ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη Συστήματος Ζωντανής και Ασύρματης Παρακολούθησης Αισθητήρων με Πρόβλεψη Μετρήσεων

Διπλωματική εργασία

-

Παναγιώτης Προύντζος

Επιβλέπων Καθηγητής

Ευάγγελος Β. Χριστοφόρου
Καθηγητής Ε.Μ.Π

Αθήνα, Ιούνιος 2025

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ
ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ανάπτυξη Συστήματος Ζωντανής και Ασύρματης Παρακολούθησης Αισθητήρων με Πρόβλεψη Μετρήσεων

Διπλωματική εργασία

-

Παναγιώτης Προύντζος

Επιβλέπων : Ευάγγελος Β. Χριστοφόρου
Καθηγητής Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή στις 30 Ιουνίου 2025

.....	.....	.....
Ευάγγελος Β. Χριστοφόρου Καθηγητής Ε.Μ.Π	Γεώργιος Παναγόπουλος Επίκουρος Καθηγητής Ε.Μ.Π	Εμμανουήλ Χουρδάκης Επίκουρος Καθηγητής Ε.Μ.Π

Αθήνα, Ιούνιος 2025

Δήλωση Λογοκλοπής

.....
Παναγιώτης Προύντζος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών ΕΜΠ

Περίληψη

Σκοπός αυτής της εργασίας είναι η ανάπτυξη του PRISM (Predictive Realtime Industrial Sensor Monitor), ενός ασύρματου συστήματος ζωντανής παρακολούθησης αισθητήρων σε διάφορα βιομηχανικά σενάρια, με έμφαση σε αισθητήρες μαγνητικών πεδίων, και μεταλλικές κατασκευές. Το σύστημα αυτό βασίζεται σε μικροελεγκτές ESP32 για την οδήγηση των αισθητήρων, ενώ η ασύρματη επικοινωνία υλοποιείται με την χρήση του πρωτοκόλλου MQTT (Message Queue Telemetry Transport). Επιπλέον, το λογισμικό αυτό παρέχει αλγορίθμους πρόβλεψης μελλοντικών μετρήσεων σε πραγματικό χρόνο. Το σύστημα PRISM είναι υλοποιημένο στην γλώσσα C++ για την επίτευξη βέλτιστων επιδόσεων, χαμηλής χρήσης υπολογιστικών πόρων, και φορητότητας σε διάφορες πλατφόρμες, π.χ. Raspberry Pi.

Αρχικά, θα γίνει μια σύντομη ανάλυση του θεωρητικού υποβάθρου των ηλεκτρομαγνητικών αρχών που διέπουν τους αισθητήρες AMR, οι οποίοι θα χρησιμοποιηθούν στο πειραματικό μέρος αυτής της εργασίας, για την λήψη δεδομένων. Στην συνέχεια θα περιγραφεί η ανάπτυξη των drivers των αισθητήρων LSM303DLHC,της εφαρμογής που θα χειρίζεται την αποστολή δεδομένων απο το ESP32 στον client του PRISM, καθώς και η σχεδίαση της συσκευασίας του αισθητήρα. Μετά θα γίνει αναλυτική παρουσίαση της αρχιτεκτονικής και της ανάπτυξης του συστήματος PRISM. Στο τέλος θα δοθούν μια τεχνική κριτική για την υλοποίηση όλων των προαναφερθέντων συστημάτων, τα συμπεράσματα που μπορούμε να εξάγουμε απο τις ληφθείσες μετρήσεις, καθώς και τα GitHub Repositories που αναπτύχθηκαν.

Λέξεις Κλειδιά: PRISM, ESP32, LSM303, MQTT, C++, AMR Sensor, Μαγνητικός αισθητήρας, Χάλυβας, SteHeMon

Abstract

The aim of this thesis is the development of PRISM (Predictive Realtime Industrial Sensor Monitor), a wireless and live sensor monitoring system designed for various industrial scenarios, with a focus on magnetic field sensors and metallic structures. The system is based on ESP32 microcontrollers for sensor interfacing, while wireless communication is implemented using the MQTT (Message Queue Telemetry Transport) protocol. Additionally, the software provides algorithms for real-time prediction of future measurements. PRISM is developed in C++, aiming for optimized performance, low resource utilization, and portability across platforms such as the Raspberry Pi.

The work begins with a brief theoretical background on the electromagnetic principles governing AMR (Anisotropic Magnetoresistive) sensors, which are used in the experimental part of the thesis for data acquisition. This is followed by the development of drivers for the LSM303DLHC sensors, the implementation of the application responsible for transmitting data from the ESP32 to the PRISM client, and the design of the sensor's 3D-printed enclosure. Next, a detailed presentation of PRISM's architecture and development process is provided. Finally, a technical evaluation of all system components is offered, followed by conclusions derived from the collected measurements as well as links to the relevant GitHub repositories which were developed.

Keywords: PRISM, ESP32, LSM303, MQTT, C++, AMR Sensor, Magnetic Sensor, Steel, SteHeMon

Ευχαριστίες

Θέλω να εκφράσω τις ευχαριστίες μου για τους γονείς μου και τον αδελφό μου, για την συνεχή τους υποστήριξη, αγάπη και καθοδήγηση.
Επιπλέον θέλω να ευχαριστήσω την Μαρία Τσιρώνη απο το εργαστήριο αισθητήρων για την βοήθεια και την συνεργασία της σε αυτή την εργασία.
Τέλος, θέλω να ευχαριστήσω τον κύριο Ε. Χριστοφόρου για την πλήρη εμπιστοσύνη που μου έδειξε για την σχεδίαση και συγγραφή αυτής της εργασίας.

Παναγιώτης Προύντζος
30 Ιουνίου 2025

Περιεχόμενα

1 Αισθητήρες Μαγνητικού Πεδίου και θεωρία

1.1 Εισαγωγή

Οι μαγνητικοί αισθητήρες βρίσκουν εφαρμογές σε ένα μεγάλο πλήθος εφαρμογών, από την ρομποτική, οπου επιτρέπουν τον ακριβή έλεγχο των κινητήρων και των αρθρώσεων, την βιοιατρική, επιτρέποντας την μαγνητοεγκεφαλογραφία, έως και την πλοήγηση οχημάτων [1]. Οι αισθητήρες αυτοί, εκμεταλλεύονται μερικές από τις βασικές αρχές του ηλεκτρομαγνητισμού, για να μετατρέψουν το μαγνητικό πεδίο σε ένα σημείο του περιβάλλοντος σε ένα ηλεκτρικό σήμα, ψηφιακό ή αναλογικό. Και για την κατανόηση της λειτουργίας τους, είναι απαραίτητη μια θεωρητική ανασκόπηση των μαγνητικών υλικών. Σκοπός αυτού του κεφαλαίου είναι η παρουσίαση αυτού του θεωρητικού υποβάθρου, και η ανάλυση της λειτουργίας των μαγνητικών αισθητήρων, για να είναι πιο κατανοητή η εφαρμογή του συστήματος που θα χτίσουμε.

1.2 Θεμελιώδεις αρχές Ηλεκτρομαγνητισμού

Τα ηλεκτρομαγνητικά μεγέθη που μας ενδιαφέρουν για την κατανόηση των αισθητήρων είναι τα ακόλουθα:

- $\vec{B}$: Πυκνότητα μαγνητικής επαγωγής
- $\vec{H}$: Ένταση μαγνητικού πεδίου
- $\vec{E}$: Ηλεκτρικό πεδίο
- $\vec{M}$: Μαγνήτιση

Το ηλεκτρικό πεδίο Ε περιγράφει την ηλεκτροστατική δύναμη που που ασκείται σε ένα φορτίο από άλλα ηλεκτρικά φορτία, και μετράται σε $\frac{V}{m}$.

Η πυκνότητα μαγνητικής επαγωγής Β εκφράζει την ένταση του μαγνητικού πεδίου που παράγεται από μεταβαλλόμενα ηλεκτρικά πεδία, και μετράται σε Τ.

Η ένταση μαγνητικού πεδίου Η περιγράφει το μαγνητικό πεδίο αγνοώντας την επίδραση του υλικού, και μετράται σε $\frac{A}{m}$.

Τέλος η μαγνήτιση Μ περιγράφει το σύνολο της μαγνητική ροπής που δημιουργείται σε ένα υλικό από ένα εξωτερικό πεδίο Η. Μετράται σε $\frac{A}{m}$.

Οι ποσότητες αυτές συνδέονται με τις εξισώσεις του Maxwell[2]:

- Νόμος του Gauss : $\nabla \cdot E = \frac{\rho}{\varepsilon 0}$
- Νόμος του Gauss για τον μαγνητισμό : $\nabla \cdot B = 0$
- Νόμος του Faraday : $\nabla \times E = -\frac{\partial B}{\partial t}$
- Νόμος Ampere Maxwell : $\nabla \times B = \mu 0(J + \varepsilon 0 \frac{\partial E}{\partial t})$

Ο νόμος του Gauss εξηγεί ότι οι γραμμές του ηλεκτρικού πεδίου ορίζονται από τις θέσεις των ηλεκτρικών φορτίων, και οτι η ροή του ηλεκτρικού πεδίου σε μια κλειστή επιφάνεια είναι ανάλογη του φορτίου στο εσωτερικό αυτής.
Ο νόμος του Gauss για τον μαγνητισμό εξηγεί ότι οι μαγνητικές γραμμές πάντα ορίζουν κλειστούς βρόγχους. Επομένως δεν μπορεί να υπάρχει "μαγνητικό μονόπολο".
Ο νόμος του Faraday συνδέει το ηλεκτρικό πεδίο με το μαγνητικό, εξηγώντας οτι η μεταβολή της ροής του μαγνητικού πεδίου σε μία επιφάνεια είναι ανάλογη του ηλεκτρικού πεδίου που δημιουργείται στο σύνορο αυτής.
Τέλος, ο νομος Ampere Maxwell εξηγεί ότι τα μαγνητικά πεδία δημιουργούνται απο τις μεταβολές στα ηλεκτρικά πεδία, ακόμα και χωρίς κάποιο ρεύμα(π.χ σε έναν πυκνωτή).
Επιπλέον, τα μεγέθη B, M και H συνδέονται με τις ακόλουθες σχέσεις:

- $\vec{B} = \mu 0(\vec{H} + \vec{M})$
- $\vec{M} = \mathrm{x_m}\vec{H}$

όπου $\mu 0 = 4\pi * 10^{-7}$ η μαγνητική διαπερατότητα του κενού,

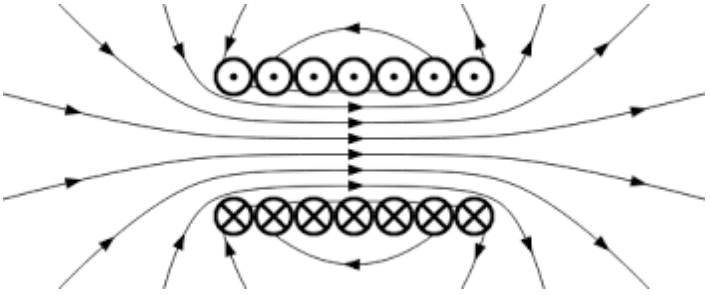
$\varepsilon 0 \approx 8.854 * 10^{-12}$ η ηλεκτρική επιτρεπτότητα του κενού, και

$\mathrm{x_m}$ η μαγνητική επιδεκτικότητα του υλικού στο οποίο εφαρμόζεται το πεδίο H. Η ηλεκτρική επιτρεπτότητα του κενού είναι μια σταθερά η οποία περιγράφει πόσο εύκολα μπορεί να δημιουργηθεί ένα ηλεκτρικό πεδίο στο κενό, σύμφωνα με τον τύπο:

$$E = \frac{1}{4\pi\varepsilon 0}\frac{q}{r^2}$$

ενώ η μαγνητική διαπερατότητα περιγράφει την ευκολία με την οποία επάγονται μαγνητικά πεδία από ρεύματα, σύμφωνα με τον τύπο:

$$B = \frac{\mu 0 * I}{2\pi r}$$

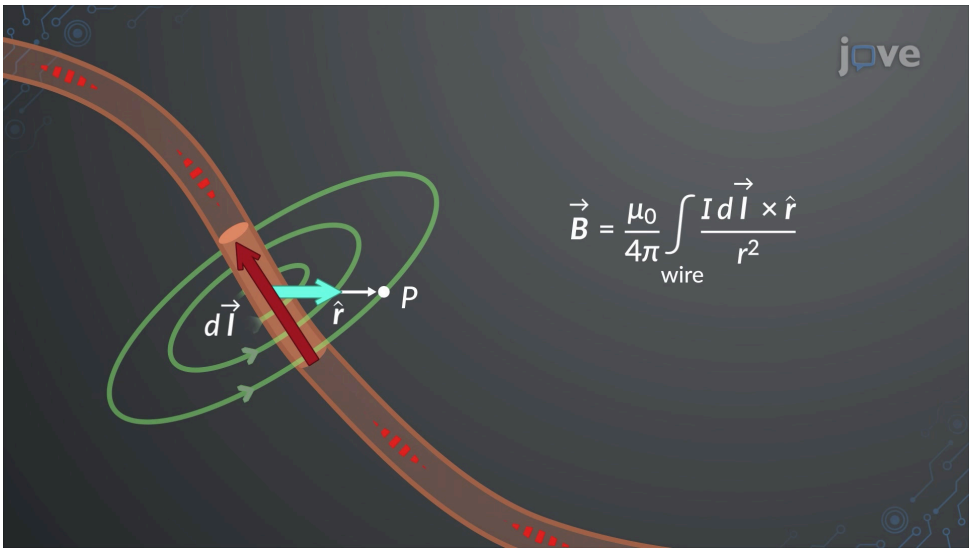


Σχήμα 1. Απεικόνιση ηλεκτρομαγνητικού πεδίου σε πηνίο

Θεωρούμε πως όταν $\mathrm{x_m} < 0$ το υλικό είναι διαμαγνητικό, ενώ όταν $\mathrm{x_m} > 0$ το υλικό είναι φερρομαγνητικό.
Οι παραπάνω τέσσερις εξισώσεις αποτελούν την βάση για ολόκληρο τον ηλεκτρομαγνητισμό, και συνεπώς για τους ηλεκτρονικούς αισθητήρες που μας ενδιαφέρουν στα πλαίσια αυτής της εργασίας.

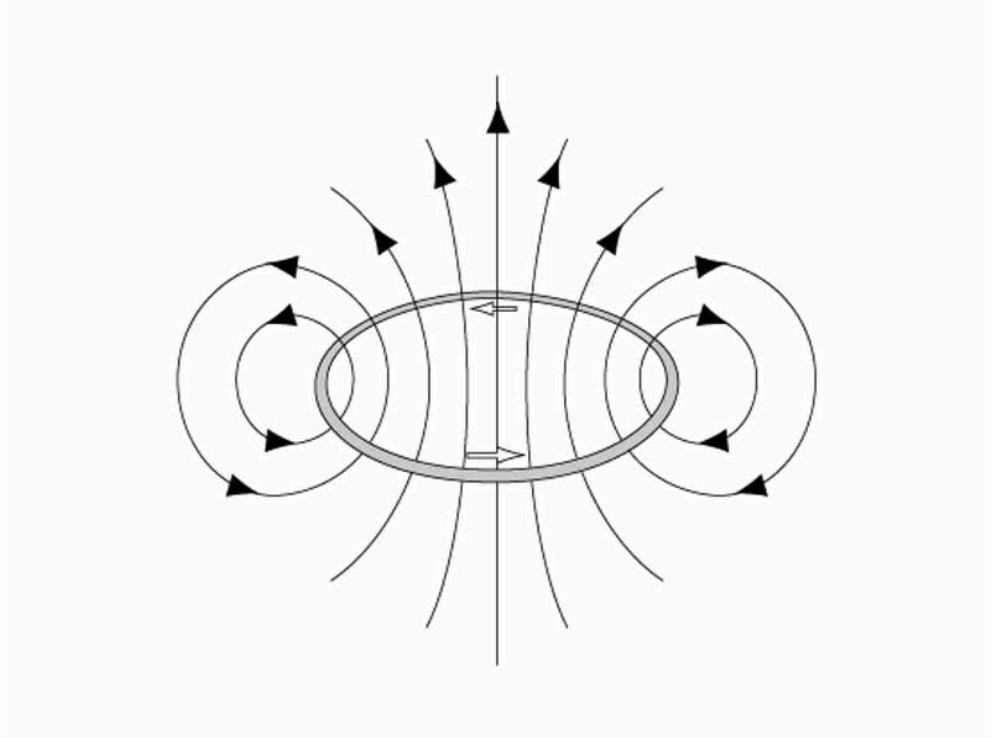
Ένα μαγνητικό πεδίο προκύπτει απο την ροή ηλεκτρικού ρεύματος, σύμφωνα με τον νόμο Biot - Savart[2]. Ο νόμος αυτός περιγράφει πώς ένα γραμμικό ρεύμα δημιουργεί ένα μαγνητικό πεδίο B :

- $B(r) = \frac{\mu 0}{4\pi} I \int \frac{dl \times \hat{r}}{r^2}$



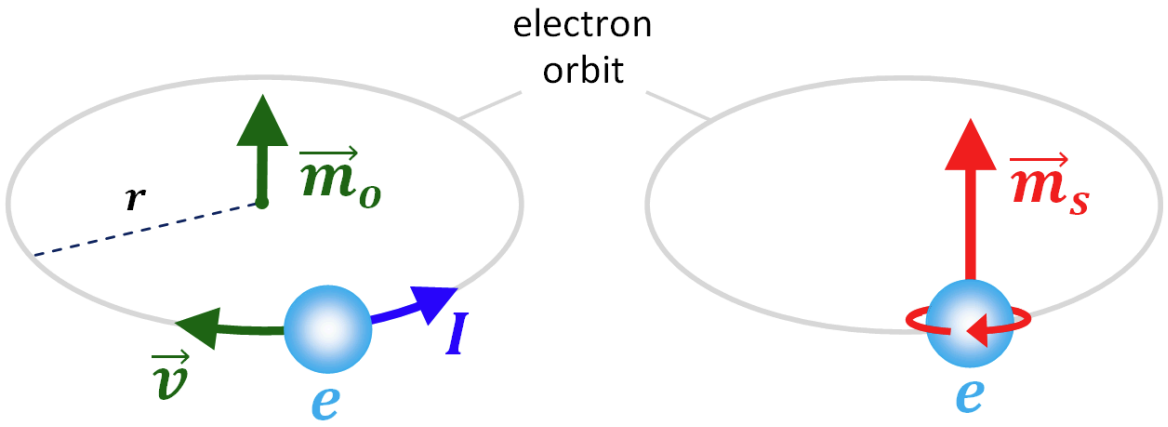
Σχήμα 1. Απεικόνιση του νόμου Biot Savart

Όπως φαίνεται και από το παραπάνω σχήμα, το μαγνητικό πεδίο "τυλίγεται" γύρω από τον αγωγό που φέρει το ρεύμα. Συνέπεια αυτού είναι ότι οι βρόγχοι ρεύματος δημιουργούν μαγνητικό πεδίο κάθετο στο επίπεδο που κινείται το ρεύμα. Αυτό θα μας φανεί χρήσιμο στον ορισμό των μαγνητικών ροπών.



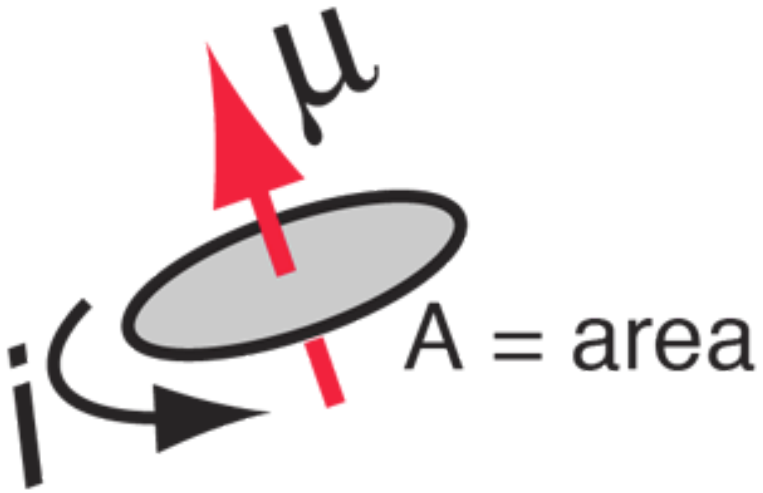
Σχήμα 2. Μαγνητικό πεδίο βρόγχου ρεύματος

Μια ακόμα έννοια που μας ενδιαφέρει για να περιγράψουμε την απόκριση υλικών σε εξωτερικά μαγνητικά πεδία είναι η μαγνητική ροπή. Η μαγνητική ροπή είναι μια έννοια η οποία είναι αρκετά χρήσιμη για την κατανόηση των μαγνητικών υλικών, καθώς εκφράζει την ισχύ και τον προσανατολισμό ενός μαγνητικού πεδίου σε μικροσκοπική κλίμακα. Πιο συγκεκριμένα, το κάθε ηλεκτρόνιο ενός υλικού, δημιουργεί το δικό του μαγνητικό πεδίο. Αν θεωρήσουμε ότι αυτό περιστρέφεται γύρω από τον πυρήνα του ατόμου, ορίζουμε έναν βρόχο ρεύματος. Επιπλέον, από το ίδιο το spin του ηλεκτρονίου, δημιουργείται ένα μαγνητικό πεδίο.



Σχήμα 3. Μαγνητικό πεδίο του ηλεκτρονίου

Η μαγνητική ροπή συμβολίζεται ως μ, και χρησιμοποιείται για την περιγραφή της ροπής που ασκείται σε έναν τέτοιο βρόχο ρεύματος[3]



Σχήμα 4. Μαγνητική ροπή βρόχου

Η ροπή που ασκείται στο βρόχο του σχήματος ισούται με

$\tau = \mu \times B$

όπου B το εξωτερικό μαγνητικό πεδίο, με το οποίο προσπαθεί να προσανατολιστεί ο βρόχος. Ο βρόχος προσπαθεί να προσανατολιστεί λόγω του δικού του μαγνητικού πεδίου, το οποίο

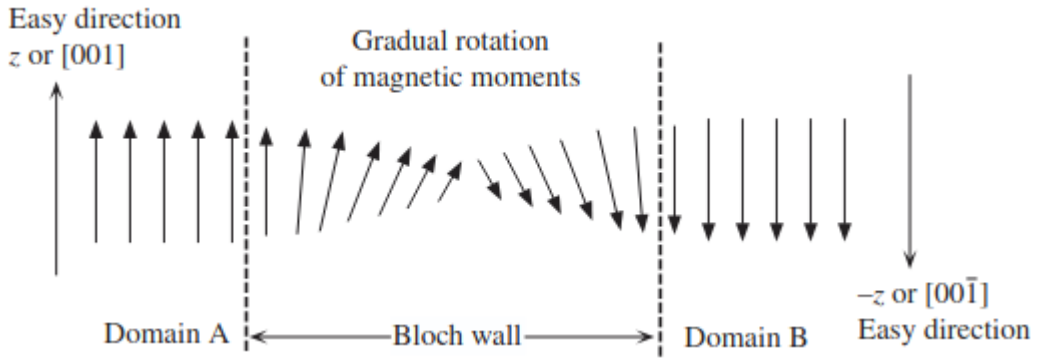
προκύπτει από το ρεύμα του, i, σύμφωνα με τον νόμο του Ampere, $\nabla \times B = \mu 0 J$, σε απουσία ηλεκτρικού πεδίου.

1.3 Μαγνητικά υλικά και η απόκριση τους

Για να δώσουμε τον ορισμό των μαγνητικών υλικών, αρχικά πρέπει να δώσουμε έναν καλό ορισμό της μαγνήτισης. Η μαγνήτιση είναι ένα διανυσματικό πεδίο που περιγράφει την πυκνότητα των μονίμων ή επαγόμενων μαγνητικών ροπών μέσα σε ένα υλικό.
Η μαγνητιση ενός υλικού ορίζεται από την σχέση:

$$\vec{M} = \frac{d\vec{m}}{dV}$$

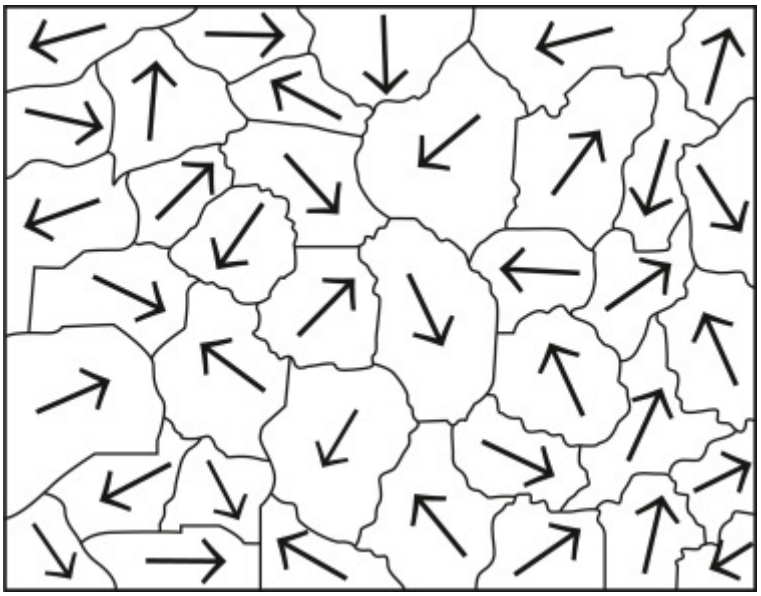
όπου m η μαγνητική ροπή μιας περιοχής του υλικού, στην οποία θεωρούμε ότι το m είναι σταθερό, και
dV ένα διαφορικό όγκου στο υλικό.
Έχοντας ορίσει την μαγνήτιση ενός υλικού, μπορούμε να κατηγοριοποιήσουμε τα μαγνητικά υλικά ανάλογα με τα χαρακτηριστικά τους. Πριν το κάνουμε αυτό, πρέπει να καταλάβουμε την γεωμετρία αυτών των υλικών, και την έννοια των μαγνητικών περιοχών. Οι μαγνητικές περιοχές είναι μικροσκοπικές περιοχές στο εσωτερικό του υλικού στις οποίες οι μαγνητικές ροπές των ατόμων είναι ευθυγραμμισμένες μεταξύ τους. Κάθε περιοχή έχει το δικό της σταθερό διάνυσμα μαγνήτισης M. Αυτό συμβαίνει για την ελαχιστοποίηση της συνολικής μαγνητοστατικής ενέργειας του υλικού αυτού[4-5]. Οι μαγνητικές περιοχές διαχωρίζονται από τα τοιχώματα Bloch. Προφανώς, η διαφορά δύο διαδοχικών μαγνητικών ροπών σε δύο διαφορετικές περιοχές δεν είναι ακαριαία. Οι μαγνητικές ροπές μεταβάλλονται ομαλά και αργά μεταξύ δύο περιοχών. Τα τοιχωματα Bloch είναι οι περιοχές στις οποίες γίνεται αυτή η μεταβολή[6-7]



Σχήμα 5. Τοιχωμα Bloch

Τυπικά, τα τοιχώματα Bloch είναι της τάξης των δεκάδων νανομέτρων, έως και μικρομέτρων. Αυτά μπορούν να μετακινούνται μέσα στο υλικό με την χρήση ενός εξωτερικού πεδίου H[8]. Πιο συγκεκριμένα, οι περιοχές στις οποίες η μαγνήτιση M έχει παρόμοια κατεύθυνση με το εξωτερικό πεδίο H, επεκτείνονται εις βάρος των περιοχών που έχουν κάθετη / αντίθετη κατεύθυνση. Προφανώς, η κίνηση αυτή δεν είναι ομαλή επειδή καθορίζεται από τις ανωμαλίες του υλικού. Ο μηχανισμός της κίνησης των τοιχωμάτων Bloch είναι σημαντικός για την κατανόηση των φαινομένων υστέρησης.

Το παρακάτω σχήμα δινει μια απλή απεικόνιση των μαγνητικών περιοχών ενός υλικού:



Σχήμα 6. Μαγνητικές περιοχές υλικού

Η συνολική μαγνήτιση αυτών των περιοχών έχει την ίδια κατεύθυνση με τις μαγνητικές ροπές. Όπως αναφέραμε προηγουμένως, η μαγνήτιση του υλικού εξαρτάται από το εξωτερικό μαγνητικό πεδίο σύμφωνα με την ακόλουθη σχέση:

$$\vec{M} = \chi_m \vec{H}$$

Μπορούμε πλέον να ξεχωρίσουμε τα υλικά ανάλογα με τις προαναφερθείσες ιδιότητες και την παραπάνω σχέση στις εξής κατηγορίες:

- Διαμαγνητικά

Τα διαμαγνητικά υλικά έχουν μια πολύ ασθενή απόκριση στο εξωτερικό μαγνητικό πεδίο, ενώ όταν αυτό είναι μηδενικό, η μαγνήτιση τους είναι πάλι κοντά στο 0(τυχαίος προσανατολισμός των περιοχών). Η μαγνητική επιδεκτικότητα τους λοιπόν είναι αρνητική και κοντά στο 0.

- Παραμαγνητικά

Τα παραμαγνητικά υλικά επίσης έχουν μικρή απόκριση στο εξωτερικό μαγνητικό πεδίο, με την διαφορά ότι αυτή είναι πλέον θετική($\chi_m > 0$). Η μαγνήτιση τους είναι ασθενής, και χάνεται πλήρως με τον μηδενισμό του εξωτερικού μαγνητικού πεδίου(Δεν έχουν παραμένων μαγνητισμό)

- Αντι Φερρομαγνητικά

Τα αντι-φερρομαγνητικά υλικά είναι υλικά στα οποία οι περιοχές μαγνήτισης δεν είναι τυχαία προσανατολισμένες, αλλα είναι αντιπαράλληλες μεταξύ τους, με αποτέλεσμα η μαγνήτιση του υλικού να είναι κοντά στο 0. Η επιδεκτικότητα τους είναι μικρή, και θετική.

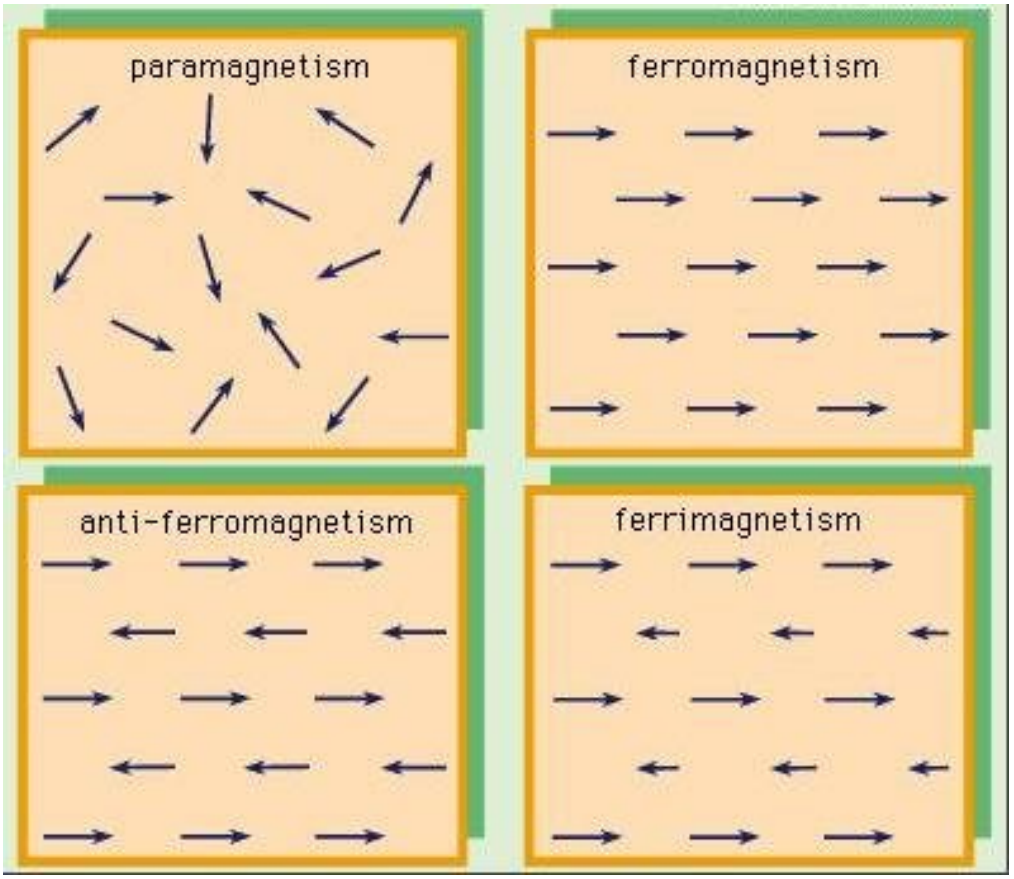
- Φερρομαγνητικά

Τα φερρομαγνητικά υλικά έχουν δυνατή και μόνιμη μαγνήτιση, ακόμα και όταν αφαιρείται το εξωτερικό πεδίο. Οι περιοχές μαγνήτισης τους έχουν παρόμοιο προσανατολισμό, με αποτέλεσμα η συνολική μαγνήτιση να είναι πολύ μεγάλη. Η επιδεκτικότητα τους είναι μεγάλη.

- Φερριμαγνητικά

Τέλος, τα φερριμαγνητικά υλικά είναι παρόμοια με τα αντι-φερρομαγνητικά υλικά, όμως οι αντιπαράλληλες ροπές δεν έχουν ίδιο μέτρο, με αποτέλεσμα να υπάρχει συνολική μαγνήτιση, όχι όμως τόσο μεγάλη όσο στα φερρομαγνητικά υλικά. Η επιδεκτικότητα τους είναι θετική.

Ως εκ τούτου, στους μαγνητικούς αισθητήρες, συνήθως χρησιμοποιούνται τα φερριμαγνητικά και τα φερρομαγνητικά υλικά, ακριβώς επειδή είναι "ευαίσθητα" στις αλλαγές του εξωτερικού μαγνητικού πεδίου, αλλα και επειδή εμφανίζουν υστέρηση.



Σχήμα 7. Μαγνητικές ροπές των υλικών

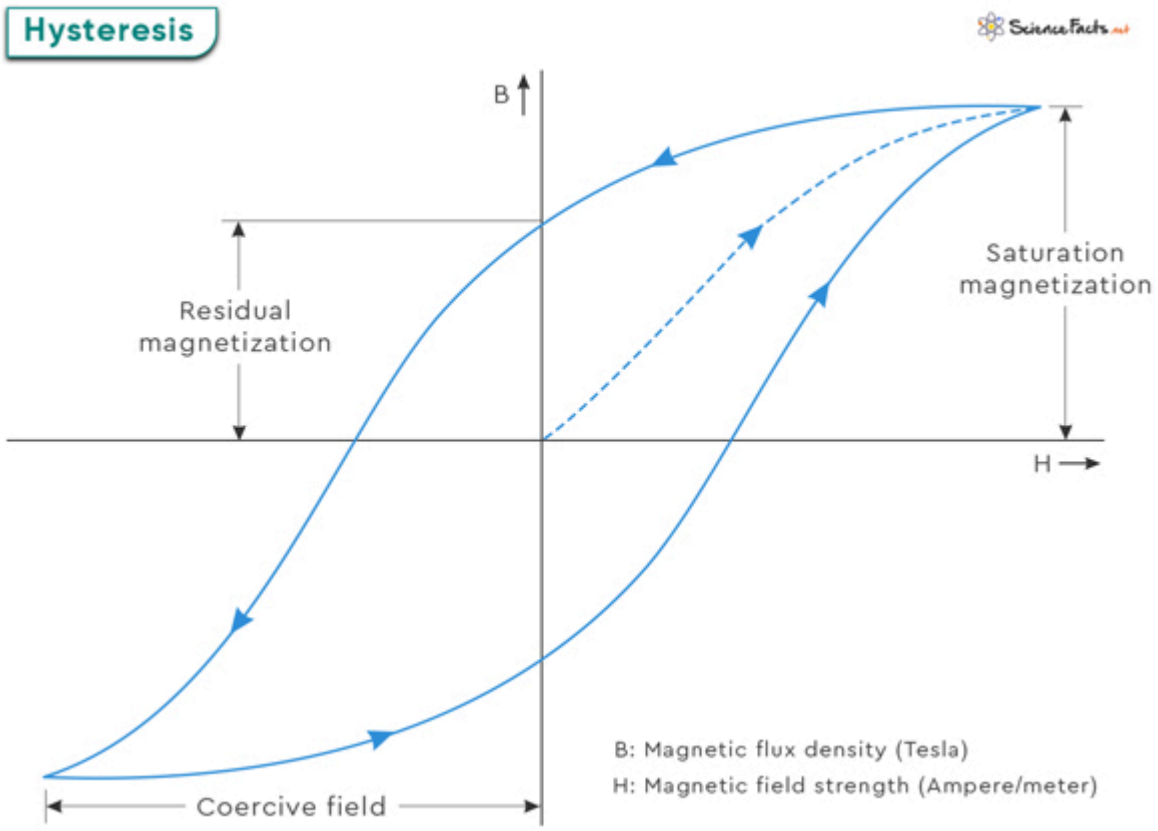
Για να κατανοήσουμε καλύτερα τα μαγνητικά υλικά και πως χρησιμοποιούνται για την κατασκευή αισθητήρων, πρέπει να αναφέρουμε τον βρόχο υστέρησης. Ο βρόχος υστέρησης ενός υλικού περιγράφει την σχέση μεταξύ της επαγόμενης μαγνήτισης M και του εφαρμοζόμενου μαγνητικού πεδίου H. Η εξίσωση που αναφέραμε προηγουμένως,

$$\vec{M} = \mathrm{x_m}\vec{H}$$

περιγράφει μια γραμμική εξάρτηση του εξωτερικού πεδίου και της μαγνήτισης. Αυτή η σχέση δεν περιγράφει πλήρως τα μαγνητικά υλικά, καθώς η γραμμική τους φύση αγνοεί τον παραμένοντα μαγνητισμό. Οι περιπτώσεις στις οποίες ισχύει είναι οι εξής:

- Υλικά χωρίς μνήμη (Παραμαγνητικά, διαμαγνητικά)
- Πολύ μικρές τιμές του H (Πολύ μικρή ένταση για να μετακινηθούν οι περιοχές)

Επομένως, ένα πιο εύστοχο μοντέλο για τα μαγνητικά υλικά είναι ο βρόχος υστέρησης. Ενώ δεν υπάρχει κάποιος γενικός τύπος που να περιγράφει τον βρόχο υστέρησης, ακολουθεί μια σιγμοειδή καμπύλη[9], όπως φαίνεται στο παρακάτω σχήμα:



Σχήμα 8. Βρόχος υστέρησης

Όπως φαίνεται στο σχήμα η σχέση του πεδίου Β ώς πρός το Η δεν είναι γραμμική. Με την αύξηση του εξωτερικού πεδίου Η αυξάνεται η μαγνήτιση του υλικού, έως ένα σημείο, στο οποίο επιτυγχάνεται κορεσμός. Περαιτέρω αύξηση του Η δεν αυξάνει την μαγνήτιση. Η μείωση του Η μειώνει την μαγνήτιση. Όταν το Η μηδενιστεί, η μαγνήτιση θα έχει μια μη μηδενική τιμή, η οποία αναπαριστά την παραμένουσα μαγνήτιση. Περαιτέρω μείωση του Η μειώνει την μαγνήτιση, ξανά, μέχρι τον κορεσμό. Ο ίδιος μηχανισμός ισχύει και στην αύξηση του Η απο το σημείο του αρνητικού κορεσμού. Η αρχική απόκριση του υλικού φαίνεται να είναι γραμμική όμως. Αυτό γίνεται πρίν δημιουργηθεί παραμένων μαγνητισμός. Ο μόνος τρόπος να μηδενιστεί η μαγνήτιση ενός υλικού είναι είτε η μείωση του εξωτερικού πεδίου στο σημείο που το Μ γίνεται 0 ή η θέρμανση του σε θερμοκρασίες μεγαλύτερες της θερμοκρασίας curie Tc του υλικού, στην οποία περίπτωση το υλικό γίνεται παραμαγνητικό[10].

Ανάλογα με την καμπύλη μαγνήτισης ενός υλικού, μπορούμε να κάνουμε την παρακάτω κατηγοριοποίηση:

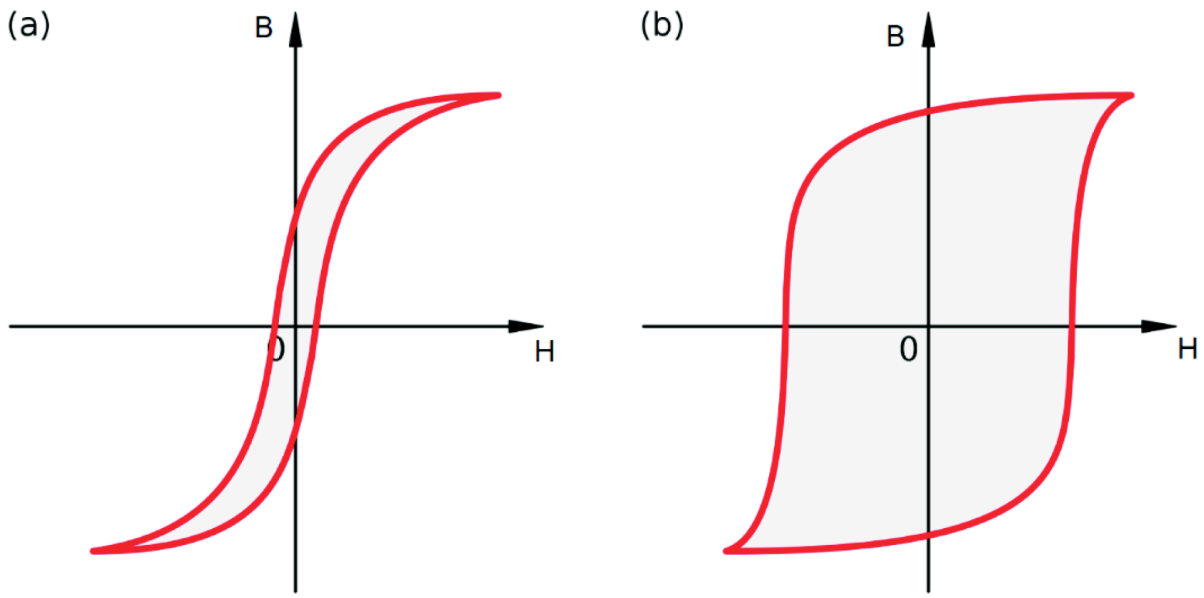
- Μαλακά μαγνητικά υλικά

Υλικά τα οποία είναι πολύ εύκολο να μαγνητιστούν

- Σκληρά μαγνητικά υλικά

Υλικά τα οποία είναι δύσκολο να μαγνητιστούν.

Η διαφορά των δύο είναι το σημείο κορεσμού της μαγνήτισης. Προφανώς, τα μαλακά μαγνητικά υλικά έχουν μικρό σημείο κορεσμού, ενώ τα σκληρά μαγνητικά υλικά έχουν μεγάλο σημείο κορεσμού



Σχήμα 9. Βρόχος υστέρησης μαλακού και σκληρού μαγνητικού υλικού

Η απόκριση των μαλακών μαγνητικών υλικών είναι αρκετά γρήγορη, οπότε αυτά βρίσκουν εφαρμογή σε[11]:

- Μετασχηματιστές
- Πηνία
- Ηλεκτρομαγνήτες

Η απόκριση των σκληρών μαγνητικών υλικών είναι αργή, αλλα ο παραμένων μαγνητισμός είναι αρκετά μεγάλος. Για αυτό βρίσκουν εφαρμογή σε

- Μόνιμους μαγνήτες
- Ηλεκτρικούς κινητήρες
- Μαγνητικοί δίσκοι

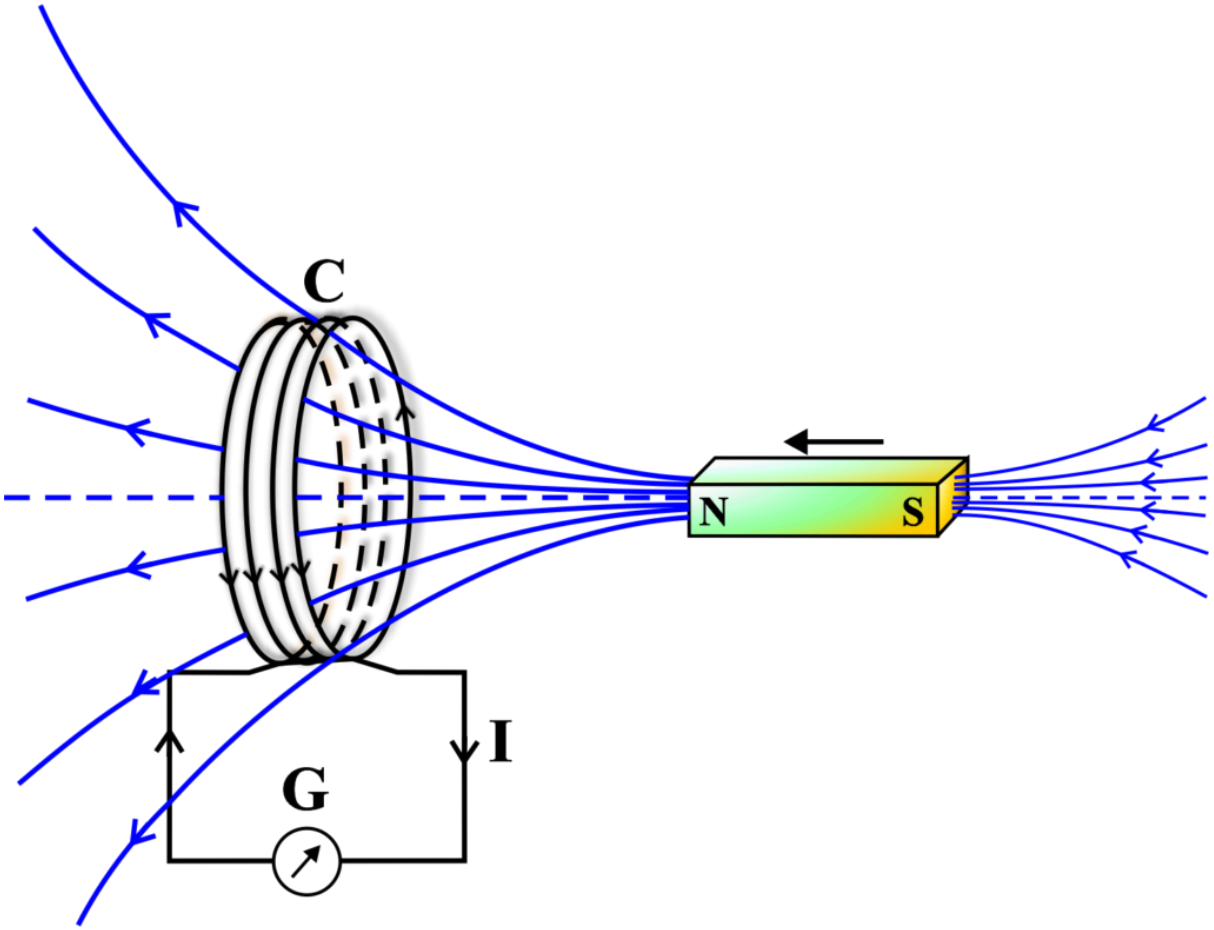
1.4 Τυπικές αρχιτεκτονικές αισθητήρων Μαγνητικού πεδίου

Έχοντας εξηγήσει τις βασικές έννοιες του μαγνητισμού και των μαγνητικών υλικών, μπορούμε να προχωρήσουμε στην ανάλυση των διαφόρων ειδών μαγνητικών αισθητήρων. Για αυτό θα χρειαστούμε έναν απλό ορισμό των αισθητήρων:
Αισθητήρας ονομάζεται μια συσκευή η οποία πραγματοποιεί μια μέτρηση ενός φυσικού μεγέθους και την μετατρέπει αυτόματα σε ένα ηλεκτρικό σήμα[12]. Τα σήματα αυτά μπορούν να χρησιμοποιηθούν για διάφορες εργασίες όπως την παρακολούθηση του φυσικού μεγέθους(ο σκοπός αυτής της εργασίας) ή για τον έλεγχο ενός αυτόματου συστήματος κλειστού βρόχου, μεταξύ άλλων. Μαγνητικοί αισθητήρες, λοιπόν, είναι οι αισθητήρες οι οποίοι περιέχουν ένα (συνήθως) φερρομαγνητικό υλικό, το οποίο έχει μια ορισμένη απόκριση στην μεταβολή ενός εξωτερικού μαγνητικού πεδίου H[12]. Οι μεταβολές αυτές αντιστοιχίζονται σε ένα ηλεκτρικό σήμα. Η αντίστροφη αντιστοίχιση λοιπόν μας δίνει μια προσέγγιση του μαγνητικού πεδίου. Οι μαγνητικοί αισθητήρες είναι ιδιαίτερα ευαίσθητοι,

και χρειάζονται προσεκτική ρύθμιση για την σωστή τους λειτουργία. Με αυτόν τον ορισμό λοιπόν, μπορούμε να χωρίσουμε τους αισθητήρες στις ακόλουθες κατηγορίες[13]:

- Αισθητήρες επαγωγής
- Αισθητήρες τύπου fluxgate
- Αισθητήρες σιδηρομαγνητικής μαγνητοαντίστασης
- Αισθητήρες φαινομένου Hall

Ξεκινώντας με τους επαγωγικούς αισθητήρες, αυτοί εκμεταλλεύονται τον νόμο του Faraday για να μετατρέψουν την ροή ενός πεδίου μέσα από ένα υλικό σε μια τάση



Σχήμα 10. Αναπαράσταση του νόμου του Faraday

Ο νόμος του Faraday, όπως αναφέραμε προηγουμένως, περιγράφει την επαγόμενη τάση σε έναν κλειστό βρόχο(στην συγκεκριμένη περίπτωση ένα πηνίο) συναρτήσει της ροής του εξωτερικού μαγνητικού πεδίου. Συγκεκριμένα, ισχύει:

$$\oint_{\partial S} E \cdot dl = -\frac{d}{dt}\int_{S} B \cdot dA$$

Αυτό σημαίνει πως η επαγόμενη τάση σε έναν κλειστό βρόχο, ισούται με τον αρνητικό ρυθμό μεταβολής της ροής του μαγνητικού πεδίου, στην επιφάνεια που ορίζει αυτός ο βρόχος. Αν θεωρήσουμε λοιπόν ένα πηνίο με N τυλίγματα, επιφάνεια διατομής A, μαγνητικό

πεδίο Η στο εσωτερικό του και σχετική διαπερατότητα μr, η εξίσωση που συνδέει την τάση σε αυτό το πηνίο με το πεδίο Η είναι η[13]:

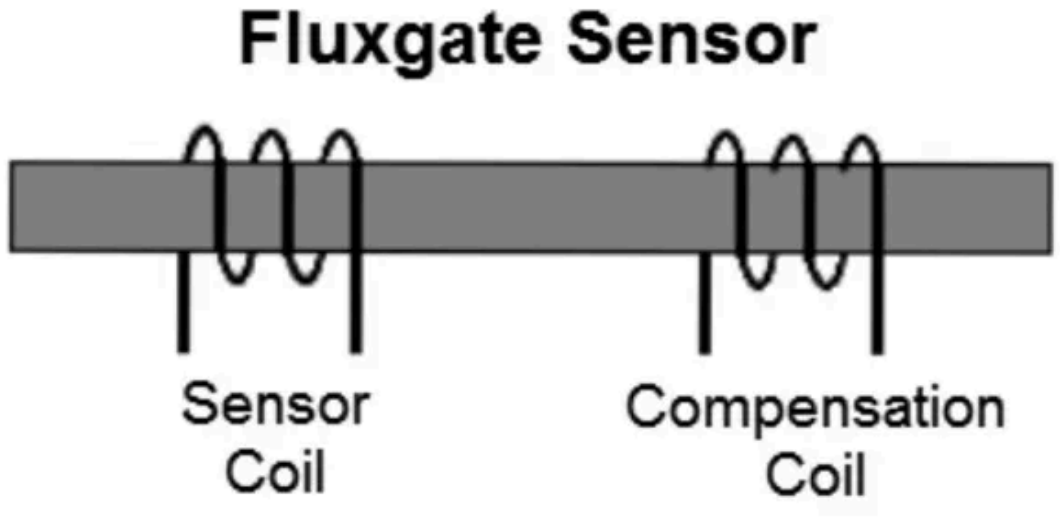
$$V = N * (A\mu 0\mu r\frac{dH}{dt} + H\mu 0\mu r\frac{dA}{dt} + A\mu 0H\frac{d\mu r}{dt})$$

Προφανώς, σχεδιάζοντας έναν αισθητήρα επαγωγής, τα ακόλουθα χαρακτηριστικά είναι επιθυμητά.

- Το πηνίο πρέπει να είναι κοντό, έτσι ώστε το πεδίο να είναι ομοιόμορφο στην περιοχή του.
- Η διατομή του πρέπει να είναι σταθερή κατα μήκος ολόκληρου του πηνίου
- Το υλικό στο εσωτερικό του πηνίου πρέπει να είναι ομοιόμορφο κατα μήκος του πηνίου.

Αυτές οι συνθήκες εξασφαλίζουν ότι η τάση στα άκρα του πηνίου θα εξαρτάται σχεδόν αποκλειστικά από τον πρώτο όρο της παραπάνω εξίσωσης. Έτσι επιτυγχάνεται μια γραμμική σχέση μεταξύ τάσης και ρυθμού επαγωγής μαγνητικού πεδίου. Η διαφορική φύση αυτής της εξίσωσης μας δίνει μια ιδέα γιατί οι μαγνητικοί αισθητήρες είναι τόσο ευαίσθητοι, και χρειάζονται προσεκτική ρύθμιση. Οι επαγωγικοί αισθητήρες μπορούν να έχουν κάποιο υλικό στο εσωτερικό του πηνίου, ή αέρα.
Η κύρια διαφορά είναι ότι οι αισθητήρες που περιέχουν κάποιο υλικό στο εσωτερικό τους, είναι πιο ευαίσθητοι(λόγω της μεγαλύτερης ροής) και συνεπώς λιγότερο σταθεροί, ενώ οι αισθητήρες με αέρα έχουν μικρότερη απόκριση αλλα περισσότερη ευστάθεια[13].

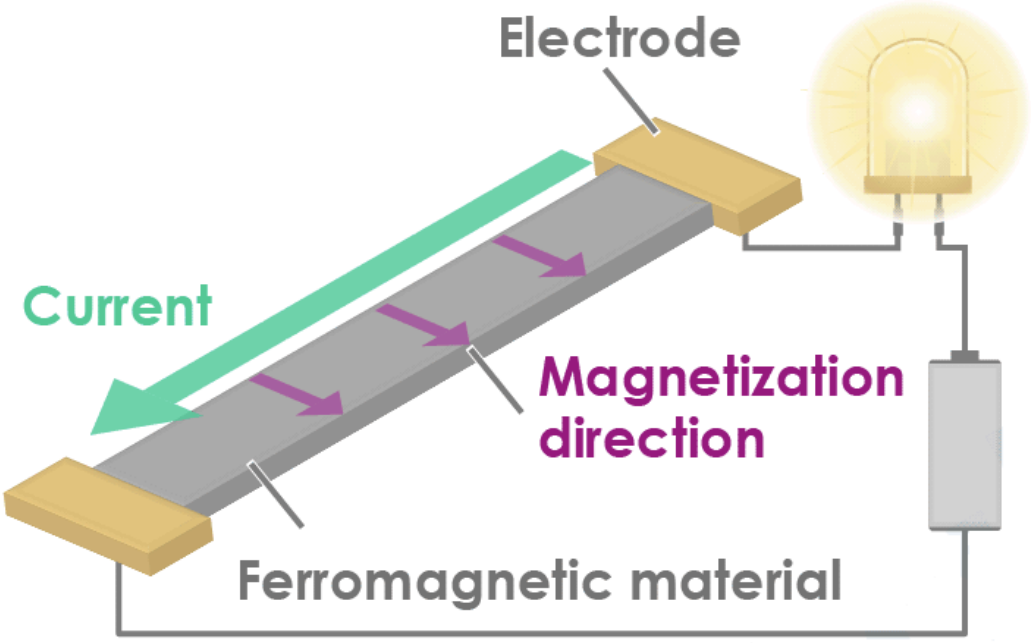
Στην συνέχεια έχουμε τους αισθητήρες τύπου fluxgate. Αυτοί χρησιμοποιούνται για την μέτρηση σταθερών ή χαμηλών συχνοτήτων μαγνητικών πεδίων[13].



Σχήμα 11. Τυπικός αισθητήρας fluxgate

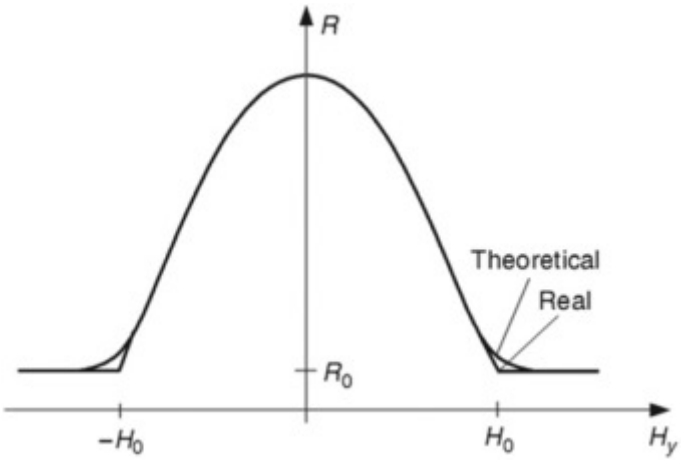
Σε έναν τυπικό αισθητήρα τύπου fluxgate, υπάρχουν 2 πηνία, το πηνίο διέγερσης, στο οποίο εφαρμόζεται μια τάση που ελέγχει την μαγνήτιση του μαλακού πυρήνα. Λόγω της μη γραμμικότητας του πυρήνα, η απόκριση που διαβάζουμε στο πηνίο του αισθητηρα είναι μη συμμετρική! Για αυτόν τον λόγο υποθέτουμε πως το εξωτερικό μαγνητικό πεδίο είναι μικρότερης συχνότητας απο την διέγερση. Αυτή η ασυμμετρία είναι που μας επιτρέπει να μετρήσουμε το εξωτερικό μαγνητικό πεδίο.
Τα πλεονεκτήματα αυτού του αισθητήρα είναι η υψηλή ευαισθησία, και η καλή ευστάθεια. Απο την άλλη, οι αισθητηρες fluxgate απαιτούν πιο πολύπλοκα συστήματα ελέγχου(οδηγός πηνίου διέγερσης) και αδυνατούν να μετρήσουν υψίσυχνες αποκρίσεις.

Το επόμενο είδος αισθητήρα που θα αναλύσουμε είναι οι αισθητήρες σιδηρομαγνητικής μαγνητοαντίστασης.



Σχήμα 12. Αισθητήρας σιδηρομαγνητικής μαγνητοαντίστασης

Η αρχή λειτουργίας τους είναι η ανισοτροπική μαγνητοαντίσταση. Πιο συγκεκριμένα, η αντίσταση ενός σιδηρομαγνητικού υλικού εξαρτάται από την γωνία που ορίζεται από την ροή του ρεύματος στο υλικό και την κατεύθυνση του εξωτερικού μαγνητικού πεδίου. Όταν αυτά τα δύο διανύσματα είναι κάθετα, η αντίσταση του υλικού ελαχιστοποιείται. Όταν αυτά είναι παράλληλα, η αντίσταση μεγιστοποιείται. Ελέγχοντας ένα μικρό ρεύμα διέγερσης στο υλικό αυτό, και μετρώντας την τάση στα άκρα του, μπορούμε να υπολογίσουμε την αντίσταση του, και να αντιστοιχίσουμε αυτή σε μια τιμή έντασης του εξωτερικού πεδίου. Μια τυπική απόκριση της αντίστασης ώς προς το εξωτερικό μαγνητικό πεδίο δίνεται στο παρακάτω σχήμα



Σχήμα 13. Απόκριση αισθητήρα AMR

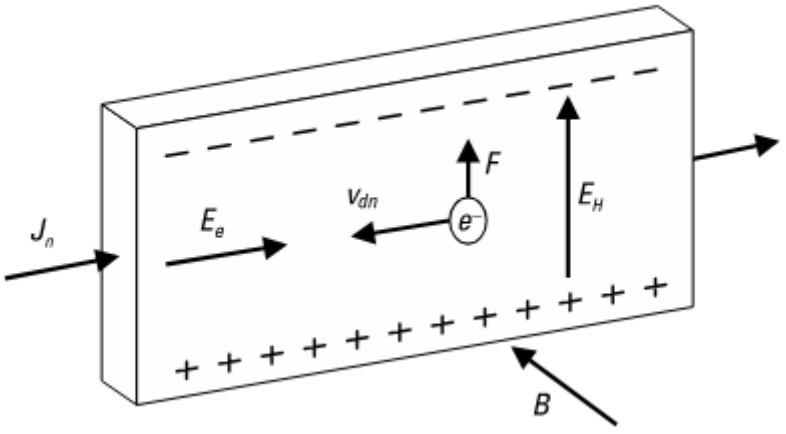
Οι αισθητήρες αυτοί έχουν τα εξής σημαντικά πλεονεκτήματα:

- Μπορούν να κατασκευαστούν σε ολοκληρωμένα κυκλώματα(μικρό μέγεθος)
- Είναι πολύ φθηνή η κατασκευή τους

Αυτά τα δύο χαρακτηριστικά τους καθιστούν ιδανικούς για μαζική κατασκευή και πώληση[13].

Μετά έχουμε τους αισθητήρες φαινομένου Hall.
Όπως υπονοεί το όνομα τους, βασίζονται στο φαινόμενο Hall. Το φαινόμενο Hall εμφανίζεται σε υλικά τα οποία διαρρέονται από ένα ρεύμα, και βρίσκονται σε ένα μαγνητικό πεδίο. Έστω ότι ο κάθε φορέας φορτίου έχει φορτίο q και ταχύτητα v, το ηλεκτρικό πεδίο που εφαρμόζεται στο υλικό είναι E, και το εξωτερικό μαγνητικό πεδίο είναι B. Απο την εξίσωση της δύναμης Lorentz, έχουμε:

$$F = q \cdot E + q(v \times B)$$



Σχήμα 14. Απεικόνιση φαινομένου Hall

Σύμφωνα με την παραπάνω εξίσωση, οι φορείς φορτίου δέχονται μια δύναμη ίση με F. Αυτή η δύναμη τους "σπρώχνει" προς την κατεύθυνση κάθετη στο ρεύμα. Οι αντίθετοι φορείς φορτίου, λοιπόν, δημιουργούν μια τάση στα άκρα του υλικού. Εισάγωντας αυτή την τάση σε ένα buffer με υψηλή(θεωρητικά άπειρη) αντίσταση εισόδου, έχουμε μια τάση εξόδου ανάλογη του εξωτερικού μαγνητικού πεδίου. Η απόκριση αυτή είναι γραμμική[14] και έχει το μεγάλο εύρος απο mT έως T, καθιστώντας τους ιδανικούς για χρήση σε ηλεκτρομαγνητικά yokes.

1.5 Περιορισμοί στην σχεδίαση μαγνητικών αισθητήρων

Όπως σε κάθε άλλο τομέα της μηχανικής, έτσι και στους μαγνητικούς αισθητήρες, η θεωρία έχει μεγάλη διαφορά από την πράξη. Οι πρακτικοί περιορισμοί που επιφέρουν οι ηλεκτρομαγνητικοί / θερμικοί θόρυβοι ή οι μη γραμμικότητες, είναι σημαντικές προκλήσεις για τον σχεδιαστή ενός τέτοιου αισθητήρα.
Τα μαγνητοσυσταλτικά υλικά για παράδειγμα εμφανίζουν θόρυβο BarkHausen[15][16], ο οποίος προκύπτει απο τις ανομοιόμορφες κινήσεις των τοιχωμάτων bloch, λόγω των ατελειών στο υλικό.

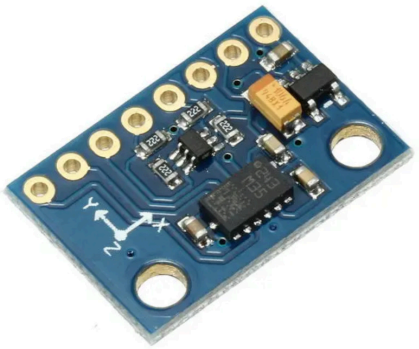
Μερικές πιθανές πηγές σφάλματος είναι[17]:

- Μεταβολές στην θερμοκρασία
- Flicker noise / shot noise στα ηλεκτρονικά ελέγχου
- Αποκλίσεις στην φυσική κατασκευή του αισθητήρα

2 Ο Αισθητήρας LSM303DLHC

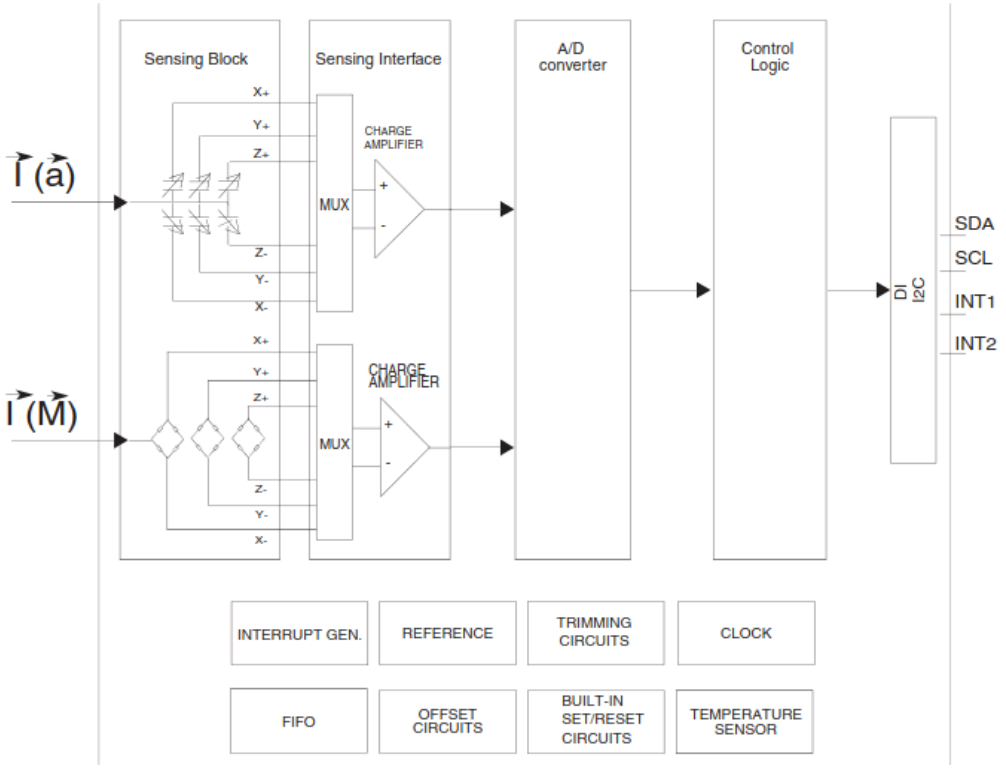
2.1 Χαρακτηριστικά του αισθητήρα

Στα πλαίσια αυτής της εργασίας θα χρησιμοποιήσουμε τον αισθητήρα LSM303DLHC, ως μια πηγή αληθινών δεδομένων για το PRISM. Επιλέξαμε αυτόν τον αισθητήρα καθώς είναι ένας από τους πιό διαδεδομένους αισθητήρες στην κατηγορία του, είναι ιδιαίτερα μικρός σε μέγεθος(14.4mm x 20.4mm), και έχει μικρό κόστος. Ο LSM303DLHC(τον οποίο θα αναφέρουμε ως “LSM” για συντομία), είναι ένα system-in-package Compass module, σχεδιασμένο και κατασκευασμένο απο την STMicroelectronics.



Σχήμα 15. LSM303DLHC PCB

Περιέχει έναν αισθητήρα μαγνητικού πεδίου, με εύρος μετρήσεων ±1.3G έως ±8.1G [18], έναν αισθητήρα γραμμικής επιτάχυνσης με εύρος ±2g έως ±16g [18], καθώς και έναν ενσωματωμένο αισθητήρα θερμοκρασίας. Ο LSM αποθηκεύει τα καταγραφόμενα μεγέθη, ανα κατεύθυνση, σε 2 registers των 8 bit. Μπορούμε να υλοποιήσουμε επικοινωνία με τον LSM(και με τα υποσυστήματα του), χρησιμοποιώντας το πρωτόκολλο επικοινωνίας I^2C, με συχνότητα 100kHz ή 400kHz. Η τροφοδοσία του μπορεί να είναι μεταξύ των 2.16V έως τα 3.6V [18], τα οποία όρια είναι ιδανικά για την χρήση με ένα τυπικό ESP32 dev-board, καθώς τα περισσότερα έχουν έναν ακροδέκτη για 3.3V. Σημαντικές για αυτή την εργασία είναι οι λειτουργίες power-down και low-power, καθώς επηρεάζουν άμεσα την κατανάλωση ενέργειας σε ένα σύστημα LSM - ESP32, συνεπώς και την αυτονομία του. Η κατανάλωση αυτή θα πρέπει να ελαχιστοποιηθεί για την μεγιστοποίηση της διάρκειας λειτουργίας. Οι αισθητήρες του LSM δημιουργούν ένα ρεύμα ανάλογο του μεγέθους που μετράνε, για κάθε άξονα μέτρησης. Το ρεύμα αυτό στην συνέχεια ενισχύεται, και τροφοδοτείται σε έναν μετατροπέα αναλογικού σήματος σε ψηφιακό. Οι μετρήσεις αυτές, σε ψηφιακή πλέον μορφή επεξεργάζονται από την μονάδα ελέγχου των registers του LSM, στους οποίους εν τέλει αποθηκεύονται. Ένα απλοποιημένο διάγραμμα υψηλού επιπέδου του συστήματος είναι το ακόλουθο [18]:



Σχήμα 16. Black box diagram του LSM303

Σε αυτή την εργασία δεν θα χρησιμοποιηθεί ο ενσωματωμένος αισθητήρας θερμοκρασίας, καθώς και τα 2 προγραμματιζόμενα interrupts του αισθητήρα γραμμικής επιτάχυνσης. Πιο συγκεκριμένα θα χρησιμοποιήσουμε τον αισθητήρα μαγνητικού πεδίου κατα την διάρκεια της λήψης αληθινών δεδομένων, και για όλα τα άλλα test θα χρησιμοποιήσουμε τον αισθητήρα επιτάχυνσης.

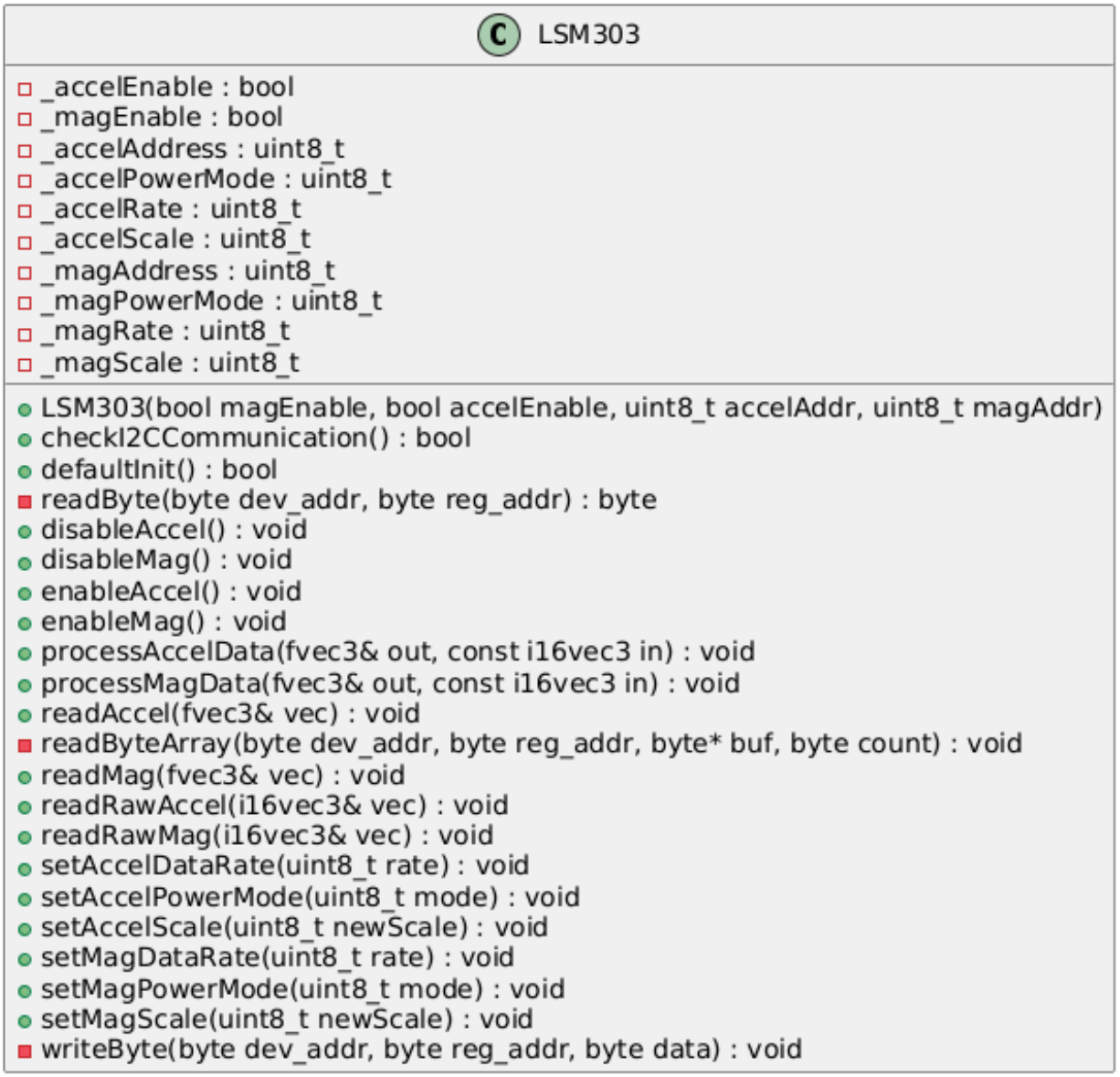
2.2 Χρήση με το ESP32

Σε αυτό το μέρος της εργασίας θα αναλύσουμε πως γίνεται να χρησιμοποιηθεί ο LSM και να οδηγηθεί από έναν μικροελεγκτή της οικογένειας ESP32. Για αυτόν τον σκοπό θα γράψουμε ένα απλό, single-header driver, το οποίο θα υποστηρίζει ακριβώς τις λειτουργίες που μας ενδιαφέρουν, το **ALSMD** (**https://github.com/PanagiotisPrountzosCS/ALSMD**).
Το ALSMD, είναι γραμμένο σε C++, και είναι σχεδιασμένο για χρήση με τους compilers του Arduino, καθώς και το Wire.h header, για την υλοποίηση της επικοινωνίας I^2C. Ο σκοπός αυτού του driver είναι να παρέχει εύκολα κατανοητές, high-level συναρτήσεις, για την χρήση του LSM, αλλα και να είναι αρκετά ελαφρύ, και να μην καταναλώνει πολλούς υπολογιστικούς πόρους. Στην επόμενη σελίδα παρατίθεται ένα παράδειγμα χρήσης του ALSMD, το αρχείο ALSMD.ino, απο το github repository. Η σχεδίαση του ALSMD, έγινε έχοντας κατα νού την χρήση που φαίνεται στο ακόλουθο παράδειγμα:

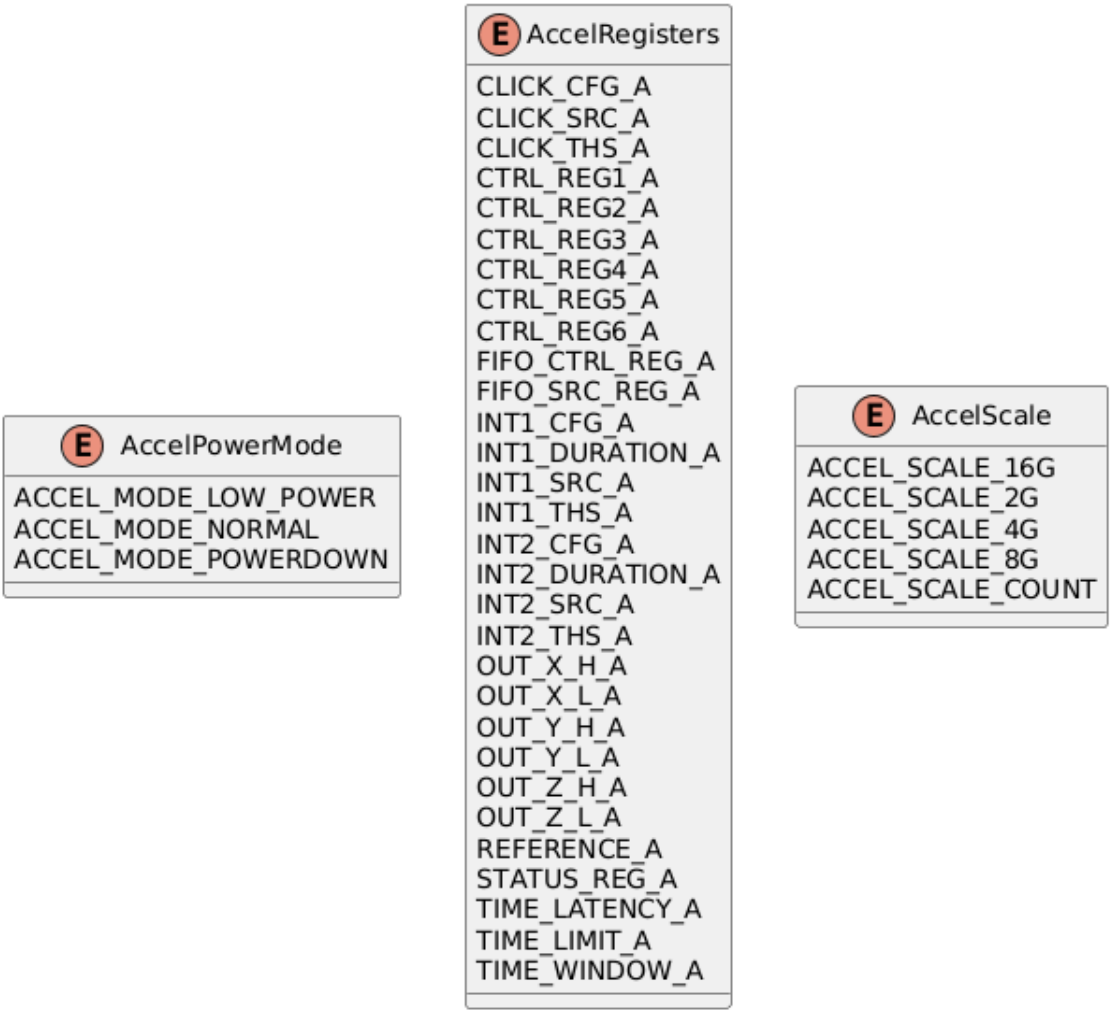
```
LSM303 sensor(1, 1); // 1,1 to enable both mag and accel
fvec3 magData;
fvec3 accelData;

void setup() {
      Serial.begin(115200);
      Wire.begin();
      if (!sensor.defaultInit()) {
          Serial.println("Failed to initialize LSM303!");
          delay(5000);
           abort();
      }
      Serial.println("LSM303 initialized successfully");
}

void loop() {
      delay(50);

      sensor.readMag(magData);
      sensor.readAccel(accelData);

      Serial.print("aX = ");
      Serial.print(accelData.x);
      Serial.print(", aY = ");
      Serial.print(accelData.y);
      Serial.print(", aZ = ");
      Serial.println(accelData.z);
      Serial.print("mX = ");
      Serial.print(magData.x);
      Serial.print(", mY = ");
      Serial.print(magData.y);
      Serial.print(", mZ = ");
      Serial.println(magData.z);
}
```

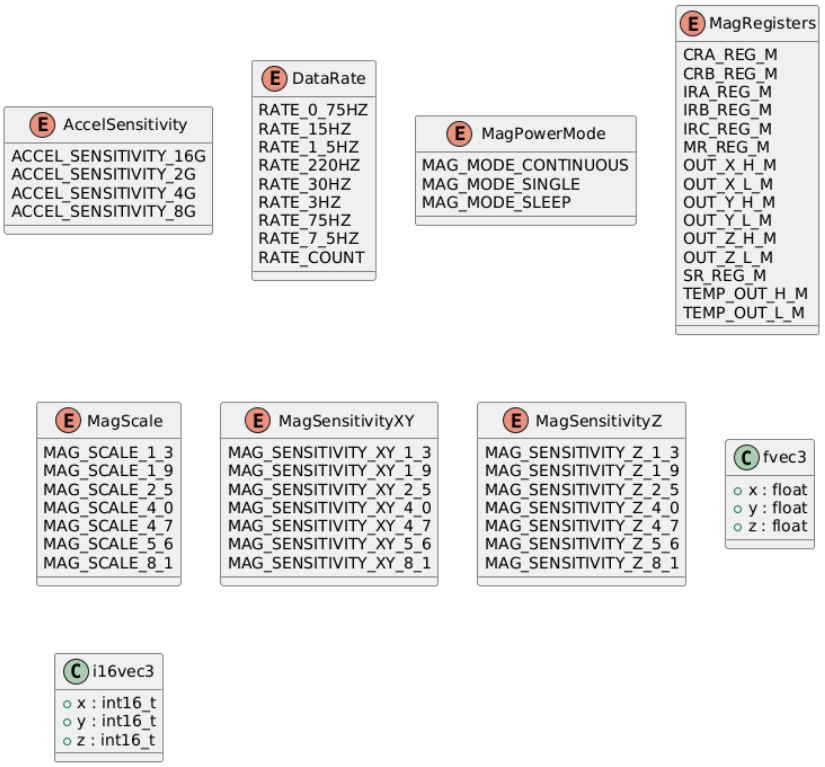
Το μόνο που έχει να κάνει ο χρήστης, όπως φαίνεται και στο παράδειγμα, είναι να ορίσει έναν LSM303, με 2 boolean ορίσματα, το πρώτο για την ενεργοποίηση του μαγνητόμετρου και το δεύτερο για την ενεργοποίηση του επιταχυνσιόμετρου, να τον αρχικοποιήσει, όπως φαίνεται στην void setup(), και τέλος να γράψει τις μετρήσεις του LSM στα fvec3 structs, τα οποία, όπως υπονοεί το όνομα τους, είναι structs με 3 floats(12 bytes total), x, y και z. Από αυτό το σημείο και έπειτα, ο χρήστης είναι ελεύθερος να χρησιμοποιήσει τα 3 αυτά floats

όπως επιθυμεί. Στην δική μας περίπτωση, θέλουμε να τα στείλουμε σε κάποιον remote υπολογιστή, μέσω WiFi. Στην συνέχεια της εργασίας, θα περιγραφεί αναλυτικά και αυτή η διαδικασία.
Όσον αφορά το ALSMD.h, το διάγραμμα UML του, δίνεται παρακάτω:



Σχήμα 17. UML diagram του ALSMD.h


AccelPowerMode
ACCEL_MODE_LOW_POWER
ACCEL_MODE_NORMAL
ACCEL_MODE_POWERDOWN
AccelRegisters
CLICK_CFG_A
CLICK_SRC_A
CLICK_THS_A
CTRL_REG1_A
CTRL_REG2_A
CTRL_REG3_A
CTRL_REG4_A
CTRL_REG5_A
CTRL_REG6_A
FIFO_CTRL_REG_A
FIFO_SRC_REG_A
INT1_CFG_A
INT1_DURATION_A
INT1_SRC_A
INT1_THS_A
INT2_CFG_A
INT2_DURATION_A
INT2_SRC_A
INT2_THS_A
OUT_X_H_A
OUT_X_L_A
OUT_Y_H_A
OUT_Y_L_A
OUT_Z_H_A
OUT_Z_L_A
REFERENCE_A
STATUS_REG_A
TIME_LATENCY_A
TIME_LIMIT_A
TIME_WINDOW_A
AccelScale
ACCEL_SCALE_16G
ACCEL_SCALE_2G
ACCEL_SCALE_4G
ACCEL_SCALE_8G
ACCEL_SCALE_COUNT



Όπως φαίνεται από τα διαγράμματα, η κλάση LSM303 περιέχει όλα τα απαραίτητα members, για την περιγραφή του LSM:
Private members:

- bool _accelEnable : Flag ενεργοποίησης επιταχυνσιόμετρου
- bool _magEnable : Flag ενεργοποίησης μαγνητόμετρου
- uint8_t _accelAddress : I2C διεύθυνση επιταχυνσιόμετρου
- uint8_t _accelPowerMode : Λειτουργία ισχύος επιταχυνσιόμετρου
- uint8_t _accelRate : Ρυθμός ανανέωσης δεδομένων επιταχυνσιόμετρου
- uint8_t _accelScale : Εύρος μετρήσεων επιταχυνσιόμετρου
- uint8_t _magAddress : I2C διεύθυνση μαγνητόμετρου
- uint8_t _magPowerMode : Λειτουργία ισχύος μαγνητόμετρου
- uint8_t _magRate : Ρυθμός ανανέωσης δεδομένων μαγνητόμετρου
- uint8_t _magScale : Εύρος μετρήσεων μαγνητόμετρου

Επιπλέον, η κλάση LSM303 περιέχει τις εξής μεθόδους που επιτρέπουν την χρήση του LSM:

- LSM303() : Constructor
- checkI2CCommunication() : Βοηθητική συνάρτηση που τεστάρει την επικοινωνία του ESP32 με τα I2C slaves του μαγνητόμετρου και του επιταχυνσιόμετρου
- defaultInit() : Default initializer, ορίζει την λειτουργία συνεχούς μετατροπής, και ορίζει τα default εύρη μετρήσεων / συχνότητες, κτλ.
- readByte(dev_addr, reg_addr) : Διαβάζει τον register στην διεύθυνση reg_addr, στην συσκευή με I2C address dev_addr
- disableAccel() : Απενεργοποιεί το επιταχυνσιόμετρο
- disableMag() : Απενεργοποιεί το μαγνητόμετρο
- enableAccel() : Ενεργοποιεί το επιταχυνσιόμετρο

- enableMag() : Ενεργοποιεί το μαγνητόμετρο
- processAccelData() : Μετατρεπει τα raw δεδομένα του επιταχυνσιόμετρου σε αληθινά μεγέθη
- processMagData() : Μετατρεπει τα raw δεδομένα του μαγνητόμετρου σε αληθινά μεγέθη
- readAccel() : Επιστρέφει την αληθινή μέτρηση του επιταχυνσιόμετρου
- readMag() : Επιστρέφει την αληθινή μέτρηση του μαγνητόμετρου
- readByteArray(dev_addr, reg_addr, buf, count) : Διαβάζει count bytes στον buf, από το reg_addr της συσκευής με I2C διεύθυνση dev_addr.
- readRawAccel() : Διαβάζει τα raw bytes του επιταχυνσιόμετρου(σε ένα i16vec3)
- readRawMag() : Διαβάζει τα raw bytes του του μαγνητόμετρου(σε ένα i16vec3)
- setAccelDataRate() : Ορίζει την συχνότητα ανανέωσης δεδομένων του επιταχυνσιόμετρου
- setAccelPowerMode() : Ορίζει την λειτουργία ισχύος του επιταχυνσιόμετρου
- setAccelScale() : Ορίζει το εύρος μετρήσεων του επιταχυνσιόμετρου
- setMagDataRate() : Ορίζει την συχνότητα ανανέωσης δεδομένων του μαγνητόμετρου
- setMagPowerMode() : Ορίζει την λειτουργία ισχύος του μαγνητόμετρου
- setMagScale() : Ορίζει το εύρος μετρήσεων του μαγνητόμετρου
- writeByte(dev_addc, reg_addr, data) : Γράφει το byte data, στον register reg_addr της συσκευής με I2C διεύθυνση dev_addr

Επιπλέον, το ALSMD.h περιέχει τα ακόλουθα enumerations με όλα τα δεδομένα του LSM, όπως αυτά δίνονται στο datasheet[18], με τις ίδιες ονομασίες:

- AccelPowerMode : Οι λειτουργίες ισχύος του επιταχυνσιόμετρου(binary format)
- AccelRegisters : Οι διευθύνσεις των registers του επιταχυνσιόμετρου
- AccelScale : Τα διάφορα εύρη του επιταχυνσιόμετρου(binary format)
- AccelSensitivity : Η ευαισθησία του κάθε εύρους μέτρησης του επιταχυνσιόμετρου
- DataRate : Ρυθμοί ανανέωσης δεδόμενων
- MagPowerMode : Οι λειτουργίες ισχύος του μαγνητόμετρου(binary format)
- MagRegisters : Οι διευθύνσεις των registers του μαγνητόμετρου
- MagScale : Τα διάφορα εύρη του μαγνητόμετρου(binary format)
- MagSensitivityXY : Η ευαισθησία του κάθε εύρους μέτρησης του μαγνητόμετρου για τους άξονες x και y
- MagSensitivityZ : Η ευαισθησία του κάθε εύρους μέτρησης του μαγνητόμετρου για τον άξονα Z

Σε αυτό το σημείο, αξίζει να αναφέρουμε πως η ευαισθησία του μαγνητόμετρου είναι διαφορετική για τους άξονες x και y από ότι είναι για τον z

Τέλος, υπάρχουν και τα εξής structs τα οποία περιέχουν τα δεδομένα των αισθητήρων:

- fvec3 : Struct με 3 floats
- i16vec3 : Struct με 3 integers των 16 bit

Σε αυτό το σημείο της εργασίας, θα δοθεί η υλοποίηση των πιο σημαντικών μεθόδων της κλάσης LSM303, και θα εξηγηθεί αναλυτικά η λειτουργία του παραδείγματος που δόθηκε προηγουμένως.
Αρχικά ο constructor της κλάσης:

```
LSM303(bool magEnable, bool accelEnable, uint8_t accelAddr = 0x19,
uint8_t magAddr = 0x1E) {
        _magEnable = magEnable;
        _accelEnable = accelEnable;
        _accelAddress = accelAddr;
        _magAddress = magAddr;
        _accelRate = RATE_15HZ;
        _magRate = RATE_15HZ;
        _accelPowerMode = ACCEL_MODE_NORMAL;
        _magPowerMode = MAG_MODE_CONTINUOUS;
        _accelScale = ACCEL_SCALE_16G;
        _magScale = MAG_SCALE_8_1;
}
```

Όπως φαίνεται στην υλοποίηση του constructor, οι 2 αισθητήρες θα ενεργοποιηθούν ανάλογα με τα boolean flags που δόθηκαν ως ορίσματα, και ο ρυθμός ανανέωσης δεδομένων τίθεται στα 15Hz. Επίσης, και οι 2 αισθητήρες τίθενται στην κανονική λειτουργία ισχύος τους. Βεβαίως, ακόμα δεν έχουμε πραγματοποιήσει επικοινωνία I2C με τους αισθητήρες, απλά έχουμε ορίσει αυτές τις παραμέτρους στην κλάση. Στην συνέχεια, θα γράψουμε τα κατάλληλα δεδομένα στους registers, για να θέσουμε τους αισθητήρες στις επιθυμητές λειτουργίες τους.

```
bool defaultInit() {
    // confirm I2C communication
    if (!checkI2CCommunication()) return false;

    if (_accelEnable) {
        setAccelPowerMode(_accelPowerMode);
        setAccelDataRate(_accelRate);
        setAccelScale(_accelScale);
    }

    if (_magEnable) {
        setMagPowerMode(_magPowerMode);
        setMagDataRate(_magRate);
        setMagScale(_magScale);
    }
    return true;
}
```

Εάν οι 2 αισθητήρες έχουν οριστεί ως ενεργοί, θέτουμε τις επιθυμητές ρυθμίσεις λειτουργίας τους.

```
void setAccelDataRate(uint8_t rate) {
    if (rate >= RATE_COUNT) return;
    _accelRate = rate;

    // Read current register value to preserve other bits
    uint8_t current = readByte(_accelAddress, CTRL_REG1_A);

    // Clear the data rate bits (bits 4-7)
    current &= 0x0F;

    // Set new data rate bits (shifted to position)
    current |= (rate << 4);

    // Add power mode bit if needed
    if (_accelPowerMode == ACCEL_MODE_LOW_POWER)
        current |= 0x08;  // Set bit 3 for low power mode

    // Write back to register
    writeByte(_accelAddress, CTRL_REG1_A, current);
}

void setAccelScale(uint8_t newScale) {
    if (newScale < ACCEL_SCALE_COUNT)
        _accelScale = newScale;
    else
        return;
    // Read current register value
    uint8_t current = readByte(_accelAddress, CTRL_REG4_A);

    // Keep other control flags
    current &= 0b11001111;

    // Set the new scale bits
    current |= (newScale << 4);

    // Rewrite control register 4
    writeByte(_accelAddress, CTRL_REG4_A, current);
}
```

```
void setAccelPowerMode(uint8_t mode) {
    if (mode > ACCEL_MODE_POWERDOWN) return;
    _accelPowerMode = mode;

    uint8_t current = readByte(_accelAddress, CTRL_REG1_A);

    switch (mode) {
        case ACCEL_MODE_NORMAL:
            // Clear low power bit, ensure axes are enabled
            current &= ~0x08;  // Clear bit 3
            current |= 0x07;   // Set bits 0-2 (XYZ axes enabled)
            break;

        case ACCEL_MODE_LOW_POWER:
            // Set low power bit, ensure axes are enabled
            current |= 0x08;  // Set bit 3
            current |= 0x07;  // Set bits 0-2 (XYZ axes enabled)
            break;

        case ACCEL_MODE_POWERDOWN:
            // Clear axes enable bits to power down
            current &= ~0x07;  // Clear bits 0-2
            break;
    }

    writeByte(_accelAddress, CTRL_REG1_A, current);
}
```

Παρόμοια είναι η λειτουργία ρύθμισης του μαγνητόμετρου :

```
void setMagDataRate(uint8_t rate) {
    if (rate >= RATE_COUNT) return;
    _magRate = rate;

    uint8_t current = readByte(_magAddress, CRA_REG_M);

    // Clear the data rate bits
    current &= 0xE3;  // 11100011 - Clear bits 2-4

    // Set new data rate bits
    current |= (rate << 2);
```

```
    // Write back to register
    writeByte(_magAddress, CRA_REG_M, current);
}

void setMagScale(uint8_t newScale) {
    if (newScale <= 7)
        _magScale = newScale;
    else
        return;

    // Write the gain setting to CRB_REG_M
    writeByte(_magAddress, CRB_REG_M, newScale << 5);
}

void setMagPowerMode(uint8_t mode) {
    if (mode > MAG_MODE_SLEEP) return;
    _magPowerMode = mode;

    uint8_t regValue;

    switch (mode) {
        case MAG_MODE_CONTINUOUS:
            regValue = 0x00;
            break;

        case MAG_MODE_SINGLE:
            regValue = 0x01;
            break;

        case MAG_MODE_SLEEP:
            regValue = 0x02;
            break;

        default:
            return;
    }

    writeByte(_magAddress, MR_REG_M, regValue);
}
```

Οι αισθητήρες αυτού του τύπου ρυθμίζονται με την εγγραφή ορισμένων τιμών σε συγκεκριμένους registers. Ας πάρουμε για παράδειγμα την ρύθμιση της ισχύος του μαγνητόμετρου. Μπορούμε να δούμε απο το datasheet, οτι ο register MR_REG_M, στην διεύθυνση 0x02, λειτουργεί σύμφωνα με τον εξής πίνακα[18]:

7.2.3 MR_REG_M (02h)

Table 76. MR_REG_M register

0(1)	0(1)	0(1)	0(1)	0(1)	0(1)	MD1	MD0

1. This bit must be set to '0' for correct operation of the device.

Table 77. MR_REG_M description

MD[1:0]	Mode select bits. These bits select the operation mode of this device (refer to *Table 78*)

Table 78. Magnetic sensor operating mode

MD1	MD0	Mode
0	0	Continuous-conversion mode
0	1	Single-conversion mode
1	0	Sleep mode. Device is placed in sleep mode
1	1	Sleep mode. Device is placed in sleep mode

Σχήμα 18. Conversion modes του LSM303

Ανάλογα λοιπόν με το byte που θα γράψουμε στην διεύθυνση αυτή, ορίζεται και η λειτουργία του μαγνητόμετρου. Αυτή ακριβώς είναι και η λειτουργία της setMagPowerMode(uint8_t mode); όπως αναφέρεται παραπάνω. Την ίδια ακριβώς λογική ακολουθούν και οι υπόλοιπες λειτουργίες του μαγνητόμετρου και του επιταχυνσιόμετρου, χρησιμοποιώντας όμως διαφορετικές διευθύνσεις. Στο default setup του μαγνητόμετρου λοιπόν, ορίσαμε οτι η λειτουργία ισχύος είναι το continuous mode. Αυτό σημαίνει πως το μαγνητόμετρο θα κάνει νέες μετρήσεις, συνεχόμενα, και με τον ρυθμό που ορίσαμε στο dataRate προηγουμένως! Αυτή η λειτουργία, είναι ιδιαίτερα χρήσιμη, αλλά, στην δική μας περίπτωση, που μας ενδιαφέρει η αυτονομία της κατασκευής, το continuous conversion θα κατανάλωνε πολλή περισσότερη ισχύ από όση μπορούμε να "ξοδέψουμε"! Στην αληθινή εφαρμογή του ALSMD, θα χρησιμοποιήσουμε το single conversion mode, το οποίο κάνει poll τον αισθητήρα μόνο όταν προσπαθούμε να διαβάσουμε μια τιμή από κάποιον output register, εξοικονομώντας ισχύ.

Τέλος, οι συναρτήσεις που χρησιμοποιούμε για να διαβάσουμε τις μετρήσεις απο τους output registers είναι οι εξής:

```
void readRawAccel(i16vec3& vec) {
    if (!_accelEnable) return;
    uint8_t buffer[6];
    // each value is 2 bytes, so we stack allocate 6(x, y, z) as a
buffer
```

```
    // read all 6
    readByteArray(_accelAddress, OUT_X_L_A, buffer, 6);

    vec.x = (int16_t)((buffer[1] << 8) | buffer[0]) >> 4;
    vec.y = (int16_t)((buffer[3] << 8) | buffer[2]) >> 4;
    vec.z = (int16_t)((buffer[5] << 8) | buffer[4]) >> 4;
    // right bit shift because accel is right adjusted by 4 bit!
}

void readRawMag(i16vec3& vec) {
    if (!_magEnable) return;
    uint8_t buffer[6];

    readByteArray(_magAddress, OUT_X_H_M, buffer, 6);

    vec.x = (int16_t)((buffer[0] << 8) | buffer[1]);
    vec.y = (int16_t)((buffer[4] << 8) | buffer[5]);
    vec.z = (int16_t)((buffer[2] << 8) | buffer[3]);
}

void readAccel(fvec3& vec) {
    i16vec3 temp{0, 0, 0};
    readRawAccel(temp);
    processAccelData(vec, temp);
}

void readMag(fvec3& vec) {
    i16vec3 temp{0, 0, 0};
    readRawMag(temp);
    processMagData(vec, temp);
}
```

Όπως φαίνεται από τις παραπάνω 4 μεθόδους, η διαδικασία λήψης δεδομένων από τους output registers δεν είναι τίποτα πιο περίπλοκο απο την ανάγνωση των registers, και την επεξεργασία των raw bytes. Πιο συγκεκριμένα, διαβάζουμε τα περιεχόμενα των 6 (contiguous) output registers, και τα αποθηκεύουμε σε έναν buffer, στο stack του ESP32. Από εκεί και πέρα, κάνουμε OR το low byte με το high byte, bit shifted κατα 8 bit, και κάνουμε cast σε int16_t. Αυτά τα δεδομένα όμως, δεν έχουν κάποια φυσική σημασία ακόμα, οπότε τα κάνουμε process, πολλαπλασιάζοντας με τις κατάλληλες σταθερές, ανάλογες του εύρους μετρήσεων, και έχουμε έτσι δημιουργήσει ένα fvec3 με τις μετρήσεις και στους 3 άξονες του επιθυμητού αισθητήρα.

2.3 Επικοινωνία μέσω MQTT

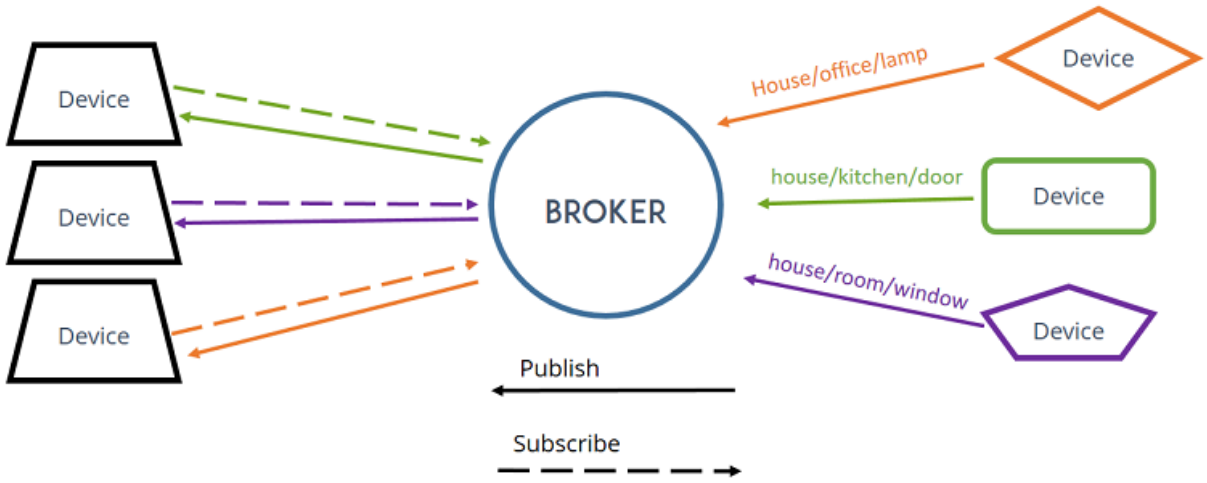
Σε αυτό το σημείο, έχουμε φτιάξει ένα απλό, ελαφρύ και συνεπώς εύχρηστο driver για τον LSM. Τώρα πρέπει να το αξιοποιήσουμε, και να στείλουμε τα δεδομένα του σε έναν υπολογιστή σε κάποιο άλλο δίκτυο. Για αυτό θα χρησιμοποιήσουμε το πρωτόκολλο επικοινωνίας MQTT, ή αλλιώς Message Queue Telemetry Transport. Το MQTT είναι ένα απο τα πιο διαδεδομένα πρωτόκολλα επικοινωνίας στον τομέα του IoT, και για καλούς λόγους. Το επιλέξαμε στα πλαίσια αυτής της εργασίας επειδή:

- Είναι ένα ελαφρύ πρωτόκολλο, με μικρό overhead στα packet sizes
- Υποστηρίζει το μοντέλο publisher - subscriber, το οποίο είναι ιδανικό για την εφαρμογή μας(Πολλαπλοί αισθητήρες / Πολλαπλοί PRISM clients)
- Επιτρέπει την επικοινωνία μεταξύ συσκευών σε διαφορετικά δίκτυα, με μικρή καθυστέρηση
- Πολλή υποστήριξη online(mosquitto, PubSubClient, κτλ.)
- Αρχιτεκτονική βασισμένη στα events, το οποίο επιτρέπει polling, αντί για συνεχή επικοινωνία

Στο μοντέλο λοιπόν του MQTT, μας ενδιαφέρουν οι ακόλουθες τρεις έννοιες:

- Broker
- Publisher
- Subscriber

Παρατίθεται ένα διάγραμμα το οποίο εξηγεί τι είναι η κάθε μία:

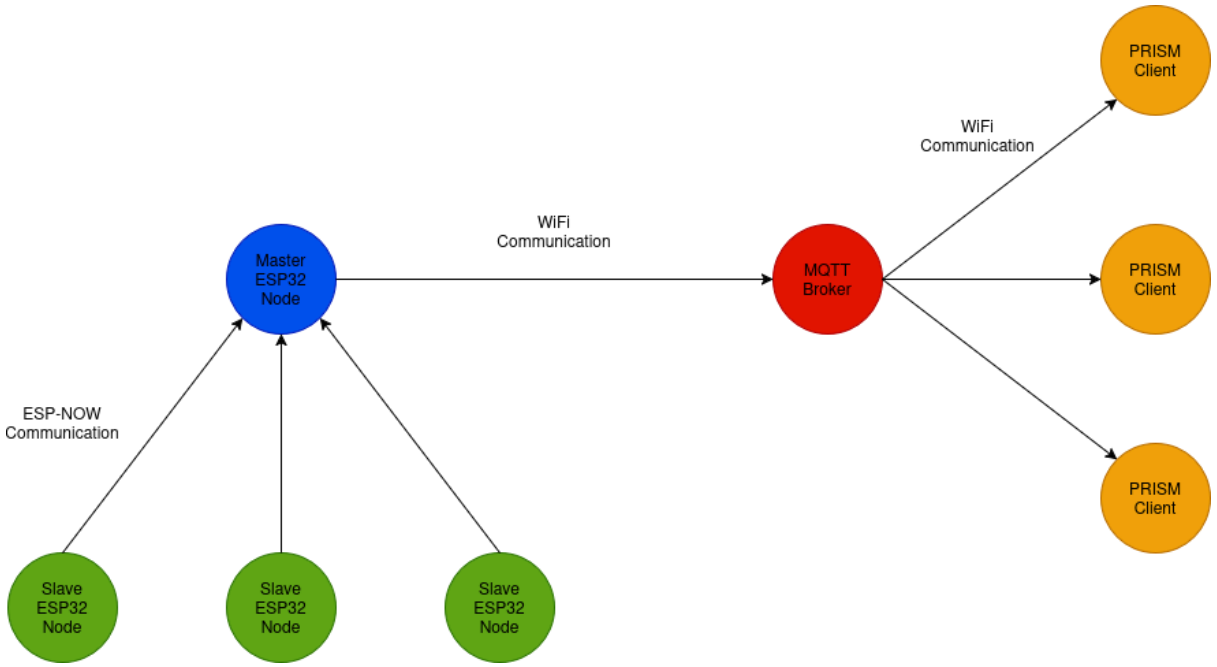


Σχήμα 19. Μοντέλο MQTT

Publishers είναι οι MQTT Clients οι οποίοι στέλνουν μηνύματα, με κάποιο συγκεκριμένο topic. Subscribers είναι οι clients οι οποίοι λαμβάνουν μηνύματα από ένα topic. Τέλος, όπως είναι προφανές, ο Broker είναι ο "μεσολαβητής" στην επικοινωνία των publishers και των subscribers, και κατευθύνει τα μηνύματα των Publishers, στους κατάλληλους Subscribers(αυτοί που ενδιαφέρονται για το ίδιο topic).

Στα πλαίσια αυτής της εργασίας, θα χρησιμοποιήσουμε το PubSubClient.h[19]. Αυτό είναι ένα header το οποίο υλοποιεί έναν MQTT Client, για τους compilers του Arduino. Υποστηρίζει μηνύματα QoS 0, μηνύματα έως 256 Byte και ένα keepalive interval ίσο με 15 δευτερόλεπτα[19]. Αυτά τα χαρακτηριστικά είναι ιδανικά για την εφαρμογή μας, καθώς τα

μηνύματα μας είναι το πολύ 12 Byte(χωρίς κάποιο packet overhead), και θα κάνουμε poll τους αισθητήρες πολύ πιο συχνά από 15 δευτερόλεπτα, οπότε δεν θα έχουμε timeouts στην επικοινωνία Broker - Publisher (τα οποία είναι σπατάλη ισχύος αλλα και δεδομένων!). Επίσης, για να υλοποιήσουμε την επικοινωνία μεταξύ διαφορετικών δικτύων, θα χρησιμοποιήσουμε την υπηρεσία HiveMQ(https://www.hivemq.com/). Αυτή η υπηρεσία λειτουργεί ώς Broker μεταξύ των subscribers και των publishers μας. Ο λόγος για τον οποίο το χρησιμοποιούμε είναι για να μην χρειαζόμαστε κάποια άλλη συσκευή να λειτουργεί ώς broker(Προφανώς ούτε κάποιο ESP θα έπρεπε να κάνει αυτή την δουλειά, επειδή θα κατανάλωνε ακόμα περισσότερη ισχύ. Θα ήταν κακή σχεδίαση να έχουμε τους αισθητήρες να κάνουν τον Broker). Όπως είναι προφανές, η χρήση ενός Broker σαν το HiveMQ θα εισάγει μια καθυστέρηση στην λήψη των δεδομένων, πολλή περισσότερη από όση θα ήταν αν τα μεταδίδαμε εντός του local network. Από τις δοκιμές όμως που κάναμε, φαίνεται ότι αυτή η καθυστέρηση είναι πολύ μικρότερη του ενός δευτερολέπτου. Αυτή η καθυστέρηση είναι ένα καλό αντάλλαγμα για την επικοινωνία μεταξύ διαφορετικών δικτύων. Για την βελτιστοποίηση του συστήματος που σχεδιάζουμε όμως, θα κάνουμε μια μικρή αλλαγή στην αρχιτεκτονική του. Πιο συγκεκριμένα, αντι να έχουμε τον ESP του κάθε αισθητήρα να επικοινωνεί με τον broker, θα επικοινωνεί με έναν κεντρικό (master) ESP, μέσω του πρωτοκόλλου ESP-NOW[20], και ο κεντρικός ESP, θα στέλνει τα δεδομένα αυτά μέσω WiFi σε οποιονδήποτε broker. Με αυτή την αρχιτεκτονική, αίρεται η ευθύνη της επικοινωνίας μέσω WiFi από τους ίδιους τους αισθητήρες, και τοποθετείται στον κεντρικό ESP. Προφανώς, η κατανάλωση ισχύος των αισθητήρων μειώνεται δραστικά. Το μειονέκτημα αυτής της επιλογής όμως είναι ότι χρειαζόμαστε περισσότερο hardware, και οτι οι αισθητήρες θα πρέπει να τοποθετούνται κοντά στον κεντρικό ESP. Ένα απλό διάγραμμα της αρχιτεκτονικής αυτής είναι το ακόλουθο:



Σχήμα 20. Διάγραμμα του συστήματος

Οι πράσινοι κόμβοι αναπαριστούν αισθητήρες

Ο μπλε κόμβος αναπαριστά τον κεντρικό esp32e

Ο κόκκινος κόμβος αναπαριστά τον MQTT Broker που θα μεταδίδει τα μηνύματα μας

Οι κίτρινοι κόμβοι αναπαριστούν clients του PRISM, οι οποίοι παίρνουν δεδομένα απο τον Broker
Σε αυτό το σημείο λοιπόν, πρέπει να υλοποιήσουμε το δεύτερο repo αυτής της εργασίας, το **DATAFLUX(https://github.com/PanagiotisPrountzosCS/DATAFLUX)**
Το DATAFLUX όπως φαίνεται και στο github repo του, έρχεται σε 2 branches, το slave_node και το master_node. Όπως υπονοούν και τα ονόματα, το ένα branch προορίζεται για τους αισθητήρες και το άλλο για τον κεντρικό ESP. Ας δούμε λοιπόν τις διαφορές αυτών και ας αναλύσουμε την λειτουργία τους, ξεκινώντας με το slave_node, το firmware των αισθητήρων.

```
/*  DATAFLUX.ino
 *
 *  Panagiotis Prountzos 2025
 *
 *  Slave node branch
 */

#include <WiFi.h>
#include <esp_now.h>

#include "DATAFLUX.h"

// firmware variables
LSM303 sensor(true, false);
uint8_t masterMAC[] = {0x30, 0xC9, 0x22, 0xB0, 0xE4, 0x6C};
uint32_t selfID = 0;

// status variables
fvec3 magReading;
message msg;
bool skipCycle = false;
RTC_DATA_ATTR uint32_t elapsedTime;

esp_now_peer_info_t masterInfo;

void dataTransmissionCallback(const uint8_t *mac_addr,
esp_now_send_status_t status) {
      deepSleep(elapsedTime);
}
```

```
void setup() {
      initSlave();

      msg.id = selfID;

      if (!initESPNOW(masterInfo, masterMAC,
dataTransmissionCallback)) skipCycle = true;

      if (!initSensor(sensor)) skipCycle = true;

      if (!sensor.checkI2CCommunication()) {
      indicateError(SENSOR_I2C_ERROR);
      skipCycle = true;
      }
      if (skipCycle) deepSleep(elapsedTime);

      sensor.readMag(magReading);
      msg.x = magReading.x;
      msg.y = magReading.y;
      msg.z = magReading.z;
      msg.timestamp = elapsedTime + millis();

      publishMessage(&msg, masterMAC, elapsedTime);
}

void loop() {
      // should NEVER reach here
      delay(1000);
}


void initSlave() {
      Wire.begin();
      WiFi.mode(WIFI_STA);
}

bool initESPNOW(esp_now_peer_info_t& peerInfo, const uint8_t*
receiver,
                  esp_now_send_cb_t onDataSent) {
      esp_wifi_set_channel(WIFI_CHANNEL, WIFI_SECOND_CHAN_NONE);
      if (esp_now_init() != ESP_OK) {
      indicateError(ESP_NOW_INIT_ERROR);
```

```
      return false;
      };

      peerInfo.channel = 0;
      peerInfo.encrypt = false;
      // Register master node(receiver)
      memcpy(peerInfo.peer_addr, receiver, 6);

      if (esp_now_add_peer(&peerInfo) != ESP_OK) {
      indicateError(ESP_NOW_ADD_PEER_ERROR);
      return false;
      };

      // Register the callback here
      if (esp_now_register_send_cb(onDataSent) != ESP_OK) {
      indicateError(ESP_NOW_REGISTER_CALLBACK_ERROR);
      return false;
      };

      return true;
}

bool initSensor(LSM303& sensor) {
      sensor.configure(ACCEL_RATE_0HZ, MAG_RATE_3HZ,
ACCEL_MODE_POWERDOWN, MAG_MODE_SINGLE,
                        ACCEL_SCALE_2G, MAG_SCALE_8_1);
      if (!sensor.init()) {
      indicateError(SENSOR_INIT_ERROR);
      return false;
      };
      return true;
}


void deepSleep(uint32_t& elapsedTime) {
      uint32_t offset = millis();
      uint32_t sleepTime = CYCLE_DURATION_MS;

      if (offset < CYCLE_DURATION_MS) {
      sleepTime -= offset;
      }
```

```
        elapsedTime += offset + sleepTime;

        esp_sleep_enable_timer_wakeup(sleepTime * 1000);

        esp_deep_sleep_start();
}

void publishMessage(const message* msg, const uint8_t* receiver,
uint32_t& elapsedTime) {
        esp_err_t result = esp_now_send(receiver, (uint8_t*)msg,
sizeof(message));
        if (result != ESP_OK) {
        indicateError(ESP_NOW_TRANSMIT_ERROR);
        deepSleep(elapsedTime);
        }
}
```

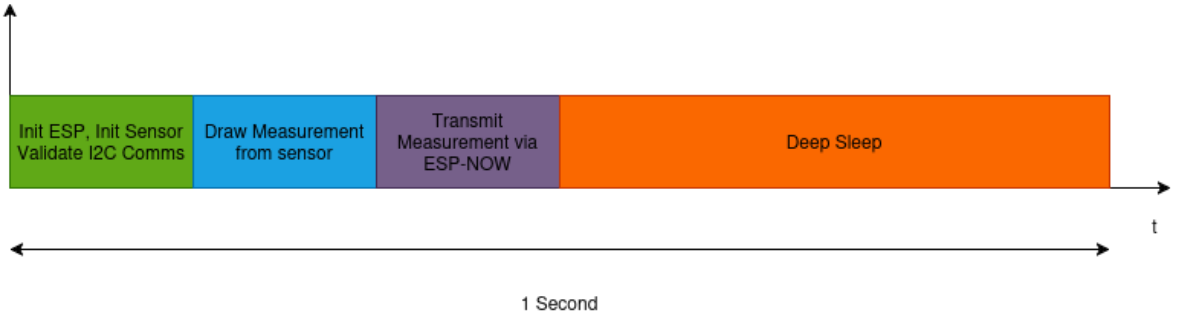
Ο κάθε κύκλος λειτουργίας ενός αισθητήρα αποτελείται από τις εξής λειτουργίες:

1) Αρχικοποίηση του esp(wifi, esp-now)
2) Αρχικοποίηση του αισθητήρα
3) Ένας γρήγορος έλεγχος για την επικοινωνία I2C μεταξύ αισθητήρα και esp32
4) Λήψη μέτρησης του αισθητήρα
5) Μετάδοση του μηνύματος
6) Εκκίνηση deep-sleep

Μια ερώτηση που δημιουργεί ο κύκλος λειτουργίας είναι η εξής: Γιατί να πρέπει να αρχικοποιούμε όλα τα υποσυστήματα σε κάθε κύκλο? Ο λόγος για αυτό είναι επειδή ο κάθε κύκλος ολοκληρώνεται με ένα deep-sleep. Το deep-sleep είναι μια λειτουργία “ύπνου”[21] του esp32, η οποία απενεργοποιεί κάθε σύστημα στο esp32, εκτός από τα ακόλουθα[21]:

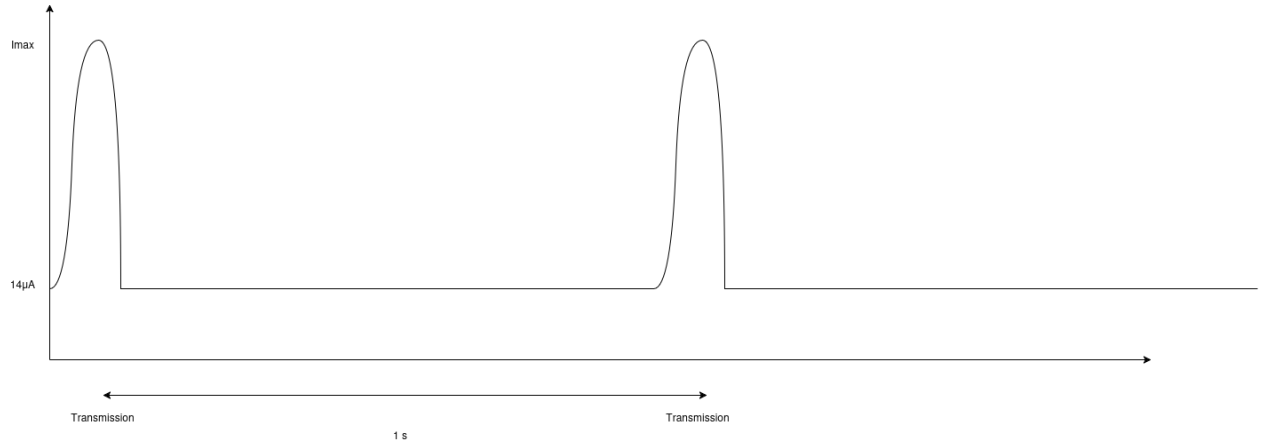
- RTC Controller
- ULP Coprocessor
- RTC Fast Memory
- RTC Slow Memory

Η μόνες λειτουργίες διαθέσιμες κατα το deep-sleep που μας ενδιαφέρουν, είναι ένας timer, ο οποίος χρησιμοποιείται για την έξοδο από το deep-sleep(στο timer overflow), και την μνήμη RTC[22], η οποία χρησιμοποιείται για την αποθήκευση δεδομένων μεταξύ των κύκλων deep-sleep. Εφόσον όλα τα υπόλοιπα συστήματα απενεργοποιούνται, είναι προφανές οτι στο “ξύπνημα” ο επεξεργαστής κάνει reboot, και ο κώδικας μας ξεκινάει από την αρχή. Για αυτό λοιπόν, πρέπει να αρχικοποιήσουμε όλα τα υποσυστήματα του κώδικά μας, όπως τον αισθητήρα. Όπως φαίνεται από τον κώδικα, η dataTransmissionCallback είναι η συνάρτηση που πραγματοποιεί το deep-sleep. Απο το όνομα της είναι κατανοητό πως αυτή καλείται ασύγχρονα με την ολοκλήρωση της αποστολής ενός μηνύματος μέσω ESP-NOW, μια λειτουργία η οποία είναι non-blocking. Παρατίθεται ένα διάγραμμα του κάθε κύκλου λειτουργίας του slave_node



Σχήμα 21. Διάγραμμα χρονισμού slave node

Οι διάρκειες της κάθε λειτουργίας του κύκλου σε αυτό το διάγραμμα δεν είναι αντιπροσωπευτικές των αληθινών διαρκειών. Σε έναν τυπικό κύκλο, οι αρχικοποιήσεις και η μετάδοση διαρκούν ~20ms, και το deep-sleep διαρκεί ~980ms. Οι esp32c6 που θα χρησιμοποιήσουμε καταναλώνουν 14μA[23]. Η κατανάλωση αυτή είναι αρκετά μικρή, και διαρκεί για το ~98% του κάθε κύκλου σύμφωνα με δοκιμές που κάναμε. Ο τελικός μας αισθητήρας λοιπόν θα έχει πολύ μεγάλη αυτονομία, ανάλογα με την μπαταρία που τον τροφοδοτεί. Στα πλαίσια αυτής της εργασίας, θα χρησιμοποιήσουμε μια μπαταρία των 3.5mAh. Ένα θεωρητικό ανώτατο όριο υπολογίζεται ώς εξής : 3500 / 14 = 250 hours ~ 10.5 μέρες! Βεβαίως αυτός ο υπολογισμός υποθέτει ότι στο 100% του κύκλου πραγματοποιείται deepsleep, το οποίο δεν είναι αληθές. Μια πιο ρεαλιστική οπτικοποίηση της κατανάλωσης ρεύματος είναι η εξής:



Σχήμα 22. Κατανάλωση ρεύματος slave node

Αυτή την στιγμή λοιπόν, θα αναλύσουμε το firmware του master node, του κεντρικού esp32. Στο δεύτερο branch του DATAFLUX, το master_node. Η λειτουργία αυτού είναι η εξής: Κάθε φορά που το ESP32 λαμβάνει ένα μήνυμα μέσω esp-now, θα αποθηκεύει την μέτρηση που περιέχει αυτό το μήνυμα σε ένα queue. Όταν αυτό το queue γεμίσει με περισσότερα από μία σταθερή ποσότητα μηνύματα, ή αν έχει περάσει πολλή ώρα από την τελευταία μετάδοση, θα πραγματοποιείται μια καινούργια. Αυτή η συμπεριφορά υλοποιείται με τον ακόλουθο κώδικα:

```
void onReceive(const esp_now_recv_info_t* info, const uint8_t*
data, int len) {
      if (len != sizeof(message)) {
      return;
      }

      message msg;
      memcpy(&msg, data, len);

      xQueueSend(dataQueue, &msg, portMAX_DELAY);
}

void setup() {
      // Serial.begin(115200);
      WiFi.mode(WIFI_AP_STA);

      initEspNow();
      // int ch = WiFi.channel();
      // Serial.println("channel " + String(ch));
      if (esp_now_register_recv_cb(onReceive) != ESP_OK)
      indicateError(ESP_NOW_REGISTER_CALLBACK_ERROR, true);

      dataQueue = xQueueCreate(MESSAGE_QUEUE_MAX_SIZE,
sizeof(message));

      if (dataQueue == NULL) indicateError(X_QUEUE_INIT_ERROR,
true);
      lastQueueClear = millis();

      setup_wifi(ssid, password);

      setup_mqtt(&secureClient, &client, root_ca, mqtt_server,
mqtt_port);
}

void loop() {
      if (!client.connected()) {
      // Serial.println("Disconnected from mqtt");
      reconnect_mqtt(&client, mqtt_username, mqtt_password);
      }
      client.loop();
      pollQueue(&client, mqtt_topic, dataQueue, lastQueueClear);
```

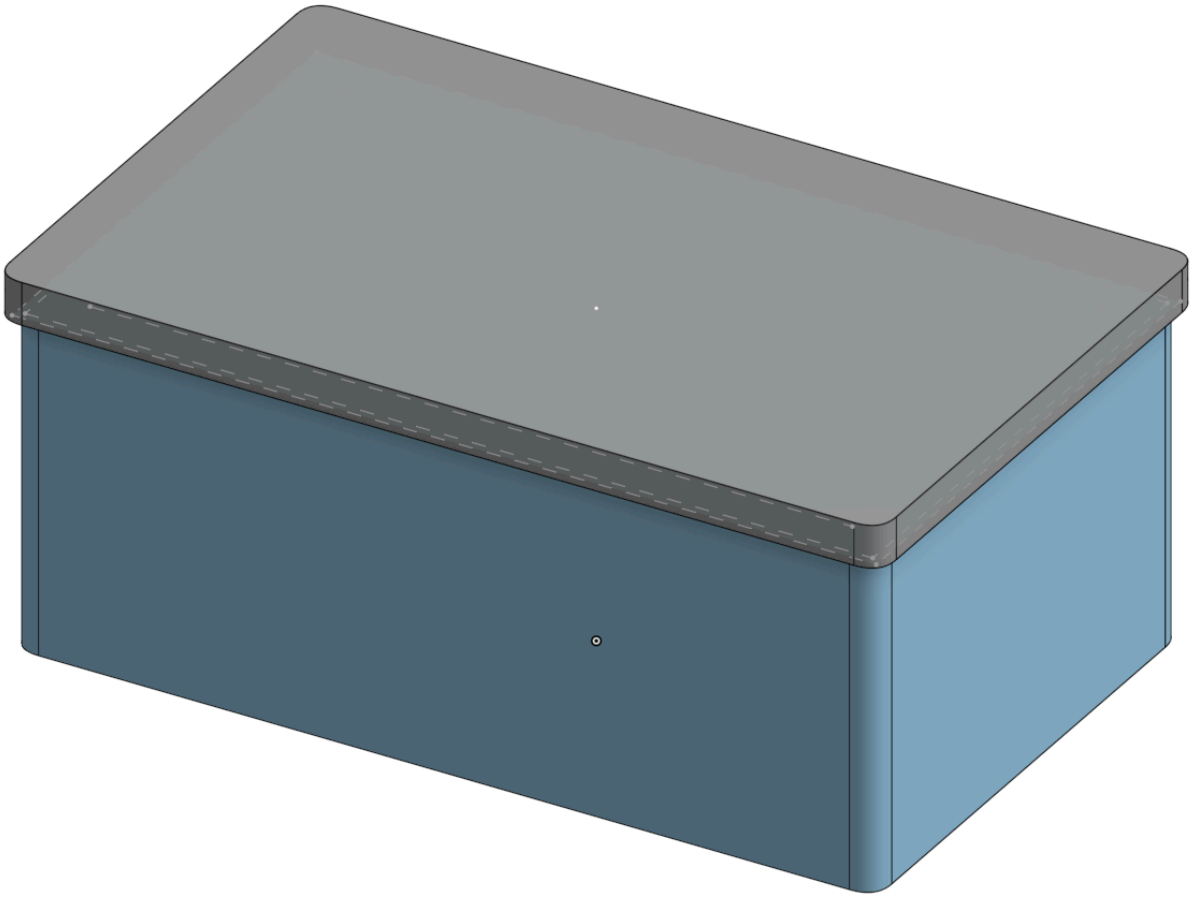
```
      delay(TICK_DURATION);
}

void pollQueue(PubSubClient* client, const char* mqtt_topic,
QueueHandle_t dataQueue,
                uint32_t& lastClear) {
      if (millis() - lastClear > MESSAGE_QUEUE_FLUSH_INTERVAL ||
      uxQueueMessagesWaiting(dataQueue) >=
MESSAGE_QUEUE_FLUSH_SIZE) {
      // empty queue in here
      uint32_t counter = 0;
      message msgArray[msgBufferSize];
      while (xQueueReceive(dataQueue, msgArray + (counter++), 0) ==
pdTRUE) {
            if (counter == msgBufferSize) {
                  client->publish(mqtt_topic, (uint8_t*)msgArray,
counter * sizeof(message));
                  // Serial.println("Transmitted data : ");
                  // Serial.println(counter * sizeof(message));
                  counter = 0;
            }
      }

      if (counter) {
            client->publish(mqtt_topic, (uint8_t*)msgArray, counter
* sizeof(message));
            // Serial.println("Transmitted data : ");
            // Serial.println(counter * sizeof(message));
      }
      lastClear = millis();
      // Serial.println("==========");
      }
}
```

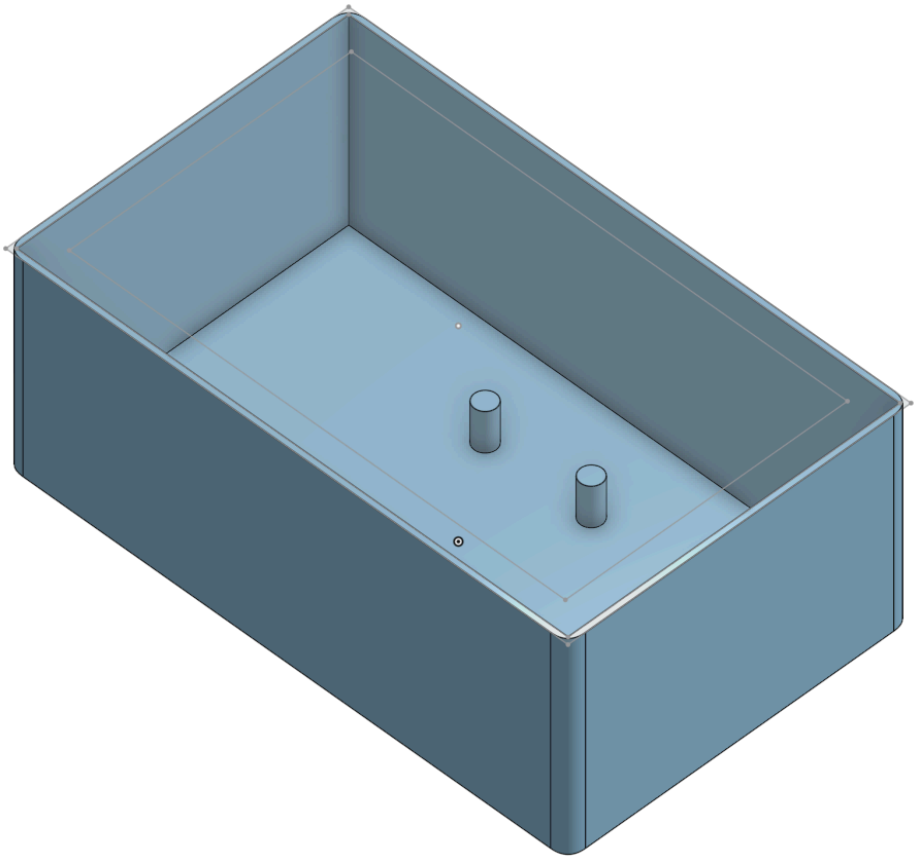
Η λειτουργία του master_node είναι αρκετά απλή. Στην αρχή δημιουργούμε ένα xQueue, απο το FreeRTOS[24]. Αυτό το queue θα λειτουργεί σαν ένα απλό message queue, στο οποίο θα εισάγονται μηνύματα απο τα slave_nodes, και από το οποίο θα αφαιρούνται τα μηνύματα την κατάλληλη στιγμή. Μετά την αρχικοποίηση του wifi και του esp-now, ξεκινάει το άπειρο loop της εφαρμογής. Σε αυτό το loop, ξεκινάμε ελέγχοντας αν έχει σταματήσει η επικοινωνία του esp32 με τον mqtt broker. Σε περίπτωση που αυτή έχει αποτύχει, επιχειρούμε να την ξανα-αρχισουμε. Έπειτα ελέγχουμε στην pollQueue() αν έχει περάσει ένα σταθερό χρονικό διάστημα από το τελευταίο mqtt transmission, ή αν το xQueue έχει περισσότερα από ένα

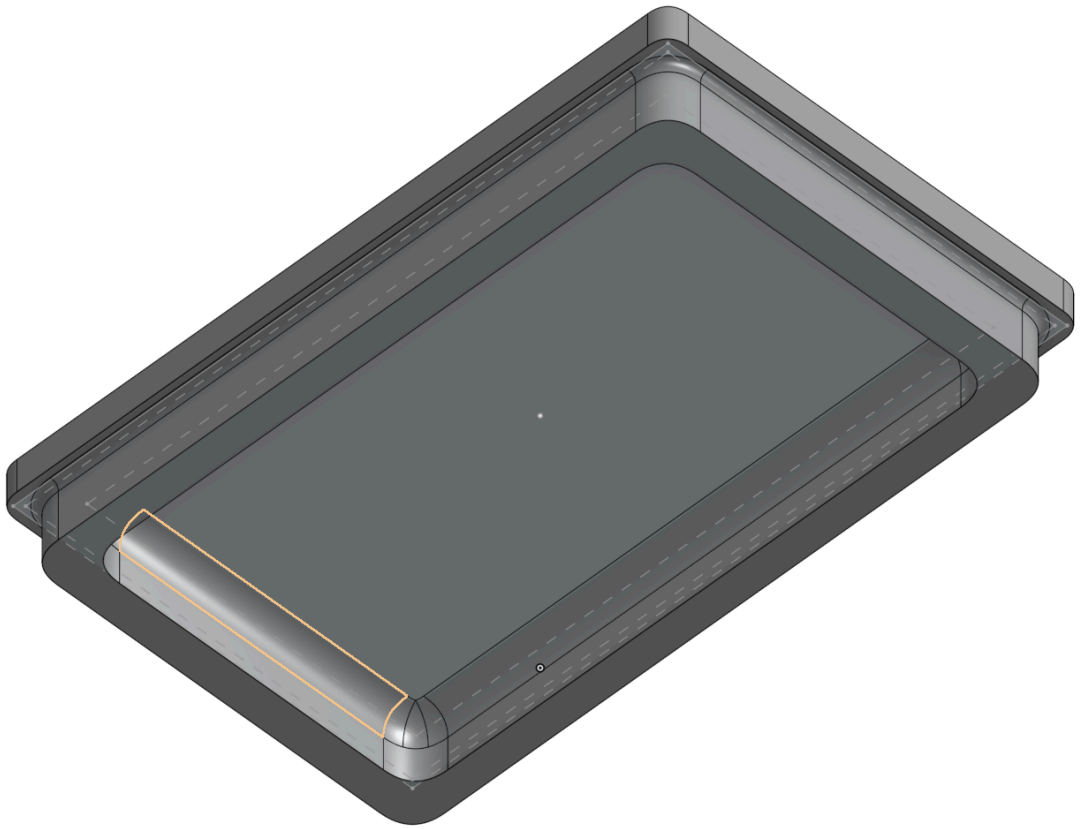
σταθερό πλήθος μετρήσεων. Αν κάποια από τις δύο συνθήκες είναι αληθής, αδειάζουμε το message queue και μεταδίδουμε τις μετρήσεις.
Για να χρησιμοποιήσουμε τους αισθητήρες και τον κεντρικό esp για αυτή την εφαρμογή λοιπόν, τοποθετούμε τους αισθητήρες σε διάφορες θέσεις στις οποίες θέλουμε να μετρήσουμε το μαγνητικό πεδίο, τοποθετούμε τον κεντρικό esp(ο οποίος τροφοδοτείται απο 5V USB-C) κοντά στους αισθητήρες. Ο κεντρικός esp συνδέεται στο τοπικό WiFi και μεταδίδει τα δεδομένα των αισθητήρων στον MQTT Broker.

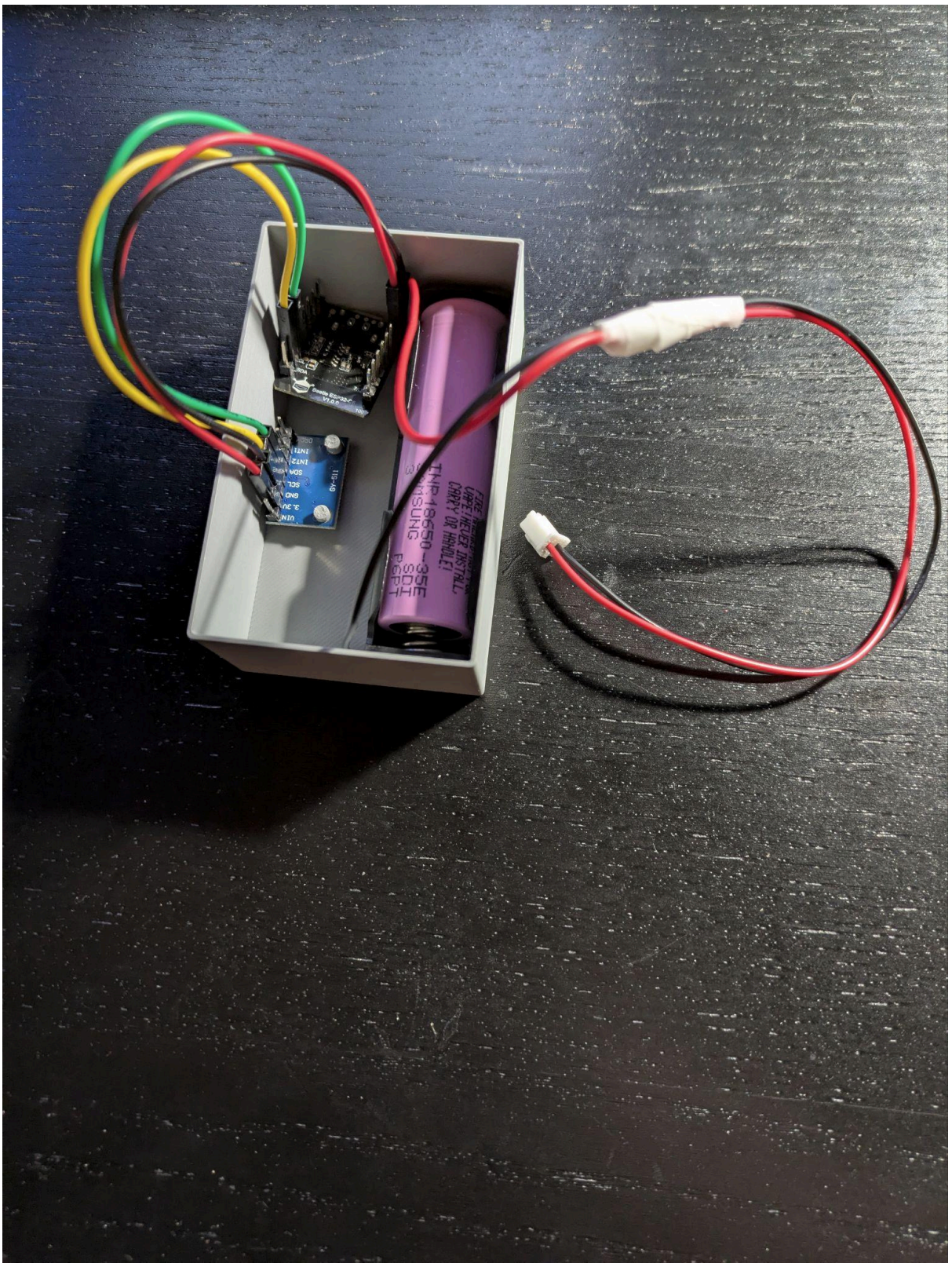
2.4 Συσκευασία του αισθητήρα

Για την ευκολία της εφαρμογής και τοποθέτησης των αισθητήρων, θα σχεδιάσουμε και θα εκτυπώσουμε ένα απλό κουτί το οποίο θα περιέχει την μπαταρία, τον αισθητήρα και τον esp32c6. Το σχέδιο στο οποίο καταλήξαμε μετά από μερικές δοκιμές είναι το ακόλουθο:

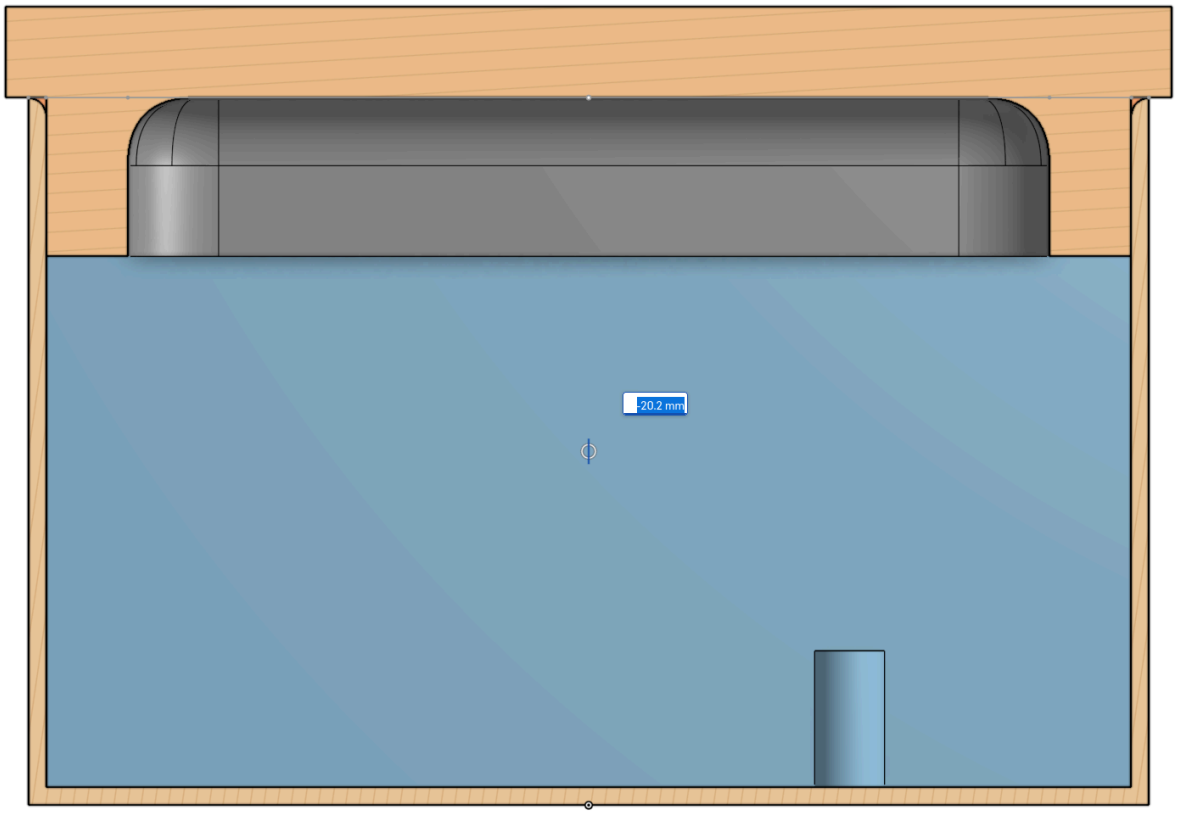







FIRE HAZARD!
CAREFULLY INSTALL,
CARRY OR HANDLE!
INR18650-35E
SAMSUNG
SDI
PBPT

Όπως φαίνεται από την τελευταία φωτογραφία, οι δύο κύλινδροι στο κέντρο του κουτιού βρίσκονται για την σταθεροποίηση του ίδιου του αισθητήρα, ενώ στο υπόλοιπο κουτί υπάρχει αρκετός χώρος για την μπαταρία, και για τον ESP32



Όπως φαίνεται παραπάνω, το κενό μεταξύ των τοιχωμάτων του κουτιού και των τοιχωμάτων στο καπάκι είναι σχεδόν μηδενικό. Δεδομένου ότι τα τοιχώματα του κουτιού είναι πολύ λεπτά(0.8mm), αυτό επιτρέπει να κάθεται το καπάκι πολύ σφιχτά πάνω στο κουτί, εξασφαλίζοντας την σταθερότητα των εξαρτημάτων.

3 Το Σύστημα PRISM

3.1 Στόχοι και απαιτήσεις του PRISM

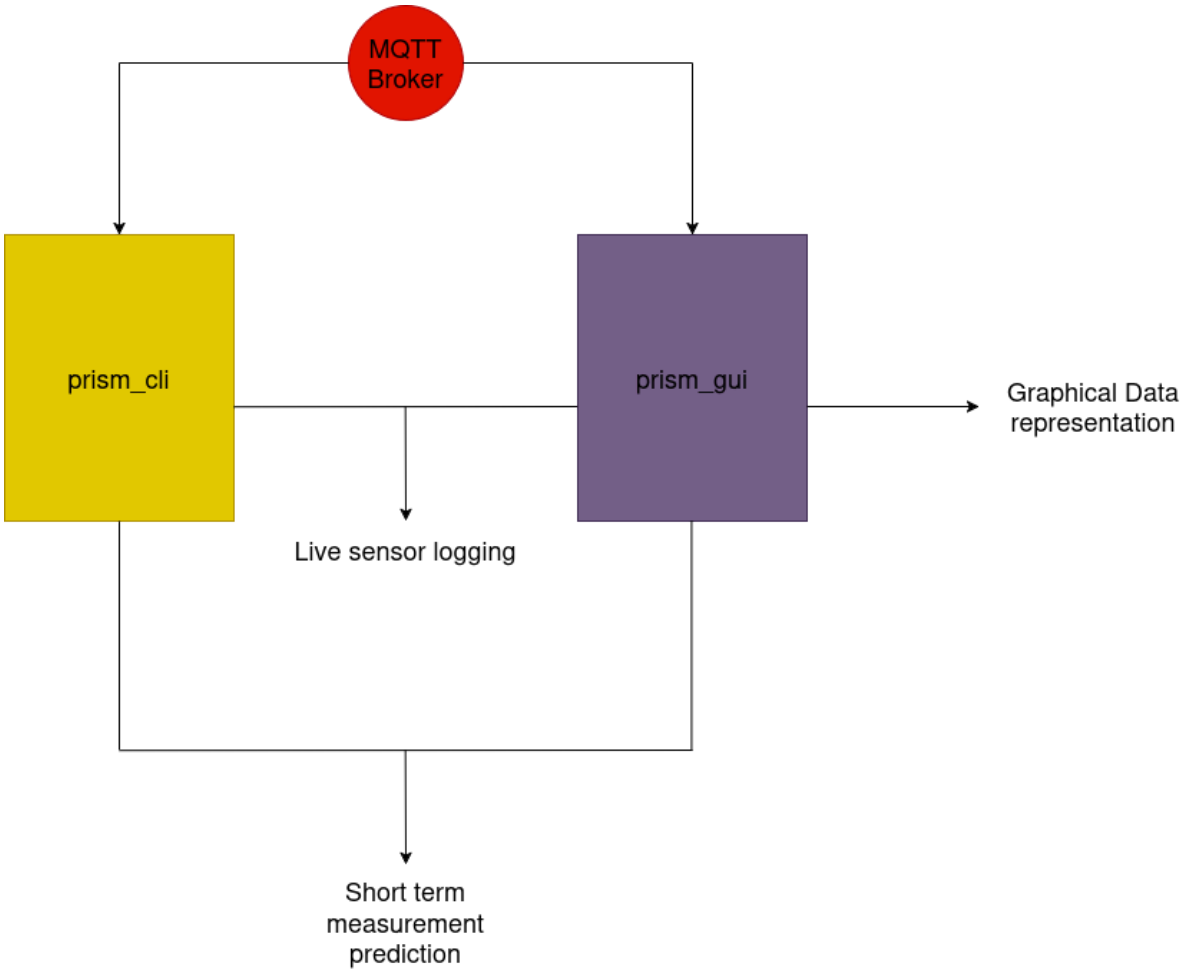
Σε αυτό το σημείο της εργασίας, έχουμε υλοποιήσει το πρώτο μισό του συστήματος παρακολούθησης αισθητήρων :

- Drivers για τους αισθητήρες LSM
- Επικοινωνία μεταξύ των αισθητήρων και του gateway
- Επικοινωνία του gateway με τον MQTT broker

Το μόνο που έχει μείνει λοιπόν για την ολοκλήρωση του συστήματος είναι η υλοποίηση ενός προγράμματος το οποίο θα λαμβάνει τα μηνύματα του gateway μέσω του MQTT broker, και θα παρακολουθεί τους αισθητήρες. Αυτός είναι ο στόχος του PRISM(Predictive realtime industrial sensor monitor). Επομένως, όπως και με κάθε άλλη εφαρμογή, η πρώτη μας ευθύνη είναι να ορίσουμε τους στόχους αυτού του προγράμματος, και μερικές πιο λεπτομερείς απαιτήσεις:

- Το πρόγραμμα αυτό θα πρέπει να μπορεί να συνδέεται σε κάποιο MQTT host, στο τοπικό ή σε κάποιο απομακρυσμένο δίκτυο, χρησιμοποιώντας ένα username, password και ίσως ένα TLS root certificate.
- Θα μπορεί να κάνει συνδρομή σε κάποιο MQTT Topic, και να αποθηκεύει όσα μηνύματα δημοσιεύονται εκεί και έχουν την μορφή των μηνυμάτων που μεταδίδει το ESP32 Gateway.
- Θα υλοποιεί ένα άπειρο loop στο οποίο θα παρακολουθεί για καινούργια μηνύματα(δηλαδή μηνύματα που δεν έχει κάνει log ακόμα), θα τα αποθηκεύει, και θα ενημερώνει τον χρήστη για αυτά(logging μέσω stdout / gui).
- Θα πρέπει να γραφτεί σε C / C++, με το λιγότερο δυνατό αριθμό απαιτούμενων βιβλιοθηκών(ανεξαρτήτων λειτουργικού συστήματος), για να είναι εύκολα φορητό σε διάφορα συστήματα, και για τις βέλτιστες επιδόσεις.
- Θα έχει 2 μορφές, CLI(command line interface) και GUI(graphical user interface) για την ζωντανή παρακολούθηση δεδομένων. Για την πιο αποτελεσματική σχεδίαση και την μέγιστη επαναχρησιμοποίηση κώδικα, το GUI app θα είναι ένα “περιτύλιγμα”(wrapper) γύρω από το CLI app, και απλά θα προσθέτει τα γραφικά.
- Θα πρέπει να ζητάει από τον χρήστη μερικά ορίσματα για την χρήση του, και να τον ενημερώνει για την ορθότητα τους.
- Θα πρέπει να μπορεί να κάνει προβλέψεις(forecasts) για μελλοντικές μετρήσεις απο τους αισθητήρες που παρακολουθεί.
- Θα μπορεί να προσθέτει καινούργιους αισθητήρες που επικοινωνούν μέσω του gateway δυναμικά στο loop παρακολούθησης, στο runtime.
- Θα μπορεί να διαγράφει τα δεδομένα από την μνήμη του συστήματος όταν αυτά ξεπερνούν ένα σταθερό μέγεθος, και να τα καταγράφει στον αποθηκευτικό χώρο(σε κάποιο log).
- Θα πρέπει να κλείνει όλες τις συνδέσεις στο τέλος του session.

Έχοντας λοιπόν αυτές τις απαιτήσεις, μπορούμε να ξεκινήσουμε να σχεδιάζουμε μια αρχιτεκτονική για το PRISM, την οποία έπειτα θα υλοποιήσουμε. Ένα απλοποιημένο black box diagram του PRISM είναι το ακόλουθο:



Από αυτό το διάγραμμα βλέπουμε πως οι δύο εφαρμογές έχουν μόνο μια διαφορά, η οποία είναι η ύπαρξη ή μη της γραφικής αναπαράστασης των δεδομένων. Επιπλέον, η επικοινωνία του PRISM και του MQTT Broker είναι μονόδρομη! Τα PRISM clients απλά κάνουν μια συνδρομή σε ένα MQTT Topic, και λαμβάνουν δεδομένα από αυτό. Αυτή η επιλογή έγινε αποκλειστικά για την διευκόλυνση της υλοποίησης του PRISM. Όπως θα εξηγήσουμε όμως και στο 4ο κεφάλαιο, η αμφίδρομη επικοινωνία, δηλαδή η δυνατότητα ελέγχου των αισθητήρων και του Gateway θα ήταν μια πολύ χρήσιμη επέκταση αυτής της εφαρμογής, ακριβώς σαν την αμφίδρομη επικοινωνία αισθητήρων - Gateway! Η πολυπλοκότητα όμως που εισάγει αυτή η λειτουργία, παρά της χρησιμότητας της, είναι υπερβολή για τους στόχους αυτής της εργασίας.

3.2 Σχεδίαση και υλοποίηση του PRISM

Πλέον, έχοντας ξεκάθαρους στόχους για την εφαρμογή, μπορούμε να ξεκινήσουμε με την σχεδίαση του PRISM. Αρχικά, θα πρέπει να αποφασίσουμε τα εξωτερικά dependencies της εφαρμογής μας. Θέλουμε τις ακόλουθες non-standard λειτουργίες:

- Επεξεργασία ορισμάτων εφαρμογής(Command line options parsing)
- Σύστημα για το Hardware Abstraction Layer(Διαχείριση της γραφικής εφαρμογής)

- Βιβλιοθήκη υλοποίησης GUI
- Βιβλιοθήκη απεικόνισης σημάτων
- Βιβλιοθήκη για επικοινωνία MQTT

Για αυτές τις λειτουργίες επιλέξαμε τα ακόλουθα open source libraries:

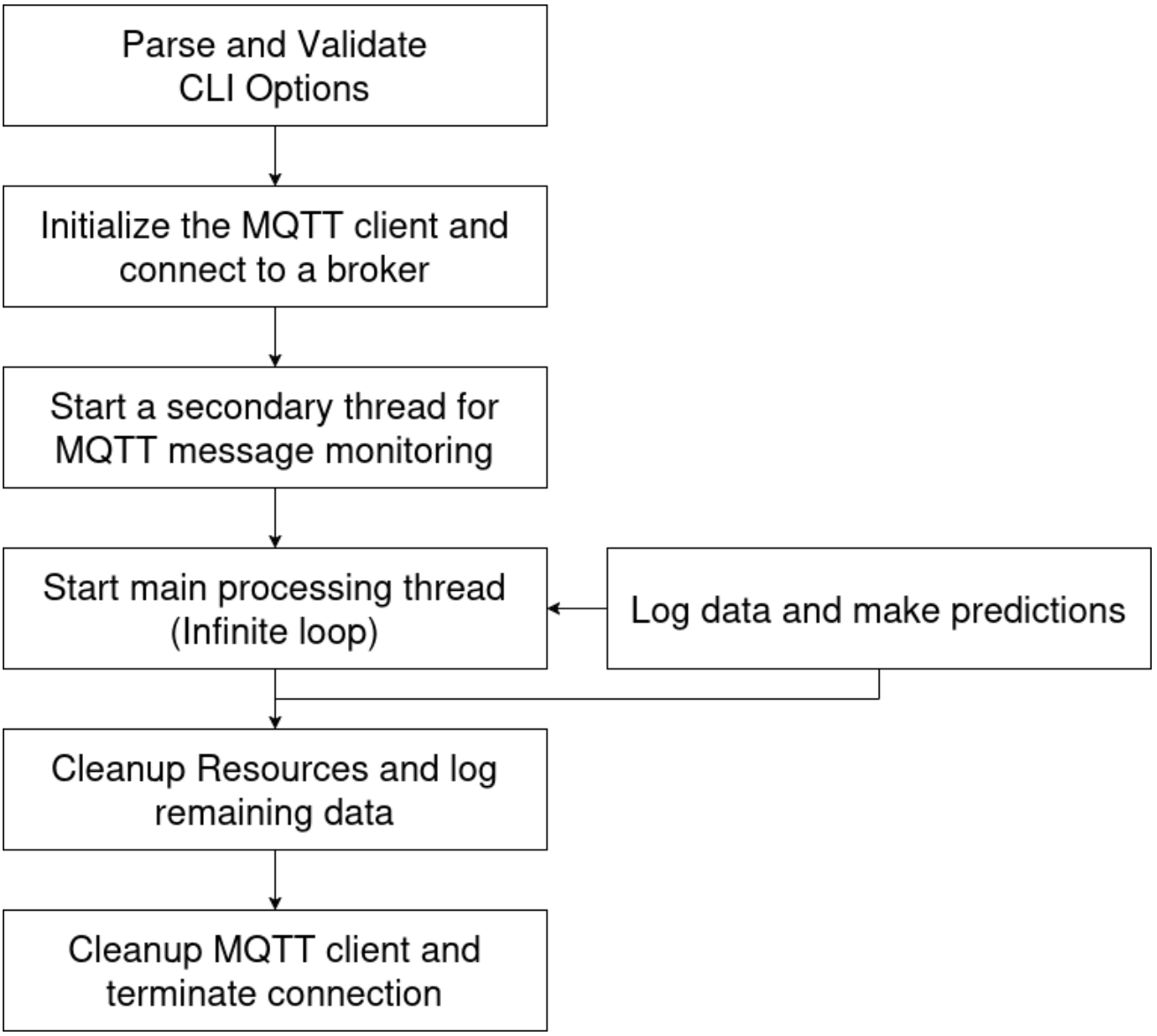
- SDL2 : Cross platform C Library που επιτρέπει τον έλεγχο του hardware του υπολογιστή(IO) και των γραφικών, μέσω διαφόρων backends(opengl / D3D)[25]
- cxxopts : Ελαφρύ options parser για εφαρμογές C++, στο στύλ του UNIX[26]
- ImGui : Ελαφρύ open source GUI library, βασισμένο σε opengl / vulkan και διάφορα άλλα APIs γραφικών[27]
- ImPlot : GPU accelerated βιβλιοθήκη απεικόνισης σημάτων, βασισμένη στο ImGui[28]
- libmosquitto-dev : Open source message broker library implementing MQTT in C[29]

Αυτά τα dependencies είναι αρκετά για τους στόχους της εφαρμογής μας, αρκετά ελαφριά όμως για να μην δημιουργούν αχρείαστες καθυστερήσεις στο runtime. Πρέπει λοιπόν για την σωστή ρύθμιση του project να γράψουμε ένα script για την εγκατάσταση όλων των προαπαιτούμενων για το χτίσιμο και την εκτέλεση της εφαρμογής. Εδώ πρέπει να τονίσουμε πως ενώ το PRISM είναι γραμμένο με standard C / C++ και cross platform libraries, η υλοποίηση του έγινε σε σύστημα Debian. Η εγκατάσταση σε διαφορετικά συστήματα είναι εφικτή, ενδεχομένως με μερικές αλλαγές στην απόκτηση των προαπαιτούμενων. Το shell script που υλοποιήσαμε είναι το εξής:

```
#!/bin/bash

set -e

apt update
apt install -y \
     build-essential \
     cmake \
     make \
     libmosquitto-dev \
     git \
     libsdl2-dev
```

Αυτό το script εγκαθιστά τους compilers του GNU (gcc), το build system (cmake), την βιβλιοθήκη του mosquitto, και την βιβλιοθήκη του SDL2.
Τα υπόλοιπα προαπαιτούμενα που αναφέραμε προηγουμένως, δεν θα εγκαταστηθούν απο αυτό το script. Αντιθέτως, θα τα εγκαταστήσουμε ώς submodules στο git repo μας, και θα τα χτίσουμε απο το source τους, καθώς χρειάζονται λίγη παραπάνω ρύθμιση από όλα τα υπόλοιπα.
Έχοντας όλα τα προαπαιτούμενα, μπορούμε να δημιουργήσουμε ένα σχέδιο υψηλού επιπέδου για την εφαρμογή μας. Προφανώς θα ξεκινήσουμε με το CLI, και το GUI θα μείνει

για το τέλος, αφού η μόνη τους διαφορά είναι η υλοποίηση των γραφικών. Στο υψηλότερο επίπεδο του, το PRISM λειτουργεί ως εξής:



Η λειτουργία αυτή υλοποιήθηκε με το ακόλουθο main.cpp file

```
#ifndef __APP_MODE
#define __APP_MODE 1  // cli
#endif
#ifndef __BUILD_MODE
#define __BUILD_MODE 1  // debug
#endif

#include <iostream>
```

```
#include "app.h"
#include "mqtt_client.h"
#include "options_parser.h"

int main(int argc, char** argv)
{
	PRISM::options opts;
	if (!validate_opts(argc, argv, opts))
	{
		std::cout << "invalid arguments, use -h for
help\n";
		exit(EXIT_FAILURE);
	}

	struct mosquitto* mosquitto_client;
	PRISM::queue_context message_q;
	PRISM::init_queue(&message_q);
	if (!init_mqtt(opts, &mosquitto_client, &message_q))
	{
		std::cout
		<< "failed to initialize mosquitto client,
aborting\n";
		exit(EXIT_FAILURE);
	}

	start_loop(mosquitto_client, message_q);

	PRISM::cleanup_queue(&message_q);
	PRISM::cleanup_mqtt(mosquitto_client);

	return 0;
}
```

Αρχικά, δημιουργούμε ένα struct για τις επιλογές που έχει δώσει ο χρήστης, μέσω του command line. Επεξεργαζόμαστε αυτές τις επιλογές και ελέγχουμε την εγκυρότητα τους στην συνάρτηση validate_opts(argc, argv, opts);
Αν αυτές είναι εντάξει, προχωράμε στον ορισμό ενός message queue, στο οποίο θα βάζει μηνύματα το MQTT thread, και απο το οποίο θα αφαιρεί μηνύματα το main thread. Μετά, ξεκινάμε το MQTT thread. Αν αυτό ξεκινήσει σωστά, ξεκινάμε και το main thread, υλοποιώντας το άπειρο loop του προγράμματος. Με το που λήξει αυτό(από την αλληλεπίδραση με τον χρήστη), απελευθερώνουμε όλους τους πόρους που έχουμε καταναλώσει ως τώρα, και τέλος σταματάμε το mqtt client.

Το επόμενο βήμα είναι η επεξεργασία των επιλογών. Για αυτό, θα χρειαστεί να ορίσουμε τα ορίσματα που περιμένει το PRISM απο τον χρήστη. Μια τυπική εκτέλεση του PRISM γίνεται ως εξής(unix build):

./build_cli -H <hostname> -t <topic> -u <username> -P <password> -p <port> -c <path-to-root-certificate>

Προφανώς, τα ορίσματα -H, -t, -u, -P, -p είναι απαραίτητα για την λειτουργία του PRISM. Ο MQTT client που θα τρέχει το δευτερεύον thread πρέπει να ξέρει σε ποιον host να συνδεθεί, σε ποιό port να επικοινωνεί, τα στοιχεία του χρήστη για τον broker, και ενδεχομένως, αλλα όχι υποχρεωτικά ένα root certificate. Σε περίπτωση που ο χρήστης δεν δώσει κάποια από τις απαιτούμενες επιλογές, λαμβάνει το ακόλουθο μήνυμα:

```
panos@pc:~/Documents/PRISM$ ./build/prism_cli
invalid arguments, use -h for help
panos@pc:~/Documents/PRISM$ ./build/prism_cli -h
Predictive realtime industrial sensor monitor
Usage:
  prism_cli / prism_gui [OPTION...]

  -h, --help           Print usage
  -H, --hostname arg   mqtt hostname
  -t, --topic arg      topic to subscribe to
  -u, --username arg   mqtt username
  -P, --password arg   mqtt password
  -p, --port arg       mqtt broker port
  -c, --cert arg       root certificate
panos@pc:~/Documents/PRISM$
```

Το ίδιο μήνυμα λαμβάνει ο χρήστης όταν δώσει λανθασμένα ορίσματα, π.χ. έναν αριθμό στην θέση του hostname, ή μία λέξη στην θέση του port.
Η λειτουργία αυτή έγινε με την ακόλουθη υλοποίηση της validate_opts():

```
bool validate_opts(int argc, char** argv, options& options)
{
  cxxopts::Options opts("prism_cli / prism_gui",
                        "Predictive realtime industrial sensor
monitor");


  opts.add_options()("h,help", "Print usage")("H,hostname", "mqtt
```

```
hostname",

cxxopts::value<std::string>())(
      "t,topic", "topic to subscribe to",
cxxopts::value<std::string>())(
      "u,username", "mqtt username",
cxxopts::value<std::string>())(
      "P,password", "mqtt password",
cxxopts::value<std::string>())(
      "p,port", "mqtt broker port", cxxopts::value<uint32_t>())(
      "c, cert", "root certificate",
cxxopts::value<std::string>());

  cxxopts::ParseResult result;
  try
  {
      result = opts.parse(argc, argv);
  }
  catch (const cxxopts::exceptions::exception& e)
  {
      std::cout << "error parsing options: " << e.what() << '\n';
  }
  if (result.count("help"))
  {
      std::cout << opts.help();
      exit(EXIT_FAILURE);
  }
  if (!result.count("hostname")) return false;
  if (!result.count("topic")) return false;
  if (!result.count("username")) return false;
  if (!result.count("password")) return false;
  if (!result.count("port")) return false;

  options.host = result["hostname"].as<std::string>();
  options.topic = result["topic"].as<std::string>();
  options.username = result["username"].as<std::string>();
  options.password = result["password"].as<std::string>();
  options.port = result["port"].as<uint32_t>();

  if (result.count("cert"))
      options.cert_path = result["cert"].as<std::string>();
  else
```

```
        options.cert_path = "";

    return true;
}
```

Η συνάρτηση αυτή επιστρέφει true μόνο αν ο χρήστης έχει δώσει τα σωστά ορίσματα και αυτά έχουν τον σωστό τύπο. Στο τέλος, οι ρυθμίσεις αυτές γράφονται στο options, το οποίο έχει περαστεί ως reference, επιτρέποντας την τροποποίηση του μέσα στην συνάρτηση.

Σε αυτό το σημείο, έχουμε μερικά έγκυρα ορίσματα τα οποία θα χρησιμοποιήσουμε στο υπόλοιπο της εφαρμογής. Για αρχή, τα χρειαζόμαστε στον mqtt client, τον οποίο αρχικοποιούμε με τα ορίσματα αυτά, μέσω της init_mqtt(), ως εξής:

```
bool init_mqtt(const options& opts, mosquitto** mosq,
queue_context* qc)
{
    mosquitto_lib_init();

    *mosq = mosquitto_new(NULL, true, qc);
    if (!*mosq)
    {
        std::cout << "Failed to create mosquitto client\n";
        return false;
    }

    mosquitto_username_pw_set(*mosq, opts.username.c_str(),
                              opts.password.c_str());

    // register the callback
    mosquitto_message_callback_set(*mosq, queue_inserter_callback);

    mosquitto_tls_set(*mosq, opts.cert_path.c_str(), NULL, NULL,
NULL,
                      NULL);  // sys certs
    if (mosquitto_connect(*mosq, opts.host.c_str(), opts.port, 60)
!=
        MOSQ_ERR_SUCCESS)
    {
        std::cout << "Failed to connect to host\n";
        return false;
    }
```

```
    mosquitto_subscribe(*mosq, NULL, opts.topic.c_str(), 0);

    return true;
}
```

Αρχικά, δημιουργούμε ένα νεο instance του mosquitto, και του θέτουμε ένα username μαζί με ένα password. Αυτά είναι τα στοιχεία με τα οποία θα συνδεθούμε στον broker. Στην συνέχεια, ορίζουμε ενα message reception callback. Το callback αυτό είναι μια συνάρτηση η οποία θα εκτελείται ασύγχρονα κάθε φορά που το mqtt client λάβει ένα μήνυμα απο τον broker. Στην συνέχεια, ορίζουμε το root certificate στην mosquitto_tls_set, με το path που δώθηκε στα ορίσματα. Όπως είπαμε προηγουμένως, ο χρήστης μπορεί να μην δώσει κάποιο certificate, στην οποία περίπτωση περνάμε στην συνάρτηση ένα κενό string “”, το οποίο δεν ορίζει κανένα certificate. Τέλος, επιχειρούμε την σύνδεση στον broker με την mosquitto_connect, περνώντας το hostname, και το port στα οποία αυτός λειτουργεί.

Αυτή η συνάρτηση είναι αρκετά απλή, αλλα πλήρης. Για την σωστή λειτουργία του message queue, πρέπει να ορίσουμε το on_message_callback :

```
void queue_inserter_callback(mosquitto* mosq, void* userdata,
                             const mosquitto_message* msg)
{
    (void)mosq;
    auto qc = static_cast<PRISM::queue_context*>(userdata);
    if (!qc || !qc->mutex || !qc->queue)
    {
        std::cout << "bad queue passed\n";
        exit(EXIT_FAILURE);
    }

    size_t message_count = msg->payloadlen / sizeof(PRISM::message);
    if (msg->payloadlen % sizeof(PRISM::message) != 0)
    {
        std::cout << "bad message size, skipping\n";
        return;
    }
    PRISM::message msg_array[message_count];
    memcpy(msg_array, msg->payload, msg->payloadlen);

    std::lock_guard<std::mutex> lock(*(qc->mutex));
```

```
for (size_t i = 0; i < message_count; i++)
  {
      qc->queue->push(msg_array[i]);
  }
}
```

Όπως φαίνεται, αρχικά παίρνουμε πρόσβαση στο message_queue, μέσω του userdata. Στην συνέχεια ελέγχουμε αν το μήνυμα που μόλις λάβαμε έχει σωστό μέγεθος, δηλαδή, ένα ακέραιο πολλαπλάσιο του μεγέθους του μηνύματος του DATAFLUX (20 bytes). Αν είναι έγκυρο, αντιγράφουμε τα bytes σε ένα stack allocated message array. Τέλος, κλειδώνουμε το mutex του queue(επειδή έχουμε 2 threads που μπορούν να το επεξεργαστούν), και εισάγουμε στο queue όλες τις μετρήσεις που λάβαμε. Με την έξοδο της συνάρτησης το mutex απελευθερώνεται, και το queue μπορεί να χρησιμοποιηθεί ξανά απο το thread που το περιμένει.
Σε αυτό το σημείο, είμαστε έτοιμοι να ξεκινήσουμε το main loop της εφαρμογής, το οποίο επεξεργάζεται τα δεδομένα που λαμβάνουμε:

```
void start_loop(mosquitto* mosq, queue_context& qc)
{
  std::signal(SIGINT, sigint_handler);

  std::unordered_map<uint32_t, std::shared_ptr<sensor>>
sensor_map;
  mosquitto_loop_start(mosq);

  while (should_run)
  {
      poll_message_queue(qc, sensor_map);


      sleep_ms(TICK_MS);
  }

  std::cout << "\nExiting...\n";
  for (auto& [id, sensor_ptr] : sensor_map)
  {
      if (!sensor_ptr->data->empty())
      {
      export_and_clear(*sensor_ptr, id);
      }
  }
```

```
  mosquitto_loop_stop(mosq, true);
}
```

Το container που χρησιμοποιούμε για την αποθήκευση των δεδομένων μας είναι ένα unordered_map, το οποίο αντιστοιχίζει unsigned ids σε αισθητήρες(custom struct). Μετά τον ορισμό αυτού του map, μπορούμε να ξεκινήσουμε το mqtt client thread, με την mosquitto_loop_start(). Αυτή δημιουργεί απο μόνη της το background thread, και το main thread προχωράει με τον υπόλοιπο κώδικα μας. Μόλις το main thread φτάσει στο mosquitto_loop_stop(), το mqtt client thread θα τερματιστεί αυτόματα. Το loop θα σταματήσει με την αλλαγή του flag should_run, το οποίο επηρεάζεται απο το sigint_handler. Με το που στείλει ο χρήστης ένα sigint επομένως, το πρόγραμμα τερματίζει, και κάνει log τα τελευταία δεδομένα. Τέλος, στο main loop, ελέγχουμε αν υπάρχουν νέα δεδομένα στο message_queue, μέσω της poll_message_queue(), και πραγματοποιούμε ένα thread sleep, για ένα σταθερό χρονικό διάστημα. Η χρήση του sleep είναι για να μην χρησιμοποιεί το main thread ένα μεγάλο ποσοστό της CPU(0.1% εν τέλει). Ούτως ή άλλως, ένα sleep των 10 ms είναι αρκετά μικρό για ένα γρήγορο update των δεδομένων(Αυτό εξαρτάται επίσης απο τον ρυθμό με τον οποίο λαμβάνουμε δεδομένα). Το μεγαλύτερο ποσοστό των υπολογισμών λοιπόν πραγματοποιείται στην poll_message_queue(), και γίνεται ως εξής:

```
void poll_message_queue(
      queue_context& qc,
      std::unordered_map<uint32_t, std::shared_ptr<sensor>>&
sensor_map)
{
  std::lock_guard<std::mutex> lock(*(qc.mutex));
  while (!qc.queue->empty())
  {
      message m = qc.queue->front();
      if (!sensor_map.count(m.id))
      {
      sensor_map.insert({m.id, std::make_shared<sensor>(m.id)});
      }
      measurement new_point{m.x, m.y, m.z,
                            static_cast<float>(m.timestamp) /
1000.0f};

      if (sensor_map[m.id]->data->size() > max_vector_size)
      export_and_clear(*sensor_map[m.id], m.id);
      sensor_map[m.id]->data->push_back(new_point);
      sensor_map[m.id]->predictor->step_window();

      auto predictions =
```

```
sensor_map[m.id]->predictor->predict_n_x(2);

      std::cout << "Sensor#" << m.id << " (" << new_point.x << " "
<< new_point.y
           << " " << new_point.z << ") @ " << new_point.timestamp;

      for (const auto& p : predictions)
      {
      std::cout << " " << p;
      }
      std::cout << '\n';

      qc.queue->pop();
  }
}
```

Ξανά, κλειδώνουμε το mutex, και όσο το queue δεν είναι άδειο, επεξεργαζόμαστε το front message. Αν το id σε αυτό το μήνυμα δεν υπάρχει στο map που ορίσαμε προηγουμένως, αυτό σημαίνει ότι δεν έχουμε λάβει ξανά μήνυμα από τον συγκεκριμένο αισθητήρα, άρα θα δημιουργήσουμε έναν καινούργιο αισθητήρα για αυτό το id. Από εκεί και πέρα, αποθηκεύουμε την νέα μέτρηση στα δεδομένα του αισθητήρα, και τα τυπώνουμε στο stdout. Τέλος, υπολογίζουμε τις νέες προβλέψεις για αυτόν τον αισθητήρα και τυπώνουμε και αυτές.

Σε αυτό το σημείο έχουμε περιγράψει πλήρως την λειτουργία του PRISM σε ένα πιο υψηλό επίπεδο αφαίρεσης. Παρακάτω δίνεται ένα παράδειγμα εκτέλεσης του prism_cli, έχοντας ενεργοποιημένο έναν αισθητήρα, με id 2(ορισμένο στο slave_node του DATAFLUX).
Τα μηνύματα που τυπώνονται στο παρακάτω παράδειγμα έχουν την ακόλουθη μορφή:

Sensor#<id> (<x> <y> <z>) @ <timestamp> <x_a> <x_b>

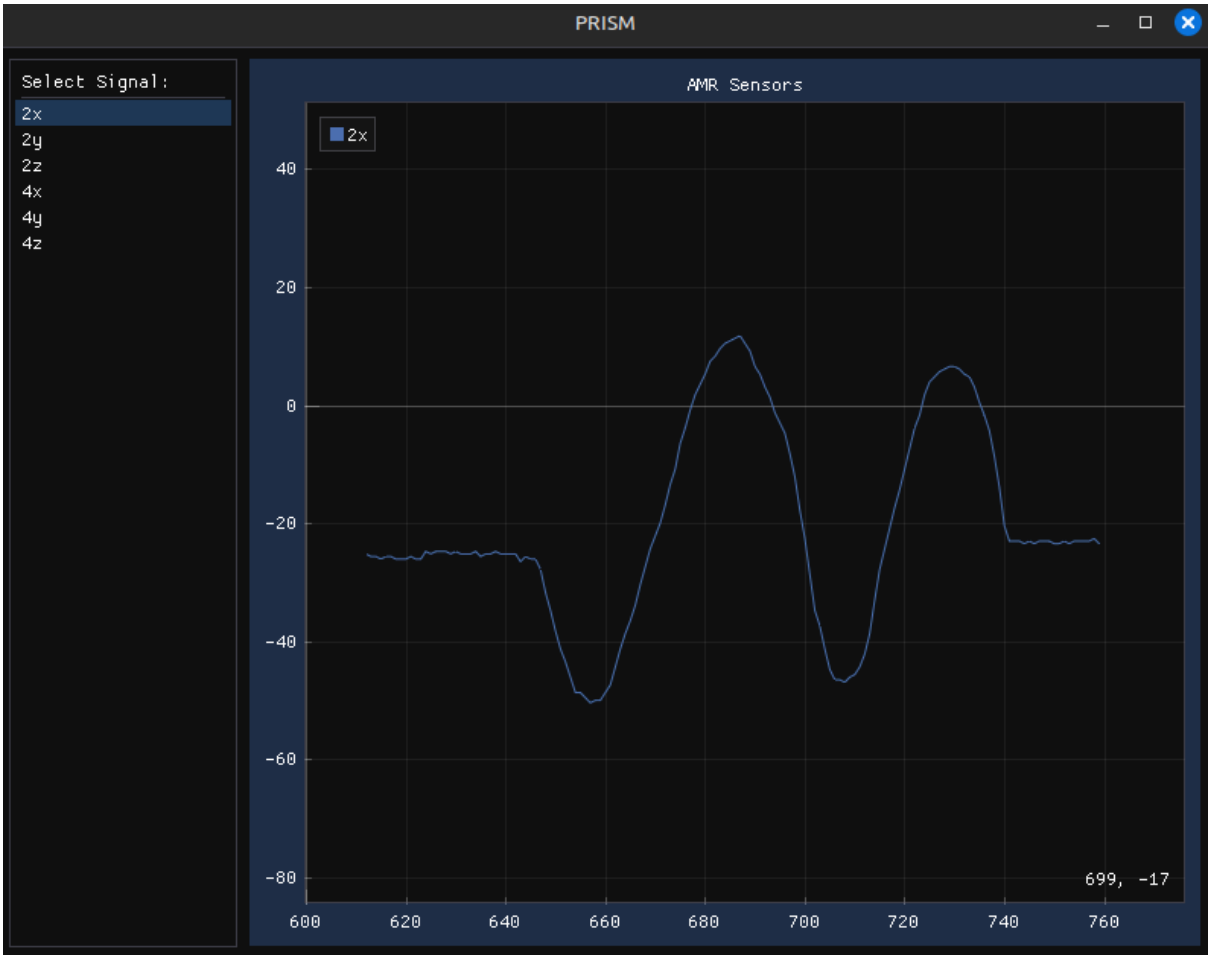
Τα x_a και x_b είναι οι συντελεστές της συνεχούς γραμμικής παλινδρόμησης, για τις μετρήσεις του άξονα χ, με τους οποίους πραγματοποιείται η πρόβλεψη. Αυτοί θα συζητηθούν στο επόμενο κεφάλαιο, αλλά, όπως φαίνεται στο παράδειγμα, το a(σταθερός όρος) πλησιάζει την τιμή του x, ενώ το b(συντελεστής κλίσης) πλησιάζει το 0, ακριβώς όπως περιμένουμε, δεδομένου ότι δεν κινείται ή περιστρέφεται ο αισθητήρας, και οι μετρήσεις παραμένουν περίπου σταθερες.

```
panos@pc:~/Documents/PRISM$ ./command
Sensor#2 (-30.8696 73.913 78.5366) @ 1143.04 0 0
Sensor#2 (-30.4348 73.913 79.0244) @ 1144.04 -362.667 0.291667
Sensor#2 (-30.4348 73.4783 78.5366) @ 1145.04 -242.286 0.1875
Sensor#2 (-31.7391 72.1739 78.5366) @ 1146.04 265.6 -0.259375
Sensor#2 (-31.7391 72.1739 78.5366) @ 1147.04 314.88 -0.3025
Sensor#2 (-31.3043 72.6087 78.0488) @ 1148.04 202.667 -0.203704
Sensor#2 (-31.3043 72.1739 78.5366) @ 1149.04 149.333 -0.157552
Sensor#2 (-31.3043 72.6087 79.0244) @ 1150.04 103.619 -0.11756
Sensor#2 (-31.3043 72.1739 78.0488) @ 1151.04 75.7647 -0.0932904
Sensor#2 (-30.8696 72.1739 78.5366) @ 1152.04 28.3077 -0.0519832
Sensor#2 (-31.7391 72.1739 78.5366) @ 1153.04 45.0526 -0.0666118
Sensor#2 (-31.7391 72.1739 79.0244) @ 1154.04 52.3364 -0.072722
Sensor#2 (-31.3043 72.1739 78.5366) @ 1155.04 36.7248 -0.0591443
Sensor#2 (-31.7391 72.1739 79.0244) @ 1156.04 39.4826 -0.0614117
Sensor#2 (-31.7391 72.1739 79.0244) @ 1157.04 39.5152 -0.061553
Sensor#2 (-31.3043 72.1739 78.0488) @ 1158.04 27.2941 -0.0509191
Sensor#2 (-31.3043 72.1739 78.5366) @ 1159.04 17.8605 -0.0427326
Sensor#2 (-31.3043 71.7391 78.5366) @ 1160.04 10 -0.0357307
Sensor#2 (-31.3043 71.7391 79.0244) @ 1161.04 3.89318 -0.0306009
Sensor#2 (-31.3043 71.7391 78.5366) @ 1162.05 -1.00726 -0.0262561
Sensor#2 (-31.3043 71.7391 78.5366) @ 1163.04 -5.20635 -0.0226935
Sensor#2 (-31.7391 71.3043 79.0244) @ 1164.04 -2.63158 -0.0247738
Sensor#2 (-31.3043 71.3043 78.5366) @ 1165.04 -6.43526 -0.021522
Sensor#2 (-31.3043 71.7391 79.0244) @ 1166.04 -9.6 -0.0188227
Sensor#2 (-31.3043 71.7391 79.0244) @ 1167.04 -12.332 -0.0164896
Sensor#2 (-31.7391 71.3043 79.0244) @ 1168.04 -10.2703 -0.0182116
Sensor#2 (-31.7391 71.3043 79.0244) @ 1169.04 -8.87083 -0.0194575
Sensor#2 (-31.7391 71.3043 78.5366) @ 1170.04 -7.8794 -0.0202968
Sensor#2 (-31.3043 71.7391 78.5366) @ 1171.04 -10.69 -0.0178766
Sensor#2 (-31.3043 71.7391 78.5366) @ 1172.04 -13.1298 -0.0157741
Sensor#2 (-31.3043 71.7391 79.0244) @ 1173.04 -15.1973 -0.0139528
Sensor#2 (-31.7391 71.3043 78.5366) @ 1174.04 -14.1856 -0.0148656
Sensor#2 (-31.7391 71.3043 78.5366) @ 1175.04 -13.4342 -0.0155231
Sensor#2 (-31.7391 71.3043 79.0244) @ 1177.04 -12.9977 -0.0158753
Sensor#2 (-31.7391 71.7391 79.0244) @ 1178.04 -12.703 -0.0160985
Sensor#2 (-31.3043 71.3043 79.0244) @ 1179.04 -14.8664 -0.0142377
Sensor#2 (-31.3043 71.7391 79.0244) @ 1180.04 -16.7023 -0.0126197
Sensor#2 (-31.7391 71.3043 79.0244) @ 1181.04 -16.2971 -0.0129911
Sensor#2 (-31.7391 71.7391 79.0244) @ 1182.04 -16 -0.0132412
Sensor#2 (-31.3043 71.3043 78.5366) @ 1183.04 -17.6462 -0.0118497
Sensor#2 (-31.3043 71.7391 78.5366) @ 1184.04 -19.0731 -0.0106015
Sensor#2 (-31.3043 71.3043 79.0244) @ 1185.04 -20.3444 -0.00950508
Sensor#2 (-31.7391 71.3043 79.0244) @ 1186.05 -19.9209 -0.00988822
Sensor#2 (-31.7391 70.8696 79.0244) @ 1187.04 -19.5959 -0.0101556
Sensor#2 (-31.3043 71.3043 79.0244) @ 1188.04 -20.784 -0.00914253
Sensor#2 (-31.3043 71.3043 78.5366) @ 1189.04 -21.8363 -0.00824738
```

Παρατίθεται ένα ακόμα παράδειγμα στο οποίο έχουμε πολλαπλούς αισθητήρες ενεργοποιημένους(εδώ προβλέπουμε τις 2 επόμενες μετρήσεις αντί για τα α και β):

```
Sensor#4 (10 14.3478 25.3659) @ 1.042 5.45801 4.73445
Sensor#5 (-129.565 -20.8696 69.2683) @ 118.042 -130.222 -130.215
Sensor#2 (-10.8696 59.5652 78.5366) @ 302.052 -10.256 -10.2586
Sensor#4 (10.4348 14.3478 25.3659) @ 2.043 4.86778 4.18712
Sensor#5 (-129.565 -20.4348 69.7561) @ 119.042 -130.219 -130.213
Sensor#5 (-129.565 -20.4348 68.7805) @ 120.041 -130.206 -130.2
Sensor#2 (-10.8696 59.1304 78.5366) @ 304.05 -10.5723 -10.5737
Sensor#5 (-129.13 -20.4348 69.2683) @ 121.041 -130.577 -130.566
Sensor#2 (-10.8696 59.5652 78.5366) @ 305.045 -10.8047 -10.8052
Sensor#5 (-129.565 -20.4348 69.2683) @ 122.043 -130.478 -130.468
Sensor#4 (11.3043 16.5217 25.3659) @ 6.042 4.20708 3.5263
Sensor#4 (11.3043 15.6522 25.8537) @ 7.042 3.60981 2.93132
Sensor#4 (10.4348 15.6522 25.3659) @ 8.043 3.01469 2.33978
Sensor#5 (-129.565 -20.4348 69.2683) @ 125.041 -130.38 -130.372
Sensor#2 (-10.8696 59.1304 79.0244) @ 309.049 -11.0093 -11.009
Sensor#4 (10.8696 15.6522 25.8537) @ 9.042 2.59466 1.93249
Sensor#5 (-129.565 -20.4348 69.2683) @ 126.041 -130.307 -130.299
Sensor#4 (11.7391 17.3913 25.8537) @ 10.043 2.47911 1.84585
Sensor#2 (-10.8696 59.5652 79.0244) @ 311.047 -11.1569 -11.1561
Sensor#4 (10.8696 16.9565 25.3659) @ 11.042 2.38161 1.77708
Sensor#5 (-130 -20.8696 69.2683) @ 128.042 -130.004 -130
Sensor#4 (10.4348 16.087 25.8537) @ 12.042 2.35868 1.78512
Sensor#5 (-130 -20.8696 69.2683) @ 129.041 -129.783 -129.782
Sensor#5 (-129.565 -20.4348 69.2683) @ 130.041 -129.807 -129.805
Sensor#2 (-10.8696 59.5652 78.5366) @ 314.056 -11.2658 -11.2646
Sensor#4 (10.4348 16.5217 25.3659) @ 14.043 2.71683 2.19319
Sensor#5 (-129.13 -20.8696 69.2683) @ 131.041 -129.995 -129.991
Sensor#4 (10.4348 15.6522 25.3659) @ 15.042 3.15025 2.67629
Sensor#5 (-129.565 -20 69.2683) @ 132.041 -129.983 -129.979
Sensor#2 (-10.4348 59.1304 79.0244) @ 316.05 -12.0655 -12.0613
Sensor#4 (11.3043 18.2609 25.3659) @ 16.043 3.85793 3.44333
Sensor#5 (-130 -20 69.2683) @ 133.045 -129.828 -129.826
Sensor#2 (-10.8696 59.1304 79.0244) @ 317.045 -11.9945 -11.9905
Sensor#5 (-130 -20 69.2683) @ 133.045 -129.71 -129.709
Sensor#4 (10.8696 16.5217 25.3659) @ 17.043 4.43233 4.06578
Sensor#5 (-130 -20.4348 69.2683) @ 134.042 -129.61 -129.611
Sensor#2 (-10.8696 59.1304 79.0244) @ 318.042 -11.936 -11.9322
Sensor#4 (10.8696 16.5217 25.3659) @ 18.042 4.98382 4.66043
Sensor#5 (-130 -20.4348 69.2683) @ 135.041 -129.527 -129.529
Sensor#2 (-10.8696 59.5652 78.5366) @ 319.042 -11.8859 -11.8823
Sensor#4 (10.8696 16.9565 25.3659) @ 19.043 5.50056 5.2154
Sensor#5 (-129.565 -20 69.2683) @ 136.041 -129.565 -129.567
Sensor#2 (-10.8696 59.5652 78.5366) @ 320.043 -11.8416 -11.8381
Sensor#5 (-129.565 -20 69.2683) @ 136.041 -129.596 -129.596
Sensor#4 (10.8696 16.5217 25.8537) @ 20.042 5.97617 5.72453
Sensor#5 (-129.565 -20.4348 69.2683) @ 137.041 -129.621 -129.621
Sensor#4 (11.3043 16.5217 25.8537) @ 21.043 6.51195 6.29394
Sensor#2 (-10.8696 59.5652 78.5366) @ 322.045 -11.7978 -11.7945
Sensor#4 (10.8696 16.9565 25.3659) @ 22.043 6.8916 6.69833
Sensor#2 (-10.8696 59.5652 79.0244) @ 323.052 -11.7583 -11.7551
Sensor#4 (10.8696 16.9565 25.3659) @ 23.042 7.23271 7.06092
```

Σε αυτό το σημείο, αξίζει να τονίσουμε ότι τα παραδείγματα του κώδικα που δώσαμε προηγουμένως περιγράφουν το CLI app, ενώ με μερικές αλλαγές, που φαίνονται στο repo προκύπτει το GUI app. Ένα παράδειγμα χρήσης αυτού είναι το εξής:



Όπως φαίνεται, το αριστερό tab widget επιτρέπει την επιλογή του σήματος που σχεδιάζουμε, και στο plot area widget σχεδιάζεται το σήμα.
Αυτό το παράδειγμα είναι μια επαλήθευση οτι ολόκληρο το σύστημα μας δουλεύει σωστά. Η ταλάντωση που φαίνεται είναι το αποτέλεσμα της περιστροφής του αισθητήρα 2 φορές γύρω από τον εαυτό του (άξονας z).

3.3 Αλγόριθμος Πρόβλεψης

Ο αλγόριθμος με τον οποίο υλοποιήσαμε την πρόβλεψη μετρήσεων είναι η απλή γραμμική παλινδρόμηση. Η απλή γραμμική παλινδρόμηση είναι μια μέθοδος με την οποία εξάγουμε μια ευθεία από ένα σύνολο σημείων 2 διαστάσεων, $y = a + b * x$
Προφανώς η ευθεία αυτή δεν γίνεται να περνάει από όλα τα σημεία του δείγματος μας. Αυτά τα σημεία είναι τυχαία. Αυτή η ευθεία όμως ελαχιστοποιεί το άθροισμα των τετραγώνων των σφαλμάτων(αποστάσεων αληθινών τιμών από την ευθεία).
Τα σφάλματα αυτά ορίζονται ως:

$$\varepsilon = y - a - b * x$$

Τα a και b λοιπόν ικανοποιούν την σχέση[30]:

$$(a,\ b)\ =\ argmin(Q(a,\ b))$$

όπου:

$$Q(a,\ b)\ =\ \sum \varepsilon^2$$

Η λύση των παραπάνω δίνει τους τύπους:

$$a\ =\ \frac{\sum y \sum x^2 - \sum x \sum xy}{n\sum x^2 - (\sum x)^2}$$

$$b\ =\ \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2}$$

όπου n το πλήθος των μετρήσεων(μήκος παραθύρου).
Η υλοποίηση του παραπάνω μοντέλου είναι αρκετά απλή, καθώς δεν απαιτεί κάποια δομή δεδομένων, η κάποιον δύσκολο υπολογισμό. Απαιτεί απλά να κρατάμε ένα άθροισμα το οποίο τροποποιούμε κάθε φορά που λαμβάνουμε μια νέα μέτρηση. Παρατίθεται η υλοποίηση του στο PRISM

```
   void step_window()
   {
       if (window_size == max_rolling_window_size)
       {
       sum_t -= data->at(window_start).timestamp;
       sum_tt -=
            data->at(window_start).timestamp *
 data->at(window_start).timestamp;
       sum_x -= data->at(window_start).x;
       sum_xt -= data->at(window_start).x *
 data->at(window_start).timestamp;
       sum_y -= data->at(window_start).y;
       sum_yt -= data->at(window_start).y *
 data->at(window_start).timestamp;
       sum_z -= data->at(window_start).z;
       sum_zt -= data->at(window_start).z *
 data->at(window_start).timestamp;
       window_start++;
       }
       else
```

```
      {
      window_size++;
      }
      window_stop++;
      sum_t += data->at(window_stop).timestamp;
      sum_tt += data->at(window_stop).timestamp *
data->at(window_stop).timestamp;
      sum_x += data->at(window_stop).x;
      sum_xt += data->at(window_stop).x *
data->at(window_stop).timestamp;
      sum_y += data->at(window_stop).y;
      sum_yt += data->at(window_stop).y *
data->at(window_stop).timestamp;
      sum_z += data->at(window_stop).z;
      sum_zt += data->at(window_stop).z *
data->at(window_stop).timestamp;
  }
```

Κάθε φορά που ένας αισθητήρας λαμβάνει μια νέα μέτρηση, καλείται αυτόματα η step_window(). Στην περίπτωση που το παράθυρο το οποίο έχει υπολογιστεί είναι μικρότερο ενός σταθερού μεγέθους, απλα επεκτείνουμε το παράθυρο κατα 1, και προσθέτουμε τους καινούργιους όρους στα αθροίσματα. Στην περίπτωση που το παράθυρο έχει μήκος ίδιο με το μέγιστο, δεν το επεκτείνουμε, προσθέτουμε τους νέους όρους και αφαιρούμε τους πιο παλιούς. Έχοντας υπολογίσει τους συντελεστές α και b λοιπόν, αρκεί απλά να υπολογίσουμε την ποσότητα

$$y = a + b * x$$

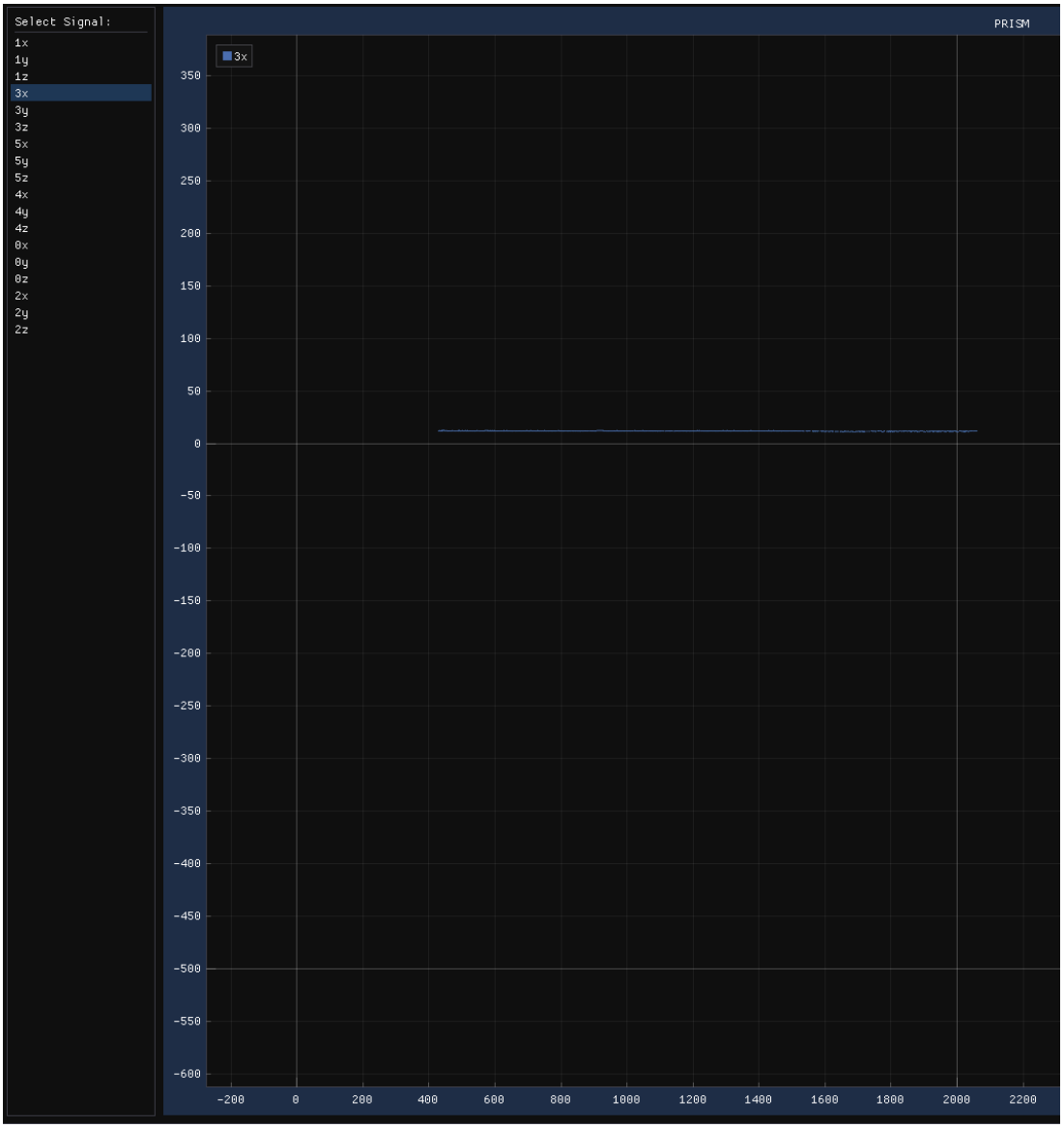
και να αντικαταστήσουμε το x με τιμές μεγαλύτερες του τελευταίου σημείου που μετρήσαμε. Αυτό το μοντέλο δουλεύει καλύτερα για βραχυπρόθεσμες μετρήσεις. Στο PRISM το χρησιμοποιήσαμε για τον υπολογισμό των 2 επομενων δευτερολέπτων.

4 Ανάλυση μετρήσεων και συμπεράσματα

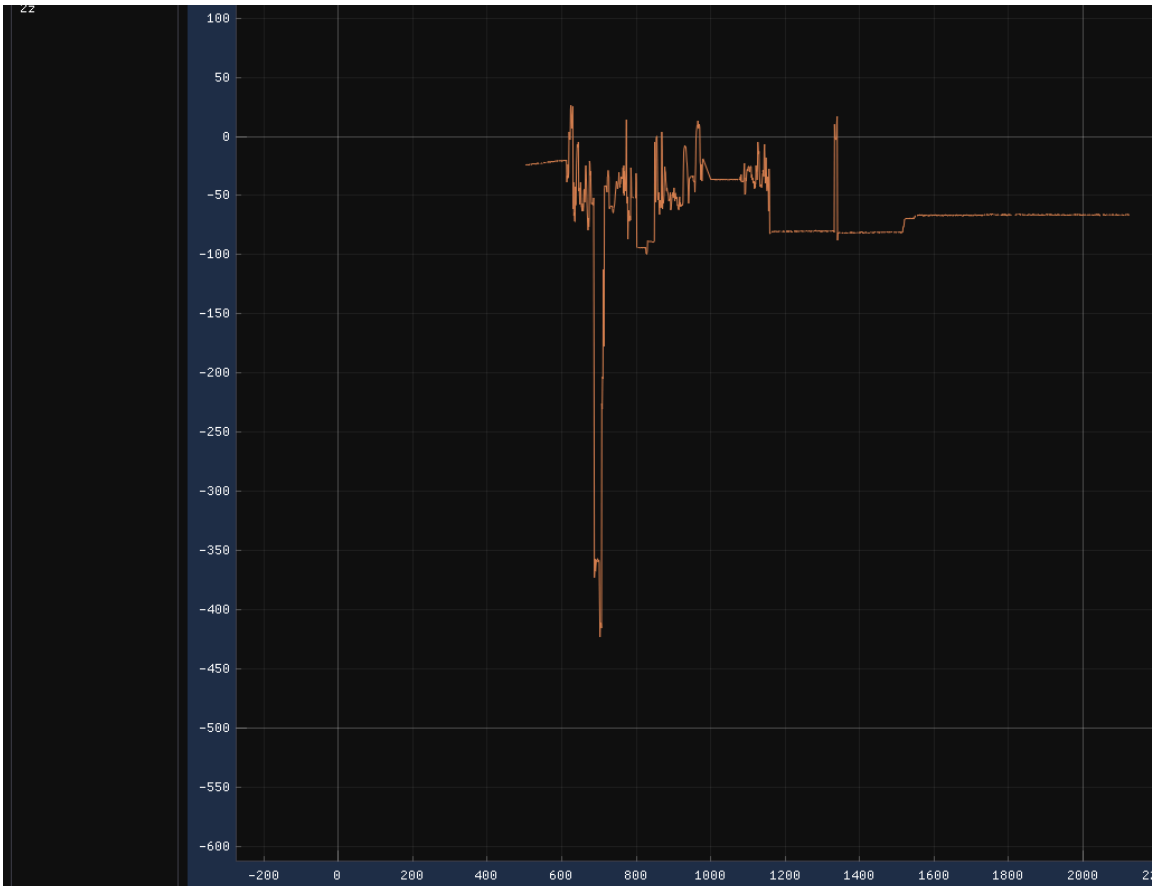
4.1 Πειραματική διάταξη

Σε αυτό το σημείο της εργασίας, έχουμε ένα ολοκληρωμένο σύστημα για την παρακολούθηση των αισθητήρων μας. Χρειαζόμαστε, επομένως, ένα κατάλληλο σενάριο για να αξιολογήσουμε τις επιδόσεις του, και την αξιοπιστία του. Το πείραμα που θα πραγματοποιήσουμε λοιπόν είναι το ακόλουθο: Αρχικά θα κατασκευάσουμε 6 αισθητήρες (slave nodes), με τα κουτιά τους, τις μπαταρίες τους και τους esp32c6 τους, ακριβώς όπως περιγράψαμε στο δεύτερο κεφάλαιο. Θα εφαρμόσουμε τους αισθητήρες αυτούς στην σιδερένια σκάλα στην πίσω μεριά του νέου κτιρίου ΣΗΜΜΥ. Με τους αισθητήρες αυτούς σκοπεύουμε να παρακολουθήσουμε την υγεία αυτής της σιδερένιας κατασκευής (SteHeMon). Η παρακολούθηση της έντασης του μαγνητικού πεδίου σε διάφορα σημεία μίας σιδερένιας κατασκευής είναι μια απο τις πιο συνηθισμένες μη καταστροφικές μεθόδους εκτίμησης της κατάστασης / υγείας της[31]. Το πλεονέκτημα αυτών των μεθόδων είναι ότι δεν αχρηστευουν το υλικό μετά την μέτρηση, ακόμα και αν τα αποτελέσματα τους είναι σε γενικές γραμμές λιγότερο αξιόπιστα. Οι αισθητήρες μας λοιπόν, θα εφαρμοστούν στην κάτω μεριά ορισμένων σκαλοπατιών, και θα στέλνουν δεδομένα τα οποία θα διαβάζουμε μέσω του PRISM. Η υπόθεση που κάνουμε λοιπόν πριν ξεκινήσουμε το πείραμα είναι ότι θα παρατηρήσουμε μια μεταβολή στην ένταση μαγνητικού πεδίου στα σκαλοπάτια, όταν κάποιος περπατάει πάνω από τους αισθητήρες. Σε κάθε άλλη στιγμή, περιμένουμε το πεδίο που θα δούμε θα είναι σταθερό. Στην συνέχεια, θα υλοποιήσουμε ένα mqtt gateway χρησιμοποιώντας έναν esp32e (master node), πάλι, όπως περιγράφηκε στο δεύτερο κεφάλαιο, για να μπορούμε να παρακολουθούμε αυτές τις μεταβολές ζωντανά. Τέλος θα ξεκινήσουμε δύο PRISM clients(ένα gui και ένα cli), και θα αξιολογήσουμε την αξιοπιστία του συστήματος.

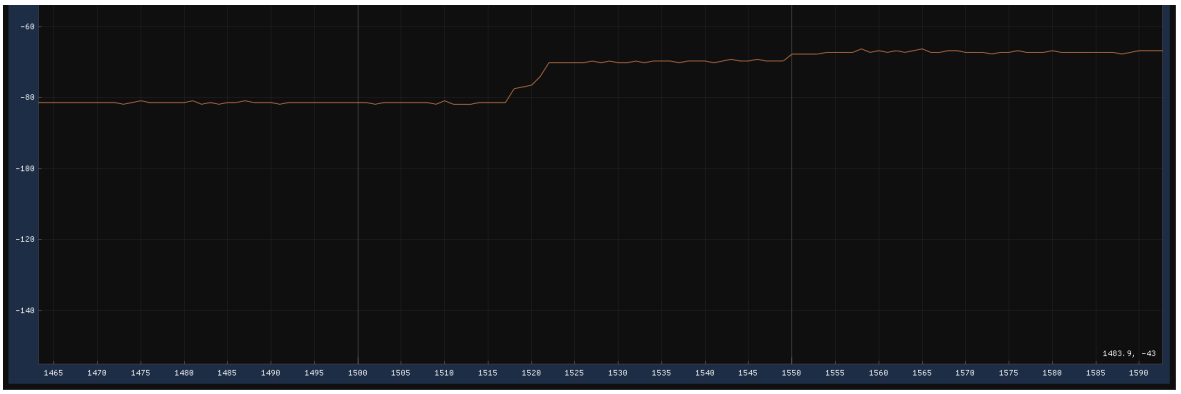
Στην συνέχεια παρατίθεται το μαγνητικό πεδίο που παρατηρήσαμε στον αισθητήρα 3



Όπως φαίνεται, το πεδίο σε αυτό το σημείο της σκάλας είναι σταθερό. Και η υπόθεση που κάναμε για αυτό το πείραμα αποδεικνύεται ότι δεν είναι αληθής. Παρά τις πολλαπλές δοκιμές που κάναμε(περπατώντας στο σκαλί του αισθητήρα 3), η ένταση του μαγνητικού πεδίου δεν είδε καμία μεταβολή. Σε αντίθεση με τον αισθητήρα 3 όμως, είδαμε μεταβολή στον αισθητήρα 1:



Οι μεταβολές που φαίνονται πριν από το δευτερόλεπτο 1400 έγιναν με την χρήση ενός μόνιμου μαγνήτη, απλά για την επαλήθευση οτι δεν υπάρχει κάποιο πρόβλημα με τους ίδιους τους αισθητήρες. Η αληθινή μεταβολή φαίνεται περίπου στο δευτερόλεπτο 1500. Τότε, η ένταση του μαγνητικού πεδίου πραγματοποιεί ένα μικρό άλμα, απο τα -80uT στα -70uT.



Εκείνη τη στιγμή, δοκιμάσαμε να κάνουμε μερικά άλματα πάνω στο σκαλί του αισθητήρα 1. Αυτά όντως προκάλεσαν μια μικρή μεταβολή στο μαγνητικό πεδίο σε εκείνο το σκαλί.

Το τελικό συμπέρασμα του πειράματος, λοιπόν, είναι ότι δεν μπορούμε να παρακολουθήσουμε πότε κάποιος περπατάει στα σιδερένια σκαλιά χρησιμοποιώντας αισθητήρες AMR, τουλάχιστον όχι με ακρίβεια. Προφανώς όμως ένα τέτοιο δίκτυο αισθητήρων είναι ιδανικό για την παρακολούθηση της υγείας του σιδήρου. Επίσης, το σύστημα που σχεδιάσαμε αποδεικνύεται αρκετά αξιόπιστο.

```
Sensor#0 (-42.6087 -6.95652 -66.8293) @ 9947.04 -42.9988 -42.9967
Sensor#1 (-23.4783 -32.1739 -66.8293) @ 9845.04 -24.2241 -24.222
Sensor#3 (11.3043 14.7826 51.2195) @ 9775.06 10.6085 10.61
Sensor#4 (16.5217 -20.8696 -23.4146) @ 9543.04 17.1355 17.1349
Sensor#5 (-11.3043 6.95652 23.4146) @ 10221 -11.37 -11.3702
Sensor#0 (-42.6087 -6.95652 -66.8293) @ 9948.04 -43.1914 -43.1893
Sensor#2 (-2.6087 36.5217 29.7561) @ 9609.04 -2.60309 -2.60303
Sensor#1 (-23.4783 -32.1739 -66.3415) @ 9846.04 -24.1997 -24.1975
Sensor#3 (10.8696 14.7826 51.2195) @ 9776.06 10.7842 10.7854
Sensor#4 (16.9565 -20.8696 -23.9024) @ 9544.04 17.1456 17.1451
Sensor#5 (-11.3043 6.95652 22.9268) @ 10222 -11.7016 -11.7017
Sensor#0 (-42.6087 -6.52174 -66.3415) @ 9949.04 -43.148 -43.146
Sensor#2 (-2.17391 36.5217 29.7561) @ 9610.04 -2.73844 -2.73794
Sensor#1 (-23.4783 -32.6087 -66.3415) @ 9847.04 -24.0461 -24.044
Sensor#3 (11.3043 15.2174 51.7073) @ 9777.06 10.7631 10.7642
Sensor#4 (16.5217 -21.3043 -23.4146) @ 9545.04 17.1442 17.1433
Sensor#5 (-11.7391 6.95652 23.4146) @ 10223 -11.5519 -11.5523
```

Στο πειραματικό σενάριο αυτό, θέσαμε την διάρκεια του κάθε κύκλου των slave nodes ακριβώς σε ένα δευτερόλεπτο. Επίσης θέσαμε το διάστημα μεταξύ διαδοχικών transmissions του master node στα 3 δευτερόλεπτα. Αυτό σημαίνει πως τα prism clients, θα λαμβάνουν 3 μετρήσεις από κάθε αισθητήρα κάθε 3 δευτερόλεπτα. Όπως φαίνεται από τα logs παραπάνω, οι αισθητήρες κατάφεραν να στέλνουν μετρήσεις συνεχόμενα με αυτόν τον ρυθμό για 3 ώρες. Δυστυχώς δεν είχαμε περισσότερο χρόνο για να δοκιμάσουμε την διάρκεια ζωής του κάθε αισθητήρα, αν και η εκτίμηση που κάναμε προηγουμένως για 10 μέρες δεν πρέπει να είναι πολύ μακριά από την αληθινή διάρκεια. Τα δεδομένα που λάβαμε επίσης, είναι ακριβώς αυτά που αναμέναμε, επαληθεύοντας την αξιοπιστία των αισθητήρων. Εν τέλει, το σύστημα που κατασκευάσαμε είναι αρκετά χρήσιμο για την παρακολούθηση οποιουδήποτε τύπου αισθητήρα (Υπενθυμίζουμε πως τα 3 repositories που γράψαμε δεν βασίζονται το ένα στο άλλο. Αυτή η υλοποίηση μας επιτρέπει να χρησιμοποιήσουμε το ίδιο δίκτυο με διαφορετικούς αισθητήρες, και προφανώς διαφορετικούς drivers).

4.2 Αξιολόγηση και προτάσεις για συνέχεια της εργασίας

Σε αυτό το σημείο, αξίζει να σημειώσουμε ορισμένες “ανεπάρκειες” του δικτύου μας, και να προτείνουμε λύσεις.

- Ο driver ALSMD δεν υλοποιεί μετρήσεις θερμοκρασίας. Οι αισθητήρες της οικογένειας LSM303 έχουν ενσωματωμένους αισθητήρες θερμοκρασίας, τις μετρήσεις των οποίων μπορούμε να λάβουμε με τρόπο όμοιο με αυτόν που ακολουθήσαμε για το μαγνητόμετρο / επιταχυνσιόμετρο. Η μέτρηση θερμοκρασίας θα μπορούσε να αποδειχθεί χρήσιμη για το SteHeMon, καθώς θα μπορούσε να εξηγήσει ορισμένες μεταβολές του μαγνητικού πεδίου.

- Η επικοινωνία μεταξύ των slave nodes και του master node είναι μονόδρομη! Στην τωρινή υλοποίηση, τα slave nodes στέλνουν μηνύματα στο master node, χωρίς να υλοποιείται η αντίστροφη κατεύθυνση. Αν μπορούσε το master node να στείλει μηνύματα στα slave nodes, θα λύνονταν αυτομάτως τα παρακάτω προβλήματα:
- Μη ύπαρξη διαγνωστικών μηνυμάτων / επικοινωνία PRISM - DATAFLUX.
- Αδυναμία ελέγχου αισθητήρων (π.χ. ρύθμιση ρυθμού μετρήσεων) απο τα PRISM clients.
- Hard coded διευθύνσεις MAC. Οι αισθητήρες στέλνουν τις μετρήσεις τους σε μια σταθερή διεύθυνση MAC(master node). Αν είχαμε επικοινωνία μεταξύ των slave nodes και των master nodes, αυτά θα μπορούσαν αυτομάτως να υπολογίζουν την διεύθυνση του καθενός, και η ρύθμιση του ολικού συστήματος θα ήταν πολύ πιο εύκολη.
- Αυτόματος υπολογισμός των identities των αισθητήρων. Αν τα master nodes υπολόγιζαν το identity του κάθε αισθητήρα αυτόματα(ίσως με κάποιο hash) τότε οι αισθητήρες δεν θα έπρεπε να στέλνουν την ταυτότητα τους σε κάθε μήνυμα(20 bytes -> 16 bytes). Από εκεί, ο master node θα μπορούσε απλά να στέλνει τις μετρήσεις της κάθε ταυτότητας σε ένα νέο mqtt topic, το όνομα του οποίου βασίζεται στην ταυτότητα αυτή. Αυτή η αλλαγή θα είχε ως αποτέλεσμα την μείωση του όγκου δεδομένων κατα 20% αφού πλέον δεν στέλνουμε τις ταυτότητες κάθε φορα!
- Η λογιστική παρεμβολή είναι ένα αρκετά απλό μοντέλο πρόβλεψης. Παρά την ευκολία στην υλοποίηση αυτού του μοντέλου, και τις εύστοχες προβλέψεις του, αυτές είναι αρκετά βραχυπρόθεσμες. Μια πιθανή λύση είναι η εκπαίδευση ενός μοντέλου τεχνητής νοημοσύνης και η χρήση αυτού για τις προβλέψεις.

Σε αυτή την εργασία, υλοποιήσαμε ένα ασύρματο δίκτυο αισθητήρων με ζωντανές μετρήσεις και προβλέψεις. Αυτή η εργασία μπορεί να χρησιμοποιηθεί σαν οδηγός για την κατασκευή ενός παρόμοιου συστήματος, ή ακόμα και σαν βάση, με την προσθήκη των προτεινόμενων ή και άλλων αλλαγών, με όποιον τρόπο θεωρεί ο κάθε χρήστης σκόπιμο. Ενώ η υπόθεση του πειράματος μας αποδείχθηκε ψευδής, αυτό το δίκτυο αισθητήρων μπορεί να υπάρξει αρκετά χρήσιμο για εφαρμογές του SteHeMon. Ενώ αυτό το δίκτυο αισθητήρων επιδέχεται αρκετές βελτιώσεις, είναι ελαφρύ, αξιόπιστο και αρκετά εύκολο στην χρήση. Τέλος, προτείνεται στον αναγνώστη η χρήση των repositories που αναπτύξαμε σε αυτή την εργασία, και η τροποποίηση τους για την χρήση σε οποιοδήποτε project χρησιμοποιεί πολλαπλούς απομακρυσμένους αισθητήρες.

5 Βιβλιογραφία

[1] Mohammed Asadullah Khan et al 2021 Eng. Res. Express 3 022005, “Magnetic sensors – A review and recent technologies”
[2]D. J. Griffiths, “*Introduction to Electrodynamics”*, 3rd ed. Cambridge, 1999
[3]R. Nave, "Magnetic Dipole Moment," *HyperPhysics*, Georgia State University. [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/magmom.html
[4]V G Bar'yakhtar et al “The physics of magnetic domains”, 1988 Sov. Phys. Usp. 31 810
[5]L. LANDAU, E. LIFSHITS, “ON THE THEORY OF THE DISPERSION OF MAGNETIC PERMEABILITY IN FERROMAGNETIC BODIES”
[6] S. O. Kasap, Principles of Electronic Materials & Devices, McGraw-Hill Education, 2018.
[7] P. Bruno, "Geometrically constrained magnetic wall," *Physical Review Letters*, vol. 83, no. 12, pp. 2425–2428, 1999.
[8] Andrä, W. “On the bloch wall motion in multilayer films”. *Czech J Phys* **21**, 522–524 (1971)
[9] D. C. Jiles and D. L. Atherton, “Theory of ferromagnetic hysteresis,” *Journal of Magnetism and Magnetic Materials*, vol. 61, no. 1–2, pp. 48–60, 1986. doi: 10.1016/0304-8853(86)90066-1
[10] K. Fabian, V. P. Shcherbakov, and S. A. McEnroe, “Measuring the Curie temperature,” *Geochemistry, Geophysics, Geosystems*, vol. 14, no. 4, pp. 943–958, Apr. 2013. doi: 10.1029/2012GC004440
[11] J. M. D. Coey, “Permanent magnet applications,” *Journal of Magnetism and Magnetic Materials*, vol. 248, no. 3, pp. 441–456, Aug. 2002. doi: 10.1016/S0304-8853(02)00352-9
[12] E. Hristoforou, A. Ktena, and S. Gong, “Magnetic Sensors: Taxonomy, Applications, and New Trends,” *Laboratory of Electronic Sensors, School of Electrical and Computer Engineering, NTUA*, 2021.
[13] P. Ripka, *Magnetic Sensors and Magnetometers*. Norwood, MA, USA: Artech House, 2001.
[14] E. V. Hristoforou, "Permeability sensors for magnetic steel structural health monitoring," *Laboratory of Electronic Sensors, Zografou Campus, National Technical University of Athens*, 15780 Athens, Greece
[15] E. Hristoforou, "Magnetic Effects in Physical Sensor Design and Development," *Laboratory of Physical Metallurgy, Department of Mining and*

Metallurgical Engineering, National Technical University of Athens, Athens 15780, Greece
[16]B. Cerruti, G. Durin, and S. Zapperi, "Hysteresis and noise in ferromagnetic materials with parallel domain walls," *Phys. Rev. B*, vol. 79, no. 13, p. 134429, Apr. 2009, doi: 10.1103/PhysRevB.79.134429.
[17] M. Ziese and M. J. Thornton, Eds., *Spin Electronics*, Lecture Notes in Physics, vol. 569. Berlin, Germany: Springer, 2001. doi: 10.1007/3-540-45258-3
[18] STMicroelectronics, LSM303DLHC - "Ultra-compact high-performance eCompass module: 3D accelerometer and 3D magnetometer", datasheet. [Online]. Available: https://www.st.com/resource/en/datasheet/lsm303dlhc.pdf
[19] Nick O'Leary, pubsubclient, v2.8, GitHub, 2020. [Online]. Available: https://github.com/knolleary/pubsubclient
[20] ESPRESSIF, esp-now - GitHub, 2022. [Online]. Available: https://github.com/espressif/esp-now
[21] ESPRESSIF, Sleep Modes - ESP-IDF Programming guide. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/sleep_modes.html
[22] ESPRESSIF, Memory Types - ESP-IDF Programming guide. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/memory-types.html
[23] DFRobot, Beetle ESP32-C6 wiki[Online]. Available: https://wiki.dfrobot.com/SKU_DFR1117_Beetle_ESP32_C6
[24] FreeRTOS, Queue Management, Documentation. [Online]. Available: https://freertos.org/Documentation/02-Kernel/04-API-references/06-Queues/00-QueueManagement
[25] SDL, "Simple DirectMedia Layer," *SDL Official Website*. https://www.libsdl.org/.
[26] J. Roach, *cxxopts: Lightweight C++ command line option parser*, GitHub. https://github.com/jarro2783/cxxopts (accessed Jun. 12, 2025).
[27] O. Cornut, *Dear ImGui: Bloat-free graphical user interface for C++ with minimal dependencies*, GitHub. https://github.com/ocornut/imgui
[28]B. Epezent, *ImPlot: Immediate Mode Plotting*, GitHub. https://github.com/epezent/implot

[29] R. Light, *Eclipse Mosquitto: An open source MQTT broker*, Eclipse Foundation. https://mosquitto.org/
[30] *Simple linear regression*, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Simple_linear_regression
[31] G. Stamou, S. Angelopoulos, A. Ktena, and E. Hristoforou, "3D Anisotropic Magnetoresistance sensor for steel health monitoring," *National Technical University of Athens, School of Electrical and Computer Engineering, Laboratory of Electronic Sensors*, Zografou, Greece, and *National and Kapodistrian University of Athens, General Department*, Zografou, Greece

6 Github Repositories

Ακολουθούν οι σύνδεσμοι για τα GitHub Repositories που αναφέρθηκαν σε αυτή την εργασία

PRISM - https://github.com/PanagiotisPrountzosCS/PRISM

ALSMD - https://github.com/PanagiotisPrountzosCS/ALSMD

DATAFLUX - https://github.com/PanagiotisPrountzosCS/DATAFLUX

PubSubClient - https://github.com/knolleary/pubsubclient

SDL2 - https://github.com/libsdl-org/SDL

cxxopts - https://github.com/jarro2783/cxxopts

IMGUI - https://github.com/ocornut/imgui

IMPLOT - https://github.com/epezent/implot

mosquitto - https://github.com/eclipse-mosquitto/mosquitto