάνουν ημερολόγιο, αριθμομηχανή, χρονόμετρο, ξυπνητήρι και τα λεγόμενα watchfaces που αλλάζουν τον τρόπο εμφάνισης της ώρας. Ιδιαίτερα διαδεδομένες στα έξυπνα ρολόγια είναι οι εφαρμογές παρακολούθησης φυσικής κατάστασης (fitness tracking) που είναι διαθέσιμες για κατέβασμα από τα online stores.

2.1.2 Activities of daily living (ADL)

Activities of daily living [4] ή αλλιώς ADL είναι οι βασικές δραστηριότητες στις οποίες επιδίδεται ο άνθρωπος στην καθημερινή του ζωή. Αποτελούν δείκτη της λειτουργικής ικανότητας του ανθρώπινου σώματος, ιδιαίτερα σε άτομα που έχουν υποστεί τραυματισμούς ή είναι ηλικιωμένα με ενδεχομένως δυσκολίες στην κίνηση. Βασικές δραστηριότητες που περιλαμβάνονται είναι η λειτουργική κίνηση (περπάτημα, ικανότητα να σηκώνεται κάποιος από την καρέκλα ή το κρεβάτι, ικανότητα να μετακινείται αυτόνομα από ένα μέρος σε κάποιο άλλο), η τήρηση της προσωπικής υγιεινής, το ντύσιμο, ικανότητα να τρέφεται κλπ. Η έννοια των ADL επίσης διευρύνεται περιέχοντας πολλές φορές και άλλες μη βασικές δραστηριότητες όπως οι δουλειές του σπιτιού, ετοιμασία φαγητού, διαχείρηση χρημάτων. Γενικά στις ADL μπορούν να περιλαμβάνονται διαφορετικές δραστηριότητες ανάλογα με τη φύση της εφαρμογής και τα προβλήματα που αντιμετωπίζουν τα άτομα τα οποία αφορούν.

2.1.3 Μηχανική μάθηση

Η μηχανική μάθηση (Machine learning) [5] αποτελεί υποπεδίο της επιστήμης υπολογιστών που αναπτύχθηκε από τη μελέτη της αναγνώρισης προτύπων και της υπολογιστικής θεωρίας μάθησης στην τεχνητή νοημοσύνη. Διερευνά τη μελέτη και την κατασκευή αλγορίθμων που μπορούν να μαθαίνουν από τα δεδομένα και να κάνουν προβλέψεις σχετικά με αυτά. Τέτοιοι αλγόριθμοι λειτουργούν κατασκευάζοντας μοντέλα από πειραματικά δεδομένα, προκειμένου να κάνουν προβλέψεις ή να εξάγουν αποφάσεις. Η μηχανική μάθηση εφαρμόζεται σε μια σειρά από υπολογιστικές εργασίες που τόσο ο σχεδιασμός όσο και ο ρητός προγραμματισμός αλγορίθμων είναι ανέφικτος όπως για παράδειγμα τα φίλτρα spam, η οπτική αναγνώριση χαρακτήρων και οι μηχανές αναζήτησης.

Οι εργασίες μηχανικής μάθησης ταξινομούνται σε τρεις μεγάλες κατηγορίες:

Επιτηρούμενη μάθηση (supervised learning):

Σε αυτή το υπολογιστικό πρόγραμμα δέχεται κάποιες παραδειγματικές εισόδους καθώς και τα επιθυμητά αποτελέσματα από έναν "δάσκαλο" και ο στόχος είναι να μάθει έναν γενικό κανόνα προκειμένου να αντιστοιχίσει τις εισόδους με τα αποτελέσματα.

Μη επιτηρούμενη μάθηση (unsupervised learning):

Εδώ δεν παρέχεται κάποια εμπειρία στον αλγόριθμο μάθησης, ο οποίος πρέπει να εντοπίσει από μόνος του κάποια δομή στα δεδομένα εισόδου.

Ενισχυτική μάθηση:

Ένα πρόγραμμα υπολογιστή αλληλεπιδρά με ένα δυναμικό περιβάλλον στο οποίο πρέπει να επιτευχθεί ένας συγκεκριμένος στόχος, χωρίς κάποιος δάσκαλος να του λέει ρητά αν έχει φτάσει κοντά στο στόχο του.

Κυριότερες προσεγγίσεις:

Εκμάθηση με δέντρο απόφασης:

Η εκμάθηση με δέντρο απόφασης χρησιμοποιεί ένα δέντρο απόφασης ως προγνωστικό μοντέλο, το οποίο αντιστοιχίζει παρατηρήσεις σχετικά με ένα στοιχείο σε συμπεράσματα σχετικά με την τιμή στόχο του αντικειμένου.

Εκμάθηση με κανόνες συσχέτισης:

Η εκμάθηση με κανόνες συσχέτισης είναι μια μέθοδος ανακάλυψης ενδιαφέρων σχέσεων μεταξύ των μεταβλητών σε μεγάλες βάσεις δεδομένων.

Μηχανές διανυσμάτων υποστήριξης:

Οι μηχανές διανυσμάτων υποστήριξης είναι ένα σύνολο μεθόδων επιτηρούμενης μάθησης που χρησιμοποιούνται για την ταξινόμηση και την παλινδρόμηση. Σ' αυτήν την περίπτωση δίνεται ένα σύνολο παραδειγμάτων εκπαίδευσης και κάθε φορά δηλώνεται σε ποια από τις δύο κατηγορίες

ανήκει το παράδειγμα. Μία μηχανή διανυσμάτων υποστήριξης κατασκεάζει ένα μοντέλο που προβλέπει αν το νέο παράδειγμα εμπίπτει στην μία κατηγορία ή την άλλη.

Ομαδοποίηση:

Η ομαδοποίηση είναι η διαδικασία κατά την οποία ένα σύνολο παρατηρήσεων χωρίζεται σε υποσύνολα έτσι ώστε οι παρατηρήσεις που ανήκουν στην ίδια ομάδα (cluster) είναι όμοιες, σύμφωνα με κάποιο ή κάποια προκαθορισμένα κριτήρια, ενώ οι παρατηρήσεις που προέρχονται από διαφορετικά υποσύνολα είναι ανόμοιες. Διαφορετικές τεχνικές κατηγοριοποίησης οδηγούν σε διαφορετικές υποθέσεις σχετικά με τη δομή των δεδομένων, οι οποίες συχνά καθορίζονται από κάποιο μέτρο ομοιότητας και αξιολογούνται για παράδειγμα ως προς την εσωτερική συνοχή (ομοιότητα μεταξύ των μελών του ίδιου cluster) και το διαχωρισμό ανάμεσα σε διαφορετικές ομάδες. Άλλες μέθοδοι βασίζονται στην εκτιμώμενη πυκνότητα και την συνεκτικότητα των γραφημάτων. Η ομαδοποίηση είναι μία μέθοδος μη επιτηρούμενης μάθησης και μία τεχνική η οποία χρησιμοποιείται επίσης στην στατιστική ανάλυση δεδομένων.

Δίκτυα Bayes:

Ένα δίκτυο Bayes, ένα δίκτυο εμπιστοσύνης ή ένα άκυκλο γραφικό μοντέλο είναι ένα πιθανοθεωρητικό γραφικό μοντέλο που απεικονίζει ένα σύνολο τυχαίων μεταβλητών και την μεταξύ τους υποθετική ανεξαρτησία διαμέσου ενός κατευθυνόμενου άκυκλου γράφου . Για παράδειγμα, ένα δίκτυο Bayes μπορεί να αναπαραστήσει την πιθανοθεωρητική σχέση μεταξύ ασθενειών και συμπτωμάτων. Δεδομένων των συμπτωμάτων, το δίκτυο μπορεί να χρησιμοποιηθεί για να υπολογίσει τις πιθανότητες παρουσίας διαφόρων ασθενειών.

2.1.3.1 Feature extraction

Στη μηχανική μάθηση, την αναγνώριση προτύπων και την επεξεργασία εικόνας χρησιμοποιείται η εξαγωγή χαρακτηριστικών [6] πάνω σε ένα αρχικό σύνολο δεδομένων που έχει μετρηθεί και δίνει παράγωγες τιμές (features) οι οποίες διευκολύνουν περαιτέρω την εκμάθηση και σε κάποιες περιπτώσεις βοηθούν στην καλύτερη κατανόηση των δεδομένων από τον άνθρωπο. Τα αρχικά δεδομένα μπορούν να χαρακτηριστούν ως πολλαπλών διαστάσεων και ουσιαστικά η εξαγωγή χαρακτηριστικών επιτρέπει να μεταβούμε σε έναν χώρο λιγότερων διαστάσεων.

2.1.3.2 Feature selection

Η επιλογή χαρακτηριστικών επιτρέπει την εύρεση ενός υποσυνόλου των χαρακτηριστικών που εξήχθησαν, τα οποία είναι πιο κατάλληλα και πιο ακριβή για την εξαγωγή συμπερασμάτων.

2.2 Τεχνολογίες

2.2.1 Pebble classic

Για την παρούσα διπλωματική, όπως έχει αναφερθεί ήδη, επιλέχθηκε το έξυπνο ρολόι Pebble

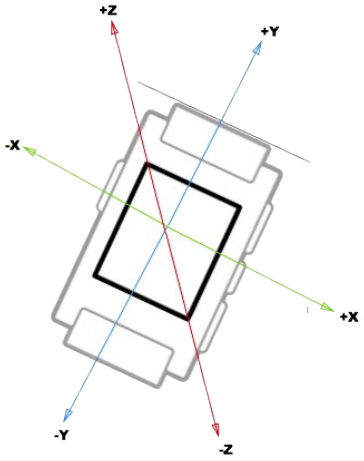
Classic [7]. Το Pebble Classic τρέχει το λειτουργικό σύστημα Pebble OS το οποίο έχει αναπτυχθεί από την Pebble Technology Corporation και αποτελεί τροποποιημένη έκδοση του FreeRTOS - ένα λειτουργικό σύστημα για ενσωματωμένα συστήματα. Το Pebble OS επιτρέπει τη ζεύξη του ρολογιού μέσω bluetooth με iOS και Android smartphones και tablets.

Σχήμα 1. Pebble Classic smartwatch.

2.2.1.1 Hardware

Το Pebble διαθέτει 4 κουμπιά (back, up, select, down) (σχήμα 1). Το back χρησιμοποιείται για να μεταφέρει το χρήστη ένα παράθυρο προς τα πίσω έως ότου φτάσει στο watchface. Τα κουμπιά up και down χρησιμοποιούνται για περιήγηση του χρήστη στα μενού και το select για επιλογή κάποιου στοιχείου από κάποια λίστα, για άνοιγμα εφαρμογής κλπ. Ο **επεξεργαστής** είναι της σειράς STM32 F2. Συγκεκριμένα χρησιμοποιεί τον STM32F205RE6 που βασίζεται στον πυρίνα ARM Cortex-M3 με μέγιστη συχνότητα ρολογιού τα 80 MHz και 512 KB μνήμης επί ψηφίδας (on-chip memory). Διαθέτει επίσης 128 KB **μνήμης RAM** από τα οποία 84 KB χρησιμοποιούνται από το σύστημα, 12 KB από διεργασίες παρασκηνίου και 32 KB από την εφαρμογή που εκτελείται την εκάστοτε στιγμή. Αξίζει να σημειωθεί ότι από τα 32 KB μόνο τα 24 KB είναι διαθέσιμα απευθείας στον προγραμματιστή. Η **οθόνη** είναι ασπρόμαυρη LCD, χαμηλής κατανάλωσης ενέργειας, με ανάλυση 144x168 pixels και διαθέτει φωτισμό backlight. Το ρολόι επίσης διαθέτει **αισθητήρα φωτός** του περιβάλλοντος (ambient light sensor), **μαγνητόμετρο** και **επιταχυνσιόμετρο** 3 αξόνων το οποίο θα παρουσιαστεί αναλυτικά αμέσως μετά λόγω της μεγάλης σημασίας του. Η σύνδεση με συσκευές Android και iOS γίνεται μέσω Bluetooth 2.1 και Bluetooth 4.0 (Bluetooth Low Energy) το οποίο υποστηρίζεται πλέον έπειτα από ενημέρωση firmware. Η **μπαταρία** είναι λιθίου, 3.7 Volts, 130 mAh και διαρκεί περίπου 7 ημερες, με μια φόρτιση, υπό φυσιολογική χρήση. Η φόρτιση γίνεται μέσω τροποποιημένου καλωδίου USB το οποίο προσκολάται μαγνητικά στο ρολόι. Το μεγαλύτερο μέρος της μπαταρίας καταναλώνεται από τον επεξεργαστή με το Bluetooth να έρχεται δεύτερο, ενώ η οθόνη καταναλώνει ελάχιστη ενέργεια συγκρητικά με τα προηγούμενα.

Το **επιταχυνσιόμετρο** [8][9] είναι ένα μικρο-ηλεκτρομηχανικό σύστημα (MEMS). Η λειτουργία του στηρίζεται σε έναν πυκνωτή του οποίου η χωρητικότητα μεταβάλλεται ανάλογα με τις μετακινήσεις ενός μικροσκοπικού βαριδίου. Η μετακίνηση του βαριδίου οφείλεται σε κίνηση του συστήματος, σε δονήσεις αλλά και στη βαρύτητα. Ο πυκνωτής πλαισιώνεται απο κύκλωμα που εντοπίζει τις μεταβολές της χωρητικότητας και τις μετατρέπει σε τάσεις από τις οποίες στη συνέχεια προκύπτουν οι τιμές των επιταχύνσεων. Υπάρχουν επιταχυνσιόμετρα που μετρούν την επιτάχυνση σε έναν, δύο ή τρείς ορθογώνιους άξονες. Το Pebble διαθέτει επιταχυνσιόμετρο τριών αξόνων [10] και επιτρέπει τη συλλογή δεδομένων που αφορούν τις επιταχύνσεις αλλά και τον προσανατολισμό του ρολογιού. Η διάταξη των αξόνων είναι όπως φαίνεται στην παρακάτω εικόνα (σχήμα 2). Οι δυνατές τιμές που μπορούν να μετρηθούν σε κάθε άξονα είναι από -4000 milli-G έως 4000 milli-G ενώ όταν η συσκευή είναι ακίνητη το διανυσματικό άθροισμα των επιταχύνσεων των τριών αξόνων δείχνει 1000 λόγω της επιτάχυνσης της βαρύτητας. Το API επιτρέπει την ανάγνωση των τιμων των επιταχύνσεων σε δομές τύπου AccelData. Η δομή AccelData δίνει έναν πίνακα από 4 μεταβλητές x, y, z, και timestamp που αντιστοιχούν στις τιμές των επιταχύνσεων στους τρεις αντίστοιχους άξονες την αντίστοιχη χρονική στιγμή. Δίνεται η δυνατότητα επιλογής του επιθυμητού μεγέθους αυτού του πίνακα από 1 έως 25, επιτρέποντας έτσι το διάβασμα των επιταχύνσεων σε πακέτα που περιέχουν έως 25 τετράδες τιμών x, y, z και timestamp. Επιπλέον υπάρχει η δυνατότητα ορισμού του χρονικού διαστήματος που θα παρεμβάλεται μεταξύ της κάθε μέτρησης με δεδομένες 4 επιλογές, 10 HZ, 25 HZ, 50 Hz και 100 Hz. Τα προαναφερθέντα πακέτα γίνονται διαθέσιμα μέσω του accel_data_handler η οποία είναι συνάρτηση (callback) που καλείται από το σύστημα κάθε φορά που συμπληρώνεται ο πίνακας της δομής AccelData.



Σχήμα 2. Οι άξονες του επιταχυνσιομέτρου του Pebble.

2.2.1.2 SDK

Το Pebble SDK είναι διαθέσιμο για κατέβασμα από την επίσημη ιστοσελίδα του Pebble και είναι συμβατό με Mac OSX και Linux αλλά όχι με Windows. Η εναλλακτική επιλογή για χρήστες Windows είναι το CloudPebble το οποίο και χρησιμοποιήθηκε. Το CloudPebble [11] είναι ένα online IDE (Integrated Development Environment) που επιτρέπει την άμεση δημιουργία εφαρμογών. Παρέχει έτοιμα templates για δημιουργία watchfaces και watchapps, αυτόματη συμπλήρωση κώδικα, compilation, άμεση εγκατάσταση των εφαρμογών στο Pebble ή σε Emulator ενώ επιτρέπει και την εισαγωγή του κώδικα στο/από το GitHub. Για την εγκατάσταση εφαρμογών

στο Pebble είναι απαραίτητο το κατέβασμα του Pebble app στο smartphone το οποίο λειτουργεί ως companion app και αναλαμβάνει την επικοινωνία με το CloudPebble, την αλλαγή watchfaces, την εγκατάσταση εφαρμογών, την επιλογή notification που ο χρήστης θέλει να εμφανίζονται στο ρολόι κ.ά. Για τη δημιουργία εφαρμογών είναι απαραίτητη η γνώση των γλωσσών προγραμματισμού C και Javascript. Η επιλογή της γλώσσας εξαρτάται από τη φύση της εφαρμογής που θέλουμε να φτιάξουμε. Είναι δυνατόν μια εφαρμογή να είναι γραμμένη εξ ολοκλήρου σε Javascript, όμως ενδεχομένως να χάνουμε σε αποδοτικότητα και ταχύτητα καθιστώντας την χρήση της C καλύτερη επιλογή, ενώ κάποιες φορές ενδείκνυται η συνδυαστική χρήση και των δύο γλωσσών.

2.2.1.3 Pebble.js

Το Pebble.js είναι ένα API (Application Programming Interface) που επιτρέπει τη δημιουργία εφαρμογών για το Pebble αποκλειστικά με χρήση της γλώσσας JavaScript. Εφαρμογές αυτού του είδους τρέχουν στο smartphone και παρέχουν πρόσβαση σε όλους τους διαθέσιμους πόρους του, όπως σύνδεση στο διαδίκτυο, GPS, απεριόριστη μνήμη κλπ. Επειδή είναι γραμμένες σε JavaScript είναι ιδανικές για αιτήματα HTTP και σύνδεση έμμεσα του Pebble στο διαδίκτυο όμως έχουν ένα μειονέκτημα, ότι εξαντλούν γρηγορότερα τη μπαταρία και παρουσιάζουν πιο αργή απόκριση στην αλληλεπίδραση με το χρήστη. Αυτό οφείλεται στο ότι χρησιμοποιείται συνεχώς το Bluetooth κάθε φορά που χρειάζεται να γίνει επικοινωνία με το ρολόι για οποιοδήποτε λόγο (πάτημα κάποιου κουμπιού από το χρήστη, ενημέρωση περιεχομένων οθόνης κλπ).

2.2.1.4 Pebble C και PebbleKit JS

Pebble C:

Είναι το API της C για το Pebble [12]. Επιτρέπει τη δημιουργία εφαρμογών σε γλώσσα C, ο κώδικας των οποίων εκτελείται πάνω στο ρολόι. Υποστηρίζει κάποια Modules της Standard C που αφορούν το Formatting, συναρτήσεις Locale, Math, Memory, Time και διαχείρηση Strings. Περιέχει επίσης μεγάλο πλήθος άλλων συναρτήσεων και εργαλείων για τη διαχείρηση του User Interface, των γραφικών και άλλες βασικές αρχές που είναι απαραίτητες για τη δημιουργία κάθε είδους εφαρμογής. Περισσότερα για το Pebble C API στον σύνδεσμο. Ακολουθούν κάποια θεμελιώδη στοιχεία για την δημιουργία εφαρμογής σε Pebble C η γνώση των οποίων θα φανεί χρήσιμη στη συνέχεια στην κατανόηση της λειτουργίας του κώδικα των εφαρμογών που υλοποιήθηκαν.

Event loop:

Το API παρέχει ένα βρόχο γεγονότων (event loop) μέσα στον οποίο γίνονται κάθε είδους αλληλεπιδράσεις μεταξύ της εφαρμογής μας και του Pebble OS. Η είσοδος στο βρόχο αυτό γίνεται μέσω της συνάρτησης void app_event_loop(void) η οποία τοποθετείται πάντα εντός της συνάρτησης main() της εφαρμογής μας. Πριν από την κλήση της app_event_loop(), πρέπει να κάνουμε εγγραφή (subscribe) σε υπηρεσίες γεγονότων (event services) και να υλοποιήσουμε τις συναρτήσεις (event handlers) που θα διαχειριστούν τα αντίστοιχα γεγονότα. Όταν κληθεί η app_event_loop(), η εκτέλεση της main διακόπτεται προσωρινά και οποιαδήποτε επιπλέον δραστηριότητα εκτελείται από τον κώδικα που είναι γραμμένος στους event handlers, όταν προκύπτουν τα αντίστοιχα γεγονότα. Η έξοδος από το βρόχο γίνεται όταν πατηθεί το κουμπί back

στο Pebble από το τελευταίο παράθυρο της στοίβας της εφαρμογής. Ο έλεγχος επανέρχεται στη συνάρτηση main() όπου πλέον πρέπει να γίνεται απεγγραφή (unsubscribe) από τις υπηρεσίες που χρησιμοποιήθηκαν πρωτού γίνει έξοδος από την εφαρμογή. Τα παραπάνω συνοψίζουν τη δομή του κώδικα οποιασδήποτε εφαρμογής του Pebble σε C.

```
static void init() {
	// Initialization code here
}

static void deinit() {
	// Deinitialization code here
}

int main(void) {
	init();
	app_event_loop();
	deinit();
}
```

Τα Event Services [13] που είναι διαθέσιμα στο Pebble μαζί με τους αντίστοιχους handlers και την επίσημη περιγραφή φαίνονται στον πίνακα που ακολουθεί:

Event Service	**Handler(s)**	**Description**
TickTimerService	TickHandler	Most useful for watchfaces. Allows apps to be notified when a second, minute, hour, day, month or year ticks by.
ConnectionService	ConnectionHandler	Allows apps to know when the Bluetooth connection with the phone connects and disconnects.
AccelerometerService	AccelDataHandler AccelTapHandler	Allows apps to receive raw data or tap events from the onboard accelerometer.
BatteryStateService	BatteryStateHandler	Allows apps to read the state of the battery, as well as whether the watch is plugged in and charging.
HealthService	HealthEventHandler	Allows apps to be notified to changes in various HealthMetric values as the user performs physical activities.
AppFocusService	AppFocusHandler	Allows apps to know when they are obscured by another window, such as when a notification modal appears.
CompassService	CompassHeadingHandler	Allows apps to read a compass

		heading, including calibration status of the sensor.

Πίνακας 1: Event services και handlers.

Callbacks:

Στο Pebble SDK, όσον αφορά το κομμάτι γραμμένο στη γλώσσα C, γίνεται εκτενής χρήση των Callbacks ή αλλιώς Handlers. Callback είναι μια συνάρτηση που παρέχεται στον προγραμματιστή η οποία καλείται από το σύστημα αργότερα, την κατάλληλη χρονική στιγμή. Για παράδειγμα υπάρχει τέτοια συνάρτηση που ενημερώνει την ώρα στην οθόνη και καλείται από το σύστημα κάθε λεπτό. Η συνάρτηση αυτή παρέχεται στον προγραμματιστή ο οποίος μπορεί σε αυτή να γράψει τον κώδικα που επιθυμεί να τρέχει κάθε φορά που ενημερώνεται η ώρα. Με αυτό τον τρόπο δίνεται η δυνατότητα να προστεθούν εύκολα δυναμικά χαρακτηριστικά στις εφαρμογές του Pebble χωρίς να χρειάζεται ο προγραμματιστής να ασχοληθεί με τον ακριβή χρονοπρογραμματισμό κάθε λειτουργίας.

PebbleKit JS:

Το PebbleKit JS [14] είναι ένα χρήσιμο εργαλείο που επιτρέπει την προσθήκη κώδικα JavaScript (ο οποίος εκτελείται σε sandbox στο επίσημο Pebble app του κινητού) σε οποιοδήποτε watchapp ή watchface. Διαφέρει από το Pebble.js στο ότι υπάρχει περιορισμός στις δυνατότητες που προσφέρει και δεν μπορεί να δημιουργηθεί με αυτό αυτόνομη εφαρμογή σε JavaScript, αντιθέτως χρησιμοποιείται με σκοπό την επέκταση των δυνατοτήτων εφαρμογής που έχει γραφεί σε C. Πρόσθετες δυνατότητες που γίνονται διαθέσιμες είναι η πρόσβαση σε επιπλέον χώρο αποθήκευσης με χρήση του localStorage, πρόσβαση στο διαδίκτυο με XMLHttpRequests, χρήση της τοποθεσίας και η δυνατότητα εμφάνισης στο χρήστη σελίδας επιλογών (configuration page) στην οποία μπορεί να εισάγει δεδομένα στην εφαρμογή και να τροποποιήσει τη συμπεριφορά της.

2.2.1.5 Αποστολή και λήψη δεδομένων μέσω Bluetooth

Για τη μεταφορά δεδομένων μεταξύ Pebble και smartphone υπάρχουν 3 δυνατές επιλογές οι οποίες παρουσιάζουν διαφορετικά χαρακτηριστικά και η χρήση τους ενδείκνυται σε διαφορετικές περιπτώσεις.

1) AppMessage:

AppMessage [15] είναι ένα αμφίδρομο σύστημα επικοινωνίας που επιτρέπει την ανταλλαγή μηνυμάτων μεταξύ smartphone και Pebble. Αυτά τα μηνύματα έχουν τη μορφή ζευγαριών κλειδού/τιμής (key/value) τα οποία οργανώνονται σε μορφή ενός λεξικού (Dictionary) [16]. Το λεξικό αποτελεί έναν τρόπο σειριοποίησης των δεδομένων τοποθετώντας τα σε έναν συνεχόμενο buffer ώστε να βρίσκονται σε ένα εννιαίο κομμάτι μνήμης, γεγονός που διευκολύνει τη μετάδοσή τους μέσω Bluetooth. Το κλειδί είναι το όνομα με το οποίο τα δεδομένα αναγνωρίζονται και στις δύο πλευρές (smartphone – Pebble) και η τιμή επιτρέπεται να είναι είτε ένας ακέραιος (integer), είτε μια συμβολοσειρά (string), είτε ένας byte array. Το AppMessage χρησιμοποιεί ένα συμμετρικό πρωτόκολλο προώθησης (push-oriented messaging protocol) για τη μεταφορά κάθε λεξικού. Αυτό σημαίνει ότι τόσο το κινητό όσο και το Pebble μπορούν να στείλουν μηνύματα, ενώ τα αιτήματα για την έναρξης της μετάδοσης γίνονται από αυτόν που θέλει να στείλει δεδομένα και όχι από τον

παραλήπτη. Η αποστολή κάθε λεξικού επίσης συνοδεύεται από αναγνώριση ή μη (ACKnowledged / Not ACKnowledged) της λήψης τους από την άλλη πλευρά. Στο Pebble γίνεται διαχωρισμός μεταξύ εισερχόμενων (Inbox) και εξερχόμενων (Outbox) κλήσεων και παρέχονται δύο ξεχωριστοί buffers για τη διαχείρηση μηνυμάτων κάθε είδους.

2) AppSync:

Το AppSync [17] στηρίζεται στο AppMessage και χρησιμοποιείται για συγχρονισμό του User Interface. Διατηρεί ένα λεξικό και παρέχει στην εφαρμογή μια συνάρτηση callback με την οποία μπορούμε να αλλάζουμε τα περιεχόμενα του UI όποτε αλλάζει η τιμή του λεξικού, για παράδειγμα όταν έρχονται νέα δεδομένα από το κινητό.

3) DataLogging:

Το DataLogging [18] επιτρέπει την καταγραφή δεδομένων ασύγχρονα σε μια εφαρμογή Android ή iOS (companion app). Παρέχει αποθηκευτικό χώρο 640 KB τα οποία μοιράζονται μεταξύ των εφαρμογών που το χρησιμοποιούν. Υπάρχει η δυνατότητα μια εφαρμογή του Pebble να δημιουργήσει, προσθέσει ή να διαγράψει δεδομένα από ένα session και στη συνέχεια τα δεδομένα αποστέλονται στην σχετική εφαρμογή κινητού τηλεφώνου όσο πιο σύντομα γίνεται. Αν το τηλέφωνο δεν είναι διαθέσιμο εκείνη τη στιγμή, τα δεδομένα αποθηκεύονται στο ρολόι μέχρι αυτό να συνδεθεί με το κινητό. Αν ο χώρος αποθήκευσης που αντιστοιχεί σε μια εφαρμογή ξεπεραστεί τότε η εφαρμογή γράφει πάνω στα ίδια δεδομένα σβήνοντας παλαιότερες τιμές.

2.2.2 JSON

Το JSON (JavaScript Object Notation) [19] είναι μια απλή μορφή αναπαράστασης δεδομένων ως ζεύγη ιδιοτήτων/τιμών. Η σύνταξή του αποτελεί υποσύνολο της γλώσσας JavaScript, όμως κώδικας για δημιουργία και ανάλυση δομών JSON υποστηρίζεται από μεγάλο πλήθος γλωσσών προγραμματισμού. Χρησιμοποιείται ευρέως στην ασύγχρονη επικοινωνία μεταξή browser και server και τείνει να αντικαταστήσει τη γλώσσα σήμανσης XML. Ένα αντικείμενο JSON περικλείεται σε αγκύλες {} και αποτελείται από ένα σύνολο ζευγών ιδιότητας/τιμής. Σε κάθε τέτοιο ζεύγος, η ιδιότητα είναι πάντα τύπου String, ενώ η τιμή ανήκει σε έναν από τους τύπους δεδομένων που υποστηρίζει το JSON, οι οποίοι είναι οι αριθμοί (δεν γίνεται διάκριση μεταξύ ακεραίων και δεκαδικών), Strings, Boolean, Arrays, Objects (συλλογή από ζεύγη ιδιοτήτων/τιμών) και το null που αντιπροσωπεύει την κενή τιμή.

Παράδειγμα αντικειμένου JSON

```
{
      "id": 1,
      "name": "Foo",
      "price": 123,
      "tags": [ "Bar", "Eek" ],
      "stock": {
            "warehouse": 300,
            "retail": 20
      }
}
```

2.2.3 Mongo DB

Η Mongo Database [20] αποτελεί ένα μη σχεσιακό μοντέλο βάσης δεδομένων ανοιχτού κώδικα στο οποίο δεν χρησιμοποιείται η παραδοσιακή οργάνωση με πίνακες. Αντιθέτως, τα δεδομένα αποθηκεύονται ως έγγραφα, η δομή των οποίων είναι παρόμοια με την αναπαράσταση JSON που παρουσιάστηκε νωρίτερα επιτρέποντας έτσι την ενσωμάτωση δεδομένων σε συγκεκριμένες εφαρμογες πολύ πιο γρήγορα και εύκολα. Συγκεκριμένα χρησιμοποιείται η μορφή BSON (Binary JSON) που αποτελεί υπερσύνολο του JSON, καθώς υποστηρίζει επιπλέον τύπους δεδομένων (όπως date) και είναι σχεδιασμένη με τέτοιο τρόπο ώστε να είναι αποδοτικότερη όσον αφορά τον αποθηκευτικό χώρο και την ταχύτητα σάρωσης.

2.2.4 ReST Services

Το ReST (Representational State Transfer) [21] είναι μια μορφή αρχιτεκτονικής για τη δημιουργία δικτυακών εφαρμογών. Είναι ανεξάρτητο πλατφόρμας και γλώσσας προγραμματισμού και χρησιμοποιεί απλά HTTP requests για κάθε λειτουργία δεδομένων (δημιουργία, ανάγνωση, ενημέρωση, διαγραφή). Αποτελεί lightweight εναλλακτική σε πιο σύνθετους μηχανισμούς όπως RPC (Remote Procedure Calls) και Web Services (SOAP, WSDL κλπ) ενώ παρά την απλότητά του είναι πλήρες, παρέχοντας τις ίδες δυνατότητες. Το ReST ακολουθεί τις εξής βασικές αρχές:

Πελάτης – Διακομιστής (ή Εξυπηρετητής) (Client – Server): Η χρήση ομοιόμορφης διαπροσωπείας διαχωρίζει τους πελάτες από τους εξυπηρετητές. Αυτός ο διαχωρισμός σημαίνει ότι για παράδειγμα έναν πελάτη δεν θα πρέπει να τον απασχολούν θέματα επάρκειας χώρου αποθήκευσης δεδομένων το οποίο αφορά αποκλειστικά τον εξυπηρετητή. Αντίστοιχα ένας εξυπηρετητής δεν χρειάζεται να ασχοληθεί με τη διεπαφή χρήστη ή με την κατάσταση χρήστη. Έτσι βελτιώνεται η φορητότητα του κώδικα του πελάτη και γίνεται ευκολότερη η επεκτασιμότητα των εξυπηρετητών.

Ομοιόμορφη Διεπαφή (Uniform Interface): Αποτελεί θεμελιώδη αρχή οποιασδήποτε εφαρμογής του ReST. Η ομοιόμορφη διεπαφή απλοποιεί και αποσυνδέει την αρχιτεκτονική πελάτη – διακομιστή, επιτρέποντας σε κάθε πλευρά να εξελιχθεί ανεξάρτητα.

Χωρίς Κατάσταση (Stateless): Κάθε αίτημα κάποιου πελάτη αντιμετωπίζεται ως ανεξάρτητο και περιέχει όλες τις απαραίτητες πληροφορίες που απαιτούνται για το χειρισμό του. Η κατάσταση συνεδρίας (session state) διατηρείται στον πελάτη και όχι στον εξυπηρετητή. Με αυτόν τον τρόπο απλοπλοιείται ο σχεδιασμός των εξυπηρετητών. Παράδειγμα stateless πρωτοκόλλων είναι το IP (internet protocol) και το HTTP (Hypertext Transfer Protocol).

Ένα σύστημα ιεραρχημένο με τη χρήση της cache μνήμης (Cacheable): Πρέπει να ορίζεται αν οι αποκρίσεις μπορούν να «κρυφτούν» στη μνήμη cache ή όχι, ώστε οι πελάτες να μην επαναχρησιμοποιούν άκυρα ή ακατάλληλα δεδομένα. Έτσι βελτιώνεται η επεκτασιμότητα και η απόδοση.

Διαστρωματωμένο Σύστημα (Layered System): Ο πελάτης δεν μπορεί να γνωρίζει εάν συνδέεται άμεσα με τον Εξυπηρετητή ή με ένα ενδιάμεσο επίπεδο. Οι ενδιάμεσοι εξυπηρετητές μπορούν να βελτιώσουν την επεκτασιμότητα του συστήματος αλλά και την ασφάλειά του.

Κώδικας κατά Ζήτηση (Code on Demand): Οι διακομιστές έχουν τη δυνατότητα να παρατείνουν προσωρινά ή να προσαρμόσουν τη λειτουργία ενός πελάτη με τη μεταφορά εκτελέσιμου κώδικα.

Το Spring Framework είναι ανοιχτού κώδικα πλαίσιο εφαρμογών. Δημιουργήθηκε από την Pivotal Software και επιτρέπει την δημιουργία κάθε είδους εφαρμογής (και RESTful μεταξύ άλλων) χρησιμοποιώντας τη γλώσσα Java. Στην επίσημη ιστοσελίδα του Spring υπάρχει μεγάλο πλήθος από σύντομους οδηγούς που παρέχουν κώδικα ο οποίος δείχνει πως η Java μπορεί να εκμεταλευτεί και να επικοινωνήσει με άλλες τεχνολογίες για τη δημιουργία ενιαίων εφαρμογών.

Spring Framework:

Το Spring Framework [22] είναι ανοιχτού κώδικα πλαίσιο εφαρμογών. Δημιουργήθηκε από την Pivotal Software και επιτρέπει την δημιουργία κάθε είδους εφαρμογής (και RESTful μεταξύ άλλων) χρησιμοποιώντας τη γλώσσα Java. Στην επίσημη ιστοσελίδα του Spring [23] υπάρχει μεγάλο πλήθος από σύντομους οδηγούς που παρέχουν κώδικα ο οποίος δείχνει πως η Java μπορεί να εκμεταλευτεί και να επικοινωνήσει με άλλες τεχνολογίες για τη δημιουργία ενιαίων εφαρμογών.

2.2.5 AJAX και XMLHttpRequest

Το AJAX (Asynchronus JavaScript and XML) [24] είναι μια τεχνική που επιτρέπει τη δημιουργία ασύγχρονων διαδικτυακών εφαρμογών. Επιτρέπει δηλαδή τα δεδομένα να στέλνονται ή να λαμβάνονται από κάποιο διακομιστή σε μορφή XML ή JSON, χωρίς να επηρεάζεται αυτό που βλέπει ο χρήστης ή γενικότερα η λειτουργία της εφαρμογής.

Το XMLHttpRequest είναι ένα API (Application Programming Interface) που παρέχει σε έναν πελάτη (client) τη δυνατότητα μεταφοράς δεδομένων μεταξύ εκείνου και ενός διακομιστή (server). Καθιστά εύκολη την ανάκτηση δεδομένων από κάποιο URL χωρίς να χρειάζεται να γίνει ανανέωση στη σελίδα. Αυτό δίνει τη δυνατότητα μια σελίδα να ανανεωθεί εν μέρει χωρίς να παρεμποδίζεται ο χρήστης. Το XMLHttpRequest χρησιμοποιείται σε μεγάλο βαθμό στο AJAX.

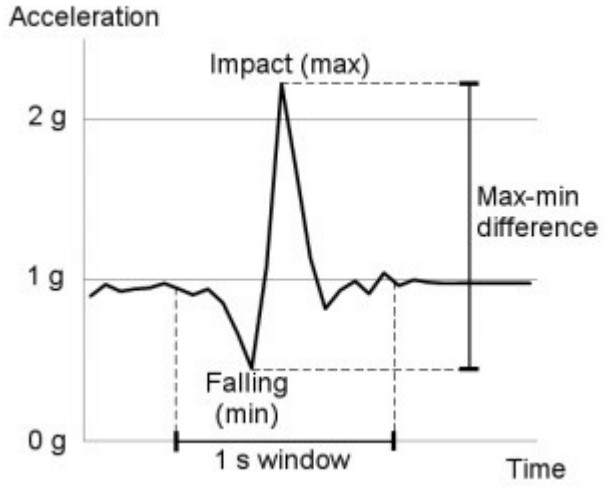
2.2.6 CORS

Το CORS (Cross-origin resource sharing) [25] είναι ένας μηχανισμός που επιτρέπει σε κάποιους πόρους μια ιστοσελίδας να χρησιμοποιηθούν από διαφορετικό domain εκτός αυτού στο οποίο βρίσκονται. Μια ιστοσελίδα γενικά μπορεί να συμπεριλάβει ελεύθερα εικόνες, stylesheets, scripts, videos και plugins που προέρχονται από οποιοδήποτε άλλο domain. Δεν ισχύει το ίδιο όμως στην περίπτωση των AJAX Requests (XMLHttpRequest) όταν προέρχονται από διαφορετικό domain από αυτό στο οποίο βρίσκονται τα δεδομένα που ζητούνται (cross-domain). Τα cross-domain AJAX request είναι απαγορευμένα από προεπιλογή για λόγους ασφαλείας. Το CORS δίνει τη δυνατότητα να καθοριστεί με ασφάλεια αν θα πρέπει να επιτραπεί η όχι κάποιο cross-domain request μεταξύ του browser και ενός server.

2.3 Έρευνες και δημοσιεύσεις

Σε αυτή την ενότητα παρατίθενται ερευνητικές εργασίες και δημοσιεύσεις από την παγκόσμια βιβλιογραφία που σχετίζονται με το αντικείμενο του εντοπισμού πτώσεων και της αναγνώρισης δραστηριοτήτων με τη βοήθεια διαφόρων ειδών συσκευών χρησιμοποιώντας είτε αλγορίθμους με thresholds είτε μηχανικής μάθησης.

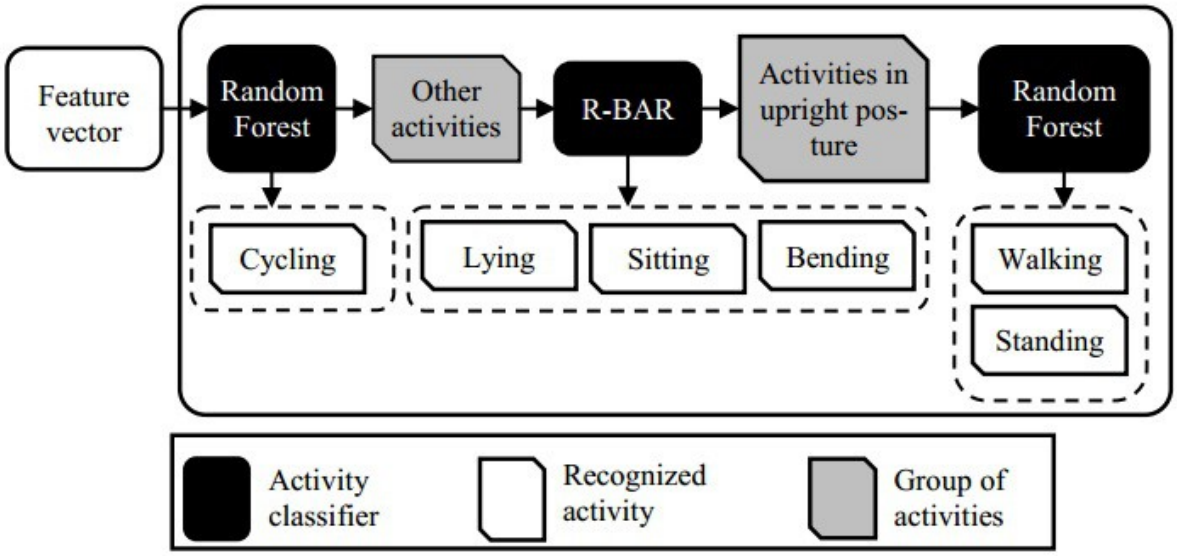
Στη δημοσίευση [1] των Simon Kozina, Hristijan Gjoreski, Matjaž Gams και Mitja Luštrek το 2013 από το Department of Intelligent Systems της Λιουμπλιάνα στη Σλοβενία, χρησιμποιούνται δύο αισθητήρες, ένας στο στέρνο κι ένας στον μηρό, οι οποίοι στέλνουν συνεχώς δεδομένα επιταχύνσεων μέσω Bluetooth σε κάποιο laptop. Στη συνέχεια γίνεται συγχρονισμός των μετρήσεων από τους δύο αισθητήρες και γίνεται διαχωρισμός σε δύο πλευρές. Η μια αναλαμβάνει τη μετατροπή των επιταχύνσεων σε feature vectors οι οποίοι χρησιμοποιούνται για την αναγνώριση δραστηριοτήτων και η άλλη εξετάζει το μοτίβο των επιταχύνσεων ελέγχοντας για πτώσεις. Σύμφωνα με τον Kozina και τους συνεργάτες του, κατά την πτώση, το τυπικό μοτίβο των επιταχύνσεων όπως μετριέται από το επιταχυνσιόμετρο που βρίσκεται στο στέρνο αποτελείται από μια μείωση της επιτάχυνσης η οποία ακολουθείται από μια άυξηση.



Σχήμα 3: Μοτίβο μετρούμενης επιτάχυνσης κατά την πτώση σύμφωνα με τη δημοσίευση των Kozina, Gjoreski, Gams και Luštrek.

Αυτό συμβαίνει επειδή το επιταχυνσιόμετρο σε κατάσταση ηρεμίας μετράει 1 g επιτάχυνση που οφείλεται στη βαρύτητα της γης και κατά τη διάρκεια μια ελεύθερης πτώσης 0 g. Όταν το άτομο ξεκινάει να πέφτει, η επιτάχυνση μειώνεται από το 1 g περίπου στο 0.5 g (ποτέ δεν επιτυγχάνεται τέλεια ελεύθερη πρώση) ενώ κατά την πρόσκρουση στο έδαφος παρουσιάζεται μια σύντομη και έντονη αύξηση στην μετρούμενη επιτάχυνση. Ο αλγόριθμος ελέγχει αν κάποιο προκαθορισμένο threshold στις επιταχύνσεις ξεπερνιέται και αν συμβαίνει αυτό στη συνέχεια γίνεται έλεγχος αν το άτομο βρίσκεται σε οριζόντια θέση. Πιο συγκεκριμένα χρησιμοποιείται το μέτρο του διανύσματος της επιταχύνσης και υπολογίζεται η διαφορά μεταξύ της μέγιστης και της ελάχιστης τιμής σε ένα χρονικό παράθυρο ενός δευτερολέπτου. Αν ξεπερνιέται το προκαθορισμένο όριο στη συνέχεια υπολογίζεται η γωνία μεταξύ του διανύσματος της επιτάχυνσης και του άξονα z του επιταχυνσιομέτρου ο οποίος όταν το άτομο είναι όρθιο δείχνει προς τη γη. Η τιμή της γωνίας επιτρέπει να καταλάβουμε αν το άτομο είναι όρθιο ή βρίσκεται σε οριζόντια θέση κάτι το οποίο σηματοδοτεί και την πτώση. Όσον αφορά την αναγνώριση δραστηριοτήτων το feature vector περνάει αρχικά από έναν Random Forest classifier ο οποίος έχει εκπαιδευτεί με κατάλληλο τρόπο ώστε να ξεχωρίζει την ποδηλασία από τις υπόλοιπες δραστηριότητες. Αν η τρέχουσα δραστηριότητα δεν είναι ποδηλασία τότε σε ένα δεύτερο επίπεδο γίνεται διαχωρισμός σε Lying, Sitting και Bending με τη βοήθεια κάποιων κανόνων (rule-based activity recognition) που

χρισιμοποιούν features εκφραζόμενα μόνο ως μέσες τιμές. Αν ούτε εδώ υπάρξει κάποια ταύτιση, και η στάση του σώματος είναι όρθια, πλέον γίνεται διαχωρισμός σε Walking ή Standing με τη χρήση του Random Forest classifier.

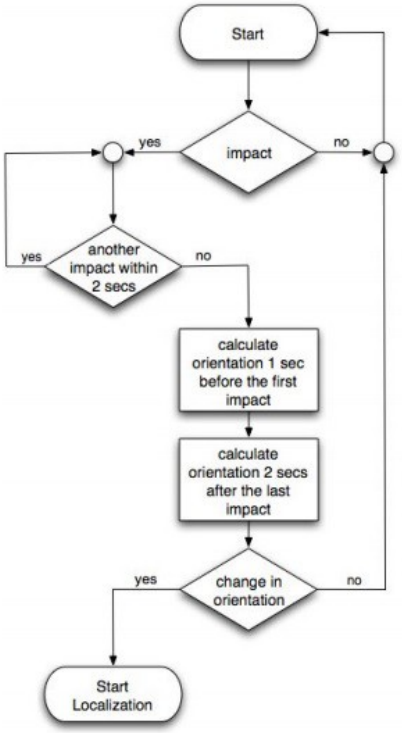


Σχήμα 4: Αναγνώριση δραστηριοτήτων 3 επιπέδων των Kozina, Gjoreski, Gams και Luštrek.

Με τις παραπάνω τεχνικές επιτυγχάνεται αναγνώριση δραστηριοτήτων με F-measure 99% που θεωρείται πάρα πολύ καλό ενώ για τον εντοπισμό πτώσεων το F-measure είναι 78% το οποίο δεν είναι τόσο καλό, όμως θεωρείται αποδεκτό δεδομένης της δυσκολίας του θέματος. ■

Στη δημοσίευση [2] των Jay Chen, Karric Kwong, Dennis Chang, Jerry Luk, Ruzena Bajcsy από το Department of Electrical Engineering and Computer Sciences του πανεπιστημίου Berkeley της Καλιφόρνια χρησιμοποιήθηκε συσκευή με επιταχυνσιόμετρο η οποία τοποθετήθηκε σε ζώνη στη μέση. Γίνεται αναφορά στις δυσκολίες που υπάρχουν στο θέμα της επιλογής του κατάλληλου σημείου που θα τοποθετηθεί ο αισθητήρας και αναφέρεται ότι ο αρχικός στόχος ήταν η τοποθέτησή του στον καρπό, παρόμοια με τη χρήση ενός ρολογιού, όμως βασιζόμενοι σε άλλες μελέτες απέφυγαν το συγκεκριμένο σημείο λόγω του ότι οι συχνές και έντονες κινήσεις του χεριού δυσκολεύουν τη διαδικασία. Αναφέρουν ότι η μεταφορά και επεξεργασία των μετρήσεων σε κάποιον ισχυρό υπολογιστή επιτρέπει ταχύτερη και βαθύτερη ανάλυση, για παράδειγμα με χρήση αναγνώρισης προτύπων, όμως επιβαρύνει σημαντικά το δίκτυο καθώς θα πρέπει να γίνεται συνεχής μεταφορά δεδομένων όσο η συσκευή είναι ανοιχτή. Για αυτό το λόγο περιορίστηκαν στην επεξεργασία πάνω στη συσκευή, ενώ η επικοινωνία με υπολογιστή γίνεται σε περίπτωση που εντοπιστεί πτώση προκειμένου να σταλεί ειδοποίηση έκτακτης ανάγκης. Κατά την πτώση, παρατηρήθηκε ότι οι επιταχύνσεις παρουσιάζουν μεγαλύτερες τιμές σε σχέση με εκείνες που παρατηρούνται κατά την εκτέλεση καθημερινών δραστηριοτήτων, όμως για τον εντοπισμό μιας πτώσης δεν αρκεί να τεθούν κάποια thresholds στις επιταχύνσεις σε κάθε άξονα διότι έτσι δεν καλύπτονται όλες οι δυνατές κατεθύνσεις στις οποίες μπορεί να συμβεί μια βίαιη σύγκρουση στο έδαφος. Έτσι χρησιμοποιήθηκε κι εδώ το μέτρο του διανύσματος των επιταχύνσεων και παρατηρήθηκε ότι όταν ξεπερνάει τα 3 g είναι πιθανό να έχει συμβεί μια πτώση χωρίς όμως αυτό να είναι αρκετό. Η τοποθέτηση του επιταχυνσιομέτρου στη μέση επιτρέπει τον εντοπισμό μεγάλων

αλλαγών στη θέση του ατόμου. Το απλούστερο παράδειγμα είναι από όρθια στάση πριν την πτώση σε οριζόντια θέση όταν φτάσει στο έδαφος, όπου έχουμε συνολικά 90^{o} αλλαγή. Έτσι, ο αλγόριθμος συμπληρώνεται ελέγχοντας το orientation της συσκευής 1 δευτερόλεπτο πριν από τη στιγμή που παρατηρήθηκε το ξεπέρασμα του threshold και 2 δευτερόλεπτα μετά, εξετάζοντας αν υπήρξε σημαντική αλλαγή. Επίσης ο αλγόριθμος συμπεριλαμβάνει και περιπτώσεις όπου παρουσιάζονται πολλαπλές συγκρούσεις όπως σε περίπτωση πτώσης από σκάλες. ■



Σχήμα 5: Διάγραμμα ροής του αλγορίθμου εντοπισμού πτώσεων των Jay Chen, Karric Kwong, Dennis Chang, Jerry Luk, και Ruzena Bajcsy.

Στη δημοσίευση [3] των Ahmet Turan Özdemir και Billur Barshan από το Department of Electrical and Electronics Engineering του πανεπιστημίου Erciyes της Τουρκίας το 2014, έγινε μια κάπως πιο διαφορετική προσέγγιση των πτώσεων που στηρίζεται στη χρήση μηχανικής μάθηση και όχι σε αλγόριθμο με thresholds. Για τον παραπάνω σκοπό χρησιμοποιήθηκαν συνολικά 6 συσκευές που τοποθετήθηκαν σε διάφορα μέρη του σώματος, κάθε μια από τις οποίες αποτελείται από επιταχυνσιόμετρο, γυροσκόπιο και μαγνητόμετρο/πυξίδα και έγιναν προσομοιώσεις πτώσεων διαφόρων τύπων και ADLs που φαίνονται στον πίνακα 2.

Fall Actions		
#	**Label**	**Description**
1	front-lying	from vertical falling forward to the floor
2	front-protecting-lying	from vertical falling forward to the floor with arm protection
3	front-knees	from vertical falling down on the knees
4	front-knees-lying	from vertical falling down on the knees and then lying on the floor
5	front-right	from vertical falling down on the floor, ending in right lateral position
6	front-left	from vertical falling down on the floor, ending in left lateral position
7	front-quick-recovery	from vertical falling on the floor and quick recovery
8	front-slow-recovery	from vertical falling on the floor and slow recovery
9	back-sitting	from vertical falling on the floor, ending sitting
10	back-lying	from vertical falling on the floor, ending lying
11	back-right	from vertical falling on the floor, ending lying in right lateral position
12	back-left	from vertical falling on the floor, ending lying in left lateral position
13	right-sideway	from vertical falling on the floor, ending lying
14	right-recovery	from vertical falling on the floor with subsequent recovery
15	left-sideway	from vertical falling on the floor, ending lying
16	left-recovery	from vertical falling on the floor with subsequent recovery
17	syncope	from standing falling on the floor following a vertical trajectory
18	syncope-wall	from standing falling down slowly slipping on a wall
19	podium	from vertical standing on a podium going on the floor
20	rolling-out-bed	from lying, rolling out of bed and going on the floor
Non-Fall Actions (ADLs)		
#	**Label**	**Description**
21	lying-bed	from vertical lying on the bed
22	rising-bed	from lying to sitting
23	sit-bed	from vertical to sitting with a certain acceleration onto a bed (soft surface)
24	sit-chair	from vertical to sitting with a certain acceleration onto a chair (hard surface)
25	sit-sofa	from vertical to sitting with a certain acceleration onto a sofa (soft surface)
26	sit-air	from vertical to sitting in the air exploiting the muscles of legs
27	walking-fw	walking forward
28	jogging	running
29	walking-bw	walking backward
30	bending	bending about 90 degrees
31	bending-pick-up	bending to pick up an object on the floor
32	stumble	stumbling with recovery
33	limp	walking with a limp
34	squatting-down	squatting, then standing up
35	trip-over	bending while walking and then continuing walking
36	coughing-sneezing	coughing or sneezing

Πίνακας 2: Διαφορετικοί τύποι πτώσεων και Activities of Daily Living που χρησιμοποιήθηκαν στις προσομοιώσεις για τον εντοπισμό πτώσεων από τη δημοσίευση των Ahmet Turan Özdemir και Billur Barshan

Έτσι συγκεντρώθηκαν δεδομένα επιταχύνσεων, δύναμης του μαγνητικού πεδίου της γης και ρυθμός περιστροφής με συχνότητα 25 Hz και στη συνέχεια εξήχθησαν τα χαρακτηριστικά ελάχιστη και μέγιστη τιμή, διακύμανση, λοξότητα, κυρτότητα, οι πρώτες 11 τιμές τις αυτοσυσχέτισης και τα πρώτα 5 peaks του διακριτού μετασχηματισμού Fourier (DFT). Για τη μηχανική μάθηση δοκιμάστηκαν οι classifiers *k*-nearest neighbor (*k*-NN), least squares method (LSM), support vector

machines (SVM), Bayesian decision making (BDM), dynamic time warping (DTW), and artificial neural networks (ANNs). Όσον αφορά τα αποτελέσματα, επιτεύχθηκε επιτυχής αναγνώριση δραστηριοτήτων με sensitivity, specificity και accuracy μεγαλύτερη από 99% με classifiers τους k-NN και LSM ενώ για τις πτώσεις επιτεύχθηκε accuracy 95% με τη χρήση support vector machines. ■

Κεφάλαιο 3. Αρχιτεκτονική

Η παρούσα διπλωματική απαρτίζεται από τα ακόλουθα συστατικά στοιχεία, καθένα από τα οποία παρουσιάζεται συνοπτικά στις παρακάτω ενότητες του παρόντος κεφαλαίου. Γενικά, αρχικά δημιουργήθηκε εφαρμογή για την συλλογή δεδομένων από το επιταχυνσιόμετρο του Pebble και χαρακτηρισμό από τον χρήστη της δραστηριότητάς του ανά τακτά χρονικά διαστήματα όπως επίσης και τον χαρακτηρισμό κάποιου χρονικού διαστήματος ως πτώση. Αυτά τα δεδομένα χρησιμοποιήθηκαν ώστε να έχουμε μια οπτική εικόνα για την κάθε δραστηριότητα, επομένως καλύτερη κατανόηση και για τη μελέτη μηχανικής μάθησης ώστε στη συνέχεια να κατασκευαστεί ένα σύστημα που θα έχει διδαχθεί να ξεχωρίζει από μόνο του τον τύπο της δραστηριότητας του χρήστη και ενδεχομένως να εντοπίζει πτώσεις. Τέλος δημιουργήθηκε διαδικτυακή εφαρμογή που παρουσιάζει γραφικά τη δραστηριότητα του χρήστη και εμφανίζει χρήσιμα στατιστικά στοιχεία.

3.1 Συλλογή δεδομένων από το επιταχυνσιόμετρο

Σκοπός αρχικά ήταν να συλλέξουμε δεδομένα από το επιταχυνσιόμετρο του Pebble σε καθημερινή βάση και να τα χαρακτηρίσουμε ανάλογα με την εκάστοτε δραστηριότητα, ώστε να μπορούμε στη συνέχεια να οπτικοποιήσουμε με κάποιο τρόπο τις μετρήσεις μας και να βγάλουμε κάποια συμπεράσματα. Για το λόγο αυτό χρειάστηκε να φτιάξουμε εφαρμογή στο Pebble η οποία συλλέγει επιταχύνσεις και τις στέλνει σε κάποιον εξυπηρετητή όπου αποθηκεύονται σε μια βάση δεδομένων. Λόγω της φύσης της εφαρμογής ήταν αναγκαίο να χρησιμοποιηθεί κώδικας σε JavaScript για την αποστολή των δεδομένων στον εξυπηρετητή. Έτσι οι επιλογές που είχαμε ήταν δύο.

1) Δημιουργία εφαρμογής στο πλαίσιο **Pebble.js** αποκλειστικά σε γλώσσα JavaScript η οποία εκτελείται στο smartphone, το οποίο με τη σειρά του χρησιμοποιεί το Bluetooth κάθε φορά που χρειάζεται να αξιοποιήσει πόρους του Pebble.

2) Δημιουργία εφαρμογής στα πλαίσια **Pebble C** και **PebbleKit JS**. Εδώ γίνεται χρήση της γλώσσας C για δημιουργία κώδικα που αναλαμβάνει τη συλλογή δεδομένων από το επιταχυνσιόμετρο σε έναν buffer μέχρι αυτός να συμπληρωθεί και στη συνέχεια την αποστολή στο smartphone μέσω Bluetooth. Πλέον αφού τα δεδομένα φτάσουν στο smartphone, το PebbleKit JS αναλαμβάνει την αποστολή τους στον εξυπηρετητή με τη χρήση της JavaScript.

Δοκιμάστηκαν και οι δύο εναλλακτικές. Η αρχική εκτίμηση ήταν ότι η χρήση Pebble.js απλοποιεί τα πράγματα λόγω της χρήσης μόνο μιας γλώσσας προγραμματισμού το οποίο και αληθεύει, όμως όπως αποδείχτηκε στη συνέχεια η εφαρμογή κατανάλωνε τη μπαταρία μέσα σε λίγες ώρες επειδή το Bluetooth έμενε συνεχώς ανοιχτό λόγω της συνεχούς μεταφοράς δεδομένων απο το επιταχυνσιόμετρο. Για αυτό, κρίθηκε αναγκαία η χρήση της δεύτερης επιλογής με σκοπό να αξιοποιηθεί ο εσωτερικός χώρος αποθήκευσης του Pebble όπου θα αποθηκεύονται προσωρινά οι μετρήσεις, έτσι ώστε το Bluetooth να χρησιμοποιείται ανά μεγαλύτερα χρονικά διαστήματα, κάτι που δεν ήταν δυνατό να ρυθμιστεί πριν. Όσον αφορά το κομμάτι της JavaScript, αναλαμβάνει την αποστολή στον εξυπηρετητή μέσω XMLHttpRequest μιας δομής JSON που αφορά τις επιταχύνσεις στους άξονες x, y και z μαζί με τα αντίστοιχα timestamps και μια δομή JSON που αφορά τον χαρακτηρισμό των δραστηριοτήτων μαζί με τις χρονικές στιγμές έναρξης και λήξης τους. Οι επιταχύνσεις στέλνονται ανά περίπου 2 ώρες από το smartphone στον εξυπηρετητή ενώ ο χαρακτηρισμός των δραστηριοτήτων αποστέλλεται μόλις ληφθεί από το Pebble.

3.2 ReSTful Server – Σχεδιασμός

Ο εξυπηρετητής υλοποιήθηκε στο Spring Framework σε λειτουργικό σύστημα Linux, ακολουθεί τις

αρχές του ReST και φιλοξενείται στις Cloud υπηρεσίες του "Ωκεανού". Έχει σχεδιαστεί έτσι ώστε να μπορεί να λαμβάνει δεδομένα σε κατάλληλη μορφή JSON όπως έρχονται από την εφαρμογή του Pebble και να τα αποθηκεύει σε κατάλληλο collection της βάσης δεδομένων Mongo. Επίσης ενσωματώσαμε φίλτρο CORS (cross-origin resource sharing) το οποίο επιτρέπει σε εξωτερικές εφαρμογές να αποκτήσουν πρόσβαση και να χρησιμοποιήσουν τα δεδομένα. Αυτό είναι απαραίτητο προκειμένου να φτιαχτεί στο τέλος η διαδικτυακή εφαρμογή που θα κατασκευάζει γραφικές παραστάσεις των δεδομένων στον browser του κάθε χρήστη.

Η πρόσβαση στα αποθηκευμένα δεδομένα ανάλογα με το collection που αποθηκεύονται γίνεται σε διαφορετική διεύθυνση URL. Συνολικά έχουμε:

1) http://83.212.115.163:8080/pebble όπου μπορούμε να δούμε τη δομή JSON που περιέχει πίνακες με τις επιταχύνσεις και τα αντίστοιχα timestamps. Μια καινούρια εγγραφή δημιουργείται όποτε η εφαρμογή στέλνει δεδομένα στον server δηλαδή κάθε δύο περίπου ώρες και οι πίνακες έχουν κατάλληλο μέγεθος έτσι ώστε να χωράνε ακριβώς αυτά τα δεδομένα.

2) http://83.212.115.163:8080/activity όπου βλέπουμε τις δραστηριότητες που χαρακτηρίστηκαν από το χρήστη μαζί με τις χρονικές στιγμές που αντιστοιχούν στην έναρξη και τη λήξη αυτών.


```
{
      "user" : "kostas",
      "activity" : [ "Walking", "Driving" ],
      "start" : [ 1463220184392, 1463226607900 ],
      "end" : [ 1463223784392, 1463227207900 ],
      "comment" : null,
      "_links" : {
        "self" : {
          "href" :
"http://83.212.115.163:8080/activity/57371373c8f2b3ed8339cd1d"
        },
        "activity" : {
          "href" :
"http://83.212.115.163:8080/activity/57371373c8f2b3ed8339cd1d"
        }
      }
    }
```


3) http://83.212.115.163:8080/tracker. Εδώ αποθηκεύονται πλέον οι δραστηριότητες όπως τις έχει συμπεράνει το σύστημα εφαρμόζοντας τον classifier που προέκυψε από τη μηχανική μάθηση.

4) http://83.212.115.163:8080/simple. Διαχωρίζει πότε ο χρήστης κοιμάται και πότε είναι ξύπνιος. Το πεδίο type στις εγγραφές είναι πάντα εναλλαξ 0 και 1 καθώς η εφαρμογή στην οποία χρησιμοποιείται έχει φτιαχτεί έτσι ώστε να στέλνει δεδομένα μόνο όταν γίνεται αλλαγή από τη μια κατάσταση στην άλλη.


```
{
    "simple" : [ {
      "user" : "kostas",
      "timestamp1" : 1468766065960,
```

```
      "timestamp2" : 1468766065960,
      "type" : 1,
      "comment" : null,
      "_links" : {
        "self" : {
          "href" :
"http://83.212.115.163:8080/simple/578b97afc8f2042f19953bdc"
        },
        "simple" : {
          "href" :
"http://83.212.115.163:8080/simple/578b97afc8f2042f19953bdc"
        }
      }
    }, {
      "user" : "kostas",
      "timestamp1" : 1468766735696,
      "timestamp2" : 1468766735696,
      "type" : 0,
      "comment" : null,
      "_links" : {
        "self" : {
          "href" :
"http://83.212.115.163:8080/simple/578b9a4bc8f2042f19953bdd"
        },
        "simple" : {
          "href" :
"http://83.212.115.163:8080/simple/578b9a4bc8f2042f19953bdd"
        }
      }
    }, {
      "user" : "kostas",
      "timestamp1" : 1468766979209,
      "timestamp2" : 1468766979209,
      "type" : 1,
      "comment" : null,
      "_links" : {
        "self" : {
          "href" :
"http://83.212.115.163:8080/simple/578b9b41c8f2042f19953bde"
        },
        "simple" : {
          "href" :
"http://83.212.115.163:8080/simple/578b9b41c8f2042f19953bde"
        }
      }
    }
```


3.3 Ανάλυση δεδομένων με αλγορίθμους μηχανικής μάθησης

Οι δραστηριότητες που θα θέλαμε αρχικά να μπορεί να διαχωρίσει το σύστημα φαίνονται στον

παρακάτω πίνακα.

Name	Description	Events it depends on
Watching tv	The user is watching tv	Tv activity Tv sound
Washing in front of mirror	The user is washing his/her face	
Toilet use	The user used the toilet	Flush toilet bathroom sound
Bathing	Performing various bathing and personal care procedures, within the bathhub	Shower Bathroom sound
Brushing teeth	The user is brushing his/her teeth	Shower Bathroom sound
Awake	The user is awake	All available evidence
Mobile	The user is mobile	All available evidence
Dressing	The user has changed clothes	Upper clothes view Lower clothes view Will try to infer dressing based on the difference of of clothes
Sitting	The user is sitting	Pose depth view
Standing	The user is standing	Pose depth view
Sleeping_early_day	The user sleeps during morning time	Lights off view Location Snoring sound
Sleeping_mid_day	The user sleeps during mid-day or afternoon	Lights off view Location Snoring sound
Sleeping_late_day	The user sleeps in the evening	Lights off view Location Snoring sound

Πίνακας 3: Δραστηριότητες και γεγονότα από τα οποία εξαρτώνται.

Εξαιτίας όμως των περιορισμένων δυνατοτήτων του Pebble και της μη ύπαρξης αισθητήρων ήχου, και φωτεινότητας όπως επίσης και συσκευής βιντεοσκόπησης δεν είναι δυνατή η διάκριση μεταξύ

πολλών από τις παραπάνω δραστηριότητες. Έτσι, έγινε ομαδοποίηση σε κατηγορίες ανάλογα με την ενεργητικότητα (intensity) που παρουσιάζεται σε κάθε δραστηριότητα. Έχουμε λοιπόν τις κατηγορίες sleep, mild, moderate και intense. Η αντιστοίχιση των δραστηριοτήτων του πίνακα 1. και κάποιων επιπλέον σε ομάδες φαίνονται στον παρακάτω πίνακα.

Sleep	Mild	Moderate	Intense
Sleeping_early_day Sleeping_mid_day Sleeping_late_day	Wathing tv Sitting Standing Toilet use Typing Eating	Washing in front of mirror Bathing Brushing teeth Cooking Chores Walking Driving	Running Dancing Exercising

Πίνακας 4: Ομαδοποίηση δραστηριοτήτων σε κατηγορίες ενεργητικότητας.

Αφού έγινε συλλογή ενός ικανοποιητικού data set δραστηριοτήτων και πτώσεων, έγινε feature extraction. Αξιοποιώντας τις γνώσεις από υπάρχουσες έρευνες και δημοσιεύσεις [3], επιλέχθηκε το μοντέλο της επιτηρούμενης μάθησης και σαν features επιλέχθηκαν τα ακόλουθα:

Μέγιστη και ελάχιστη τιμή:

Είναι η μέγιστη και η ελάχιστη τιμή του μέτρου του διανύσματος των επιταχύνσεων. $\sqrt{x^2+y^2+z^2}$ που παρατηρήθηκε σε όλοκληρο το δείγμα.

Διακύμανση (variance):

Είναι η αναμενόμενη τιμή της τετραγωνικής απόκλισης από τη μέση τιμή του μέτρου του διανύσματος των επιταχύνσεων. Μετρά το πόσο μακριά απλώνεται ένα σύνολο τιμών από τη μέση τιμή του.

$$\text{variance}(\mathbf{s}) \quad = \quad \sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(s_n - \mu)^2$$

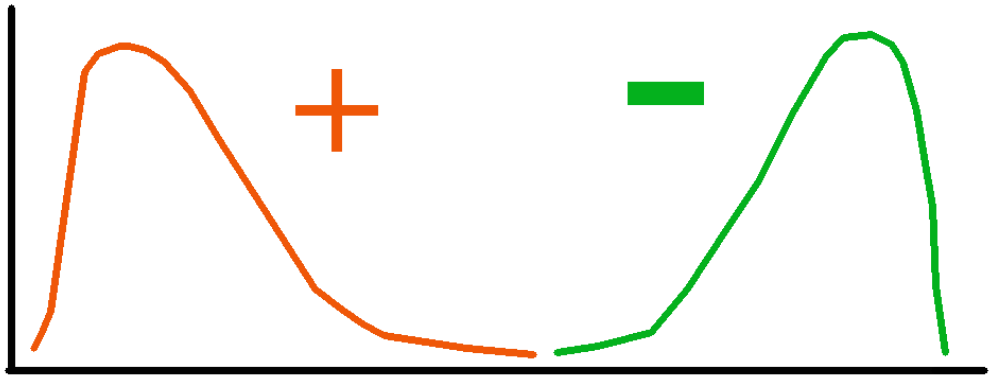
Τυπική απόκλιση (standard deviation):

Είναι η τετραγωνική ρίζα της διακύμανσης και αποτελεί μέτρο της διασποράς ενός συνόλου τιμών. Χρήσιμη ιδιότητα της τυπικής απόκλισης σε αντίθεση με τη διακύμανση είναι ότι εκφράζεται στις ίδιες μονάδες με τα δεδομένα.

Λοξότητα (skewness):

Αναφέρεται στο επίπεδο ασυμμετρίας που μπορεί να έχει η διασπορά ενός συνόλου δεδομένων σε σχέση με τον μέσο όρο τους. Αν παρουσιάζεται στα αριστερά ενός αριθμητικού συνόλου μικρότερη διασπορά από ότι στα δεξιά του, τότε λέγεται πως υπάρχει θετική λοξότητα αλλιώς αρνητική.

$$\text{skewness}(\mathbf{s}) = \frac{1}{N\sigma^3}\sum_{n=1}^{N}(s_n - \mu)^3$$

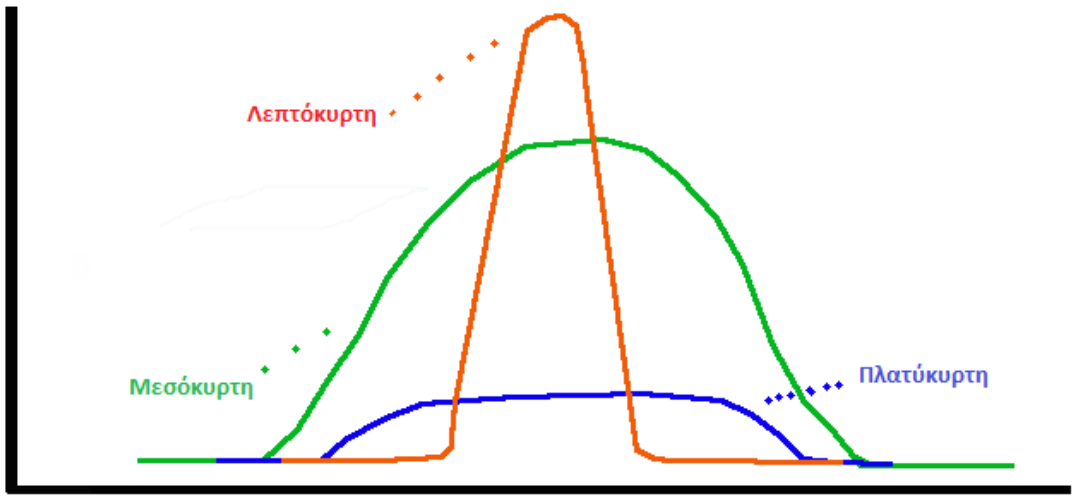


Σχήμα 6: Αναπαράσταση της λοξότητας.

Κύρτωση (kurtosis):

Είναι έννοια που προσπαθεί να περιγράψει το σχήμα που έχει η κορυφή μιας κατανομής. Υπάρχουν παραδοσιακά τρεις περιπτώσεις που συναντώνται εδώ. Πλατύκυρτη η οποία περιγράφει κορυφές που πλησιάζουν την επιπεδότητα. Λεπτόκυρτη η οποία περιγράφει κορυφές που είναι "μυτερές". Μεσόκυρτη η οποία περιγράφει κατανομές που κινούνται ενδιάμεσα.

$$\text{kurtosis}(\mathbf{s}) = \frac{1}{N\sigma^4}\sum_{n=1}^{N}(s_n - \mu)^4$$



Σχήμα 7: Αναπαράσταση των τύπων της κυρτότητας.

Angle φ:

Είναι το συνημίτονο της γωνίας μεταξύ του τελευταίου δείγματος του διανυσματικού αθροίσματος και του προηγούμενου. Βοηθά στην ανίχνευση δραστηριοτήτων των οποίων οι επιταχύνσεις ανάμεσα στα επακόλουθα δείγματα είναι διαφορετικές.

$$cos\varphi = \frac{x_1 * x_2 + y_1 * y_2 + z_1 * z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2} * \sqrt{x_2^2 + y_2^2 + z_2^2}}$$

Distance:

Είναι το μέτρο του διανύσματος που προκύπτει από τη διαφορά δύο διαδοχικών δειγμάτων.

$$Distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Translation Change:

Η ποσότητα $tChange = \left|(x_2 + y_2 + z_2)^2 - (x_1 + y_1 + z_1)^2\right|$

Vertical Change:

Η ποσότητα $vChange = \frac{x_1 * x_2 + y_1 * y_2 + z_1 * z_2}{\sqrt{x_2^2 + y_2^2 + z_2^2}}$

Στη συνέχεια χρησιμοποιήθηκε πάνω στο data set **10-Fold Cross-validation** με τη βοήθεια του λογισμικού Weka. Αυτό σημαίνει ότι χωρίζεται το data set σε 10 σύνολα ίδιου μεγέθους, εφαρμόζεται ο classification algorithm θεωρώντας άγνωστο το πρώτο από αυτά και γνωστά τα υπόλοιπα 9 και η διαδικασία επαναλαμβάνεται άλλες 9 φορές θεωρώντας ως άγνωστο διαδοχικά κάθε ένα από τα 10 αρχικά σύνολα. Ο τελικός classifier, αποτελεί τον μέσο όρο που προκύπτει από την παραπάνω διαδικασία.

Classification algorithm J48:

Ως classification algorithm ξεχώρισε και επιλέχθηκε ο J48, ο οποίος αποτελεί υλοποίηση ανοιχτού κώδικα σε Java, του αλγορίθμου C4.5. Ο C4.5 είναι ένας αλγόριθμος που παράγει ένα δέντρο αποφάσεων το οποίο μπορεί να χρησιμοποιηθεί για classification και ενσωματώνεται πολύ εύκολα σε οποιοδήποτε κώδικα λόγω της μορφής του. Αναπτύχθηκε από τον Ross Quinlan και αποτελεί επέκταση προηγούμενης δουλειάς του (αλγόριθμος ID3). Ο C4.5 χρησιμοποιεί την έννοια της εντροπίας της πληροφορίας για να παράγει το δέντρο αποφάσεων. Το σύνολο δεδομένων εκπαίδευσης είναι ένα σύνολο $S = s_1, s_2, ... , s_n$ από δείγματα. Κάθε δείγμα, είναι ένα διάνυσμα k διαστάσεων όπου οι διαστάσεις είναι τα attributes με ένα από αυτά το class. Σε κάθε κόμβο του δέντρου, ο C4.5 επιλέγει το attribute που διαχωρίζει το σύνολο των δεδομένων πιο αποδοτικά σε υποσύνολα που ανήκουν σε διαφορετικά classes. Το κριτήριο διαχωρισμού αποτελεί τη γνώση που κερδίζουμε. Ο αλγόριθμος επιλέγει το attribute που δίνει τη μεγαλύτερη πληροφορία.

Ο παραπάνω αλγόριθμος έδωσε classifier με 97% σωστή πρόβλεψη όσον αφορά τις δραστηριότητες και 72.5% όσον αφορά τις πτώσεις. Για τις δραστηριότητες το ποσοστό θεωρήθηκε

πολύ ικανοποιητικό και ο classifier στη συνέχεια εφαρμόστηκε σε ένα test set με 97.5% σωστή πρόβλεψη. Για τις πτώσεις το ποσοστό των 72.5% θεωρήθηκε κακό. Για αυτό δεν υλοποιήθηκε αντίστοιχος αλγόριθμος βασισμένος σε μηχανική μάθηση και προτιμήθηκε η υλοποίηση αλγορίθμου με thresholds. Τα αποτελέσματα της μηχανικής μάθησης παρουσιάζονται αναλυτικά στο κεφάλαιο 5.

3.4 Εφαρμογή Pebble για αναγνώριση δραστηριοτήτων και εντοπισμό πτώσεων

Υλοποιήθηκαν συνολικά δύο εφαρμογές για αναγνώριση δραστηριοτήτων και εντοπισμό πτώσεων. Και οι δύο χρησιμοποιούν Pebble C και PebbleKit JS συνδυαστικά. Η διαφορά τους είναι ότι η στην πρώτη υλοποίηση ο κώδικας για την αναγνώριση δραστηριοτήτων και εντοπισμό πτώσεων εκτελείται στο smartphone ενώ στη δεύτερη στο Pebble. Η υλοποίηση στο smartphone έχει το πλεονέκτημα ότι επιτρέπει την εκτέλεση πιο πολύπλοκου κώδικα κάνοντας εκμετάλλευση των δυνατοτήτων ενός σύγχρονου smartphone, έχει όμως το σημαντικότατο μειονέκτημα ότι προϋποθέτει τη μεταφορά των επιταχύνσεων από το Pebble στο smartphone μέσω Bluetooth ανά σύντομα χρονικά διαστήματα ώστε να γίνει η επεξεργασία τους. Αυτό έχει σαν αποτέλεσμα ο εντοπισμός πτώσεων να γίνεται με καθυστέρηση περίπου ενός λεπτού και λόγω της συχνής χρήσης του Bluetooth η διάρκεια της μπαταρίας μειώνεται στις 20 περίπου ώρες. Τα παραπάνω στοιχεία οδήγησαν στην υλοποίηση της δεύτερης εφαρμογής, η οποία έχει μινιμαλιστικό χαρακτήρα, αποτελεί απλοποιημένη μορφή της πρώτης και έχει σαν στόχο να γίνεται αναγνώριση δραστηριοτήτων και εντοπισμός πτώσεων πάνω στο Pebble με όσο το δυνατόν οικονομικότερο τρόπο. Και οι δύο εφαρμογές αναλύονται στη συνέχεια για λόγους πληρότητας όμως τελικώς, η τελευταία κρίθηκε πιο κατάλληλη για τη λειτουργία στο συγκεκριμένο σύστημα και σε αυτή θα δοθεί περισσότερη βάση.

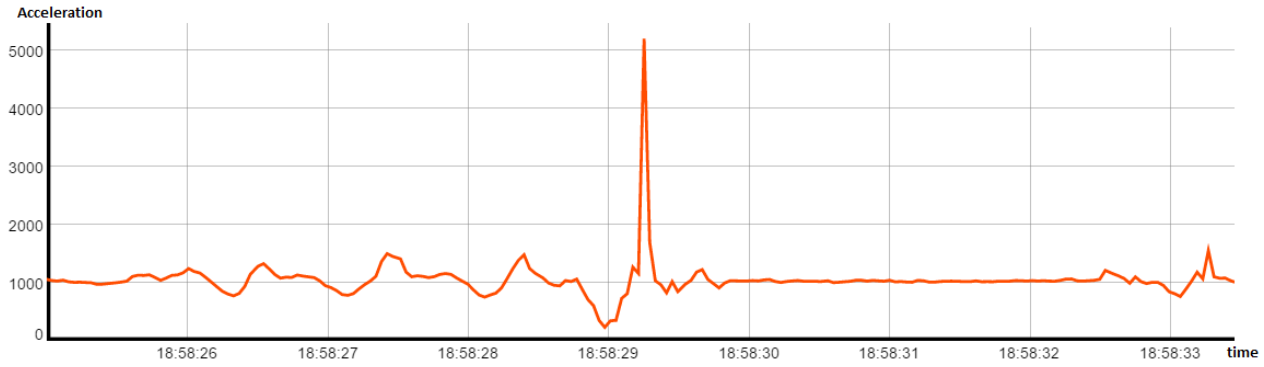
Σύμφωνα με τα στοιχεία των δημοσιεύσεων αλλά και με τα πειραματικά δεδομένα που προέκυψαν από την εφαμοργή συλλογής δεδομένων, οι πτώσεις παρουσιάζουν κάποια κοινά χαρακτηριστικά μεταξύ τους. Εξετάζοντας τη συμπεριφορά του **μέτρου του διανύσματος** των επιταχύνσεων βλέπουμε τα εξής:

1. Η πτώση διαρκεί ένα με δύο δευτερόλεπτα.

2. Σε ένα χρονικό παράθυρο 0.5 - 1 sec πριν από τη στιγμή της πρόσκρουσης ξεκινάει το άτομο που πέφτει να κάνει ελεύθερη πτώση. Το διάνυσμα της επιτάχυνσης παίρνει την ελάχιστη τιμή του λίγο πριν την πρόσκρουση. Ιδανικά αυτή η τιμή είναι 0g, στην πραγματικότητα όμως και σύμφωνα με τις μετρήσεις η επιτάχυνση φτάνει κοντά στο 0.5g.

3. Αμέσως μετά την στιγμή της πρόσκρουσης στο έδαφος η επιτάχυνση παίρνει την μέγιστη τιμή της.

4. Μετά από ένα χρονικό παράθυρο 0.5-1 sec μετά από τη μέγιστη τιμή της επιτάχυνσης το άτομο βρίσκεται για κάποιο χρονικό διάστημα στο έδαφος και η επιτάχυνση παρουσιάζει ιδιαίτερα χαμηλή δραστηριότητα.



Σχήμα 8: Δείγμα πτώσης όπως καταγράφηκε από το επιταχυνσιόμετρο του Pebble σε συχνότητα 25 Hz. Η αρχική δραστηριότητα είναι το περπάτημα και ακολουθεί η πτώση.

Α) Εντοπισμός πτώσεων και αναγνώριση δραστηριοτήτων στο smartphone:

Ο αλγόριθμος για τον εντοπισμό πτώσεων βασίστηκε στις παραπάνω παρατηρήσεις, χρησιμοποιεί thresholds για τη διαδοχική αναγνώριση του μοτίβου που παρουσιάζεται κατά την πτώση και αποτελεί μικρή τροποποίηση του αλγορίθμου των Kozina, Gjoreski, Gams και Luštrek [1]. Η σημαντικότερη διαφορά είναι στο τελευταίο βήμα όπου ο Kozina ελέγχει το orientation του επιταχυνσιομέτρου. Επειδή εμείς χρησιμοποιούμε το επιταχυνσιόμετρο του smartwatch που είναι τοποθετημένο στο χέρι, δεν είναι εφικτό να συμπεράνουμε αν ο χρήστης βρίσκεται σε οριζόντια θέση σε αντίθεση με το επιταχυνσιόμετρο στο στέρνο του οποίου οι άξονες είναι συνεχώς σε σταθερή θέση σε σχέση με το σώμα του χρήστη. Για αυτό το λόγο σαν τελευταίο στάδιο προσπαθούμε να ανιχνεύσουμε ιδιαίτερα χαμηλή δραστηριότητα στις επιταχύνσεις που προκύπτει από την παραμονή του χρήστη στο έδαφος μετά την πτώση.

Συγκεκριμένα ο αλγόριθμος έχει ως εξής:

1) Ελέγχει τα δείγματα διάρκειας περίπου 1.5 λεπτού όπως έρχονται από το Pebble αναζητώντας την υπέρβαση κάποιου high threshold στο μέτρο του διανύσματος της επιτάχυνσης. Σε αυτό το στάδιο προσπαθούμε να εντοπίσουμε τη μέγιστη επιτάχυνση που προκαλείται λόγω της πτώσης. Αν εντοπιστεί τέτοια υπέρβαση τότε υπάρχει πιθανότητα να έχει συμβεί πτώση, οπότε γίνονται κάποιοι επιπλέον έλεγχοι πριν και μετά από το σημείο αυτό.

2) Εξετάζουμε τις τιμές του μέτρου του διανύσματος της επιτάχυνσης που αντιστοιχούν σε 600 ms πριν από την υπέρβαση του high threshold και βρίσκουμε τη μικρότερη. Στη συνέχεια ελέγχουμε αν αυτή η τιμή είναι μικρότερη από κάποιο low threshold. Σε αυτό το στάδιο προσπαθούμε να εντοπίσουμε αν συνέβη ελεύθερη πτώση.

3) Ελέγχουμε αν μετά την υπέρβαση του high threshold οι επιταχύνσεις που λήφθηκαν για χρονική διάρκεια 2 sec αντιστοιχούν σε χαμηλή δραστηριότητα.

Επειδή τα δείγματα φτάνουν από το Pebble ανά περίπου 1.5 λεπτό, χρησιμοποιούνται παράθυρα επικάλυψης που επιτρέπουν στον αλγόριθμο να λειτουργεί συνεχόμενα ακόμα και σε περιπτώσεις που χρειάζονται δεδομένα από το επόμενο ή το προηγούμενο δείγμα επιταχύνσεων. Κάθε φορά που εντοπίζεται πτώση, στέλνεται μήνυμα από το smartphone στον server που περιέχει τις συντεταγμένες του χρήστη και την ηλεκτρονική διεύθυνση στην οποία θα σταλεί ειδοποίηση μέσω email.

Για την αναγνώριση δραστηριοτήτων, εφαρμόζεται ο classifier που προέκυψε από τη μηχανική μάθηση. Υπολογίζονται αρχικά οι τιμές των features που επέλεξε ο J48 στα δείγματα 1.5 λεπτού

καθώς λαμβάνονται από το smartphone και έπειτα ανάλογα με τις τιμές αυτές, το κάθε δείγμα χαρακτηρίζεται ως sleep, mild, moderate ή intense και το συμπέρασμα αποστέλλεται στον server όπου αποθηκεύεται σε κατάλληλη συλλογή στη βάση Mongo.

Β) Εντοπισμός πτώσεων και αναγνώριση δραστηριοτήτων στο Pebble:

Εδώ ο αλγόριθμος εντοπισμού πτώσεων αποτελεί απλοποιημένη μορφή του προηγούμενου έτσι ώστε να λειτουργεί πολύ πιο αποδοτικά πάνω στο ρολόι με το ελάχιστο δυνατό κόστος και βασίζεται και πάλι στον αλγόριθμο των Kozina, Gjoreski, Gams και Luštrek [1]. Το επιταχυνσιόμετρο του Pebble συλλέγει δεδομένα στη συχνότητα των 25 Hz. Η συγκεκριμένη συχνότητα επιλέχθηκε διότι είναι αρκετά μεγάλη ώστε να μπορεί να εντοπιστεί μια πτώση και αρκετά μικρή ώστε να μην επιβαρύνεται η λειτουργία του ρολογιού. Τα δεδομένα γίνονται διαθέσιμα σε έναν buffer μέσω ενός callback σε πακέτα των 25 επιταχύνσεων για κάθε άξονα, κάθε δευτερόλεπτο. Εκμεταλλευόμαστε το παραπάνω γεγονός για εντοπισμό πτώσεων σε πραγματικό χρόνο εκτελώντας κάποια βήματα. Καθένα από αυτά έχει διάρκεια ενός δευτερολέπτου και μεταβάλλει την κατάσταση στην οποία βρίσκεται το σύστημα με σκοπό να εντοπιστεί η πτώση σε περίπτωση που έχουμε πέρασμα διαδοχικά σε 4 συγκεκριμένες καταστάσεις. Εδώ γίνεται χρήση του απλού αθροίσματος απόλυτων τιμών των επιταχύνσεων |x|+|y|+|z| εν αντιθέσει με το μέτρο του διανύσματος της επιτάχυνσης λόγω του ελάχιστου κόστους υπολογισμού. Η ποσότητα αυτή παρουσιάζει παρόμοια συμπεριφορά με το μέτρο του διανύσματος της επιτάχυνσης κατά την πτώση. Οι διαφορές είναι η μεγαλύτερη ευαισθησία και ότι κατά την ακινησία, ανάλογα με το orientation του ρολογιού, το μετρούμενο άθροισμα ηρεμεί σε τιμές από τα 1000 έως 1900 milli-g. Ο τρόπος όμως με τον οποίο χρησιμοποιείται το απλό άθροισμα, ελαχιστοποιεί την επίδραση των παραπάνω ιδιαιτεροτήτων και το συνολικό κέρδος στο κόστος υπολογισμού κρίθηκε σημαντικότερο σε σχέση με την μεγαλύτερη ακρίβεια που θα είχαμε με τη χρήση του μέτρου του διανύσματος των επιταχύνσεων. Ο αλγόριθμος επικεντρώνεται στον εντοπισμό των δύο ουσιαστικότερων χαρακτηριστικών της πτώσης που είναι η υπέρβαση του high threshold και η χαμηλή δραστηριότητα στη συνέχεια όσο ο χρήστης βρίσκεται στο έδαφος. Αποτελείται λοιπόν από τα εξής στάδια:

1) Υπολογισμός σε κάθε πακέτο επιταχύνσεων της διαφοράς του μέγιστου από το ελάχιστο άθροισμα των απόλυτων τιμών των επιταχύνσεων (|x|+|y|+|z|) και έλεγχος για το αν γίνεται υπέρβαση κάποιου threshold αυτής της ποσότητας. Αν ξεπερνιέται αυτό το threshold το σύστημα μπαίνει σε μια κατάσταση πιθανής πτώσης.

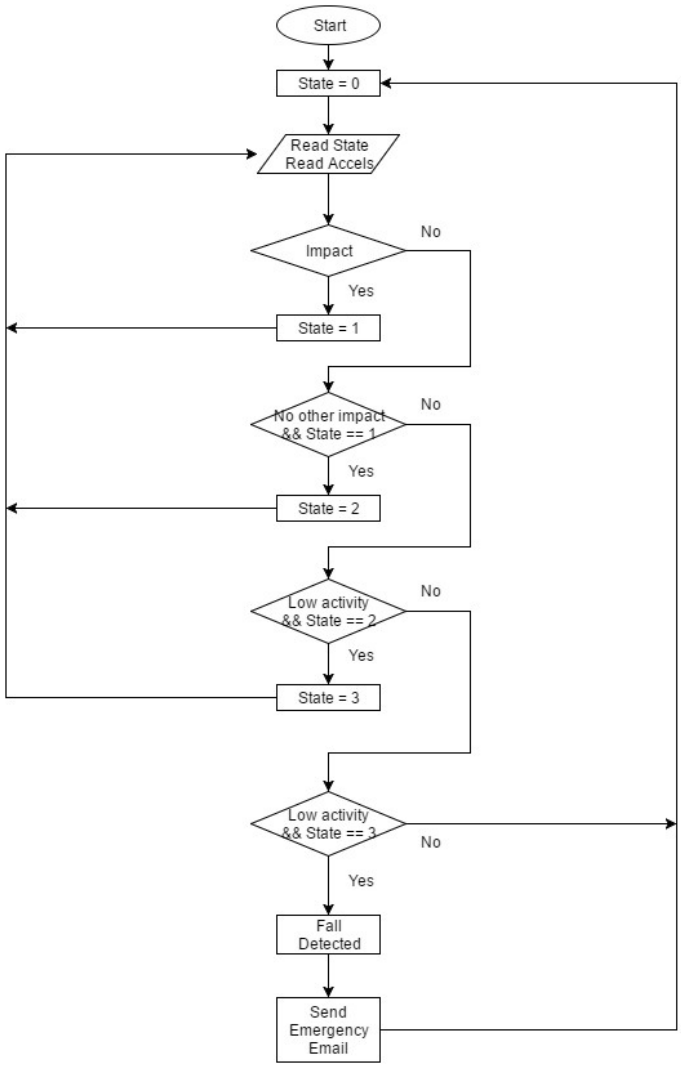
2) Εφόσον το σύστημα βρίσκεται σε κατάσταση πιθανής πτώσης, το επόμενο δευτερόλεπτο, στα νέα δεδομένα που θα γίνουν διαθέσιμα από το callback, ελέγχεται αν αυτή τη φορά δεν ξεπερνιέται το threshold και αν αυτό συμβαίνει το σύστημα μεταβαίνει σε νέα κατάσταση όπου πλέον είναι εξαιρετικά πιθανό να έχει συμβεί πτώση. Αυτό το βήμα έχει σκοπό να απορροφήσει τις διακυμάνσεις του αθροίσματος μετά την πρόσκρουση.

3) Στο τελευταίο στάδιο ελέγχεται για το επόμενα δύο δευτερόλεπτα αν η δραστηριότητα του χρήστη είναι ιδιαίτερα χαμηλή όπως θα προέκυπτε από την παραμονή του στο έδαφος.

Ο παραπάνω αλγόριθμος όπως είναι φυσικό μπορεί να παρουσιάσει false positives το οποίο είναι όμως αποδεκτό. Αυτό που έχει μεγαλύτερη σημασία είναι σε περίπτωση πραγματικής πτώσης να είναι σε θέση να την αντιληφθεί πάντα. Εκ κατασκευής λοιπόν είναι σε θέση να αντιληφθεί το 100% των πτώσεων που προσομοιώθηκαν.

Όσον αφορά την αναγνώριση δραστηριοτήτων εδώ διακρίνουμε μόνο δύο καταστάσεις, asleep και awake κάθε μια από τις οποίες αντιπροσωπεύεται από έναν αριθμό. Δεν χρησιμοποιείται μηχανική μάθηση, αντιθέτως η τεχνική που θα περιγραφεί στηρίζεται στη συχνότητα κινήσεων του χρήστη

και είναι ιδιαίτερα αποτελεσματική. Ο διαχωρισμός των δύο καταστάσεων γίνεται και πάλι με χρήση του απλού αθροίσματος των απόλυτων τιμών των επιταχύνσεων. Συγκεκριμένα σε κάθε πακέτο δεδομένων υπολογίζεται η διαφορά της μέγιστης από την ελάχιστη τιμή του απλού αθροίσματος απόλυτων τιμών των επιταχύνσεων και ανάλογα αν αυτή η διαφορά είναι μικρή ή μεγάλη για το συγκεκριμένο δευτερόλεπτο γίνεται διακίνηση πόντων από τη μια κατάσταση στην άλλη κρατώντας πάντα το άθροισμα των δύο καταστάσεων σταθερό. Κάθε λεπτό γίνεται σύγκριση των πόντων των δύο καταστάσεων και καθορίζεται η κατάσταση του χρήστη από τη νικήτρια. Η μεταφορά των πόντων από τη μια κατάσταση στην άλλη γίνεται με διαφορετική βαρύτητα με σκοπό συχνές κινήσεις του χρήστη να οδηγούν σε γρήγορη μεταβίβαση βαθειά στην κατάσταση awake ενώ όταν βρίσκεται ακίνητος να έχουμε αργή μεταβίβαση στην κατάσταση asleep. Έτσι αποφεύγεται η περίπτωση μικρά χρονικά διαστήματα ακινησίας να καθορίζουν την κατάσταση ως asleep.



Σχήμα 9: Διάγραμμα ροής αλγορίθμου εντοπισμού πτώσεων.

Η εφαρμογή στέλνει δεδομένα στο smartphone μόνο όταν συμβαίνει αλλαγή από τη μια κατάσταση στην άλλη ώστε να ενημερωθεί ο server. Είναι επίσης πλήρως παραμετροποιήσιμη επιτρέποντας στο χρήστη να θέσει τα thresholds και την ευαισθησία του αλγορίθμου για τον εντοπισμό πτώσεων και άλλων παραμέτρων σχετικών με το πόσο γρήγορα επιτρέπεται να μεταβαίνει το σύστημα από

τη μια κατάσταση στην άλλη. Έτσι, η χρήση του Bluetooth είναι η ελάχιστη δυνατή και η εφαρμογή μπορεί να εκτελείται συνεχώς χωρίς να επηρεάζει τη διάρκεια της μπαταρίας.

3.5 Διαδικτυακή εφαρμογή εμφάνισης δραστηριότητας του χρήστη

Η διαδικτυακή εφαρμογή επιτρέπει στο χρήστη να κάνει login στον browser με τα στοιχεία που έχει δώσει στο configuration page της εφαρμογής του Pebble, να διαλέξει μια ημερομηνία και να δει σε σχετική γραφική παράσταση τα δεδομένα που καταγράφηκαν τα οποία είναι είτε η ένταση της δραστηριότητάς του σε sleep, mild, moderate και intense κάθε στιγμή της ημέρας είτε οι ώρες που ήταν asleep και awake ανάλογα με την εφαρμογή που χρησιμοποιήθηκε. Εμφανίζονται επίσης τα ίδια στοιχεία σαν στατιστικά. Στην εφαρμογή αυτή μπορεί να κάνει login οποιοσδήποτε γνωρίζει το username που έχει επιλεχθεί και να ελέγξει τη δραστηριότητα αυτού που χρησιμοποιεί το Pebble ώστε να σιγουρευτεί ότι όλα λειτουργούν όπως πρέπει (πχ ο ηλικιωμένος δεν έχει βγάλει ή απενεργοποιήσει το ρολόι) και να παρακολουθήσει τις συνήθειές του. Μελλοντικά θα μπορούσε να βελτιωθεί το σύστημα ώστε να μελετάει από μόνο του τις συνήθειες του χρήστη και να ενημερώνει όταν παρουσιάζονται τυχόν αποκλίσεις από τη συνηθισμένη δραστηριότητα που θα μπορούσε ενδεχομένως να έχει χαρακτήρα πρόληψης διαφόρων καταστάσεων.

Κεφάλαιο 4. Υλοποίηση

4.1 Συλλογή δεδομένων

4.1.1 Υλοποίηση στο πλαίσιο Pebble.js

Βασικό δομικό στοιχείο για τη δημιουργία διεπαφής χρήστη είναι οι κάρτες. Μια κάρτα είναι ένα είδος παραθύρου που καταλαμβάνει ολόκληρη την οθόνη του Pebble και επιτρέπει την εμφάνιση στοιχείων με έναν προκαθορισμένο τρόπο: Ένας τίτλος στην κορυφή, ένας υπότιτλος ακριβώς από κάτω και έπειτα ένα κυρίως μέρος για μεγαλύτερες παραγράφους. Για την παρούσα εφαρμογή τροποποιήθηκε το template που παρέχεται από το cloudpebble για τη δημιουργία εφαρμογών αυτού του είδους. Αρχικά εμφανίζεται μια εισαγωγική κάρτα που προτρέπει τον χρήστη να πιέσει το επάνω κουμπί του Pebble (up) ώστε να εμφανίσει το μενού επιλογών:

```
var main = new UI.Card({
      title: 'Pebble.js',
      icon: 'images/menu_icon.png',
      subtitle: 'Data collection',
      body: 'Press top button for options',
      subtitleColor: 'indigo', // Named colors
      bodyColor: '#9a0036' // Hex colors
});

main.show();
```

Η επόμενη κάρτα περιλαμβάνει το μενού, το οποίο δίνει τη δυνατότητα με το πάτημα του μεσαίου κουμπιού του Pebble (select) να ξεκινήσει η συλλογή και αποστολή δεδομένων στον εξυπηρετητή.


```
main.on('click', 'up', function(e) {
      var menu = new UI.Menu({
            sections: [{
                  items: [{
                        title: 'Data collection',
                        subtitle: onOff[i]
                        },{
                        title: 'Fall detection',
                        subtitle: 'Disabled'
                  }]
```

```
                }]
        });
        menu.on('select', function(e) {
                if (e.itemIndex===0){
                        if (i===0){
                                i=1;
                                menu.items(0,[{title:'Data collection', subtitle:
onOff[i]}]);
                                flag = true;
                        }
                        else{
                                i=0;
                                menu.items(0, [{title:'Data collection', subtitle:
onOff[i]}]);
                                flag = false;
                        }
                }
        });
        menu.show();
}
```

Στη συνέχεια γίνεται χρήση της μονάδας Accel (module) η οποία λαμβάνει συνεχόμενη ροή δεδομένων από το επιταχυνσιόμετρο του Pebble. Σε αυτή τη λειτουργία, το Pebble συλλέγει επιταχύνσεις σύμφωνα με κάποια προεπιλεγμένη συχνότητα μεταξύ 10Hz, 25Hz, 50Hz και 100Hz, τις τοποθετεί σε έναν πίνακα ο οποίος μόλις συμπληρωθεί, καλείται ένας event handler που αναλαμβάνει την αποστολή των δεδομένων μέσω Bluetooth στο smartphone.

Στον προγραμματιστή παρέχονται οι συναρτήσεις Accel.on('data', callback) και Accel.config(accelConfig). Στην Accel.on('data', callback), το callback δέχεται ένα event που περιλαμβάνει τα ακόλουθα πεδία:

samples: Το πλήθος των δειγμάτων από το επιταχυνσιόμετρο σχετικά με το συγκεκριμένο γεγονός (event). Επιτρεπτές τιμές είναι από 1 έως 25 δείγματα.

accel: Το πρώτο δείγμα (sample) που περιέχει ο πίνακας, το οποίο παρέχεται για συνέπεια.

accels: Όλα τα δείγματα του επιταχυνσιόμετρου σε μορφή πίνακα. Ο πίνακας accels επιτρέπει πρόσβαση στις μεταβλητές x, y, z που αφορούν την τιμή της επιτάχυνσης στον αντίστοιχο άξονα και στη μεταβλητή time που αποτελεί τη χρονική στιγμή σε miliseconds που έγινε η μέτρηση.

Η Accel.config(accelConfig) επιτρέπει τη ρύθμιση παραμέτρων του επιταχυνσιόμετρου. Οι προκαθορισμένες τιμές, οι οποίες και χρησιμοποιήθηκαν στον κώδικα που ακολουθεί, είναι συχνότητα δειγματοληψίας 100Hz και συγκέντρωση 25 δειγμάτων πριν την κλήση του event handler για την αποστολή δεδομένων. Στην παρούσα μορφή της εφαρμογής δεν χρειάστηκε να γίνει χρήση αυτής της συνάρτησης για αλλαγή των προκαθορισμένων τιμών αλλά παρουσιάστηκε για λόγους πληρότητας.

Τέλος, εφόσον έχει ενεργοποιηθεί η συλλογή δεδομένων από το μενού, γίνεται η χρήση του AJAX για την αποστολή τους σε μορφή JSON στον εξυπηρετητή με τη μέθοδο POST.

```
	Accel.on('data', function(e) {
		for(var i=0;i<e.samples;i++){
			myArray.array.push({
				time:JSON.stringify(e.accels[i].time),
				xvalue:JSON.stringify(e.accels[i].x),
				yvalue: JSON.stringify(e.accels[i].y),
				zvalue: JSON.stringify(e.accels[i].z)
			 });
		}
		if (flag) {
			for(i=0;i<e.samples;i++){
				sendData={
					"timestamp":myArray.array[i].time,
					"xvalue":myArray.array[i].xvalue,
					"yvalue":myArray.array[i].yvalue,
					"zvalue":myArray.array[i].zvalue
				};
				 ajax({
					url: 'http://83.212.115.163:8080/pebble',
					method: 'post',
					type: 'json',
					data: sendData,
					crossDomain: true
				 });
			}
		}
		myArray = {"array":[]};
	});
```

4.1.2 Υλοποίηση στα πλαίσια Pebble C και PebbleKit JS

Σε αυτή την προσέγγιση, η εφαρμογή αποτελείται από δύο αρχεία κώδικα. Το πρώτο αποτελείται από κώδικα σε γλώσσα C και ασχολείται με τη διαχείρηση του επιταχυνσιομέτρου, το διάβασμα και την αποθήκευση των μετρήσεων σε έναν buffer με κατάλληλη κωδικοποίηση (ώστε να καταλαμβάνουν όσο το δυνατόν λιγότερη μνήμη) και την διαχείρηση του Bluetooth για την αποστολή των δεδομένων στο smartphone. Επίσης περιλαμβάνει λίστα δραστηριοτήτων (ADL) με την οποία δίνεται η δυνατότητα στο χρήστη να χαρακτηρίσει τα δεδομένα ενός χρονικού διαστήματος. Το δεύτερο αρχείο είναι γραμμένο σε γλώσσα JavaScript και ο σχετικός κώδικας εκτελείται στο smartphone. Αυτό αναλαμβάνει τη λήψη των δεδομένων, την αποκωδικοποίησή τους και την αποστολή τους στον εξυπηρετητή. Τέλος, το PebbleKit JS έδωσε τη δυνατότητα δημιουργίας configuration page για την εφαρμογή στην οποία ο χρήστης μπορεί να συμπληρώσει μια συμβολοσειρά η οποία θα χρησιμεύσει αργότερα σαν Login στη διαδικτυακή εφαρμογή για να μπορεί να δει τα δεδομένα που τον αφορούν σε γραφικές παραστάσεις. Ακολουθούν κάποιες σημαντικές λεπτομέρειες υλοποίησης και τα κυριότερα σημεία του κώδικα με επεξήγησή τους.

Κώδικας C:

Ο κώδικας στη C έχει σε όλες τις εφαρμογές την ίδια δομή όπως έχει ήδη αναφερθεί. Περιλαμβάνει μια συνάρτηση main() η οποία καλεί την init() όπου γίνεται εγγραφή στα διάφορα Event Services που θα χρησιμοποιηθούν. Στη συνέχεια εισερχόμαστε στο βρόχο και περιμένουμε από το σύστημα να προκύψουν γεγονότα που θα αναλάβουν να διαχειριστούν οι κατάλληλοι handlers που ορίστηκαν, και τέλος κάνουμε απεγγραφή απο τα Event Services στην συνάρτηση deinit().

```
int main( void ) {
	init();
	app_event_loop();
	deinit();
}
```

Μέσα στη συνάρτηση static void init(void) αρχικά γίνεται ορισμός των παραθύρων που θα χρησιμοποιηθούν στην εφαρμογή και των αντίστοιχων handlers που διαχειρίζονται την μεταξύ τους εναλλαγή και τη συμπεριφορά τους κατά το πάτημα των κουμπιών του Pebble. Η διαδικασία θεωρείται τετριμμένη. Αυτό που έχει ενδιαφέρον είναι η εγγραφή στην υπηρεσία του επιταχυνσιόμετρου και του AppMessage που χρησιμοποιήθηκε για τη μεταφορά των δεδομένων του επιταχυνσιόμετρου από το Pebble στο κινητό.

Όσον αφορά το επιταχυνσιόμετρο, με την εντολή accel_data_service_subscribe(NUM_SAMPLES, data_handler) κάνουμε εγγραφή στην αντίστοιχη υπηρεσία. Η πρώτη παράμετρος ορίζει το μέγεθος του πίνακα AccelData που αναφέρθηκε στην εισαγωγή. Η τιμή αυτή είναι ίση με 25 που αποτελεί και τη μέγιστη δυνατή τιμή. Η δεύτερη παράμετρος (data_handler) ορίζει τον handler που θα κληθεί από το σύστημα - όσο βρισκόμαστε στον βρόχο - και μας επιτρέπει να διαχειριστούμε τις εγγεγραμμένες τιμές. Το event που κάνει το σύστημα να ενεργοποιήσει την κλήση του handler είναι η συμπλήρωση του πίνακα AccelData. Με την εντολή accel_service_set_sampling_rate(ACCEL_SAMPLING_25HZ) θέτουμε τον ρυθμό δειγματοληψίας στα 25 Hz. Στη συνέχεια ορίζονται οι handlers που σχετίζονται με το AppMessage και τις 4 δυνατές καταστάσεις που μπορούν να προκύψουν κατά την ανταλλαγή μηνυμάτων με το κινητό, δηλαδή επιτυχή ή αποτυχή λήψη και επιτυχή ή αποτυχή αποστολή μηνύματος από/στο PebbleKit JS. Οι σημαντικότεροι είναι οι handlers για επιτυχή λήψη και αποστολή ενώ οι άλλοι δύο χρησιμοποιήθηκαν για λόγους debugging. Τέλος ορίζεται το μέγεθος των buffers για τα εισερχόμενα και εξερχόμενα μηνύματα. Οι επιλεχθείσες τιμές είναι inbox_size = 128 και outbox_size = 6000. Το μεγεθος του Inbox είναι αδιάφορο και πρέπει να είναι όσο μικρότερο γίνεται διότι το smartphone περιορίζεται απλά στην αποστολή μηνυμάτων επιβεβαίωσης και ετοιμότητας της JavaScript ενώ το Outbox πρέπει να είναι όσο μεγαλύτερο γίνεται ώστε να μπορούμε να στείλουμε μεγάλο όγκο δεδομένων σε ένα μήνυμα. Οι παραπάνω τιμές λοιπόν, έχουν επιλεγεί προσεκτικά ώστε να επιτρέπουν τη καλύτερη δυνατή χρήση της διαθέσιμης μνήμης.

```
#define NUM_SAMPLES 25

//...

static void init(void) {

	//...

	// Subscribe to batched data events
	accel_data_service_subscribe(NUM_SAMPLES, data_handler);
	accel_service_set_sampling_rate(ACCEL_SAMPLING_25HZ);

	// Register AppMessage handlers
	app_message_register_inbox_received(in_received_handler);
	app_message_register_inbox_dropped(in_dropped_handler);
	app_message_register_outbox_failed(out_failed_handler);
	app_message_register_outbox_sent(out_sent_handler);
```

```
        // Initialize AppMessage inbox and outbox buffers
        // with a suitable size
        const int inbox_size = 128;
        const int outbox_size = 6000;
        app_message_open(inbox_size, outbox_size);

        //...

}
```

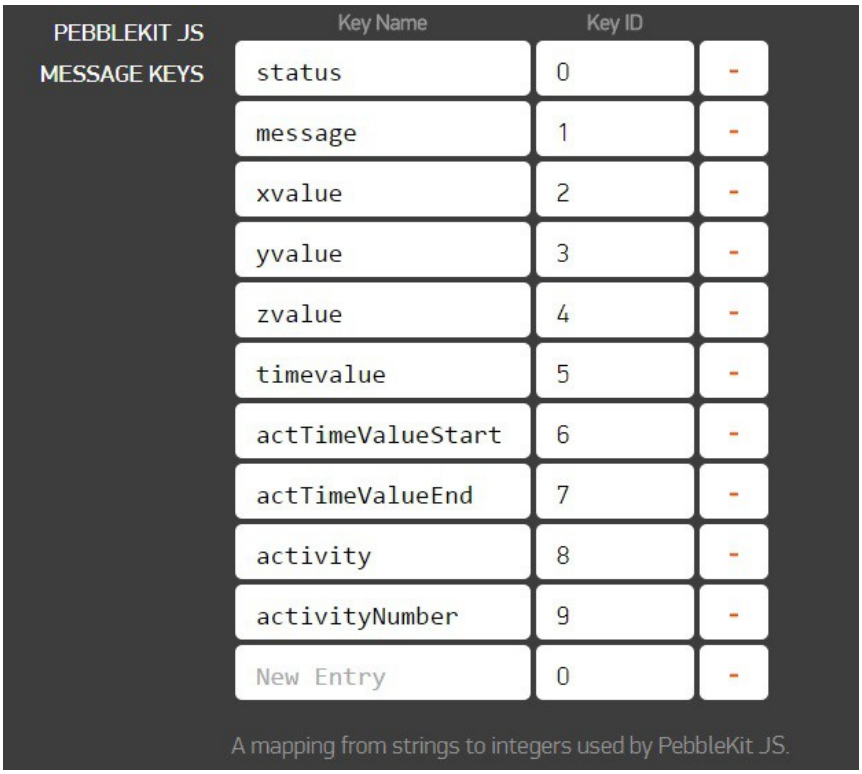
Όπως έχει αναφερθεί στην εισαγωγή, το AppMessage χρησιμοποιεί τη δομή λεξικού για την μεταφορά δεδομένων στο κινητό σε ζεύγη key/value. Σε κάθε μήνυμα AppMessage, το value που αντιστοιχεί σε ένα συγκεκριμένο key δεν μπορεί να αλλάξει. Έτσι, καθίσταται απαραίτητο το value να είναι κάποιας μορφής πίνακας. Σε αντίθετη περίπτωση θα έπρεπε είτε να στείλουμε ένα μήνυμα AppMessage για κάθε τιμή, χρησιμοποιώντας το ίδιο key, το οποίο θα οδηγούσε σε αποστολή μεγάλου πλήθους μηνυμάτων σε σύντομο χρονικό διάστημα, είτε να φτιάξουμε προγραμματιστικά διαφορετικά keys για κάθε διαφορετική τιμή που θέλουμε να στείλουμε σε ένα εννιαίο μήνυμα, πράγμα που είναι αδύνατο διότι τα keys πρέπει να είναι αυστηρά καθορισμένα και γνωστά από πριν και στις δύο πλευρές Pebble - κινητό. Από τους επιτρεπτούς τύπους για το value λοιπόν, η μόνη μορφή πίνακα που είναι διαθέσιμη είναι ο πίνακας bytes. Όπως είναι γνωστό, το επιταχυνσιόμετρο του Pebble δίνει τετραψήφιες τιμές (+- 4000 milli-G). Η βέλτιστη αναπαράσταση κάθε τιμής λοιπόν, είναι σε 2 Bytes. Στο λιγότερο σημαντικό byte τοποθετούνται τα δύο τελευταία ψηφία της μέτρησης ενώ στο πιο σημαντικό byte κωδικοποιούνται τα δύο πρώτα ψηφία μαζί με το πρόσημο. Για την παραπάνω κωδικοποίηση δημιουργήσαμε τη δομή byteForm. Σημειωτέον ότι σε περίπτωση που η ακρίβεια δεν θεωρηθεί σημαντική, θα μπορούσαμε να παραλείψουμε το λιγότερο σημαντικό Byte και να κωδικοποιήσουμε κάθε τιμή σε ένα μόνο Byte κρατώντας μόνο τα δύο αριστερότερα ψηφία της. Όσον αφορά τα timestamps, δηλαδή τις χρονικές στιγμές στις οποίες έγινε η κάθε μέτρηση, το επιταχυνσιόμετρο τις δίνει σε μορφή Unix time, δηλαδή σε δευτερόλεπτα που έχουν περάσει από 00:00:00 UTC της 1ης Ιανουαρίου του 1970. Οι τιμές αυτές είναι τύπου unsigned long long int και καταλαμβάνουν 13 ψηφία. Για την κωδικοποίηση ενός timestamp λοιπόν, δημιουργήθηκε η δομή timeByteForm που χρησιμοποιεί 7 bytes. Επομένως, για το πέρασμα των timestamps σε byte array χρειαζόμαστε 7 συνεχόμενες θέσεις πίνακα και ακολουθήθηκε η ίδια λογική με νωρίτερα, δηλαδή αριστερά στον πίνακα βρίσκονται τα περισσότερο σημαντικά bytes και δεξιά τα λιγότερο σημαντικά. Ακολουθούν οι δομές και οι συναρτήσεις που υλοποιήθηκαν για τη μετατροπή των μετρήσεων σε μορφή κατάλληλη για αποθήκευση σε byte array.

```
struct byteForm {
        uint8_t first_half;
        uint8_t second_half;
};

struct timeByteForm {
        uint8_t byte_0;
        uint8_t byte_1;
        uint8_t byte_2;
        uint8_t byte_3;
        uint8_t byte_4;
        uint8_t byte_5;
        uint8_t byte_6;
};
```

```
//Converts accelerometer values to bytes
struct byteForm convert_xyz_to_bytes(int value){

	struct byteForm split;

	//OR operation with 128 is equal to making MSB = 1
	//which represents the negative sign
	if(value<0)
		split.first_half = (abs(value)/100) | 128;
	else
		split.first_half = abs(value)/100;

	split.second_half = abs(value)%100;

	return split;
}

//Converts timestamps to bytes.
struct timeByteForm convert_time_to_bytes(int value1,int value2){

	struct timeByteForm split;
	value2 = abs(value2);
	value1 = abs(value1);

	split.byte_0 = value2 % 100;
	split.byte_1 = (value2/100) % 100;
	split.byte_2 = (value2/10000) % 100;
	split.byte_3 = (value2/1000000);
	split.byte_4 = value1 % 100;
	split.byte_5 = (value1/100) % 100;
	split.byte_6 = (value1/10000);

	return split;
}

struct byteForm temp;
struct timeByteForm temp2, referenceTimestamp, choiceTimestamp;
```

Συνολικά χρησιμοποιήθηκαν τέσσερις byte arrays για την μεταφορά των δεδομένων. Οι XBytes, YBytes και ZBytes χρησιμοποιήθηκαν για την αποστολή των επιταχύνσεων που αντιστοιχούν στους άξονες x, y και z, όπου κάθε τιμή καταλαμβάνει δύο συνεχόμενες θέσεις πίνακα. Τα μεγέθη αυτών των πινάκων είναι 1800 άρα κάθε ένας περιέχει 900 τιμές επιταχύνσεων. Σημαντικό στοιχείο για τα timestamps είναι το γεγονός ότι σε κάθε 25άδα μετρήσεων, το χρονικό διάστημα μεταξύ τους είναι σταθερό και ανάλογο της συχνότητας δειγματοληψίας που επιλέχθηκε (40 ms για συχνότητα 25 Hz). Για να μην υπάρχει πλεονασμός πληροφορίας αρκεί η αποστολή στο κινητό, μόνο του πρώτου timestamp κάθε 25άδας. Στη συνέχεια στην πλευρά του κινητού μπορούμε έχοντας γνώση της συχνότητας δειγματοληψίας να ανασυνθέσουμε όλες τις χρονικές τιμές πρωτού τις στείλουμε στον εξυπηρετητή. Έτσι, για τα timestamps χρησιμοποιήθηκε ο byte array TimeBytes ο οποίος έχει μέγεθος 252 (900 μετρήσεις / 25 * 7 θέσεις πίνακα). Οι παραπάνω byte arrays στάλθηκαν μέσω AppMessage σε ζεύγη της μορφής X_KEY / XBytes, Y_KEY / YBytes, Z_KEY / ZBytes και TIME_KEY / TimeBytes όπου οι τιμές των keys ορίστηκαν στον κώδικα της C σε μια δομή απαρίθμησης. Οι ίδιες τιμές των keys καθώς και τα ονόματα με τα οποία θα αντιστοιχούν στη JavaScript πρέπει να δηλωθούν και στην πλευρά του κινητού το οποίο γίνεται μέσω των settings

στο CloudPebble. Από τα παραπάνω ορίζεται το συνολικό μέγεθος κάθε μηνύματος AppMessage περίπου ίσο με 5700 Bytes, το οποίο είναι μέσα στα όρια ορίστηκαν από το outbox size.



Σχήμα 10: Αντιστοιχία των κλειδιών του appmessage σε αριθμούς από τα settings του CloudPebble.

Ο data_handler καλείται από το σύστημα κάθε φορά που συμπληρώνονται 25 τιμές στον πίνακα τύπου AccelData. Αυτό που κάνει ο κώδικας που έχουμε γράψει μέσα στον handler είναι να συγκεντρώνει όλες τις μετρήσεις στους αντίστοιχους byte arrays που αναφέρθηκαν νωρίτερα, διατηρώντας και αυξάνοντας έναν counter μέχρι να γεμίσουν αυτοί οι πίνακες. Μόλις γίνει αυτό, τους στέλνει στο κινητό σε μορφή λεξικού με χρήση του AppMessage. Ακολουθούν τα σχετικά σημεία του κώδικα.

```
#define BUFFER_SIZE 900
#define XYZ_BYTE_ARRAY_SIZE BUFFER_SIZE*2
#define TIME_BYTE_ARRAY_SIZE BUFFER_SIZE/NUM_SAMPLES*7

//...

//Byte arrays
uint8_t XBytes[XYZ_BYTE_ARRAY_SIZE];
uint8_t YBytes[XYZ_BYTE_ARRAY_SIZE];
uint8_t ZBytes[XYZ_BYTE_ARRAY_SIZE];
uint8_t TimeBytes[TIME_BYTE_ARRAY_SIZE];

//...

//create a dictionary
```

```
DictionaryIterator *iter;
int counter = 0;

//...

//Definition of the keys used by AppMessage
enum {
	STATUS_KEY = 0,
	MESSAGE_KEY = 1,
	X_KEY = 2,
	Y_KEY = 3,
	Z_KEY = 4,
	TIME_KEY = 5,
	TIME_VALUE_START = 6,
	TIME_VALUE_END = 7,
	ACTIVITY = 8,
	ACTIVITY_NUMBER = 9
};

//...

static void data_handler(AccelData *data, uint32_t num_samples) {

	//javascript is ready and we can start gathering data
	if((js_flag)&&(data_col)){

		for(int i=0;i<NUM_SAMPLES;i++){

			temp = convert_xyz_to_bytes(data[i].x);
			XBytes[2*counter] = temp.first_half;
			XBytes[2*counter+1] = temp.second_half;

			temp = convert_xyz_to_bytes(data[i].y);
			YBytes[2*counter] = temp.first_half;
			YBytes[2*counter+1] = temp.second_half;

			temp = convert_xyz_to_bytes(data[i].z);
			ZBytes[2*counter] = temp.first_half;
			ZBytes[2*counter+1] = temp.second_half;

			// store only the first timestamp
			if(i==0){

				temp2 = convert_time_to_bytes(data[i].timestamp /
100000000,data[i].timestamp % 100000000);

				TimeBytes[timeCounter] = temp2.byte_6;
				TimeBytes[timeCounter+1] = temp2.byte_5;
				TimeBytes[timeCounter+2] = temp2.byte_4;
				TimeBytes[timeCounter+3] = temp2.byte_3;
				TimeBytes[timeCounter+4] = temp2.byte_2;
				TimeBytes[timeCounter+5] = temp2.byte_1;
				TimeBytes[timeCounter+6] = temp2.byte_0;

				timeCounter+=7;
			}
```

```
                        counter++;

                }

                //When all buffers are full we need to send data to PebbleKitJS.
                if(counter==BUFFER_SIZE){

                        //Begin writing to the Outbox's Dictionary buffer.
                        app_message_outbox_begin(&iter);

                        //Adds a key with a byte array value pair to the dictionary.
                        dict_write_data(iter,X_KEY,&XBytes[0],sizeof(XBytes));
                        dict_write_data(iter,Y_KEY,&YBytes[0],sizeof(YBytes));
                        dict_write_data(iter,Z_KEY,&ZBytes[0],sizeof(ZBytes));
                       dict_write_data(iter,TIME_KEY,&TimeBytes[0],sizeof(TimeBytes)
);

                        //End a series of writing operations to a dictionary. This
//must be called before reading back from the dictionary.
                        dict_write_end(iter);

                        //Sends the outbound dictionary.
                        app_message_outbox_send();
                        APP_LOG(APP_LOG_LEVEL_DEBUG,    "Message    was    sent    to
PebbleKitJS");
                        counter = 0;
                        timeCounter = 0;
                }
        }
}
```

Παράλληλα με τη συλλογή δεδομένων και την αποστολή τους στο κινητό, έχουν υλοποιηθεί handlers που καλούνται όταν ο χρήστης χρησιμοποιεί τα κουμπιά του Pebble. Δίνεται έτσι η δυνατότητα από το αρχικό παράθυρο με πάτημα του επάνω κουμπιού (Up) του Pebble να εμφανιστεί λίστα επιλογών οποιαδήποτε στιγμή. Στις επιλογές ο χρήστης μπορεί να διαλέξει κάποια δραστηριότητα από μια λίστα με ADLs μαζί με την χρονική διάρκειά τους ώστε να γίνει αντιστοίχιση των επιταχύνσεων που λήφθηκαν στο αντίστοιχο χρονικό διάστημα με κάποια δραστηριότητα. Η επιλογή δραστηριότητας, αποθηκεύει απλά έναν ακέραιο αριθμό που αντιστοιχεί στον αύξοντα αριθμό (index) που έχει η συγκεκριμένη δραστηριότητα μέσα στη λίστα. Η παραπάνω διαδικασία εκτελείται από έναν handler που καλεί το σύστημα κατά το πάτημα του μεσαίου κουμπιού select ενώ βρισκόμαστε στο παράθυρο με τη λίστα επιλογών. Στη συνέχεια ο handler αναλαμβάνει να φορτώσει το επόμενο παράθυρο που αφορά την επιλογή χρονικής διάρκειας. Η επιλογή χρονικής διάρκειας αποθηκεύει την χρονική στιγμή που πατήθηκε το κουμπί select ως χρονική στιγμή αναφοράς σε μια δομή timeByteForm με όνομα referenceTimestamp και ανάλογα με την επιλογή, γίνεται αφαίρεση αντίστοιχου αριθμού δευτερολέπτων. Το νέο timestamp που προκύπτει αποθηκεύεται σε μια δεύτερη δομή ίδιου τύπου με όνομα choiceTimestamp. Μπορούμε να αποθηκεύσουμε σε έναν buffer μέχρι 10 δραστηριότητες με τις αντίστοιχες χρονικές διάρκειες πρωτού τις στείλουμε στο κινητό, ενώ χρησιμοποιείται η ίδια κωδικοποίηση για τα timestamps με αυτά του επιταχυνσιομέτρου σε byte arrays.

Προκειμένου να αποφευχθεί πιθανή σύγκρουση δεδομένων εξαιτίας ταυτόχρονης αποστολής προς το κινητό, δραστηριοτήτων και δεδομένων από το επιταχυνσιομέτρο, έχουμε υποχρεώσει τα

δεδομένα επιλογής δραστηριοτήτων, αν υπάρχουν, να αποστέλλονται ακριβώς μετά από την περιοδική αποστολή των δεδομένων του επιταχυνσιομέτρου. Αυτό γίνεται με εκμετάλλευση του out_sent_handler ο οποίος καλείται από το σύστημα κάθε φορά που η πλευρά του κινητού επιβεβαιώνει τη λήψη κάποιου μηνύματος από το Pebble.

Για λόγους λειτουργικότητας της εφαρμογής, έχει προστεθεί στην αρχική οθόνη η εμφάνιση ώρας και η στάθμη της μπαταρίας τα οποία γίνονται με επιπλέον handlers που καλούνται όποτε αλλάζει η ώρα ή μεταβάλλεται η στάθμη της μπαταρίας.

Τέλος, έχουμε τη συνάρτηση deinit() στην οποία απεγγραφόμαστε από όλα τα Event Services που έχουμε χρησιμοποιήσει.

```
static void deinit(void) {

      battery_state_service_unsubscribe();
      accel_data_service_unsubscribe();
      app_message_deregister_callbacks();
      tick_timer_service_unsubscribe();
      window_destroy(third_window);
      window_destroy(first_window);
      window_destroy(s_window);
}
```

Κώδικας JavaScript:

Το PebbleKit JS δίνει στο χρήστη τη δυνατότητα τροποποίησης διάφορων παραμέτρων της εφαρμογής. Για την ενεργοποίηση αυτής της λειτουργίας πρέπει να επιλεχθεί το checkbox Configurable στα settings του CloudPebble. Εφόσον γίνει αυτό, εμφανίζεται εικονίδιο ρυθμίσεων δίπλα στην εφαρμογή μας, στη λίστα εφαρμογών στο companion app Pebble. Πάτημα σε αυτό το εικονίδιο ενεργοποιεί έναν Event Listener, τον οποίο μπορούμε να διαχειριστούμε μέσω του PebbleKit JS. Αυτός ο Event Listener έχει υλοποιηθεί έτσι ώστε να ανοίγει μια ιστοσελίδα (configuration page) που περιέχει μια φόρμα στην οποία ο χρήστης μπορεί να συμπληρώσει κάποιο username και ένα script το οποίο ενεργοποιείται με το πάτημα του κουμπιού submit (SAVE). Αυτό το script αναλαμβάνει να κάνει διαθέσιμο το username στο PebbleKit JS σε μορφή JSON όταν κλείσει η σελίδα όπου στη συνέχεια το αποθηκεύουμε στο localStorage της συσκευής.

```
// Get a handle to the button's HTML element
var submitButton = document.getElementById('submit_button');

// Add a 'click' listener
submitButton.addEventListener('click', function() {
      // Get the config data from the UI elements
      var user = document.getElementById('username_input');
      var email = document.getElementById('email_input');

      // Make a data object to be sent, coercing value types to integers
      var options = {
            'userName': user.value,
            'email': email.value
      };
```

```
        // Determine the correct return URL (emulator vs real watch)
        function getQueryParam(variable, defaultValue) {
            var query = location.search.substring(1);
            var vars = query.split('&');
            for (var i = 0; i < vars.length; i++) {
                var pair = vars[i].split('=');
                if (pair[0] === variable) {
                    return decodeURIComponent(pair[1]);
                }
            }
            return defaultValue || false;
        }
        var return_to = getQueryParam('return_to', 'pebblejs://close#');

        // Encode and send the data when the page closes
        document.location = return_to + encodeURIComponent(
                                JSON.stringify(options) );
    });
```

Κώδικας JavaScript του Configuration Page

```
    //...

    //called when configuration button is pressed on Pebble app
    Pebble.addEventListener('showConfiguration', function() {

        // Open configuration page
        var url = 'http://83.212.115.163/config.html';
        console.log('Showing configuration page: '+ url);
        Pebble.openURL(url);
    });

    //called when configuration page closes
    Pebble.addEventListener('webviewclosed', function(e) {

        // Decode the user's preferences
        var configData = JSON.parse(decodeURIComponent(e.response));
        username = configData.userName;

        //save on mobile's local storage
        localStorage.setItem("username",username);
    });

    //...
```

Απόσπασμα κώδικα JavaScript στο PebbleKit JS

Έχουν δημιουργηθεί επίσης κατάλληλες συναρτήσεις που μετατρέπουν τα δεδομένα των επιταχύνσεων και των timestamps από bytes ξανά πίσω στην αρχική τους μορφή.

```
    //Converts encoded acceleration data back to 4 digit numbers
    function convertBytesToAccel(msB,lsB){
        var temp = msB | 127;
        if (temp!=255)
            return (msB*100 + lsB);
        else{
```

```
                msB = msB & 127;
                return -(msB*100 + lsB);
        }
    }

    //converts timestamps back to 13 digit numbers
    function convertBytesToTime(byte6,byte5,byte4,byte3,byte2,byte1,byte0){
          return (byte6*1000000000000 + byte5*10000000000 + byte4*100000000
                      + byte3*1000000 + byte2*10000 + byte1*100 + byte0);
    }
```

Στην πλευρά του κινητού κατά τη λήψη ενός μηνύματος appmessage (ζεύγη key/value), για να διαχωρίσουμε αν το μήνυμα περιλαμβάνει επιταχύνσεις ή δραστηριότητες ελέγχουμε αν υπάρχουν δεδομένα στο αντίστοιχο key. Στην περίπτωση των επιταχύνσεων, ανακατασκευάζονται οι τιμές με τις συναρτήσεις που παρουσιάστηκαν πιο πάνω και αποθηκεύονται σε πίνακες. Αυτοί οι πίνακες συγκεντρώνουν δεδομένα από πολλαπλά μηνύματα appMessage μέχρι να συμπληρωθεί κάποιος αριθμός numberOfMessages ο οποίος καθορίζει κάθε πότε θα στέλνει το κινητό δεδομένα στον εξυπηρετητή. Στη συνέχεια οι πίνακες μπαίνουν σε ένα αντικείμενο JSON sendData και αποστέλλονται στον εξυπηρετητή με τη μέθοδο POST με τη χρήση XMLHttpRequest. Ο εξυπηρετητής αναλαμβάνει την αποθήκευση των δεδομένων στην βάση Mongo σε μορφή BSON. Χρησιμοποιήθηκε η ίδια συλλογή με αυτή που είχε χρησιμοποιηθεί στην πρώτη εφαρμογή στο Pebble.js αφού τροποποιήθηκε πρώτα, έτσι ώστε οι εγγραφές x, y, z και timestamp να είναι πλέον πίνακες integer και long αντί για απλές εγγραφές των ίδιων τύπων. Επίσης προστέθηκε το πεδίο user τύπου string ώστε να γίνεται διάκριση από ποιον χρήστη προέρχονται τα δεδομένα και τα πεδία starttime και endtime τύπου long που αντιπροσωπεύουν απλά το πρώτο και το τελευταίο timestamp που περιέχει ο αντίστοιχος πίνακας του συγκεκριμένου εγγράφου και χρησιμεύουν στη διευκόλυνση πραγματοποίησης κάποιων queries για τη γραφική απεικόνιση των δεδομένων ανά ημέρα.

```
    var buffer_size = 900;
    // Changes with sampling rate
    // 10 Hz : ms = 100 | 25 Hz : ms = 40 | 50 Hz : ms = 20 | 100 Hz : ms = 10
    var ms = 40;
    var time_samples = buffer_size / 25;
    var xarray = [];
    var yarray = [];
    var zarray = [];
    var timearray = [];
    var startarray = [];
    var endarray = [];
    var activityarray = [];
    var sendData = {};
    var username = localStorage.getItem("username");
    var numberOfMessages = 0;
    var request = new XMLHttpRequest();

    //...

    // Called when incoming message from the Pebble is received
    Pebble.addEventListener("appmessage", function(e) {

          var start_temp;
          var end_temp;
```

```
	/*** MESSAGE CONTAINS ACCELERATIONS ***/

	// We are currently only checking the "timevalue" appKey defined in
//appinfo.json/Settings
	//if timevalue is defined the appmessage contains accelerations
	if(e.payload.timevalue !== undefined){

		numberOfMessages++;

		//save accelerations into the corresponding arrays.
		for(var i=0;i<buffer_size;i++){

			xarray.push(convertBytesToAccel(
			e.payload.xvalue[2*i],e.payload.xvalue[2*i+1]));

			yarray.push(convertBytesToAccel(
			e.payload.yvalue[2*i],e.payload.yvalue[2*i+1]));

			zarray.push(convertBytesToAccel(
			e.payload.zvalue[2*i],e.payload.zvalue[2*i+1]));
		}

		//save timestamps into timearray.
		for(i=0;i<time_samples;i++){

			var temp = convertBytesToTime(e.payload.timevalue[7*i],
			     e.payload.timevalue[7*i+1],e.payload.timevalue[7*i+2],
			     e.payload.timevalue[7*i+3],e.payload.timevalue[7*i+4],
			     e.payload.timevalue[7*i+5],e.payload.timevalue[7*i+6]);

			for(var j=0;j<25;j++){
			       timearray.push(temp);
			       temp += ms;
			}
		}

		//when we have gathered enough data send it to server.
		//numberOfMessages = 120 is equal to sending
		//data to server almost every two hours.
		if(numberOfMessages==120){
			sendData={"timestamp":timearray,
			           "user":username,"xvalue":xarray,
			           "yvalue":yarray,"zvalue":zarray,
			          "starttime":timearray[0],
			           "endtime":timearray[timearray.length-1]};
			request.open('POST','http://83.212.115.163:8080/pebble');
			request.setRequestHeader('Content-Type',
			                         'application/json; charset=utf-8');
			request.send(JSON.stringify(sendData));

			//reset arrays and variables to initial values.
			xarray.length = 0;
			yarray.length = 0;
			zarray.length = 0;
			timearray.length = 0;
```

```
                        numberOfMessages = 0;

                }

        }


        /*** MESSAGE CONTAINS ACTIVITIES ***/
        //...
});
```

Στην περίπτωση των δραστηριοτήτων, λαμβάνεται ένας πίνακας integer που αντιστοιχεί σε κάποιο είδος δραστηριότητας και δύο πίνακες από timestamps αρχής και τέλους σε μορφή bytes που οριοθετούν τη διάρκεια της αντίστοιχης δραστηριότητας. Γίνεται στη συνέχεια αντιστοίχιση των δραστηριοτήτων με το όνομά τους και μετατροπή των timestamps από bytes σε κανονική μορφή όμοια με πριν, τοποθετούνται σε δομή JSON sendData και αποστέλλονται με τη μέθοδο POST στον εξυπηρετητή ο οποίος τις αποθηκεύει στη βάση δεδομένων σε μια νεά συλλογή.

```
// Called when incoming message from the Pebble is received
Pebble.addEventListener("appmessage", function(e) {

        var start_temp;
        var end_temp;

        /*** MESSAGE CONTAINS ACCELERATIONS ***/

        //...

        //if actTimeValueStart is defined the appmessage contains activities
        if(e.payload.actTimeValueStart !== undefined){

            //...

            for(var k=0;k<e.payload.activityNumber;k++){
                start_temp=convertBytesToTime( e.payload.actTimeValueStart[7*k],
                                               e.payload.actTimeValueStart[7*k+1],
                                               e.payload.actTimeValueStart[7*k+2],
                                               e.payload.actTimeValueStart[7*k+3],
                                               e.payload.actTimeValueStart[7*k+4],
                                               e.payload.actTimeValueStart[7*k+5],
                                              e.payload.actTimeValueStart[7*k+6]);

                end_temp=convertBytesToTime( e.payload.actTimeValueEnd[7*k],
                                             e.payload.actTimeValueEnd[7*k+1],
                                             e.payload.actTimeValueEnd[7*k+2],
                                             e.payload.actTimeValueEnd[7*k+3],
                                             e.payload.actTimeValueEnd[7*k+4],
                                             e.payload.actTimeValueEnd[7*k+5],
                                             e.payload.actTimeValueEnd[7*k+6]);
                startarray.push(start_temp);
                endarray.push(end_temp-5000);
                switch(e.payload.activity[k]){

                    case 0:
                          activityarray.push("Mild");
```

```
                                break;

                        case 1:
                                activityarray.push("Moderate");
                                break;

                        case 2:
                                activityarray.push("Intense");
                                break;

                        // case 3 to case 17
                        //...
                    }
                }

                //send tagged activities and corresponding durations to server
                sendData={"user":username, "activity":activityarray,
                          "start":startarray, "end":endarray};
                 request = new XMLHttpRequest();
                 request.open('POST','http://83.212.115.163:8080/activity');
                 request.setRequestHeader('Content-Type', 'application/json;
                                                                charset=utf-8');
                 request.send(JSON.stringify(sendData));

                 //reset arrays.
                 startarray.length = 0;
                 endarray.length = 0;
                 activityarray.length = 0;
        }

    });
```

4.2 Εξυπηρετητής

Ο εξυπηρετητής έχει υλοποιηθεί σε λειτουργικό σύστημα Linux Ubuntu 14.04.4 LTS και φιλοξενείται στον "Ωκεανό". Χρησιμοποιήθηκε το πλάισιο εφαρμογών Spring και η γλώσσα Java.

4.2.1 Mongo Repositories

Η κλάση MongoRepository επιτρέπει στον εξυπηρετητή λειτουργίες πρόσβασης δεδομένων πάνω σε συλλογές της βάσης Mongo. Κάθε τέτοια λειτουργία ενεργοποιεί κάποιο ερώτημα (query) στη βάση και η υλοποίησή τους στο Spring είναι πολύ εύκολη, καθώς αφορά απλά την δήλωση αντίστοιχων μεθόδων χρησιμοποιώντας τα ονόματα οντοτήτων που υπάρχουν στο Repository που θέλουμε να γίνει η αναζήτηση σε συνδυασμό με λογικές εκφράσεις και λέξεις κλειδιά. Ο ακόλουθος πίνακας δείχνει τις λέξεις κλειδιά που μπορούν να χρησιμοποιηθούν και τη σημασία τους:

Keyword	Sample	Logical result
After	findByBirthdateAfter(Date date)	{"birthdate" : {"$gt" : date}}
GreaterThan	findByAgeGreaterThan(int age)	{"age" : {"$gt" : age}}

GreaterThanEqual	findByAgeGreaterThanEqual(int age)	{"age" : {"$gte" : age}}
Before	findByBirthdateBefore(Date date)	{"birthdate" : {"$lt" : date}}
LessThan	findByAgeLessThan(int age)	{"age" : {"$lt" : age}}
LessThanEqual	findByAgeLessThanEqual(int age)	{"age" : {"$lte" : age}}
Between	findByAgeBetween(int from, int to)	{"age" : {"$gt" : from, "$lt" : to}}
In	findByAgeIn(Collection ages)	{"age" : {"$in" : [ages…]}}
NotIn	findByAgeNotIn(Collection ages)	{"age" : {"$nin" : [ages…]}}
IsNotNull, NotNull	findByFirstnameNotNull()	{"firstname" : {"$ne" : null}}
IsNull, Null	findByFirstnameNull()	{"firstname" : null}
Like,StartingWith,EndingWith	findByFirstnameLike(String name)	{"firstname" : name} (name as regex)
Containing on String	findByFirstnameContaining(String name)	{"firstname" : name} (name as regex)
NotContaining on String	findByFirstnameNotContaining(String name)	{"firstname" : { "$not" : name}} (name as regex)
Containing on Collection	findByAddressesContaining(Address address)	{"addresses" : { "$in" : address}}
NotContaining on Collection	findByAddressesNotContaining(Address address)	{"addresses" : { "$not" : { "$in" : address}}}
Regex	findByFirstnameRegex(String firstname)	{"firstname" : {"$regex" : firstname }}
(No keyword)	findByFirstname(String name)	{"firstname" : name}
Not	findByFirstnameNot(String name)	{"firstname" : {"$ne" : name}}
Near	findByLocationNear(Point point)	{"location" : {"$near" : [x,y]}}
Near	findByLocationNear(Point point, Distance max)	{"location" : {"$near" : [x,y], "$maxDistance" : max}}
Near	findByLocationNear(Point point, Distance min, Distance max)	{"location" : {"$near" : [x,y], "$minDistance" : min, "$maxDistance" : max}}
Within	findByLocationWithin(Circle circle)	{"location" : {"$geoWithin" : {"$center" : [[x, y], distance]}}}
Within	findByLocationWithin(Box box)	{"location" : {"$geoWithin" : {"$box" : [[x1, y1], x2, y2]}}}
IsTrue, True	findByActiveIsTrue()	{"active" : true}

IsFalse, False	findByActiveIsFalse()	{"active" : false}
Exists	findByLocationExists(boolean exists)	{"location" : {"$exists" : exists }}

Πίνακας 5: Λέξεις κλειδιά του Spring Framework για τη δημιουργία queries στη βάση δεδομένων.

Έτσι για παράδειγμα η αναζήτηση στη συλλογή activity γίνεται με μεθόδους που έχουν υλοποιηθεί με αντίστοιχο τρόπο μέσα στον κώδικα του αντίστοιχου repository όπως φαίνεται και στον ακόλουθο κώδικα.

"ActivityRepository.java"

```
package pebble;

import java.util.List;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource(collectionResourceRel = "activity", path = "activity")
public interface ActivityRepository extends MongoRepository<Activity, String> {

	// method 1
	List<Activity> findByUser(@Param("user") String user);

	// method 2
	List<Activity> findByUserAndStartGreaterThanEqualAndEndLessThan(
	    @Param("user") String user, @Param("start") long from,
	    @Param("end") long to);
}
```

Η "method 1" επιτρέπει την εύρεση των δραστηριοτήτων ενός συγκεκριμένου χρήστη γενικά και η "method 2" την εύρεση των δραστηριοτήτων ενός συγκεκριμένου χρήστη που ανήκουν σε ένα συγκεκριμένο χρονικό διάστημα πάνω στη συλλογή "activity". Με παρόμοιο τρόπο έχουν υλοποιηθεί οι μέθοδοι για αναζήτηση και στα υπόλοιπα repositories.

Για την υλοποίηση της κάθε συλλογής δεδομένων χρειάζεται η δημιουργία μιας αντίστοιχης κλάσης όπου ορίζονται οι τύποι δεδομένων και οι μεταβλητές που θα χρησιμοποιεί μαζί με τις αντίστοιχες μεθόδους get και set όπως φαίνεται παρακάτω.

"Activity.java"

```
package pebble;

import org.springframework.data.annotation.Id;

public class Activity {

	@Id private String id;

	private String user;
	private String[] activity = new String[10];
```

```
	private long[] start = new long[10];
	private long[] end = new long[10];;
	private String comment;


	public String getUser(){
		return user;
	}
	public void setUser(String user){
		this.user = user;
	}


	public String[] getActivity(){
		return activity;
	}
	public void setActivity(String[] activity){
		this.activity = activity;
	}


	public long[] getStart(){
		return start;
	}
	public void setStart(long[] start){
		this.start = start;
	}


	public long[] getEnd(){
		return end;
	}
	public void setEnd(long[] end){
		this.end = end;
	}


	public String getComment(){
		return comment;
	}

	public void setComment(String comment){
		this.comment = comment;
	}

}
```

4.2.2 Αποθήκευση στη Mongo DB

Η αποθήκευση στα repositories γίνεται από τον κώδικα του εξυπηρετητή αυτόματα μέσω ReST οπότε δεν χρειάστηκε να επέμβουμε κάπως.

4.2.3 CORS filter

```
package pebble;
```

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Component;

@Component
public class SimpleCORSFilter implements Filter {

    public void doFilter(ServletRequest req, ServletResponse res,
    FilterChain  chain) throws IOException, ServletException {
        HttpServletResponse response = (HttpServletResponse) res;
        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Methods", "POST, GET,
                                                            OPTIONS, DELETE");
        response.setHeader("Access-Control-Max-Age", "3600");
        response.setHeader("Access-Control-Allow-Headers", "Origin,
                                    X-Requested-With, Content-Type, Accept");
        chain.doFilter(req, res);
    }
    public void init(FilterConfig filterConfig) {}
    public void destroy() {}
}
```

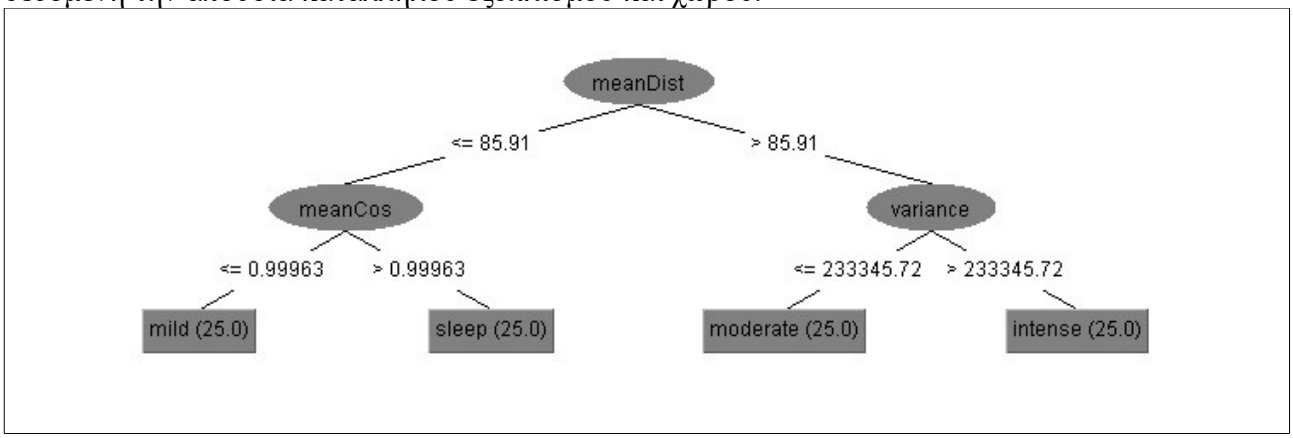
4.3 Μηχανική μάθηση στο λογισμικό Weka

Το Weka (Waikato Enironment for Knowledge Analysis) είναι ένα λογισμικό γραμμένο σε Java που περιέχει εργαλεία απεικόνισης και αλγορίθμους ανάλυσης δεδομένων και μοντέλα προβλέψεων. Υποστηρίζει λειτουργίες εξόρυξης δεδομένων όπως preprocessing, clustering, classification, regression, visualization και feature selection. Δέχεται δεδομένα σε μορφή αρχείων τύπου .arff στα οποία κάθε σημείο εκφράζεται ως ένα πλήθος χαρακτηριστικών (feature vector) ή απευθείας από κάποια βάση δεδομένων. Η χρησιμότητα του Weka στην παρούσα διπλωματική περιορίζεται στο classification. Χρησιμοποιήθηκε αρχικά για την ανάλυση ενός συνόλου δεδομένων εκπαίδευσης από το επιταχυνσιόμετρο του Pebble όπως προέκυψε από την εφαρμογή συλλογής δεδομένων ώστε να δημιουργηθεί ένα μοντέλο το οποίο θα μπορεί να ενσωματωθεί σε μια άλλη εφαρμογή ώστε να παράγει προβλέψεις σχετικά με την ένταση της δραστηριότητας του χρήστη με επαρκή ακρίβεια.

Αυτό το σύνολο δεδομένων μετατράπηκε σε feature vectors που αποτελείται από τα χαρακτηριστικά που αναλύθηκαν στην ενότητα 3.3. Συνολικά περιέχει 100 στιγμιότυπα, 25 για την κατηγορία sleep, 25 για την κατηγορία mild, 25 για την moderate και 25 για την intense. Οι τιμές όλων των χαρακτηριστικών συγκεντρώθηκαν σε ένα αρχείο τύπου .arff και δόθηκαν ως input στο Weka. Το συγκεκριμένο σύνολο δεδομένων χρησιμοποιήθηκε για την εκμάθηση. Επιλέχθηκε 10-Fold cross validation και δοκιμάστηκαν όλοι οι classification algorithms που έχει διαθέσιμο το Weka. Αυτός που ξεχώρισε ήταν ο J48 που αναλύθηκε στο προηγούμενο κεφάλαιο, ο οποίος έδωσε ποσοστό 97% όπως προέκυψε πάνω στο σύνολο εκμάθησης με cross validation. Η δενδρική δομή που παρήγαγε είναι αυτή που φαίνεται στο σχήμα 11 που ακολουθεί. Στη συνέχεια συγκροτήθηκε με παρόμοιο τρόπο ένα διαφορετικό σύνολο δεδομένων που χρησιμοποιήθηκε ως test set δηλαδή

ως εξ ολοκλήρου άγνωστο στον αλγόριθμο σύνολο δεδομένων για να γίνει επαλήθευση των αποτελεσμάτων. Το σύνολο αυτό αποτελείται από συνολικά από 40 στιγμιότυπα, 10 από κάθε κατηγορία και ο classifier μάντεψε σωστά στο 97.5% των περιπτώσεων. Ενδεικτικά το αρχείο .arff του test set φαίνεται πιο κάτω.

Όσον αφορά τις πτώσεις, συγκεντρώθηκαν 20 στιγμιότυπα από τις προσομοιώσεις τα οποία είναι διάρκειας 4 δευτερολέπτων και είναι κεντραρισμένα γύρο από την υψηλότερη τιμή του μέτρου του διανύσματος της επιτάχυνσης και 20 στιγμιότυπα ίσης διάρκειας από τυχαίες έντονες κινήσεις που προέκυψαν κατά την εκτέλεση δραστηριοτήτων και αυτά κεντραρισμένα με παρόμοιο τρόπο. Οι συνθήκες κατά τις οποίες προσομοιώθηκαν οι πτώσεις ήταν σε μαλακό στρώμα από όρθια και καθιστή θέση. Σε καμία περίπτωση δεν πρέπει να θεωρηθεί ότι καλύπτονται όλα τα είδη πτώσεων που μπορούν να συμβούν σε κάποιο ηλικιωμένο άτομο, όμως έγινε ό,τι ήταν εφικτό για τα πλαίσια της διπλωματικής εργασίας και την γενική κατανόηση της συμπεριφοράς κατά την πτώση, με δεδομένη την απουσία κατάλληλου εξοπλισμού και χώρου.



Σχήμα 11: Classifier που προέκυψε για την αναγνώριση δραστηριοτήτων από τον J48.

Έγινε αντίστοιχα μετατροπή των επιταχύνσεων του παραπάνω συνόλου δεδομένων στα χαρακτηριστικά της ενότητας 3.3 και εφαρμόστηκαν οι διάφοροι classification algorithms του Weka. Ο J48 έδωσε classifier με 72.5% επιτυχία ενώ η καλύτερη απόδοση ήρθε από τον Naive Bayes με 82.5%. Και στις 2 περιπτώσεις το ποσοστό θεωρήθηκε χαμηλό για τη σημασία του εντοπισμού των πτώσεων. Τα αποτελέσματα λεπτομερώς αναλύονται στο κεφάλαιο 5.

“accelTestSet.arff”

```
@RELATION myAccelDataSet


@ATTRIBUTE maxSum REAL
@ATTRIBUTE minSum REAL
@ATTRIBUTE variance REAL
@ATTRIBUTE stDevSum REAL
@ATTRIBUTE skewness REAL
@ATTRIBUTE kurtosis REAL
@ATTRIBUTE meanCos REAL
@ATTRIBUTE meanDist REAL
@ATTRIBUTE meanTch REAL
```

```
@ATTRIBUTE meanVch REAL
@ATTRIBUTE class {intense,mild,sleep,moderate}

@DATA
6225.81, 209.38, 2319587.62, 1523.01, 0.7269, 3398.69, 0.7073, 1745.97, 8203161.6, 0.3588, intense
6211.40, 185.73, 1555426.48, 1247.16, 0.9496, 3548.40, 0.7133, 1463.95, 4512632.18, 0.1267, intense
6159.31, 164.92, 1883510.33, 1372.41, 0.7351, 3205.59, 0.7133, 1599.66, 5073457.19, 0.2942, intense
6168.51, 165.90, 1892876.17, 1375.81, 0.7773, 3338.07, 0.7294, 1516.01, 5137811.78, 0.3499, intense
6182.75, 163.92, 1641244.33, 1281.11, 1.1754, 4118.30, 0.7681, 1251.81, 4017877.32, 0.1206, intense
6662.44, 181.72, 1884390.97, 1372.73, 1.0146, 3870.48, 0.7294, 1466.43, 4896575.02, 0.22331, intense
6642.24, 192.72, 1433654.84, 1197.35, 1.6751, 5603.06, 0.8255, 966.61, 3372775.00, 0.1327, intense
6040.25, 208.76, 1872823.62, 1368.51, 0.9090, 3313.25, 0.7258, 1526.85, 4272035.55, 0.9793, intense
5221.90, 247.09, 988230.22, 994.09, 0.8077, 2541.43, 0.8728, 820.83, 4241197.29, 1.9790, intense
5388.49, 131.20, 717024.74, 846.77, 1.5284, 4797.51, 0.9287, 569.20, 2447105.22, 0.2947, intense
1529.21, 590.32, 1484.25, 38.52, 3.7621, 3547.26, 0.9988, 33.05, 50592.59, 0.7834, mild
1942.84, 565.91, 2639.12, 51.37, 3.6235, 3954.99, 0.9978, 37.73, 39553.73, 0.3608, mild
1447.49, 531.20, 1672.48, 40.89, 0.7783, 1895.76, 0.9980, 41.97, 104512.58, 0.3481, mild
1692.33, 220.25, 5379.09, 73.34, -0.4971, 2222.15, 0.9970, 57.18, 127218.35, 0.3580, mild
1389.76, 728.52, 2126.13, 46.10, 0.7815, 658.69, 0.9984, 56.95, 100781.33, 0.3690, mild
1266.57, 850.18, 1199.16, 34.62, 0.3230, 318.08, 0.9989, 49.77, 66410.60, 0.7570, mild
2943.40, 210.29, 4671.31, 68.34, 6.0527, 12420.29, 0.9960, 65.96, 125779.19, 0.1465, mild
2447.92, 147.29, 4052.57, 63.65, 1.6937, 3921.59, 0.9965, 55.58, 70422.39, 0.0316, mild
1817.00, 318.09, 4037.17, 63.53, -0.2077, 2615.66, 0.9969, 45.73, 79662.75, 0.3120, mild
1729.51, 384.74, 2995.24, 54.72, 0.6486, 1595.29, 0.9976, 55.89, 106709.96, 0.1436, mild
1460.95, 992.80, 295.75, 17.19, 10.4386, 4241.31, 0.9998, 18.87, 38595.58, 0.7373, sleep
1128.25, 1000.31, 185.74, 13.62, 0.2790, 86.73, 0.9999, 18.19, 34406.4, 0.7735, sleep
1076.67, 968.52, 77.56, 8.80, 0.0777, 48.45, 0.9998, 17.75, 24780.80, 0.1380, sleep
1160.22, 960.06, 174.20, 13.19, 0.0553, 124.58, 0.9999, 17.11, 32486.29, 0.1452, sleep
1103.01, 997.94, 99.83, 9.99, 0.3652, 58.07, 0.9998, 17.51, 3737.49, 0.3677, sleep
1128.14, 968.52, 191.73, 13.84, -0.4154, 117.31, 0.9999, 18.16, 33999.57, 0.7788, sleep
1120.74, 960.96, 193.26, 13.90, -0.2939, 102.62, 0.9999, 18.27, 34033.87, 0.7913, sleep
1059.56, 964.95, 57.99, 7.61, -0.5113, 56.08, 0.9998, 16.45, 36295.67, 0.3588, sleep
1140.69, 981.42, 115.03, 10.72, 0.3954, 122.13, 0.9999, 14.69, 32336.48, 0.7707, sleep
1124.47, 972.45, 135.81, 11.65, 0.1803, 94.63, 0.9999, 16.19, 41254.74, 0.3669, sleep
2278.23, 361.15, 66536.76, 257.94, 0.6930, 1477.58, 0.9717, 246.24, 506384.94, 0.8571, moderate
2982.98, 73.32, 65673.01, 256.26, 1.2143, 2294.61, 0.9679, 241.29, 514385.72, 0.6545, moderate
4521.16, 73.31, 72798.89, 269.81, 2.1549, 5153.70, 0.9657, 248.27, 515103.64, 0.4011, moderate
```

```
2707.93, 144.88, 69888.26, 264.36, 0.5984, 1571.04, 0.9707, 243.78, 454834.44, 0.7572, moderate
2486.98, 107.92, 62261.48, 249.52, 0.9945, 1567.17, 0.9714, 227.01, 395796.02, 0.7378, moderate
2303.52, 551.59, 42795.72, 206.87, 1.0731, 1367.66, 0.9873, 166.72, 348172.98, 0.7440, moderate
2448.32, 36.66, 35502.91, 188.42, 1.2194, 2372.81, 0.9867, 148.65, 192053.72, 0.4196, moderate
4611.12, 141.76, 397069.61, 630.13, 2.8035, 7571.80, 0.9140, 394.60, 646498.87, 0.1782, moderate
3358.01, 351.81, 36107.30, 190.01, 3.6506, 5350.79, 0.9797, 153.44, 421728.22, 0.4502, moderate
4717.45, 102.44, 98414.53, 313.71, 2.7154, 6216.22, 0.9317, 311.50, 749591.89, 0.1048, moderate
```

4.4 Αναγνώριση δραστηριοτήτων και εντοπισμός πτώσεων

Σε αυτή την ενότητα παρουσιάζονται τα σημαντικότερα σημεία του κώδικα και λεπτομέρειες υλοποίησης που παρουσιάζουν ενδιαφέρον.

4.4.1 Υλοποίηση στο Smartphone

Σε αυτή την υλοποίηση δεν θα σταθούμε ιδιαίτερα για τους λόγους που έχουν ήδη αναφερθεί στο προηγούμενο κεφάλαιο. Η υλοποίηση αυτή έχει παρόμοια δομή με την εφαρμογή συλλογής δεδομένων. Το Pebble δηλαδή συγκεντρώνει σε έναν buffer επιταχύνσεις και τις στέλνει περιοδικά στο smartphone. Στο smartphone αρχικά υπολογίζονται τα features που χρειαζόμαστε για να εφαρμόσουμε το δέντρο αποφάσεων που έδωσε ο classifier που προέκυψε από τον J48 και στη συνέχεια εφαρμόζεται αυτό το δέντρο αποφάσεων κατατάσοντας τα δεδομένα σε δραστηριότητες. Στη συνέχεια γίνεται αποστολή των δραστηριοτήτων στον server στη συλλογή `http://83.212.115.163:8080/tracker`.

```
	//...

	//Apply J48's Decision Tree
	if(meanDistance <= 85.91){
		if(meanCos <= 0.99963){
			activityType.push("Mild");
			typeStart.push(timearray[0]);
			typeEnd.push(timearray[timearray.length-1]);
		}
		else{
			activityType.push("Sleep");
			typeStart.push(timearray[0]);
			typeEnd.push(timearray[timearray.length-1]);
		}
	}
	else{
		if(variance <= 233345.72){
			activityType.push("Moderate");
			typeStart.push(timearray[0]);
			typeEnd.push(timearray[timearray.length-1]);
		}
		else{
```

```
                activityType.push("Intense");
                typeStart.push(timearray[0]);
                typeEnd.push(timearray[timearray.length-1]);
            }
        }

        //...
```

Ο αλγόριθμος εντοπισμού πτώσεων εφαρμόζεται κι αυτός πάνω στα πακέτα δεδομένων 1.5 λεπτού που λαμβάνει το smartphone και είναι ακριβώς όπως περιγράφεται στο κεφάλαιο 3.4. Μια λεπτομέρεια στην υλοποίηση που έχει ενδιαφέρον αφορά τους πίνακες επικάλυψης που χρησιμοποιούνται. Σε περίπτωση που εντοπιστεί impact στην αρχή του τρέχοντος δείγματος δεδομένων που εξετάζεται μπορεί να χρειαστεί να ελέγξουμε κάποια ή κάποιες από τις τελευταίες τιμές του προηγούμενου δείγματος για το βήμα 2. Αυτές οι τιμές βρίσκονται σε έναν πίνακα overlappingLeft. Σε περίπτωση που εντοπιστεί impact στο τέλος του τρέχοντος δείγματος δεδομένων, μπορεί αντίστοιχα να χρειαστεί να ελέγξουμε τις πρώτες τιμές του επόμενου δείγματος που θα έρθει το επόμενο λεπτό. Για αυτό χρησιμοποιούμε τον πίνακα overlappingRight που περιέχει τις τελευταίες τιμές του τρέχοντος δείγματος ώστε να εξεταστούν μαζί με τις πρώτες τιμές του επόμενου δείγματος όταν αυτό ληφθεί και να αποφασιστεί αν πληρούνται οι προϋποθέσεις της πτώσης.

Όσον αφορά τον κώδικα που εκτελείται στο ρολόι, είναι στο μεγαλύτερο μέρος του όμοιος με την εφαρμογή συλλογής δεδομένων αφού η λειτουργία του είναι η ουσιαστικά η ίδια καθώς περιορίζεται στη συλλογή και αποστολή δεδομένων στο κινητό περιοδικά.

4.4.2 Υλοποίηση στο Pebble

Σκοπός σε αυτή την υλοποίηση είναι τόσο η αναγνώριση δραστηριοτήτων όσο και ο εντοπισμός των πτώσεων να γίνεται απευθείας στο ρολόι σε πραγματικό χρόνο με όσο το δυνατόν πιο απλό τρόπο. Ο στόχος είναι να μην επιβαρύνεται το ρολόι και να διατηρείται η μεγάλη διάρκεια της μπαταρίας. Για το λόγο αυτό η αναγνώριση δραστηριοτήτων περιορίζεται σε δύο καταστάσεις, asleep και awake. Η ανάγκη για μεταφορά επιταχύνσεων στο κινητό πλέον δεν υφίσταται και η χρήση του PebbleKit JS περιορίζεται στην επικοινωνία με τον εξυπηρετητή σε περίπτωση που εντοπίστηκε κάποια πτώση ώστε να σταλεί ειδοποίηση μέσω email ή όταν αλλάζει η κατάσταση του χρήστη από asleep σε awake και αντίστροφα. Έτσι, ελαχιστοποιείται η χρήση του Bluetooth και η διάρκεια της μπαταρίας επηρεάζεται ελάχιστα.

Για την αναγνώριση δραστηριοτήτων και για τον εντοπισμό πτώσεων χρησιμοποιείται η ποσότητα simpleSum που αποτελεί απλά το άθροισμα των απόλυτων τιμών των επιταχύνσεων σε κάθε άξονα όπως έχει ήδη αναφερθεί και έχουν αναλυθεί οι λόγοι. Για την αναγνώριση δραστηριοτήτων υπολογίζουμε αρχικά σε ένα πίνακα 25 στοιχείων το simpleSum για κάθε μια από τις 25 μετρήσεις επιταχύνσεων που γίνονται διαθέσιμες από τη data_handler και ταυτόχρονα κρατάμε τη μέγιστη και την ελάχιστη τιμή. Η διαφορά της μέγιστης από την ελάχιστη τιμή του simpleSum ανά 25 μετρήσεις (maxMinDifference) αποτελεί μέτρο του κατά πόσο έντονη είναι η δραστηριότητα που συμβαίνει στο αντίστοιχο χρονικό διάστημα ενός δευτερολέπτου. Πολύ μικρό maxMinDifference σημαίνει ότι ο χρήστης είναι ακίνητος ενώ όσο πιο μεγάλη είναι αυτή η ποσότητα σημαίνει κινήσεις ανάλογης έντασης. Γίνεται χρήση στη συνέχεια ενός πίνακα ακεραίων state[2] ο οποίος συγκεντρώνει ψήφους οι οποίοι διακινούνται ανάμεσα στις δύο θέσεις του. Η θέση state[0] αντιπροσωπεύει την κατάσταση ύπνου και η θέση state[1] την κατάσταση εγρήγορσης. Όταν

ξεκινάει για πρώτη φορά να εκτελείται η εφαρμογή, η state[0] έχει προεπιλεγμένη τιμή pointThreshold 1200 και η state[1] τιμή 0. Ανά κλήση της data_handler, ανάλογα με την τιμή του maxMinDifference, γίνεται μεταφορά pointStep πόντων (pointStep = 20 από προεπιλογή) από το state[0] στο state[1] εφόσον κριθεί ότι ο χρήστης κινείται και μεταφορά 1 πόντου από το state[1] στο state[0] εφόσον κριθεί ότι ο χρήστης είναι ακίνητος. Σύγκριση των τιμών στις δύο θέσεις ανά λεπτό επιτρέπει την λήψη απόφασης για το αν ο χρήστης κοιμάται η είναι ξύπνιος. Η μεγαλύτερη βαρύτητα στη διακίνηση πόντων από το state[0] στο state[1] επιτρέπει στην εφαρμογή κατά την έναρξη αλλά και όταν ο χρήστης ξυπνάει να το αντιλαμβάνεται άμεσα, μέσα στο επόμενο λεπτό που θα κληθεί να αποφασίσει. Αντίστοιχα, η μικρότερη βαρύτητα στη διακίνηση πόντων από το state[1] στο state[0] έχει σαν αποτέλεσμα μικρές περίοδοι ακινησίας να μην καθορίζουν αμέσως την κατάσταση ως ύπνου. Αντιθέτως στην ακραία περίπτωση που έχουμε την μέγιστη τιμή 1200 στο state[1] που αντιπροσωπεύει την κατάσταση εγρήγορσης και ο χρήστης πέφτει για ύπνο, θα χρειαστεί να περάσουν 10 λεπτά μέχρι η εφαρμογή να αποφασίσει την αλλαγή της κατάστασης, κάτι το οποίο έρχεται και σε συμφωνία με το μέσο χρόνο που χρειάζεται ένας φυσιολογικός άνθρωπος για να κοιμηθεί από τη στιγμή που θα ξαπλώσει. Οι παράμετροι της εφαρμογής μπορούν να ρυθμιστούν ανά πάσα στιγμή από το configuration page της εφαρμογής.

```
static void data_handler(AccelData *data, uint32_t num_samples) {

    //...

    for(int i=0;i<NUM_SAMPLES;i++){

        simpleSum[i]=abs(data[i].x)+abs(data[i].y)+abs(data[i].z);
        meanSimpleSum += simpleSum[i];
        if (simpleSum[i]>=maxSimpleSum)
            maxSimpleSum = simpleSum[i];
        if (simpleSum[i]<minSimpleSum)
            minSimpleSum = simpleSum[i];
    }
    //meanSimpleSum = meanSimpleSum / NUM_SAMPLES;
    maxMinDifference = maxSimpleSum - minSimpleSum;

    //SIMPLE ACTIVITY TRACKING
    if(maxMinDifference<=328){
        //User Immobile
        if(  ((state[0]+1)<pointThreshold) && ((state[1]-1)>0)  ){
            state[0]++;
            state[1]--;
        }
    }
    else{
        //User mobile
        if (  ((state[1]+20)<pointThreshold) && ((state[0]-pointStep)>0)  )  {
            state[1]=state[1]+pointStep;
            state[0]=state[0]-pointStep;
        }
    }

    //...

    counter++;
```

```
            if(counter==60){
                counter=0;
                if(state[0]>state[1]){
                      if(previousState==1){
                            sendStateChange(timestampFirstHalf,timestampSecondHalf,0);
                      }
                      previousState=0;
                }
                else{
                      if(previousState==0){
                            sendStateChange(timestampFirstHalf,timestampSecondHalf,1);
                      }
                      previousState=1;
                }
            }
      }
```

Όσον αφορά τον εντοπισμό πτώσεων η διαδικασία ξεκινάει όταν διαπιστωθεί κάποια υψηλή τιμή στο maxMinDifference που ξεπερνάει κάποιο fall threshold η οποία ενδέχεται να προκλήθηκε από την πρόσκρουση στο έδαφος. Η μεταβλητή fall threshold τέθηκε αρχικά ίση με 5000 που είναι μικρότερη από αυτή που παρατηρήθηκε κατά τις προσομοιώσεις διότι έχουμε ανεκτικότητα στα false positives. Τα παραπάνω αποτελούν το πρώτο στάδιο. Στο δεύτερο στάδιο, δηλαδή στην επόμενη κλήση της data_handler και εφόσον έχει ήδη διαπιστωθεί στο προηγούμενο στάδιο η ηψηλή τιμή, ελέγχουμε αν η μέγιστη τιμή του τρέχοντος δείγματος δεδομένων είναι μικρότερη από το fall threshold. Αυτό το στάδιο έχει σκοπό να απορροφήσει τις διακυμάνσεις στο simpleSum που προκύπτουν αφού το άτομο βρεθεί στο έδαφος. Στο τρίτο και τέταρτο στάδιο ελέγχουμε πάλι το maxMinDifference και αν βρεθεί σχετικά χαμηλό αυτό σημαίνει ότι το άτομο παρουσιάζει ιδιαίτερα χαμηλή κινητικότητα. Σε αυτή την περίπτωση ενεργοποιείται ένας timer που εμφανίζει μήνυμα στην οθόνη του ρολογιού και ρωτάει τον χρήστη αν είναι καλά. Σε περίπτωση που δεν απαντήσει μέσα σε 30 δευτερόλεπτα ενεργοποιείται η αποστολή ειδοποίησης.

```
      static void data_handler(AccelData *data, uint32_t num_samples) {

            //...

            //SIMPLE FALL DETECTION
            //stage 4
            if(highlyPossibleFall && (maxMinDifference<=maxMinThreshold)){
                 APP_LOG(APP_LOG_LEVEL_DEBUG, "FALL DETECTED ");
                 vibes_short_pulse();
                 text_layer_set_text(fall_layer, "Are you alright?");
                 appTimer = app_timer_register(30000,appTimerCallback,NULL);
            }
            highlyPossibleFall = false;

            //stage 3
            if(possibleFall && (maxMinDifference<=maxMinThreshold)){
                 highlyPossibleFall = true;
            }
            possibleFall = false;

            //stage 2
            if(highPeak && (maxMinDifference<fallThreshold)){
```

```
                possibleFall = true;
        }
        highPeak = false;

        //stage 1
        if(maxMinDifference>=fallThreshold){
                APP_LOG(APP_LOG_LEVEL_DEBUG, "High peak noticed %d",maxSimpleSum);
                highPeak = true;
        }

        //...
}
```

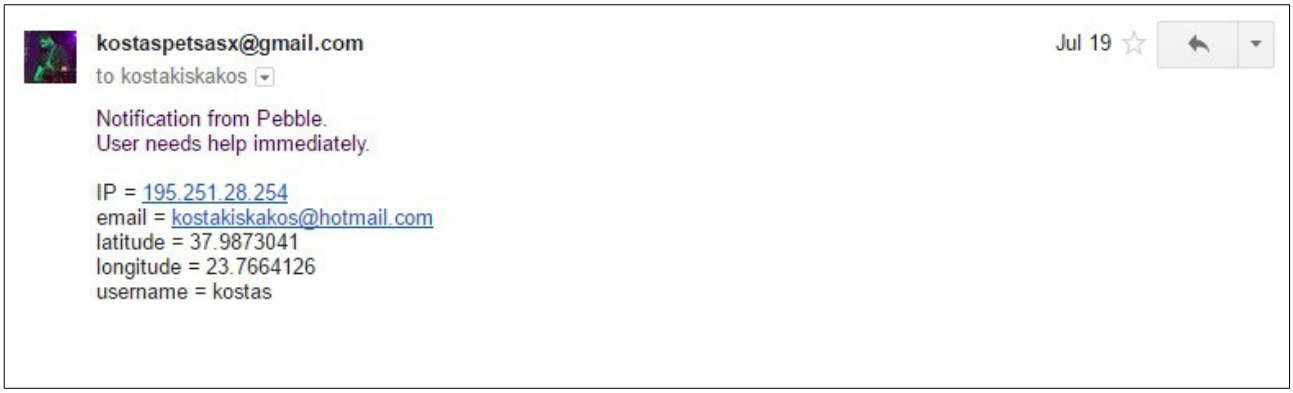
Τέλος, υπάρχει η δυνατότητα ο χρήστης να στείλει κατευθείαν ειδοποίηση ότι χρειάζεται βοήθεια πατώντας το κουμπί down του Pebble.

4.5 Sendmail server – αποστολή ειδοποίησης

Για την αποστολή της ειδοποίησης με email χρησιμοποιήθηκε το πακέτο sendmail και ένας λογαριασμός gmail ως διαπιστευτήριο για την πρόσβαση στον smtp server. Στη συνέχεια δημιουργήθηκε ένα script σε γλώσσα php το οποίο εκτελείται όταν το smartphone στείλει στον server κατάλληλη ειδοποίηση με τις συντεταγμένες του χρήστη και το email προς ειδοποίηση και αναλαμβάνει την αποστολή μηνύματος ηλεκτρονικού ταχυδρομίου προς το λογαριασμό αυτό.

“EmailSender.php”

```
<?php

        $SubjectOfEmail = "Emergency";
        $content = "Notification from Pebble. \nUser needs help immediately.\n\n";

        if( isset($_GET['IP']) ) {
                $_GET['IP '] = $_SERVER['REMOTE_ADDR'];
        }
        else {
                $_GET['IP'] = $_SERVER['REMOTE_ADDR'];
        }

        ksort($_GET);

        foreach( $_GET as $k => $v ) {
                $content .= "$k = $v\n";
                if($k==email)
                        $AddressToEmailTo = "$v";
        }

        mail($AddressToEmailTo,$SubjectOfEmail,$content,"From: $AddressToEmailTo");

?>
```



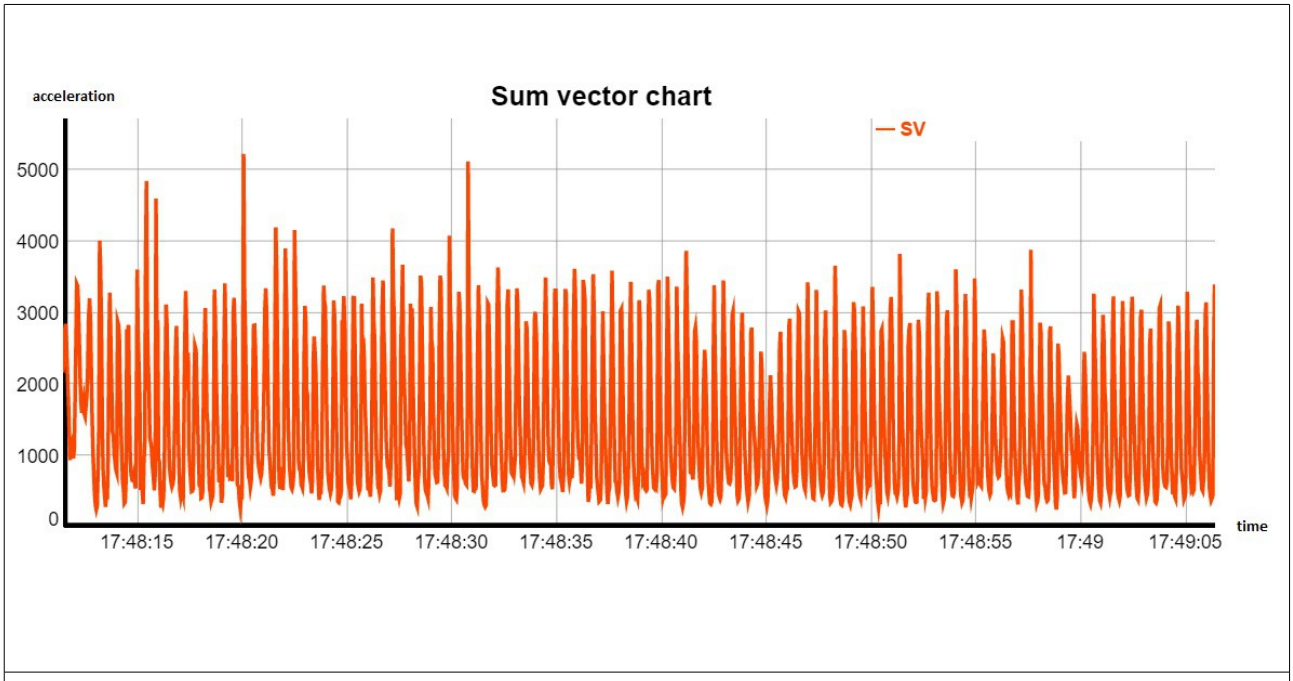
Σχήμα 12: Λήψη email ειδοποίησης με τις συντεταγμένες του χρήστη.

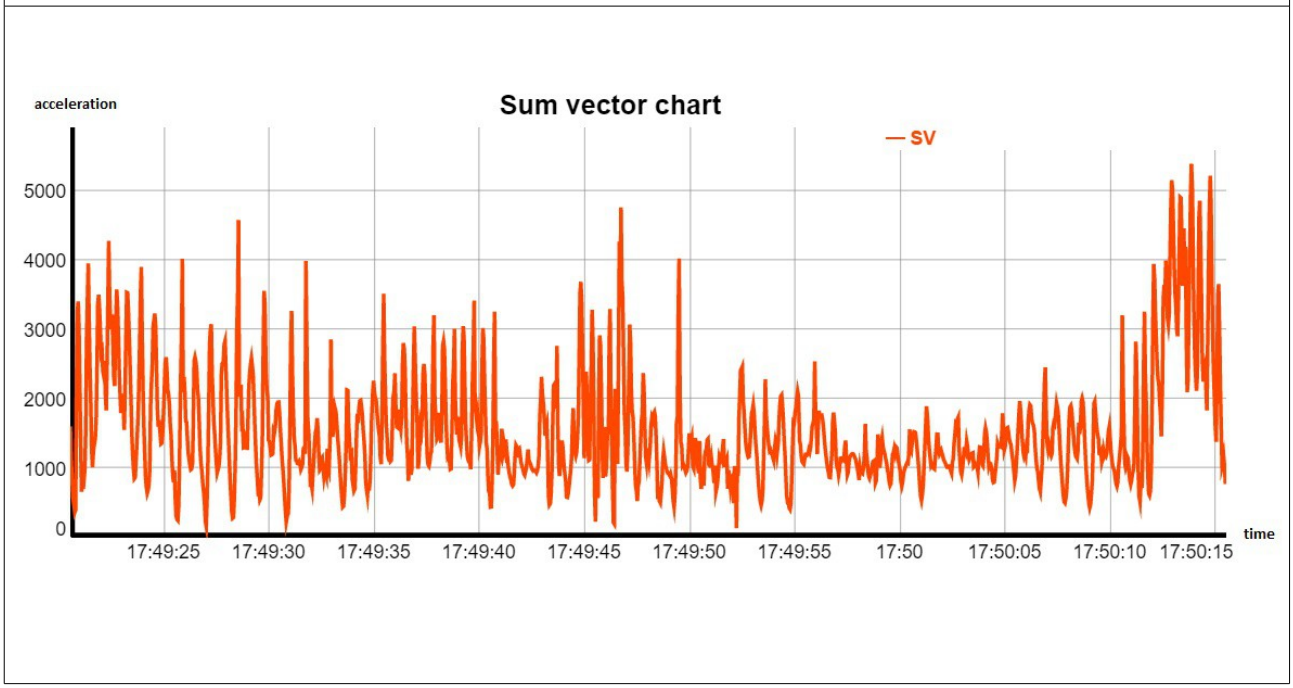
Κεφάλαιο 5. Αποτελέσματα

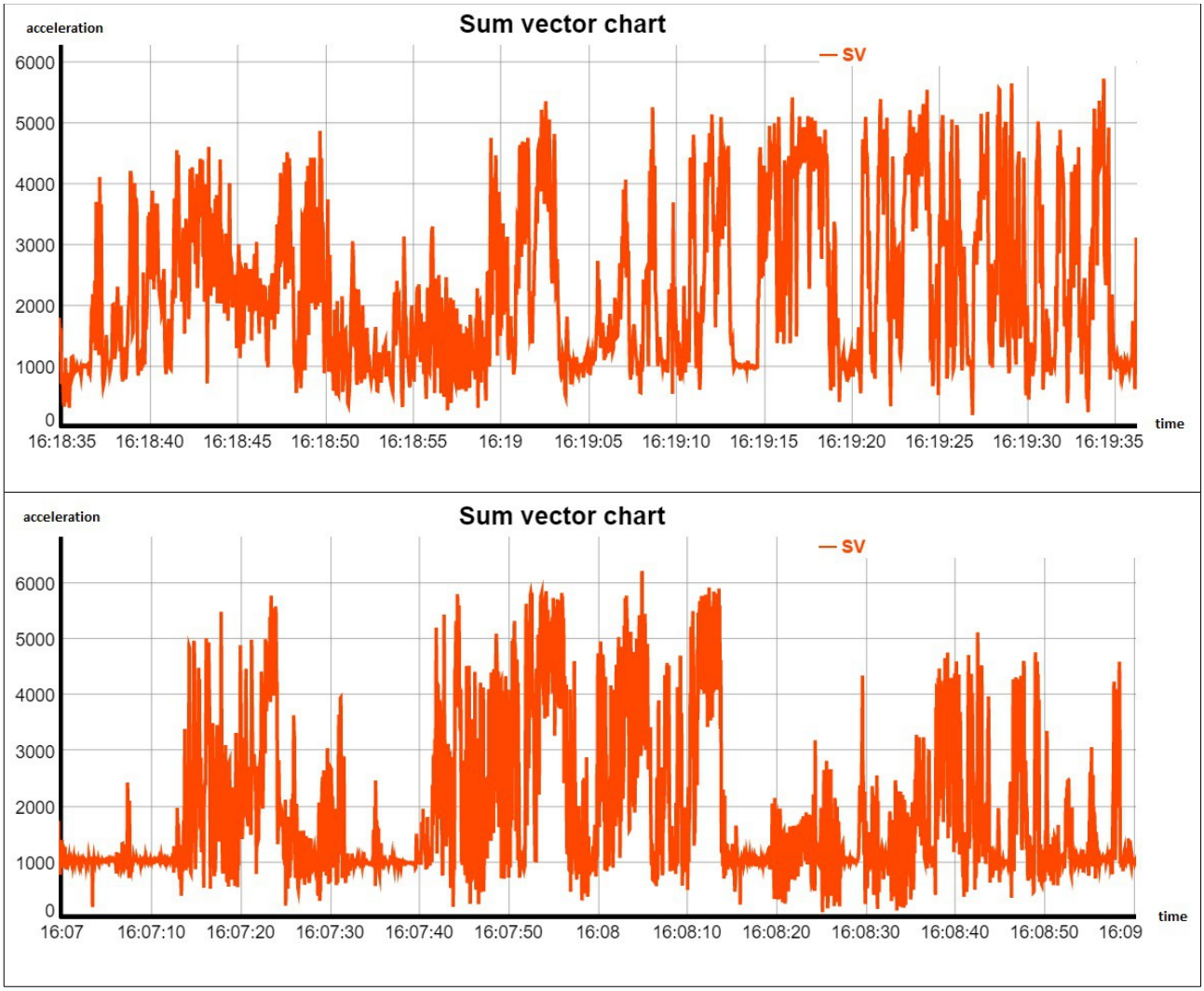
5.1 Γραφικές παραστάσεις δραστηριοτήτων (ADLs)

Ακολουθούν ενδεικτικές γραφικές παραστάσεις από τους τέσσερις τύπους στους οποίους χωρίσαμε τις δραστηριότητες ανάλογα με την ένταση που παρουσιάζουν.

Intense samples

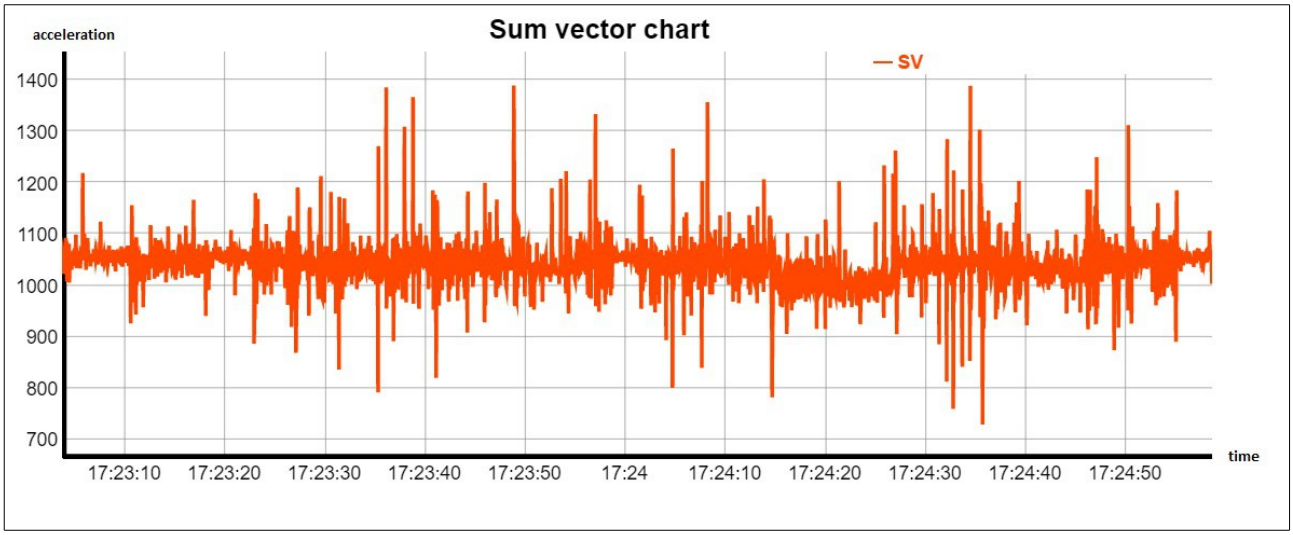


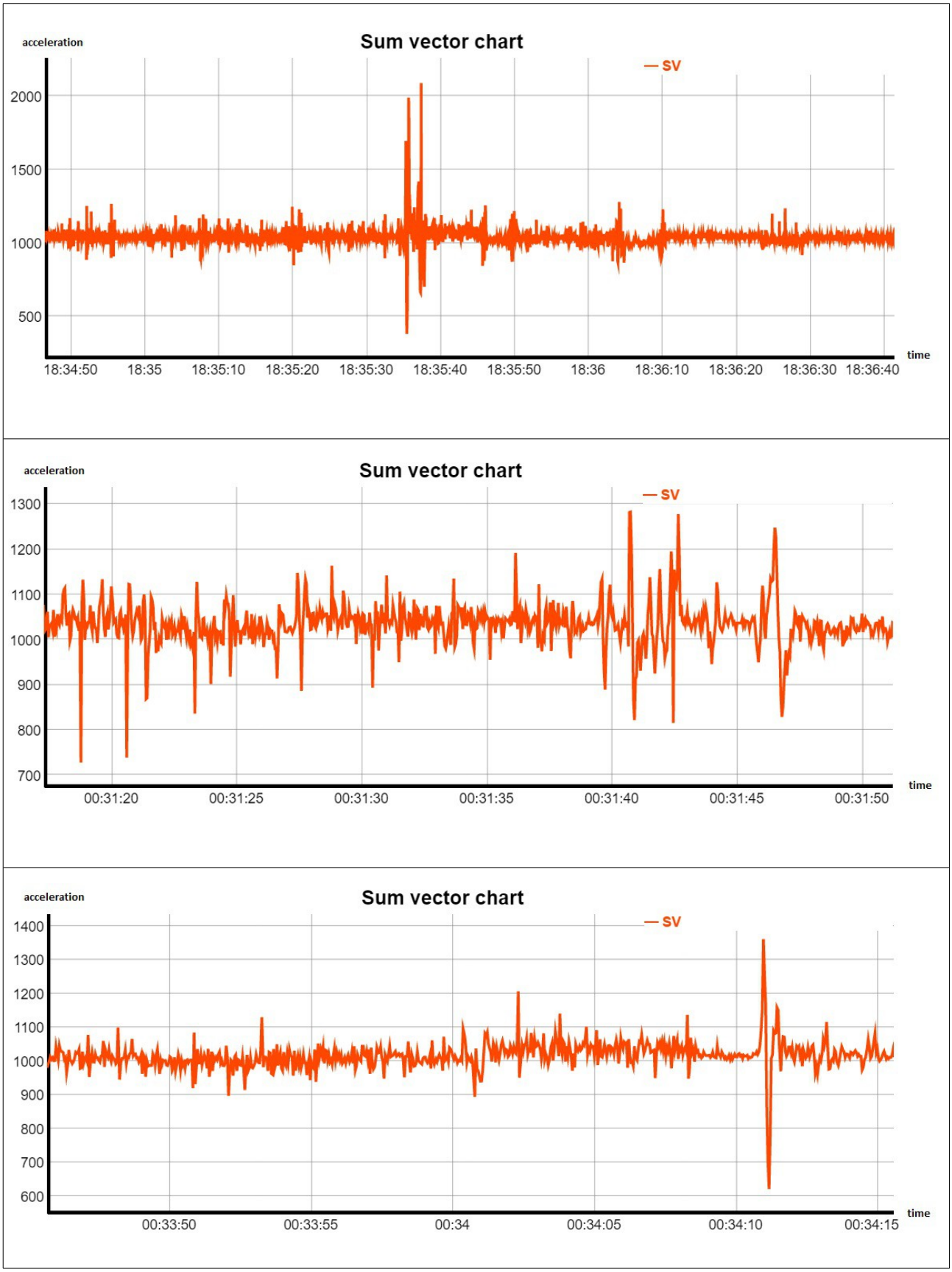




Σχήμα 13: Γραφικές παραστάσεις του μέτρου του διανύσματος των επιταχύνσεων από δείγματα δραστηριοτήτων έντονης ενεργητικότητας.

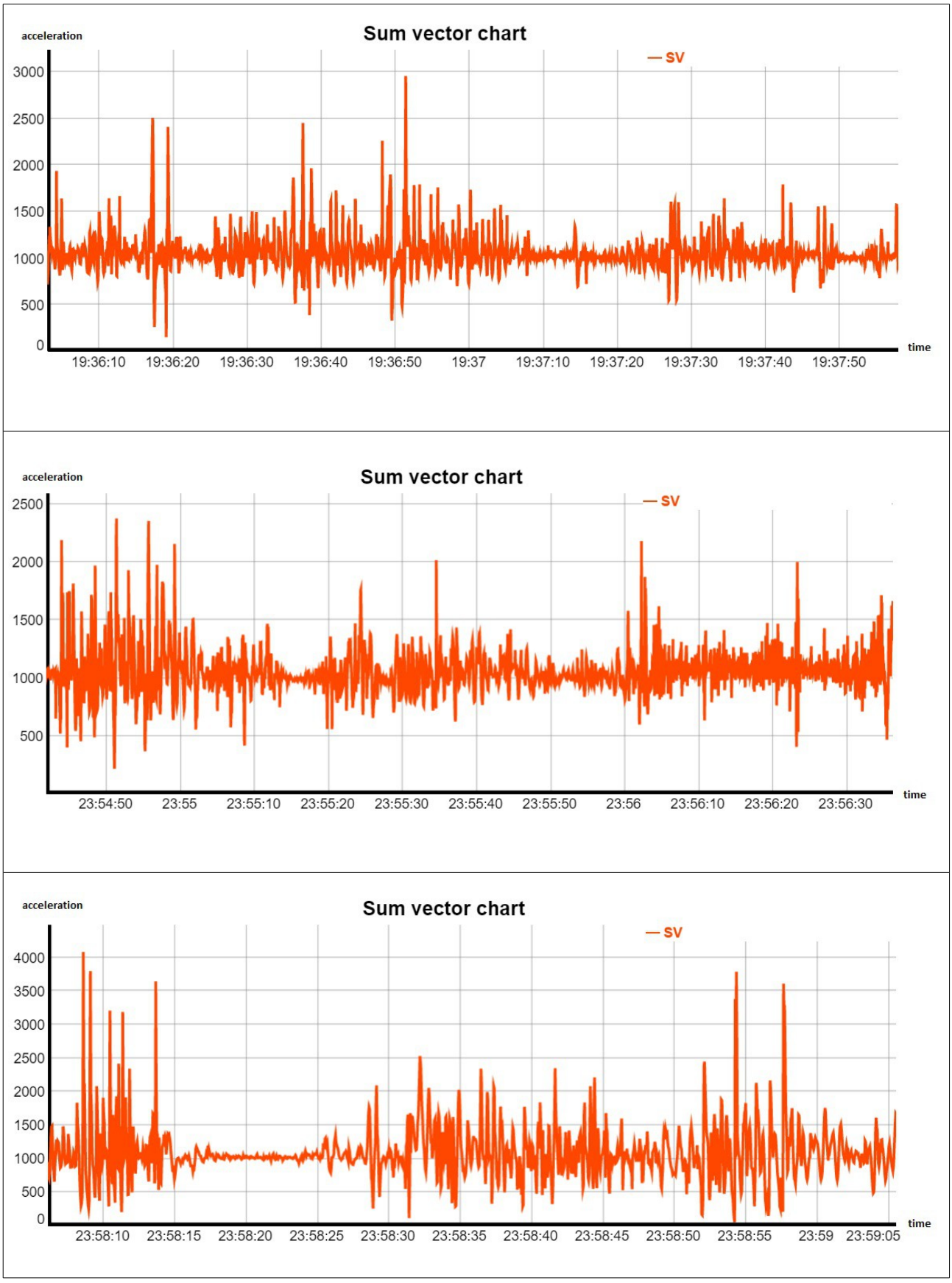
Mild samples

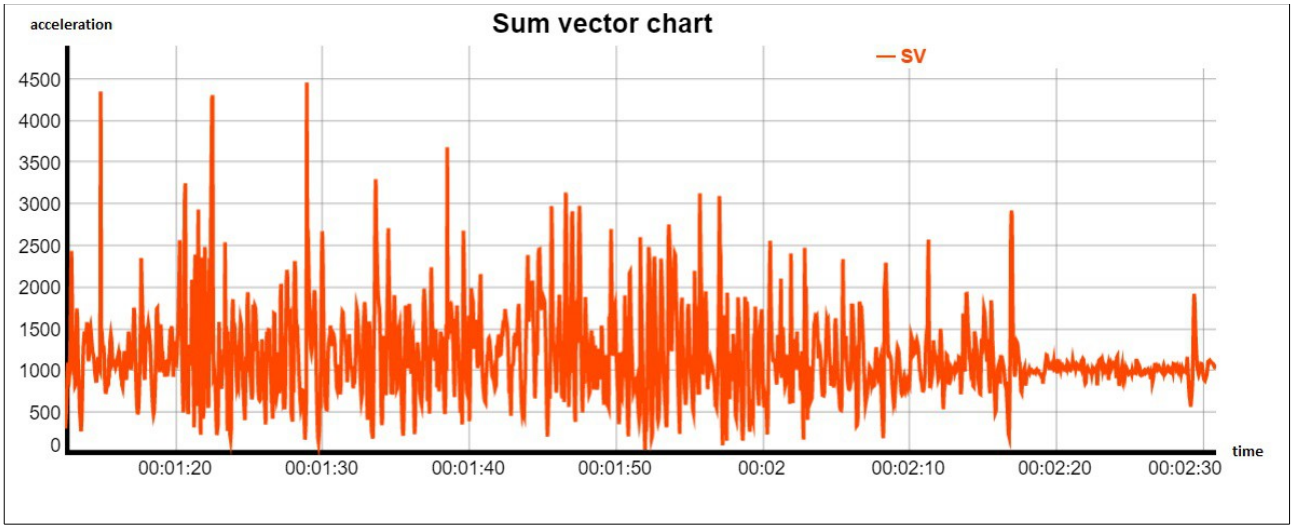




Σχήμα 14: Γραφικές παραστάσεις του μέτρου του διανύσματος των επιταχύνσεων από δείγματα δραστηριοτήτων χαμηλής ενεργητικότητας.

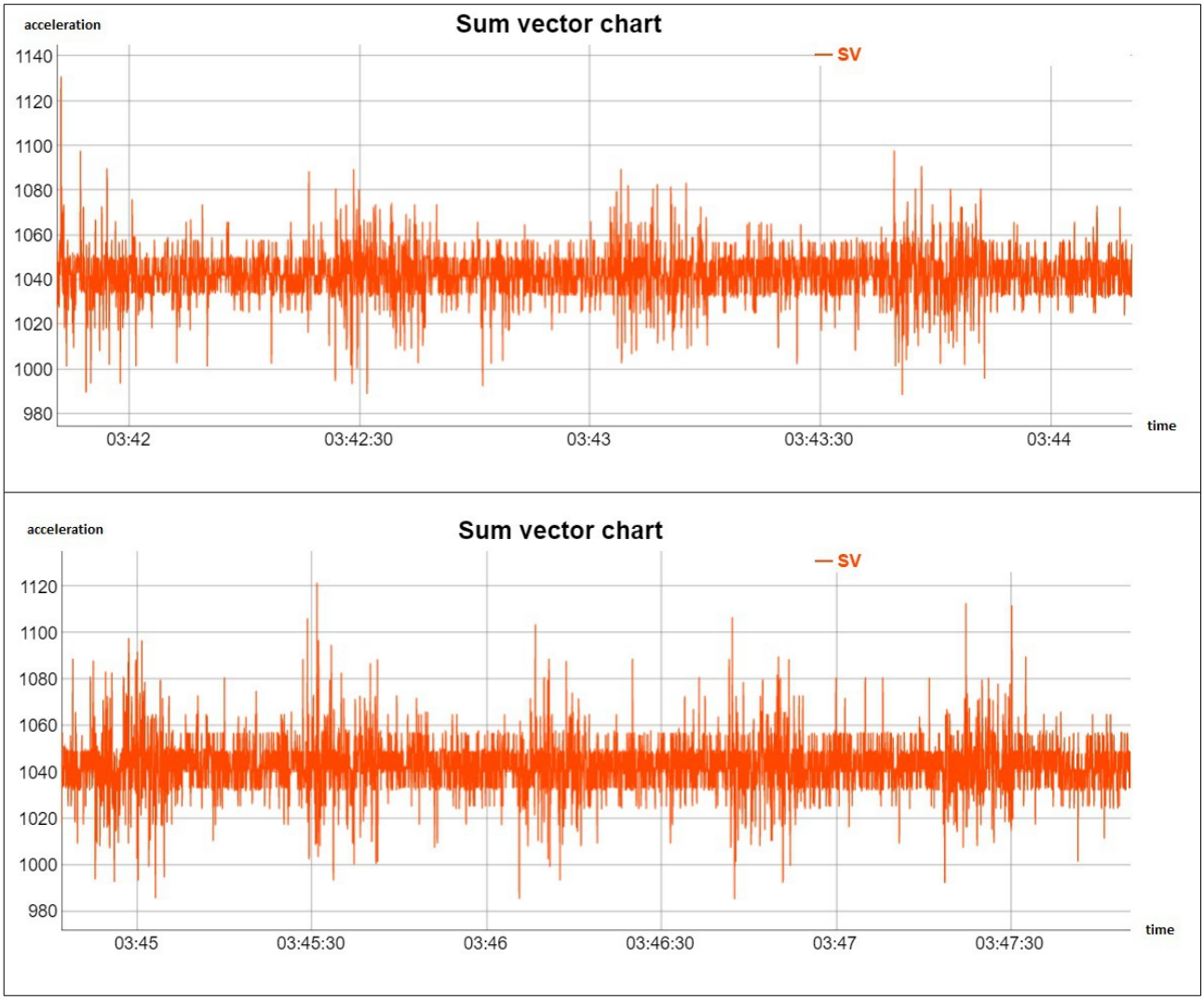
Moderate samples

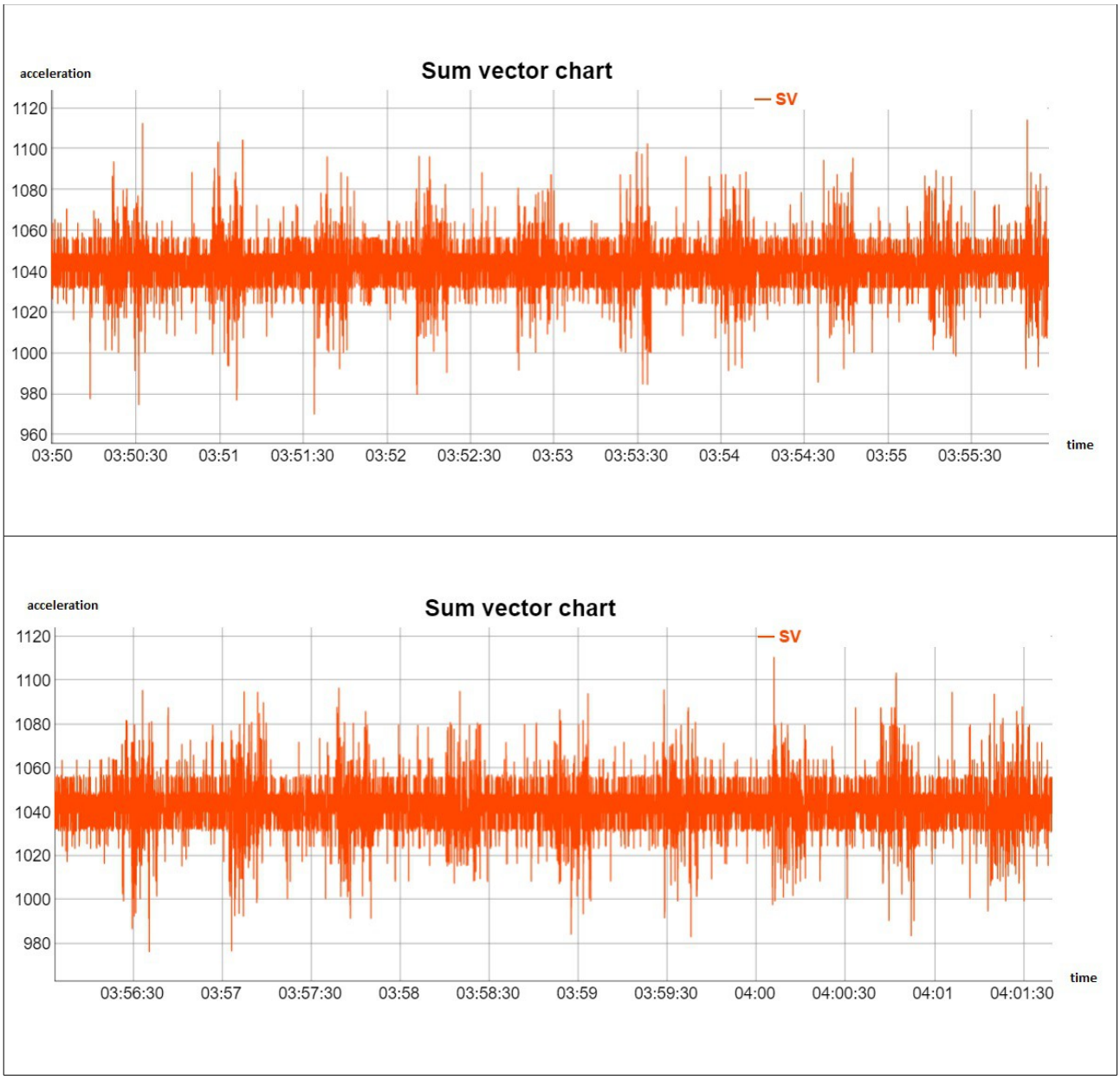




Σχήμα 15: Γραφικές παραστάσεις του μέτρου του διανύσματος των επιταχύνσεων από δείγματα δραστηριοτήτων μέτριας ενεργητικότητας.

Sleep samples





Σχήμα 16: Γραφικές παραστάσεις του μέτρου του διανύσματος των επιταχύνσεων από δείγματα ύπνου.

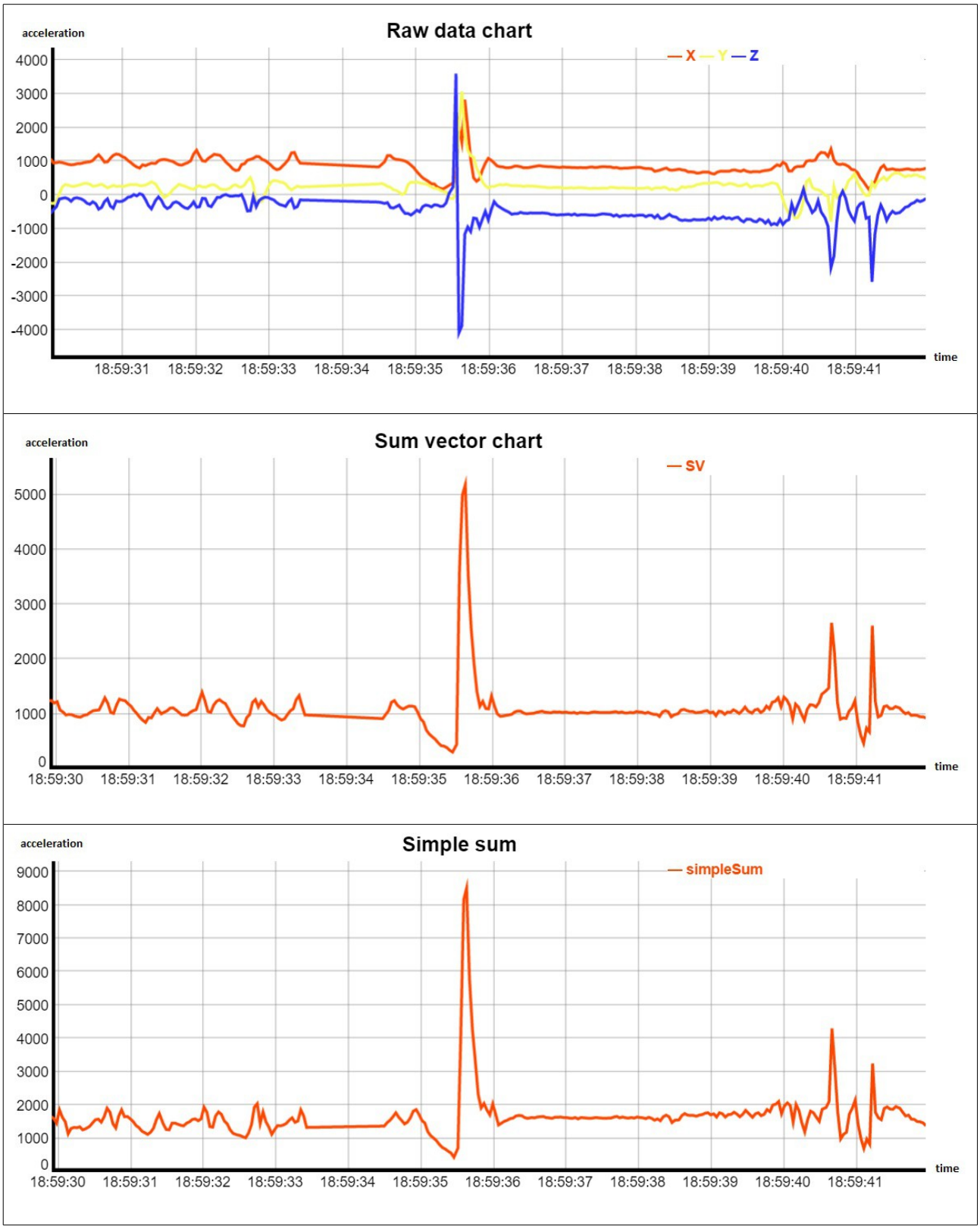
5.2 Γραφικές παραστάσεις πτώσεων

Ακολουθούν κάποιες ενδεικτικές γραφικές παραστάσεις προσομοιωμένων πτώσεων. Σε κάθε μια η πρώτη γραφική παράσταση αντιπροσωπεύει τις επιταχύνσεις όπως μετρήθηκαν από το Pebble, η δεύτερη το μέτρο του διανύσματος της επιτάχυνσης και η τρίτη το απλό άθροισμα των απόλυτων τιμών των επιταχύνσεων των τριών αξόνων.

Πτώση 1

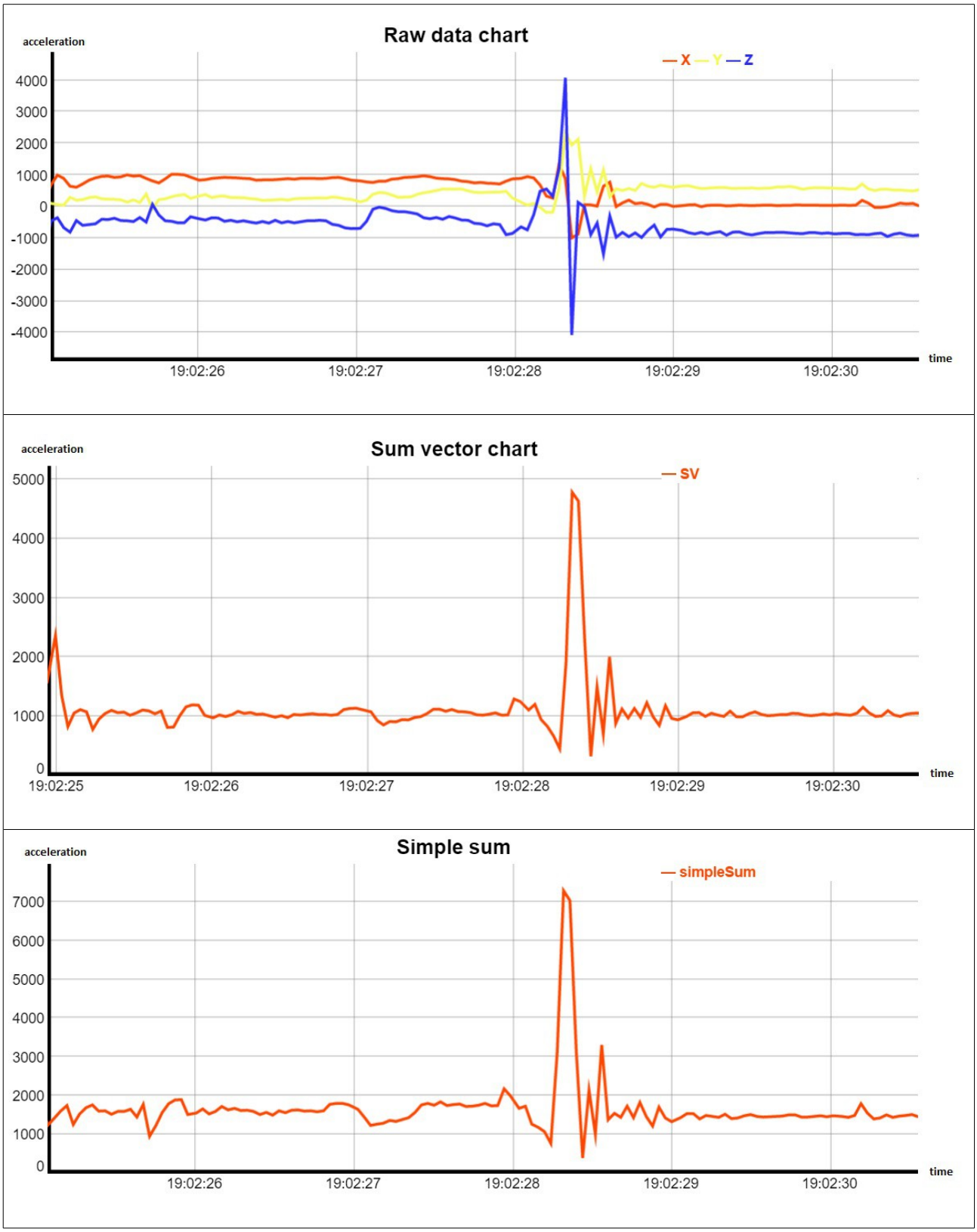
Σχήμα 17: Γραφικές παραστάσεις για την προσομοιωμένη πτώση 1. Raw data chart: Επιταχύνσεις στους τρεις άξονες. Sum vector chart: Μέτρο του διανύσματος των επιταχύνσεων. Simple sum: Άθροισμα των απόλυτων τιμών των επιταχύνσεων.

Πτώση 2



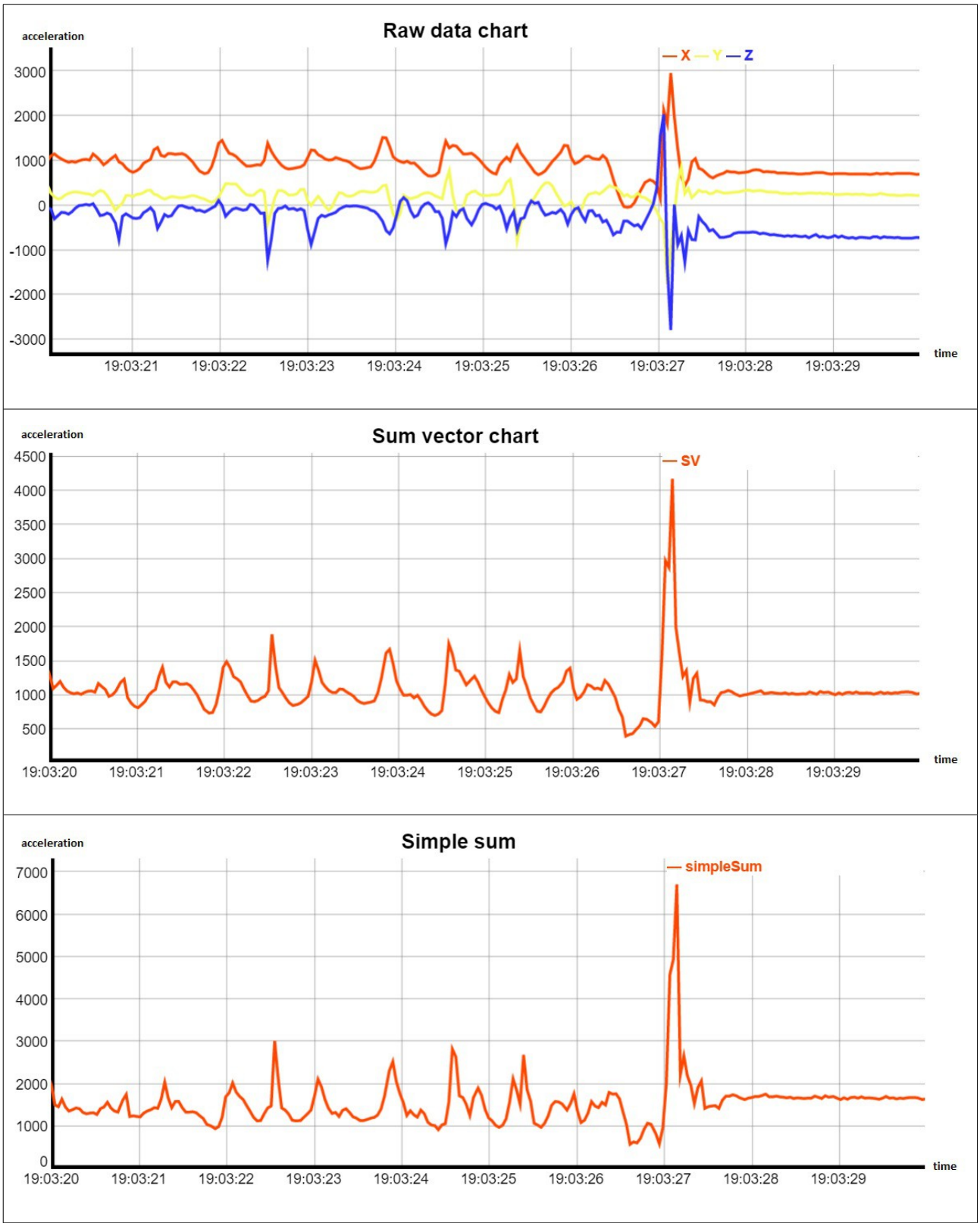
Σχήμα 18: Γραφικές παραστάσεις για την προσομοιωμένη πτώση 2. Raw data chart: Επιταχύνσεις στους τρεις άξονες. Sum vector chart: Μέτρο του διανύσματος των επιταχύνσεων. Simple sum: Άθροισμα των απόλυτων τιμών των επιταχύνσεων.

Πτώση 3



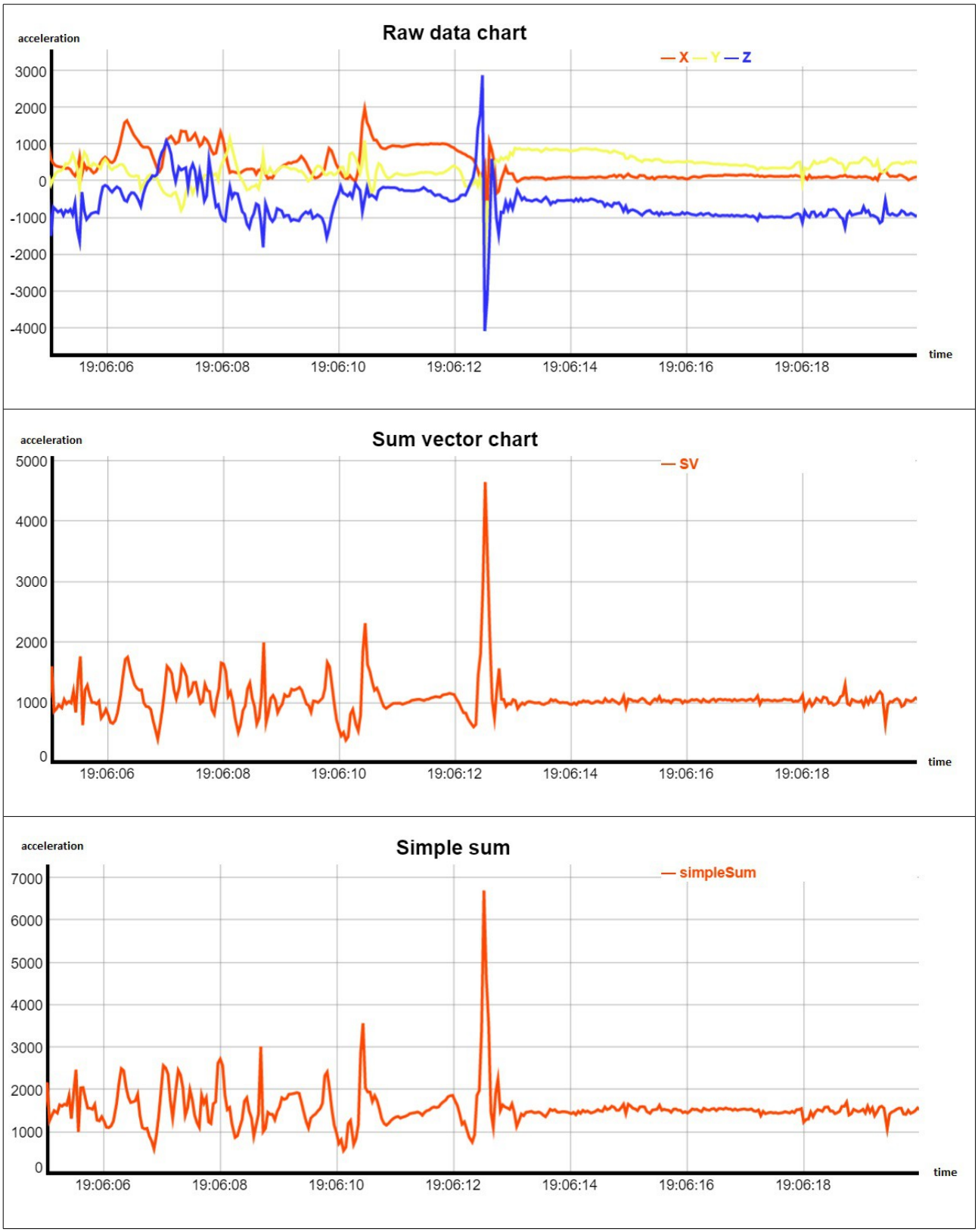
Σχήμα 19: Γραφικές παραστάσεις για την προσομοιωμένη πτώση 3. Raw data chart: Επιταχύνσεις στους τρεις άξονες. Sum vector chart: Μέτρο του διανύσματος των επιταχύνσεων. Simple sum: Άθροισμα των απόλυτων τιμών των επιταχύνσεων.

Πτώση 4



Σχήμα 20: Γραφικές παραστάσεις για την προσομοιωμένη πτώση 4. Raw data chart: Επιταχύνσεις στους τρεις άξονες. Sum vector chart: Μέτρο του διανύσματος των επιταχύνσεων. Simple sum: Άθροισμα των απόλυτων τιμών των επιταχύνσεων.

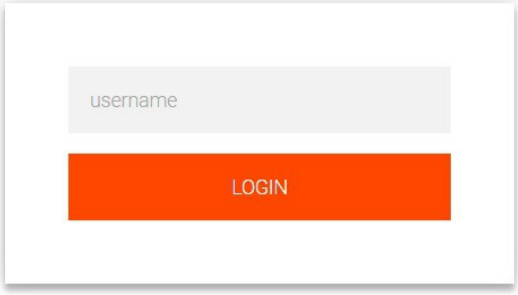
Πτώση 5



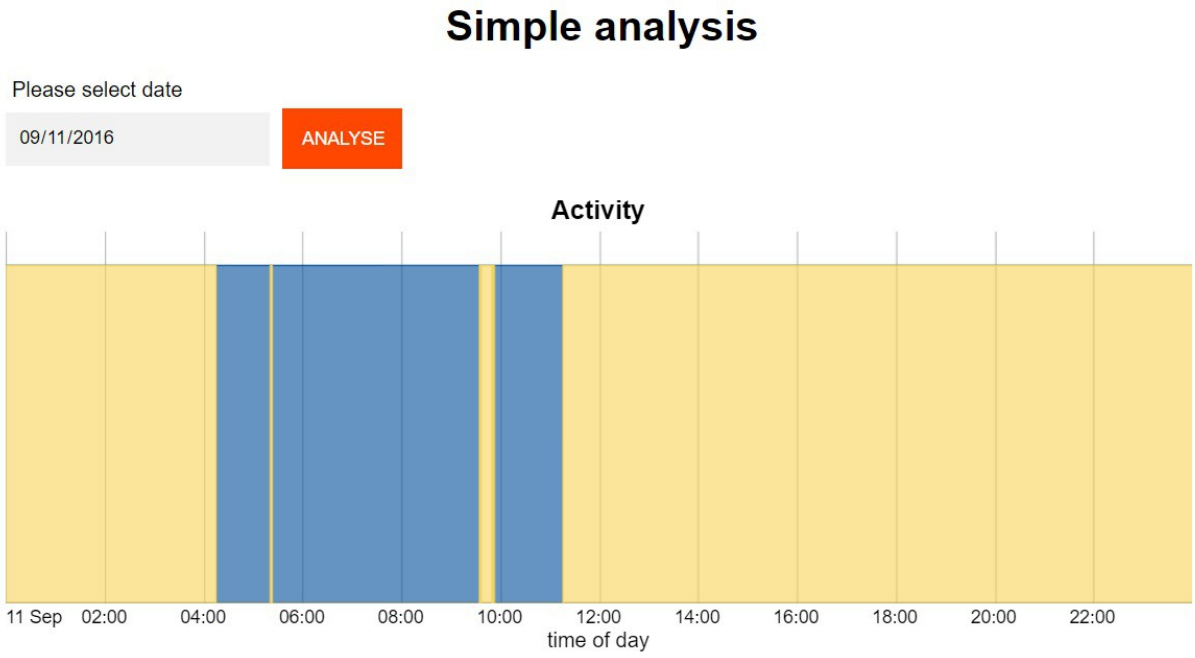
Σχήμα 21: Γραφικές παραστάσεις για την προσομοιωμένη πτώση 5. Raw data chart: Επιταχύνσεις στους τρεις άξονες. Sum vector chart: Μέτρο του διανύσματος των επιταχύνσεων. Simple sum: Άθροισμα των απόλυτων τιμών των επιταχύνσεων.

5.3 Γραφικές παραστάσεις διαδικτυακής εφαρμογής

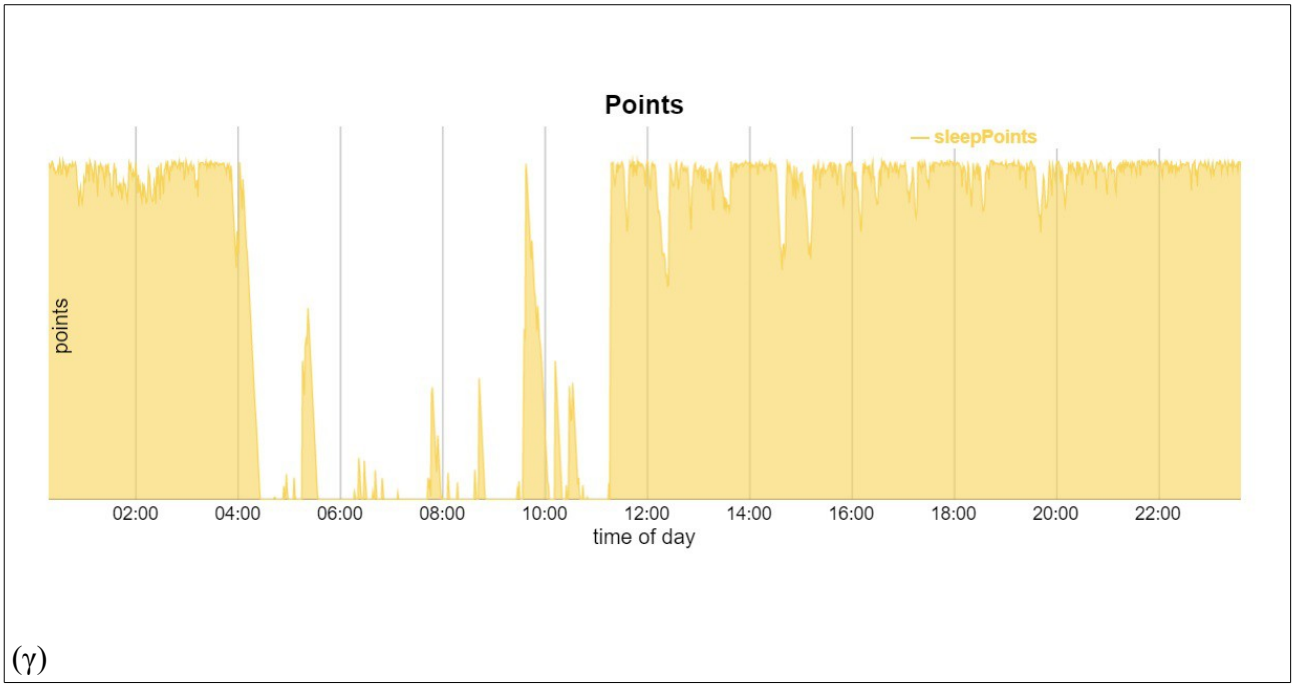
Ακολουθούν screenshots από τη διαδικτυακή εφαρμογή στην οποία εμφανίζεται η δραστηριότητα του χρήστη.



(α)



(β)



(γ)

Σχήμα 22: Φόρμα εισόδου (α). Σελίδα γραφικής παρουσίασης δραστηριότητας και στατιστικών στοιχείων (β). Διάγραμμα πόντων που καθορίζουν τον τρόπο μετάβασης από την κατάσταση asleep σε awake και αντίστροφα (γ).

Στο σχήμα 22 (β) φαίνονται οι πληροφορίες που μπορεί να δει ο χρήστης από στη web εφαρμογή κάνοντας login. Βλέπουμε ότι κατά τη διάρκεια της νύχτας ο χρήστης ξύπνησε δύο φορές το οποίο όντως συνέβη. Στο σχήμα 22 (γ) φαίνεται ο τρόπος με τον οποίο καθορίζει η εφαρμογή πότε ο χρήστης κοιμάται και πότε όχι. Κατά τη διάρκεια του ύπνου η τιμή των πόντων είναι χαμηλή ενώ κατά τη διάρκεια της ημέρας είναι ιδιαίτερα υψηλή. Παρ'όλα αυτά μικρές πτώσεις ή μικρές αυξήσεις αντίστοιχα δεν είναι ικανές να αλλάξουν την κατάσταση παρά μόνο όταν οι μεταβολές υπερβαίνουν κάποιο όριο το οποίο έχει τεθεί περίπου στα μισά στη συγκεκριμένη περίπτωση. Στο διάγραμμα βλέπουμε επίσης μικρές αυξήσεις κατά τη διάρκεια του ύπνου οι οποίες οφείλονται σε ακούσιες κινήσεις του σώματος του χρήστη ενώ κοιμάται.

Στο σχήμα 23 παρουσιάζεται η δραστηριότητα του χρήστη γραφικά από κάποια άλλη μέρα. Βλέπουμε στο σχήμα 23 (α) ότι ο χρήστης παρουσίασε καλύτερο ύπνο χωρίς να ξυπνήσει κατά τη διάρκεια της νύχτας αυτή τη φορά. Στο 23 (β) φαίνονται πάλι οι πόντοι που καθορίζουν τη μετάβαση από τη μια κατάσταση στην άλλη. Είναι επίσης ευδιάκριτη η αργή μετάβαση από την κατάσταση awake στην κατάσταση asleep και η γρήγορη μετάβαση από την κατάσταση asleep στην κατάσταση awake κάτι το οποίο οφείλεται στον τρόπο κατασκευής και έχει αναλυθεί νωρίτερα.

Σχήμα 23: Σελίδα γραφικής παρουσίασης δραστηριότητας και στατιστικών στοιχείων (α). Διάγραμμα πόντων που καθορίζουν τον τρόπο μετάβασης από την κατάσταση asleep σε awake και αντίστροφα (β).

5.4 Αποτελέσματα μηχανικής μάθησης σε δραστηριότητες

Ακολουθούν πίνακες με τα στατιστικά της μηχανικής μάθησης με χρήση των πιο αντιπροσωπευτικών classifiers. Αρχικά παρουσιάζονται τα αποτελέσματα πάνω στο σύνολο με το οποίο έγινε η εκμάθηση με cross-validation και στη συνέχεια τα αποτελέσματα πάνω στο άγνωστο σύνολο. Η ενότητα αυτή αφορά την αναγνώριση δραστηριοτήτων.

Naive Bayes Classifier στο σύνολο εκμάθησης.

Stratified cross-validation Summary	
Correctly Classified Instances	94,00%
Incorrectly Classified Instances	6,00%
Kappa statistic	0.92
Mean absolute error	0.0276
Root mean squared error	0.1545
Relative absolute error	7.3498 %
Root relative squared error	35.6273 %
Total Number of Instances	100

Detailed Accuracy By Class									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1,000	0,013	0,962	1,000	0,980	0,974	1,000	1,000	intense
	0,96	0,053	0,857	0,960	0,906	0,874	0,986	0,923	mild
	0,92	0,000	1,000	0,920	0,958	0,947	0,999	0,998	sleep
	0,88	0,013	0,957	0,880	0,917	0,892	0,993	0,983	moderate
Weight. Avg.	0,94	0,020	0,944	0,940	0,940	0,922	0,995	0,976	

Confusion Matrix				
Intense	**Mild**	**Sleep**	**Moderate**	**← Classified as**
25	0	0	0	**Intense**
0	24	0	1	**Mild**
0	2	23	0	**Sleep**
1	2	0	22	**Moderate**

Πίνακας 6: Αποτελέσματα του Naive Bayes Classifier πάνω στο σύνολο εκμάθησης όσον αφορά την αναγνώριση δραστηριοτήτων.

Naive Bayes Classifier στο άγνωστο σύνολο.

Stratified cross-validation Summary	
Correctly Classified Instances	85,00%
Incorrectly Classified Instances	15,00%
Kappa statistic	0.8
Mean absolute error	75
Root mean squared error	0.2739
Relative absolute error	20.0089 %
Root relative squared error	63.2454 %
Total Number of Instances	40

Detailed Accuracy By Class									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	intense
	1,000	0,200	0,625	1,000	0,769	0,707	0,870	0,605	mild
	0,4	0,000	1,000	0,400	0,571	0,577	0,993	0,983	sleep
	1	0,000	1,000	1,000	1,000	1,000	1,000	1,000	moderate
Weight. Avg.	0,85	0,050	0,906	0,850	0,835	0,821	0,966	0,897	

Confusion Matrix				
Intense	**Mild**	**Sleep**	**Moderate**	**← Classified as**
10	0	0	0	**Intense**
0	10	0	0	**Mild**
0	6	4	0	**Sleep**
0	0	0	10	**Moderate**

Πίνακας 7: Αποτελέσματα του Naive Bayes Classifier πάνω στο άγνωστο σύνολο όσον αφορά την αναγνώριση δραστηριοτήτων.

Simple Logistic Classifier στο σύνολο εκμάθησης.

Stratified cross-validation Summary	
Correctly Classified Instances	97,00%
Incorrectly Classified Instances	3,00%
Kappa statistic	0.96
Mean absolute error	0.0439
Root mean squared error	0.1209
Relative absolute error	11.6837 %
Root relative squared error	27.8697 %
Total Number of Instances	100

Detailed Accuracy By Class									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	intense
	0,92	0,013	0,958	0,920	0,939	0,919	0,996	0,989	mild
	0,96	0,013	0,960	0,960	0,960	0,947	0,999	0,997	sleep
	1	0,013	0,962	1,000	0,980	0,974	0,999	0,998	moderate
Weight. Avg.	0,97	0,010	0,970	0,970	0,970	0,960	0,999	0,996	

Confusion Matrix				
Intense	**Mild**	**Sleep**	**Moderate**	**← Classified as**
25	0	0	0	**Intense**
0	21	1	1	**Mild**
0	1	24	0	**Sleep**
0	0	0	25	**Moderate**

Πίνακας 8: Αποτελέσματα του Simple Logistic Classifier πάνω στο σύνολο εκμάθησης όσον αφορά την αναγνώριση δραστηριοτήτων.

Simple Logistic Classifier στο άγνωστο σύνολο.

Stratified cross-validation Summary	
Correctly Classified Instances	100,00%
Incorrectly Classified Instances	0,00%
Kappa statistic	1
Mean absolute error	0.0048
Root mean squared error	0.0387
Relative absolute error	1.2817 %
Root relative squared error	8.9438 %
Total Number of Instances	40

Detailed Accuracy By Class									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	intense
	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	mild
	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	sleep
	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	moderate
Weight. Avg.	1,000	1,000	1,000	1,000	1,000	1,000	1,000	1,000	

Confusion Matrix				
Intense	**Mild**	**Sleep**	**Moderate**	**← Classified as**
10	0	0	0	**Intense**
0	10	0	0	**Mild**
0	0	10	0	**Sleep**
0	0	0	10	**Moderate**

Πίνακας 9: Αποτελέσματα του Simple Logistic Classifier πάνω στο άγνωστο σύνολο όσον αφορά την αναγνώριση δραστηριοτήτων.

K-nearest neighbours Classifier στο σύνολο εκμάθησης.

Stratified cross-validation Summary	
Correctly Classified Instances	95,00%
Incorrectly Classified Instances	5,00%
Kappa statistic	0.9333
Mean absolute error	0.0399
Root mean squared error	0.1558
Relative absolute error	10.6232 %
Root relative squared error	35.9284 %
Total Number of Instances	100

Detailed Accuracy By Class

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	intense
	0,88	0,027	0,917	0,880	0,898	0,865	0,927	0,837	mild
	0,96	0,000	1,000	0,960	0,980	0,973	0,980	0,970	sleep
	0,96	0,040	0,889	0,960	0,923	0,897	0,960	0,863	moderate
Weight. Avg.	0,95	0,017	0,951	0,950	0,950	0,934	0,967	0,918	

Confusion Matrix

Intense	**Mild**	**Sleep**	**Moderate**	**← Classified as**
25	0	0	0	**Intense**
0	22	0	3	**Mild**
0	1	24	0	**Sleep**
0	1	0	24	**Moderate**

Πίνακας 10: Αποτελέσματα του K-nearest neighbours Classifier πάνω στο σύνολο εκμάθησης όσον αφορά την αναγνώριση δραστηριοτήτων.

K-nearest neighbours Classifier στο άγνωστο σύνολο.

Stratified cross-validation Summary	
Correctly Classified Instances	90,00%
Incorrectly Classified Instances	10,00%
Kappa statistic	0.8667
Mean absolute error	0.0625
Root mean squared error	0.2199
Relative absolute error	16.6667 %
Root relative squared error	50.7828 %
Total Number of Instances	40

Detailed Accuracy By Class

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	intense
	0,700	0,033	0,875	0,700	0,778	0,722	0,833	0,688	mild
	0,900	0,000	1,000	0,900	0,947	0,933	0,950	0,925	sleep
	1,000	0,100	0,769	1,000	0,870	0,832	0,950	0,769	moderate
Weight. Avg.	0,900	0,033	0,911	0,900	0,899	0,872	0,933	0,845	

Confusion Matrix

Intense	**Mild**	**Sleep**	**Moderate**	**← Classified as**
10	0	0	0	**Intense**
0	7	0	3	**Mild**
0	1	9	0	**Sleep**
0	0	0	10	**Moderate**

Πίνακας 11: Αποτελέσματα του K-nearest neighbours Classifier πάνω στο άγνωστο σύνολο όσον αφορά την αναγνώριση δραστηριοτήτων.

J48 στο σύνολο εκμάθησης.

Stratified cross-validation Summary	
Correctly Classified Instances	97,00%
Incorrectly Classified Instances	3,00%
Kappa statistic	0.96
Mean absolute error	15
Root mean squared error	0.1225
Relative absolute error	3.9943 %
Root relative squared error	28.2421 %
Total Number of Instances	100

Detailed Accuracy By Class									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1,000	0,013	0,962	1,000	0,980	0,974	0,993	0,962	intense
	0,92	0,000	1,000	0,920	0,958	0,947	0,960	0,940	mild
	1	0,013	0,962	1,000	0,980	0,974	0,993	0,962	sleep
	0,96	0,013	0,960	0,960	0,960	0,947	0,973	0,932	moderate
Weight. Avg.	0,97	0,010	0,971	0,970	0,970	0,960	0,980	0,949	

Confusion Matrix				
Intense	**Mild**	**Sleep**	**Moderate**	**← Classified as**
25	0	0	0	**Intense**
0	23	1	1	**Mild**
0	0	25	0	**Sleep**
1	0	0	24	**Moderate**

Πίνακας 12: Αποτελέσματα του J48 πάνω στο σύνολο εκμάθησης όσον αφορά την αναγνώριση δραστηριοτήτων.

J48 στο άγνωστο σύνολο.

Stratified cross-validation Summary	
Correctly Classified Instances	97,50%
Incorrectly Classified Instances	2,5%
Kappa statistic	0.9667
Mean absolute error	0.0125
Root mean squared error	0.1118
Relative absolute error	3.3333 %
Root relative squared error	25.8199 %
Total Number of Instances	40

Detailed Accuracy By Class									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1,000	0,033	0,909	1,000	0,952	0,937	0,983	0,909	intense
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	mild
	1,000	0,000	1,000	1,000	1,000	1,000	1,000	1,000	sleep
	0,900	0,000	1,000	0,900	0,947	0,933	0,950	0,925	moderate
Weight. Avg.	0,975	0,008	0,977	0,975	0,975	0,968	0,983	0,959	

Confusion Matrix				
Intense	**Mild**	**Sleep**	**Moderate**	**← Classified as**
10	0	0	0	**Intense**
0	10	0	0	**Mild**
0	0	10	0	**Sleep**
1	0	0	9	**Moderate**

Πίνακας 13: Αποτελέσματα του J48 πάνω στο άγνωστο σύνολο όσον αφορά την αναγνώριση δραστηριοτήτων.

5.5 Αποτελέσματα μηχανικής μάθησης σε πτώσεις.

Ακολουθούν τα αποτελέσματα της μηχανικής μάθησης στις πτώσεις με τους κυριότερους classifiers πάνω στο σύνολο εκμάθησης με cross-validation.

Naive Bayes στο σύνολο εκμάθησης.

Stratified cross-validation Summary	
Correctly Classified Instances	82,50%
Incorrectly Classified Instances	17,5%
Kappa statistic	0,65
Mean absolute error	0,174
Root mean squared error	0,4159
Relative absolute error	34,79%
Root relative squared error	83,18%
Total Number of Instances	40

Detailed Accuracy By Class									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,900	0,250	0,783	0,900	0,837	0,657	0,908	0,861	fall
	0,750	0,100	0,882	0,750	0,811	0,657	0,903	0,896	Random Peak
Weight. Avg.	0,825	0,175	0,832	0,825	0,824	0,657	0,905	0,879	

Confusion Matrix		
Fall	**Random Peak**	**← Classified as**
18	2	**Fall**
5	15	**Random Peak**

Πίνακας 14: Αποτελέσματα του Naive Bayes πάνω στο σύνολο εκμάθησης όσον αφορά τον εντοπισμό πτώσεων.

Simple Logistic στο σύνολο εκμάθησης.

Stratified cross-validation Summary	
Correctly Classified Instances	77.5%
Incorrectly Classified Instances	22.5%
Kappa statistic	0.55
Mean absolute error	0.2925
Root mean squared error	0.3976
Relative absolute error	58.5008 %
Root relative squared error	79.5189 %
Total Number of Instances	40

Detailed Accuracy By Class									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,900	0,350	0,720	0,900	0,800	0,568	0,863	0,875	fall
	0,650	0,100	0,867	0,650	0,743	0,568	0,863	0,856	Random Ppike
Weight. Avg.	0,775	0,225	0,793	0,775	0,771	0,568	0,863	0,865	

Confusion Matrix		
Fall	**Random Peak**	**← Classified as**
18	2	**Fall**
7	13	**Random Peak**

Πίνακας 15: Αποτελέσματα του Simple Logistic πάνω στο σύνολο εκμάθησης όσον αφορά τον εντοπισμό πτώσεων.

K-Nearest Neighbours στο σύνολο εκμάθησης.

Stratified cross-validation Summary	
Correctly Classified Instances	77.5%
Incorrectly Classified Instances	22.5%
Kappa statistic	0.55
Mean absolute error	0.2395
Root mean squared error	0.4624
Relative absolute error	47.8947 %
Root relative squared error	92.4879 %
Total Number of Instances	40

Detailed Accuracy By Class									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,850	0,300	0,739	0,850	0,791	0,556	0,775	0,703	fall
	0,700	0,150	0,824	0,700	0,757	0,556	0,775	0,726	Random Ppike
Weight. Avg.	0,775	0,225	0,781	0,775	0,774	0,556	0,775	0,715	

Confusion Matrix		
Fall	**Random Peak**	**← Classified as**
13	7	**Fall**
6	14	**Random Peak**

Πίνακας 16: Αποτελέσματα του K-Nearest Neighbours πάνω στο σύνολο εκμάθησης όσον αφορά τον εντοπισμό πτώσεων.

J48 στο σύνολο εκμάθησης

Stratified cross-validation Summary	
Correctly Classified Instances	72.5%
Incorrectly Classified Instances	27.5%
Kappa statistic	0.45
Mean absolute error	0.2861
Root mean squared error	0.5047
Relative absolute error	57.2138 %
Root relative squared error	100.9417 %
Total Number of Instances	40

Detailed Accuracy By Class									
	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0,750	0,300	0,714	0,750	0,732	0,451	0,713	0,639	fall
	0,700	0,250	0,737	0,700	0,718	0,451	0,713	0,688	Random Ppike
Weight. Avg.	0,725	0,275	0,726	0,725	0,725	0,451	0,713	0,663	

Confusion Matrix		
Fall	**Random Peak**	**← Classified as**
15	5	**Fall**
6	14	**Random Peak**

Πίνακας 17: Αποτελέσματα του J48 πάνω στο σύνολο εκμάθησης όσον αφορά τον εντοπισμό πτώσεων.

Κεφάλαιο 6. Συμπεράσματα

Όσον αφορά την αναγνώριση δραστηριοτήτων η μηχανική μάθηση είναι ιδιαίτερα αξιόπιστη σε αντίθεση με τον εντοπισμό πτώσεων όπου τα αποτελέσματα δεν ήταν ικανοποιητικά. Αυτό σημαίνει ότι το feature vector που χρησιμοποιήθηκε δεν ήταν κατάλληλο για τον εντοπισμό πτώσεων, επομένως είναι αναγκαία η προσθήκη κάποιων πιο εξειδικευμένων χαρακτηριστικών όπως για παράδειγμα του μετασχηματισμού Fourier.

Αναφορικά με το Pebble, οποιαδήποτε εφαρμογή συλλογής δεδομένων από το επιταχυνσιόμετρο που χρειάζεται να στέλνει περιοδικά τα δεδομένα στο smartphone, παρουσιάζει χαμηλή διάρκεια μπαταρίας. Για τον λόγο αυτό, η επεξεργασία των δεδομένων στο smartphone χρίζεται επίσης ακατάλληλη παρά τις αυξημένες δυνατότητες που προσφέρει οι οποίες θα μας επέτρεπαν να χρησιμοποιήσουμε μηχανική μάθηση και πολύπλοκους υπολογισμούς για το feature vector όπως μετασχηματισμό Fourier που αναφέρθηκε.

Για τους παραπάνω λόγους, η καταλληλότερη μέθοδος είναι η τοπική επεξεργασία επάνω στο Pebble με μόνο μειονέκτημα τον περιορισμό λόγω των χαμηλών προδιαγραφών του ρολογιού. Το γεγονός αυτό καθιστά ιδανική την εφαρμογή που προτείνεται στην παρούσα διπλωματική η οποία ενσωματώνει τον αλγόριθμο εντοπισμού πτώσεων με το απλό άθροισμα των απόλυτων τιμών των επιταχύνσεων και διαχωρίζει πότε ο χρήστης βρίσκεται στην κατάσταση asleep και awake χωρίς να ενσωματώνει κάποιον classifier και αποφεύγωντας οποιοδήποτε πολύπλοκο υπολογισμό. Η διάρκεια της μπαταρίας με αυτό τον τρόπο δεν επηρεάζεται.

Φυσική προέκταση της παρούσας διπλωματικής θα μπορούσε να αποτελέσει η ενσωμάτωση της μηχανικής μάθησης για την αναγνώριση δραστηριοτήτων με τους υπολογισμούς να γίνονται στο Pebble, ώστε να έχουμε διάκριση σε περισσότερες κατηγορίες. Η εφαρμογή αυτή θα παρουσιάζει όπως είναι φυσικό μικρότερη διάρκεια μπαταρίας με αντάλλαγμα την καλύτερη ακρίβεια στη διάκριση των δραστηριοτήτων. Στη συνέχεια θα μπορούσε να δημιουργηθεί εξωτερική εφαρμογή που θα μαθαίνει τις συνήθειες του χρήστη και θα μπορεί να εντοπίζει τυχόν παρεκκλίσεις ειδοποιώντας τον χρήστη ή κάποιον τρίτο όποτε το κρίνει απαραίτητο.

Βιβλιογραφία

[1] Simon Kozina, Hristijan Gjoreski, Matjaž Gams, Mitja Luštrek: Efficient Activity Recognition and Fall Detection Using Accelerometers (2013), Volume 386 of the series Communications in Computer and Information Science pp 13-23.

[2] Jay Chen, Karric Kwong, Dennis Chang, Jerry Luk, Ruzena Bajcsy: Wearable Sensors for Reliable Fall Detection (2005), Engineering in Medicine and Biology 27th Annual Conference.

[3] Ahmet Turan Özdemir and Billur Barshan: Detecting Falls with Wearable Sensors Using Machine Learning Techniques (2014), Multidisciplinary Digital Publishing Institute.

[4] Wikipedia, [Online]. Available:

https://en.wikipedia.org/wiki/Activities_of_daily_living. [Accessed: 3- Oct- 2016]

[5] Wikipedia, [Online]. Available:

https://en.wikipedia.org/wiki/Machine_learning. [Accessed: 3- Oct- 2016]

[6] Wikipedia, [Online]. Available:

https://en.wikipedia.org/wiki/Feature_extraction. [Accessed: 3- Oct- 2016]

[7] Wikipedia, [Online]. Available:

https://en.wikipedia.org/wiki/Pebble_(watch). [Accessed: 3- Oct- 2016]

[8] Wikipedia, [Online]. Available:

https://en.wikipedia.org/wiki/Accelerometer. [Accessed: 3- Oct- 2016]

[9] Sensor Wikipedia, [Online]. Available:
http://www.sensorwiki.org/doku.php/sensors/accelerometer. [Accessed: 3- Oct- 2016]

[10] Pebble Guides, [Online]. Available:

https://developer.pebble.com/guides/events-and-services/accelerometer. [Accessed: 3- Oct- 2016]

[11] Pebble Guides, [Online]. Available:

https://developer.pebble.com/guides/tools-and-resources/cloudpebble. [Accessed: 3- Oct- 2016]

[12] Pebble Documentation, [Online]. Available:

https://developer.pebble.com/docs/c/. [Accessed: 3- Oct- 2016]

[13] Pebble Guides, [Online]. Available:

https://developer.pebble.com/guides/events-and-services/events/. [Accessed: 3- Oct- 2016]

[14] Pebble Guides, [Online]. Available:
https://developer.pebble.com/guides/communication/using-pebblekit-js/. [Accessed: 3- Oct- 2016]

[15] Pebble Documentation, [Online]. Available:
https://developer.pebble.com/docs/c/Foundation/AppMessage. [Accessed: 3- Oct- 2016]

[16] Pebble Documentation, [Online]. Available:
https://developer.pebble.com/docs/c/Foundation/Dictionary/. [Accessed: 3- Oct- 2016]

[17] Pebble Documentation, [Online]. Available:
https://developer.pebble.com/docs/c/Foundation/AppSync/. [Accessed: 3- Oct- 2016]

[18] Pebble Documentation, [Online]. Available: https://developer.pebble.com/docs/c/Foundation/DataLogging/. [Accessed: 3- Oct- 2016]

[19] Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/JSON#Using_JSON_in_JavaScript. [Accessed: 3- Oct- 2016]

[20] Wikipedia, [Online]. Available:

https://en.wikipedia.org/wiki/MongoDB. [Accessed: 3- Oct- 2016]

[21] Wikipedia, [Online]. Available:

https://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed: 3- Oct- 2016]

[22] Wikipedia, [Online]. Available:

https://en.wikipedia.org/wiki/Spring_Framework. [Accessed: 3- Oct- 2016]

[23] Spring Guides, [Online]. Available:

https://spring.io/guides. [Accessed: 3- Oct- 2016]

[24] Wikipedia, [Online]. Available:

https://en.wikipedia.org/wiki/Ajax_(programming). [Accessed: 3- Oct- 2016]

[25] Wikipedia, [Online]. Available:

https://en.wikipedia.org/wiki/Cross-origin_resource_sharing. [Accessed: 3- Oct- 2016]

[26] Tinetti ME, Liu WI, Claus EB (1993) Predictors and prognosis of inability to get up after falls among elderly persons. J Am Med Ass 269: 65–70

[27] Wild D, Nayak US, Isaacs B (1981) How dangerous are falls in old people at home? Br Med J (Clin Res Ed) 282(6260): 266–268.