ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ασύρματο Δίκτυο Αισθητήρων για τον Αυτόματο Έλεγχο Φωτισμού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλέξανδρος Π. Πεϊτσίνης

Επιβλέπων: Συκάς Ευστάθιος
Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ασύρματο Δίκτυο Αισθητήρων για τον Αυτόματο Έλεγχο Φωτισμού

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλέξανδρος Π. Πεϊτσίνης

Επιβλέπων: Συκάς Ευστάθιος
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή επιτροπή το Μάρτιο του 2016

.....	.....	.....
Ε. Συκάς	Μ. Θεολόγου	Γ. Στασινόπουλος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2015

……

Αλέξανδρος Π. Πεϊτσίνης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός υπολογιστών Ε.Μ.Π.

Πρόλογος

Η κατανάλωση ενέργειας σε κτίρια (εταιρίες ή σπίτια) ανέρχεται στο 40% της συνολικής κατανάλωσης ετησίως στις οικονομικά ανεπτυγμένες χώρες. Γι' αυτό τον λόγο και τα τελευταία χρόνια έχουν έρθει στο προσκήνιο έννοιες όπως "πράσινη ενέργεια", "εξοικονόμηση ενέργειας" και "επαναχρησιμοποίηση ενέργειας", οι οποίες μπορούν να ενταχθούν σε μια γενικότερη έννοια, την "ενεργειακή αποδοτικότητα".

Ένα ενεργειακά αποδοτικό κτίριο εξασφαλίζει κατ'αρχάς ότι θα χρησιμοποιείται σημαντικά μεγάλο ποσοστό της παρεχόμενης ενέργειας, ελαχιστοποιώντας τις απώλειες. Αυτό επιτυγχάνεται για παράδειγμα χάρη σε τεχνολογίες όπως θερμικά και φωτοβολταϊκά panels, ανεμογεννήτριες, και άλλα.

Επιπρόσθετα, ένα κτίριο κατάλληλα σχεδιασμένο μπορεί να εξασφαλίσει τη μειωμένη χρήση πόρων, διακόπτοντας κάποιες λειτουργίες όταν δεν είναι απαραίτητες. Αυτό το αποτέλεσμα προσφέρει για παράδειγμα η χρήση θερμοστάτη για τον έλεγχο της θέρμανσης, και φωτοκυττάρου για τον έλεγχο του φωτισμού.

Ειδικά τα τελευταία χρόνια, η κατασκευή ενεργειακά αποδοτικών κτιρίων έχει επεκταθεί και στον τομέα της αρχιτεκτονικής, λαμβάνοντας υπ' όψη τη σκίαση, τον προσανατολισμό και το landscaping.

Όλα τα παραπάνω έχουν διττό σκοπό και αποτέλεσμα: ενώ υπάρχουν οικονομικά οφέλη για την εταιρία, κυρίως χάρη στη μείωση των εξόδων για λογαριασμούς, αυτό έχει ως αποτέλεσμα να εισάγεται ο κοινωνικός και ο περιβαλλοντικός παράγοντας, μέσω της μείωσης των εκπομπών και της εξοικονόμησης πρώτων υλών.

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η διερεύνηση και ανάπτυξη μιας καινοτόμας λύσης, μέσω της οποίας θα εφαρμοσθούν σύγχρονες τεχνολογίες για τον έλεγχο συστημάτων φωτισμού, με σκοπό την εξοικονόμηση ενέργειας και γενικά την ενεργειακή αποδοτικότητα.

Λέξεις κλειδιά

φωτισμός, ενέργεια, αποδοτικότητα, αυτοματοποίηση, αισθητήρας, έλεγχος, ασύρματο

Abstract

Power consumption in buildings (houses or offices) constitutes roughly 40% of the yearly total in economically advanced regions. For this reason, in the last years there is an emersion of several terms, such as "green energy", "power saving" and "energy reuse", which can be described by a broader term, "energy efficiency".

First of all, an energy efficient building ensures that a substantial amount of the energy provided will be used, minimizing dissipation. This is achieved with the use of thermal and photovoltaic panels, wind turbines and many more.

Additionally, an appropriately designed building can minimize the use of resources by cutting off several power draining tasks when not necessary. This can be achieved by using a thermostat for controlling heat, or a photoelectric cell for controlling lighting.

Especially in the last few years, the idea of constructing energy efficient bultings has been extended to the field of architecture, taking into account the shading of the building, its orientation and its landscaping.

Energy efficiency has a twofold purpose and result: while it can lead to company profits by cutting off expenses for bills, there is also a social and environmental factor, due to the minimization of emissions and raw materials saving.

The purpose of this diploma thesis is the development of an innovative system, which will allow the application of modern technologies to lighting systems, with the utter purpose of rendering them energy efficient.

Key words

lighting, energy, efficiency, automated, sensor, measurement, LDR, Arduino, Raspberry Pi, control, wireless, RF, NRF, NRF24L01, dimmer, Python, Flask, JavaScript, HTML, GUI, WebGUI, task scheduling

Πίνακας περιεχομένων

Κεφάλαιο 1 Εισαγωγή

1.1 Σκοπός της διπλωματικής εργασίας

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η διεξαγωγή και υλοποίηση μιας έρευνας που θα επιτρέψει τη χρήση σύγχρονων τεχνολογιών για την ελαχιστοποίση της κατανάλωσης από πηγές φωτισμού σε έναν χώρο (εταιρία ή σπίτι).

Υπάρχουν τομείς στους οποίους έχουν γίνει σημαντικά βήματα για την ένταξη σύγχρονων τεχνολογιών με σκοπό την εξοικονόμηση ενέργειας. Ένα πολύ χαρακτηριστικό παράδειγμα είναι η χρήση θερμοστάτη για τη διατήρηση του επιπέδου θερμοκρασίας σε μια προκαθορισμένη τιμή, χωρίς να είναι συνεχώς σε λειτουργία όλες οι πηγές θέρμανσης.

Όσον αφορά τον φωτισμό, έχουν επίσης γίνει πολυάριθμες τεχνολογικές καινοτομίες. Η αντικατάσταση των λαμπτήρων πυρακτώσεως από λαμπτήρες φθορισμού, οι οποίοι παρέχουν το ίδιο επίπεδο φωτισμού καταναλώνοντας το 1/3 της ενέργειας, και η πρόσφατη άνοδος στην χρήση της τεχνολογίας LED, είναι πολύ σημαντικά βήματα στην ενεργειακή αποδοτικότητα των κτιρίων.

Επιπρόσθετα, υπάρχει η δυνατότητα τοποθέτησης αισθητήρων κίνησης για τον έλεγχο του φωτισμού, με σκοπό την παροχή ενέργειας σε λαμπτήρες μόνο όταν χρειάζεται. Αυτή η λύση όμως δεν είναι εφαρμόσιμη π.χ. σε έναν χώρο γραφείου.

Στα επόμενα κεφάλαια θα αναλυθεί η ανάπτυξη ενός συστήματος που εξασφαλίζει ομοιόμορφο επίπεδο φωτεινότητας σε έναν χώρο. Το σύστημα θα έχει τη δυνατότητα να λαμβάνει υπ' όψη περιβαλλοντικούς παράγοντες όπως το φυσικό φως, και θα μπορεί να ρυθμίζει κατάλληλα το επίπεδο φωτεινότητας σε έναν χώρο, παρεμβαίνοντας στην τροφοδοσία των λαμπτήρων. Με αυτό τον τρόπο αυτοματοποιείται το σύνολο των λαμπτήρων ενός χώρου, ανεξάρτητα ο ένας από τον άλλο, ανάλογα με το συνολικό αντίκτυπο που έχουν στο επίπεδο φωτεινότητας του χώρου αυτού.

1.2 Δομή της διπλωματικής εργασίας

Στο κεφάλαιο 2 γίνεται μια παρουσίαση των χαρακτηριστικών και των απαιτήσεων του συστήματος, καθώς και των κύριων δομικών χαρακτηριστικών. Επίσης αναλύονται οι υπηρεσίες που αναμένεται να προσφέρει το σύστημα, και γίνεται μια εκτίμηση σχετικά με τα κόστη υλοποίησης και λειτουργίας.

Το 3ο κεφάλαιο περιέχει μια εκτενή παρουσίαση και ανάλυση όλων των υποψήφιων τεχνολογικών εξαρτημάτων που λήφθηκαν υπ'όψη κατά την ανάπτυξη του συστήματος. Στο τέλος παρουσιάζονται και τα κριτήρια επιλογής κάθε επιμέρους εξαρτήματος.

Στο κεφάλαιο 4 εξετάζεται ο τρόπος υλοποίησης του συστήματος βάσει των επιλεγμένων τεχνολογιών. Εδώ το σύστημα διαχωρίζεται ιεραρχικά, και γίνεται μια διερεύνηση για κάθε επίπεδο ξεχωριστά, παρουσιάζοντας τους αλγορίθμους που οδηγούν κάθε κόμβο.

Το κεφάλαιο 5 αποτελεί το κύριο μέρος της εργασίας. Περιλαμβάνει μια εκτενή διερεύνηση των χαρακτηριστικών του συστήματος σε επίπεδο hardware (ηλεκτρικά χαρακτηριστικά) και software (υλοποίηση των αλγορίθμων). Έπειτα γίνεται μια επεξήγηση της διαδικασίας κατασκευής όλων των τερματικών σε χρησιμοποιήσιμες συσκευές. Τέλος, παρουσιάζεται ο τελικός αλγόριθμος βέλτιστου φωτισμού, καθώς οι λύσεις σε κάποια προβλήματα που υπήρξαν εμπόδια κατά τη σχεδίαση.

Το 6ο κεφάλαιο περιλαμβάνει τη διαδικασία των δοκιμών που διεξήχθησαν, για να εκτιμηθεί η απόδοση του συστήματος.

Το 7ο κεφάλαιο έχει σκοπό την αποτίμηση της απόδοσης του όλου συστήματος, καθώς και την περιγραφή κάποιων δυνατών επεκτάσεων που θα μπορούσαν να εφαρμοσθούν στο μέλλον.

Στο 8ο κεφάλαιο παρατίθεται η βιβλιογραφία διάφορες πηγές, και τέλος στα παραρτήματα παρουσιάζεται ο εξοπλισμός που χρησιμοποιήθηκε, καθώς και ο πηγαίος κώδικας (source code) του συστήματος.

1.3 Σύνδεσμοι

Η παρούσα εργασία, ο κώδικας και άλλα χρήσιμα αρχεία και links βρίσκονται στο GitHub repository https://github.com/alexpeits/lights-diploma-thesis

Κεφάλαιο 2 Ανάλυση της λύσης

Όπως αναφέραμε στην εισαγωγή, ακόμα και με την πληθώρα των καινοτομιών που έχουν γίνει στον τομέα της ενεργειακής αποδοτικότητας σε κτίρια, δεν έχει υπάρξει ακόμα κάποια αποδοτική λύση σχετικά με την αυτοματοποίηση του φωτισμού. Ακόμα και η χρήση αισθητήρων κίνησης για τον έλεγχο της λειτουργίας ή μη μιας ομάδας λαμπτήρων δεν λύνει το πρόβλημα αυτό, καθώς έχει εφαρμογή μόνο σε χώρους που δεν υπάρχει κινητικότητα κατά το μεγαλύτερο ποσοστό της ημέρας.

Μια καταλληλότερη λύση είναι ένα σύστημα αντίστοιχο με αυτό του θερμοστάτη. Αυτό σημαίνει ότι ο φωτισμός ελέγχεται με κριτήριο την φωτεινότητα, και όχι την κίνηση. Παρ' όλα αυτά, ούτε αυτή η λύση είναι κατάλληλη, καθώς σε ένα θερμαντικό σώμα έχουμε σταδιακή αλλαγή της εκπεμπόμενης θερμότητας αν τεθεί σε λειτουργία ή εκτός λειτουργίας, ενώ αντιθέτως σε έναν λαμπτήρα όχι. Αυτό σημαίνει ότι δε μπορεί να πραγματοποιείται ρύθμιση για μεγάλο εύρος τιμών έντασης φωτεινότητας, παρά μόνο για μερικές διακριτές και ακραίες μεταξύ τους τιμές.

Το λογικό συμπέρασμα είναι ο συνδυασμός της παραπάνω λύσης με τη δυνατότητα επιλογής του επιπέδου φωτεινότητας κάθε λαμπτήρα, με το τελευταίο να μπορεί εύκολα να πραγματοποιηθεί με χρήση dimmer. Ένα dimmer όμως δεν παρέχει τη δυνατότητα ρύθμισης με αυτόματο τρόπο, αλλά αντιθέτως χειροκίνητα. Η αυτοματοποίηση του dimmer με παράμετρο την φωτεινότητα είναι και η καινοτομία του συστήματος που διερευνάται. Έχοντας ένα επιθυμητό επίπεδο φωτισμού, το σύστημα θα μπορεί να αξιολογεί ανά τακτά χρονικά διαστήματα αν το επίπεδο φωτεινότητας στον χώρο ανταποκρίνεται με το επιθυμητό. Εαν όχι, τότε θα γίνεται η κατάλληλη ρύθμιση έτσι ώστε να φτάσει η φωτεινότητα μέσα σε κάποιο κατάλληλο όριο.

2.1 Απαιτήσεις

Το σύστημα προς ανάπτυξη χρειάζεται να τηρεί κάποιες απαιτήσεις, οι οποίες αφορούν κυρίως τον χρήστη του τελικού προϊόντος, και έχουν να κάνουν με ζητήματα απόδοσης, αισθητικής και λειτουργικότητας. Το σύστημα λοιπόν:

- Θα αποτελεί μια λύση χαμηλού κόστους, εύκολη στην εγκατάσταση και στην παραμετροποίηση (αρχικοποίηση παραμέτρων, εισαγωγή νέων δεδομένων/παραμέτρων, κτλ.).

- Θα έχει ως σκοπό όχι μόνο την εξοικονόμηση ενέργειας, αλλά και την διατήρηση της ομοιομορφίας της έντασης της φωτεινότητας στο μεγαλύτερο μέρος του χώρου που θα τοποθετηθεί. Ενώ ο σκοπός του συστήματος είναι η ελαχιστοποίηση της κατανάλωσης, μια παράμετρος που υπερτίθεται σε αυτή είναι η διατήρηση της αισθητικής του χώρου. Αυτό, για παράδειγμα, σημαίνει ότι το σύστημα θα στοχεύει σε διατήρηση της φωτεινότητας του χώρου σε ένα επίπεδο που θέτει ο χρήστης και, αν αυτό δεν είναι δυνατόν να συμβεί, σε επίπεδο υψηλότερο από αυτό.

- Οι αλλαγές στην ένταση του φωτισμού θα είναι ομαλές και σταδιακές, ώστε να μην είναι άμεσα αντιληπτές από τον χρήστη.

- Θα διαθέτει γραφικό περιβάλλον (Graphical User Interface), μέσω του οποίου ο χρήστης θα μπορεί να παραμετροποιεί το σύστημα (π.χ., επιθυμητή ένταση φωτός, αλλαγή κατάστασης συγκεκριμένου λαμπτήρα ή ομάδας λαμπτήρων), θα έχει πρόσβαση σε πληροφορίες σχετικά με την κατανάλωση αλλά και την εξοικονόμηση ενέργειας (και σε πραγματικό χρόνο αλλά και μέσω ιστορικών δεδομένων), θα ενημερώνεται σχετικά με την εμφάνιση βλαβών (π.χ. λαμπτήρας εκτός λειτουργίας), κτλ.

- Τέλος, το σύστημα θα κατασκευαστεί έτσι ώστε να είναι αυτόματα προσαρμόσιμο σε όσο το δυνατόν περισσότερες παραμέτρους, όπως π.χ. αλλαγή στον αριθμό και στη θέση των αισθητήρων, χωρίς να απαιτείται τροποποίηση στη συνδεσμολογία ή/και στον κώδικα.

- Το σύστημα θα λειτουργεί σε «τοπικό» επίπεδο, χωρίς πρόσβαση στο διαδίκτυο. Η διασύνδεση με το διαδίκτυο (π.χ., αποθήκευση πληροφοριών στο cloud) θα υποστηρίζεται. Στην περίπτωση αυτή θα επιτρέπεται η απομακρυσμένη πρόσβαση, με ασφαλή τρόπο, σε εξουσιοδοτημένους χρήστες με σκοπό την παραμετροποίηση του συστήματος ή/και την αντιμετώπιση/αναίρεση σφαλμάτων

2.2 Χαρακτηριστικά

Τα χαρακτηριστικά του συστήματος αφορούν όχι τόσο στη χρήση του συστήματος, αλλά στις παραμέτρους κατασκευής του. Παρ' όλα αυτά, εξαρτώνται κατά πολύ από τις παραπάνω απαιτήσεις.

- Η εκτέλεση της κύριας λειτουργίας, δηλαδή η προσαρμογή της έντασης των λαμπτήρων ανάλογα με την φωτεινότητα στον χώρο, απαιτεί την ύπαρξη κατάλληλης συσκευής που θα μπορεί να λαμβάνει μετρήσεις. Οι μετρήσεις αυτές θα λαμβάνονται από διάφορα σημεία του χώρου.

- Το σύστημα θα μπορεί να λειτουργήσει σε μια πολύ απλή διάταξη, αλλά και σε μια αρκετά πολύπλοκη. Αυτό σημαίνει ότι θα μπορεί να ανταποκριθεί και στη ύπαρξη πολλών λαμπτήρων και αισθητήρων. Προφανώς δε θα υπάρχει σχέση ένα προς ένα αλλά κάθε μία συσκευή επηρεάζει πολλές από τις άλλες, σε διαφορετικό βαθμό, και θα πρέπει το σύστημα να γνωρίζει αυτό το βαθμό για να λάβει την κατάλληλη απόφαση. Μια εύλογη λύση για αυτό είναι η χρήση μηχανικής μάθησης (supervised machine learning).

- Η ύπαρξη πολλών κόμβων στο σύστημα απαιτεί διευθυνσιοδότηση. Άμεσο συμπέρασμα είναι η ανάγκη για ανάπτυξη ενός πρωτοκόλλου επικοινωνίας, το οποίο θα εξασφαλίζει την ορθή αποστολή και λήψη πληροφοριών, ανεξάρτητα από το μέσο επικοινωνίας.

- Η ανάπτυξη ενός standalone GUI είναι αρκετά δύσκολη και χρονοβόρα, ιδιαίτερα αν ληφθεί υπ' όψη το γεγονός ότι θα πρέπει να είναι σε επικοινωνία με το σύστημα. Ένα web based GUI έχει το ίδιο αποτέλεσμα, και είναι πολύ ευκολότερο να αναπτυχθεί.

- Η απαίτηση να μπορεί το GUI να παρουσιάζει δεδομένα δημιουργεί τις εξής περαιτέρω απαιτήσεις: ύπαρξη βάσης δεδομένων (για ιστορική παρουσίαση), και ύπαρξη μιας δομής τύπου message queuing (για παρουσίαση real time δεδομένων).

Το σύστημα που θα υλοποιήσουμε λειτουργεί ως εξής: αφού απαιτούμε ομοιομορφία στην ένταση φωτός σε όλο τον χώρο που τοποθετείται το σύστημα, χρησιμοποιούμε αισθητήρες φωτός, η ποσότητα των οποίων εξαρτάται από το μέγεθος του χώρου. Οι αισθητήρες παρέχουν στο σύστημα πληροφορίες/μετρήσεις σχετικά με το τρέχον επίπεδο φωτεινότητας. Το σύστημα, μέσα από τον αλγόριθμο που περιγράφεται στα κεφάλαια 4 και 5, αποφασίζει σε πραγματικό χρόνο για το αν θα χρειαστεί αύξηση ή μείωση του φωτισμού, σε ποιά σημεία, καθώς και ποιών λαμπτήρων η ένταση θα αλλάξει για να επιτευχθεί η ομοιομορφία, χωρίς όμως να γίνει αυτή η αλλαγή άμεσα αντιληπτή από τους χρήστες. Η απόφαση του συστήματος εξαρτάται από ένα επιθυμητό επίπεδο έντασης, το οποίο δίνεται από τον χρήστη-administrator, και μπορεί ανά πάσα στιγμή να αλλάξει η τιμή του.

Για να αποσαφηνιστούν κάποιες έννοιες, καθώς και να δοθούν κάποιες παράμετροι του συστήματος, περιγράφουμε παρακάτω κάποια δομικά στοιχεία της αρχιτεκτονικής, καθώς και τις παρεχόμενες υπηρεσίες. Τέλος, θα γίνει και μια εκτίμηση σχετικά με το κόστος υλοποίησης, το κόστος λειτουργίας, καθώς και τα οικονομικά οφέλη της εγκατάστασης.

2.3 Παρεχόμενες υπηρεσίες

Με σκοπό τη βελτίωση της εμπειρίας του χρήστη, καθώς και τη διευκόλυνση της χρήσης του, το σύστημα παρέχει, εκτός από την εξασφάλιση ελάχιστης κατανάλωσης, ένα σύνολο υπηρεσιών.

2.3.1 WebGUI

Μέσω ενός γραφικού περιβάλλοντος (Graphical User Interface - GUI) ο χρήστης μπορεί να επικοινωνεί με το σύστημα, να το παραμετροποιεί, καθώς και να παρακολουθεί τη λειτουργία του.

Πρώτα από όλα, είναι διαθέσιμος στον χρήστη ένας πίνακας ελέγχου του συστήματος. Μέσω αυτού, ο χρήστης μπορεί να επιλέξει ένα επιθυμητό επίπεδο έντασης φωτισμού, στο οποίο θα προσαρμοστούν οι λαμπτήρες. Το σύστημα αποθηκεύει την επιλογή και την εφαρμόζει σε κάθε εκκίνηση του, μέχρι ο χρήστης να την αλλάξει. Επίσης, υπάρχει η δυνατότητα εναλλαγής της λειτουργίας του συστήματος από αυτόματο σε καθορισμένο από τον χρήστη. Έτσι, δίνεται στον χρήστη η δυνατότητα να καθορίζει χειροκίνητα την τιμή της έντασης των λαμπτήρων που ελέγχονται από το σύστημα.

Επιπρόσθετα, το GUI παρέχει τις πληροφορίες σχετικά με την κατανάλωση. Υπάρχει η δυνατότητα εμφάνισης, σε μορφή γραφήματος, της κατανάλωσης σε έκταση ώρας, μέρας, εβδομάδας, μήνα και χρόνου για κάθε λαμπτήρα, σύγκρισης της κατανάλωσης ενέργειας σε σχέση με την κατανάλωση που θα υπήρχε χωρίς το σύστημα, καθώς και εκτίμηση για την οικονομία που έχει γίνει σε χρηματικές μονάδες. Το γράφημα μπορεί να εξαχθεί σε αρχείο pdf ή εικόνας (png, jpeg, svg). Επιπλέον είναι διαθέσιμο και γράφημα της κατανάλωσης σε πραγματικό χρόνο, το οποίο ανανεώνεται αυτόματα με την άφιξη νέας μέτρησης. Επιπρόσθετα, το γράφημα δίνει τη δυνατότητα εστίασης στην παροχή ενός λαμπτήρα, καθώς και σε κάθε μέτρηση ξεχωριστά, με πληροφορίες για το επίπεδο κατανάλωσης και την ημέρα και ώρα λήψης της μέτρησης.

2.3.2 Επιπρόσθετες υπηρεσίες

Η υλοποίηση του συστήματος περιλαμβάνει και κάποιες υπηρεσίες που δεν είναι ορατές από τον χρήστη, αλλά είναι απαραίτητες για την εξασφάλιση σωστής λειτουργίας.

Προσαρμοστικότητα

Η προσαρμοστικότητα του συστήματος αναφέρεται στο γεγονός ότι αυτό θα λειτουργεί με τον ίδιο τρόπο, ανεξάρτητα από την τοπολογία του και τους εξωτερικούς παράγοντες, όπως π.χ. τα έπιπλα ή τα διακοσμητικά αντικείμενα του χώρου. Η υπηρεσία αυτή είναι ζωτικής σημασίας, καθώς είτε η μετακίνηση ενός αισθητήρα φωτεινότητας, είτε η τοποθέτηση στον χώρο αντικειμένων που μπορεί να επηρεάζουν τις μετρήσεις των αισθητήρων, μπορεί να προκαλέσει προβλήματα στη λειτουργία του, αν έχουμε μια συγκεκριμένη προεγκατεστημένη ρύθμιση. Το σύστημα έχει αναπτυχθεί έτσι ώστε, σε κάθε εκκίνησή του, υπάρχει ένα στάδιο προρύθμισης, κατά το οποίο συλλέγουμε μια σειρά μετρήσεων, και "κατασκευάζουμε" βάρη, τα οποία υποδηλώνουν την εξάρτηση ενός αισθητήρα από κάθε λαμπτήρα. Μια αλλαγή στην τοπολογία του συστήματος προκαλεί μεταβολή στα βάρη, η οποία διασφαλίζει την ορθή λειτουργία του συστήματος.

Αλγόριθμος επιλογής φωτισμού

Το σύστημά μας έχει αναπτυχθεί κυρίως για χώρους γραφείων, όπου η κατανάλωση είναι πολλαπλάσια από έναν χώρο κατοικίας, και απαιτείται μείωσή της. Το γεγονός ότι ο χρήστης θα είναι, κατά πάσα πιθανότητα, κάποιος που θα εργάζεται πολλές ώρες μπροστά από οθόνη υπολογιστή, πρέπει να διασφαλίζεται ότι η λειτουργία του συστήματος δε θα έχει καμιά επίδραση στην αποδοτικότητα το εργαζόμενου, και ακόμα περισσότερο ότι δε θα γίνεται αντιληπτή με κανένα τρόπο. Γι'αυτό και υλοποιήθηκε ένας αλγόριθμος επιλογής φωτισμού ο οποίος εξασφαλίζει ομαλή ρύθμιση της έντασης σε μια χρονική περίοδο κατά την οποία, όσο μεγάλο και να είναι το εύρος, η αλλαγές του φωτισμού γίνονται όσο το δυνατόν λιγότερο αισθητές. Επιπλέον, ο αλγόριθμος απορρίπτει μικρές αλλαγές στον φωτισμό, οι οποίες θα κόστιζαν σε ισχύ λειτουργίας του συστήματος και θα είχαν μικρή επίδραση στην κατανάλωση από τον λαμπτήρα.

Απομακρυσμένη διαχείριση

Λόγω της πολυπλοκότητάς του, το σύστημα ενδέχεται να παρουσιάσει κάποια βλάβη σε επίπεδο λογισμικού. Γι'αυτό το λόγο έχει αναπτυχθεί έτσι ώστε το μεγαλύτερο μέρος του να είναι προσβάσιμο από κάποιον υπολογιστή εκτός του τοπικού δικτύου, για να γίνονται οι απαραίτητες ρυθμίσεις από κάποιον ειδικό. Προφανώς και παρέχεται η απαραίτητη ασφάλεια, ούτως ώστε να μην είναι το σύστημα επιρρεπές σε διαδικτυακές επιθέσεις.

2.4 Σύνοψη

Βάσει των απαιτήσεων που θέσαμε, καθώς και των επιθυμητών υπηρεσιών που θα παρέχονται, στο πλαίσιο της διπλωματικής εργασίας, πρέπει να υπλοποιήσουμε τα εξής, σε επίπεδο hardware:

- Μια συσκευή μέτρησης της έντασης φωτός, μικρού μεγέθους και χαμηλής κατανάλωσης, με δυνατότητα να αποστέλλει πληροφορίες ενσύρματα ή ασύρματα, ανάλογα με την

υλοποίηση.

- Μια συσκευή μέτρησης της κατανάλωσης, με τα ίδια χαρακτηριστικά.

- Ένα dimmer της τεχνολογίας που αναλύθηκε στο προηγούμενο κεφάλαιο, το οποίο θα έχει επίσης τη δυνατότητα να ανταλλάσσει πληροφορίες με άλλες συσκευές.

- Ένα gateway, το οποίο συγκεντρώνει και αποθηκεύει όλες τις απαραίτητες πληροφορίες, και είναι υπεύθυνο για την προσαρμογή των χαρακτηριστικών του συστήματος.

Και τα εξής σε επίπεδο software:

- Έναν αλγόριθμο που θα συγκεντρώνει τις μετρήσεις, και με κατάλληλη επεξεργασία τους θα εξασφαλίζει την επιθυμητή ένταση φωτισμού, με όλα τα χαρακτηριστικά που προαναφέρθηκαν.

- Ένα πρωτόκολλο επικοινωνίας μεταξύ των κόμβων του συστήματος, που θα επιτρέπει τη μεταφορά της πληροφορίας, καθώς και κατάλληλα διαμορφωμένη επικεφαλίδα με πληροφορίες για τη διεύθυνση του αποστολέα, του παραλήπτη, του τύπου πακέτου που αποστέλλεται και άλλα.

- Μια υποδομή αποθήκευσης και παρουσίασης πληροφοριών σχετικές με την κατανάλωση, καθώς και επεξεργασίας εντολών εισόδου από τον χρήστη (WebGUI).

Στο επόμενο κεφάλαιο θα αναλυθούν και θα παρουσιαστούν συγκριτικά οι επιλογές για κάθε υλικό του συστήματος που προέκυψαν κατά την έρευνα αγοράς, καθώς το υλικό που τελικά επιλέξαμε.

Κεφάλαιο 3 Έρευνα αγοράς

Σε αυτό το κεφάλαιο θα παρουσιαστεί αναλυτικά η διαδικασία της έρευνας αγοράς που πραγματοποιήθηκε, οργανωμένη σε ενότητες, ανάλογα με τη χρήση του κάθε component. Στο τέλος θα γίνει η δικαιολόγηση της τελικής επιλογής κάθε εξαρτήματος.

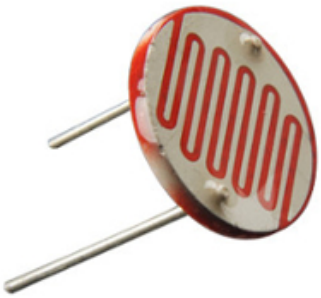
3.1 Συσκευές μέτρησης

Οι συσκευές που θα κατασκευάσουμε για τη μέτρηση της έντασης φωτεινότητας και για τη μέτρηση της κατανάλωσης σε κάθε λαμπτήρα, θέλουμε να περιέχουν τα εξής:

- Έναν κατάλληλο αισθητήρα, που θα παρέχει μια κατάλληλη τιμή σχετικά με το επίπεδο έντασης φωτός ή την κατανάλωση.
- Μια συσκευή που θα λαμβάνει τη μέτρηση και θα τη μεταφράζει, προσαρμόζοντάς στην αντίστοιχη μονάδα μέτρησης για να γίνει κατανοητή στον χρήστη. Η συσκευή αυτή λοιπόν πρέπει να μπορεί να προγραμματιστεί για να εκτελεί έναν συγκεκριμένο αλγόριθμο όποτε λαμβάνει μια μέτρηση.
- Τη διεπαφή που θα επιτρέπει την αποστολή αυτής της μέτρησης, η οποία επίσης συνδέεται στην προαναφερθείσα συσκευή.

3.1.1 Οι αισθητήρες

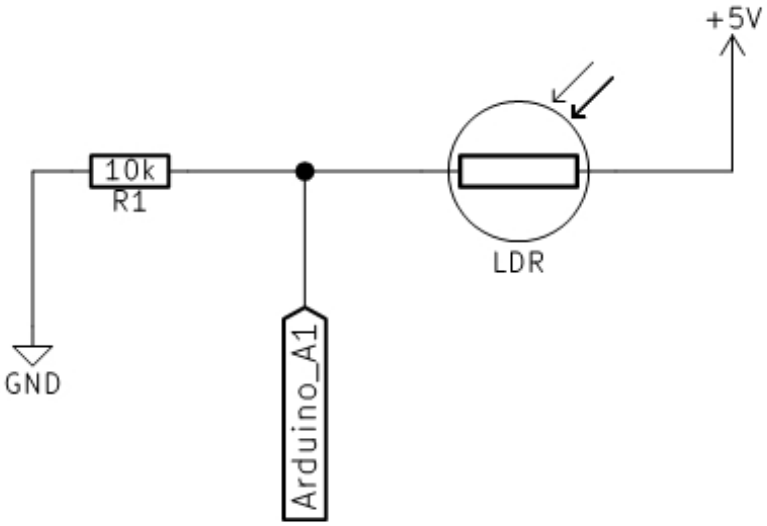
3.1.1.1 Αισθητήρας φωτεινότητας



3.1 - LDR

Για την υλοποίηση του αισθητήρα φωτεινότητας, η λύση που μπορεί πολύ εύκολα να εφαρμοσθεί στο σύστημά μας είναι ένα light dependent resistor (LDR). Το LDR είναι μια αντίσταση με γραμμική εξάρτηση από το φως του περιβάλλοντος. Το κύκλωμα εφαρμογής φαίνεται στην εικόνα 3.16.

Όταν το φως περιβάλλοντος αυξηθεί, η αντίσταση της φωτοδιόδου μειώνεται, και ως αποτέλεσμα έχουμε αύξηση της τάσης στην έξοδο που λαμβάνεται η μέτρηση. Όταν μειώνεται το φως, συμβαίνει το αντίθετο. Στην αγορά κυκλοφορούν και έτοιμες πλακέτες, που περιλαμβάνουν το LDR, την αντίσταση του κυκλώματος και τα τερματικά pins.



Εικόνα 3.2 - Κύκλωμα προσαρμογής LDR

3.1.1.2 Μετρητής κατανάλωσης

Για τη μέτρηση την κατανάλωσης σε κάθε λαμπτήρα, βρέθηκαν στην αγορά οι εξής επιλογές:

Μετρητής τοίχου

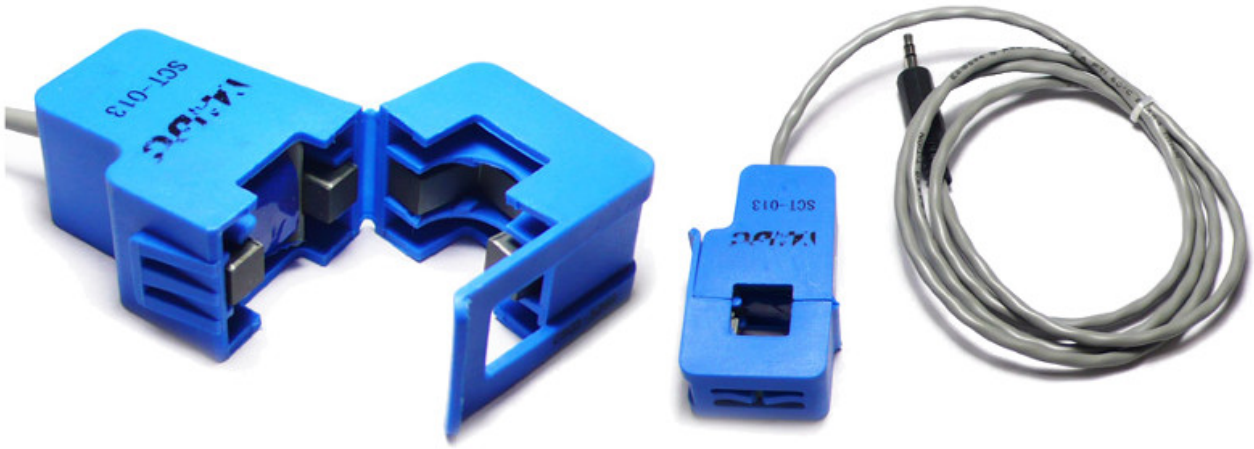
Ο μετρητής τοίχου είναι μια απλή συσκευή που συνδέεται σε υποδοχή πρίζας, και έπειτα παρέχει ανάγνωση για την κατανάλωση όσων συσκευών συνδέονται πάνω της. Υπάρχουν προϊόντα που παρέχουν πληροφορίες για την τάση, την ένταση του ρεύματος, τη συχνότητα και την κατανάλωση σε Watt και Wh σε ενσωματωμένη οθόνη. Με κατάλληλη διαμόρφωση υπάρχει η δυνατότητα να προσαρμοσθεί στη γραμμή τροφοδοσίας του λαμπτήρα, καθώς και να προσαρμοσθεί μικροελεγκτής, ο οποίος στέλνει τις μετρήσεις στο κεντρικό gateway.

Power clamp

Εικόνα 3.3 - Power Clamp

Όπως και η συσκευή τοίχου, το power clamp παρέχει πληροφορίες για την τάση, την ένταση του ρεύματος, την κατανάλωση και τη συχνότητα. Η διαφορά βρίσκεται στο ότι η συγκεκριμένη συσκευή έχει μορφή τσιμπίδας και προσαρμόζεται έτσι ώστε το καλώδιο που τροφοδοτεί μια συσκευή με ρεύμα να περνά από μέσα της. Επιπλέον, παρέχει θύρα USB, την οποία μπορούμε να συνδέσουμε σε υπολογιστή αλλά και σε μικροελεγκτή, με το κατάλληλο USB shield και κώδικα.

Μετασχηματιστής ρεύματος



Εικόνα 3.4 - Μετασχηματιστής ρεύματος (SCT-013)

Ο μετασχηματιστής ρεύματος που βρήκαμε στην αγορά έχει τη μορφή ενός power clamp σε μικρότερο μέγεθος. Λειτουργεί μετασχηματίζοντας το ρεύμα του καλωδίου που περικλείει σε χαμηλή τιμή ρεύματος ή τάσης. Η έξοδός του μπορεί να τροφοδοτηθεί σε έναν μικροελεγκτή μέσω ενός μικρού κατάλληλου κυκλώματος, το οποίο φέρνει την έξοδο σε τιμές κατάλληλες για τον μικροελεγκτή. Παρ' όλο που οι μετρήσεις που παρέχει δεν είναι απόλυτα ακριβείς, το πλεονέκτημά του, σε σχέση με τις προηγούμενες επιλογές, είναι ότι, εκτός από το μικρό του μέγεθος, τιμή και εύκολη προσαρμογή σε κύκλωμα, παρέχει συνεχείς μετρήσεις, και μπορούμε να τις αντλούμε ανά πάσα στιγμή.

Product	Type	Specs	Price (€)
Kill A Watt 4460	Applied directly to power socket	15A, 125V, Power measurement	29.00
MD 9240 TRMS	Power Clamp	1000A, 600V, Power measurement, USB	182.04
MD 9220 TRMS	Power Clamp	2000A, 600V	118.08
MMD9225 TRMS AC/DC	Power Clamp	400A, 600V	177.12
MD 9210 Mini	Power Clamp	600A, 600V	100.86
Extech PQ2071	Power Clamp	1000A, 750V, Power measurement, USB	199.26
Extech PX720	Power Clamp	800A, 600V, Power measurement	110.70
YHDC SCT013-000	Current Transformer	100A:50mA	14.37
YHDC SCT013-030	Current Transformer	30A/1V	13.47
YHDC HST21	Current Transformer	500A/4V	17.47
Sparkfun SEN-11005	Current Transformer	30A:15mA	12.36
HOBUT Clip Fit	Current Transformer	100A:5A	57.81

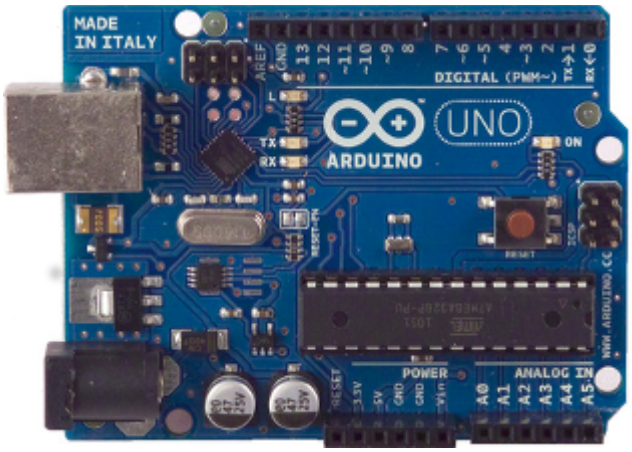
Πίνακας 3.3 - Σύγκριση χαρακτηριστικών συσκευών μέτρησης κατανάλωσης

3.1.2 Ο μικροελεγκτής

Με την άνοδο των εφαρμογών Internet of things (IoT) και home automation, έχουν έρθει στο προσκήνιο πολλοί τύποι μικροελεγκτών, καθώς και έτοιμες πλακέτες με in-out συνδέσεις και usb συνδεσιμότητα, που καθιστούν τη χρήση τους πολύ εύκολη από κάποιον με βασικές γνώσεις προγραμματισμού, και οι περισσότερες σε προσιτές τιμές. Θα γίνει μια συνοπτική παρουσίαση των πλακετών που βρέθηκαν κατά την έρευνα αγοράς, καθώς και ένας συγκριτικός πίνακας με τα χαρακτηριστικά τους.

Arduino

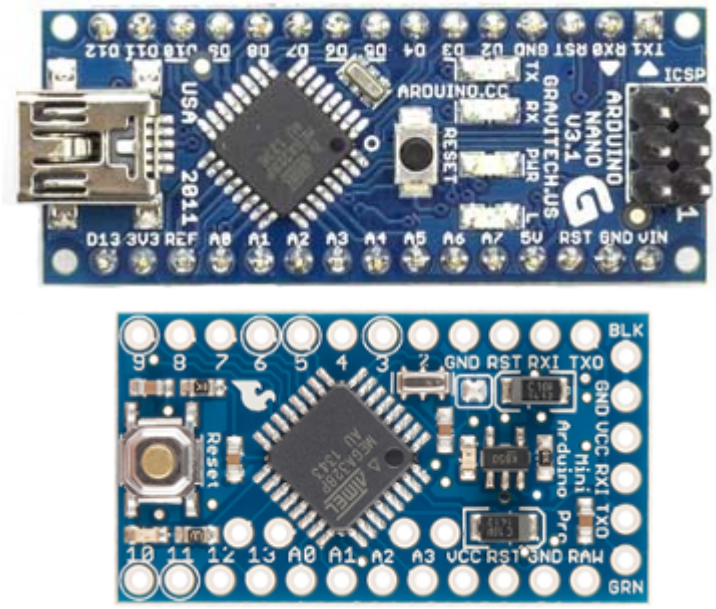
Το Arduino είναι μια open source μητρική πλακέτα που περιλαμβάνει έναν μικροελεγκτή με τις απαραίτητες αντιστάσεις και πυκνωτές, τον κρύσταλλο (xtal), καθώς και τα in-out pins σε έτοιμες συνδέσεις. Επίσης παρέχεται on-board ο μετατροπέας USB-Serial για τον προγραμματισμό μέσω υπολογιστή, και voltage regulators για 5 και 3.3 V για την παροχή τάσης σε περιφερειακά ή την τροφοδοσία από εξωτερική πηγή. Το Arduino καθιστά πολύ εύκολη τη σύνδεση περιφερειακών όπως αισθητήρες, οθόνες κτλ, καθώς και τον προγραμματισμό του μικροελεγκτή, μέσω του Arduino IDE, σε κώδικα C++. Το Arduino είναι πλέον η κυρίαρχη και πρώτη σε πωλήσεις πλακέτα μικροελεγκτή, και ως αποτέλεσμα το support που παρέχεται μέσω internet είναι αρκετό για την ανάπτυξη πολύπλοκων projects και την αντιμετώπιση προβλημάτων. Για τη διασύνδεσή του με περιφερειακά, χρησιμοποιεί βιβλιοθήκες έτοιμου κώδικα που παρέχονται από τον κατασκευαστή του εκάστοτε περιφερειακού. Στη συνέχεια παρουσιάζουμε τα κυριότερα μοντέλα που υπάρχουν στην αγορά:



Εικόνα 3.5 - Arduino UNO

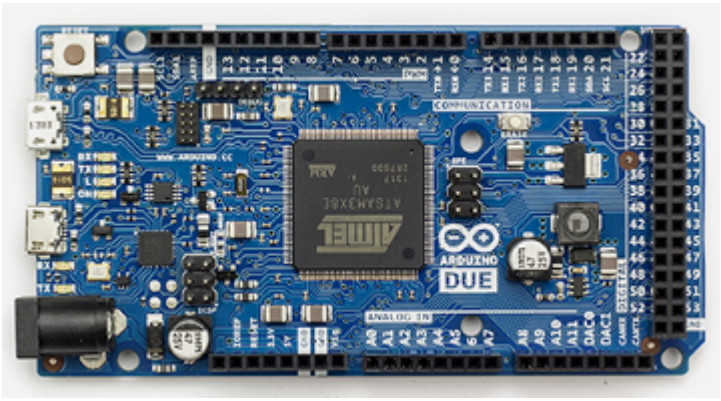
Το UNO είναι η entry-level πλακέτα Arduino. Για τον προγραμματισμό και την τροφοδοσία του από υπολογιστή χρησιμοποιεί θύρα USB Type B. Μπορεί να τροφοδοτηθεί και μέσω του Vin pin ή του barrel jack adapter που περιλαμβάνει, με τάση 7-12 V, χάρη στο on-board 5V regulator. Παρέχει 20 pins, εκ των οποίων τα 6 είναι αναλογικά PWM, και χρησιμοποιεί τον μικροελεγκτή Atmega328p.

Το Nano είναι κατά βάση μια μικρότερη έκδοση του UNO. Έχει τον ίδιο αριθμό pins, όρια τροφοδοσίας 7-9V, συνδέεται σε υπολογιστή για τροφοδοσία-προγραμματισμό μέσω θύρας, USB Mini, και χρησιμοποιεί τον μικροελεγκτή Atmega328p ή τον Atmega168 σε κάποια μοντέλα.



Εικόνα 3.6 - Arduino Nano, Arduino Pro Mini

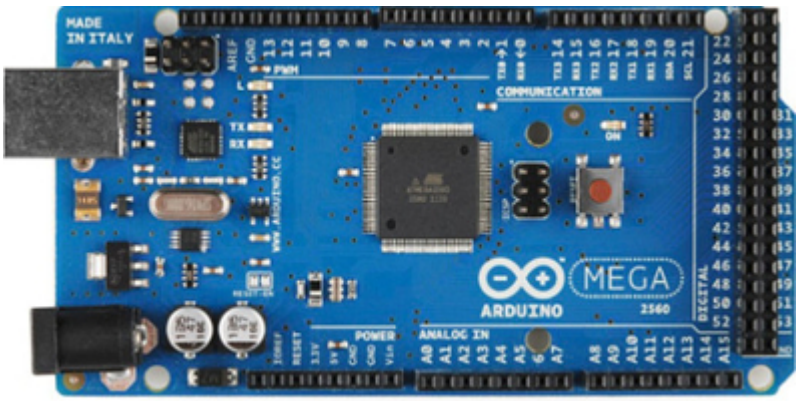
Το Pro Mini είναι μια ακόμα μικρότερη έκδοση παρόμοια με το UNO. Διατίθεται σε εκδόσεις 3.3 και 5V, καθώς και 8 ή 16 MHz. Για να χρησιμοποιηθεί σε project, χρειάζεται η κόλληση τερματικών, θηλυκών ή αρσενικών, στις συνδέσεις των pins. Επιπλέον, δεν παρέχει θύρα USB ούτε ολοκληρωμένο USB-Serial, και ο προγραμματισμός του γίνεται μέσω εξωτερικού USB-Serial ή μέσω άλλης συσκευής Arduino. Τροφοδοτείται μέσω σταθεροποιημένης τάσης 3.3 ή 5V από το Vin pin, ή μέσω τάσης 7-12V μέσω του RAW pin.



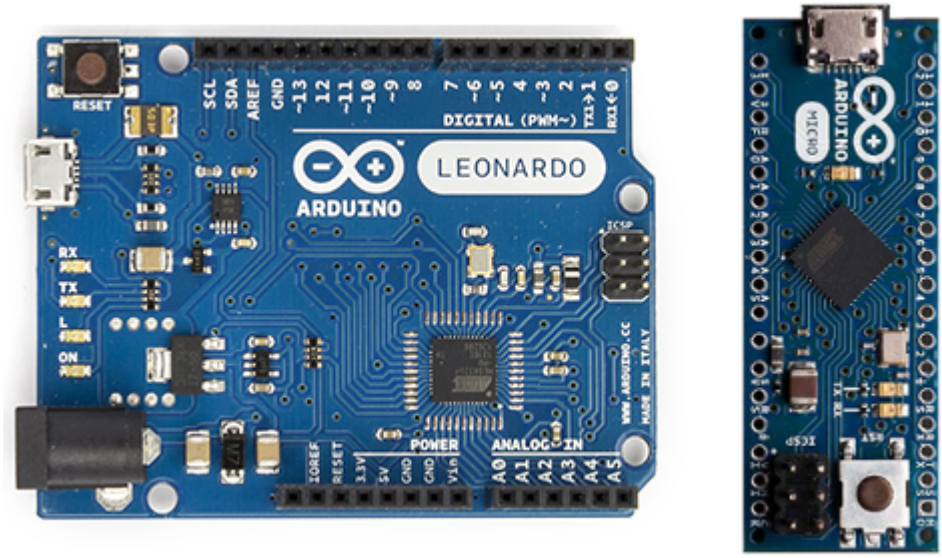
Εικόνα 3.7 - Arduino Due

Το Due είναι σε πολλά χαρακτηριστικά διαφορετικό από τα προηγούμενα μοντέλα. Χρησιμοποιεί μικροελεγκτή Atmega SAM3X8E, που είναι βασισμένος σε επεξεργαστή 32-bit ARM core και λειτουργεί στα 3.3V. Περιλαμβάνει 54 digital IO pins, εκ των οποίων τα 12 μπορούν να χρησιμοποιηθούν σαν αναλογικά PWM και 12 σαν αναλογικά inputs, και λειτουργεί στα 84 MHz. Επιπλέον, παρέχει συνδεσιμότητα USB OTG.

Το Mega 2560 βασίζεται στον μικροελεγκτή Atmega2560. Περιλαμβάνει 54 digital IO pins, εκ των οποίων τα 15 μπορούν να χρησιμοποιηθούν σαν αναλογικά PWM και 16 σαν αναλογικά inputs. Όπως και το UNO, μπορεί να τροφοδοτηθεί μέσω barrel jack με τάση 7-12V, ή μέσω USB Type B.



Εικόνα 3.8 - Arduino Mega 2560



Εικόνα 3.9 - Arduino Leonardo, Arduino Micro

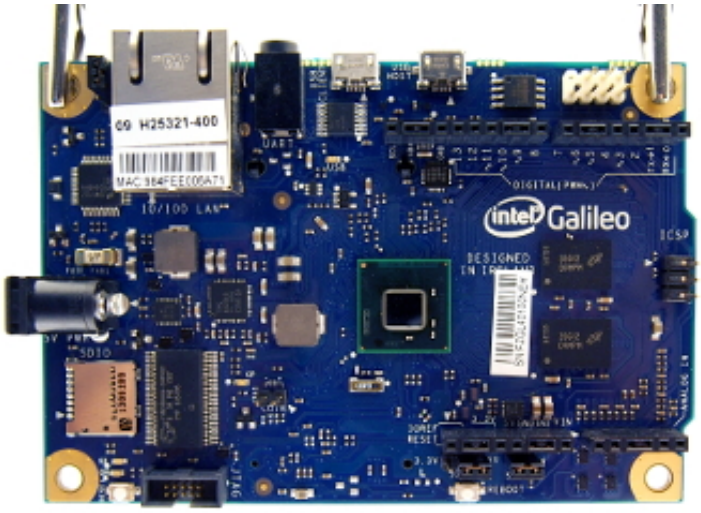
Το Leonardo βασίζεται στον μικροελεγκτή Atmega32u4, ο οποίος ενσωματώνει τη δυνατότητα μετατροπής USB-Serial. Περιλαμβάνει 20 digital IO pins, εκ των οποίων τα 7 μπορούν να χρησιμοποιηθούν σαν αναλογικά PWM και 12 σαν αναλογικά inputs. Τροφοδοτείται μέσω barrel jack στα 7-12V ή μέσω USB Micro.

Το Micro έχει τα ίδια χαρακτηριστικά με το Leonardo, σε μικρότερο μέγεθος, και χωρίς το barrel jack connector.

Το Yun είναι μια συσκευή Arduino που επίσης διαφέρει πολύ από τις προηγούμενες. Βασίζεται στον μικροελεγκτή Atmega 32u4, αλλά περιλαμβάνει και το chip Atheros AR9331, έναν επεξεργαστή που δίνει στο Yun τη δυνατότητα να τρέχει μια διανομή Linux βασισμένη στην OpenWrt, την OpenWrt-Yun. Έχει ενσωματωμένη θύρα ethernet και υποστήριξη WiFi, μια θύρα USB Type A, υποδοχή micro-SD και θύρα micro USB για προγραμματισμό του μικροελεγκτή.

Εικόνα 3.10 - Arduino Yun

Intel Galileo



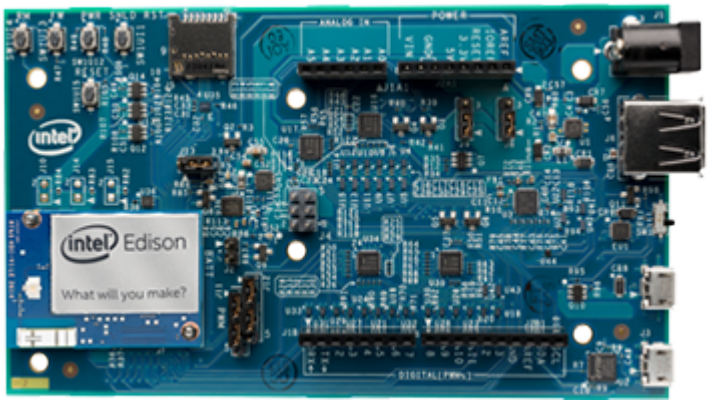
Εικόνα 3.11 - Intel Galileo

Το Galileo είναι μια μητρική πλακέτα πιστοποιημένη από την εταιρία Arduino, και βασίζεται στην αρχιτεκτονική x86 της Intel. Αναπτύχθηκε για να προσφέρει έτοιμη λύση σε εφαρμογές IoT, χάρη στην out of the box δυνατότητα σύνδεσης στο internet. Παρέχει πλήρη υποστήριξη των περιφερειακών και των επεκτάσεων (shields) του Arduino, και λειτουργεί με λογισμικό Yocto Linux. Η διαφορά του, πέρα από τον επεξεργαστή, είναι ότι παρέχει εσωτερικό αποθηκευτικό χώρο, καθώς και συνδεσιμότητα μέσω ethernet, σε αντίθεση με το Arduino, το οποίο χρειάζεται επέκταση. Επιπλέον, μπορεί να λειτουργήσει στα 3.3 ή στα 5V χάρη σε ενσωματωμένο διακόπτη, και παρέχει επέκταση με κάρτα μνήμης, υποδοχή PCI, θύρα USB host για σύνδεση συσκευών, θύρα USB client για προγραμματισμό και ενσωματωμένη μνήμη Flash 8MB τύπου NOR. Λόγω του ότι παρέχει εσωτερική μνήμη, μπορεί να διαβάσει κώδικα που είναι αποθηκευμένος σε αυτή, και όχι μόνο από υπολογιστή.

Intel Edison

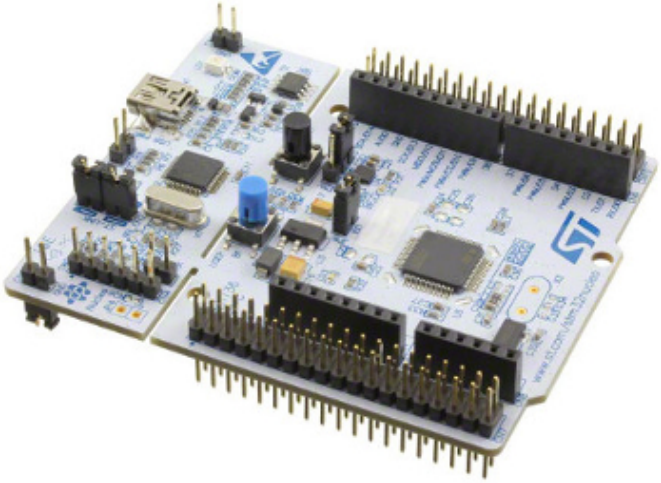
Όπως και το Galileo, το Edison ανήκει στη νέα γενιά των SoC (System on a Chip). Λειτουργεί με επεξεργαστή Intel Atom, και τρέχει λογισμικό Yocto Linux. Το αρχικό module είναι μια μητρική

μικρού μεγέθους, η οποία όμως χρησιμοποιείται είτε μέσω ειδικά σχεδιασμένου breakout board, είναι μέσω του Arduino Kit, που την καθιστά συμβατή με όλα τα shields του Arduino. Μερικά χαρακτηριστικά είναι ότι περιλαμβάνει WiFi, Bluetooth, θύρα USB host, υποδοχή κάρτας SD, barrel jack για τροφοδοσία και 4GB ενσωματωμένη μνήμη. Μπορεί να προγραμματιστεί σε C++, Python, Nodejs και HTML5. Χάρη στην out-of-the-box δυνατότητα σύνδεσης στο internet, το Edison αποτελεί πολύ καλό εργαλείο για εφαρμογές IoT.



Εικόνα 3.12 - Intel Edison

STM Nucleo



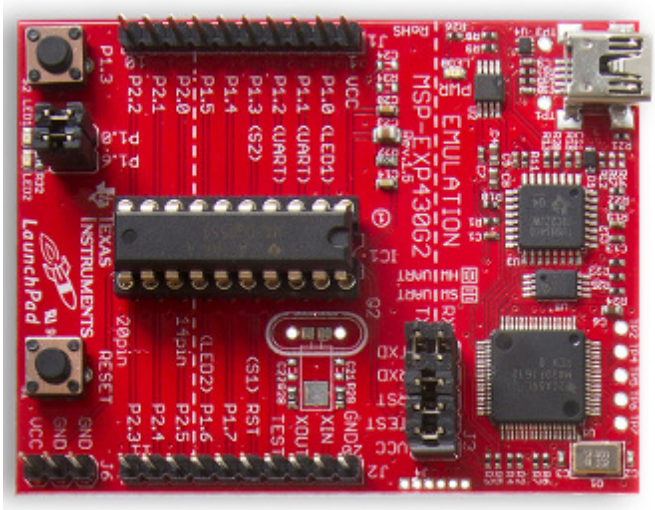
Εικόνα 3.13 - STM Nucleo

Το Nucleo είναι μια ακόμα μητρική πλακέτα, η οποία χρησιμοποιεί μικροελεγκτή STM32. Παρ'όλο που παρέχει περισσότερα in-out pins από τα προηγούμενα μοντέλα που αναλύσαμε, έχει και αυτό τη δυνατότητα διασύνδεσης με Arduino shields. Η θύρα USB που παρέχει, δίνει τη δυνατότητα σύνδεσης σε υπολογιστή για προγραμματισμό, σύνδεσης εξωτερικής συσκευής αποθήκευσης, είτε σύνδεσης για λειτουργία debug. Τέλος, λειτουργεί στα 3.3 ή στα 5V, και μπορεί να τροφοδοτηθεί με τάση 7-12V.

TI MSP-EXP430G2 Launchpad

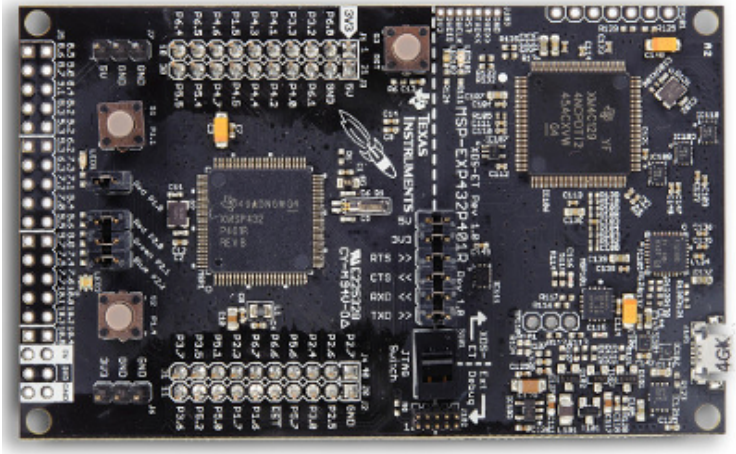
Το MSP430 Launchpad της εταιρίας Texas Instruments είναι μια μητρική πλακέτα βασισμένη στον μικροελεγκτή TI MSP430G2553. Λειτουργεί στα 16 MHz, έχει εσωτερική μνήμη Flash 16KB και 512B RAM. Παρέχει 16 digital pins, εκ των οποίων τα 7 μπορούν να χρησιμοποιηθούν σαν

analog in και τα 8 έχουν δυνατότητα PWM. Τα σημεία στα οποία πλεονεκτεί είναι η πολύ χαμηλή τιμή του (εντος US) και η πολύ μικρή κατανάλωση.



Εικόνα 3.14 - TI MSP-EXP430G2 Launchpad

TI MSP-EXP432P401R Launchpad



Εικόνα 3.15 - TI MSP-EXP432P401R Launchpad

Άλλο ένα μοντέλο της σειράς launchpad, το MSP-EXP432P401R, προσφέρει επεξεργαστή 32-bit ARM Cortex M4F στα 48 MHz, 256 KB μνήμη FLASH, 64 KB RAM, μετατροπέα ADC 24-bit 12 καναλιών, και εξίσου χαμηλή τιμή με το προηγούμενο μοντέλο.

Στον πίνακα 3.1 παρουσιάζονται όλα τα χαρακτηριστικά των μικροελεγκτών που αναλύθηκαν.

Το αμέσως επόμενο στοιχείο που θα παρουσιάσουμε είναι η συσκευή που θα χρησιμοποιηθεί ως κεντρικό gateway.

Name	Processor	Clock speed	Digital pins	PWM pins	Analog i/o pins	FLASH (KB)	RAM (KB)	EEPROM (KB)	Price (€)
Arduino UNO	Atmega328p	16 MHz	14	6	6/0	32	2	1	27.64
Arduino Nano	Atmega328p	16 MHz	14	6	6/0	16	1	0.5	34.20
	Atmega168								
Arduino Pro Mini	Atmega328p	16 MHz or 8 MHz	14	6	6/0	32	2	1	12.36
	Atmega168					16	1	0.5	
Arduino Due	Atmega SAM3X8E	84 MHz	54	12	12/2	512	2.5	-	50.55
Arduino Mega 2560	Atmega2560	16 MHz	54	15	16/0	256	8	4	61.79
Arduino Leonardo	Atmega32u4	16 MHz	20	7	12/0	32	2.5	1	25.28
Arduino Micro	Atmega32u4	16 MHz	20	7	12/0	32	2.5	1	25.28
Arduino Yun	Atmega32u4, AR331 Linux	16 MHz 400 MHz	20	7	12/0	32 64 MB	2.5 16 MB	1 -	89.89
Intel Galileo	Intel® Quark SoC X1000	400 MHz	14	6	6/0	8 MB	512	-	79.95
Intel Edison	Intel® Atom 500MHz CPU, Intel® Quark 100MHz MC	500 MHz 100 MHz	20	4	6/0	4 GB	1 GB	-	134.83[1]
STM Nucleo	STM32	72 MHz	14 or 50 GPIO	6	6/0	128	20	-	10.54
TI MSP430 EXP430G2 Launchpad	MSP430G2553	16 MHz	20 GPIO	With timer	8/-	16	0.5		14.95
TI MSP430 EXP432P401R Launchpad	MSP432P401R 32-bit ARM Cortex M4F	48 MHz	40 GPIO	With timer	24/-	256	64	-	14.95

[1]Price for the Intel Edison with the Arduino Breakout Kit

Πίνακας 3.1 - Σύγκριση χαρακτηριστικών των μικροελεγκτών

3.2 To gateway

Το gateway πρέπει να είναι μια συσκευή που θα παρέχει δυνατότητες υπολογιστή, όπως για παράδειγμα να υποστηρίζει ενσωμάτωση web server με τοπική βάση δεδομένων και server side scripting, απομακρυσμένη διαχείριση, FTP και άλλα. Επειδή η λύση που προτείνεται για το σύστημα προς ανάπτυξη θέλουμε να είναι οικονομική (σε τιμή αγοράς αλλά και σε κατανάλωση) αλλά και μικρή σε μέγεθος, θα αποκλείσουμε την χρήση κανονικού υπολογιστή, όπως desktop ή laptop, λόγω του ότι η ενέργεια που θα καταναλώνει δεν ανταποκρίνεται στον σκοπό (εξοικονόμηση ενέργειας). Αντ' αυτού, θα αναλύσουμε κάποιες συσκευές SoC, που παρέχουν παρόμοιες δυνατότητες σε μικρότερο μέγεθος και τιμή.

Raspberry Pi

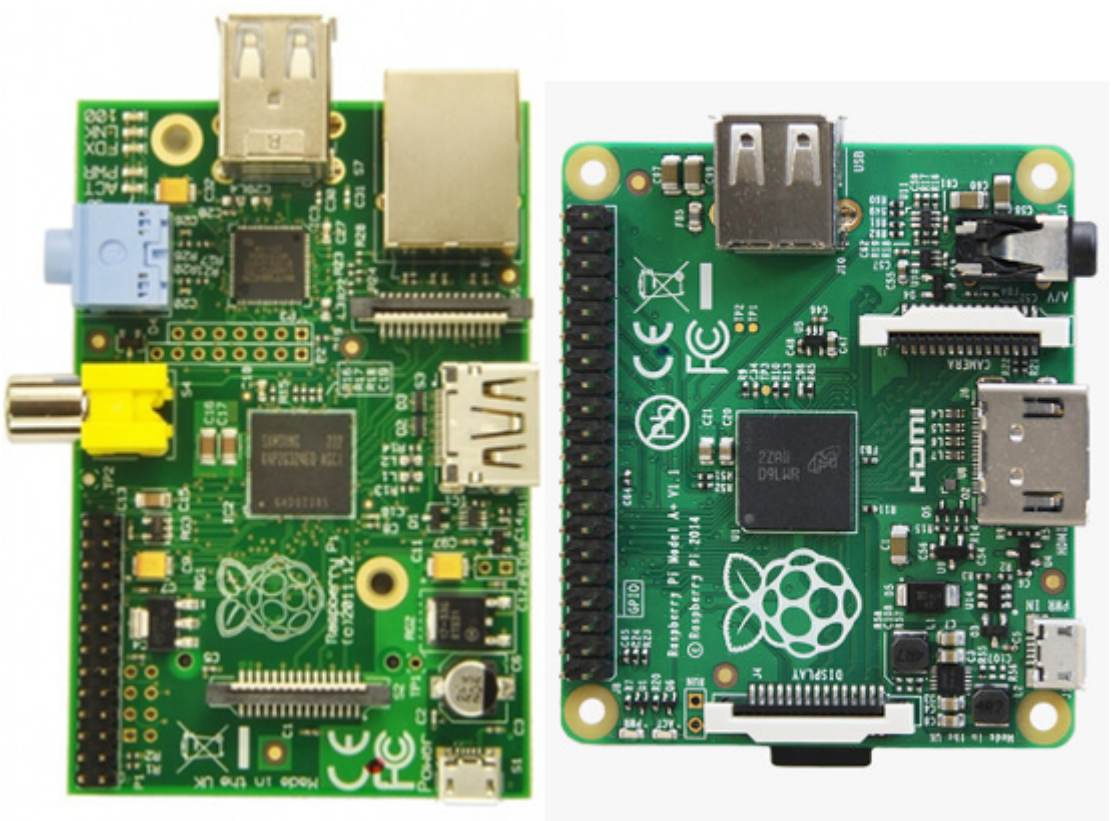
Το Raspberry Pi (RPi), είναι ένας μικρός υπολογιστής σε μορφή SoC, που έχει αναπτυχθεί από

την εταιρία Raspberry Pi Foundation, με κύριο σκοπό την εκμάθηση επιστήμης υπολογιστών και προγραμματισμού σε σχολεία. Βέβαια, λόγω των δυνατοτήτων του, και λόγω του ότι παρέχει έτοιμα in-out pins, όπως και το Arduino, το καθιστά πολύ χρήσιμο σε εφαρμογές IoT. Το πρώτο μοντέλο, Raspberry Pi 1 Model A, κυκλοφόρησε το 2012, και τον Φεβρουάριο του 2015 κυκλοφόρησε το RPi 2 Model B, με πολύ περισσότερες δυνατότητες και υπολογιστική ισχύ. Όλες οι εκδόσεις περιλαμβάνει έξοδο HDMI και audio, προσαρμογέα οθόνης και προσαρμογέα camera module. Στη συνέχεια αναλύονται τα διαφορετικά μοντέλα RPi.

Raspberry Pi 1 Model A

Η πρώτη έκδοση RPi που κυκλοφόρησε στην αγορά λειτουργεί με τον επεξεργαστή Broadcom BCM2835, τεχνολογίας single-core ARM1176JZF-S (παρόμοια με αυτή των smartphones) στα 700MHz. Όσον αφορά τα γραφικά, περιλαμβάνει GPU Broadcom VideoCore IV στα 250MHz. Επίσης παρέχει 256MB RAM, 1 θύρα USB host, θύρα κάρτας SD και 26 GPIO pins. Να σημειωθεί ότι το RPi 1 Model A δεν παρέχει θύρα ethernet.

Raspberry Pi 1 Model A+



Εικόνα 3.16 - Raspberry Pi Model A, Model A+

Η δεύτερη αυτή έκδοση του RPi είναι σε πολλά χαρακτηριστικά ίδια με την προηγούμενη. Τα πρόσθετα χαρακτηριστικά είναι η αύξηση των GPIO στα 40 pins, η τροποποίηση ώστε να χρησιμοποιεί μικρότερη κάρτα μνήμης MicroSD, το μικρότερο μέγεθος και η μικρότερη κατά 0.5W κατανάλωση.

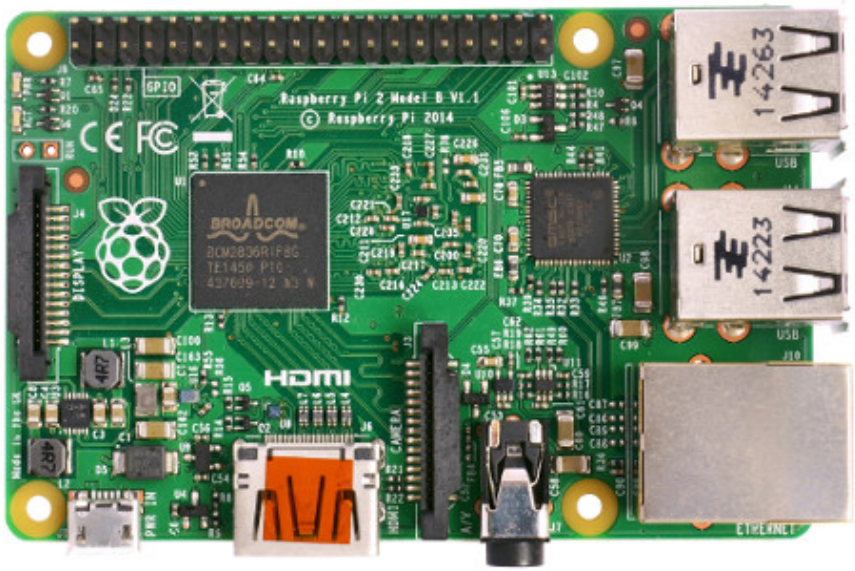
Raspberry Pi 1 Model B

Στο Model B της 1ης σειράς Raspberry Pi βλέπουμε αύξηση της RAM στα 512MB, 2 θύρες USB host, επαναφορά της υποστήριξης κάρτας SD, 26 GPIO και πρόσθεση θύρας ethernet.

Raspberry Pi 1 Model B+

Το Model B+ έχει 40 GPIO, 4 θύρες USB και θύρα MicroSD. Τα υπόλοιπα χαρακτηριστικά είναι ίδια με το Model B.

Raspberry Pi 2 Model B



Εικονα 3.17: Raspberry Pi 2 Model B

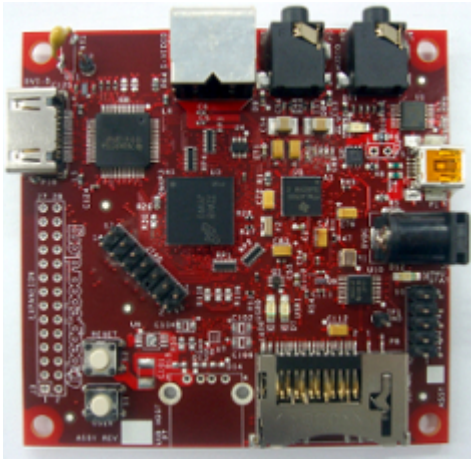
Η πιο πρόσφατη έκδοση στη σειρά περιλαμβάνει τον επεξεργαστή Broadcom BCM2836 τεχνολογίας quad-core ARM Cortex-A7 στα 900MHz, με μνήμη RAM 1GB. Κατά τ' άλλα, η κυριότερη διαφορά του σε σχέση με το προηγούμενο μοντέλο είναι η αύξηση της κατανάλωσης, για την εξυπηρέτηση των διαφορετικών σε απαιτήσεις χαρακτηριστικών.

Το Raspberry Pi υποστηρίζει πολλά OS, με το κυριότερο να είναι το Raspbian, μια παραλλαγή του Debian για το RPi. Το RPi δε διαθέτει εσωτερικό χώρο αποθήκευσης, αντ' αυτού το OS και όλα τα αρχεία αποθηκεύονται στην εξωτερική κάρτα μνήμης.

BeagleBoard

Επόμενα στη λίστα μας είναι τα προϊόντα της σειράς BeagleBoard της εταιρίας TI. Το BeagleBoard, το BeagleBone και το BeagleBone Black είναι οι κυριότεροι ανταγωνιστές του Raspberry Pi στην αγορά. Παρέχουν πολλά GPIO, άριστη απόδοση και περισσότερη ευχρηστία out-of-the-box, μιας και η εγκατάσταση του OS είναι πολύ πιο εύκολη από άλλα αντίστοιχα προϊόντα. Επιπλέον παρέχουν και on board μνήμη, εκτός από την δυνατότητα προσαρμογής κάρτας μνήμης. Ας αναλύσουμε μερικά χαρακτηριστικά τους.

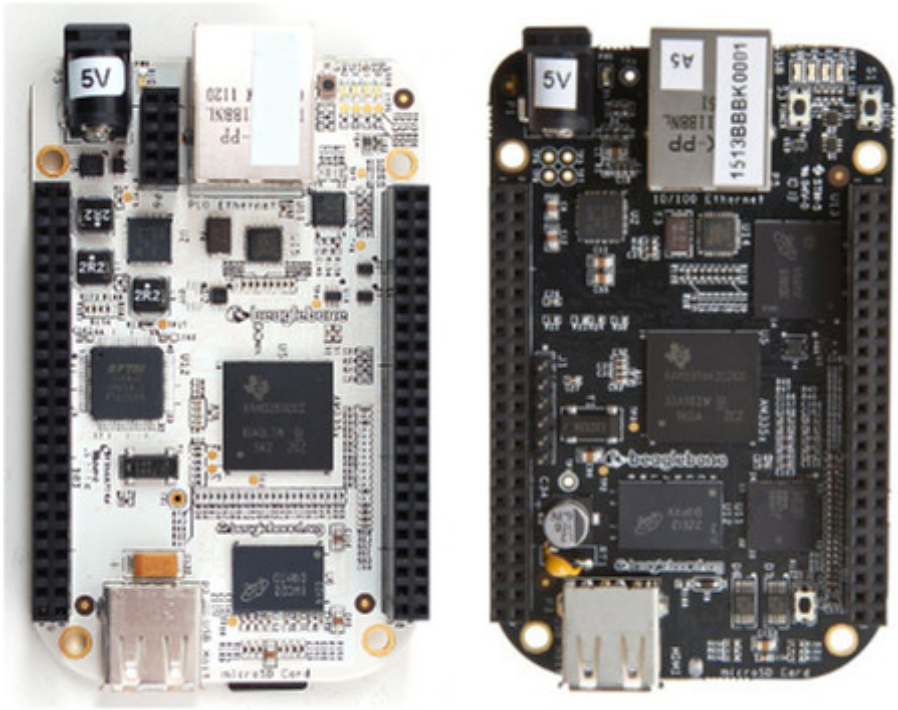
BeagleBoard rev. C



Εικόνα 3.18 - BeagleBoard Rev. C

Το BeagleBoard βασίζεται στον επεξεργαστή TI OMAP3530 SoC, που χρησιμοποιεί τεχνολογία ARM Cortex-A8 στα 720MHz. Δεν περιλαμβάνει θύρα ethernet, όμως παρέχει 256MB RAM και 256MB FLASH, θύρες HDMI και USB OTG mini, προσαρμογέα κάρτας SD, έξοδο audio και barrel jack για τροφοδοσία 5V. Επίσης δίνει τη δυνατότητα expansion με κολλήσεις τερματικών, για υποστήριξη 26 GPIO pins.

BeagleBone



Εικόνα 3.19 - BeagleBone, BeagleBone Black

Σε αντίθεση με το προηγούμενο μοντέλο, το BeagleBone λειτουργεί με τον επεξεργαστή TI Sitara AM335x τεχνολογίας ARM Cortex-A8 στα 720MHz και 256MB RAM. Παρέχει έτοιμα

τερματικά για 92 GPIO, θύρα ethernet, USB host και προσαρμογέα κάρτας microSD.

BeagleBone Black

Το πιο γνωστό μοντέλο της σειράς, το BeagleBone Black (BBB), λειτουργεί με τον ίδιο επεξεργαστή. Οι αναβαθμίσεις σε σχέση με τις προηγούμενες εκδόσεις είναι η μνήμη RAM των 512MB, η λειτουργία του επεξεργαστή στα 1GHz, καθώς και τα 2GB της on-board μνήμης FLASH τα οποία αυξήθηκαν στα 4GB στο μοντέλο rev.C. Η πολύ μεγάλη χωρητικότητα μνήμης FLASH δίνει και την δυνατότητα σε αυτό το μοντέλο να έχει προεγκατεστημένα τα Debian Linux.

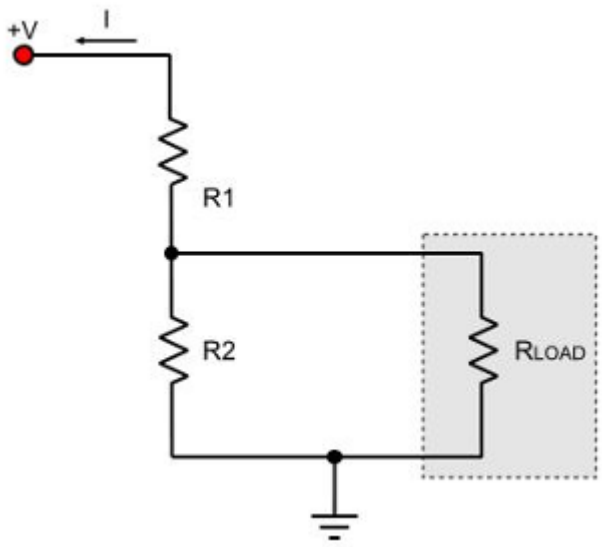
Εκτός από αυτές τις προτάσεις, υπάρχουν και πολλά άλλα αντίστοιχα SoC υποψήφια για τον ρόλο του gateway, τα οποία παρατίθενται με τα χαρακτηριστικά τους στον συγκεντρωτικό πίνακα 3.2.

3.3 Το dimmer

Το dimmer είναι η συσκευή που επιτρέπει την επιλογή ρύθμισης της παρεχόμενης τροφοδοσίας σε ένα κύκλωμα. Ιστορικά έχουν υπάρξει διάφορες υλοποιήσεις για το dimmer, οι οποίες μπορούν να ενταχθούν στις κατηγορίες που περιγράφουμε στη συνέχεια.

Κλασσικό dimmer (διαίρεσης τάσης)

Πριν φτάσουν στη μορφή που έχουν πάρει τώρα, τα dimmers ήταν πλήρως αναλογικά, και αποτελούνταν από ένα κύκλωμα διαιρέτη τάσης. Αυτό σημαίνει ότι, ενώ ο λαμπτήρας ρυθμιζόταν ανάλογα με τις ανάγκες του εκάστοτε χρήστη, όση ενέργεια δεν καταναλωνόταν από αυτόν, καταναλωνόταν από την αντίσταση μέσα στο ίδιο το dimmer σε μορφή θερμότητας. Παρ'όλο που υπάρχει η δυνατότητα να κατασκευασθεί ένα αυτόματο σύστημα που θα χρησιμοποιεί τέτοιου τύπου dimmer, η μη εξοικονόμηση ενέργειας δείχνει την ανάγκη για υπολοίηση άλλης λύσης. Στην εικόνα 3.20 φαίνεται ένα απλό κύκλωμα διαιρέτη τάσης. Το dimmer περιλαμβάνει ένα ποτενσιόμετρο (εικόνα 3.21), του οποίου το κινητό μέρος αλλάζει την τιμή των αντιστάσεων του διαιρέτη τάσης.

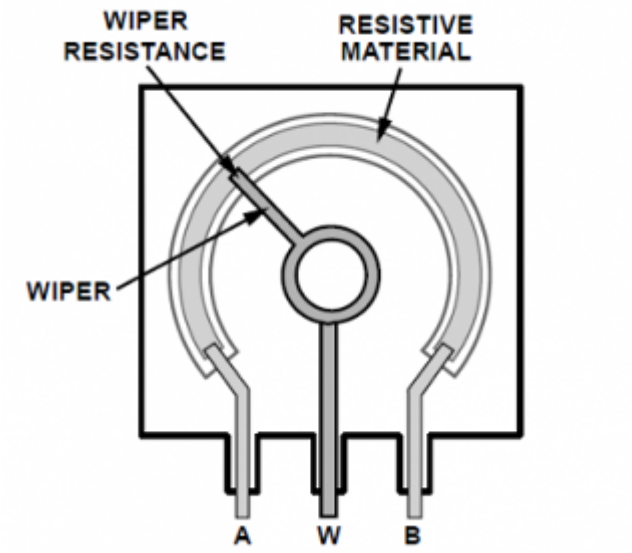


Εικόνα 3.20 - Διαιρέτης τάσης

Name	Processor	Clock speed	GPIO pins	RAM	Storage	Supported OS	Eth	WiFi	BT	Price (€)
Raspberry Pi 1 Model A	Broadcom BCM2835	700MHz	26	256MB	SD	Linux, FreeBSD, WEC7, RISC OS, OpenElec, Raspbian				25.40
Raspberry Pi 1 Model A+	Broadcom BCM2835	700MHz	40	256MB	MicroSD	Same as above				31.45
Raspberry Pi 1 Model B	Broadcom BCM2835	700MHz	26	512MB	SD	Same as above	√			50.55
Raspberry Pi 1 Model B+	Broadcom BCM2835	700MHz	40	512MB	MicroSD	Same as above	√			43.55
Raspberry Pi 2 Model B	Broadcom BCM2836	1GHz	40	1GB	MicroSD	Linux, FreeBSD, Windows 10 IoT, OpenElec, Raspbian	√			50.55
BeagleBoard	TI OMAP3530	720MHz	26	256MB	256MB, SD	Linux, Windows				124.95
BeagleBone	TI Sitara AM335x	720MHz	92	256MB	MicroSD	Android, Linux, FreeBSD, Windows	√			60.00
BeagleBone Black	TI Sitara AM335x	1GHz	92	512MB	4GB, MicroSD	Same as above	√			52.32
Banana Pi	Allwinner A20	1GHz	26	1GB	MicroSD	Android, Linux, FreeBSD, OpenWrt	√			40.00
Banana Pi M2	Allwinner A31	1GHz	40	1GB	MicroSD	Android, Linux, FreeBSD, OpenWrt	√			45.00
Cubieboard	Allwinner A10	1GHz	96	1GB	4GB FLASH, MicroSD	Android, Linux, FreeBSD	√			52.50
Cubieboard 2	Allwinner A20	1GHz	96	1GB	4GB FLASH, MicroSD	Android, Linux, FreeBSD	√			70.21
Cubieboard 3	Allwinner A20	1GHz	54	2GB	8GB FLASH, MicroSD	Android, Linux, FreeBSD	√			88.95
Intel Galileo	Intel Quark SoC X1000	400MHz	14	512MB	8MB, MicroSD	Yocto Linux	√			79.95
Intel Edison	Intel Atom	500MHz	20	1GB	4GB, MicroSD	Yocto Linux		√	√	134.83[1]
ODROID-C1	Amlogic S805	1.5GHz	40	1GB	MicroSD, eMMC	Android, Linux	√			49.50
ODROID-XU4	Exynos 5422	2GHz	30	2GB	MicroSD, eMMC	Android, Linux	√			80.00
PandaBoard ES	TI OMAP4460	1.2GHz	Expansion	1GB	SD	Android, Linux, FreeBSD, RISC	√	√	√	197.54

[1]Price for the Intel Edison with the Arduino Breakout Kit

Πίνακας 3.2 - Σύγκριση χαρακτηριστικών συσκευών υποψήφιων για ρόλο gateway



Εικόνα 3.21 - Ποτενσιόμετρο

Σύγχρονο dimmer (TRIAC dimmer)

Ένα σύγχρονο dimmer, εκτός από το ότι παρέχει την ίδια ακριβώς υπηρεσία, συγχρόνως εξασφαλίζει ότι η κατανάλωση στον κάθε λαμπτήρα θα είναι αντίστοιχη του επιπέδου στο οποίο έχει ρυθμιστεί η φωτεινότητά του. Αυτό συμβαίνει γιατί παρεμβαίνει απ'ευθείας στην τάση που παρέχεται στο λαμπτήρα, έχοντας ως αποτέλεσμα να παρέχεται, για παράδειγμα, τάση στον μισό συνολικά χρόνο αν θέλουμε επίπεδο έντασης 50% του μεγίστου. Η ανάλυση του κυκλώματος αυτού γίνεται στο κεφάλαιο 4.

Το συγκεκριμένο dimmer λειτουργεί διακόπτοντας την τάση τροφοδοσίας, η οποία είναι ένα ημίτονο για μονοφασική παροχή, πολλές φορές μέσα σε ένα χρονικό διάστημα. Αυτό έχει ως αποτέλεσμα, όσο δεν παρέχεται ρεύμα στο λαμπτήρα, να έχουμε ανοιχτό κύκλωμα, οπότε και μηδενική κατανάλωση. Βεβαίως, για να μην έχουμε προβλήματα, πρέπει να συγχρονίζουμε τη διακοπή της παροχής με τους μηδενισμούς της. Ο πιο διαδεδομένος τρόπος για να έχουμε ακριβή τοποθέτηση των μηδενισμών της τάσης είναι η χρήση ενός zero-crossing detector, ο οποίος δίνει έναν παλμό σε κάθε μηδενισμό, που χρησιμοποιείται για τον χρονισμό του dimmer. Η λειτουργία αυτή περιγράφεται εκτενέστερα στο κεφάλαιο 4 ενώ η λειτουργία zero crossing απεικονίζεται στην εικόνα 3.22.

Ένα ακόμα πλεονέκτημα της επιλογής τέτοιου τύπου dimmer (ονομάζεται και TRIAC dimmer γιατί χρησιμοποιεί ολοκληρωμένο κύκλωμα TRIAC για αποκοπή της τάσης) είναι ότι, επειδή περιλαμβάνει από μόνο του απομόνωση της τάσης τροφοδοσίας (230V) από την χαμηλή τάση του κυκλώματος ελέγχου (3.3-9V) μπορούμε να χρησιμοποιήσουμε έναν μικροελεγκτή 5V για τον έλεγχο της λειτουργίας του, καθώς και να τροφοδοτούμε σε αυτόν τον παλμό εξόδου του zero-crossing detector.

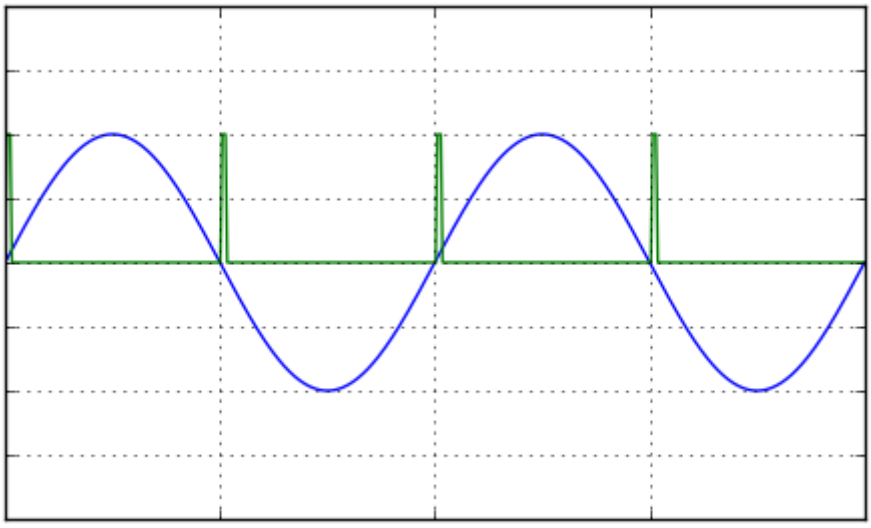
Προφανώς θα επιλέξουμε να υλοποιήσουμε ένα TRIAC dimmer. Τα υλικά που χρειάζονται για την κατασκευή ενός λειτουργικού κυκλώματος είναι τα εξής:

- *Δίοδοι* για την ανόρθωση της τάσης τροφοδοσίας, έτσι ώστε να χρειαστεί μόνο μια συσκευή zero-cross detector. Αν δε χρησιμοποιούσαμε ανόρθωση, θα χρειαζόμασταν ένα detector για τη μεταβολή της τάσης από θετική σε αρνητική, και ένα για την αντίθετη περίπτωση, γεγονός που οδηγεί σε αύξηση της απαιτούμενης τροφοδοσίας, και συνεπώς

της κατανάλωσης. Η ανόρθωση λειτουργεί παθητικά, και συνεπώς δε χρειάζεται τροφοδοσία.

- Ένα στοιχείο που θα πραγματοποιεί τη λειτουργία του zero-crossing detection. Η συσκευή αυτή θα πρέπει να παρέχει την έξοδό της σε έναν μικροελεγκτή σε χαμηλή τάση, οπότε θα πρέπει να λειτουργεί και ως απομονωτής μεταξύ των δυο επιπέδων τάσης. Με κατάλληλη διασύνδεση και πόλωση, μπορούμε να χρησιμοποιήσουμε για αυτή τη λειτουργία ένα *opto-coupler*. Μια άλλη επιλογή είναι η χρήση μετασχηματιστή στην τροφοδοσία, ώστε να φέρουμε την τάση σε επίπεδα αποδεκτά για μικροελεγκτή, και έπειτα να χρησιμοποιήσουμε κατάλληλα κυκλώματα με *τελεστικό ενισχυτή* ή με *transistors*.
- Ένα στοιχείο απομόνωσης των δύο επιπέδων τάσης στο σημείο που θα παρέχουμε τον παλμό στο TRIAC. Εδώ μπορεί να χρησιμοποιηθεί αντίστοιχα ένα *opto-TRIAC,* το οποίο ουσιαστικά αντιγράφει την είσοδο στην έξοδο με ελάχιστη καθυστέρηση, απομονώνοντας παράλληλα τα δύο μέρη.
- Το *TRIAC*, το οποίο, όταν δέχεται τον παλμό λειτουργίας, θα λειτουργεί σαν διακόπτης και θα επιτρέπει την παροχή τάσης στον λαμπτήρα. Μόλις η τάση αλλάξει φορά, το TRIAC ανοιχτοκυκλώνει, και περιμένει τον επόμενο παλμό για να κλείσει το κύκλωμα. Έτσι σε κάθε περίοδο, χάρη στον χρονισμό που παρέχει ο μικροελεγκτής μέσω του zero-cross detector, μπορούμε να ρυθμίζουμε εύκολα το χρονικό διάστημα που δε θα παρέχει τάση το TRIAC, και έπειτα να το θέτουμε σε λειτουργία. Να σημειωθεί ότι για τάση συχνότητας 50Hz, σε συνδυασμό με την ανόρθωση, που δημιουργεί σήμα 100Hz, το TRIAC θα τίθεται σε λειτουργία κάθε 10ms, δηλαδή 100 φορές κάθε δευτερόλεπτο.
- *Αντιστάσεις*, για τη ρύθμιση των πολώσεων και λοιπές λειτουργίες.

Στον πίνακα 3.3 φαίνονται τα ενδεικτικά χαρακτηριστικά μερικών components.



Εικόνα 3.22 - Zero-cross detection

Component or Use	Name	Characteristics	Price (€)
Rectifier	1N400x diode	Max. input Voltage: 50-1000V, Rated Current: 1A	0.02
	DB104 bridge rectifier	Max. input Voltage: 400V, Rated Current: 1A	0.20
	DB105 bridge rectifier	Max. input Voltage: 600V, Rated Current: 1A	0.20
	RS504 bridge rectifier	Max. input Voltage: 400V, Rated Current: 5A	0.90
Opto-coupler	4N25	Insulation test Voltage: 5000Vrms, C-E Voltage: 80V, Forward Current: 60mA, Power Dissipation: 100mW	0.39
	4N35	Insulation test Voltage: 5000Vrms, C-E Voltage: 70V, Forward Current: 1A, Power Dissipation: 70mW	0.39
	PC817	Insulation test Voltage: 5000Vrms, C-E Voltage: 30V, Forward Current: 50mA, Power Dissipation: 70mW	0.25
Opto-TRIAC	MOC3062	Insulation test Voltage: 5300Vrms, Output Voltage: 600V, Trigger Current: 10mA, Forward Curent: 50mA, Power Dissipation: 250mW	0.50
	MOC3041	Insulation test Voltage: 5300Vrms, Output Voltage: 400V, Trigger Current: 15mA, Forward Curent: 60mA, Power Dissipation: 250mW	1.00
	MOC3021	Insulation test Voltage: 7500Vrms, Output Voltage: 400V, Trigger Current: 15mA, Forward Curent: 50mA, Power Dissipation: 330mW	0.50
TRIAC	BT136	Off state max Voltage: 600V, Load Current max: 4A, Gate Current: 70mA, Slew rate: 250V/μs	0.39
	BT137	Off state max Voltage: 600V, Load Current max: 8A, Gate Current: 25mA, Slew rate: 50V/μs	0.60
	BTA08	Off state max Voltage: 800V, Load Current max: 8A, Gate Current: 50mA, Slew rate: 2000V/μs	1.50
	BTA12	Off state max Voltage: 800V, Load Current max: 12A, Gate Current: 50mA, Slew rate: 1500V/μs	1.60
	MAC97A8	Off state max Voltage: 600V, Load Current max: 8A, Gate Current: 1A, Slew rate: 45V/μs	0.20
Others	Resistor 1/4W	Carbon film, Tolerance: ±5%	0.01
	Resistor 1/2W	Carbon film, Tolerance: ±5%, Max voltage: 350V	0.01

Πίνακας 3.3 - Υλικά για την κατασκευή του dimmer

3.4 Επικοινωνία μεταξύ τερματικών

Όπως προαναφέραμε, πρέπει να υλοποιήσουμε τη συσκευή του κάθε αισθητήρα με τρόπο που θα την καθιστά ικανή να αποστέλλει και να δέχεται πληροφορίες. Η ανταλλαγή των πληροφοριών απαιτείται να πραγματοποιείται σε υψηλή ταχύτητα, χωρίς σφάλματα, και με δυνατότητα επαναποστολής σε περίπτωση απώλειας πακέτου. Έχουμε δύο επιλογές, που η κάθε μια επιτρέπει διάφορες υλοποιήσεις και τεχνολογίες: ενσύρματη και ασύρματη επικοινωνία.

3.4.1 Ενσύρματη επικοινωνία

Καλώδιο Ethernet

Η χρήση ενός καλωδίου ethernet δίνει τη δυνατότητα μετάδοσης πακέτων μεγάλου μεγέθους, με πολύ υψηλή ταχύτητα. Μια σύνδεση ethernet μπορεί να φτάσει στα 100Mbps σε ταχύτητα, και η ραγδαία ανάπτυξη των υλικών που χρησιμοποιούνται στα καλώδια αναμένεται να οδηγήσει σε ακόμη μεγαλύτερη αύξηση. Για να πραγματοποιηθεί μια σύνδεση με καλώδιο ethernet μεταξύ δυο σταθμών, υπάρχει η δυνατότητα απ' ευθείας σύνδεσης μεταξύ τους, σύνδεσης σε ένα router σε τοπικό δίκτυο, ή σε έναν μεταγωγέα (ethernet switch). Κάθε συσκευή που παρέχει δυνατότητα σύνδεσης ethernet, έχει μια ταυτότητα που ονομάζεται διεύθυνση MAC, και χρησιμοποιείται στην ανταλλαγή πληροφοριών και στον καθορισμό δέκτη-πομπού σε ένα δίκτυο με πολλά τερματικά. Αν μια συσκευή δεν έχει θύρα ethernet, μπορεί να χρησιμοποιηθεί ένα περιφερειακό, όπως π.χ. το Arduino Ethernet Shield σε ένα Arduino. Μέσω του περιφερειακού και κατάλληλων βιβλιοθηκών κώδικα, η συσκευή αποκτά μια διεύθυνση MAC και, αν είναι συνδεμένη σε router, μια IP, και μπορεί έπειτα να χρησιμοποιηθεί κανονικά σαν σταθμός ethernet.

Σειριακή επικοινωνία

Η σειριακή επικοινωνία χρησιμοποιείται σε όλους τους μικροελεγκτές που αναλύσαμε, και πραγματοποιείται μέσω μιας σύνδεσης USB και μιας συσκευής USB-to-Serial. Εκτός από τη χρήση της για προγραμματισμό, μπορεί να εφαρμοσθεί για ανταλλαγή πληροφοριών μεταξύ συμβατών συσκευών, π.χ. μεταξύ ενός μικροελεγκτή και ενός υπολογιστή. Η σειριακή επικοινωνία παίρνει το όνομά της από το γεγονός ότι μεταδίδεται 1 bit σε κάθε κύκλο, σε αντίθεση με την παράλληλη επικοινωνία, που επιτρέπει σύγχρονη αποστολή πολλών bits. Ας σημειωθεί ότι τα περισσότερα καλώδια και γενικά οι περισσότερες συνδέσεις επικοινωνίας, όπως π.χ. η σύνδεση Ethernet, χρησιμοποιούν σειριακό τρόπο επικοινωνίας. Η επικοινωνία μέσω USB, που αναλύουμε σε αυτή την παράγραφο, ξεκίνησε με τη χρήση του USB 1.0 σε ταχύτητα 1.5 Mbps, και σήμερα έχει φτάσει μέχρι και τα 10 Gbps με χρήση USB 3.0. Η πιο ευρέως χρησιμοποιούμενη έκδοση σε συσκευές όπως οι μικροελεγκτές και γενικά τα SoC είναι η USB 2.0, σε ταχύτητα 480 Mbps. Η συνήθης τάση λειτουργίας μιας σύνδεσης μέσω USB είναι τα 5V, με ρεύμα 500mA για USB 1.0 και 2.0, και 900mA για USB 3.0.

3.4.2 Ασύρματη επικοινωνία

Wi-Fi

Το Wi-Fi είναι μια τεχνολογία ασύρματης τοπικής δικτύωσης που χρησιμοποιεί τις ραδιοσυχνότητες των 2.4 GHz UHF και 5 GHz SHF. Λειτουργεί βάσει του πρωτοκόλλου IEEE 802.11, χρησιμοποιεί κρυπτογράφηση WEP ή WPA/WPA2 για την ασφαλή ανταλλαγή δεδομένων, και μπορεί να φτάσει τις ταχύτητες των 11Mbps (802.11b), 54Mbps (802.11a/g), 300Mbps (802.11n) και 1Gbps (802.11ac). Όπως και στην περίπτωση του ethernet, η επικοινωνία μέσω Wi-Fi απαιτεί την ύπαρξη διεύθυνσης MAC για την ονοματοδοσία αποστολέα/παραλήπτη, καθώς και διεύθυνσης IP, η οποία συνήθως παρέχεται αυτόματα από τον DHCP server του router. Σε περίπτωση μη ύπαρξης προσαρμογέα Wi-Fi, δίνεται η δυνατότητα πρόσθεσης εξωτερικού δέκτη, όπως για παράδειγμα το Wi-Fi shield του Arduino.

Bluetooth

Το Bluetooth είναι μια σχετικά νέα τεχνολογία ανταλλαγής πληροφοριών σε μικρές αποστάσεις. Χρησιμοποιεί τις ραδιοσυχνότητες 2.4 έως 2.485 GHz, και το πρωτόκολλο IEEE 802.15.1. Χρησιμοποιείται ευρέως για ανταλλαγή δεδομένων μεταξύ κινητών, υπολογιστών, εκτυπωτών, και πρόσφατα για σύνδεση περιφερειακών όπως ποντίκια και πληκτρολόγια. Όπως και στην περίπτωση του Wi-Fi, υπάρχουν διαθέσιμα Bluetooth Shields που προσαρμόζονται σε πλακέτες μικροελεγκτών.

Radio modules

Εικόνα 3.23 - NRF24L01, NRF24L01+, RFM43B-868

Εκτός από το Wi-Fi, μπορεί να πραγματοποιηθεί ανταλλαγή πληροφοριών χάρη σε ειδικά ασύρματα modules, που προσαρμόζονται σε GPIO pins συσκευών όπως τους μικροελεγκτές που αναλύσαμε. Δεν απαιτούν σύνδεση σε δίκτυο, ούτε ύπαρξη διεύθυνσης MAC και IP, παρά μόνο διευθύνσεις διαύλων επικοινωνίας, που καθορίζονται από τον χρήστη μέσω κώδικα, και πρέπει να είναι ίδιες για τις συσκευές που επιθυμούμε να είναι σε σύζευξη. Όσον αφορά τη ραδιοσυχνότητα λειτουργίας, υπάρχουν μοντέλα στα 418, 434, 868 MHz και στα 2.4 GHz, με διαφορετικές εμβέλειες, μέγεθος πληροφορίας και τάση λειτουργίας, καθώς και μοντέλα που ενσωματώνουν τον δέκτη και τον πομπό σε ένα module.

Ο πίνακας 3.4 συνοψίζει τα χαρακτηριστικά των τεχνολογιών που αναλύθηκαν, καθώς και κάποια συγκεκριμένα μοντέλα.

3.5 Ετυμηγορία

Σε αυτή την ενότητα θα αναλυθούν οι τελικές επιλογές μας, και οι λόγοι που καταλήξαμε σε αυτές. Μερικές από τις παραμέτρους που λήφθηκαν υπ' όψη κατά την τελική επιλογή, είναι οι εξής:

- Ικανοποίηση απαιτήσεων
- Μέγεθος
- Τιμή
- Ευκολία εύρεσης σημείου αγοράς
- Διαθεσιμότητα
- Συμβατότητα μεταξύ τους
- Ύπαρξη υποστήριξης, documentation και βιβλιοθηκών στο internet και σε βιβλιογραφία
- Γλώσσα προγραμματισμού, αν χρειάζεται

Technology/Name		Data Rate	Max frame (bytes)	Output Power (dbm)	Operating Voltage (V)	Frequency	Price (€)
Ethernet		>100 Mbps	1500	-	-	-	17.99[1]
Serial		480 Mbps	1518	-	-	-	
Wi-Fi		11, 54, 300 Mbps, 1Gbps	2346	100	-	2.4 GHz	15.9[2]
Bluetooth		>24 Mbps	672	10	-	2.4 GHz	24.39[3]
Radio Modules	NRF24L01	2 Mbps	32	-18 to +0	1.9-3.6	2.4 GHz	3.25
	NRF24L01+	2 Mbps	32	-18 to +0	1.9-3.6	2.4 GHz	3.5
	RFM43B-868	256 kbps	64	-8 to +13	1.8-3.6	868 MHz	11
	RFM12B-S2	115 Kbps	16	+2 to +4	2.2-3.8	915 MHz	7
	RFM22B-S2	256 Kbps	64	>+20	1.8-3.6	434 MHz	12
	RFM12BSP	256 Kbps	16	+7	2.2-3.8	434 MHz	7
	XBee ZigBee	250 kbps	87	+5, +8	2.1-3.6	2.4 GHz	24.99
	Xbee-PRO 900HP	10, 200 Kbps	87	>+24	2.1-3.6	928 MHz	40
	XBee 802.15.4	250 Kbps	100	+0	2.8-3.4	2.4 GHz	47.99
	TI CC1101 SMD	600 Kbps	64	>+12	3.6	315-915 MHz	2

[1]Price for Arduino Ethernet Shield
[2]Price for Arduino Wi-Fi Shield
[3]Price for Arduino Bluetooth Shield

Πίνακας 3.4 - Σύγκριση τεχνολογιών επικοινωνίας

3.5.1 Μικροελεγκτής

Σύμφωνα με τον αναλυτικό πίνακα που παρουσιάστηκε, προκύπτουν πολλές πιθανές λύσεις όσον αφορά τον μικροελεγκτή. Για το σύστημά μας επιλέξαμε να χρησιμοποιήσουμε **Arduino**, για τους εξής λόγους:

- Έχει αρκετά χαμηλή τιμή σε σχέση με τους ανταγωνιστές του
- Είναι πολύ εύκολη η εύρεση σημείου αγοράς εντός Ελλάδας
- Είναι μια από τις πρώτες μητρικές πλακέτες μικροελεγκτή, κάτι που σημαίνει ότι υπάρχει εκτενέστατο support για πολλά modules, αρκετό documentation, και κυκλοφορούν στην αγορά προσαρμόσιμα shield που καθιστούν εύκολη τη χρήση του Arduino σε projects
- Τα μοντέλα Nano, Micro και Pro Mini προσφέρουν χαρακτηριστικά όπως ταχύτητα επεξεργαστή, interrupts, timer και αριθμό pins που ικανοποιούν τις ανάγκες μας, σε πολύ μικρό μέγεθος, κάτι που λείπει από τους ανταγωνιστές
- Η κύρια γλώσσα προγραμματισμού είναι C++

Ο κυριότερος λόγος υπήρξε το πολύ μικρό μέγεθος που προσφέρουν τα 3 μοντέλα που αναφέρθηκαν. Καθώς το σύστημα πρέπει να είναι πολύ εύκολα εγκαταστάσιμο και θεωρητικά "αόρατο", ώστε να μην επηρεάζεται η αρχιτεκτονική του χώρου, είναι λογικό να επιλέξουμε κάποιες συσκευές βάσει του μεγέθους τους, αν τα χαρακτηριστικά ικανοποιούν επαρκώς τις απαιτήσεις.

3.5.2 Επικοινωνία

Για την επιλογή διεπαφής επικοινωνίας, ας θεωρήσουμε ότι έχουμε τις εξής παραμέτρους απόφασης:

- *Ταχύτητα μετάδοσης:* λόγω του ότι γίνεται διάδοση πολλών πληροφοριών, εκ των οποίων πολλές μεταδίδονται και ταυτόχρονα, θέλουμε υψηλή ταχύτητα, έτσι ώστε να μην παρακωλύεται η λειτουργία του συστήματος.
- *Εμβέλεια:* αν και τα τερματικά βρίσκονται στον ίδιο χώρο, πολύ πιθανό είναι να υπάρχουν εμπόδια στη μετάδοση. Οπότε, στην περίπτωση ασύρματου πρωτοκόλλου, μεγάλη εμβέλεια σημαίνει και μικρότερη εξάρτηση από το πλήθος και τη σύσταση των εμποδίων.
- *Τάση τροφοδοσίας:* οι περισσότεροι μικροελεγκτές που κυκλοφορούν στην αγορά έχουν τη δυνατότητα παροχής τάσης 3.3V ή 5V, οπότε καλό είναι να χρησιμοποιηθεί ασύρματο περιφερειακό που περιέχει μια από αυτές τις τάσεις στις προδιαγραφές του.
- *Μέγεθος πληροφορίας:* σε πολλές περιπτώσεις, η πληροφορία που στέλνεται μπορεί να είναι αρκετά μεγάλη, οπότε ένα μέγεθος 32B είναι πολύ καλό για να αποφευχθεί απώλεια. Επιπλέον, η προσθήκη δυναμικής ρύθμισης μεγέθους πληροφορίας σε πολλά πρωτόκολλα είναι ιδιαίτερα χρήσιμη, καθώς έτσι αποφεύγεται η άσκοπη χρήση και κατανάλωση ενέργειας.
- *Ενσωμάτωση πομπού-δέκτη σε μια συσκευή:* επειδή όλα τα τερματικά στο σύστημα δρουν ως πομποί και δέκτες, η ενσωμάτωση σε ένα περιφερειακό συμβάλλει στην φορητότητα και εξοικονόμηση χώρου στο σύστημα, και στην αποφυγή πολύπλοκων συνδέσεων και κώδικα.

Για να αποφευχθεί η χρήση πολύπλοκου κώδικα, απορρίψαμε τις επιλογές που εντάσσουν το σύστημα σε τοπικό δίκτυο, δηλαδή το Ethernet και το Wi-Fi. Με αυτή την επιλογή προκύπτει και το πλεονέκτημα ότι δεν εξαρτάται το σύστημα από σύνδεση internet, οπότε μπορεί να λειτουργήσει ακόμα και όταν το τοπικό router δε λειτουργεί, πράγμα που συμβαίνει συχνά σε δίκτυα χώρων εργασίας.

Έπειτα απορρίψαμε το Bluetooth λόγω χαμηλής εμβέλειας, και τη σειριακή επικοινωνία USB λόγω του ότι θα απαιτούσε τουλάχιστον 6 θύρες USB από το gateway, δηλαδή και περισσότερη κατανάλωση ενέργειας.

Τα παραπάνω μας αφήνουν την επιλογή των ασύρματων modules, και μάλιστα εκείνων που ικανοποιούν τις παραπάνω απαιτήσεις. Επιλέξαμε τα **NRF24L01** και **NRF24L01+** λόγω του μικρού τους μεγέθους, της χαμηλής τους τιμής, και κυρίως λόγω του εκτενέστατου support που έχουν για πολλούς μικροελεγκτές και γενικά συσκευές SoC.

3.5.3 Μετρητής κατανάλωσης

Όσον αφορά τη συσκευή για μέτρηση κατανάλωσης, απορρίψαμε εξ' αρχής το power clamp, λόγω του πολύ μεγάλου κόστους. Επίσης απορρίφθηκε το Kill-A-Watt λόγω του ότι απαιτεί προσαρμογή με κολλήσεις για να μπορεί να λειτουργήσει στο σύστημά μας, και λόγω μεγάλου μεγέθους. Από τους μετασχηματιστές ρεύματος, καταφέραμε να βρούμε μόνο τα YHDC και το Sparkfun, και επιλέχθηκε το **YHDC SCT013-030**. Το πλεονέκτημά του σε σχέση με τα υπόλοιπα μοντέλα είναι ότι η προδιαγραφή 30A/1V βοηθά στην κατασκευή του κυκλώματος προσαρμογής, όπως θα αναλύσουμε σε επόμενο κεφάλαιο.

3.5.4 Αισθητήρας φωτεινότητας

Εδώ επιλέξαμε ένα απλό **LDR**, καθώς το κόστος αγοράς LDR και μερικών αντιστάσεων, για να υλοποιηθεί το κύκλωμα, είναι σαφώς μικρότερο από το κόστος έτοιμης πλακέτας, που πρακτικά δεν προσφέρει κάτι παραπάνω.

3.5.5 Gateway

Στον αναλυτικό πίνακα είδαμε πολλές επιλογές για το gateways, που μπορούν να καλύψουν με διάφορους τρόπους τις απαιτήσεις μας. Ας παραθέσουμε πάλι τις απαιτήσεις που παρουσιάσαμε στην αρχή του κεφαλαίου, μαζί με κάποιες πιο συγκεκριμένες:

- Θέλουμε τυπικά το gateway να συμπεριφέρεται σαν υπολογιστής. Αυτό σημαίνει ότι χρειαζόμαστε ταχύτητα, δυνατότητα εγκατάστασης υπηρεσιών δικτύου, όπως web hosting, ssh, ftp και άλλα.
- Όλα τα παραπάνω είναι αναγκαίο να περιέχονται σε μια αρκετά μικρή συσκευή, για να πληροί τον χαρακτήρα του συστήματος προς ανάπτυξη.
- Επειδή θα αποθηκεύουμε δεδομένα εντός του gateway σε τοπική βάση δεδομένων, προσθέτουμε την απαίτηση για αρκετή μνήμη, εσωτερική ή εξωτερική.
- Βάσει του ότι έχουμε επιλέξει το NRF24L01, θα αναζητήσουμε μια επιλογή για την οποία προσφέρονται libraries, μιας και η ανάπτυξη του πρωτοκόλλου εξ' αρχής ή η μετατροπή των ήδη υπαρχόντων libraries σε άλλη γλώσσα προγραμματισμού είναι αρκετά δύσκολες διαδικασίες. Επίσης, για τον ίδιο λόγω θέλουμε να παρέχονται GPIO pins και SPI, κάτι που έχουν όλες οι συσκευές που αναλύθηκαν.
- Όσον αφορά το operating system, μας ενδιαφέρει να υποστηρίζει κάποια έκδοση Linux πολύ σταθερή, για να μπορέσουμε να αναπτύξουμε το WebGUI με ευκολία. Εάν υπάρχει κάποια έκδοση που έχει περάσει αρκετό testing, μπορούν να αποφευχθούν δυσλειτουργίες του συστήματος. Ο λόγος που δεν προτιμούμε κάποιο άλλο OS είναι η ασφάλεια και η ευκολία στις δοκιμές και debugging κώδικα που προσφέρουν τα Linux.

Σχεδόν όλες οι συσκευές που αναλύσαμε πληρούν αυτές τις προϋποθέσεις. Εμείς επιλέξαμε το **Raspberry Pi 2 Model B**, για τους εξής λόγους:

- Μαζί με το BeagleBone Black, όλα τα μοντέλα RPi ήταν τα μόνα που βρέθηκαν άμεσα διαθέσιμα στην αγορά, καθώς τα υπόλοιπα ήταν διαθέσιμα μόνο από εξωτερικό, κάτι που αυξάνει πολύ την τιμή τους, αν συμπεριληφθούν φόροι, κλπ.
- Σε σχέση με το BBB, το RPi παρέχει πολύ καλύτερο support στο internet, μέσω boards και official documentation. Επίσης, η βιβλιοθήκη που διατίθεται για το NRF24L01 σε python αποδείχτηκε πολύ αξιόπιστη.
- Από όλα τα μοντέλα RPi, επιλέχθηκε το τελευταίο λόγω των σαφώς καλύτερων χαρακτηριστικών του.
- Το λογισμικό Raspbian χρησιμοποιείται στα RPi εδώ και πολύ καιρό, και έχει υποστεί πολλές δοκιμές. Παρέχει updates σε πολύ σύντομες χρονικές περιόδους, και τρέχει το πολύ ελαφρύ LXDE desktop environment σε συνδυασμό με το επίσης ελαφρύ Openbox Window Manager. Όλα αυτά προσφέρουν ένα πολύ ελκυστικό desktop, στο οποίο μπορεί κανείς να συνδεθεί με VNC ή RDP, αν και η περισσότερη δουλειά έγινε μέσω SSH από άλλον υπολογιστή στο τοπικό δίκτυο.

3.5.6 Dimmer

Όσον αφορά τα στοιχεία του dimmer, σχεδόν όσες επιλογές αναφέραμε βρέθηκαν και στην αγορά άμεσα. Από αυτές επιλέξαμε, για κάθε λειτουργία:

- Rectifier: **DB105**, γιατί έχει ανοχή έως 600V. Καθώς ο ανορθωτής παρέχει ρεύμα μόνο στο opto-coupler, δεν υπάρχει περίπτωση να απαιτηθούν περισσότερο από 1A, οπότε δε χρειάζεται να επιλεγεί το RS504, που έχει προδιαγραφή στα 5A αλλά μόνο στα 400V. Επίσης δεν επιλέξαμε μεμονωμένες διόδους διότι με το IC εξοικονομείται περισσότερος χώρος στην πλακέτα και παρέχεται καλύτερη μόνωση.
- Opto-coupler: **4N35**, λόγω του ότι έχει προδιαγραφή κατανάλωσης 70mW.
- Opto-TRIAC: **MOC3062**, λόγω του χαμηλού trigger current (ανταποκρίνεται στα χαμηλά ρεύματα εξόδου του Arduino) και της χαμηλότερης σε σχέση με τα υπόλοιπα κατανάλωσης.
- TRIAC: **MAC97A8**, λόγω του χαμηλού slew rate, που σημαίνει ότι ανταποκρίνεται στους παλμούς που του στέλνει το Opto-TRIAC πολύ γρήγορα (Slew rate είναι ο ρυθμός με τον οποίο φτάνει σε μια απαιτούμενη τάση, και το μετράμε από το 10% μέχρι το 90% της τάσης, από όταν τεθεί το TRIAC σε λειτουργία).

Κεφάλαιο 4 Προτεινόμενη λύση - σχεδιασμός

Σύμφωνα με την ανάλυση του προηγούμενου κεφαλαίου, έχουμε καταλήξει στα εξής components για το σύστημα:

- Arduino για την επεξεργασία των πληροφοριών που παρέχονται από τους αισθητήρες και προς το dimmer
- Raspberry Pi για την υλοποίηση του gateway, με ενσωμάτωση web server και υπηρεσιών απομακρυσμένης διαχείρισης
- NRF24L01(+) για την επικοινωνία μεταξύ των Arduino και του Raspberry Pi
- LDR για υλοποίηση του αισθητήρα φωτεινότητας
- Μετασχηματιστή ρεύματος (YHDC SCT013-030) για υλοποίηση του αισθητήρα (μετρητή) κατανάλωσης
- Dimmer που χρησιμοποιεί zero cross detection και TRIAC

Το τελικό σύστημα θα έχει τη μορφή πολλών τερματικών που επικοινωνούν με ένα κεντρικό gateway. Τα τερματικά θα υλοποιηθούν πάνω σε πλακέτα, και θα είναι τα εξής:

- Αισθητήρας φωτεινότητας
- Μετρητής κατανάλωσης
- Dimmer

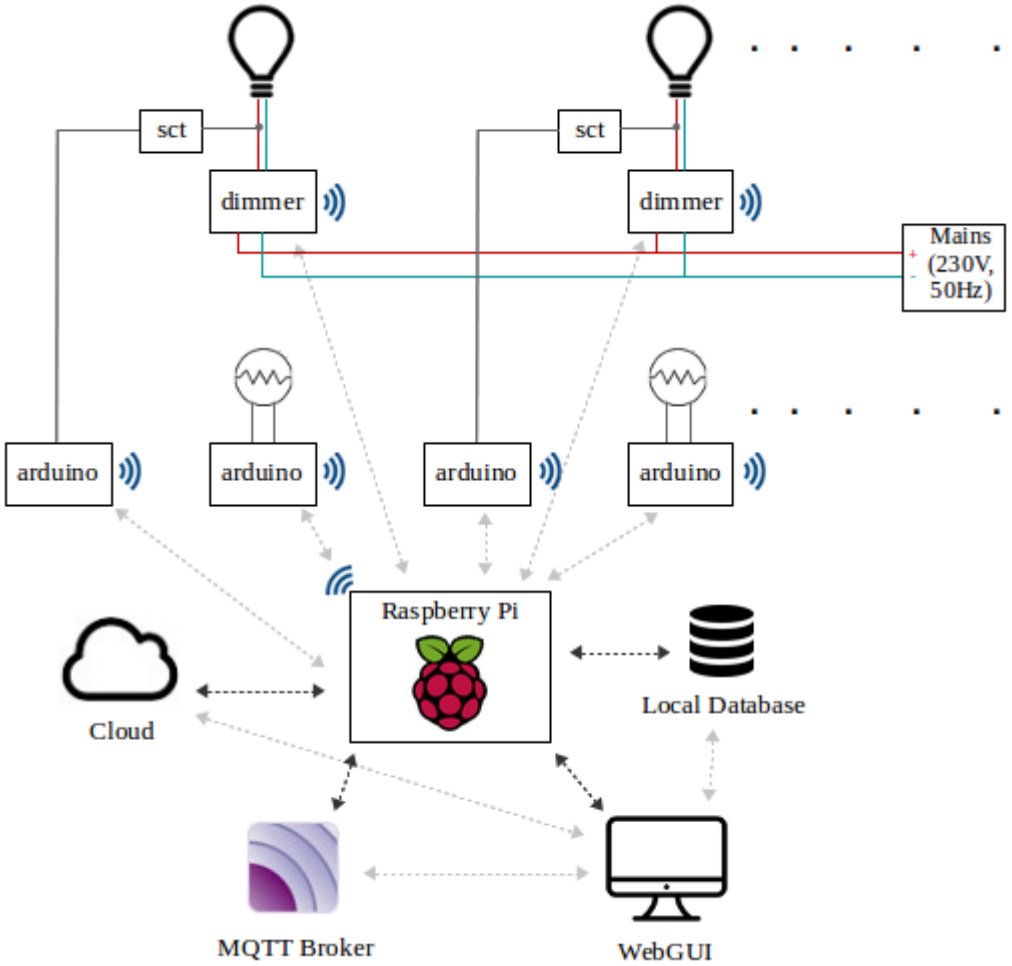
Στη συνέχεια θα γίνει μια ανάλυση, ακολουθώντας την ιεραρχία που θεωρούμε ότι έχει το σύστημα:

1. Τερματικά
    - Συσκευές μέτρησης
        - Αισθητήρες φωτεινότητας
        - Μετρητές κατανάλωσης
    - Dimmer
2. Gateway
3. Εργαλεία και δομές ενσωματωμένες στο gateway
    - Graphical User Interface (WebGUI)
    - Database
    - MQTT Broker

Ο όρος "ιεραρχία" αναφέρεται στο γεγονός ότι τα υψηλότερα σε αυτή components καθορίζουν την ανάπτυξη των υπολοίπων.

4.1 Τερματικά

Ξεκινώντας, θα θέσουμε τον εξής κανόνα, και θα εξηγήσουμε το γιατί: *Τα τερματικά, δηλαδή οι συσκευές που είναι υπεύθυνες για το χειρισμό των φυσικών χαρακτηριστικών του συστήματος, δε θα πρέπει να επικοινωνούν μεταξύ τους.* Λόγω του ότι θα υλοποιήσουμε ένα πρωτόκολλο επικοινωνίας, αυτό σημαίνει την επεξεργασία μετρήσεων και μεταβλητών, και τη λήψη



Εικόνα 4.1 - Αρχιτεκτονική συστήματος

αποφάσεων βάσει αυτών. Επειδή τα τερματικά (και συγκεκριμένα τα Arduino) θα εκτελούν διεργασίες που θα καταλαμβάνουν μεγάλο ποσοστό της υπολογιστικής τους δύναμης, δε θέλουμε να προσθέσουμε το περαιτέρω υπολογιστικό βάρος της αποστολής πληροφοριών, γι'αυτό και θα το αναθέσουμε στο gateway. Ως αποτέλεσμα, τα τερματικά αποστέλλουν και δέχονται πληροφορίες μόνο προς και από το gateway.

4.1.1 Συσκευές μέτρησης

4.1.1.1 Γενική λειτουργία

Οι συσκευές μέτρησης αποτελούνται απαιραίτητα από ένα Arduino (Nano ή Micro, λόγω του μικρού τους μεγέθους) και από το NRF radio module. Τα υπόλοιπα στοιχεία τους εξαρτώνται από τον σκοπό που εξυπηρετούν. Στην περίπτωσή μας είναι ο αισθητήρας (LDR, SCT) και το κύκλωμα προσαρμογής του, που μετατρέπει το ηλεκτρικό σήμα της εξόδου του αισθητήρα σε ένα σήμα κατάλληλο για είσοδο στο Arduino.

Η "ρουτίνα" που θα εκτελούν οι συσκευές μέτρησης είναι η εξής:

1. Αναμονή για εντολή από το gateway.
2. Λήψη εντολής.
3. Αν η εντολή δεν προορίζεται για την ίδια συσκευή, τότε αγνόηση και επιστροφή στην αρχή.
4. Αν η εντολή προορίζεται για την ίδια συσκευή και ζητά αποστολή μέτρησης, τότε γίνεται ανάγνωση της τιμής παρέχεται από τον αισθητήρα αυτή τη χρονική στιγμή.
5. Επεξεργασία της μέτρησης, μετατροπή της σε αναγνώσιμη μορφή και ενσωμάτωση σε header.
6. Αποστολή της μέτρησης στο gateway.
7. Επανάληψη.

Βλέπουμε ότι, αν υποθέσουμε ότι έχουμε αναμονή π.χ. 30 δευτερολέπτων, η συσκευή λειτουργεί για πολύ μικρό χρονικό διάστημα (η ρουτίνα έχει διάρκεια μικρότερη από 1 δευτερόλεπτο), και μέχρι να τεθεί πάλι σε λειτουργία, έχουμε πολύ μικρή κατανάλωση.

4.1.1.2 Αισθητήρας φωτεινότητας

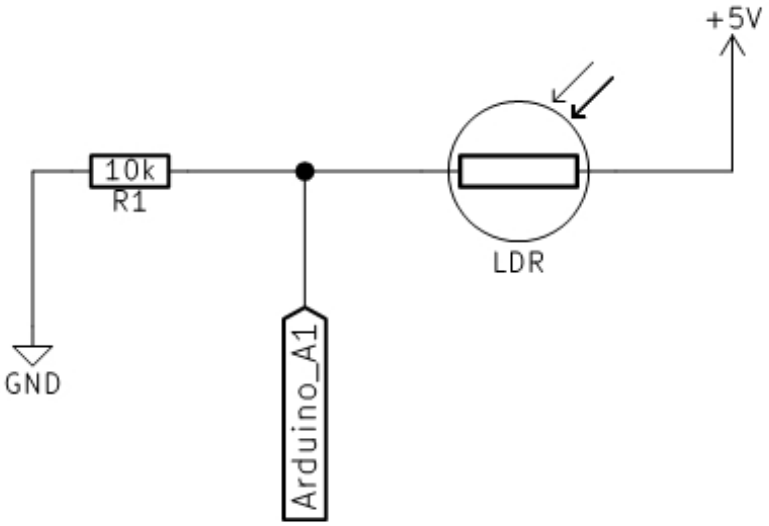
Έχουμε καταλήξει ότι για τη μέτρηση του επιπέδου φωτεινότητας θα χρησιμοποιηθεί ένα LDR, δηλαδή μια αντίσταση με τιμή που εξαρτάται από τη φωτεινότητα του χώρου (Light Dependent Resistor). Για να χρησιμοποιηθεί όμως το LDR, θα χρειαστεί η τοποθέτησή του σε ένα κύκλωμα προσαρμογής.

Στην περίπτωση του LDR, το πιο απλό και διαδεδομένο κύκλωμα προσαρμογής είναι ο διαιρέτης τάσης, δηλαδή η πρόσθεση μιας ακόμα αντίστασης. Ο λόγος που χρειάζεται η αντίσταση είναι ο εξής: αν χρησιμοποιηθεί μόνο το LDR, θα πρέπει ο ένας ακροδέκτης του να συνδεθεί σε analog pin του Arduino, και ο άλλος σε μια τάση αναφοράς, 5V ή 0V. Έτσι, θεωρητικά, καθώς θα αλλάζει τιμή η αντίσταση του LDR ανάλογα με την φωτεινότητα, θα υπάρχει και μια αντίστοιχη πτώση τάσης κατά μήκους του, οπότε θα παίρνουμε μια μέτρηση στο pin. Αυτό όμως δε συμβαίνει.

Έχοντας έναν ακροδέκτη συνδεμένο το analog pin, δεν παρέχουμε μια τάση ως αναφορά, έτσι ώστε να μπορεί να "κλείσει" το κύκλωμα. Ως αποτέλεσμα, ο ακροδέκτης αυτός έχει απροσδιόριστη τάση, είναι δηλαδή floating point.

Υλοποιώντας το κύκλωμα του διαιρέτη τάσης που έχουμε δείξει στο κεφάλαιο 3 και παραθέτουμε και παρακάτω, χρησιμοποιούμε την αντίσταση (10kΩ στη συγκεκριμένη περίπτωση) σαν pull-down resistor, δηλαδή σαν ένα component που κρατάει τη συγκεκριμένη πλευρά του κυκλώματος δε μια χαμηλή τάση (ή τουλάχιστον χαμηλότερη των 5V). Από εκεί και πέρα, η μέτρηση λαμβάνεται από τον κοινό ακροδέκτη του LRD και της αντίστασης R1. Σε περίπτωση χαμηλής φωτεινότητας, το LDR λαμβάνει υψηλή τιμή και έχουμε μεγαλύτερη πτώση τάσης κατά μήκους του, άρα και χαμηλότερη τάση στο pin της μέτρησης. Σε υψηλή φωτεινότητα, και όσο αυξάνεται το επίπεδο, συμβαίνει το αντίθετο, και η τάση στο pin της μέτρησης αυξάνεται.

Να σημειωθεί ότι το Arduino μεταφράζει μια τάση 0-5V στο analog pin του σε μια τιμή 0-1023. Αυτό δε σημαίνει ότι λειτουργεί σαν ADC, καθώς μπορεί να μετρήσει με κάποια ακρίβεια μόνο DC τάση, ενώ ένα ADC μεταφράζει και AC τάσεις όπως ημιτονοειδείς παλμούς, κλπ.



Εικόνα 4.2 - Κύκλωμα προσαρμογής LDR

4.1.1.3 Μετρητής κατανάλωσης

Όπως έχουμε ήδη αναφέρει, για τη σωστή λήψη των μετρήσεων είναι απαραίτητη η ενσωμάτωση ενός κυκλώματος προσαρμογής. Για να καταλάβουμε γιατί, θα πρέπει να γίνει μια θεωρητική ανάλυση σχετικά με τη λειτουργία ενός μετασχηματιστή ρεύματος.

Οι μετασχηματιστές ρεύματος είναι συσκευές-αισθητήρες που έχουν τη δυνατότητα μέτρησης εναλλασσόμενου ρεύματος. Μπορούν να προσαρμοσθούν πάνω στην τάση ή στον ουδέτερο κλώνο του καλωδίου τροφοδοσίας (αρκεί αυτό να γίνει με τη σωστή φορά του ρεύματος, η οποία είναι από την τροφοδοσία προς τη συσκευή, αν προσαρμόσουμε τον μετασχηματιστή στην τάση, και αντίθετη αν τον προσαρμόσουμε στο ουδέτερο).

Όπως όλοι οι μετασχηματιστές, ένας μετασχηματιστής ρεύματος αποτελείται από το πρωτεύον και από το δευτερεύον τύλιγμα, και χαρακτηρίζεται από την αναλογία τους, ως εξής:

$$n = \frac{primary\ turns}{secondary\ turns}$$

Έτσι, το ρεύμα εξόδου καθορίζεται από αυτή την αναλογία, βάσει της σχέσης:

$$n = \frac{I_o}{I_i} \quad \text{ή} \quad I_o = n \cdot I_i$$

Επειδή ο μετασχηματιστής ρεύματος συμπεριφέρεται σαν πηγή ρεύματος στην έξοδό του, θα πρέπει να προστεθεί μεταξύ των 2 ακροδεκτών της εξόδου μια αντίσταση, που ονομάζεται burden resistor. Επειδή το Arduino λαμβάνει τάση στα pins του, και η έξοδος του μετασχηματιστή δίνει γνωστό ρεύμα αλλά όχι γνωστή τάση, είναι αναγκαίο να μετατραπεί αυτό το ρεύμα σε μια γνωστή τάση, και έπειτα να μετατραπεί, σε επίπεδο αποδεκτό από το Arduino. Βέβαια, στην περίπτωσή μας, η προδιαγραφή 30A/1V υπονοεί ότι το burden resistor είναι ενσωματωμένο στο SCT, και δε χρειάζεται να το προσθέσουμε στο κύκλωμα προσαρμογής.

Εκτός από το burden resistor, χρειάζεται η ενσωμάτωση ενός κυκλώματος προσαρμογής, το οποίο θα εξασφαλίζει ότι η τάση θα μπορεί να είναι αναγνώσιμη από το Arduino, δηλαδή η μέγιστη τιμή της θα είναι μικρότερη ή ίση από 5V. Έτσι, όπως φαίνεται και στην εικόνα 4.3, χάρη στον διαιρέτη τάσης που κατασκευάζουν οι 2 αντιστάσεις R1 και R2 (R1=R2), το σήμα που προέρχεται από τον μετασχηματιστή πολώνεται στα 2.5V. Ο πυκνωτής εξασφαλίζει ένα δρόμο για το εναλλασσόμενο ρεύμα, έτσι ώστε να μην περάσει από τις αντιστάσεις αυτές.

Όπως αναλύθηκε, η αντίσταση 33Ω, δηλαδή η αντίσταση burden, μετατρέπει την έξοδο του μετασχηματιστή ρεύματος, που είναι μια συγκεκριμένη τιμή ρεύματος, σε μια συγκεκριμένη τιμή τάσης, την τιμή $V_{out} = I_{out}*R_b$. Αυτό προφανώς συμβαίνει για να μπρορεί να τροφοδοτηθεί στο Arduino μια τάση, μιας και δε μπορεί να διαβάσει ρεύμα.

Η τιμή της αντίστασης R_b προκύπτει ως εξής: στο μέγιστο ονομαστικό ρεύμα του μετασχηματιστή 100A:50mA, έχουμε στο πρωτεύον τύλιγμα:

$$I_{primary} = I_{pr,RMS} \cdot \sqrt{2} = 100\,A \cdot \sqrt{2} = 141.4\,A$$

Και στο δευτερεύον:

$$I_{secondary} = \frac{I_{primary}}{n} = \frac{141.4\,A}{2000} = 0.0707\,A$$

Όπου το n=2000 είναι ο αριθμός τυλιγμάτων, και προκύπτει από τη σχέση n=100A/50mA.

Έτσι, για να μεγιστοποιηθεί η ακρίβεια των μετρήσεων, οι τιμές της εναλλασόμενης τάσης στην είσοδο του Arduino θα πρέπει να κυμαίνονται γύρω από το μισό της τάσης αναφοράς του, δηλαδή τα 2.5V:

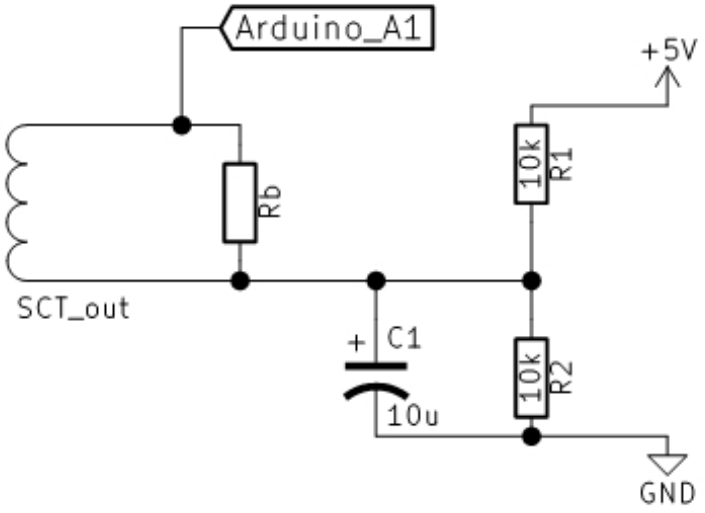
$$R_b = \frac{A_{REF}/2}{I_{secondary}} = \frac{2.5\,V}{0.0707\,A} = 35.4\,\Omega$$

Οι κοντινότερες τιμές αντίστασης που μπορούν να βρεθούν είναι 33Ω και 39Ω, και επιλέγουμε τη μικρότερη, στα 33Ω. Σε γενικές γραμμές, η R_b καθορίζει το εύρος της τάσης που θα πάρει ως είσοδο το Arduino. Αν είμαστε σίγουροι για το εύρος των ρευμάτων που θα παρέχουμε στον μετρητή, τότε μπορούμε να επιλέξουμε μεγαλύτερη τιμή R_b για να αυξήσουμε την ακρίβεια.

Για να εξασφαλίσουμε ότι η τάση θα κυμαίνεται γύρω από τα 2.5V, θα χρησιμοποιήσουμε έναν διαιρέτη τάσης με ίσες τιμές αντιστάσεων, σε τάση 5V από την παροχή του Arduino. Έτσι η R_b πολώνεται στα 2.5V.

Στην περίπτωσή μας, που χρησιμοποιούμε μετασχηματιστή 30A/1V, μπορούμε να πολώσουμε την (εσωτερική στο SCT) R_b ακόμα και στα 4V, καθώς η μέγιστη τάση που θα δίνει σε αυτή την περίπτωση είναι 5V. Παρ'όλα αυτά, για να είναι το κύκλωμα προσαρμόσιμο, σε περίπτωση που χρειαστεί να χρησιμοποιηθεί μετασχηματιστής 100A:50mA, αλλά και για να βρισκόμαστε στην ασφαλή πόλωση $A_{REF}/2$, θα κρατήσουμε την πόλωση στα 2.5V.

Αξίζει να αναφερθεί πάλι ότι ο πυκνωτής λειτουργεί σαν φίλτρο θορύβου, έτσι ώστε να υπάρχει αναγνώσιμο σήμα στην είσοδο του Arduino. Θόρυβο θα έχουμε όταν μετράμε την κατανάλωση σε μια γραμμή που συνδέονται πολλές συσκευές, αλλά μπορεί να προκληθεί και από επαγωγικά φαινόμενα.



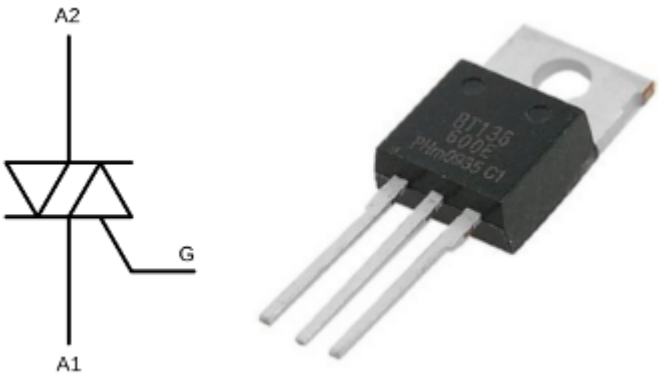
Εικόνα 4.3 - Κύκλωμα προσαρμογής SCT

4.1.2 Dimmer

Έχουμε αναφέρει κάποιες αρχές σχετικά με τη λειτουργία των TRIAC dimmers, και σε αυτό το κεφάλαιο θα γίνει μια εκτενέστερη ανάλυση. Συγκεκριμένα, εδώ θα γίνει πρώτα η ανάλυση του κυκλώματος, και μετά η περιγραφή της λειτουργίας του.

4.1.2.1 Ανάλυση κυκλώματος

Η λειτουργία αυτού του τύπου dimmer είναι ως εξής: αντί να λειτουργεί σαν διαιρέτης τάσης, με αποτέλεσμα να υπάρχει η ίδια κατανάλωση όσο και να μειωθεί η ένταση του λαμπτήρα, χρησιμοποιεί παλμούς για τον χειρισμό ενός TRIAC. Με την ορολογία TRIAC αναφερόμαστε στο ολοκληρωμένο **TRI**ode for **A**lternate **C**urrent, το οποίο είναι μια ηλεκτρονική συσκευή τριών τερματικών, που άγει ρεύμα προς οποιαδήποτε κατεύθυνση όταν ενεργοποιείται, λειτουργία παρόμοια με αυτή του ρελέ. Η παρακάτω εικόνα απεικονίζει ένα τέτοιο TRIAC, όπου A1 είναι η άνοδος 1, A2 είναι η άνοδος 2 και G η πύλη.



Εικόνα 4.4 - BT136 TRIAC και το αντίστοιχο schematic

Στο A1 συνδέεται η τροφοδοσία, στην περίπτωσή μας ο κλώνος που παρέχει τα 230V, και στο

τερματικό A2 το φορτίο. Οποτεδήποτε το ρεύμα στο G ξεπεράσει μια συγκεκριμένη τιμή, το threshold current, η επαφή μεταξύ των A1 και A2 κλείνει, και έτσι παρέχεται ρεύμα από την τροφοδοσία στο φορτίο. Να σημειωθεί ότι το φορτίο συνδέεται κατευθείαν στον ουδέτερο κλώνο του καλωδίου τροφοδοσίας, και έτσι για να κλείσει το κύκλωμα αρκεί να κλείσει η επαφή στον κλώνο της τάσης.

Αν υπάρξει ένα μικρό ρεύμα στην πύλη του TRIAC, κάτι που συνήθως πραγματοποιείται με εφαρμογή παλμού, αυτό λειτουργεί σαν κλειστός διακόπτης μεταξύ του λαμπτήρα και της τάσης. Επειδή τροφοδοτούμε μια τάση 230V AC, και ρεύμα 1kΩ/230V = 4.34 A AC, ακόμα και όταν το ρεύμα είναι κοντά σε μηδενισμό, το TRIAC θα είναι σε λειτουργία.

Το ζητούμενο σε αυτό το πρόβλημα είναι να βρούμε έναν τρόπο να δημιουργούμε κατάλληλα χρονισμένους παλμούς στο Arduino, και να μπορούμε να τους παρέχουμε στο TRIAC. Εδώ όμως προκύπτουν τα εξής θέματα:

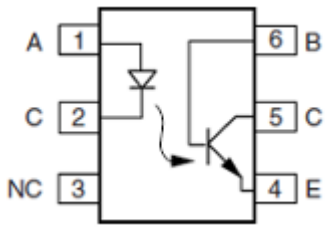
1. Το TRIAC λειτουργεί σε υψηλή τάση 230V RMS, παρ' όλο που ο παλμός που απαιτεί είναι αρκετά μικρότερος. Επειδή το Arduino έχει μέγιστη αποδεκτή τάση στα pins του 5V, πρέπει να υπάρξει μια απομόνωση μεταξύ αυτών των 2 επιπέδων τάσης.
2. Παρ' όλο που γνωρίζουμε ότι η συχνότητα της τάσης τροφοδοσίας είναι 50Hz, άρα και έχουμε ένα πλήρες ημίτονο κάθε 0.2 sec, δεν είναι αποδοτικό να στέλνουμε παλμούς σε τυχαία σημεία, καθώς αυτό θα οδηγήσει σε flickering των λαμπτήρων. Άρα πρέπει να βρεθεί ένας τρόπος να χρονίσουμε το Arduino σε κάθε περίοδο της τάσης τροφοδοσίας, ή ακόμα καλύτερα σε κάθε περίοδο της ανορθωμένης τάσης, δηλαδή 2 φορές σε κάθε περίοδο του πλήρους ημιτόνου.

Σύμφωνα με την ανάλυση του προηγούμενου κεφαλαίου, για την επίλυση αυτών των προβλημάτων έχουμε επιλέξει ένα opto-TRIAC και ένα opto-coupler αντίστοιχα.

Opto-coupler

Το opto-coupler είναι ένα ολοκληρωμένο κύκλωμα της οικογένειας των opto-isolators. Πρόκειται για ολοκληρωμένα που χρησιμοποιούν φως και υπέρυθρη ακτινοβολία αντί για φυσική σύνδεση καλωδίων-τερματικών, και χρησιμοποιούνται κυρίως για απομόνωση τάσης. Μπορούν να λειτουργήσουν σε τάση ακόμα και μέχρι 10kV, και υπάρχουν λύσεις που προσφέρουν slew rate, δηλαδή καθυστέρηση μεταφοράς σήματος, ακόμα και 10kV/μs.

Συγκεκριμένα το opto-coupler που χρησιμοποιούμε στο σύστημά (4N35) μας έχει τη μορφή της εικόνας 5.3, και αποτελείται από ένα LED και ένα transistor, το οποίο ονομάζεται phototransistor, και πρόκειται για ένα απλό BJT του οποίου η ένωση βάσης-συλλέκτη είναι οπτικά ευαίσθητη. Αυτό σημαίνει ότι η τιμή της τάσης που εφαρμόζεται στην ένωση έχει εξάρτηση από το φως.



Εικόνα 4.5 - Opto-coupler

Εμείς θα χρησιμοποιήσουμε την ιδιότητα του opto-coupler σαν zero-cross detector, ως εξής: πολώνουμε τον συλλέκτη του npn BJT phototransistor (pin 5) στα 5V, μέσω ενός pull-up resistor 10kΩ, και τον εκπομπό (pin 4) στη γείωση (Arduino GND). Ως αποτέλεσμα, ο συλλέκτης έχει τιμή λίγο μικρότερη από 5V, και η τιμή της τάσης στην βάση (pin 6) εξαρτάται από το εσωτερικό LED.

Όταν η τάση στην είσοδο (pins 1, 2) φτάσει στα 0V και αρχίσει να αυξάνεται (όταν έχουμε ημίτονο), τότε το LED εκπέμπει, και έχουμε αύξηση της τάσης στην βάση του phototransistor. Έτσι αυτό εισέρχεται στην ενεργό περοχή και άγει, οπότε παίρνουμε στην έξοδο (pin 5), τάση λίγο χαμηλότερη από 5V, καθώς υπάρχει πτώση τάσης στην αντίσταση.

Παρ'όλα αυτά, επειδή η είσοδος κυμαίνεται σε πολύ μεγαλύτερες τιμές από την πόλωση, είναι λογικό σε κάποιο σημείο, το οποίο είναι πολύ κοντά στα 0V του ημιτόνου, η βάση του phototransistor να έχει τιμή μεγαλύτερη από 5V. Σε εκείνο το σημείο το phototransistor εισέρχεται στον κόρο (για τον κόρο στα npn BJT είναι E<B>C), οπότε και λειτουργεί σαν κλειστός διακόπτης μεταξύ E και C. Δηλαδή έχουμε βραχυκύκλωμα του pin 5 με το pin 4, το οποίο συνδέεται στη γείωση, οπότε η τιμή στην έξοδο γίνεται 0V. Αυτό έχει ως αποτέλεσμα να δημιουργηθεί ένας παλμός μικρού πλάτους και τάσης ~5V.

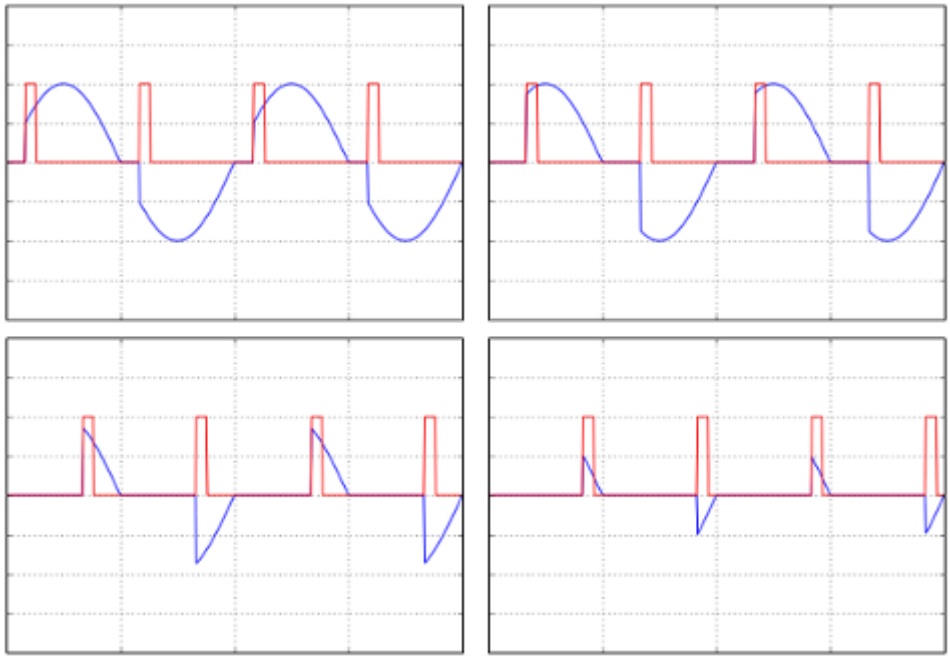
Τα παραπάνω συμβαίνουν προφανώς όποτε υπάρχει αύξηση της τάσης από τα 0V. Επειδή θέλουμε παλμούς σε κάθε μηδενισμό του ημιτόνου, μπορούμε να χρησιμοποιήσουμε 2 opto-couplers, εκ των οποίων το ένα θα έχει την είσοδό του συνδεμένη αντίθετα, και να υπερθέσουμε τις εξόδους τους στην είσοδο του Arduino. Η άλλη λύση, την οποία και θα υλοποιήσουμε, είναι να τροφοδοτήσουμε ένα πλήρως ανορθωμένο ημίτονο, έτσι ώστε μετά από κάθε μηδενισμό να υπάρχει αύξηση της τάσης.

Opto-TRIAC

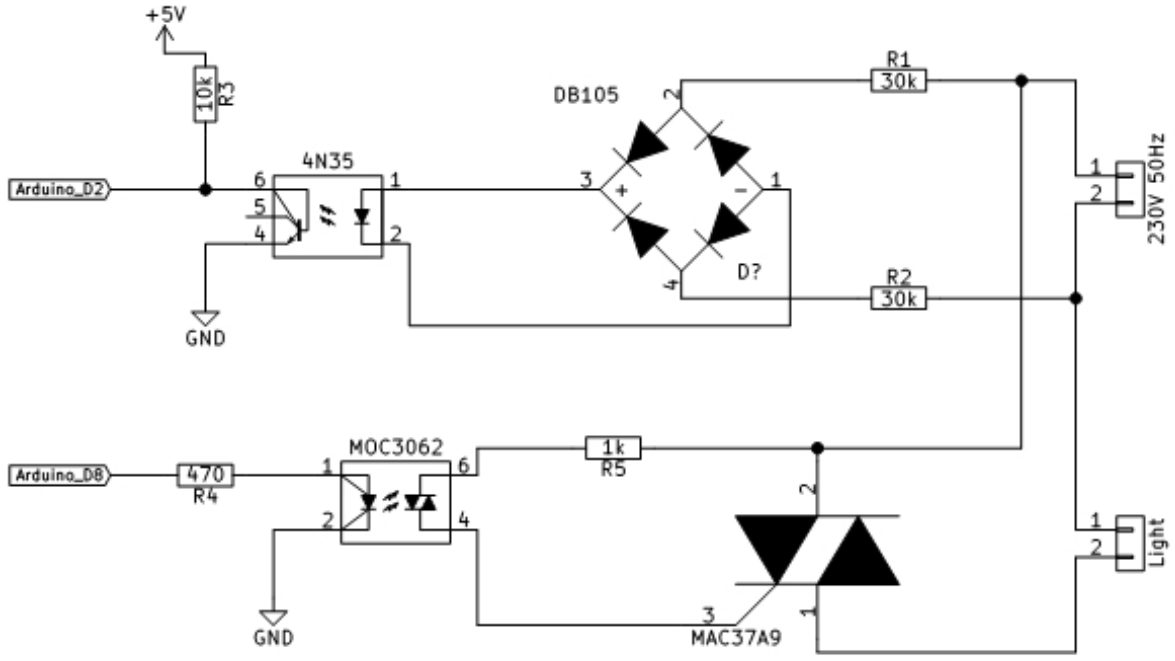
Η λειτουργία του opto-TRIAC που χρησιμοποιήσαμε (MOC3062) είναι αρκετά απλή. Σε αντίθεση με το opto-coupler, εδώ εφαρμόζουμε έναν παλμό 5V (έξοδος του Arduino) στην είσοδο. Το LED εκπέμπει φως που πολώνει την πύλη του εσωτερικού TRIAC, το οποίο έπειτα λειτουργεί σαν ένα κανονικό TRIAC, δηλαδή σαν ρελέ. Εδώ έχουμε πολώσει το ένα τερματικό του TRIAC στην τάση 230V AC μέσω μιας αντίστασης 1kΩ-1/2W. Ως αποτέλεσμα, κατά τη διάρκεια που ο παλμός του Arduino είναι σε τάση HIGH (5V), το TRIAC του ολοκληρωμένου λειτουργεί σαν βραχυκύκλωμα, και η τάση 230V AC περνά στο άλλο τερματικό, οπότε και τροφοδοτείται στο TRIAC.

Με τα ζητήματα να έχουν λυθεί σε επίπεδο hardware, τώρα προκύπτει το ερώτημα του πως θα αξιοποιήσουμε την πληροφορία του zero-cross detector, και πως έπειτα θα παράγουμε τον παλμό που χρειάζεται το opto-TRIAC για να λειτουργήσει. Η λύση παρέχεται από τον πολύ δυνατό μικροελεγκτή Atmel328p του Arduino UNO: τα interrupts. Αν και για να γίνει κατανοητό το interrupt πρέπει να συνοδευτεί από τον κατάλληλο κώδικα, κάτι που θα γίνει στο επόμενο κεφάλαιο, η γενική του λειτουργία είναι η εξής: όταν αντιληφθεί απότομη αλλαγή της τάσης σε ένα προκαθορισμένο pin, τρέχει μια ακολουθία εντολών αποθηκευμένων σε μια επίσης προκαθορισμένη συνάρτηση (δηλαδή σε ένα function της γλώσσας C++), που ονομάζεται interrupt, καθώς έχει ως αποτέλεσμα τη διακοπή της ροής του κυρίως προγράμματος. Σε συνδυασμό με ένα από τα πολύ ακριβή timers που παρέχονται ενσωματωμένα στον μικροελεγκτή, μπορούμε να δημιουργήσουμε έναν κατάλληλα χρονισμένο παλμό, σχεδόν ταυτόχρονα με την ανάγνωση του παλμού zero-crossing στην είσοδο του Arduino. Ο παλμός τροφοδοτείται στο TRIAC, και ελέγχει την περίοδο λειτουργίας του. Το αποτέλεσμα για διάφορους χρονισμούς παλμών φαίνεται στην εικόνα 4.6. Το τελικό κύκλωμα του dimmer

φαίνεται στην εικόνα 4.7.



Εικόνα 4.6 - Αποκοπή παλμού τροφοδοσίας



Εικόνα 4.7 - Το κύκλωμα του dimmer

Όπως παρατηρούμε, σε σειρά με την τάση εισόδου τοποθετούμε αντιστάσεις 30kΩ, 1/2W. Αυτό εξασφαλίζει ότι η είσοδος στην ανόρθωση, καθώς και η ανορθωμένη είσοδος στο opto-coupler, θα είναι στα 120V RMS. Αυτό δεν είναι απολύτως απαραίτητο, αλλά σε γενικές γραμμές δε χρειάζεται πολύ υψηλή τάση για να λειτουργήσει ένα κύκλωμα zero crossing. Θα μπορούσαμε να μειώσουμε ακόμα περισσότερο την τάση, με μεγαλύτερες τιμές αντιστάσεων, αλλά αυτό θα οδηγούσε σε υπερθέρμανσή τους, αλλά και σε αύξηση της αντίστασης εισόδου

του κυκλώματος, δηλαδή αύξηση και του ρεύματος λειτουργίας (και επομένως της κατανάλωσης). Να σημειωθεί ότι στην εικόνα 5.4 οι σχέσεις μεταξύ των τάσεων δεν είναι πραγματικές: οι ημιτονοειδείς κυματομορφές είναι στα 230V RMS, δηλαδή έχουν μέγιστη τιμή 325V, και οι παλμοί έχουν μέγιστο στα 5V.

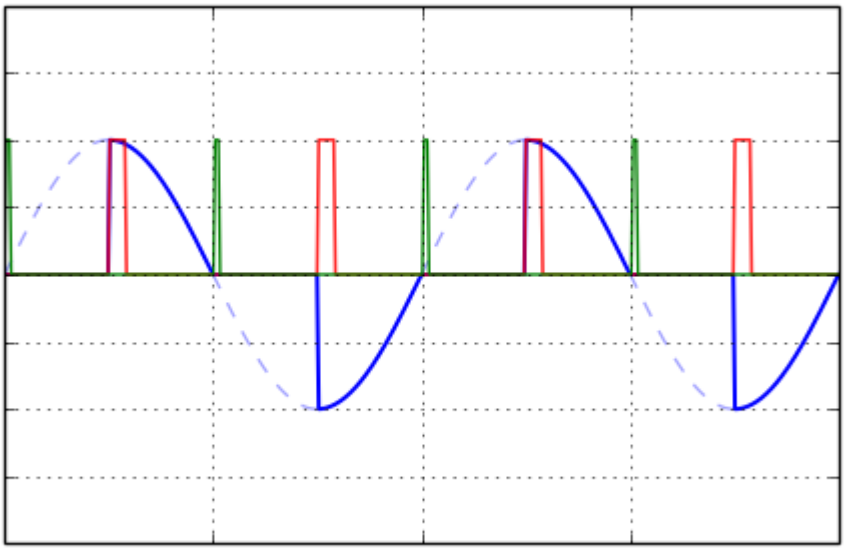
4.1.2.2 Λειτουργία

Συνοψίζοντας, η λειτουργία του dimmer έχει ως εξής:

- Opto-coupler για τη δημιουργία των παλμών zero-crossing, οι οποίοι θα τροφοδοτούνται στο Arduino.
- Χρήση software interrupts στο Arduino, τα οποία πυροδοτούνται όταν δεχτεί παλμό, και, με χρήση ενός timer, παράγουμε τον παλμό εξόδου την κατάλληλη χρονική στιγμή, ανάλογα με το επίπεδο dimming.
- Τροφοδότηση αυτού του παλμού σε ένα opto-TRIAC, το οποίο τον αναπαράγει στην έξοδό του, εξασφαλίζοντας απομόνωση του Arduino από την υψηλή τάση (mains, 230V AC).
- Τροφοδότηση της εξόδου του opto-TRIAC σε ένα TRIAC, το οποίο με τη σειρά του κλείνει το κύκλωμα και τροφοδοτεί ρεύμα στον λαμπτήρα.

Και, όπως προαναφέρθηκε, αυτή η ρουτίνα εκτελείται κάθε 0.1 sec, και όχι κάθε 0.2 sec (περίοδος ημιτόνου 50Hz), για λόγους που θα αναφερθούν αμέσως.

Όλα τα παραπάνω μπορούν να γίνουν κατανοητά βάσει της εικόνας 4.8, στην οποία φαίνεται με πράσινο ο παλμός zero crossing, με κόκκινο ο παλμός εξόδου του Arduino προς το opto-TRIAC, και με μπλε η τελική κυματομορφή της τάσης, που είναι και η τάση του λαμπτήρα, για επίπεδο dimming 50%. Η διακεκκομένη γραμμή είναι η είσοδος του dimmer, δηλαδή η κανονική τάση 230V/50Hz.



Εικόνα 4.8 - Λειτουργία του dimmer

Βέβαια, για να γίνει πλήρως κατανοητή η λειτουργία του dimmer, θα πρέπει να αναλυθεί και ο ρόλος του Arduino στη λειτουργία του. Σε αντίθεση με τα υπόλοιπα τερματικά, στα οποία το Arduino εκτελεί κάποιες αρκετά βασικές λειτουργίες, ο ρόλος του στο dimmer είναι ιδιαίτερα σημαντικός. Μπορούμε να πούμε δηλαδή ότι το Arduino είναι μέρος της συσκευής που ονομάζουμε dimmer. Η ανάλυση απαιτεί παρουσίαση κώδικα, καθώς και εισαγωγή νέων εννοιών, και θα γίνει στο κεφάλαιο 5.

4.2 Gateway

Το gateway θα αναλαμβάνει το κυριότερο μέρος της επεξεργασίας των πληροφοριών, καθώς και της επικοινωνίας μεταξύ κόμβων. Αυτό δικαιολογεί και την επιλογή του Raspberry Pi αντί για έναν μικροελεγκτή. Στη συνέχεια γίνεται μια περιγραφή της λειτουργίας του gateway, χωρισμένη σε ενότητες για διευκόλυνση της ανάλυσης.

4.2.1 Λήψη, επεξεργασία και αποστολή πληροφοριών

Αμέσως μετά το setup stage, το gateway μπαίνει στον επαναλαμβανόμενο βρόχο της κανονικής λειτουργίας. Η πρώτη του ενέργεια είναι να περιμένει ως listener για εισερχόμενες πληροφορίες από τους αισθητήρες φωτός και τους μετρητές κατανάλωσης. Παράλληλα περιμένει και για εισερχόμενες εντολές, όπως π.χ. τη αλλαγή του threshold ή την χειροκίνητη ρύθμιση της φωτεινότητας ενός λαμπτήρα, από το GUI.

Στο γεγονός της άφιξης μιας πληροφορίας, το gateway εκτελεί μια σειρά εντολών: πρώτα αξιολογεί την πληροφορία, και επιβεβαιώνει αν ακολουθεί το προδιαγεγραμμένο πρωτόκολλο (δηλαδή ότι δεν πρόκειται για αποτέλεσμα δυσλειτουργίας ή παρεμβολή). Έπειτα "αναγνωρίζει" τα διάφορα segments που ορίζει το πρωτόκολλο, έτσι ώστε να εξακριβώσει την προέλευση της, τον τύπο της συσκευής που την απέστειλε, το είδος της πληροφορίας που μεταφέρει, και τελικά το περιεχόμενο του μηνύματος. Τα επόμενα βήματα εξαρτώνται από το είδος του τερματικού προέλευσης της πληροφορίας:

- Αισθητήρας φωτός: η τιμή αποθηκεύεται σε κατάλληλο πίνακα, έτσι ώστε να συγκριθεί με το threshold και, αν είναι απαραίτητο, να γίνει εκτέλεση του αλγορίθμου επιλογής της βέλτιστης φωτεινότητας, ώστε να εξαχθούν οι τιμές που πρέπει να αποσταλούν στα dimmer.
- Μετρητής κατανάλωσης: η τιμή αποθηκεύεται στην κατάλληλη δομή (local database, cloud database, file) και παράλληλα αποστέλλεται σε μια δομή που επιτρέπει την real time αναπαράσταση δεδομένων στο GUI (η δομή αυτή θα περιγραφεί στο επόμενο κεφάλαιο).
- Dimmer: όπως προαναφέραμε, θέλουμε να αποφύγουμε την αποστολή πληροφοριών από το dimmer. Παρ' όλα αυτά, είναι πιο αποδοτικό για το σύστημα, όταν το dimmer ρυθμιστεί στο 0 ή 100%, να ειδοποιεί το gateway, έτσι ώστε να αποφευχθεί οποιαδήποτε αίτηση για αλλαγή της φωτεινότητας άσκοπα και πέρα από τα όρια.

Έπειτα γίνεται, εφ' όσον χρειάζεται, η εκτέλεση του αλγορίθμου, και τελικά η αποστολή του αιτήματος για αλλαγή της φωτεινότητας σε κάθε dimmer.

4.2.2 Αλγόριθμος επιλογής έντασης φωτισμού

Στα προηγούμενα κεφάλαια έχουν γίνει αρκετές αναφορές στον αλγόριθμο επιλογής

έντασης. Σε αυτή την ενότητα θα γίνει μια περαιτέρω ανάλυση, αλλά θα προσθέσουμε και άλλα στοιχεία στο επόμενο κεφάλαιο, μιας και χρειάζεται να ληφθούν υπ' όψη οι περιορισμοί του κώδικα. Εξ' άλλου, ο αλγόριθμος αυτός είναι ίσως το σημαντικότερο κομμάτι της διπλωματικής εργασίας, από την άποψη ότι από αυτό τον αλγόριθμο εξαρτάται η απόδοση του συστήματος.

Ας αναφέρουμε πρώτα κάποια πιο τεχνικά ζητήματα. Κατ' αρχάς, θα περιγράψουμε πως αντιλαμβάνεται το dimmer τις αλλαγές που ζητάει το gateway να γίνουν στην ένταση του κάθε λαμπτήρα.

Όπως αναφέρθηκε στην ενότητα 4.1.3, το dimmer χρονίζεται βάσει του παλμού zero cross, ο οποίος "πυροδοτεί" τη ρουτίνα interrupt του Arduino. Έπειτα το Arduino, ανάλογα με την τιμή που θα τεθεί η φωτεινότητα του λαμπτήρα, αποστέλλει τον κατάλληλο παλμό που με τη σειρά του ενεργοποιεί το TRIAC. Όταν αυτό ενεργοποιηθεί, κλείνει το κύκλωμα και τίθεται σε λειτουργία ο λαμπτήρας. Αυτό βέβαια συμβαίνει αρκετές φορές μέσα σε ένα μικρό χρονικό διάστημα, για την ακρίβεια 10 φορές κάθε δευτερόλεπτο, μιας και σε συχνότητα τροφοδοσίας 50 Hz υπάρχουν 2 παλμοί zero-crossing σε κάθε περίοδο. Να σημειωθεί επίσης ότι ο παλμός είναι μικρής διάρκειας, όσο χρειάζεται για να ενεργοποιηθεί το TRIAC, το οποίο απενεργοποιείται με το που η τάση τροφοδοσίας φτάσει σε μηδενική τιμή.

Αυτό που θέλουμε να καταλήξουμε είναι ότι το dimmer δεν αντιλαμβάνεται ένταση φωτισμού σε ποσοστό %, αλλά αντιθέτως χρονικό διάστημα από τη στιγμή που θα δεχθεί τον παλμό zc μέχρι τη στιγμή που θα στείλει τον παλμό στο TRIAC. Εμείς εξάγουμε μια αντιστοίχιση του χρονικού αυτού διαστήματος (ας το ονομάσουμε δt_{dim}) και του % ποσοστού της φωτεινότητας, η οποία αντιστοίχηση μάλιστα δείχνει και αντιστρόφως ανάλογη σχέση μεταξύ τους (κάτι το λογικό, καθώς όσο αυξάνεται το δt_{dim} τόσο μειώνεται η ένταση του λαμπτήρα.

Ένα ζήτημα που προκύπτει είναι ότι η αντιστοίχιση αυτή δε μπορεί να γίνει με ακρίβεια. Αν είχαμε παλαιού τύπου dimmer, που η ένταση του φωτός εξαρτάται από την τιμή μιας αντίστασης, η οποία είναι γραμμικό στοιχείο, θα μπορούσαμε πολύ εύκολα να κάνουμε μια αντιστοίχιση γραμμικά. Εδώ όμως πρόκειται για ημιτονοειδή τροφοδοσία οπότε το δt_{dim} για μεταβολή 0-10% να διαφέρει από το δt_{dim} για μεταβολή 10-20%.

Παρ' όλα αυτά, αυτό το πρόβλημα μπορεί να μη ληφθεί υπ' όψη, μιας και έχουμε προκαθορίσει ότι οι αλλαγές στις εντάσεις θα είναι ανεπαίσθητες. Αυτό σημαίνει ότι μπορούμε να θέσουμε ένα μέσο δt_{dim} για κάθε μετάβαση κατά 10%, αν και μόνο αν υλοποιήσουμε το σύστημα έτσι ώστε να μη ζητά μεταβάσεις μεγαλύτερες του 2-3% σε κάθε κύκλο λειτουργίας (του gateway). Ως αποτέλεσμα, θα συμβαίνει μεγαλύτερη αλλαγή στον φωτισμό στη μετάβαση από το 10% στο 13%, σε σχέση με τη μετάβαση από το 50% στο 53%. Γενικά θα συμβαίνει μεγαλύτερη αλλαγή φωτεινότητας, για ίδιο δt_{dim}, στα σημεία της περιόδου του ημιτόνου που έχουν μεγαλύτερη κλίση (ή παράγωγο), δηλαδή κοντά στους μηδενισμούς. Αυτό όμως δε θα γίνεται αντιληπτό, και είναι κάτι που μπορεί να αγνοηθεί, χάριν απλούστευσης και εξοικονόμησης υπολογιστικών πόρων.

Ένα ακόμα ζήτημα είναι η ικανοποίηση ενός περιορισμού που θέσαμε στο κεφάλαιο 2: το σύστημα να είναι προσαρμόσιμο σε αλλαγές της τοπολογίας, χωρίς να χρειαστεί παρέμβαση στο software του συστήματος. Γενικά είναι λογικό να υπάρχει κάποια σχέση εξάρτησης μεταξύ λαμπτήρων και αισθητήρων φωτεινότητας, η οποία θα έχει αντίκτυπο στη λειτουργία του αλγορίθμου. Αυτό σημαίνει ότι η ρύθμιση ενός λαμπτήρα θα εξαρτάται περισσότερο από την φωτεινότητα στο σημείο που βρίσκονται αισθητήρες με μεγάλη εξάρτηση από τον συγκεκριμένο λαμπτήρα.

Αυτό μπορεί να επιτευχθεί, αν το gateway διατηρεί ένα είδος λίστας με κάποιες μετρήσεις για κάποια συγκεκριμένα επίπεδα φωτεινότητας. Από αυτή τη λύση όμως προκύπτει ένα ακόμα πρόβλημα: στην περίπτωση που κάποιος αισθητήρας αλλάξει θέση, ή που κάποιο αντικείμενο που επηρεάζει την οπτική επαφή μεταξύ αισθητήρα και κάποιου λαμπτήρα τοποθετηθεί μόνιμα στον χώρο, η προαναφερθείσα λίστα θα είναι obsolete, καθώς πλέον θα υπάρχει άλλη συνάρτηση που να περιγράφει τη σχέση που θέλουμε.

Η λύση που προτείνεται είναι η εξής: πριν την κανονική λειτουργία του συστήματος, θα υπάρχει ένα στάδιο προρύθμισης (setup stage), κατά το οποίο οι αισθητήρες θα συλλέγουν τις απαραίτητες πληροφορίες. Κάθε αισθητήρας θα αποστέλλει μια μέτρηση για διάφορα επίπεδα φωτεινότητας κάθε λαμπτήρα, με τους υπόλοιπους σβηστούς.

4.2.3 Setup stage

Όπως αναφέραμε, θα πρέπει να διατηρούμε μιας μορφής δομή, η οποία θα μας παρέχει όλες τις απαραίτητες πληροφορίες που συσχετίζουν την ένταση του κάθε λαμπτήρα με την ανάγνωση που δίνει ο κάθε αισθητήρας. Αυτό θα γίνει ως εξής, στην περίπτωση που έχουμε 2 αισθητήρες και 2 λαμπτήρες: με την εκκίνηση του συστήματος, το gateway ζητάει από τα dimmer να σβήσουν τα φώτα, δηλαδή να θέσουν τη φωτεινότητα σε επίπεδο 0%. Έπειτα αποστέλλει εντολή στους αισθητήρες, οι οποίοι με τη σειρά τους εκτελούν ανάγνωση της φωτεινότητας του χώρου και επιστρέφουν το αποτέλεσμα στο gateway. Με τη σειρά του το gateway επαναλαμβάνει αυτή τη διαδικασία για επίπεδα φωτεινότητας 10-100% στον 1ο λαμπτήρα, ενώ ο 2ος είναι στο 0%, και έπειτα πάλι για επίπεδα 10-100% στον 2ο ενώ ο 1ος είναι σβηστός.

Έτσι χτίζουμε στην πλευρά του gateway έναν πίνακα, ο έχει την εξής μορφή (συμβολίζοντας τις μετρήσεις με *v*-value):

Light level (%)		**0**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Sensor 1	**Light 1**	*v11*	*v12*	*v13*	*v14*	...	..	.				
	Light 2	*v21*	*v22*	*v23*	...	..	.					
Sensor 2	**Light 1**	*v31*	*v32*	...	..	.						
	Light 2	*v41*	...	..	.							

Προφανώς, επειδή οι μετρήσεις της 1ης στήλης εκτελούνται για μηδενική φωτεινότητα και των 2 λαμπτήρων, v11=v21 και v31=v41.

Αν έπειτα εξάγουμε τη διαφορά μεταξύ διαδοχικών μετρήσεων σε κάθε αισθητήρα, καταλήγουμε με τον εξής πίνακα:

Sensor 1	**Light 1**	*v12-v11*	*v13-v12*	*v14-v13*	*v15-v14*	…	..	.			
	Light 2	*v22-v21*	*v23-v22*	*v24-v23*	…	..	.				
Sensor 2	**Light 1**	*v32-v31*	*v33-v32*	…	..	.					
	Light 2	*v42-v41*	…	..	.						

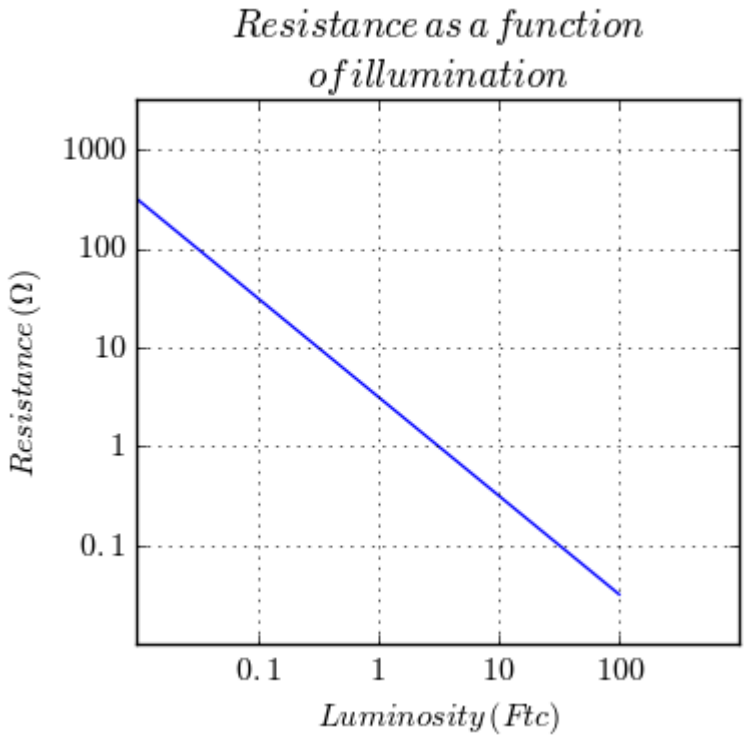
ή αλλιώς (συμβολίζοντας τη διαφορά με *d*-difference):

Sensor 1	**Light 1**	*d11*	*d12*	*d13*	*d14*	…	..	.			
	Light 2	*d21*	*d22*	*d23*	…	..	.				
Sensor 2	**Light 1**	*d31*	*d32*	…	..	.					
	Light 2	*d41*	…	..	.						

Τελικά παίρνουμε τον μέσο όρο κάθε γραμμής (συμβολίζοντάς τον με *m*-mean):

Sensor 1	**Light 1**	*m11*
	Light 2	*m12*
Sensor 2	**Light 1**	*m21*
	Light 2	*m22*

Τι συμβολίζει τελικά ο όρος $\mathbf{m_{xy}}$; Ότι για αλλαγή 10% της φωτεινότητας του λαμπτήρα y, έχουμε κατά μέσο όρο αλλαγή m_{xy} στη μέτρηση που θα λαμβανόταν από τον αισθητήρα x, δεδομένου ότι δε θα συνέβαινε αλλαγή σε άλλο λαμπτήρα. Η χρήση του μέσου όρου βασίζεται στην παραδοχή ότι το LDR αλλάζει τιμή γραμμικά, δηλαδή με σταθερή διαφορά για σταθερή αλλαγή της φωτεινότητας. Πράγματι, το datasheet ενός τυπικού LDR δίνει την σχέση φωτεινότητας-αντίστασης που φαίνεται στο διάγραμμα της εικόνας 4.9.



Εικόνα 4.9 - Σχέση φωτεινότητας-αντίστασης ενός LDR

Το τελευταίο βήμα του σταδίου προρύθμισης είναι να θέσουμε το αρχικό επίπεδο φωτεινότητας, έτσι ώστε να ανταποκρίνεται στο επιθυμητό επίπεδο (threshold) που έχει δώσει ο χρήστης μέσω του GUI, το οποίο έχει αποθηκευτεί σε τοπικό αρχείο. Θα μπορούσαμε να συμβουλευτούμε τους πίνακες που κατασκευάσαμε, αλλά αυτό δεν καταλήγει σε μια συγκεκριμένη τιμή, καθώς οι τιμές ανταποκρίνονται σε κάθε αισθητήρα όταν μόνο ένας

λαμπτήρας είναι σε λειτουργία. Αυτό το πρόβλημα λύνεται ως εξής:

Αν υποθέσουμε ότι θέλουμε μεταβολή A στον λαμπτήρα 1 και B στον λαμπτήρα 2, τότε, μετά από αυτή την αλλαγή, η ανάγνωση που παρέχουν οι αισθητήρες θα υποστεί αλλαγή:

$$\frac{m_{xy}}{10}\cdot A+\frac{m'_{xy}}{10}\cdot B$$

Αυτό σημαίνει ότι θα καταλήξουμε σε ένα γραμμικό σύστημα:

$$\frac{m_{11}}{10}\cdot A \ + \ \frac{m_{12}}{10}\cdot B \ = \ threshold-v_1$$
$$\frac{m_{21}}{10}\cdot A \ + \ \frac{m_{22}}{10}\cdot B \ = \ threshold-v_2$$

Να σημειώσουμε ότι ο όρος $m_{xy}/10$ δείχνει την αλλαγή του επιπέδου φωτεινότητας κατά 1%, καθώς ο όρος m_{xy} δείχνει το ίδιο για 10%. Επίσης, οι όροι $(m_{xy}/10)\cdot A$ και $(m_{xy}/10)\cdot B$ δείχνουν το ποσοστό συνεισφοράς του κάθε λαμπτήρα στην αλλαγή της μέτρησης του κάθε αισθητήρα. Τέλος, οι όροι v_1 και v_2 έχουν τιμές v11=v21 και v31=v41 αντίστοιχα.

Το γραμμικό σύστημα μας δίνει τις τιμές A, B κατά τις οποίες πρέπει να αυξηθεί η φωτεινότητα του κάθε λαμπτήρα. Αν όμως έχουμε θέσει προηγουμένως την φωτεινότητα στο 0%, οι τιμές δείχνουν και την ακριβή φωτεινότητα στην οποία πρέπει να θέσουμε τον κάθε λαμπτήρα, την οποία μπορούμε να την εξάγουμε με απλή γραμμική παρεμβολή, αν δεν είναι κάποια από τις προκαθορισμένες (0, 10, 20... %). Αυτή η γραμμική εξίσωση είναι και αυτή που θα χρησιμοποιηθεί στον αλγόριθμο επιλογής φωτισμού.

4.3 Εργαλεία και δομές ενσωματωμένες στο gateway

Σε αυτό το σημείο θα πρέπει να γίνει μια ανάλυση για τις απαιτήσεις του συστήματος περί software, που παρουσιάστηκαν στο κεφάλαιο 2, αντίστοιχη με αυτή που έγινε για το hardware στο κεφάλαιο 3.

4.3.1 Graphical User Interface (GUI)

Όπως έχει ήδη αναλυθεί σε προηγούμενες ενότητες, το σύστημα θα συνοδεύεται από ένα GUI, το οποίο θα λειτουργεί σαν διεπαφή μεταξύ του συστήματος και του χρήστη. Το GUI προσφέρει όλες τις απαραίτητες πληροφορίες σχετικά με τη λειτουργία του συστήματος, και παράλληλα παρέχει στον χρήστη τη δυνατότητα να αλλάξει κάποιες παραμέτρους του.

Το GUI είναι ο μοναδικός τρόπος με τον οποίο ο χρήστης έχει πρόσβαση στο σύστημα. Αυτό συμβαίνει από τη μια για λόγους ασφαλείας, δηλαδή για να εξασφαλιστεί η λειτουργία του συστήματος από αλλαγές που μπορούν να το καταστήσουν μη λειτουργικό. Από την άλλη, το GUI παρέχει όλες τις απαραίτητες μεταβλητές που χρειάζεται ο χρήστης για να ρυθμίσει τη λειτουργία του συστήματος στα μέτρα του.

Σαν δομή, είναι πλήρως συμβατή με άλλες δομές του gateway:

- Τη βάση δεδομένων (τοπική ή cloud), από την οποία αντλεί, ανάλογα με την χρονική περίοδο που αιτεί ο χρήστης, δεδομένα για την κατανάλωση σε κάθε λαμπτήρα, και τα τοποθετεί σε γράφημα.
- Μια δομή που θα υποβοηθά την real time αναπαράσταση των δεδομένων, και θα αναλυθεί στο επόμενο κεφάλαιο.
- Το filesystem, στο οποίο θα αποθηκεύονται οι ρυθμίσεις που αλλάζουν σε μορφή configuration files, έτσι ώστε να μπορεί ο αλγόριθμος να αλλάζει τη συμπεριφορά του σε κάθε κύκλο ανάλογα με τις προτιμήσεις του χρήστη. Επίσης από το filesystem θα διαβάζει το GUI τις ρυθμίσεις κατά την εκκίνησή του.

Η λειτουργικότητα του GUI βασίζεται στη βασική ιδέα πίσω από τους web servers και συγκεκριμένα τα web apps: τη διαδραστικότητα μεταξύ ενός client side script που ο χρήστης βλέπει στην οθόνη του, που αποτελείται από κώδικα HTML, CSS και jQuery/javascript, και κάποιων server side scripts, τα οποία εκτελούνται έπειτα από αίτηση του χρήστη μέσω του κυρίως script. Αυτά τα server side scripts δέχονται την είσοδό τους σε μορφή HTTP POST ή GET, και επιστρέφουν τις απαραίτητες πληροφορίες, που είναι μετά δουλειά του client side script να παρουσιάσει. Οι πιο ευρέως χρησιμοποιημένες γλώσσες για server side scripting είναι οι PHP, Python και Ruby. Ο λόγος που ακολουθούμε αυτή την πρακτική (αν και δε θα μπορούσε να υλοποιηθεί αλλιώς το GUI), είναι ότι προτιμάται να μην είναι εκτεθειμένη η δομή του filesystem και των databases του server προς τα έξω, αλλά αντιθέτως να γίνονται αυτά προσβάσιμα μέσω των "κρυμμένων" server side scripts, τα οποία εκθέτουν στους clients μόνο τις τελικές πληροφορίες (σε HTML).

4.3.2 Database

Η ύπαρξη βάσης δεδομένων στο gateway δεν χρειάζεται ιδιαίτερη ανάλυση, καθώς είναι απολύτως απαραίτητη για την αναπαράσταση δεδομένων στο GUI. Το database μπορεί να υλοποιηθεί είτε τοπικά στο Raspberry Pi, είτε σε cloud σε κάποιον απομακρυσμένο server.

Το πλεονέκτημα της υλοποίησης του database σε ένα cloud είναι ο σαφώς μεγαλύτερος προσφερόμενος αποθηκευτικός χώρος. Όπως αναλύσαμε στην παράγραφο 3.2, το RPi χρησιμοποιεί κάρτα microSD για την αποθήκευση του OS και όλων των αρχείων, και η ανάπτυξη ενός database στη μνήμη του μπορεί να καταλάβει πολύ αποθηκευτικό χώρο.

Από την άλλη, η υλοποίηση του database στον ίδιο server με το GUI προσφέρει υψηλότερη ταχύτητα στην εξαγωγή των δεδομένων, καθώς δεν υπάρχει εξάρτηση από σύνδεση internet. Ακόμα και αν ο cloud server είναι στο τοπικό δίκτυο, ένα πρόβλημα στο router ή στο ethernet switch μπορεί να έχει ως αποτέλεσμα την αδυναμία λειτουργίας του GUI.

Όσον αφορά το ζήτημα της χωρητικότητας, ένα database όπως αυτό που σκοπεύουμε να υλοποιήσουμε, δηλαδή ένα database που θα περιέχει ελάχιστο αριθμό πεδίων (ημερομηνία, μέτρηση, αύξων αριθμός αισθητήρα κατανάλωσης), δε θα καταλαμβάνει ιδιαίτερα μεγάλο χώρο. Θα χρειαστούν δηλαδή μερικά εκατομμύρια μετρήσεις για να γεμίσει μια κάρτα SD 16GB. Μπορεί βεβαίως να υλοποιηθεί και μια ρουτίνα που θα αποθηκεύει παλαιότερες μετρήσεις σε zip file και θα τις διαγράφει από το database, όπως ακριβώς λειτουργούν τα συστήματα Linux με τα log files.

Στο σύστημα θα χρησιμοποιήσουμε βάση δεδομένων MySQL.

4.3.3 Real time data

Η εύλογη λύση για το πρόβλημα της real time αναπαράστασης δεδομένων είναι μια απλή υλοποίηση message queuing. Ο τρόπος που λειτουργεί το message queuing είναι απλός: σε μια ουρά (queue) στέλνονται μηνύματα από ένα ή πολλά τερματικά. Σε αυτή την ουρά μπορούν να κάνουν subscribe άλλα τερματικά, που μπορούν να παρακολουθούν την κίνηση και τα updates. Ένα παράδειγμα message queuing είναι ένα live twitter feed σε κάποια ιστοσελίδα.

Υπάρχει άπλετος αριθμός δομών και implementations για message queuing. Πολύ δημογιλές είναι το RabbitMQ και το Apache ActiveMQ, καθώς και το StormMQ, το οποίο είναι cloud-based. Αυτές οι δομές λειτουργούν πάνω το HTTP πρωτόκολλο. Εμείς θα επιλέξουμε έναν άλλο τύπο message queuing, το MQTT (MQ Telemetry Transport), ο οποίος είναι πολύ πιο lightweight σε πόρους, μιας και λειτουργεί πάνω σε TCP/IP. Αυτό έχει ως αποτέλεσμα ένα μήνυμα MQTT να είναι σημαντικά μικρότερου μεγέθους από ένα μήνυμα π.χ. RabbitMQ.

Το πρωτόκολλο MQTT χρησιμοποιείται ευρέως σε εφαρμογές Internet of Things (IoT), και καθίσταται πολύ χρήσιμο και στο συγκεκριμένο project. Αποτελεί ίσως τον πιο εύκολο τρόπο δημιουργίας μιας διεπαφής μεταξύ των μετρήσεων κατανάλωσης και του WebGUI, έτσι ώστε να προστεθεί η δυνατότητα της real time αναπαράστασης δεδομένων.

Το πρωτόκολλο αυτό λειτουργεί χρησιμοποιώντας έναν message broker, ο οποίος είναι απαραίτητος, καθώς το MQTT βασίζεται σε διαδικασίες δημοσίευσης-συνδρομής (publish-subscribe). Ο χρήστης μπορεί να κάνει publish κάποια τιμή (αριθμό ή string) σε κάποιο topic, το οποίο περιγράφεται απλά από ένα όνομα. Για να γίνει αυτό δυνατό, πρέπει να έχει πρόσβαση σε έναν MQTT server, τοπικό ή remote.

Έπειτα, οποιοσδήποτε χρήστης (που επίσης έχει πρόσβαση στον ίδιο server), μπορεί μετά από ένα subscription στο συγκεκριμένο topic να διαβάζει το περιεχόμενό του.

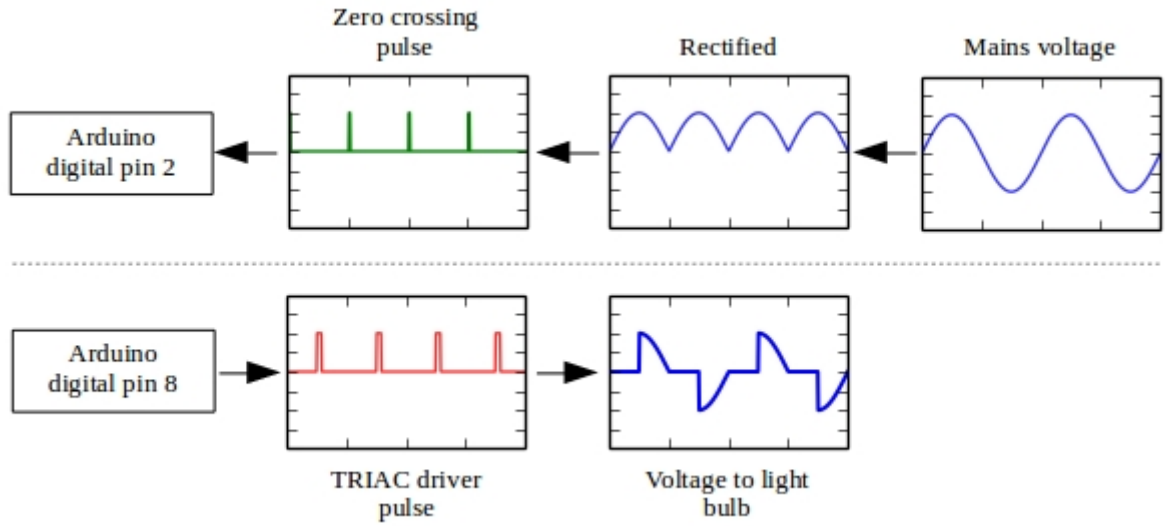
Όλη η λειτουργία ενός MQTT broker μπορεί να περιγραφεί από τις λειτουργίες του: connect, disconnect, publish, subscribe και unsubscribe, οι οποίες είναι αυτονόητες.

Στο σύστημα θα χρησιμοποιήσουμε τον mosquitto MQTT server, ο οποίος "τρέχει" διαρκώς στο Raspberry Pi, και τον Eclipse Paho MQTT Python client, για τη συνδρομή και δημοσίευση μηνυμάτων σε topics.

5.1 Λειτουργία του dimmer

Πριν αναπτύξουμε το κύκλωμα του dimmer προς υλοποίηση, θα πρέπει να γίνει μια εμβάθυνση στην ανάλυσή του, και να παρουσιαστεί ο ρόλος του Arduino στη λειτουργία του.

Επεκτείνοντας την ανάλυση του κυκλώματος της εικόνας 4.7, στην εικόνα 5.1 φαίνεται ένα flow chart με τις κυματομορφές σε κάθε στάδιο λειτουργίας του dimmer, καθώς και τα in/out points του Arduino.



Εικόνα 5.1 - Τα στάδια της λειτουργίας του dimmer

Σε γενικές γραμμές, το Arduino συμβάλλει στα εξής σημεία στη λειτουργία του dimmer:

- Λαμβάνει την έξοδο του κυκλώματος zero cross detection, δηλαδή έναν παλμό κάθε 0.1 sec.
- Τροφοδοτεί έναν παλμό στην άλλη πλευρά του κυκλώματος, ο οποίος χρησιμοποιείται για τη λειτουργία του dimming.
- Πολώνει το opto-coupler στα 5V, χάρη στο 5V pin του.
- Παρέχει τη "γείωση", δηλαδή το Arduino GND, στο opto-coupler και στο opto-TRIAC.

Η λειτουργικότητα του Arduino στο dimmer βασίζεται στη χρήση των ενσωματωμένων timers που προσφέρει, η λειτουργία των οποίων θα παρουσιαστεί συνοπτικά αμέσως.

5.1.1 Arduino AVR timers

Ο μικροελεγκτής Atmega328 είναι εξοπλισμένος με AVR timers διαφορετικών προδιαγραφών. Τα timers λειτουργούν σαν registers, και είναι τα εξής:

- TIMER0: 8-bit timer
- TIMER1: 16-bit timer

- TIMER2: 8-bit timer

Τα bits του timer υποδεικνύουν πόσα βήματα μπορεί να μετρήσει μέχρι τον μηδενισμό του. Αυτό σημαίνει ότι ένα 16-bit timer μπορεί να μετρήσει 2^{16} = 65536 steps από το 0 έως το 65536. Επίσης, έχουν την ιδιότητα να λειτουργούν παράλληλα με τον επεξεργαστή του μικροελεγκτή, χωρίς να επηρεάζουν τη λειτουργία του.

Η έννοια του step είναι αντίστοιχη με έναν κύκλο του clock του μικροελεγκτή. Αυτό σημαίνει ότι ένα step είναι ίσο με 1/16MHz = 0.0000000625s = 62.5ns, δηλαδή πολύ μικρό χρονικό διάστημα. Εδώ εισάγεται η έννοια του prescale που έχουμε αναφέρει, και μπορεί να γίνει κατανοητή με ένα απλό παράδειγμα:

Αν θέλουμε το timer να μετράει περιόδους των 10ms, θα πρέπει να εκτελέσει 10ms/62.5ns − 1 = 160000 − 1 = 159999 κύκλους πριν τον μηδενισμό του (που λέγεται και overflow). Αυτό έχει ως αποτέλεσμα να μη μπορούμε να χρησιμοποιήσουμε 8-bit ούτε 16-bit timers. Ένας prescaler ουσιαστικά έχει ως αποτέλεσμα το timer να μη χρησιμοποιεί κάθε κύκλο του clock, αλλά μια υποδιαίρεση. Αν χρησιμοποιήσουμε έναν 4 prescaler, δηλαδή συχνότητα 16/4 = 4MHz, παίρνουμε περίοδο 250ns=0.00025ms, και χρειάζονται 10/0.00025 − 1 = 39999 βήματα, δηλαδή μπορεί να χρησιμοποιηθεί το 16-bit timer. Αυτό έχει βέβαια ως αποτέλεσμα τη μείωση της ανάλυσης (resolution) από τα 62.5ns στα 250ns.

Τέλος, ένας timer έχει διάφορα registers, τα οποία βοηθούν στη ρύθμιση των timer interrupts. Το TIMER1, συγκεκριμένα, έχει τα εξής registers:

- TCNT (Timer/Counter register): μετράει ticks από το System Clock, τον prescaler ή κάποιο external pin.
- TCCR (Timer/Counter Control Register): θέτει το timer mode και τον prescaler.
- OCR (Output Compare Register): στέλνει ένα interrupt μετά από τον αριθμό των clock ή prescaler ticks που έχουν γραφτεί σε αυτό το register.
- TIMSK (Timer Interrupt Mask Register): ελέγχει ποια interrupts χρησιμοποιούνται, θέτοντας κάποια αντίστοιχα bits στην τιμή του.
- TIFR (Timer Interrupt Flag): ελέγχει ποια interrupts περιμένουν να εκτελεστούν.

5.1.2 Ανάλυση του κώδικα

Ας αναλύσουμε τη λειτουργία του, παραθέτοντας τον αντίστοιχο κώδικα.

Αρχικά θέτουμε κάποιες σταθερές:

```
#define DETECT 0  // interrupt 0 = pin D2
#define GATE   8  // triac gate = pin D8
#define PULSE  3  // triac gate trigger pulse width (timer counts)

// Values determines DARKNESS not brightness.
#define MAX_LEVEL 625 // with 256 prescale, 625 will be 10ms
                      // (one half-wave on 50 hz)
#define MIN_LEVEL 50
```

- DETECT: Ο αύξων αριθμός του pin στο οποίο τροφοδοτείται ο παλμός zc. Στον μικροελεγκτή Atmega328, το pin D2 αντιστοιχεί στο interrupt pin 0.
- GATE: Το digital pin που λειτουργεί ως έξοδος, δηλαδή το pin που στέλνει τον παλμό dimming στο opto-TRIAC.
- PULSE: Πλήθος επαναλήψεων του εσωτερικού timer, για ρύθμιση του πλάτους του παλμού dimming. Το timer μετράει δηλαδή 3 κύκλους από όταν αντιληφθεί τον παλμό zc, κατά τους οποίους θέτει την τάση στο pin D8 ίση με HIGH, και στο πέρας των 3 κύκλων ίση με LOW.
- MAX_LEVEL: Όπως αναφέρει το comment, οι τιμές max και min εξαρτώνται από το prescale του timer, το οποίο θα αναλύσουμε σε λίγο. Για 256 prescale, η τιμή 625 ισοδυναμεί με μέτρηση 0.1s στο timer. Αυτό σημαίνει ότι το timer περιμένει 0.1s από τη λήψη παλμού zc, και έπειτα στέλνει παλμό dimming. Δηλαδή έχουμε επίπεδο dimming ίσο με 0%.
- MIN_LEVEL: Τα ίδια με πάνω, αλλά ο παλμός αποστέλλεται 0.1/50 s μετά από τη λήψη του zc, δηλαδή σχεδόν αμέσως. Άρα έχουμε φωτεινότητα ~100%. Να σημειώσουμε ότι, για να αποφύγουμε το flickering, δε θα θέτουμε τιμές μικρότερες από αυτή. Αυτό έχει ως αποτέλεσμα να παίρνουμε μέγιστη φωτεινότητα λίγο μικρότερη από την κανονική του λαμπτήρα, αλλά τόσο ώστε να μη γίνεται αντιληπτό.

Το επόμενο βήμα είναι να αρχικοποιήσουμε το interrupt pin, τα pins εξόδου, καθώς και τις παραμέτρους του timer:

```
void setup()
{
  attachInterrupt(DETECT, zero_cross_detect, RISING);   // Attach
                              // an Interupt to Pin 2 (interrupt 0)
                              // for Zero Cross Detection
  dim_level = 250;
  //output setup:
  pinMode(GATE, OUTPUT);   // triac gate control
  digitalWrite(GATE, LOW); // turn off triac gate


/* Timer Stuff */
cli();
  TCCR1A = 0x00; // clear
  TCCR1B = 0x00; // clear

  TIMSK1 = 0x03; // Enable the Output Compare A and Overflow
                 interrupt
  OCR1A  = dim_level; // starting with long pause
sei();
}
```

Σε αυτό το κομμάτι κώδικα αρχικοποιούμε ως εξής:

- Με την εντολή `attachInterrupt` ορίζουμε ότι, όταν εντοπιστεί απότομη άνοδος της τάσης (RISING) στο DETECT pin (pin D2 - interrupt pin 0), να "τρέξει" η ρουτίνα που ονομάζεται `zero_cross_detect`.
- Θέτουμε ένα αρχικό επίπεδο έντασης, `dim_level=250`. Η ένταση παίρνει τιμές από το 50

(max) έως το 625 (min).

- Ορίζουμε το GATE pin (pin D8) ως output pin.
- Γράφουμε τάση HIGH στο GATE pin.
- Αρχικοποιούμε τα registers του TIMER1, TCCR1A και TCCR1B.
- Ορίζουμε 2 timer interrupts: το ένα όταν το register TCCR1A έχει ίδια τιμή με το OCR1A, και το άλλο όταν το timer ολοκληρώσει τους κύκλους του (overflow).
- Ορίζουμε την τιμή του register OCR1A ίση με το αρχικό `dim level`.

Έπειτα ορίζουμε 3 ρουτίνες: μια για κάθε timer interrupt, και μία για το `zero_cross_detect`:

```
ISR(TIMER1_OVF_vect) {
  TCCR1B = 0x00;           // stop timer
  digitalWrite(GATE, LOW); // turn off triac gate
}

ISR(TIMER1_COMPA_vect) {
  digitalWrite(GATE, HIGH);  // set triac gate to high
  TCNT1 = 65536-PULSE;       // trigger pulse width
}

void zero_cross_detect() {
   start_low_fix = false;

   TCNT1  = 0;          // reset timer - count from zero
   OCR1A  = dim_level;
   TCCR1B = 0x04;       // start timer with 256 prescale
```

Και τώρα μπορούμε να αναλύσουμε το ρόλο του Arduino στη λειτουργία του dimmer:

- Όταν έρχεται ένας παλμός zc στην είσοδο, το interrupt που έχουμε ορίσει τον αντιλαμβάνεται, και δίνει την εντολή να τρέξει η ρουτίνα `zero_cross_detect`.
- Η εντολή `start_low_fix = false` χρησιμοποιείται μόνο την πρώτη φορά που τρέχει η ρουτίνα, και είναι εκεί απλά για συγχρονισμό των υπόλοιπων εντολών του προγράμματος.
- Μηδενίζεται το timer (TCNT1 = 0), και το OCR1A τίθεται ίσο με `dim_level`.
- Έπειτα ξεκινά η λειτουργία του TIMER1, με 256 prescale (η τιμή του prescale εξαρτάται από τα bits που θέτουμε στο register TCCR1B).
- Όταν η τιμή του timer (TCNT1) γίνει ίση με το OCR1A, δηλαδή όταν φτάσουμε στη στιγμή που πρέπει να στείλει το Arduino τον παλμό στο opto-TRIAC, τότε τρέχει η ρουτίνα `ISR(TIMER1_COMPA_vect)`.
- Εδώ ξεκινά ο παλμός, θέτοντας HIGH το GATE pin, και ξεκινάμε πάλι το timer, αλλά με την τιμή 65536-PULSE. Αυτό σημαίνει ότι ξεκινά με τιμή λίγο πριν το overflow, κατάλληλο setup για να δημιουργηθεί ένας μικρού πλάτους παλμός.
- Όταν συμβεί overflow στο timer, τότε τρέχει η ρουτίνα `ISR(TIMER1_OVF_vect)`.
- Εδώ σταματά η λειτουργία του timer και θέτουμε LOW στο GATE pin, τερματίζοντας έτσι τον παλμό.

Η παραπάνω διαδικασία επαναλαμβάνεται σε κάθε zc, δηλαδή κάθε 10ms.

5.2 Διασύνδεση συσκευών και κατασκευή τερματικών

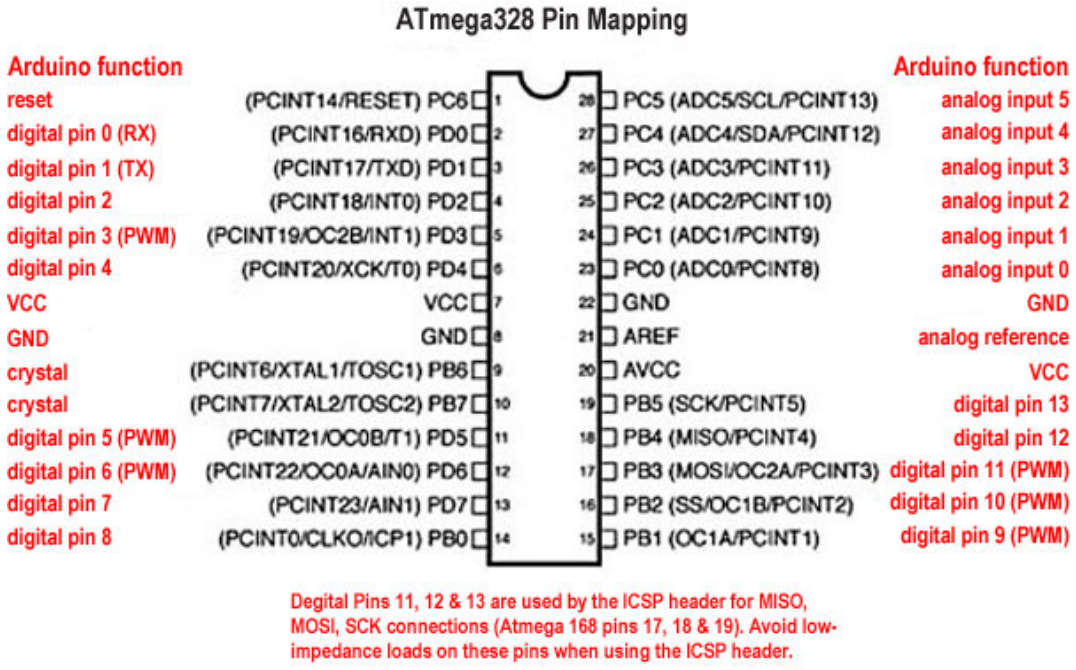
Σε αυτό το κεφάλαιο θα γίνει μια ανάλυση σχετικά με τα pin mappings των συσκευών που χρησιμοποιήθηκαν, ο τρόπος που συνδέονται μεταξύ τους και με τα επιμέρους κυκλώματα της προηγούμενης ενότητας, καθώς και η τελική κατασκευή του κάθε τερματικού.

5.2.1 Pinout

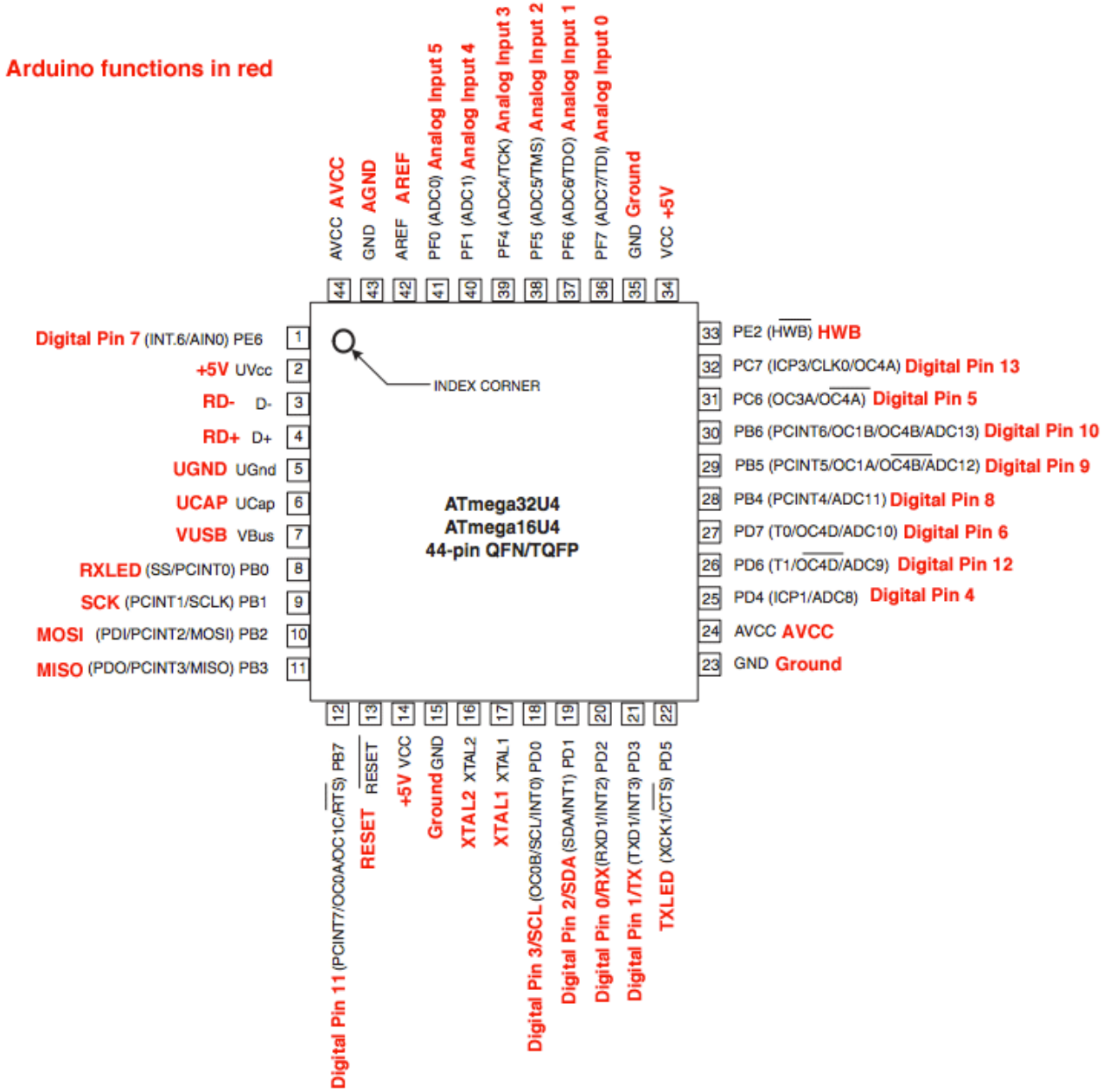
5.2.1.1 Arduino

Στο σύστημά μας χρησιμοποιήσαμε 3 διαφορετικά Arduino: UNO, Micro και Nano, για λόγους που έχουν αναλυθεί στην ενότητα 4. Να σημειωθεί ότι τα Micro και Nano είναι παρόμοια από άποψη μεγέθους και απόδοσης, παρ'όλο που περιλαμβάνουν διαφορετικό μικροελεγκτή, και ο λόγος που χρησιμοποιήθηκαν και τα 2 είναι επειδή προϋπήρχαν από παλαιότερα projects.

Οι μικροελεγκτές που χρησιμοποιούνται από τα Arduino αυτά είναι οι Atmega328 (Uno, Nano) και Atmega32u4 (Micro). Στις εικόνες 5.2 και 5.3 φαίνεται το pin mapping του κάθε μικροελεγκτή σε σχέση με τα φυσικά pins του Arduino.



Εικόνα 5.2 - Atmega328 pin mapping



Εικόνα 5.3 - Atmega32u4 pin mapping

Ας αναλύσουμε επιγραμματικά κάποια σημαντικά χαρακτηριστικά:

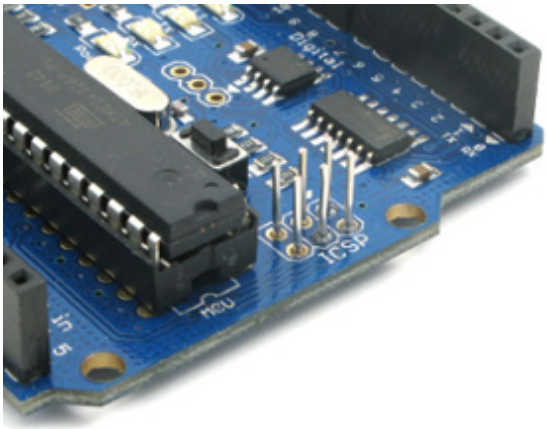
- VCC: Η τάση τροφοδοσίας του μικροελεγκτή, η οποία πρέπει να είναι 5V regulated. Σε περίπτωση τροφοδοσίας του Arduino μέσω USB, η τάση παρέχεται στον μικροελεγκτή απ'ευθείας. Αν χρησιμοποιούμε εξωτερική πηγή 7-20V, η οποία συνδέεται στο VIN pin, τότε η τάση περνά πρώτα από τον ενσωματωμένο στο Arduino 5V regulator. Περαιτέρω, το Arduino παρέχει και ένα pin 3V3, το οποίο μπορεί να τροφοδοτήσει με τάση 3.3V χάρη στον άλλο ενσωματωμένο 3.3V regulator.
- GND: Η "γείωση", δηλαδή η τάση αναφοράς 0V στην τροφοδοσία.
- MISO: Master In Slave Out, πρόκειται για pin που υλοποιεί τη λειτουργία MISO του SPI interface, το οποίο είναι μεγάλο κεφάλαιο και δε θα αναλυθεί εδω. Σε γενικές γραμμές, το SPI χρησιμοποιείται για χειρισμό συμβατών συσκευών, όπως το NRF. Μπορούμε μέσω της κατάλληλης βιβλιοθήκης (spilib) να γράφουμε εντολές σε συγκεκριμένα registers κάθε

συσκευής.

- MOSI: Master Out Slave In, επίσης του SPI interface.
- SCK: Serial Clock του SPI.
- Digital Pins: Χρησιμοποιούνται για ανάγνωση ή εγγραγή τάσης HIGH (VCC) και LOW (GND).
- Analog Pins: Χρησιμοποιούνται για ανάγνωση DC τάσης από τιμή GND έως VCC, την οποία μεταφράζουν σε αριθμό 0-1023. Παρέχουν επίσης τη δυνατότητα εγγραφής παλμού PWM, ρυθμίζοντας το duty cycle σε τιμές 0-255, οι οποίες δίνουν τιμές GND έως VCC σε RMS.

Η διαφορά του μικροελεγκτή Atmega 32u4 με τον Atmega328 είναι η ενσωμάτωση του FTDI chip. Αυτό σημαίνει ότι, ενώ τα Arduino που χρησιμοποιούν τον Atmega328p χρειάζονται επιπλέον ένα ολοκληρωμένο FTDI,δηλαδή ενα μετατροπέα USB to Serial, τα Arduino που περιέχουν τον 32u4 δεν έχουν αυτή την ανάγκη. Ένα side effect φαίνεται στον τρόπο που προγραμματίζεται το Micro: κατά το upload του κώδικα, φαίνεται σαν να αποσυνδέεται από τον υπολογιστή, και έπειτα να ξανασυνδέεται.

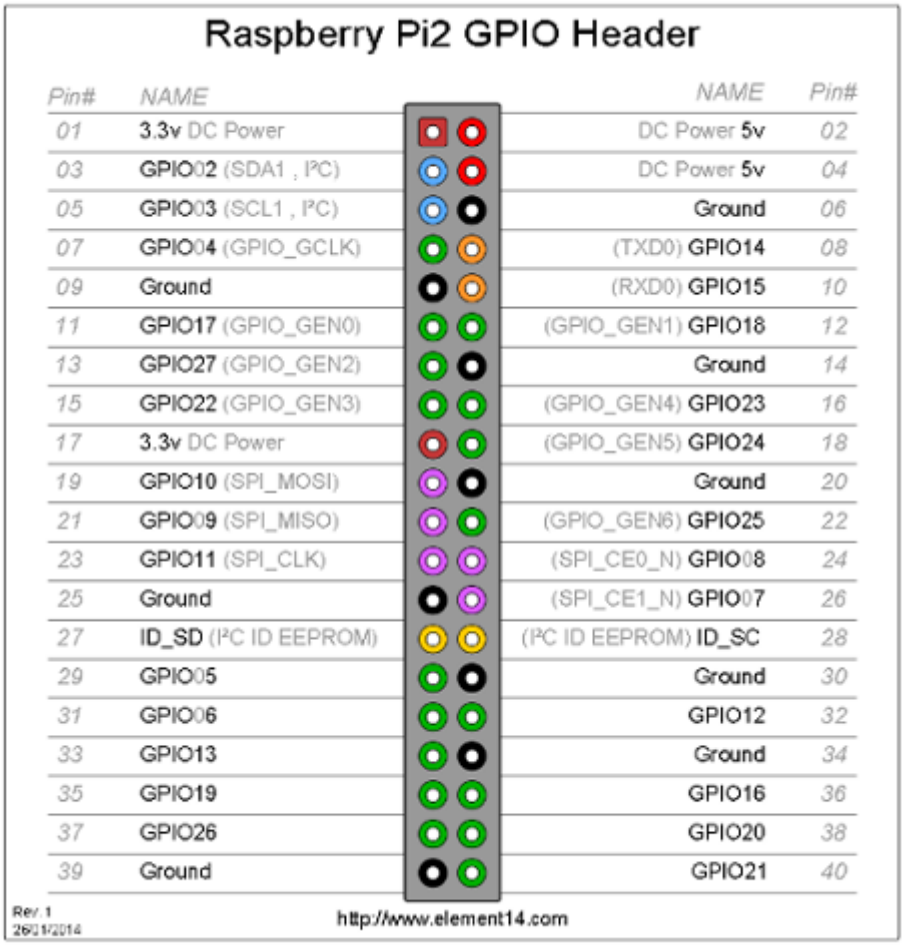
Τέλος να σημειώσουμε ότι το pin mapping του Micro είναι παρόμοιο με αυτό του Nano, με τη μόνη διαφορά ότα τα SPI pins βρίσκονται σε διαφορετικές θέσεις. Έτσι, για να είναι οι πλακέτες των τερματικών επαναχρησιμοποιήσιμες (να μπορούμε δηλαδή να χρησιμοποιούμε στην ίδια πλακέτα το Nano ή το Micro), θα συνδέσουμε με κολλήσεις όλα τα pins εκτός από τα MISO, MOSI και SCK, τα οποία θα συνδέονται μέσω του ICSP header (τα 6 αρσενικά pins στην πάνω μεριά του Arduino, που χρησιμοποιούνται όταν "καίμε" το bootloader σε ένα μικροελεγκτή).



Εικόνα 5.4 - Arduino UNO ICSP header

5.2.1.2 Raspberry Pi 2 Model B

Στην εικόνα 5.5 βλέπουμε ότι το RPi έχει μια πολύ διαφορετική διάταξη, και αυτό οφείλεται στο chip που χρησιμοποιεί (BCM2836). Τα GPIO pins είναι γενικής χρήσης, και διαβάζουν τάσεις HIGH και LOW. Εκτός από αυτά, έχουμε και εδώ τα τυπικά pins VCC και GND (σε διάφορες θέσεις του layout, για να γίνουν πιο εύκολες οι συνδέσεις), καθώς και τα SCK, MOSI και MISO για λειτουργίες SPI. Τέλος, παρέχονται και pins με λειτουργικότητα UART και I2C.



Εικόνα 5.5 - Raspberry Pi 2 Model B pinout

5.2.1.3 NRF24L01, NRF24L01+

NRF pinout (top view)			
NRF24L01		**NRF24L01+**	
VCC	VCC	GND	VCC
CE	CSN	CE	CSN
SCK	MOSI	SCK	MOSI
MISO	IRQ	MISO	IRQ
GND	GND		

Πίνακας 5.1 - NRF24L01 και NRF24L01+ pinout

Από άποψη pinout, η μόνη διαφορά είναι ότι το μοντέλο "+" δεν προσφέρει 2 pins για GND και VCC.

Το NRF είναι ένα παράδειγμα συσκευής SPI. Για την αποστολή - λήψη δεδομένων, πρέπει να γράψουμε κάποια συγκεκριμένα bits σε κάποια συγκεκριμένα registers. Προφανώς, τα pins MOSI, MISO και SCK συνδέονται στα αντίστοιχα pins της συσκευής master.

5.2.2 Διασύνδεση

5.2.2.1 Raspberry Pi – NRF

Η σύνδεση μεταξύ του RPi και του NRF γίνεται ως εξής:

NRF24L01(+)	Raspberry Pi 2 Model B
VCC	3.3V (Pin 17)
GND	GND (Pin 25)
CE	GPIO 25 (Pin 22)
CSN	GPIO 8 (Pin 24)
SCK	GPIO 11 (Pin 23)
MISO	GPIO 9 (Pin 21)
MOSI	GPIO 10 (Pin 19)

Πίνακας 5.2 - Διασύνδεση NRF24L01(+) με RPi 2B

Όσον αφορά το Raspberry Pi, η αρίθμηση των pins ξεκινά από πάνω αριστερά και προχωράει σε κάθε σειρά.

Όπως είδαμε και στο layout, τα pins 19, 21 και 23 είναι τα MISO, MOSI και SCK αντίστοιχα. Επίσης η επιλογή των pins 22 και 24 για τα CE και CSN είναι η πιο σύνηθης, ανταποκρίνεται στο documentation του library και είναι κοντά στα προηγούμενα, άρα και μας εξοικονομεί χώρο στο layout. Για τον ίδιο λόγο επιλέχθηκαν και τα VCC και GND, από τα πολλά που παρέχονται.

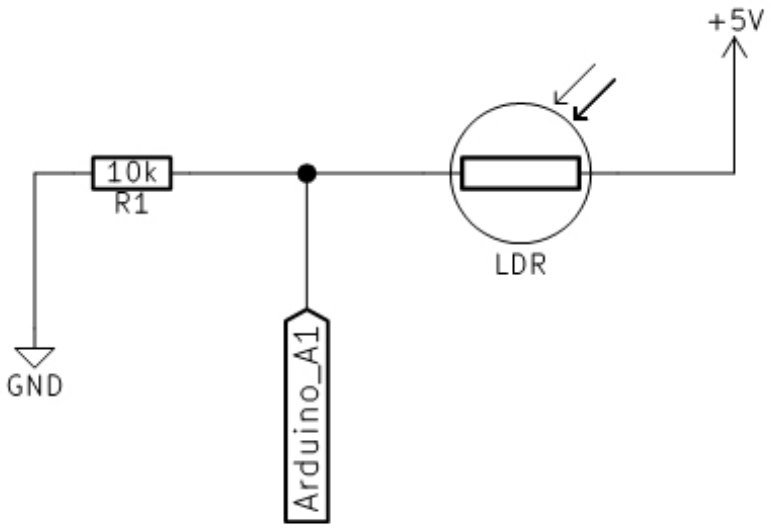
5.2.2.2 Arduino – NRF24L01

NRF24L01(+)	Arduino UNO, Nano	Arduino Micro
VCC	3V3	3V3
GND	GND	GND
CE	D9	D9
CSN	D10	D10
SCK	D13	ICSP3
MISO	D11	ICSP4
MOSI	D12	ICSP1

Πίνακας 5.3 - Διασύνδεση NRF24L01(+) με Arduino

5.2.2.3 Arduino – LDR circuit

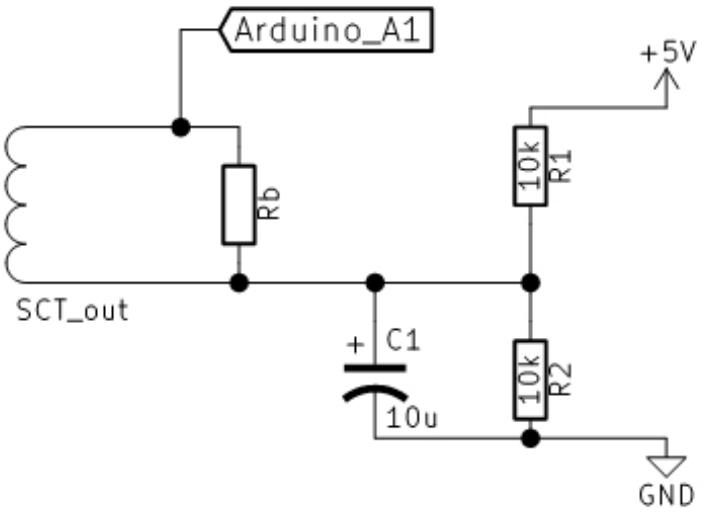
Σύμφωνα με προηγούμενες ενότητες, συνδέουμε το LDR στα 5V του Arduino, την αντίσταση στο GND pin και το κοινό σημείο τους σε ένα analog pin (εδώ το A1). Προκύπτει κύκλωμα της εικόνας 5.6.



Εικόνα 5.6 - Κύκλωμα προσαρμογής LDR

5.2.2.4 Arduino – Power measurement circuit

Επίσης σύμφωνα με την ανάλυση της ενότητας 4.1.1.3, θα χρησιμοποιήσουμε τα 5V και GND pins για την πόλωση, και το A1 pin για τη λήψη των μετρήσεων από τον θετικό πόλο του δευτερεύοντος τυλίγματος του μετασχηματιστή ρεύματος.



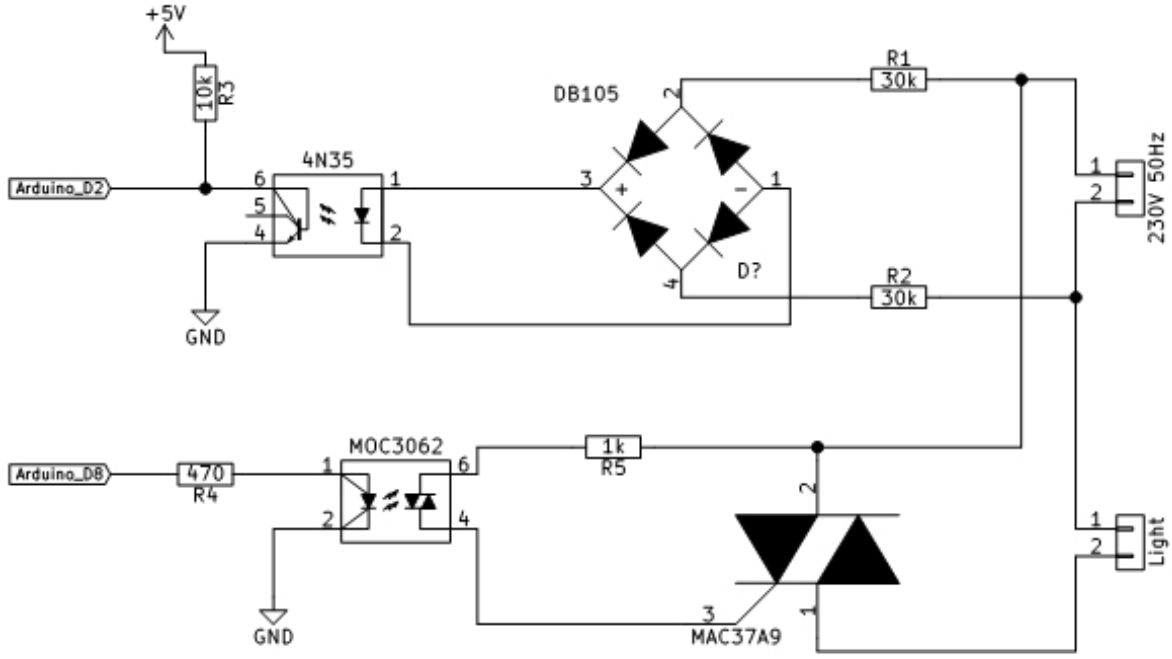
Εικόνα 5.7 - Κύκλωμα προσαρμογής SCT

5.2.2.5 Arduino – Dimmer circuit

Η σύνδεση αυτή αναλύθηκε στην προηγούμενη ενότητα, και παρατίθεται ξανά εδώ.

Arduino	Dimmer circuit
5V	Opto-coupler pin 5 through 10kΩ resistor
GND	Opto-coupler pin 4 Opto-TRIAC pin 2
D2 (interrupt pin 0)	Opto-coupler pin 5
D8 (pulse output)	Opto-TRIAC pin 1 through 470Ω resistor

Πίνακας 5.4 - Διασύνδεση κυκλώματος dimmer με Arduino



Εικόνα 5.8 - Κύκλωμα του dimmer

5.2.3 Κατασκευή τερματικών σε πλακέτες

Για την κατασκευή των τερματικών συσκευών χρειάστηκε να προμηθευτούμε έτοιμες πλακέτες, καλώδιο μονόκλωνο, καλάι, καθώς και κάποια headers (αρσενικά, θηλυκά κ.α).

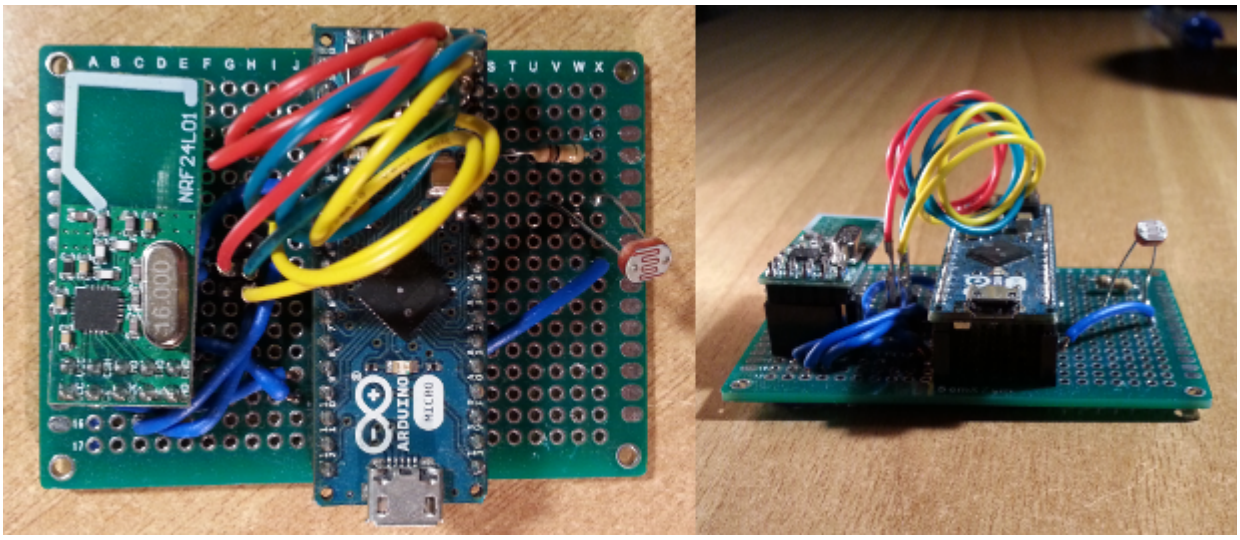
5.2.3.1 Αισθητήρες φωτεινότητας

Οι αισθητήρες φωτεινότητας υλοποιήθηκαν εξ'ολοκλήρου σε έτοιμη πλακέτα 50x70. Τα υλικά που χρησιμοποιήθηκαν στην τελική κατασκευή ήταν τα εξής:

Component	Arduino Micro	NRF24L01	LDR (10kΩ)	10 kΩ resistor	6-pin male header (3x2)
Qty	1	1	1	1	1

Το αρσενικό 3x2 header χρησιμοποιήθηκε για τη σύνδεση του ICSP header με το NRF, έτσι ώστε να είναι εύκολη η αφαίρεση του καλωδίου ανά πάσα στιγμή. Στη συνέχεια παρατίθενται οι

αντίστοιχες εικόνες της κατασκευής.

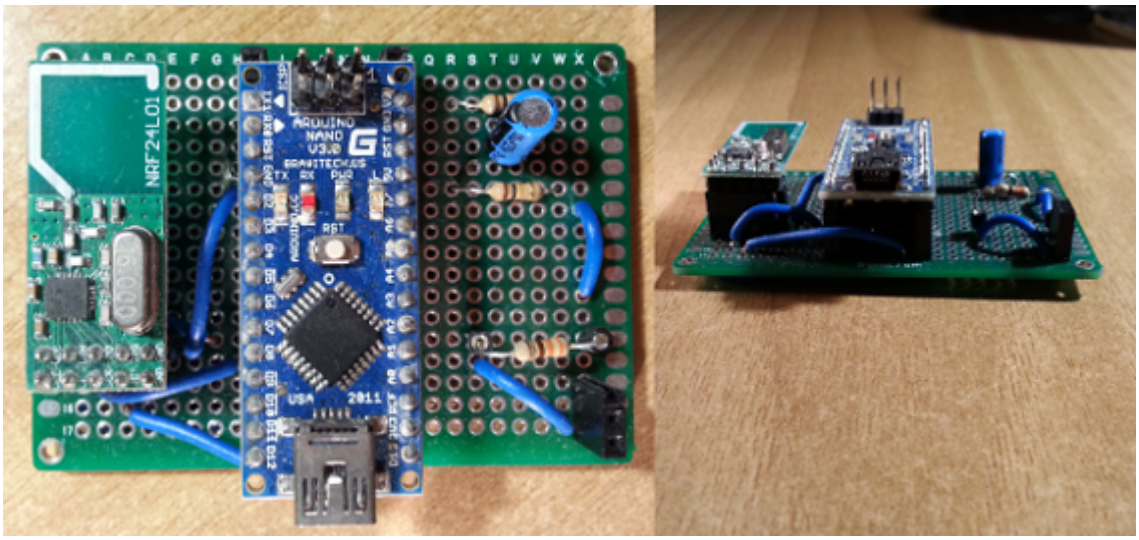


Εικόνα 5.9 - Αισθητήρας φωτισμού

5.2.3.2 Μετρητές κατανάλωσης

Οι μετρητές κατανάλωσης υλοποιήθηκαν επίσης σε πλακέτα 50x70, αλλά όχι εξ'ολοκλήρου. Στην πλακέτα προστέθηκε ένα 2-pin female header, πάνω στο οποίο συνδέεται το SCT. Επίσης, επιλέξαμε να τοποθετήσουμε 2 μονά female headers, πάνω στα οποία προσαρμόζεται το burden resistor, έτσι ώστε να μπορεί να προστεθεί σε περίπτωση που χρησιμοποιηθεί SCT που δεν έχει ενσωματωμένο. Χρησιμοποιήθηκαν τελικά τα εξής υλικά:

Component	Arduino Nano	NRF24L01	10μF capacitor	10kΩ resistor	33Ω resistor (for future use)	2 pin female header	1 pin female header
Qty	1	1	1	2	1	1	2



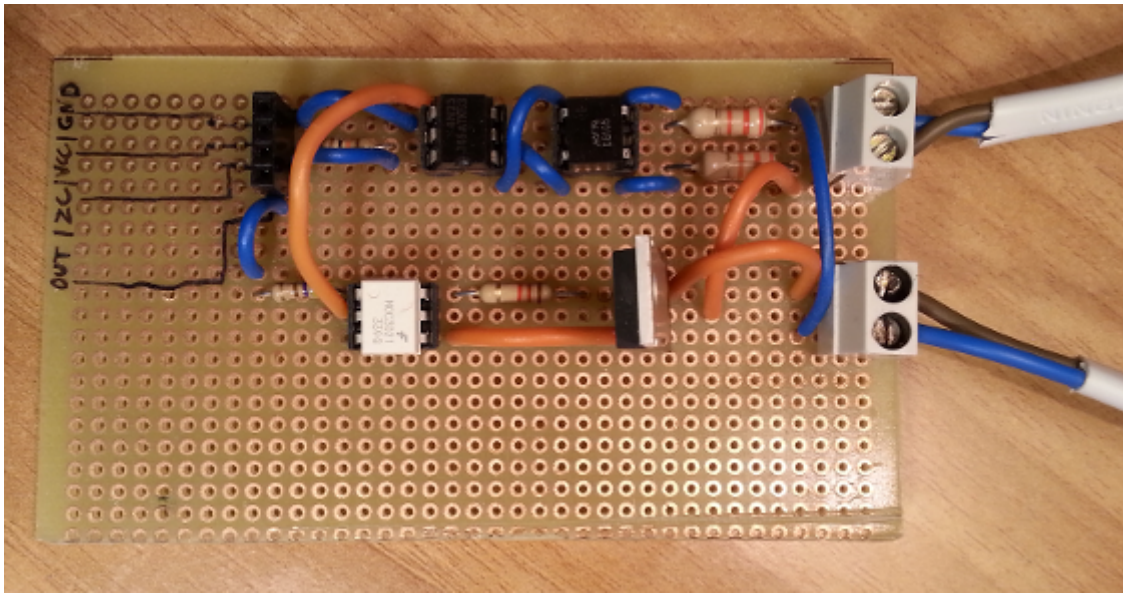
Εικόνα 5.10 - Μετρητής κατανάλωσης

5.2.3.3 Dimmer

Το dimmer υλοποιήθηκε με διαφορετικό τρόπο από τα προηγούμενα τερματικά.

Χρησιμοποιήσαμε μια πλακέτα 10x5 για το κυρίως κύκλωμα, και προσαρμόσαμε ένα 4 pin θηλυκό header για τη σύνδεση με το Arduino. Θα μπορούσαμε να κατασκευάσουμε μια πλακέτα που προσαρμόζεται πάνω στο UNO, αλλά θέλουμε γενικά να αποφύγουμε την επαφή του Arduino με υψηλές τάσεις, γι' αυτό και προσπαθήσαμε την υλοποίηση ενός τερματικού που τοποθετεί το Arduino όσο πιο μακριά γίνεται από τέτοιου είδους επαφή. Το opto-coupler και το opto-TRIAC προσαρμόσθηκαν σε 6-pin IC sockets, και το TRIAC πάνω σε 3-pin θηλυκό header, για εύκολη αλλαγή σε περίπτωση βλάβης. Τέλος, για τη σύνδεση των καλωδίων τροφοδοσίας και λαμπτήρα χρησιμοποιήθηκαν 2-pin terminal blocks με βίδες.

Component	Qty
Arduino UNO	1
NRF24L01	1
4N35 opto-coupler IC	1
MOC3062 opto-TRIAC IC	1
MAC97A8 TRIAC IC	1
DB105 bridge rectifier IC	1
30kΩ, 1/2W resistor	2
1kΩ, 1/2W resistor	1
470Ω resistor	1
2-pin screw terminal block	2
6-pin DIP IC socket	2
3-pin female header	1
4-pin female header	1



Εικόνα 5.11 - Dimmer

5.2.4 Οδηγίες εγκατάστασης

Η εγκατάσταση του συστήματος στον χώρο που θα λειτουργεί είναι αρκετά εύκολη, από άποψη hardware αλλά και software. Απαιτείται ελάχιστη παρέμβαση στις ηλεκτρολογικές

εγκαταστάσεις του χώρου, και το σύστημα προσαρμόζεται στην πρόσθεση αισθητήρων και λαπτήρων αυτόματα. Παρακάτω περιγράφεται η εγκατάσταση κάθε στοιχείου ξεχωριστά.

5.2.4.1 Αισθητήρες φωτεινότητας

Οι αισθητήρες φωτεινότητας αποτελούνται μόνο από την πλακέτα που κατασκευάσαμε στην προηγούμενη ενότητα. Προφανώς, απαιτείται να τοποθετηθούν σε στρατηγικές θέσεις, στις οποίες έχει νόημα να υπάρχει εξάρτηση από την φωτεινότητα. Καλές θέσεις είναι κοντά στον χώρο του γραφείου, κοντά σε ένα κεντρικό τραπέζι σε μια αίθουσα συνεδριάσεων, κλπ. Πρέπει να έχουν "οπτική" επαφή με όλους τους λαμπτήρες του χώρου, καθώς και να βρίσκονται μακριά ο ένας από τον άλλο, έτσι ώστε να υπάρχει καλύτερος έλεγχος της φωτεινότητας στον χώρο.

Οι αισθητήρες φωτεινότητας χρειάζονται ένα επιπλέον καλώδιο για την τροφοδοσία, η οποία μπορεί να γίνει κατ' ευθείαν από έναν υπολογιστή μέσω USB. Αυτό δε συνίσταται, γιατί έτσι θα πρέπει ο υπολογιστής να μένει διαρκώς σε λειτουργία. Εναλλακτικά, μπορούμε να συνδέσουμε τάση 5V regulated στη θύρα USB των Arduino, και με αυτό τον τρόπο να χρειάζεται μόνο μια πρίζα ή ένα πολύπριζο τοίχου.

5.2.4.2 Μετρητές κατανάλωσης

Οι μετρητές κατανάλωσης περιλαμβάνουν ως επιπρόσθετο στοιχείο τον μετασχηματιστή SCT, ο οποίος προσαρμόζεται όπως ένα clamp στο καλώδιο τροφοδοσίας, μεταξύ του dimmer και του αντίστοιχου λαμπτήρα. Αυτό σημαίνει ότι πρέπει να τοποθετηθεί κοντά στον λαμπτήρα, κάτι το οποίο δεν είναι δύσκολο, καθώς οι περισσότεροι χώροι γραφείων έχουν οροφή χωρισμένη σε αφαιρούμενα τμήματα με γυψοσανίδα. Μια άλλη απαίτηση είναι να βρίσκεται εντός του εύρους των NRF, κάτι το οποίο δεν είναι πρόβλημα, καθώς έχουμε αρκετά μεγάλο εύρος για έναν μικρό χώρο γραφείου.

Όσον αφορά την τροφοδοσία, μπορεί να γίνει με τον ίδιο τρόπο. Επίσης μπορεί ο μετασχηματιστής DC που παρέχει τάση στο Arduino να συνδεθεί παράλληλα με το dimmer, στο σημείο που θα γίνει ούτως ή άλλως παρέμβαση στα ηλεκτρολογικά.

5.2.4.3 Dimmer

Το dimmer θα τοποθετηθεί αναγκαστικά στο ίδιο σημείο με τον αντίστοιχο μετρητή κατανάλωσης. Αυτό που πρέπει να προσέξουμε είναι να μην είναι εκτεθειμένα καλώδια ή κολλήσεις, καθώς μπορούν να προκαλέσουν σοβαρό πρόβλημα στις ηλεκτρολογικές εγκαταστάσεις του χώρου, ή ακόμα και κίνδυνο πυρκαγιάς. Μια λύση είναι να τοποθετηθεί σε κάποιο μη αγώγιμο περίβλημα, π.χ. πλαστικό, όπως δηλαδή συμβαίνει με όλα τα dimmer που κυκλοφορούν στην αγορά.

Η τροφοδοσία του Arduino του dimmer γίνεται επίσης μέσω μετασχηματιστή 5V regulated, και συνδέεται παράλληλα στην τροφοδοσία 230V και στην τροφοδοσία του μετασχηματιστή του μετρητή κατανάλωσης.

5.2.4.4 Gateway

Το gateway χρειάζεται συνδεσιμότητα ethernet, γι' αυτό και συνίσταται να τοποθετηθεί κοντά σε router. Επιπρόσθετα, πρέπει βρίσκεται σε σημείο που αποφεύγονται οι παρεμβολές

χώρου στο NRF, καθώς και κοντά σε πρίζα τροφοδοσίας. Η τροφοδοσία, κατά την ανάπτυξη του συστήματος, έγινε με έναν απλό μετασχηματιστή smartphone, στα 5V και 1.5A, χωρίς προβλήματα.

5.2.5 Αλγόριθμος επιλογής φωτισμού

Σε αυτή την ενότητα θα γίνει η τελική ανάλυση του αλγορίθμου που θα καθορίζει την τιμή κατά την οποία θα αλλάζει η ένταση του κάθε λαμπτήρα σε κάθε κύκλο λειτουργίας του συστήματος. Σε αυτό το σημείο, θα υπενθυμίσουμε ότι, κατά το στάδιο setup του συστήματος, έχουμε κατασκευάσει έναν πίνακα, που ονομάσαμε m_list, ο οποίος κρατά τις μέσες τιμές που περιγράφουν την εξάρτηση κάθε αισθητήρα από κάθε λαμπτήρα, και έχει την εξής μορφή:

Sensor 1	**Light 1**	*m11*
	Light 2	*m12*
Sensor 2	**Light 1**	*m21*
	Light 2	*m22*

Να υπενθυμίσουμε επίσης ότι ο όρος m_{xy} συμβολίζει ότι για αλλαγή 10% της φωτεινότητας του λαμπτήρα y, έχουμε κατά μέσο όρο αλλαγή m_{xy} στη μέτρηση που θα λαμβανόταν από τον αισθητήρα x, δεδομένου ότι δε θα συνέβαινε αλλαγή σε άλλο λαμπτήρα. Περαιτέρω, διατηρούμε και μια τιμή m_dim, η οποία είναι αντίστοιχα ο μέσος όρος των διαφορών του πίνακα dim_table (ο πίνακας dim_table περιέχει τις τιμές του dimmer που δίνουν φωτεινότητα 0, 10, 20...100%).

Όταν ληφθεί μια ανάγνωση των τιμών κάθε αισθητήρα φωτεινότητας, θα γίνει ο υπολογισμός των τιμών που θα αποσταλούν στα dimmer, και θα αποθηκευτούν στον πίνακα send_next. Οι τιμές αυτές θα υποδεικνύουν κατά πόσο θα πρέπει να αλλάξει (δηλαδή να αυξηθεί ή να μειωθεί) το επίπεδο της φωτεινότητας του κάθε λαμπτήρα, έτσι ώστε να επιτύχουμε το επιθυμητό threshold, και προκύπτουν από τη γραμμική εξίσωση που επαναλαμβάνουμε εδώ:

$$\frac{m_{11}}{10}\cdot A + \frac{m_{12}}{10}\cdot B = threshold - v_1$$
$$\frac{m_{21}}{10}\cdot A + \frac{m_{22}}{10}\cdot B = threshold - v_2$$

Έτσι, μετά από κάποιες επαναλήψεις του αλγορίθμου, θα φτάσουμε στο επιθυμητό επίπεδο φωτεινότητας. Θα μπορούσαμε εύκολα να το επιτύχουμε αυτό με μία επανάληψη, αλλά αυτό αντιπαρέρχεται στην παραδοχή ότι οι αλλαγές θα είναι ομαλές. Χρησιμοποιώντας μια μέση τιμή m_dim ως "εικονική" τιμή, που περιγράφει αλλαγή 1% της έντασης του φωτισμού, εξασφαλίζεται η ομαλότητα στις αλλαγές της (ο όρος "εικονική" αναφέρεται στο γεγονός ότι, λόγω της ημιτονοειδούς μορφής της τροφοδοσίας του dimmer, δεν είναι δυνατόν να έχουμε μια σταθερή τιμή για κάθε αλλαγή 1% της φωτεινότητας. Γι'αυτό και χρησιμοποιούμε μια μέση τιμή, η οποία όμως δεν επηρεάζει κατά πολύ τη λειτουργία, λόγω της παραδοχής ότι θα έχουμε ούτως ή άλλως ανεπαίσθητες αλλαγές).

Αυτή η ανάλυση όμως έχει ήδη γίνει. Αυτό που αλλάζει, είναι το δεδομένο ότι ο χρήστης θα μπορεί να επιλέξει τον τρόπο λειτουργίας του κάθε λαμπτήρα: αυτόματο, για κανονική λειτουργία, σβηστό, για 0% φωτεινότητα, ή καθορισμένο από τον χρήστη, για 0-100% φωτεινότητα, ανεξάρτητη από τον αλγόριθμο. Προφανώς, θα πρέπει με κάποιον τρόπο να

εξασφαλίσουμε ότι, ακόμα και αν ένας λαμπτήρας δε βρίσκεται σε αυτόματη λειτουργία, το σύστημα θα λειτουργεί σωστά.

Ας υποθέσουμε το εξής σενάριο: σε ένα σύστημα με 2 αισθητήρες φωτός και 2 λαμπτήρες, έχουμε φτάσει σε κατάσταση όπου η μέτρηση από τους αισθητήρες είναι πολύ κοντά στο threshold, και έτσι το σύστημα δεν αλλάζει περαιτέρω την φωτεινότητα. Εκείνη τη στιγμή, ο χρήστης του GUI θέτει την φωτεινότητα ενός από τους 2 λαμπτήρες ίση με 0%, δηλαδή σε κατάσταση 'off'.

Αν δεν αλλάξει καμία παράμετρος του αλγορίθμου, τότε οι αισθητήρες αντιλαμβάνονται μείωση της φωτεινότητας, και ζητούν από τους λαμπτήρες κάποια αύξηση της έντασης, μέσα από τις λύσεις της γραμμικής εξίσωσης που χρησιμοποιείται. Αυτό σημαίνει ότι το σύστημα θεωρεί πως ο λαμπτήρας που θέσαμε στο 0 είναι σε αυτόματη κατάσταση, και μπορεί να αυξήσει την φωτεινότητά του, για να φτάσουμε στο κατάλληλο επίπεδο. Ακόμα χειρότερα, υπάρχει περίπτωση να ζητηθεί αύξηση της φωτεινότητας από τον σβηστό λαμπτήρα τόσο πολύ, που να ζητηθεί από τον άλλο λαμπτήρα μείωση.

Μια λύση είναι η εξής: με κάποιον τρόπο το σύστημα αποθηκεύει την πληροφορία ότι κάποιος λαμπτήρας είναι σβηστός και όχι σε αυτόματη λειτουργία, και έτσι υπολογίζει τη νέα φωτεινότητα αλλάζοντας τις παραμέτρους. Στο προηγούμενο παράδειγμα θα είχαμε, αν ο λαμπτήρας B έχει σβήσει:

$$\frac{m_{11}}{10}\cdot A + \frac{m_{12}}{10}\cdot 0 = threshold - v_1$$
$$\frac{m_{21}}{10}\cdot A + \frac{m_{22}}{10}\cdot 0 = threshold - v_2$$

άρα

$$\frac{m_{11}}{10}\cdot A_1 = threshold - v_1$$
$$\frac{m_{21}}{10}\cdot A_2 = threshold - v_2$$

οπότε παίρνουμε τα αποτελέσματα

$$A_1 = \frac{10\cdot(threshold - v_1)}{m_{11}}$$
$$A_2 = \frac{10\cdot(threshold - v_2)}{m_{21}}$$

Για να πάρουμε κάποια λύση, μιας και A είναι ένας λαμπτήρας, και πρέπει να αποστείλλουμε μια πληροφορία μόνο, χρησιμοποιούμε μέση τιμή με βάρη (weighted average), όπου τα βάρη είναι προφανώς η τιμή κατά την οποία επηρεάζεται κάθε αισθητήρας από αυτό το λαμπτήρα, δηλαδή οι όροι m_{xy}:

$$A = \frac{m_{11}\cdot A_1 + m_{21}\cdot A_2}{m_{11} + m_{21}}$$

Αυτή η προσέγγιση όμως εισάγει άλλο ένα πρόβλημα: αν ο ένας αισθητήρας εντοπίσει πολύ

υψηλή φωτεινότητα, ενώ ο άλλος χαμηλότερη από το threshold, ακόμα και αν το βάρος του πρώτου είναι πολύ χαμηλό για τον λαμπτήρα Α, υπάρχει κίνδυνος να ζητήσει μείωση της φωτεινότητας. Έτσι η φωτεινότητα στο σημείο του δεύτερου αισθητήρα θα μειωθεί ακόμη περισσότερο.

Σε αυτό το σημείο είναι πρέπον να ορίσουμε και την έννοια του threshold. Παρ'όλο που threshold σημαίνει κατώφλι, στην περιπτωσή μας ανταποκρίνεται περισσότερο στην έννοια του κάτω ορίου. Παρ'όλο που το σύστημα αναπτύσσεται με σκοπό την εξοικονόμηση ενέργειας, υπάρχει πάντα ο ανθρώπινος παράγοντας. Αν για παράδειγμα το σύστημα λειτουργεί σε ένα χώρο γραφείου, παρ'όλο που στοχεύει στη βελτιστοποίηση της κατανάλωσης, δε θα πρέπει να επηρεάζει την αισθητική του χώρου όσον αφορά τον φωτισμό. Δηλαδή, αν δε μπορεί να γίνει κάποια βελτιστοποίηση, θα πρέπει να στοχεύουμε στη διατήρηση της φωτεινότητας στο επιθυμητό επίπεδο. Γι'αυτό και το threshold θα πρέπει να ορίζει το χαμηλότερο επίπεδο φωτεινότητας που επιθυμεί ο χρήστης, και όχι το υψηλότερο, όπως ορίζει για παράδειγμα ένας θερμοστάτης.

Η λύση που χρησιμοποιήσαμε είναι η εξής. Όποτε ένας λαμπτήρας αλλάζει από αυτόματη σε ορισμένη από τον χρήστη λειτουργία, γίνεται επεξεργασία των όρων της γραμμικής εξίσωσης, έτσι ώστε να μη λαμβάνεται υπ'όψη ο λαμπτήρας αυτός, αλλά ούτε και ο αισθητήρας που επηρεάζεται περισσότερο από αυτόν. Αν δηλαδή είχαμε το ίδιο παράδειγμα με το προηγούμενο, αλλά με 3 αισθητήρες και 3 λαμπτήρες, εκ των οποίων ο 3ος είναι σε μη αυτόματη λειτουργία, και επηρεάζει περισσότερο τον αισθητήρα 3 (δηλαδή η max τιμή m_{x3} είναι η m_{33}), η γραμμική εξίσωση διαμορφώνεται από:

$$\frac{m_{11}}{10}\cdot A + \frac{m_{12}}{10}\cdot B + \frac{m_{13}}{10}\cdot C = threshold - v_1$$
$$\frac{m_{21}}{10}\cdot A + \frac{m_{22}}{10}\cdot B + \frac{m_{23}}{10}\cdot C = threshold - v_2$$
$$\frac{m_{31}}{10}\cdot A + \frac{m_{32}}{10}\cdot B + \frac{m_{33}}{10}\cdot C = threshold - v_3$$

σε:

$$\frac{m_{11}}{10}\cdot A + \frac{m_{12}}{10}\cdot B = threshold - v_1$$
$$\frac{m_{21}}{10}\cdot A + \frac{m_{22}}{10}\cdot B = threshold - v_2$$

η οποία είναι εύκολα επιλύσιμη.

Η λύση αυτή είναι αποδοτική για τους εξής λόγους:

- Οι λαμπτήρες που λειτουργούν αυτόματα επικεντρώνονται στους αισθητήρες που επηρεάζουν περισσότερο. Αν δε συνέβαινε αυτό, θα είχαμε ανεπιθύμητα αποτελέσματα, όπως αυτό που αναφέραμε προηγουμένως
- Αυτό που αλλάζει δεν είναι ο αλγόριθμος, αλλά οι παράμετροι που του παρέχουμε για να εκτελεστεί
- Μπορούμε να μειώσουμε την κατανάλωση αποστολής δεδομένων, ακυρώνοντας την αποστολή πληροφορίας σε λαμπτήρα, αν πρόκειται να στείλουμε μηδενική αλλαγή

5.3 Λειτουργία συστήματος

Σε αυτή την ενότητα θα γίνει μια διεξοδική ανάλυση της λειτουργίας του συστήματος, συμπεριλαμβανομένου και του setup stage, παρουσιάζοντας παράλληλα και σημαντικά κομμάτια κώδικα από τα Arduino και το RPi. Κατ' αρχάς όμως, ας κάνουμε μια σύντομη αναφορά στο πρωτόκολλο που θα χρησιμοποιηθεί για την επικοινωνία.

5.3.1 Πρωτόκολλο επικοινωνίας

Όλη η λειτουργία του συστήματος βασίζεται στην επιλογή ενός αξιόπιστου και αποδοτικού πρωτοκόλλου επικοινωνίας. Αυτό σημαίνει ότι πρέπει να έχουμε συνέπεια στις επιπλέον πληροφορίες που θα συνοδεύουν το πακέτο με την εκάστοτε μεταδιδόμενη πληροφορία.

Η ανάπτυξη ενός πρωτοκόλλου επικοινωνίας μπορεί να γίνει αρκετά εύκολη, ιδιαίτερα στην περίπτωση ενός συστήματος με λίγους τύπους συσκευών και μικρή βιβλιοθήκη εντολών. Η απαίτηση είναι να συμπεριλάβουμε όλες τις απαραίτητες πληροφορίες, ώστε να μπορεί ο παραλήπτης να αποκωδικοποιήσει το μήνυμα με ακρίβεια, αφού πρώτα ελέγξει ότι προορίζεται γι' αυτόν. Για να συμβεί αυτό θα πρέπει να κρατάμε σε κάθε συσκευή κάποιες σταθερές: τη διεύθυνσή της, τον τύπο κάθε συσκευής που δύναται να είναι σε επικοινωνία μαζί, καθώς και τη βιβλιοθήκη με την αντιστοίχιση των εντολών για κάθε συσκευή.

Στο συγκεκριμένο σύστημα, θα έχουμε 4 τύπους συσκευών: gateway, light sensor, dissipation sensor και dimmer. Η γκάμα εντολών δεν είναι ιδιαίτερα μεγάλη, και, όπως θα αναλύσουμε στη συνέχεια, απαιτούνται περισσότερες από μία εντολές μόνο όταν αποστέλλουμε στο dimmer.

Έχουμε, όπως είναι αναμενόμενο, κάποιους περιορισμούς κατά την ανάπτυξη του πρωτοκόλλου. Ο ένας περιορισμός αφορά το μέγεθος της πληροφορίας, και εισάγεται από τα specifications του NRF24L01(+). Σύμφωνα με το datasheet, το NRF επιτρέπει αποστολή πληροφοριών από 1 έως 32 bytes. Αυτό σημαίνει ότι μπορούμε να αποστείλουμε συνολικά έως και 32 ASCII χαρακτήρες.

Ο επόμενος περιορισμός εισάγεται από τον κώδικα που χρησιμοποιείται για να οδηγήσει το NRF. Όπως έχουμε δει, χρησιμοποιούμε C++ από τη μεριά των τερματικών, και Python από τη μεριά του gateway για την αποστολή και λήψη, καθώς και για την κωδικοποίηση-αποκωδικοποίηση πληροφοριών. Και οι δυο γλώσσες μπορεί να εισαγάγουν αρκετά σφάλματα κατά την εκτέλεση αυτών των διαδικασιών. Από τη μία, η C++ καθιστά λίγο δύσκολη την αποκωδικοποίηση μεγάλων πληροφοριών, και μετατροπή τους από ASCII char σε απλούς αριθμούς. Έχουμε στη διάθεσή μας βέβαια τις ενσωματωμένες συναρτήσεις atol, strtol και strncpy, αλλά και αυτές μπορεί να αποδειχτούν μη αξιόπιστες, σε περίπτωση που η πληροφορία είναι αρκετά μεγάλη και δεν έχουμε ένα καθαρό πρωτόκολλο να ακολουθήσουμε. Από την άλλη, η Python, ενώ καθιστά πολύ εύκολη τη μετατροπή σε int, καθώς και τον τεμαχισμό πληροφοριών, παρουσιάζει κάποιες ασυμβατότητες σχετικά με την έκδοση που χρησιμοποιείται. Για παράδειγμα, η Python 2.7 χρησιμοποιεί κωδικοποίηση ASCII by default, ενώ από την Python 3 αι μετά χρησιμοποιείται unicode. Να σημειωθεί ότι η ανάπτυξη έγινε σε περιβάλλον Python 2.7.

Για να κρατήσουμε το πρωτόκολλο επικοινωνίας απλό και αρκετά βασικό, θα το κατασκευάσουμε ως εξής: η πληροφορία θα ενσωματώνεται σε ένα header, το οποίο θα περιέχει τις διευθύνσεις αποστολέα και παραλήπτη, τον τύπο της συσκευής που αποστέλλει, καθώς και τον τύπο της εντολής που αποστέλλεται. Κάθε πεδίο του header καταλαμβάνει 2 bytes, και

παίρνει τιμές 00-99. Το πεδίο της κυρίως πληροφορίας έχει μέγεθος 4 bytes, οπότε έχουμε συνολικό μέγεθος ενός πακέτου ίσο με 12 bytes. Παρακάτω φαίνεται και μια αναπαράσταση του ενός τυπικού πακέτου.

Source addr (2 bytes)	Dest addr (2 bytes)	Src type (2 bytes)	Command (2 bytes)	Data (4 bytes)

Εικόνα 5.12 - Πρωτόκολλο επικοινωνίας

5.3.2 Arduino

Εδώ θα περιγράψουμε τη λειτουργία κυρίως των αισθητήρων, καθώς και ό,τι παραλείψαμε στην ενότητα 5.1 από την ανάλυση του dimmer.

Κατ' αρχάς, σε όλες τις τερματικές συσκευές χρησιμοποιούμε Arduino με NRF, με τη συνδεσμολογία που περιγράψαμε στην ενότητα 5.2. Για να είναι το NRF χρησιμοποιήσιμο από το Arduino, θα πρέπει να συμπεριλάβουμε την κατάλληλη βιβλιοθήκη εντολών, καθώς και τις κατάλληλες εντολές αρχικοποίησης. Αυτό γίνεται με τον ίδιο τρόπο και στους 3 τύπους συσκευών που περιλαμβάνουν Arduino:

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"

RF24 radio(9,10);
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

//CONFIGURE RADIO
radio.begin();
radio.enableDynamicPayloads();
radio.setAutoAck(false);
radio.setDataRate(RF24_250KBPS);
radio.setPALevel(RF24_PA_MAX);
radio.setChannel(0x4c);
radio.setRetries(5,15);
radio.setCRCLength(RF24_CRC_8);
radio.openWritingPipe(pipes[0]);
radio.openReadingPipe(1,pipes[1]);
//power on radio
radio.powerUp();
radio.startListening();
radio.printDetails();
```

Οι πρώτες 3 γραμμές εξασφαλίζουν την πρόσθεση των βιβλιωθηκών του NRF και του SPI interface στο πρόγραμμά μας. Έπειτα αρχικοποιούμε το NRF. Η εντολή `RF24 radio(9, 10)` υποδεικνύει τη δημιουργία ενός instance του RF24 class, με όνομα radio, και τα CE και CSN pins να είναι τα digital pins 9 και 10 του Arduino αντίστοιχα. Έπειτα δημιουργούμε 2 διευθύνσεις για τους διαύλους επικοινωνίας, που θα χρησιμοποιηθούν σε όλο το σύστημα, για λήψη και

αποστολή.

Οι επόμενες γραμμές κώδικα εμφανίζονται στις περισσότερες εφαρμογές με NRF. Σε γενικές γραμμές, δημιουργούμε το εξής configuration:

- Dynamic payloads: το μέγεθος της πληροφορίας δεν έχει σταθερό μέγεθος, αλλά μπορεί να είναι μεταξύ 1 και 32 bytes.
- Απενεργοποιούμε το auto acknowledgement, καθώς μπορεί να προκαλέσει παρεμβολές και σύγχυση στο σύστημα. Αυτό συμβαίνει γιατί, αν είναι ενεργοποιημένο, όταν μια συσκευή στείλει μια πληροφορία, μπαίνει σε listener mode, και περιμένει επιβεβαίωση από την άλλη συσκευή ότι έλαβε αυτή την πληροφορία. Η επιβεβαίωση όμως αποστέλλεται παντού στο δίκτυο, και μπορεί να τη λάβουν και άλλα NRF, επιβραδύνοντας σημαντικά τη λειτουργία.
- Θέτουμε το data rate στα 250Kbps.
- Θέτουμε το επίπεδο του PA (Power Amplifier, ενισχυτής ισχύος) στο μέγιστο. Αυτή η τιμή, σύμφωνα με το datasheet του NRF, είναι ίση με 0dB. Αυτό εξασφαλίζει καλύτερη ποιότητα αποστολής και μεγαλύτερη πιθανότητα η πληροφορία να μην αλλοιωθεί από παρεμβολές.
- Η εντολή `setChannel` γίνεται κατανοητή αν διαβάσει κανείς τον κώδικα της βιβλιοθήκης, καθώς και το datasheet. Η συγκεκριμένη εντολή θέτει τη συχνότητα του καναλιού, μέσω του RF_CH SPI register, και μέσω της ακόλουθης συνάρτησης: F_0 = 2400 + RF_CH [MHz]. Εδώ, με την hex τιμή 0x4C = 76 [dec], θέτουμε συχνότητα F_0 = 2476 MHz.
- Η εντολή `setCRCLength` επεξεργάζεται το CONFIG register, και έχει να κάνει με το Cyclic Redundancy Check, το οποίο δε θα αναλυθεί εδώ.
- Οι επόμενες 2 εντολές δημιουργούν τους διαύλους για αποστολή και λήψη.
- Τέλος, με την εντολή `powerUp` θέτουμε το NRF σε λειτουργία, με την εντολή startListening το θέτουμε σε listener mode, δηλαδή έτοιμο για λήψη, και με την εντολή `printDetails` εμφανίζουμε κάποιες πληροφορίες, που χρησιμοποιούνται για debugging. Το output μιας τυπικής εντολής `printDetails` φαίνεται στην παρακάτω εικόνα.

```
STATUS           = 0x0e RX_DR=0 TX_DS=0
RX_ADDR_P0-1     = 0xf0f0f0f0e1 0xf0f0f0f0d2
RX_ADDR_P2-5     = 0xc3 0xc4 0xc5 0xc6
TX_ADDR          = 0xf0f0f0f0e1
RX_PW_P0-6       = 0x20 0x20 0x00 0x00 0x00 0x00
EN_AA            = 0x00
EN_RXADDR        = 0x03
RF_CH            = 0x4c
RF_SETUP         = 0x27
CONFIG           = 0x0b
DYNPD/FEATURE    = 0x3f 0x04
Data Rate        = 250KBPS
Model            = nRF24l01+
CRC Length       = 8 bits
PA Power         = PA_HIGH
```

Εικόνα 5.13 - Χαρακτηριστικά NRF

Μετά την αρχικοποίηση του NRF, το Arduino είναι έτοιμο να κάνει ένα σύντομο setup και έπειτα να τρέξει το κυρίως βρόχο, ο οποίος αποτελείται από τα εξής στάδια:

- Αναμονή για λήψη εντολής
- Λήψη εντολής
- Αξιολόγηση της εντολής
- Αν προορίζεται για το ίδιο τερματικό, συνέχιση, αλλιώς επανάληψη του βρόχου

Η η υλοποίηση, καθώς και τα συμφραζόμενα των εντολών αυτών διαφέρουν για κάθε τύπο συσκευής και θα αναλυθούν ξεχωριστά.

5.3.2.1 Αισθητήρας φωτεινότητας

Στον αισθητήρα φωτεινότητας χρειαζόμαστε κατ'αρχάς μια ρουτίνα που θα διαβάζει και θα αποθηκεύει την τιμή που παρέχει το κύκλωμα του LDR. Η ρουτίνα αυτή ονομάζεται readSensor και είναι η εξής:

```
void readSensor() {

  sensorValue = analogRead(sensorPin);
  sensorValue = map(sensorValue, sensorMin, sensorMax, 0, 800);
  sensorValue = constrain(sensorValue, 0, 800);

}
```

Η ρουτίνα αυτή ακολουθεί τα εξής βήματα:

- Λαμβανει την τιμή από το `sensorPin`, το οποίο είναι το Analog Pin στο οποίο συνδέεται το κοινό σημείο LDR και αντίστασης.
- Εκτελεί την εντολή της C++ `map`. Αυτό που κάνει αυτή η εντολή είναι να δέχεται μια ελάχιστη και μια μέγιστη τιμή, να τις αντιστοιχίζει σε κάποιες άλλες επιθυμητές τιμές, και έπειτα όποια ενδιάμεση τιμή δεχτεί να αντιστοιχίζεται αναλογικά. Έτσι, για παράδειγμα, η εντολή `map(5, 0, 10, 0, 20)` δίνει 10. Οι τιμές `sensorMin` και `sensorMax` λαμβάνονται έπειτα από δοκιμές με το συγκεκριμένο κύκλωμα σε ακραίες περιπτώσεις, τις οποίες μετρήσαμε περίπου ίσες με 0 και 1000 αντίστοιχα. Η `map` είναι χρήσιμη έτσι ώστε να μην έχουμε ακανόνιστα μεγάλο εύρος, κάτι που θα χρειαστεί αργότερα για τη ρύθμιση του threshold. Αν δηλαδή χρησιμοποιούσαμε όλο το εύρος, τότε η αλλαγή του threshold κατά 10 μονάδες δε θα είχε καμία διαφορά.
- Εκτελεί την εντολή της C++ `constrain`, η οποία περιορίζει μια τιμή σε κάποια όρια. Αν δηλαδή διαβάσουμε μια τιμή μεγαλύτερη από 1000, η `map` θα την αντιστοιχίσει σε μια τιμή μεγαλύτερη από 800. Με τη χρήση της `constrain`, εξασφαλίζουμε ότι κάθε τιμή θα είναι εντός των ορίων που θέλουμε.
- Η μεταβλητή `sensorValue` έχει δημιουργηθεί στην αρχή του κώδικα, οπότε έχει global scope, και κάθε τοπική αλλαγή της έχει ως αποτέλεσμα να αλλάζει globally, αφού δεν παρέχεται στη ρουτίνα `readSensor` ως argument.

Έπειτα θα χρειαστεί μια ρουτίνα που θα κατασκευάζει το πακέτο προς αποστολή, βάσει της συγκεκριμένης μέτρησης και του πρωτοκόλλου που χρησιμοποιούμε, και θα το αποστέλλει στο gateway:

```
void sendOverRadio() {

  /*****LUMINOSITY*****/
  strncpy(SendPayload, gwAddr, 31);
  strcat(SendPayload, myAddr);
  strcat(SendPayload, "0101010110");
  dtostrf(sensorValue,1,0,SendValue);

  if ((sensorValue>=0) && (sensorValue<10)) {
    strcat(SendPayload, "000");
  }
  else if ((sensorValue>=10) && (sensorValue<100) {
    strcat(SendPayload, "00");
  }
  else if ((sensorValue>=100) && (sensorValue<1000)){
    strcat(SendPayload, "0");
  }

  strcat(SendPayload, SendValue);

  // Stop listening and write to radio
  radio.stopListening();

  digitalWrite(13, HIGH); // only in case of Arduino Micro

  radio.write(&SendPayload,strlen(SendPayload))

  delay(1000); // keep the D13 pin LED on for 1s

  digitalWrite(13, LOW); //only in case of Arduino Micro
}
```

Ας κάνουμε μια επεξήγηση των διεργασιών που εκτελούνται:

- Θα πρέπει να δημιουργήσουμε το πακέτο από την αρχή, και η C++ μας επιτρέπει να ενώσουμε strings με την εντολή `strcat`, από αριστερά προς τα δεξιά. Γι'αυτό και αρχικοποιούμε το string που θα σταλεί (`SendPayload`) με την πρώτη πληροφορία του πρωτοκόλλου επικοινωνίας, τη διεύθυνση του gateway, μέσω της εντολής `strncpy`.
- Έπειτα με την εντολή `strcat` προσθέτουμε τη διεύθυνση του συγκεκριμένου τερματικού.
- Μετά προσθέτουμε το υπόλοιπο header, δηλαδή τα bytes με τον τύπο του αισθητήρα και την εντολή.
- Ανάλογα με το μέγεθος της πληροφορίας, το οποίο είναι μεταξύ 0 και 800, θα πρέπει να ρυθμίσουμε το μέγεθος του SendPayload, το οποίο θέλουμε να είναι σταθερό. Έτσι, αν η τιμή `sensorValue` αποτελείται από 1 ψηφίο, θα προσθέσουμε 3 μηδενικά bytes σαν padding, αν αποτελείται από 2 θα προσθέσουμε 2 bytes, και αν αποτελείται από 3 ψηφία θα προσθέσουμε 1 byte. Έτσι, έχουμε τη δυνατότητα στο μέλλον να επιτρέψουμε την αποστολή πληροφοριών με 4 ψηφία. Κρατάμε δηλαδή 1 byte σαν reserved για το πιθανές μελλοντικές επεκτάσεις.
- Τέλος, προσθέτουμε την τιμή του αισθητήρα, και η συσκευή είναι έτοιμη να αποστείλει την πληροφορία.

Η αποστολή γίνεται ως εξής:

- Με την εντολή `stopListening` θέτουμε το NRF σε "sender mode".
- Η εντολή `write` αποστέλλει την πληροφορία, και απαιτεί σαν arguments τη διεύθυνση μνήμης της (`&SendPayload`) και το μέγεθός της.

Εδώ αξίζει να σημειώσουμε τα εξής: το digital pin 13 των Arduino συνδέεται παράλληλα σε ένα ενσωματωμενο LED, το οποίο ανάβει όποτε έχουμε τιμή HIGH στο D13. Επίσης, συμβαίνει στα UNO και Nano το SCK pin να είναι το pin 13. Έτσι, όταν εκτελείται η εντολή write, κατά την οποία το NRF θέτει το SCK στο HIGH, έχουμε μια επιβεβαίωση ότι γίνεται αποστολή, χωρίς να χρειάζεται να παρακολουθούμε το monitor του Arduino IDE.

Όμως, στο Arduino Micro, το SCK pin είναι μεμονωμένο, και όχι ίδιο με το D13. Έτσι, προσθέτοντας 2 εντολές, για να θέσουμε HIGH και μετά LOW το D13, μπορούμε και με το Arudino Micro να επιβεβαιώνουμε την αποστολή.

Τέλος, οι εντολές που θα τοποθετηθούν στον κυρίως κορμό του προγράμματος, για να εκτελείται κανονικά η λειτουργία του αισθητήρα, είναι οι εξής:

```
void loop() {
while true {
  while (!(radio.available())) {
    delay(10);
  }

  bool done = false;
  while (!done) {
      // Fetch the payload
      uint8_t len = radio.getDynamicPayloadSize();
      done = radio.read( &ReceivePayload, len );
  }

  char destAddr[] = {ReceivePayload[0], ReceivePayload[1], '\0'};
  if (strcmp(destAddr, myAddr) != 0) {
    continue;
  }

  char command_ch[] = {ReceivePayload[2], ReceivePayload[3], '\0'};
  command = atol(command_ch);
  if (command == 1) {
    delay(1000);
    readSensor();
    sendOverRadio();
    delay(20);
  }
  radio.startListening();
  delay(20);
  }
}}
```

Ας επεξηγήσουμε και εδώ τις εντολές που εκτελούνται:

- Κατ'αρχάς, βρισκόμαστε πλέον στην ρουτίνα `loop()` του Arduino, η οποία "τρέχει" διαρκώς, σαν να είχαμε όλο το block του κώδικα μέσα σε έναν έλεγχο `while True`.
- Όσο δεν αντιλαμβάνεται το Arduino ότι του αποστέλλεται κάποια πληροφορία, παραμένει στο while block, με τη συνθήκη `not radio.available()`.
- Όταν αντιληφθεί κάποια πληροφορία, την διαβάζει με το επόμενο block εντολών, και την αποθηκεύει τη μεταβλητή `ReceivePayload`. Ο λόγος που χρειαζόμαστε block και όχι μια εντολή για τη λήψη, είναι ότι τα NRF αποστέλλουν την πληροφορία σε μικρά κομμάτια, και όχι ολόκληρη.
- Έπειτα επεξεργαζόμαστε την πληροφορία, με γνώση βάσει του πρωτοκόλλου ότι τα 2 πρώτα bytes είναι η διεύθυνση του παραλήπτη. Αν δε συμπίπτουν με τη διεύθυνση του Arduino (`myAddr`), τότε με την εντολή continue τερματίζει η πιο πρόσφατη επαναληπτική δομή (δηλαδή η '`while true`'), και επιστρέφουμε στην αρχή της.
- Αν η πληροφορία προορίζεται για μας, τότε, γνωρίζοντας ότι ο τύπος πληροφορίας περιέχεται στα επόμενα 2 bytes, τα συγκρίνουμε με όλες τις δυνατές εντολές που μπορεί να έχουμε ορίσει στο σύστημα. Μέχρι στιγμής ο αισθητήρας λαμβάνει μόνο μια εντολή, με την οποία το gateway ζητάει αποστολή μέτρησης.
- Οι εντολές έπειτα είναι γνωστές: εκτελείται πρώτα η ρουτίνα `readSensor` και έπειτα η `sendOverRadio`, και έτσι έχουμε αποστολή της πληροφορίας στο gateway.

Οι εντολές `delay(n)` έχουν ως αποτέλεσμα να σταματά η εκτέλεση των εντολών για n milliseconds, και χρησιμοποιούνται σε συνδυασμό με τον κώδικα του gateway, έτσι ώστε να έχουμε την κατάλληλη αναμονή και να μη χάνονται πληροφορίες. Ένας τρόπος να χαθεί κάποιο πακέτο είναι να αποστέλλεται σε μια συσκευή η οποία δεν έχει προλάβει να μπει σε listener mode με την εντολή `startListening`.

Τέλος, υπάρχει λόγος που έχουμε τοποθετήσει όλο το block του τελευταίου κώδικα μέσα στην εντολή `while true`. Ενώ το Arduino εξασφαλίζει από μόνο του τη συνεχή επανάληψη του κώδικα που βρίσκεται στη ρουτίνα loop(), αυτή η ρουτίνα δε λαμβάνεται υπ' όψη σαν επαναληπτική δομή από την εντολή continue. Προσθέτοντας ένα είδος "wrapper", δηλαδή την εντολή `while true`, μπορούμε να σταματάμε την εκτέλεση των εντολών όταν φτάσει στην continue, και να ξεκινάμε πάλι από την αρχή του block.

5.3.2.2 Μετρητής κατανάλωσης

Ο μετρητής κατανάλωσης μπορεί να επαναχρησιμοποιήσει τον κώδικα του αισθητήρα φωτεινότητας, καθώς εκτελούν σχεδόν τις ίδιες λειτουργίες. Αυτό που αλλάζει είναι ο τρόπος με τον οποίο λαμβάνεται η μέτρηση από το SCT, δηλαδή η ρουτίνα `readSensor`:

```
void readSensor() {

  sensorValue = emon1.calcIrms(1480);
  sensorValue = sensorValue * 230;

}
```

Και για να λειτουργήσει, χρειάζεται την εξής αρχικοποίηση:

```
#include "EmonLib.h"

EnergyMonitor emon1;

emon1.current(CTpin, calib_val);
```

Η εντολή `emon1.calcIrms(1480)` δίνει μια μέτρηση για την RMS τιμή του ρεύματος. Η τιμή 1480 αφορά το ρυθμό δειγματοληψίας του SCT. Για να αποφευχθούν αποκλίσεις, η βιβλιοθήκη λαμβάνει πολλές τιμές, και με κατάλληλη επεξεργασία δίνει την κατάλληλη μέτρηση. Στην επόμενη γραμμή πολλαπλασιάζουμε την RMS τιμή του ρεύματος με την RMS τιμή της τάσης (230V), για να πάρουμε την ισχύ.

Στο δεύτερο κομμάτι κώδικα, το οποίο τοποθετείται στην αρχή του προγράμματος, συμπεριλαμβάνουμε τη βιβλιοθήκη EmonLib (energy monitor library), η οποία βοηθά πολύ στη λήψη μετρήσεων από το κύκλωμα που έχουμε κατασκευάσει. Έπειτα δημιουργούμε ένα instance του EnergyMonitor class, το οποίο ονομάζουμε emon1. Τέλος, θέτουμε κάποια options. Συγεκριμένα τα options είναι το Arduino analog pin στο οποίο συνδέεται το SCT, και μια τιμή `calib_val`, η οποία εξαρτάται από την αντίσταση R_b. Η βιβλιοθήκη δίνει τιμή 111.1 για αντίσταση R_b=33Ω, και τιμή 60 για τον μετασχηματιστή 30A/1V.

5.3.2.3 Dimmer

Έχουμε ήδη αναλύσει ένα σημαντικό κομμάτι του κώδικα του dimmer. Στη συνέχεια θα αναλυθεί το υπόλοιπο μέρος, που αφορά τη λήψη εντολών από το gateway μέσω του NRF.

```
void loop() {

  if (start_low_fix) // waiting for first zero-crossing
    return;

  if ( radio.available() ) {
    bool done = false;
    while (!done) {
      uint8_t len = radio.getDynamicPayloadSize();
      done = radio.read( &ReceivePayload, len );
    }

    delay(20);

    char destAddr[] = {ReceivePayload[0], ReceivePayload[1], '\0'};
    if (strcmp(destAddr, myAddr) != 0) { action = false; }
    else { action = true; }
    char cmd_char[] = {ReceivePayload[2], ReceivePayload[3],\
                       '\0'};
    char next_level_char[] = {ReceivePayload[4],\
                              ReceivePayload[5],\
                              ReceivePayload[6], '\0'};
    cmd = atol(cmd_char);
    if ((cmd == 3) || (cmd == 4)) {
      incr_val = 0;
      next_level = atol(next_level_char);
```

```
    }
    else if ((cmd == 1) || (cmd == 2)) {
      incr_val = atol(next_level_char);
      next_level = dim_level;
    }
  }

  if ((cmd != 0) && (action)) {
    if (((cmd==1) || (cmd==2)) && incr_val != 0) {
    dim_routime_increment(dim_level, cmd, incr_val);
    }
    else if ((cmd==3) && (dim_level != next_level)) {
      dim_routine_fade_value(dim_level, next_level);
    }
    else if ((cmd==4) && (dim_level != next_level)) {
      dim_routine_value(dim_level, next_level);
    }
    cmd=0;
  }
  strncpy(ReceivePayload, gwAddr, 32);
}
```

Ας κάνουμε μια σύντομη ανάλυση του κώδικα:

- Η ρουτίνα θα ξεκινήσει μόνο όταν λάβουμε το πρώτο zero crossing. Η τιμή της μεταβλητής `start_low_fix` ελέγχεται από τη ρουτίνα `zero_cross_detect` που παρουσιάστηκε προηγουμένως, και, άπαξ και τεθεί σε `false`, μετά δεν επηρεάζεται ξανά.
- Το επόμενο block έχει αναλυθεί και για τον αισθητήρα φωτός, μόνο που εδώ εκτελείται αν έχουμε κάποια πληροφορία στο buffer του NRF. Τότε τι διαβάζουμε, και αποθηκεύεται στο char array που ονομάζεται `ReceivePayload`.
- Έπειτα πρέπει, πάλι σύμφωνα με το πρωτόκολλο επικοινωνίας, να διασπαστεί η ληφθείσα πληροφορία στα απαραίτητα μέρη: τη διεύθυνση του παραλήπτη, τον τύπο της εντολής, και την κυρίως πληροφορία.
- Ο τρόπος για να ελέγξουμε αν θα πρέπει να εκτελέσει κάποια λειτουργία η συγκεκριμένη συσκευή, που συμβαίνει μόνο όταν η πληροφορία προορίζεται γι'αυτή, είναι η boolean μεταβλητή `action`. Αν δε συμπίπτει η διεύθυνση του παραλήπτη με τη διεύθυνση της συσκευής, θέτουμε `action=false`, ειδάλλως `true`.
- Έπειτα αποθηκεύουμε τον τύπο της εντολής και την κυρίως πληροφορία, στις μεταβλητές `cmd` και `next_level`. Οι τύποι των εντολών αναλύονται παρακάτω.
- Έπειτα, με ένα if block, αξιολογούμε την εντολή, και την αντιστοιχίζουμε στην κατάλληλη ρουτίνα που θα τρέξει.

Έχουμε συνολικά 4 εντολές αντιστοιχισμένες στο dimmer, οι οποίες επεξηγούνται στον πίνακα 5.5.

Οι δυο πρώτες εντολές χρησιμοποιούνται στην κανονική λειτουργία, και έχουν ως αποτέλεσμα τη σταδιακή αλλαγή της έντασης φωτισμού κατά μια τιμή. Η τρίτη χρησιμοποιείται στο setup stage, και θέτει σε μια συγκεκριμένη τιμή την ένταση, επίσης σταδιακά. Η τελευταία εντολή έχει το ίδιο αποτέλεσμα με την προηγούμενη, αλλά η αλλαγή της έντασης γίνεται στιγμιαία. Η εντολή αυτή δε χρησιμοποιείται στην κανονική λειτουργία του συστήματος, παρά

μόνο κατά την ανάπτυξη, για έλεγχο των επιπέδων φωτεινότητας και της κατανάλωσης.

Command no.	Command translation	Relevant routine
1	*Increment down by this value, gradually*	`dim_routine_increment`
2	*Increment up by this value, gradually*	`dim_routine_increment`
3	*Dim to this value, gradually*	`dim_routine_fade_value`
4	*Dim to this value, at once*	`dim_routine_value`

Πίνακας 5.5 - Λειτουργίες του dimmer

Χάριν συντομίας, θα παρουσιαστεί μόνο η κυριότερη ρουτίνα, η `dim_routine_increment`. Οι υπόλοιπες λειτουργούν με παρόμοιο τρόπο, και σκοπός είναι να αναλυθεί η φιλοσοφία πίσω από την υλοποίηση του αυτόματου dimming στο σύστημά μας.

```
void dim_routime_increment(int& dim_level,int& cmd,int& incr_val){

  if (cmd == 1) {
    int next = dim_level + incr_val;
    if (next > MAX_LEVEL) {
      next = MAX_LEVEL;
    }
    while (dim_level < next) {
      dim_level++;
      delay(50);
    }
  }

  else if (cmd == 2) {
    int next = dim_level - incr_val;
    if (next < MIN_LEVEL) {
      next = MIN_LEVEL;
    }
    while (dim_level > next) {
      dim_level--;
      delay(50);
    }
  }

}
```

Ο τρόπος που λειτουργεί είναι ο εξής:

- Αρχικά ελέγχουμε τη μεταβλητή `cmd`. Αν είναι ίση με 1, θα πρέπει να μειώσουμε την φωτεινότητα, ενώ αν είναι ίση με 2 να την αυξήσουμε. Ας υποθέσουμε ότι είναι ίση με 1. Να σημειώσουμε ξανά ότι η μείωση της φωτεινότητας επιτυγχάνεται με αύξηση της μεταβλητής `dim_level`, καθώς έτσι αυξάνουμε την περίοδο αναμονής μέχρι να σταλεί ο παλμός ελέγχου για να λειτουργήσει το TRIAC, οπότε καθυστερείται περισσότερο η παροχή ρεύματος στο λαμπτήρα.
- Η μεταβλητή `next` απεικονίζει το επόμενο επίπεδο φωτισμού που θέλουμε, ή καλύτερα

την επόμενη τιμή που θα θέλαμε να έχει η `dim_level`. Για `cmd=1` ισούται δηλαδή με `dim_level+incr_val`, καθώς θέλουμε μείωση της φωτεινότητας, άρα αύξηση του `dim_level`.

- Αν η `next` είναι έξω από τα όρια που θέτουν οι σταθερές `MIN_LEVEL` και `MAX_LEVEL` (στην περίπτωσή μας μόνο η `MAX_LEVEL`), τότε θα πρέπει να γίνει ίση με το αντίστοιχο όριο. Αν δηλαδή είχαμε προηγούμενο `dim_level` ίσο με 590 και το gateway απαιτούσε μείωση της φωτεινότητας κατά 20, τότε η τιμή του `next` θα γινόταν ίση με 610, που είναι έξω από το όριο του `MAX_LEVEL=600`. Τότε θέτουμε `next=MAX_LEVEL=600`.
- Έπειτα αρχίζει η διαδικασία του dimming. Όσο δεν έχει φτάσει η μεταβλητή `dim_level` να γίνει ίση με τη `next`, την αυξάνουμε κατά 1 και περιμένουμε 50ms. Ο χρόνος αναμονής μπορεί να αλλάξει, αλλά βρίσκουμε ότι τα 50ms είναι μια καλή τιμή έτσι ώστε η αλλαγή να είναι ανεπαίσθητη, αλλά και να μην παίρνει υπερβολικά πολλή ώρα να εκτελεστεί.
- Εν τω μεταξύ, επειδή έχουμε περάσει τη `dim_level` "by reference" και όχι "by value", η τιμή της αναφέρεται σε όλο τον κώδικα, δηλαδή globally. Οπότε, επειδή συγχρόνως τρέχει η ρουτίνα `zero_cross_detect`, όποτε αλλάζει η `dim_level`, αλλάζει και το επίπεδο φωτεινότητας!

5.3.3 Gateway (κυρίως πρόγραμμα)

Σε αυτό το σημείο θα ξεκινήσει η παρουσίαση του κυρίως μέρους της διπλωματικής εργασίας. Η υλοποίηση του αλγορίθμου επιλογής φωτισμού, καθώς και η ενσωμάτωσή του στο περιβάλλον της Python ήταν η πιο απαιτητική εργασία κατά την ανάπτυξη του συστήματος, καθώς και αυτή που απαίτησε τις περισσότερες καινοτομίες. Στη συνέχεια, και πριν την παρουσίαση του κώδικα, θα περιγραφούν μερικά προβλήματα που παρουσιάστηκαν, καθώς και η λύση που εφαρμόσθηκε.

5.3.3.1 Εμπόδια πριν τη σχεδίαση του κυρίως προγράμματος

Πρόβλημα 1ο: Αντιστοίχιση κατανάλωσης με τιμές του dimmer

Το πρώτο, και ίσως πιο βασικό πρόβλημα που παρουσιάστηκε είναι το εξής: θέλοντας να έχουμε την επιλογή (στο στάδιο του setup, αλλά και μέσω του GUI) να ρυθμίζουμε τους λαμπτήρες στιγμιαία σε ένα επίπεδο φωτεινότητας, για παράδειγμα 0,10...100%, έπρεπε να βρούμε έναν τρόπο να παίρνουμε με ακρίβεια αυτές τις τιμές από το dimmer. Να θυμίσουμε ότι οι τιμές που παίρνει το dimmer είναι 50 για μέγιστη φωτεινότητα, έως 600 για ελάχιστη, και ανταποκρίνονται στην αναμονή του timer μέχρι να στείλουμε παλμό στο κύκλωμα. Επίσης να τονίσουμε ότι, λόγω της γραμμικής φύσης των λαμπτήρων, το επίπεδο τοις εκατό σε φωτεινότητα, δηλαδή αυτό που φαίνεται, αντιστοιχίζεται και σε επίπεδο τοις εκατό σε κατανάλωση.

Λύση:

Γνωρίζουμε ότι για την ισχύ ενός κυκλώματος ισχύει:

$$P = V_{rms} \cdot I_{rms} = \frac{V_{rms}^2}{R} = I_{rms}^2 \cdot R$$

Αυτός ο υπολογισμός είναι εύκολος όταν έχουμε κανονική ημιτονοειδή κυματομορφή, καθώς

η rms τιμή ενός ημιτόνου ισούται με το max/sqrt(2). Στην περίπτωσή μας όμως έχουμε να μετρήσουμε κυματομορφή στην οποία το ρεύμα και η τάση είναι "κομμένα" ημίτονα. Έτσι ακολουθούμε μια άλλη μέθοδο.

Μια άλλη γνωστή σχέση είναι η εξής, και αφορά στιγμιαίες τιμές:

$$P(t) = V(t) \cdot I(t) = \frac{V^2(t)}{R} = I^2(t) \cdot R$$

Ως αποτέλεσμα, η ισχύς που καταναλώνεται για ένα χρονικό διάστημα t, ξεκινώντας από μηδενική τάση, δίνεται από τη σχέση:

$$\int_0^t P(t)dt = \int_0^t (V(t) \cdot I(t))dt = \int_0^t \frac{V^2(t)}{R}dt = \int_0^t (I^2(t) \cdot R)dt$$

Αυτό που ψάχνουμε είναι μια αντιστοίχιση των τιμών 0,10...10% στις τιμές 600...50 του dimmer. Ξέρουμε ότι σε ένα ολόκληρο ημίτονο αντιστοιχίζεται το 50, και σε μηδενική ισχύ το 600. Από εκεί και πέρα, αν υπολογίσουμε μια υποθετική τιμή της ισχύος Pmax για ολόκληρο το ημίτονο, τότε εξετάζουμε κάθε τιμή κατά την οποία θα πρέπει να "κοπεί" αυτό το ημίτονο, έτσι ώστε να πάρουμε τις τιμές Pmax*0.1, Pmax*0.2 κλπ, ή ακόμα καλύτερα τις τιμές Pmax*0.01, Pmax*0.02 κλπ, για μεγαλύτερη ακρίβεια. Αυτό το καταφέραμε ακολουθώντας τακτική brute-forcing, δηλαδή παίρνοντας ένα-ένα ημίτονα κομμένα κατά μια τιμή, και συγκρίνοντας το ολοκλήρωμά τους για μια περίοδο με αυτό του κανονικού ημιτόνου. Η τιμή που βρίσκουμε αντιστοιχίζεται έπειτα σε τιμή κατάλληλη για το dimmer με απλή γραμμική παρεμβολή.

Η διαδικασία αυτή υλοποιήθηκε μέσω ενός Python script που παρατίθεται στο τέλος του βιβλίου, και δίνει τα εξής αποτελέσματα:

Power	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Dimmer value	*600*	*457*	*415*	*381*	*352*	*325*	*297*	*268*	*234*	*192*	*50*

Να σημειωθεί ότι η τιμή της αντίστασης ενός λαμπτήρα βρίσκεται εύκολα δεδομένου της ονομαστικής του κατανάλωσης, και της τάσης τροφοδοσίας:

$$R = \frac{V_{mains,rms}^2}{P_{D,max}}$$

Πρόβλημα 2ο: Σφάλμα μετρήσεων κατανάλωσης

Όπως είναι αναμενόμενο, το SCT εισάγει ένα ποσοστό θορύβου στις μετρήσεις. Για παράδειγμα, ο επόμενος πίνακας παρουσιάζει τις μετρήσεις που λαμβάνουμε, καθώς και τις αναμενόμενες, για έναν λαμπτήρα πυρακτώσεως 40W.

Light intensity	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Measured power (W)	*11*	*22.5*	*27.2*	*31.33*	*34.5*	*37.3*	*39.45*	*41.35*	*42.5*	*45.35*	*47*
Expected power (W)	0	4	8	12	16	20	24	28	32	36	40

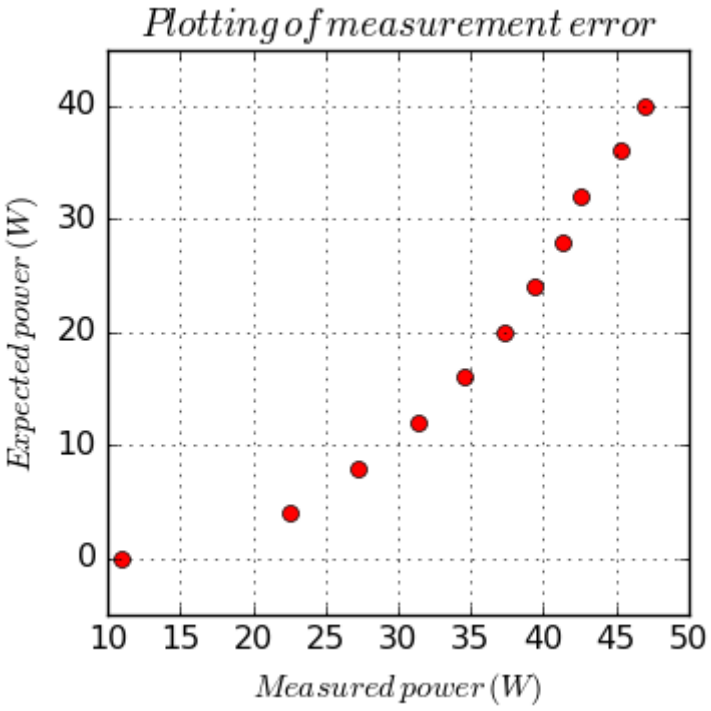
Ο λόγος που εισάγεται αυτός ο θόρυβος έχει να κάνει με τα εσωτερικά χαρακτηριστικά του SCT, καθώς και με τον τύπο ρεύματος που μετράμε: η μέτρηση τεμαχισμένων κυματομορφών είναι πολύ πιο επίπονη από τη μέτρηση απλών ημιτόνων, καθώς, όπως έχουμε προαναφέρει, καθίσταται πολύ δυσκολότερη η μετατροπή σε RMS τιμή. Αυτές οι μετρήσεις εξαρτώνται επίσης από την τροφοδοσία του κυκλώματος οδήγησης του SCT, όπως θα αναλύσουμε στο επόμενο πρόβλημα.

Ας πάρουμε όμως ως παράδειγμα τις συγκεκριμένες μετρήσεις. Παρατηρούμε ότι όσο αυξάνεται η ισχύς, τόσο μικρότερη είναι και η απόκλιση από την αναμενόμενη μέτρηση. Αυτό είναι κάτι το αναμενόμενο, καθώς το σήμα που επάγεται στα τυλίγματα του μετασχηματιστή ρεύματος του SCT υπερτίθεται του θορύβου, από κάποια τιμή και πάνω. Δηλαδή, ο σηματοθορυβικός λόγος (SNR, Signal to Noise Ratio) μειώνεται, όσο αυξάνεται η μέτρηση.

Το πρόβλημα, τελικά, είναι να βρούμε κάποια σχέση, ή καλύτερα κάποια συνάρτηση, στην οποία, όταν εφαρμόζουμε τη ληφθείσα μέτρηση, να μας δίνει την πραγματική ισχύ, ακόμα και για εντάσεις φωτισμού όχι πολλαπλάσιες του 10%.

Λύση:

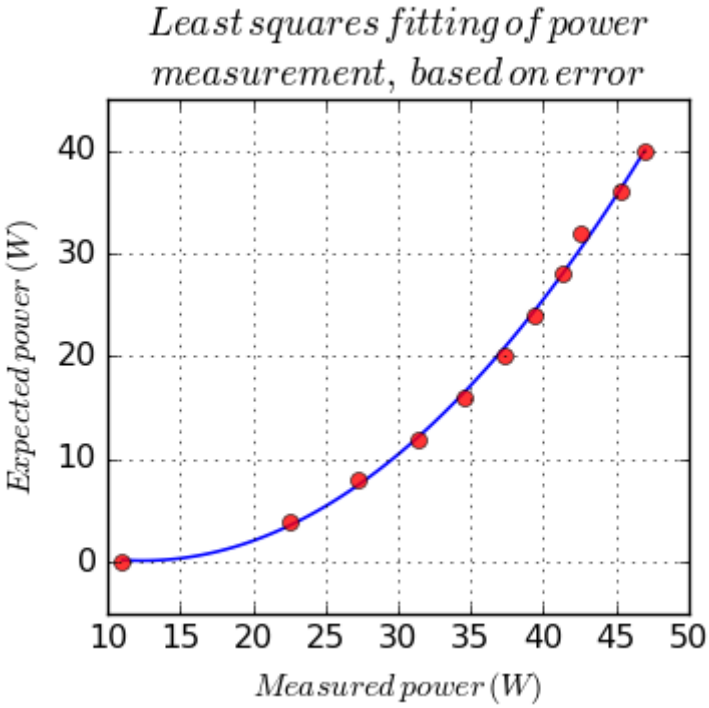
Η λύση παρέχεται αν παρατηρήσουμε τη γραφική αναπαράσταση της διασποράς του συνόλου των σημείων (x, y) = (ισχύς που μετρήθηκε, αναμενόμενη ισχύς):



Εικόνα 5.14 - Ληφθείσα-αναμενόμενη μέτρηση κατανάλωσης

Όπως βλέπουμε, είναι εύκολο να ενταχθούν τα σημεία σε μια καμπύλη, κατά πάσα πιθανότητα δευτέρου βαθμού, η οποία θα έχει όσο δυνατόν ελάχιστες αποστάσεις από αυτά τα σημεία, και θα μπορεί να αντιστοιχίσει ενδιάμεσες τιμές. Όντως, αν εφαρμόσουμε τον αλγόριθμο που είναι γνωστός ως ελάχιστα τετράγωνα, δηλαδή ένα είδος interpolation δευτέρου βαθμού,

παίρνουμε το εξής αποτέλεσμα:



Εικόνα 5.15 - Γραμμική παλινδρόμηση στα δεδομένα της εικόνας 5.14

Η καμπύλη αυτή, καθώς και όλοι οι απαραίτητοι υπολογισμοί, συναρτήσεις και μετατροπές έγιναν χάρη σε ένα άλλο Python script, με χρήση επιστημονικών εργαλείων όπως τα numpy, scipy, και matplotlib για κατασκευή γραφικών παραστάσεων. Συγκεκριμένα χρησιμοποιήσαμε το function `scipy.interpolate.interp1d`, μέσω του οποίου γίνεται interpolation σε 1 διάσταση.

Έτσι, μπορούμε να "περνάμε" κάθε ληφθείσα μέτρηση από αυτή τη συνάρτηση, για να πάρουμε την πραγματική ισχύ.

Να σημειωθεί ότι αυτή η υλοποίηση είναι μια μορφή μηχανικής μάθησης, και πιο συγκεκριμένα supervised machine learning. Παρέχουμε ένα σύνολο μετρήσεων, που αποτελούν τα training data, και το πρόγραμμα αποφασίζει τη καλύτερη δυνατή συνάρτηση, μέσω της οποίας μπορούμε να έχουμε μια εκτίμηση για το αποτέλεσμα όταν παρέχουμε τιμές που δεν υπάρχουν στα training data. Ένας κανονικός αλγόριθμος machine learning χρησιμοποιεί γραμμική παλινδρόμηση (linear regression), και εξάγει το αποτέλεσμα δημιουργώντας ένα cost function, το οποίο περιγράφει το σφάλμα μεταξύ μιας τυχαίας πρόβλεψης, και της τιμής που δίνουν τα training data:

$$J(\theta) = \frac{1}{m} \cdot \sum_{1}^{m} (f(x_i) - y_i)^2$$

Όπου *f(x)* είναι η συνάρτηση που δίνει τις προβλέψεις, και συνήθως είναι ένα πολυώνυμο βαθμού ίδιο με τον βαθμό της παλινδρόμησης (στην περίπτωσή μας δευτέρου βαθμού), και m το πλήθος των training data.

Το τελικό αποτέλεσμα για τις τιμές θ, οι οποίες είναι οι συντελεστές του πολυωνύμου $f(x)$, θα βρεθεί με την επανάληψη:

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} \cdot J(\theta)$$

για κάθε θ, μέχρι η σχέση αυτή να συγκλίνει. Αυτή η επαναληπτική διαδικασία ονομάζεται gradient descent. Ο αριθμός α καθορίζει το ποσοστό της μεταβολής του θ, και συνήθως επιλέγεται έτσι ώστε να είναι επαρκώς μικρός.

Γενικά, τα ελάχιστα τετράγωνα σε αυτή την περίπτωση δίνουν ακριβώς το ίδιο αποτέλεσμα, και αποτελούν ευκολότερη λύση, μιας και παρέχονται άμεσα από τις βιβλιοθήκες numpy και scipy της Python.

Πρόβλημα 3ο: Συνέπεια μετρήσεων κατανάλωσης

Για να προσθέσουμε στην προηγούμενη ανάλυση, θεωρούμε ότι η απόδοση του SCT σε συνδυασμό με ένα Arduino εξαρτάται από πολλές παραμέτρους, από τις οποίες οι σημαντικότερες είναι:

- Η ποιότητα του καλωδίου στο οποίο διεξάγεται η μέτρηση
- Τα εσωτερικά χαρακτηριστικά, απώλειες κλπ του SCT
- Το κύκλωμα προσαρμογής του SCT στο Arduino
- Το ρεύμα τροφοδοσίας του Arduino

Η τελευταία παράμετρος είναι και αυτή που επηρεάζει τις μετρήσεις περισσότερο. Για παράδειγμα, κατά την ανάπτυξη του συστήματος, όταν δηλαδή δεν υπήρχε NRF στο κύκλωμα της μέτρησης, οι μετρήσεις ήταν κατά πολύ διαφορετικές από αυτές που πήραμε από το τελικό κύκλωμα. Αυτό οφείλεται στο ρεύμα που τραβάει το NRF από το Arduino. Οι μετρήσεις έχουν επίσης μεγάλη απόκλιση για διαφορετικές πηγές τροφοδοσίας του Arduino, δηλαδή εξαρτώνται από το αν θα συνδέεται σε USB θύρα υπολογιστή, σε USB hub, σε μετασχηματιστή, και επηρεάζεται ακόμα και από το αν ο υπολογιστής που τροφοδοτεί το Arduino λειτουργεί με μετασχηματιστή ή με μπαταρία.

Λύση:

Η λύση είναι η ίδια που εφαρμόσαμε με τους αισθητήρες φωτεινότητας. Επειδή δε θέλουμε να έχουμε hard-coded δεδομένα στον κώδικα του συστήματος, έτσι ώστε να μπορεί να αλλάξει η τοπολογία του και τα χαρακτηριστικά του, επεξεργαζόμαστε το στάδιο της ρύθμισης, έτσι ώστε, εκτός από τους αισθητήρες φωτεινότητας, να αποθηκεύουμε και μετρήσεις από τα SCT. Αυτές οι μετρήσεις έπειτα τροφοδοτούνται στο script του προηγούμενου προβλήματος, και κατασκευάζεται η κατάλληλη συνάρτηση.

Να σημειώσουμε ότι, παρ'όλο που δεν είναι ιδιαίτερα δύσκολη η υλοποίηση αυτής της ιδέας, δεν την εφαρμόσαμε στο σύστημα, για τους εξής λόγους:

- Χρησιμοποιήσαμε σταθερή πηγή τροφοδοσίας για τα κυκλώματα μέτρησης κατανάλωσης, οπότε δεν είχαμε απόκλιση στις μέτρησης, και χρησιμοποιήθηκε ένα σετ μετρήσεων για τα ελάχιστα τετράγωνα.

- Η προσθήκη του setup των SCT στο setup stage το επιβραδύνει σημαντικά, κάτι που δε βοηθά κατά τη διεξαγωγή των tests του συστήματος κατά την ανάπτυξη. Εξ' άλλου, σε αντίθεση με τη διαδικασία του setup των αισθητήρων φωτεινότητας, ένα setup για τα SCT θα είχε αναμενόμενα αποτελέσματα.

5.3.3.2 Εμπόδια κατά τη σχεδίαση του κυρίως προγράμματος

Τα προηγούμενα προβλήματα αφορούσαν τη λειτουργία του συστήματος, και η λύση τους είναι στατική, εκτελείται πριν την κανονική λειτουργία, και αποθηκεύεται σαν configuration. Στη συνέχεια παρουσιάζονται κάποια προβλήματα που εντοπίστηκαν κατά την κανονική λειτουργία, και απαίτησαν πιο προσαρμόσιμες λύσεις, οι οποίες αφορούν περισσότερο τον κώδικα.

Πρόβλημα 4ο: Επικοινωνία του GUI με το κυρίως module

Το πρώτο πρόβλημα που εμφανίστηκε κατά την ανάπτυξη του κώδικα του gateway, ήταν να γίνει δυνατή η επικοινωνία του GUI με το κυρίως module, δηλαδή να μπορούν οι εντολές που στέλνει ο χρήστης να έχουν αντίκτυπο στη ροή του προγράμματος, και να ρυθμίζεται ανάλογα η ένταση του φωτισμού.

Η πρώτη λύση που υλοποιήθηκε ήταν, όποτε υπάρχει κάποια αλλαγή στις ρυθμίσεις του gateway, να γράφεται ένα αρχείο (π.χ. σε μορφή json) με όλες τις απαραίτητες πληροφορίες: την κατάσταση του κάθε λαμπτήρα, την έντασή του, αν την έχει θέσει ο χρήστης, και το επιθυμητό επίπεδο (threshold). Αυτό αποδείχτηκε πολύ εύκολα υλοποιήσιμο: ο χρήστης επιλέγει τις επιθυμητές ρυθμίσεις, και όταν τις υποβάλλει, αυτές προωθούνται σε ένα server side script (το οποίο γράφτηκε σε Python), σε μορφή http GET ή POST request. Έπειτα αυτό το script συγκρίνει το αρχείο και, αν εντοπίσει διαφορές, εκτελεί overwrite με το νέο configuration.

Έπειτα μπορεί το κυρίως πρόγραμμα να διαβάζει ανά κάποια χρονικά διαστήματα αυτό το αρχείο, και, αν εντοπίσει κάποια αλλαγή σε σχέση με την προηγούμενη ανάγνωση, να δρα κατάλληλα.

Σύμφωνα όμως με τις υποθέσεις μας, το κυρίως πρόγραμμα θα τρέχει περιοδικά, κάθε περίπου 30 δευτερόλεπτα ή και παραπάνω. Το ζητούμενο της ύπαρξης του GUI είναι ο χρήστης να μπορεί να "παραβλέπει" τη λειτουργία του προγράμματος, και να επιβάλλει κάποια ρύθμιση στον φωτισμό. Ακόμα και αν τοποθετήσουμε πολλές αναγνώσεις μέσα σε κάθε επανάληψη, αυτό καθιστά το πρόγραμμα δύσχρηστο, και αντιτίθεται σε βασικές αρχές προγραμματισμού (DRY – Don't Repeat Yourself).

Λύση:

Η λύση γενικά έχει ως εξής: μαζί με το πρόγραμμα, τρέχει παράλληλα ένα δεύτερο, το οποίο παρακολουθεί την κατάσταση αυτού του αρχείου. Όταν εντοπίσει αλλαγή, ειδοποιεί το κυρίως module, και αυτό κανονίζει περαιτέρω ενέργειες.

Η λύση αυτή μπορεί να υλοποιηθεί με αρκετούς τρόπους σε Python. Μπορεί να χρησιμοποιηθεί παράλληλη εκτέλεση χάρη στο *multiprocessing* module, ασύγχρονη επικοινωνία χάρη στην *asyncio* βιβλιοθήκη της Python, και πολυνηματική εκτέλεση μέσω του *threading* module. Η λύση που επιλέχθηκε είναι η τελευταία, καθώς διατίθεται η βιβλιοθήκη ***watchdog***, που δημιουργεί ένα δεύτερο thread, το οποίο δρα σαν observer για ένα αρχείο ή ένα φάκελο.

Η υλοποίηση ενός watchdog είναι πολύ εύκολη, καθώς απαιτεί τη δημιουργία μιας κλάσης που κληρονομεί την κλάση *FileSystemEventHandler* και ορίζει μια μέθοδο `run`, που εκτελείται όποτε εντοπιστεί κάποια αλλαγή. Έπειτα δημιουργείται ένα instance της κλάσης *Observer*, στο οποίο τροφοδοτούμε ένα instance της κλάσης που δημιουργήσαμε.

Πρόβλημα 5ο: Αδυναμία του NRF να εκτελέσει πολλές διεργασίες

Ας υποθέσουμε το εξής σενάριο: το NRF βρίσκετα σε κατάσταση που περιμένει τη λήψη μέτρησης από κάποιον αισθητήρα, δηλαδή έχει ακολουθήσει εκτέλεση της εντολής `radio.startListening()`. Την ίδια στιγμή, ο χρήστης του GUI αιτεί κάποια αλλαγή στο σύστημα, π.χ. κάποιος λαμπτήρας να τεθεί σε κατάσταση 'on', με φωτεινότητα 50%. Το server side script γράφει στο αρχείο, και ειδοποιείται το watchdog.

Αν υλοποιηθεί το watchdog με τέτοιον τρόπο, ώστε να εκτελεί εντολές του NRF όταν τις λάβει, υπάρχει κίνδυνος να μην αποσταλλεί αυτή η εντολή, αν το NRF αναμένει κάποια λήψη, ή να δυσλειτουργήσει, αν ήδη αποστέλλει κάποιο πακέτο. Να υπενθυμίσουμε ότι το NRF διαθέτει εσωτερικό buffer, με δυνατότητα έως και 5 επαναποστολών κάθε 15ms, αλλά αυτή η δυνατότητα δε βοηθά, καθώς το buffer χρησιμοποιείται μόνο από την πληροφορία που ήδη αποστέλλεται, και μόνο αν έχει προηγηθεί κλήση της εντολής `radio.stopListening()`.
Η λύση που θα υλοποιηθεί, πρέπει να έχει ως αποτέλεσμα να μην υπάρχει περίπτωση για παρεμβατικότητα στη λειτουργία του NRF. Πιο απλά, απαγορεύεται να εκτελεστούν παράλληλα εντολές του NRF από 2 ή περισσότερα threads.

Λύση:

Η λύση που υλοποιήθηκε είναι παρόμοια με τη διαδικασία που ακολουθεί μια μονάδα CPU με έναν πυρήνα και πολλά threads: θα κάνουμε scheduling κάθε εργασίας που χρειάζεται να εκτελεστεί.

Μέχρι στιγμής έχουμε το κυρίως thread, το οποίο αιτεί λήψη μετρήσεων από κάποια τερματικά, και αποστέλλει εντολές για αλλαγή φωτεινότητας στα dimmer, και το thread του watchdog, το οποίο παρακολουθεί το conf αρχείο του GUI.

Η λύση είναι να ενθυλακώσουμε αυτές τις εντολές σε κάποιου είδους δομή, και έπειτα να τις εισάγουμε σε μια ουρά FIFO (FIFO queue). Έπειτα δημιουργούμε ένα ακόμα thread, το οποίο, χωρίς να έχει καμία γνώση των εξωτερικών χαρακτηριστικών του προγράμματος, διαβάζει την πρώτη εντολή στην ουρά και την εκτελεί. Έπειτα, όταν ολοκληρώσει την εκτέλεση, διαβάζει την επόμενη, και ούτω καθ' εξής.

Η βιβλιοθήκη της Python παρέχει ένα πολύ καλό implementation μιας ουράς κατάλληλης για χρήση με threads (thread-safe FIFO queue). Η ουρά αυτή, που υλοποιείται στην κλάση *Queue* του *Queue* module, χρησιμοποιεί locks και conditions από το *threading* module (τα οποία δε θα αναλυθούν εδώ), για να συγχρονίζει την εισαγωγή και εξαγωγή στοιχείων στην ουρά, χωρίς να υπάρχει κίνδυνος να διαβάσουν από αυτή παραπάνω από ένα thread ταυτόχρονα.

Τα στοιχεία που μπορούν να εισαχθούν σε αυτού του είδους ουρά είναι όλα τα αντικείμενα που έχουν τη δυνατότητα *serialization* (σε όρους Python, όλα τα picklable objects, αυτά που δέχεται το *pickle* module). Εμείς τοποθετούμε σε ένα tuple (ίδιο με list, αλλά δεν μπορεί να μεταβληθεί) το όρισμα της μεθόδου που θα καλεστεί, και έπειτα όσα arguments χρειάζονται, σε ένα εσωτερικό tuple. Για παράδειγμα, αν απαιτήσουμε στο GUI σβήσιμο του λαμπτήρα με

διεύθυνση '06', που στο πρόγραμμα τον έχουμε ονομάσει light_1, το watchdog εκτελεί τις εξής εντολές:

```
task = (light_1.dim_to_val, (0))
queue.put(task)
```

Και η διαδικασία που κάνει dequeue και εκτελεί τις εντολές, θα εκτελέσει:

```
task = queue.get()
task[0](*task[1])
queue.task_done()
```

Το `task[0]()` ισοδυναμεί με την εντολή `light_1.dim_to_val()`, και όταν γράφουμε `*task[1]` κάνουμε "unpacking" το δεύτερο όρισμα του tuple, δηλαδή το tuple `(0)`. Η εντολή `f(*a)` για μια μέθοδο f και έναν πίνακα `a = ['foo', 'bar']` ισοδυναμεί με την κλήση `f('foo', 'bar')`. Τέλος, καλώντας τη μέθοδο `task_done` της ουράς, ειδοποιούμε ότι η ουρά δεν είναι γεμάτη, αν έχουμε ορίσει κάποιο μέγιστο μέγεθος.

Μια ακόμα καλύτερη υλοποίηση είναι να ορίσουμε το μέγεθος της ουράς ίσο με 1. Έτσι η ουρά γίνεται ένα καλύτερο παράδειγμα scheduler, καθώς, αν περιέχει ήδη μια διαδικασία προς εκτέλεση και προσπαθήσουμε να εισάγουμε μια ακόμα, τότε πραγματοποιείται blocking, και το πρόγραμμα αναμένει σε αυτό το σημείο μέχρι η enqueued διαδικασία να εκτελεστεί και να αδειάσει η ουρά. Το προτέρημα αυτής της υλοποίησης, εκτός από την εξοικονόμηση μνήμης, είναι η γρηγορότερη απόκριση στις εντολές του GUI.

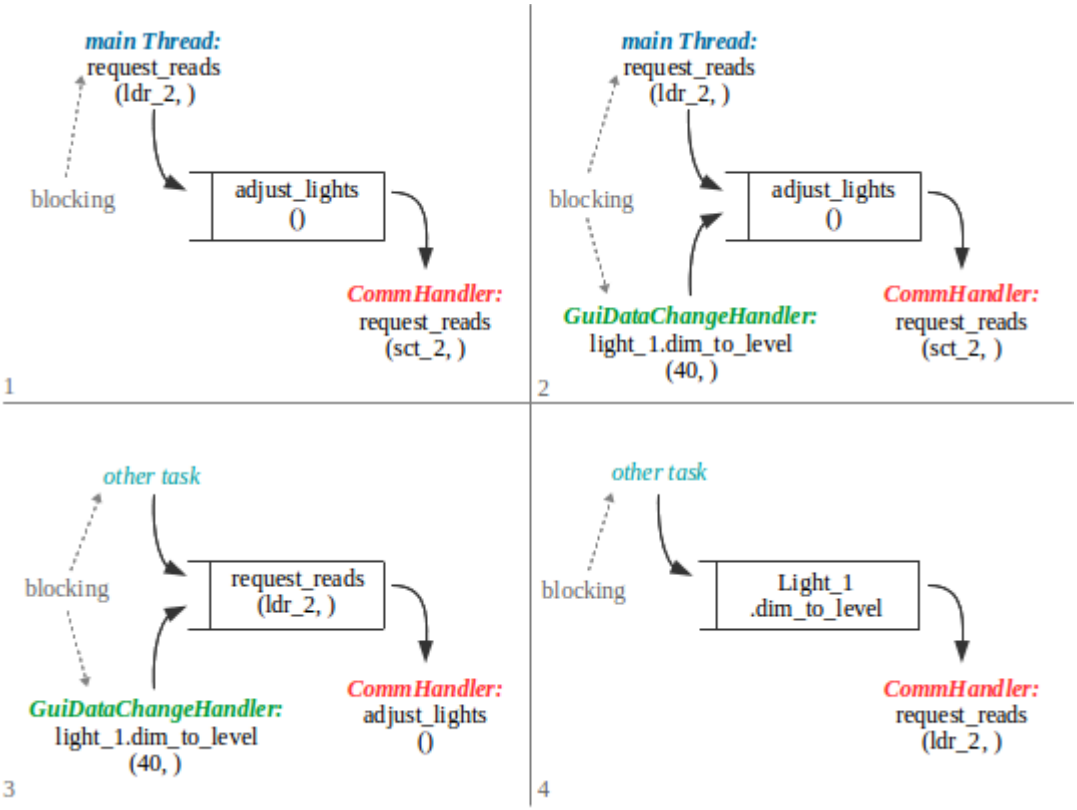
Σε ένα υποθετικό σενάριο όπου δεν έχουμε μοναδιαίο μέγεθος ουράς, και η ουρά περιέχει αρκετές διαδικασίες προς εκτέλεση, τότε αν ληφθεί μια εντολή από το GUI για αλλαγή στην ένταση φωτισμού, τότε θα πρέπει να εκτελεστούν όλες οι enueued διαδικασίες. Αυτό είναι μη αποδοτικό, μιας και οποιαδήποτε ενδιάμεση ρύθμιση στα φώτα θα είναι μι επιθυμητή από τον χρήστη, ο οποίος έχει απαιτήσει αλλαγή στην ένταση.

Αν το μέγεθος της ουράς είναι 1, τότε έχουμε το πολύ μια διαδικασία να εκτελείται, το πολύ μια να αναμένει στην ουρά, και το πολύ μια να κάνει blocking, περιμένοντας την ουρά να αδειάσει. Έτσι, αν ληφθεί εντολή από το GUI σε αυτό το σημείο, θα αναμένει, στην χειρότερη περίπτωση, την εκτέλεση 3 διαδικασιών (ο χρόνος εκτέλεσης των οποίων είναι αρκετά μικρός).

Η υλοποίηση μιας ουράς μεγέθους 1 δεν αντικατοπτρίζει τη χρήση ουρών σε multithreaded προγράμματα. Αντίθετα, έχει ίδια λειτουργικότητα με το να περικλείονται τα κρίσιμα μέρη του κώδικα, δηλαδή αυτά που το καθιστούν μη thread-safe, με locks. Επειδή η υλοποίηση της ουράς στην Python είναι thread-safe, δε χρειάζεται να τοποθετούνται τα locks άμεσα και να έχουμε άσκοπη επανάληψη κώδικα, αλλά αντιθέτως τοποθετούνται έμμεσα, χάρη στην ουρά. Ένα τυπικό σενάριο χρήσης της ουράς αυτής φαίνεται στην εικόνα 5.16.

Πρόβλημα 6ο: Κίνδυνος να βρεθεί το πρόγραμμα σε κατάσταση infinite loop

Όπως έχουμε αναφέρει, η μέτρηση καταλήγει στο gateway ως εξής: το gateway στέλνει μια αίτηση στον εκάστοτε αισθητήρα, και έπειτα εκτελεί την εντολή `radio.startListening()` για να μπει σε listener mode, δηλαδή σε αναμονή, μέχρι ο αισθητήρας να απαντήσει. Να σημειώσουμε ότι, λόγω του scheduling των εργασιών δεν υπάρχει περίπτωση να λάβουμε μέτρηση από κάποια άλλη συσκευή, παρά μόνο από αυτή που έχουμε αιτήσει αποστολή.



Εικόνα 5.16 - Task scheduling στο κυρίως πρόγραμμα

Βέβαια, ακόμα και στη χειρότερη περίπτωση, υπάρχει πάντα έλεγχος διεύθυνσης αποστολέα-παραλήπτη.

Η αναμονή στο gateway υλοποιείται ως εξής, σε ψευδοκώδικα:

```
request reading
enter listener mode
while nothing incoming:
    wait(time)
read incoming data
evaluate
store
```

Ο όρος `time` είναι ένα πολύ μικρό χρονικό διάστημα, έτσι ώστε να μην υπάρχει κίνδυνος να χαθεί η πληροφορία.

Εδώ όμως εισάγεται ο εξής κίνδυνος: αν, για οποιονδήποτε λόγο (είτε δυσλειτουργία του gateway, ώστε να καθυστερήσει η μετάβαση σε listener, είτε του τερματικού, ώστε να μη λάβει την αίτηση) δεν αποσταλλεί η ζητούμενη μέτρηση από το τερματικό, τότε το πρόγραμμα παραμένει σε αυτό το loop.

Λύση:

Για να αντιμετωπιστεί αυτό το πρόβλημα, προσθέτουμε μια παράμετρο timeout στις συνθήκες του loop. Θέτοντας ένα starting time πριν, και συγκρίνοντάς το με την τρέχουσα ώρα, μπορούμε να ορίσουμε ότι θα υπάρχει έξοδος από το loop, εφ' όσον ληφθεί κάποια μέτρηση ή

όταν φτάσουμε στο timeout. Αν δηλαδή ορίσουμε ένα timeout ίσο με 5s, έχουμε τον εξής κώδικα:

```
TIMEOUT = 5
request reading
enter listener mode
start = current_time
while (nothing incoming) or (current_time-start > TIMEOUT):
    wait(time)
read incoming data
evaluate
if there was a timeout:
    store the previous reading
else:
    store
```

Έτσι, στην περίπτωση που καταλάβει το σύστημα ότι αργεί η λήψη της μέτρησης, συνεχίζει την εκτέλεσή του. Επειδή όμως θέλουμε να συνεχιστεί κανονικά η λειτουργία του, αποθηκεύουμε στη θέση της μέτρησης που θα λαμβάναμε την προηγούμενη μέτρηση. Αυτό δεν αλλάζει πολύ την απόδοση, καθώς το σύστημα βασίζεται στην αρχή όλες οι αλλαγές να είναι ανεπαίσθητες και μη αντιληπτές από τον χρήστη.

5.3.3.3 Κυρίως πρόγραμμα

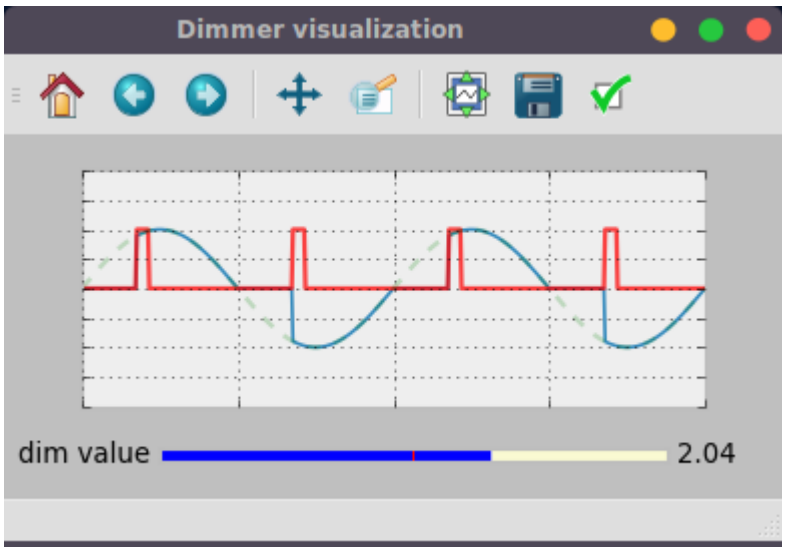
Για να διευκολυνθεί η ανάπτυξη και η συντήρηση του κώδικα του κυρίως προγράμματος στο Raspberry Pi, έχουμε δομήσει το project σε ένα Python package, ως εξής:

```
energy_efficient_lights/
  |-- __init__.py
  |-- app_run.py
  |-- setup.py
  |-- config.py
  |__ components/
        |-- __init__.py
        |-- devices.py
        |-- functions.py
        |-- servers.py
  |__ extern/
        |-- nrf24.py
  |__ plots/
        |-- __init__.py
        |-- power_estimate.py
        |-- dim_table.json
        |-- poly_fit.py
        |-- dimmer_auto.py
```

- Τα αρχεία `__init__.py` είναι αναγκαία σε ένα python package, καθώς σηματοδοτούν ένα φάκελο ως επιμέρους package. Αυτό σημαίνει ότι μπορούμε να κάνουμε `import components`, και να αναφερόμαστε στο module `devices.py` ως `components.devices`.
- Στον φάκελο components περιλαμβάνονται τα δομικά στοιχεία του project, δηλαδή τα εξής:

    - `devices.py`: Περιλαμβάνει τις κλάσεις των αισθητήρων και των λαμπτήρων. Τα τερματικά έχουν δομηθεί σε κλάσεις για να αποφευχθούν επαναλήψεις, και να μειωθεί σημαντικά το μέγεθος του κώδικα. Έτσι, όταν θέλουμε να λάβουμε μέτρηση από έναν αισθητήρα φωτεινότητας με διεύθυνση 04, αφού έχουμε κάνει το απαραίτητο instantiation του αντικειμένου (`light_sensor_04 = LightSensor(addr='04')`), μπορούμε απλά να τρέξουμε `light_sensor_04.get_reading()`.
    - `functions.py`: Εδώ εμπεριέχονται όλες οι απαραίτητες μαθηματικές διαδικασίες που περιγράφηκαν το 4ο κεφάλαιο, μέσω των οποίων κατασκευάζονται τα βάρη των αισθητήρων φωτεινότητας, και υπολογίζονται οι επόμενες τιμές των λαμπτήρων.
    - `servers.py`: Βασικές διαδικασίες για την αναπαράσταση δεδομένων στο GUI, δηλαδή αποθήκευση σε μια βάση δεδομένων, για ιστορική αναπαράσταση, καθώς και αποστολή σε ένα topic ενός MQTT broker, για real-time αναπαράσταση.
- Στον φάκελο extern περιέχεται η βιβλιοθήκη που είναι απαραίτητη για τη λειτουργία του NRF.
- Στον φάκελο plots έχουμε πρώτα από όλα το αρχείο `dim_table.json`, το οποίο δημιουργείται μέσω του `power_estimate.py`, και υλοποιεί τη λύση του 1ου προβλήματος. Τα υπόλοιπα αρχεία χρησιμοποιούνται για εξαγωγή γραφικών παραστάσεων:
    - `poly_fit.py`: Δημιουργεί τις γραφικές παραστάσεις που φαίνονται στις εικόνες 5.14 και 5.15.
    - `dimmer_auto.py`: Πρόκειται για ένα interactive γράφημα που δείχνει τον τρόπο λειτουργίας ενός σύγχρονου dimmer. Ένα snapshot φαίνεται στην εικόνα 5.17.
- Το αρχείο `config.py` περιέχει όλες τις σταθερές, όπως το dim table, τις εντολές κάθε συσκευής και κάποια paths. Εδώ γίνονται επίσης τα απαραίτητα instantiations για κάποιες κλάσεις, όπως το NRF, ο MQTT broker και μια MySQL βάση δεδομένων.
- Το αρχείο `setup.py` υλοποιεί τη διαδικασία του setup, με τη λήψη μετρήσεων από αισθητήρες και αποθήκευσή τους στα ανάλογα instances και σε αρχείο. Υπάρχει η επιλογή να μη τρέξει η διαδικασία του setup, και να χρησιμοποιηθεί ένα αρχείο που έχει προηγουμένως δημιουργηθεί.
- Τέλος, το αρχείο `app_run.py` περιέχει κάποιες διαδικασίες που είναι project specific, καθώς και τον κυρίως κώδικα που οδηγεί το σύστημα.

Θα επεκταθούμε μόνο σε συγκεκριμένα αρχεία, καθώς κάποια είναι εύκολο να κατανοηθούν. Όλος ο κώδικας βρίσκεται επίσης στο παράρτημα Β.



Εικόνα 5.17 - Interactive dimmer vizualization

devices.py

Σε αυτό το αρχείο δημιουργούμε 2 κύριες κλάσεις: `Sensor` και `Light`. Η κλάση `Sensor` είναι και parent στις κλάσεις `LightSensor` και `DissipationSensor`.

Το σύστημα, όπως αναφέραμε στο 2ο κεφάλαιο, θέλουμε να είναι εύκολα επεκτάσιμο. Αυτό σημαίνει ότι, σε περίπτωση που προστεθεί κάποιο ακόμα τερματικό, θα πρέπει να αλλάξει ο κώδικας το λιγότερο δυνατόν. Αυτό υλοποιήθηκε προσθέτοντας σε επίπεδο κλάσης έναν πίνακα, στον οποίο αποθηκεύεται κάθε στοιχείο που δημιουργείται, καθώς και ένα mapping, το οποίο αντιστοιχίζει διευθύνσεις με τα αντίστοιχα στοιχεία. Έτσι, αν γίνουν τα εξής instantiations:

```
light_1 = Light('06')
light_2 = Light('07')
light_sensor_1 = LightSensor('03')
light_sensor_2 = LightSensor('04')
sct_1 = DissipationSensor('08')
sct_2 = DissipationSensor('09')
```

Παίρνουμε ως αποτέλεσμα:

```
>>> Light.instances
[light_1, light_2]
>>> Light.mapping
{'06': light_1, '07': light_2}
>>> LightSensor.instances
[light_sensor_1, light_sensor_2]
>>> LightSensor.mapping
{'03': light_sensor_1, '04': light_sensor_2}
>>> DissipationSensor.instances
[sct_1, sct_2]
>>> DissipationSensor.mapping
{'08': sct_1, '09': sct_2}
```

(Στην πραγματικότητα παίρνουμε διευθύνσεις και όχι ονόματα αντικειμένων, αλλά εδώ παρουσιάζονται έτσι χάριν συντομίας)

Η δημιουργία αυτών των δομών έχει το εξής αποτέλεσμα: όταν θέλουμε να λάβουμε μετρήσεις από όλους τους αισθητήρες, είτε να στείλουμε κάποια εντολή στους λαμπτήρες, διατρέχουμε τον πίνακα instances, εκτελώντας τις κατάλληλες μεθόδους σε κάθε entry. Έτσι, αν προστεθεί κάποιο τερματικό, αρκεί ο χρήστης να προσθέσει μια γραμμή, με την κατάλληλη κλάση, καθώς και διεύθυνση ίδια με αυτή του τερματικού.

Στην κλάση `Lights` υλοποιούμε μεθόδους για αλλαγή φωτεινότητας κατά κάποια τιμή, αλλαγή φωτεινότητας σε κάποια τιμή του dim table (0,10..100%), καθώς και ομαλή ή απότομη αλλαγή.

Στην κλάση `Sensor` υλοποιούμε μια μέθοδο, η οποία αιτεί μέτρηση από έναν αισθητήρα, και την αποθηκεύει στην μεταβλητή `current_read`. Συγκεκριμένα στην υπο-κλάση `DissipationSensor` έχουμε επεξεργαστεί αυτή τη μέθοδο, έτσι ώστε να μετατρέπει τη μέτρηση κατάλληλα για να εξαλείφεται το σφάλμα, μέσω μιας ακόμα μεθόδου

(`DissipationSensor.remove_error`).

app_run.py

Το κυρίως πρόγραμμα, δηλαδή του αρχείο app_run.py περιέχει τις εξής λειτουργίες:

- `request_readings(sensor_type)` *[function]*: διατρέχει τα instances της κλάσης που παρέχεται ως μεταβλητή `sensor_type`, και για κάθε instance εκτελείται η ρουτίνα `get_reading`. Έτσι αποθηκεύονται σε κάθε αντικείμενο οι μετρήσεις την παρούσα χρονική στιγμή.
- `adjust_lights()` *[function]*: τοποθετεί τις μετρήσεις κάθε αισθητήρα φωτεινότητας σε έναν πίνακα, τον οποίον περνά σαν argument στην ρουτίνα `functions.get_next_dim`. Αυτή η ρουτίνα λύνει τη γραμμική εξίσωση που έχουμε περιγράψει σε προηγούμενα κεφάλαια, και επιστρέφει έναν πίνακα, με τις τιμές κατά τις οποίες πρέπει να αλλάξει η φωτεινότητα του κάθε λαμπτήρα. Έπειτα στέλνουμε τις τιμές αυτές σε κάθε λαμπτήρα, διατρέχοντας τον πίνακα `Light.instances`. Στην περίπτωση που δεν είναι όλα τα φώτα σε κατάσταση auto, οι πίνακες που παρέχονται στη ρουτίνα επίλυσης της γραμμικής εξίσωσης επεξεργάζονται κατάλληλα (ενότητα 5.3, αλγόριθμος επιλογής φωτισμού).
- `init_lights()` *[function]*: μέσω αυτής της ρουτίνας, επιλέγεται η κατάλληλη αρχική τιμή για τα φώτα, αμέσως μετά το στάδιο setup. Αυτό γίνεται αν, αντί για την τρέχουσα μέτρηση κάθε αισθητήρα, περάσουμε στη ρουτίνα `get_next_dim` τις ελάχιστες τιμές φωτισμού που διαβάσαμε στο setup stage. Έτσι η ρουτίνα τρέχει θεωρώντας ότι τα φώτα είναι στο 0% της φωτεινότητας, και υπολογίζει την τιμή που θα πρέπει να πάρουν για να φτάσουν στο threshold.
- `GUIDataChangeHandler(FileSystemEventHandler)` *[class]*: πρόκειται για το watchdog class που παρατηρεί το αρχείο που γράφεται από το GUI για τυχόν αλλαγές. Αν εντοπιστεί αλλαγή, κάνει update τις τρέχουσες ρυθμίσεις, και προγραμματίζει μέσω του FIFO queue αλλαγές στις φωτεινότητες, αν χρειάζονται.
- `CommHandler(Thread)` *[class]*: αυτή η κλάση περιμένει μέχρι να έχουμε κάποιο task στην ουρά. Αν υπάρχει, τότε εκτελείται.

Χάρη σε αυτές τις ρουτίνες και κλάσεις, ο προγραμματισμός της λειτουργίας του συστήματος καθίσταται πολύ απλός: αφού κάνουμε τις αρχικοποιήσεις και δημιουργήσουμε τους απαραίτητους πίνακες με βάρη, έχουμε συνεχή επανάληψη των εξής εντολών:

```
task = (request_readings, (LightSensor,))
queue.put(task)

task = (adjust_lights, ())
queue.put(task)

task = (request_readings, (DissipationSensor,))
queue.put(task)

for sct in DissipationSensor.instances:
    task = (_db_insert, (sct,))
    queue.put(task)
    task = (_mqtt_publish, (sct,))
    queue.put(task)
```

Η επεξήγηση της λειτουργίας είναι πολύ απλή:

- Τοποθετούμε στην ουρά το task της λήψης μετρήσεων από τους αισθητήρες φωτεινότητας.
- Έπειτα τοποθετούμε το task της ρύθμισης των λαμπτήρων. Προφανώς, θα εκτελεστεί αφότου γίνει η ανάγνωση των μετρήσεων.
- Μετά γίνεται προγραμματισμός για λήψη των μετρήσεων από τους μετρητές κατανάλωσης.
- Όταν θα έχει ολοκληρωθεί αυτή η διεργασία, θα εκτελεστούν οι υπόλοιπες εργασίες που τοποθετούμε στην ουρά: η αποστολή κάθε μέτρησης κατανάλωσης στη βάση δεδομένων, καθώς και στον MQTT broker.
- Παράλληλα, αν υπάρξει κάποιο update από το GUI, το watchdog θα τοποθετήσει στην επόμενη θέση της ουράς την κατάλληλη εργασία, η οποία θα εκτελεστεί στη σειρά της.

Να σημειώσουμε ότι η σύνταξη `(element, )` σε ένα tuple που περιέχει ένα μοναδικό στοιχείο είναι απαραίτητη για την Python, έτσι ώστε να διαχωρίζεται από ένα generator expression.

Να σημειώσουμε επίσης ότι δεν γίνεται ακριβώς αυτή η ακολουθία εντολών. Επειδή οι μετρήσεις από τους αισθητήρες λαμβάνονται αρκετά συχνά, δεν είναι αποδοτικό να λαμβάνουμε μετρήσεις κατανάλωσης με τον ίδιο ρυθμό. Αντίθετα, τοποθετούμε και εδώ ένα timer, μέσω του οποίου λαμβάνουμε μετρήσεις και τις στέλνουμε στη βάση δεδομένων και στον broker κάθε 1 λεπτό.

5.3.4 GUI

Σύμφωνα με τις απαιτήσεις, το GUI αναπτύχθηκε με σκοπό να προσφέρει τις εξής λειτουργίες:

- Δυνατότητα ελέγχου του επιθυμητού επιπέδου φωτεινότητας στον χώρο.
- Δυνατότητα διακοπής της αυτόματης λειτουργίας ενός λαμπτήρα, και έλεγχος της φωτεινότητάς του.
- Παρουσίαση μετρήσεων κατανάλωσης σε πραγματικό χρόνο.
- Παρουσίαση αποθηκευμένων μετρήσεων που έχουν διεξαχθεί, σε χρονικό εύρος που καθορίζει ο χρήστης.

Η υλοποίηση των λειτουργιών αυτών έχει αναλυθεί ήδη. Παρακάτω θα γίνει μια περαιτέρω παρουσίαση της διαδικασίας για την κατασκευή του GUI.

5.3.4.1 Front-end

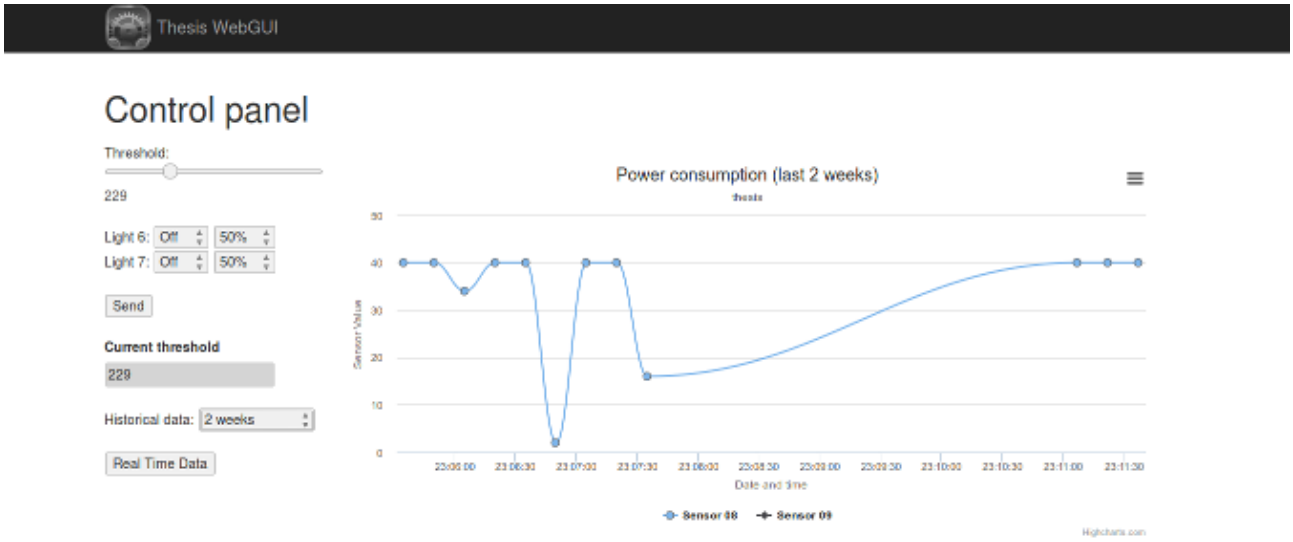
Το front end του interface αποτελεί μια πολύ βασική υλοποίηση με χρήση HTML και JavaScript/jQuery. Ο κώδικας HTML υλοποιεί τα βασικά controls, τις ενδείξεις, καθώς και το layout του interface.

Η JavaScript και η jQuery χρησιμοποιήθηκαν για την προσθήκη των εξής δυναμικών δυνατοτήτων:

- Φόρτωση των αρκικών τιμών για το threshold και την κατάσταση του κάθε λαμπτήρα, βάσει των πιο πρόσφατων αποθηκευμένων ρυθμίσεων.

- Σε περίπτωση αλλαγής των ρυθμίσεων, αυτές αποθηκεύονται τοπικά, αφού προηγηθεί το κατάλληλο evaluation όσον αφορά τις τιμές τους.
- Διαμόρφωση των γραφικών παραστάσεων, οι οποίες δημιουργούνται με χρήση του JavaScript framework **Highcharts**, ανάλογα με την αίτηση του χρήστη.

Το τελικό interface έχει τη μορφή που φαίνεται στην εικόνα 5.18.



Εικόνα 5.18 - WebGUI

5.3.4.2 Back-end

Οι παραπάνω λειτουργίες πραγματοποιούνται με χρήση απλών GET και POST requests, καθώς και με τη χρήση ορισμένων frameworks.

Η φόρτωση των αρχικών τιμών γίνεται ως εξής: όταν η HTML σελίδα έχει ολοκληρώσει το loading, γίνεται εκτέλεση των εντολών της jQuery:

```
$.get("initvals", function(initValues) {
    $.each(init, function(key, value){
...
```

Στην πρώτη γραμμή εκτελείται ένα GET που αιτεί απάντηση από το server-side Python script. Αυτό το script, το οποίο έχει υλοποιηθεί με το framework Flask, επιστρέφει απάντηση σε μορφή JSON, με τα εξής πεδία:

- `thresh`: το επίπεδο φωτισμού σε μονάδες του LDR
- `light_i_state`: η κατάσταση του λαμπτήρα με διεύθυνση i. Μπορεί να έχει την τιμή 'on', off' ή 'auto'
- `light_i_int`: η ένταση (intensity) του λαμπτήρα με διεύθυνση i, με τιμές 0,10,20,...90,100

Αυτές οι τιμές έπειτα αντιστοιχίζονται στο κάθε control, έτσι ώστε ο χρήστης να μπορεί να βλέπει τις τρέχουσες τιμές του configuration του συστήματος. Αυτό γίνεται αν έχουμε δώσει ένα ξεχωριστό attribute στο κάθε control element της HTML. Έπειτα μπορούμε να αλλάξουμε την τιμή

του, ανεφερόμενοι στο attribute, με ένα # (hashtag) στην αρχή:

```
$('#showthresh').val(initValues[0]);
```

Η επόμενη λειτουργία υλοποιείται χάρη στα built in functions της JavaScript: `change` (που τρέχει όταν αλλάζει η παράμετρος που παρέχουμε) και `click` (που τρέχει όταν υπάρξει ένα click event σε ένα button ή submit button που παρέχουμε). Για παράδειγμα, αν ο χρήστης κάνει κάποια αλλαγή στα controls, και πατήσει το κουμπί Send, τότε έχουμε trigger στην εξής συνάρτηση:

```
$('#sendvals').click(function() {
...
```

Αυτό το block κώδικα τοποθετεί τις τρέχουσες τιμές κάθε control σε ένα POST request, που αποστέλλεται στο server side script. Αυτό με τη σειρά του ελέγχει το configuration, και, αν αντιληφθεί ότι υπάρχει αλλαγή, εκτελεί update.

Η παρουσίαση των δεδομένων γίνεται με χρήση Highcharts. Η real time αναπαράσταση δε θα αναλυθεί εδώ καθώς εξαρτάται περισσότερο απο built in δυνατότητες, και λιγότερο από κώδικα. Θα γίνει μια σύντομη ανάλυση της αναπαράστασης παλαιότερων μετρήσεων.

Όταν ο χρήστης επιλέξει μια χρονική περίοδο για την οποία αιτεί γραφική αναπαράσταση των μετρήσεων, καλείται η συνάρτηση που τοποθετεί το χρονικό εύρος σε ένα URL. Το URL αυτό αποτελεί το GET request που στέλνεται το server-side script. Αυτό το script εκτελεί SQL εντολές, και αντλεί από τη MySQL βάση δεδομένων όλες τις μετρήσεις που ανταποκρίνονται σε αυτές που αντιστοιχούν στο συγκεκριμένο εύρος. Να σημειωθεί ότι το table της MySQL που έχουμε δημιουργήσει είναι το εξής:

"power"	
Field	**Type**
datetime	datetime
sensor_id	int(11)
sensor_value	int(11)

Πίνακας 5.6 - Database schema

Οπότε για κάθε μέτρηση, έχουμε την χρονική στιγμή της τοποθέτησής της στη βάση δεδομένων, τη διεύθυνση του εκάστοτε αισθητήρα, και την τιμή της μέτρησης.

Η απάντηση του script είναι JSON encoded, και έχει τη μορφή `{(timestamp, sensor_addr, sensor_value)}` για κάθε row του table που επιστρέφεται. Η JavaScript εξάγει τα δεδομένα, και τα τοποθετεί στους άξονες που ορίζει το framework Highcharts. Χάρη σε αυτό το framework, μπορούμε να θέσουμε τον άξονα x να έχει τη μορφή datetime, έτσι ώστε οι μετρήσεις να παρουσιάζονται με λογικό τρόπο. Αυτό σημαίνει ότι, αν δεν υπάρχει μέτρηση για κάποιο μεγάλο χρονικό διάστημα, αυτό θα παρουσιάζεται ως κενό, ή καλύτερα ως μηδενική μέτρηση.

Πρέπει να αναφερθεί επιγραμματικά και ο τρόπος με τον οποίο λειτουργεί το Flask, πως δηλαδή χειρίζεται τα GET-POST requests που του στέλνουμε, και πως απαντάει.

Για να κατασκευάσουμε ένα webapp σε Flask, αυτό που κάνουμε είναι το εξής: αρχικά

γράφουμε τις συναρτήσεις που θα τρέχουν για κάθε request. Έπειτα, για να αντιστοιχίσουμε κάθε συνάρτηση (δηλαδή κάθε ενέργεια σε request) σε ένα συγκεκριμένο URL (σχετικό με το base URL), τοποθετούμε ένα wrapper, το οποίο παρέχει το Flask σε μορφή decorator, με argument το σχετικό URL:

```
@app.route('/', methods=['GET', 'POST'])
def index():
    req = request.args.get('time', None)
    if req:
        print req
        g.cur.execute("""SELECT * FROM %s\
                      WHERE datetime >= (NOW() - INTERVAL %s)""",
                      (SQL_TABLE, req))
        data = g.cur.fetchall()
        arr_send = [[str(time.mktime(i[0].timetuple())),
                     str(int(i[1])),
                     str(int(i[2]))]
                    for i in data]
        data_send = json.dumps(arr_send)
        return data_send
    else:
        return render_template('index.html')
```

Εδώ, όταν ο χρήστης πληκτρολογήσει το URL στο οποίο "τρέχει" το WebGUI, τότε θα ζητήσει τη σελίδα με σχετικό URL: "/". Επειδή δεν έχουμε arguments (GET ή POST), τότε η πρώτη if συνθήκη δεν τηρείται, άρα τρέχει η εντολή στη συνθήκη else: `return render_template('index.html')`. Με αυτή την εντολή φορτώνεται στον browser η σελίδα `index.html`.

Όταν όμως εκτελεστεί μέσω της JavaScript το GET request που στέλνει το χρονικό εύρος και ζητά μετρήσεις, τότε η πρώτη if συνθήκη είναι αληθής. Έτσι, το script εκτελεί ένα query στη βάση δεδομένων, του οποίου τα αποτελέσματα αποστέλλονται στη σελίδα σε μορφή JSON.

Κεφάλαιο 6 Δοκιμές – Αποτελέσματα

Σε αυτό το κεφάλαιο θα γίνει δοκιμή της λειτουργίας του συστήματος που αναπτύχθηκε. Για να γίνει με σωστό τρόπο η διεξαγωγή των δοκιμών στο σύστημα, καθώς και για να εξαχθούν σωστά αποτελέσματα, θα χρειαστεί να οριστούν κάποιες παράμετροι που θα ορίσουν το πλαίσιο των δοκιμών αυτών. Για παράδειγμα, δε θα ήταν σωστό να γίνει δοκιμή μόνο για την κανονική λειτουργία, χωρίς υποθετικές παρεμβάσεις του χρήστη, ή μόνο για μια συγκεκριμένη χρονική περίοδο της ημέρας, όπου το φυσικό φως δεν παρουσιάζει διακυμάνσεις. Οι παράμετροι που επιλέχθηκαν είναι:

- Η διάταξη του συστήματος, που ανάγεται στη διάταξη των αισθητήρων στον χώρο, μιας και οι λαμπτήρες σε έναν χώρο βρίσκονται σε σταθερά σημεία
- Το φυσικό φως
- Το προκαθορισμένο επίπεδο φωτεινότητας
- Η κατάσταση του κάθε λαμπτήρα, αν δηλαδή είναι σε κατάσταση αυτόματης λειτουργίας ή καθορισμένης από τον χρήστη μέσω του WebGUI
- Τα ηλεκτρικά χαρακτηριστικά, που, όπως έχουμε αναφέρει, επηρεάζουν τη λειτουργία των μετρητών κατανάλωσης

Μια ακόμα παράμετρος είναι η ομαλότητα μεταξύ των καταστάσεων των παραπάνω παραμέτρων. Είναι δηλαδή αναμενόμενο να μη λειτουργήσει ορθά το σύστημα, αν ο χρήστης απαιτεί αλλαγές μεγάλου εύρους στο επίπεδο φωτεινότητας ή στην ένταση κάποιου λαμπτήρα, σε πολύ μικρό χρονικό διάστημα. Το ζήτημα σε αυτή την περίπτωση είναι να εξαχθεί μια εκτίμηση σχετικά με το κατά πόσον θα επηρεαστεί η λειτουργία.

Στη συνέχεια θα γίνουν δοκιμές πάνω σε διάφορα σενάρια, ξεκινώντας από τη λειτουργία κάποιων κόμβων ανεξάρτητα από τις καταστάσεις και τα στάδια λειτουργίας, και συνεχίζοντας με την λειτουργία του συστήματος σαν σύνολο.

Στο τέλος του κεφαλαίου θα επαληθευτεί και η σωστή λειτουργία του GUI.

Όλες οι δοκιμές θα διεξαχθούν με τη χρήση λαμπτήρων 40W.

6.1 Δοκιμές συστήματος φωτισμού

Δοκιμή 1:
Λειτουρία του μετρητή κατανάλωσης

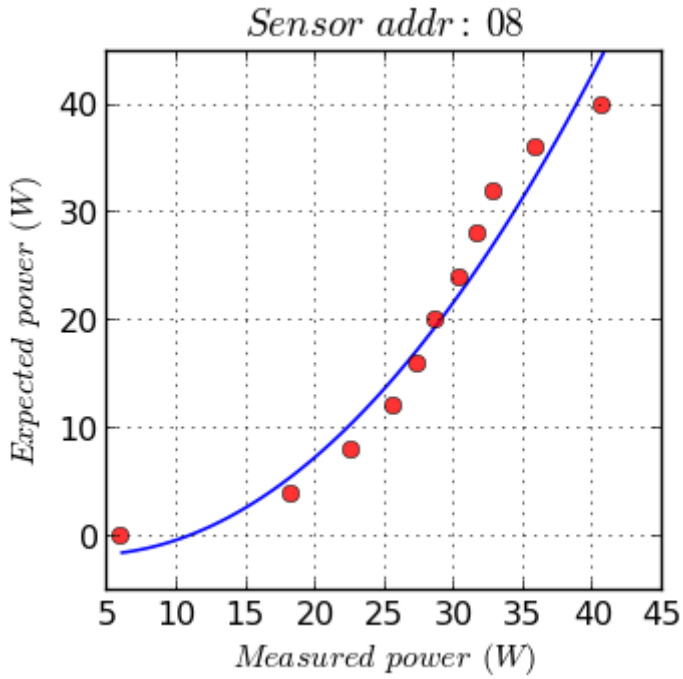
Σύμφωνα με τα συμπεράσματα της ενότητας 5.3.3.1, πριν από την έναρξη της λειτουργίας του συστήματος θα πρέπει να έχει γίνει αντιστοίχιση των τιμών που δίνει ο μετρητής κατανάλωσης με τα τρέχοντα δεδομένα, με τις αναμενόμενες τιμές. Αυτό σημαίνει ότι, αν αλλάξει η πηγή ρεύματος που τροφοδοτεί το Arduino του μετρητή κατανάλωσης, θα πρέπει να γίνει εκ νέου υπολογισμός. Αυτό γίνεται εφ'όσον έχουμε καταλήξει να μη συμπεριλάβουμε τους μετρητές κατανάλωσης στο setup stage, με σκοπό την εξοικονόμηση χρόνου στο testing.

Αρχικά θέτουμε το σύστημα σε λειτουργία, και λαμβάνουμε τις εξής μετρήσεις κατανάλωσης

για τα αντίστοιχα επίπεδα φωτεινότητας:

Light intensity	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Measured power (W)	*7.96*	*18.21*	*22.53*	*25.65*	*27.31*	*28.68*	*30.47*	*31.75*	*32.92*	*35.85*	*40.71*
Expected power (W)	0	4	8	12	16	20	24	28	32	36	40
Error	7.96	14.21	14.53	13.65	11.31	8.68	6.47	3.75	0.92	-0.15	0.71

Έπειτα χρησιμοποιούμε την αντιστοίχιση που περιγράφεται στην ίδια ενότητα, η οποία δίνει την καμπύλη που φαίνεται στην εικόνα 6.1.



Εικόνα 6.1 - Γραμμική παλινδρόμηση στα δεδομένα της δοκιμής

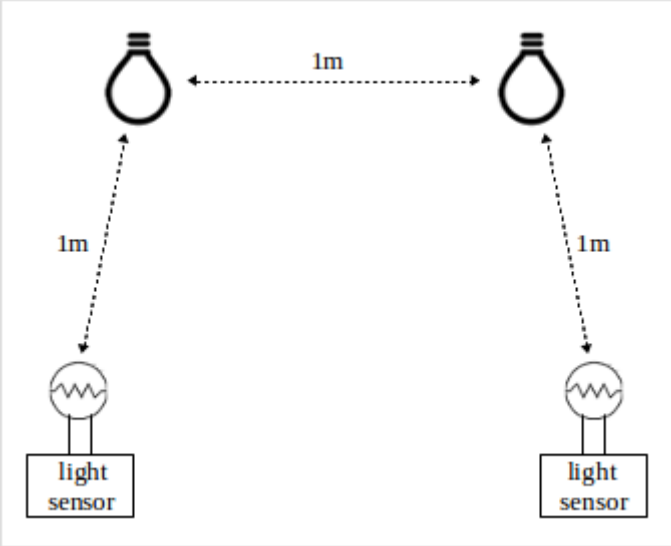
Επαναλαμβάνοντας τις μετρήσεις, αφού τις μετατρέψουμε μέσω της αντιστοίχισης που εξαλείφει το σφάλμα, παίρνουμε τα εξής αποτελέσματα:

Light intensity	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Measured power (W)	*0*	*4.97*	*9.78*	*13.73*	*16.42*	*18.80*	*21.43*	*23.83*	*26.63*	*33.99*	*40*
Expected power (W)	0	4	8	12	16	20	24	28	32	36	40
Error	0	0.97	1.78	1.73	0.42	-1.2	-2.57	-4.17	-5.37	-2.01	0

Το μέσο απόλυτο σφάλμα για αυτές τις μετρήσεις είναι 1.83, τιμή αρκετά μικρή ώστε να μην επηρεάζει τα αποτελέσματα. Είναι λογικό να υπάρχει σφάλμα, μιας και οι μετρήσεις που παρέχει

το SCT στο κύκλωμα μέτρησης βασίζονται σε δειγματοληψία. Το ζητούμενο είναι να υπάρχει μια αρκετά καλή εκτίμηση για την τρέχουσα κατανάλωση σε κάθε γραμμή, κάτι που ισχύει, όπως δείχνουν τα αποτελέσματα.

Δοκιμή 2:
Setup stage



Εικόνα 6.2 - Διάταξη που χρησιμοποιήθηκε για τις δοκιμές

Η δοκιμή του setup stage αφορά κυρίως την αξιολόγηση των αποτελεσμάτων, δηλαδή τον πίνακα m_list που περιλαμβάνει τα βάρη του κάθε αισθητήρα φωτεινότητας σε σχέση με τον κάθε λαμπτήρα. Γι'αυτή τη δοκιμή χρησιμοποιήθηκε η διάταξη της εικόνας 6.2. Λόγω του ότι οι αισθητήρες και οι λαμπτήρες είναι τοποθετημένοι συμμετρικά, αναμένουμε συμμετρικές τιμές. Αυτό σημαίνει ότι κατά πάσα πιθανότητα η εξάρτηση του αισθητήρα 1 από τον λαμπτήρα 2 θα είναι ίδια με αυτή του αισθητήρα 2 από τον λαμπτήρα 1, κ.ο.κ.

"Τρέχοντας" τη ρουτίνα του setup stage, παίρνουμε τις εξής μετρήσεις:

Light level (%)		**0**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Sensor 1	**Light 1**	*138*	*155*	*205*	*261*	*303*	*350*	*390*	*413*	*438*	*475*	*508*
	Light 2	*131*	*135*	*156*	*185*	*210*	*242*	*261*	*294*	*312*	*346*	*369*
Sensor 2	**Light 1**	*79*	*105*	*155*	*209*	*256*	*294*	*324*	*346*	*379*	*410*	*429*
	Light 2	*84*	*122*	*199*	*270*	*324*	*362*	*399*	*429*	*451*	*483*	*501*

Οι οποίες παράγουν τα εξής βάρη (πίνακας m_list):

Sensor 1	**Light 1**	*37.0*
	Light 2	*23.8*
Sensor 2	**Light 1**	*35.0*
	Light 2	*41.7*

Λαμβάνοντας υπ' 'οψιν τη διάταξη, και το γεγονός ότι αυτός ο πίνακας προσδιορίζει την εξάρτηση των μετρήσεων από την φωτεινότητα του κάθε λαμπτήρα, παρατηρούμε ότι οι τιμές είναι αρκετά λογικές. Οι αποκλίσεις οφείλονται στους εξής παράγοντες:

- Ηλεκτρικά χαρακτηριστικά: διαφορετική πόλωση του Arduino δίνει και διαφορετική μέτρηση. Ένα LDR με μικρότερο εύρος αντιστάσεων μπορεί να δώσει τιμές με μικρότερες αποκλίσεις για διαφορετική πόλωση, αλλά έτσι θυσιάζεται το εύρος των μετρήσεων.

- Τοποθέτηση του αισθητήρα: λόγω του ότι ο αισθητήρας 1 είναι τοποθετημένος κοντά σε παράθυρο, επηρεάζεται περισσότερο από τον λαμπτήρα 2 απ' όσο θα έπρεπε. Αυτό οφείλεται στις ανακλάσεις, που έχουν ως αποτέλεσμα τη δημιουργία μιας επιπλέον πηγής φωτός. Αυτό δεν αλλοιώνει τις μετρήσεις, αλλά αντιθέτως κάνει τη λειτουργία του συστήματος πιο "ρεαλιστική".

Δοκιμή 3:
Παράκαμψη του setup stage

Το σύστημα έχει σχεδιαστεί έτσι ώστε να μη χρειάζεται το setup stage κάθε φορά που τίθεται σε λειτουργία. Αυτό σημαίνει ότι, στην περίπτωση που τεθεί το σύστημα εκτός λειτουργίας (π.χ. σε διακοπή ρεύματος), δε θα χρειαστεί να εκτελεστεί η ρουτίνα setup ξανά. Αντιθέτως, τα αποτελέσματα της τελευταίας εκτέλεσης της ρουτίνας θα διαβάζονται από τοπικό αρχείο, και θα εφαρμόζονται στο σύστημα.

Αυτό πραγματοποιείται, αν διαμορφώσουμε το πρόγραμμα `app_run.py`, έτσι ώστε να δέχεται κάποια flags. Αν δηλαδή εκτελεστεί η εντολή `python app_run.py -s`, τότε έχουμε εκτέλεση της ρουτίνας setup, ενώ αν εκτελεστεί η εντολή `python app_run.py -f`, το πρόγραμμα διαβάζει τις ρυθμίσεις από αρχείο.

Για να επιβεβαιώσουμε την ορθή λειτουργία, εκτελούμε το πρόγραμμα με το `-f` flag, και παίρνουμε το εξής m_list:

Sensor 1	**Light 1**	*37.0*
	Light 2	*23.8*
Sensor 2	**Light 1**	*35.0*
	Light 2	*41.7*

Το αποτέλεσμα είναι ίδιο με αυτό της προηγούμενης δοκιμής.

Δοκιμή 4:
Αρχικά επίπεδα φωτεινότητας λαμπτήρων

Όπως αναλύσαμε στο κεφάλαιο 4, πριν την έναρξη της κανονικής λειτουργίας θα πρέπει να τεθούν οι λαμπτήρες σε ένα αρχικό επίπεδο φωτεινότητας. Αυτό καθορίζεται αμέσως μετά το setup, και οι τιμές εξάγονται από τον αλγόριθμο επιλογής φωτισμού, αν εκτελεστεί με την προϋπόθεση ότι οι αρχικές φωτεινότητες είναι 0%. Έτσι, δίνοντας στον αλγόριθμο τη μέτρηση κάθε αισθητήρα που έχει ληφθεί κατά το setup, για 0% φωτεινότητα από κάθε λαμπτήρα, το αποτέλεσμα που δίνει είναι το ποσοστό τοις εκατό που θα πρέπει να αυξηθεί η ένταση κάθε λαμπτήρα, δηλαδή το ποσοστό στο οποίο πρέπει να τεθεί (αφού η αρχική τιμή είναι 0).

Το μειονέκτημα αυτής της μεθόδου είναι ότι είναι (θεωρητικά) ακριβής μόνο για τις συνθήκες που υπάρχουν κατά το setup stage. Αν δηλαδή γίνει setup από αρχείο, και εκείνη τη στιγμή υπάρχει φυσικό φως διαφορετικής έντασης από το setup stage, τότε τα επίπεδα φωτεινότητας δε θα είναι ακριβή. Αν όμως θεωρήσουμε ότι το setup από αρχείο θα είναι μια διαδικαία που θα εκτελείται αρκετά σπάνια, λόγω του ότι όποτε χρειαστεί επανεκκίνηση του συστήματος θα χρειαστεί και ρύθμισή του από την αρχή, τότε δε μας απασχολεί αυτή η απόκλιση.

Η δυνατότητα του συστήματος να βρίσκεται κατά την εκκίνησή του σε σταθερή κατάσταση (κατάσταση όπου δε χρειάζεται ρύθμιση της φωτεινότητας) είναι πολύ σημαντική, κυρίως για αισθητικούς λόγους. Αν ο χρήστης έχει θέσει το επιθυμητό επίπεδο φωτεινότητας πολύ ψηλά και εκτελεστεί κάποια ρύθμιση, τότε κατά πάσα πιθανότητα θα απαιτείται αρκετός χρόνος μέχρι να ρυθμιστούν οι φωτεινότητες κατάλληλα. Η λειτουργία αυτή αφαιρεί αυτή την καθυστέρηση.

Πριν γίνει η δοκιμή, να σημειωθεί ότι με την τρέχουσα υλοποίηση του συστήματος, έχουμε πρόσβαση σε "σωστά" επίπεδα φωτισμού που αντιστοιχούν σε τιμές πολλαπλάσιες του 10. Έτσι, αν κατά αυτό το στάδιο ο αλγόριθμος δώσει κάποια τιμή που δεν είναι 0,10,20..90,100, τότε θα πρέπει να πάρουμε το πολλαπλάσιο του 10 στο όριο [0, 100] που είναι πλησιέστερο σε αυτή την τιμή.

Το setup που εκτελέστηκε προηγουμένως, με το επίπεδο φωτεινότητας να έχει οριστεί από τον χρήστη στην τυχαία τιμή thresh=348, δίνει ως αποτέλεσμα τις εξής αρχικές τιμές για τους λαμπτήρες, οι οποίες πρέπει να στογγυλοποιηθούν στο πλησιέστερο πολλαπλάσιο του 10 (υπενθυμίζουμε ότι το threshold παίρνει τιμές 0-800):

	Result	*Level to send*
Light 1	*33.17*	30
Light 2	*36.66*	40

Και οι μετρήσεις που δίνουν οι αισθητήρες φωτεινότητας αμέσως μετά είναι οι εξής:

	Reading	*Abs. Error (thresh-reading)*
Sensor 1	*341*	7
Sensor 2	*366*	18

Παρατηρούμε ότι οι μετρήσεις δεν αποκλίνουν ιδιαίτερα από το threshold. Συγκεκριμένα, το πρόγραμμα χρειάστηκε 4 κύκλους λειτουργίας για να φέρει το σύστημα σε σταθερή κατάσταση, όπου δε χρειάζονταν περαιτέρω ρυθμίσεις φωτεινότητας.

Δοκιμή 5:
Κανονική λειτουργία χωρίς παρεμβάσεις από τον χρήστη

Κατά την κανονική λειτουργία, το πιο ζωτικό σημείο για την ορθότητα της λειτουργίας είναι το queuing/multithreading που έχουμε υλοποιήσει.

Η σειρά με την οποία εκτελούνται τα διάφορα στάδια της κανονικής λειτουργίας είναι η εξής:

1. Αίτηση για λήψη μέτρησης από τους αισθητήρες φωτισμού
2. Ρύθμιση φωτισμού
3. Αίτηση για λήψη μέτρησης από τους μετρητές κατανάλωσης
4. Εισαγωγή μετρήσεων στη βάση δεδομένων
5. Αποστολή μετρήσεων στον MQTT broker

Τα στάδια αυτά δεν εκτελούνται αμέσως, αλλά αντιθέτως προστίθενται στην ουρά με αυτή τη σειρά. Η σειρά έχει μεγάλη σημασία, έτσι ώστε να μην υπάρξει καμία δυσλειτουργία.

Για να γίνει εύκολο το debugging του συστήματος, το ρυθμίσαμε έτσι ώστε να εξάγει ένα log με όλες τις απαραίτητες πληροφορίες της λειτουργίας, καθώς και την χρονική στιγμή της εκτέλεσής τους. Ένα log, από την εκκίνηση της λειτουργίας του συστήματος, και για τη διάρκεια 2 κύκλων, έχει την εξής μορφή:

```
Initial values: [67.89, 18.66]
[19:46:23] Executing task:request_readings with args [...]
[19:46:25] sensor 03, received 404
[19:46:28] sensor 04, received 395
[19:46:28] Executing task:adjust_lights with args:[]
[19:46:28] Algorithm result: [0.0, 0.0]
[19:46:28] Executing task:request_readings with args:[...]
[19:46:31] sensor 08, received 3086
[19:46:31] SCT reading: 3086
[19:46:31] Converted to: 25.05
[19:46:35] sensor 09, received 2062
[19:46:35] SCT reading: 2062
[19:46:35] Converted to: 7.96
[19:46:35] Executing task:_db_insert with args:[..., addr 08]
[19:46:35] Executing task:_mqtt_publish with args:[..., addr 08]
[19:46:35] Executing task:_db_insert with args:[..., addr 09]
[19:46:35] Executing task:_mqtt_publish with args:[..., addr 09]
[19:46:43] Executing task:request_readings with args:[...]
[19:46:45] sensor 03, received 395
[19:46:48] sensor 04, received 392
[19:46:48] Executing task:adjust_lights with args:[]
[19:46:48] Algorithm result: [-4.76, 7.21]
[19:46:51] Executing task:request_readings with args:[...]
[19:46:54] sensor 08, received 3127
[19:46:54] SCT reading: 3127
[19:46:54] Converted to: 25.82
[19:46:58] sensor 09, received 2158
[19:46:58] SCT reading: 2158
[19:46:58] Converted to: 8.94
[19:46:58] Executing task:_db_insert with args:[..., addr 08]
[19:46:58] Executing task:_mqtt_publish with args:[..., addr 08]
[19:46:58] Executing task:_db_insert with args:[..., addr 09]
[19:46:58] Executing task:_mqtt_publish with args:[..., addr 09]
[19:47:03] Executing task:request_readings with args:[...]
...
```

Παρατηρούμε ότι οι εντολές εκτελούνται με τη σειρά που χρειάζεται. Επιπρόσθετα, βλέπουμε ότι

κατά τον πρώτο κύκλο οι ληφθείσες τιμές φωτεινότητας είναι αρκετά κοντά στο threshold (που ήταν ίσο με 406 κατά αυτή τη δοκιμή), οπότε δεν χρειάστηκε ρύθμιση. Στον επόμενο κύκλο, η φωτεινότητα που διάβασε ένας από τους αισθητήρες δεν ανταποκρινόταν στο threshold, και έτσι ο αλγόριθμος επιλογής φωτισμού δίνει το αποτέλεσμα `[-4.76, 7.21]` , που υποδεικνύει κατά πόσο θα πρέπει να αλλάξει η φωτεινότητα κάθε λαμπτήρα.

Δοκιμή 6:
Κανονική λειτουργία με αλλαγές στην φωτεινότητα από εξωτερικές πηγές

Ένας σκοπός του συστήματος είναι να μπορεί να προσαρμοσθεί αυτόματα σε αλλαγές της φωτεινότητας του χώρου. Σε αυτή τη δοκιμή θα προσθέσουμε μια εξωτερική πηγή φωτισμού, η οποία θα προσωμοιώνει για παράδειγμα το φυσικό ηλιακό φως. Έπειτα θα παρουσιάσουμε τα σημεία του log file στα οποία φαίνονται αυτές οι αλλαγές, για να ελέγξουμε αν το σύστημα έρχεται σε σταθερή κατάσταση, και πόση ώρα χρειάζεται για να συμβεί αυτό.

Θα προσθέσουμε μια φωτεινή πηγή κοντά στον αισθητήρα φωτισμού με διεύθυνση '03', έτσι ώστε να επηρεάζει και τους 2 αισθητήρες:

Το log file είναι ως εξής (threshold=406):

```
...
[20:08:06] Executing task:request_readings with args:[...]
[20:08:06] sensor 03, received 428
[20:08:07] sensor 04, received 431
[20:08:07] Executing task:adjust_lights with args:[]
[20:08:08] Algorithm result: [-2.10, -3.98]
...
[20:08:10] Executing task:request_readings with args:[...]
[20:08:10] sensor 03, received 420
[20:08:11] sensor 04, received 424
[20:08:11] Executing task:adjust_lights with args:[]
[20:08:12] Algorithm result: [-0.62, -3.6]
...
[20:08:14] Executing task:request_readings with args:[...]
[20:08:14] sensor 03, received 412
[20:08:15] sensor 04, received 425
[20:08:15] Executing task:adjust_lights with args:[]
[20:08:17] Algorithm result: [6.46, -9.79]
...
[20:08:19] Executing task:request_readings with args:[...]
[20:08:19] sensor 03, received 412
[20:08:20] sensor 04, received 399
[20:08:20] Executing task:adjust_lights with args:[]
[20:08:21] Algorithm result: [0.0, 0.0]
...
```

Παρατηρούμε ότι οι αισθητήρες δίνουν επίπεδο φωτεινότητας υψηλότερο από το threshold, και το σύστημα λαμβάνει τις κατάλληλες αποφάσεις για τη ρύθμιση του φωτισμού. Η προσαρμογή έγινε μέσα σε χρονικό διάστημα 20 δευτερολέπτων. Αν λάβουμε υπ'όψη ότι χρειάστηκε αυτό το χρονικό διάστημα για επαναφορά του threshold κατά 20 μονάδες, δηλαδή μια αρκετά μικρή απόκλιση που δεν απαίτησε μεγάλες αλλαγές στην ένταση των λαμπτήρων, μπορούμε να

θεωρήσουμε ότι ικανοποιείται η απαίτηση που αφορά τις ομαλές αλλαγές.

Δοκιμή 7:
Κανονική λειτουργία με παρεμβάσεις του χρήστη μέσω του WebGUI

α) Αλλαγή του threshold

Αν κατά την κανονική λειτουργία, όταν το σύστημα είναι σε σταθερή κατάσταση, αλλάξουμε το threshold μέσω του GUI, για παράδειγμα από 406 σε 460, τότε παρατηρούμε τα εξής στο log file:

```
[20:33:42] sensor 03, received 394
[20:33:43] sensor 04, received 408
[20:33:43] Executing task:adjust_lights with args:[]
[20:33:44] Algorithm result: [0.0, 0.0]
[20:33:44] Executing task:request_readings with args:[...]
[20:33:46] Received command from WebGUI
...
[20:33:48] Executing task:request_readings with args:[...]
[20:33:48] sensor 03, received 403
[20:33:49] sensor 04, received 397
[20:33:49] Executing task:adjust_lights with args:[]
[20:33:50] Algorithm result: [11.72, 13.84]
[20:33:50] Executing task:request_readings with args:[...]
...
[20:33:58] Executing task:request_readings with args:[...]
[20:33:58] sensor 03, received 423
[20:33:59] sensor 04, received 426
[20:33:59] Executing task:adjust_lights with args:[]
[20:34:00] Algorithm result: [11.94, 6.98]
...
[20:34:02] Executing task:request_readings with args:[...]
[20:34:02] sensor 03, received 424
[20:34:03] sensor 04, received 424
[20:34:03] Executing task:adjust_lights with args:[]
[20:34:04] Algorithm result: [10.78, 8.42]
...
...
...
[20:35:22] sensor 03, received 456
[20:35:25] sensor 04, received 445
[20:35:25] Executing task:adjust_lights with args:[]
[20:35:25] Algorithm result: [-5.10, 7.73]
```

Παρατηρούμε ότι η λήψη της εντολής έγινε σε τυχαίο σημείο. Το σύστημα προσαρμόζεται σιγά σιγά, μέχρι να έρθει σε σταθερή κατάσταση, κάτι που έγινε μετά από 20 δευτερόλεπτα, δηλαδή μετά από περίπου 8 κύκλους λειτουργίας.

β) Αλλαγή της κατάστασης των λαμπτήρων

Ας υποθέσουμε πάλι ότι βρισκόμαστε σε σταθερή κατάσταση, με νέο threshold=460. Σε αυτό το σημείο θα εξετάσουμε την αντίδραση του συστήματος, αν θέσουμε έναν λαμπτήρα σε ένα σταθερό επίπεδο φωτεινότητας, π.χ. τον λαμπτήρα 7 σε φωτεινότητα 30%.

```
[20:42:12] Executing task:_db_insert with args:[..., addr 08]
[20:42:12] Executing task:_mqtt_publish with args:[..., addr 08]
[20:42:12] Executing task:_db_insert with args:[..., addr 09]
[20:42:12] Executing task:_mqtt_publish with args:[..., addr 09]
[20:42:13] Executing task:request_readings with args:[...]
[20:42:13] Received command from WebGUI
[20:42:13] sensor 03, received 457
[20:42:14] sensor 04, received 464
[20:42:14] Executing task:adjust_lights with args:[]
[20:42:15] Algorithm result: [0.0, 0]
[20:42:15] Executing task:dim_to_val with args:[30]
[20:42:17] Executing task:request_readings with args:[...]
```

Παρατηρούμε ότι η εντολή δεν εκτελείται με το που λαμβάνεται, αλλά αντίθετα τοποθετείται στην ουρά, και εκτελείται μετά από 2 δευτερόλεπτα.

Έπειτα βλέπουμε τον αλγόριθμο σε λειτουργία. Παρατηρούμε ότι οι γραμμές που αναφέρουν το Algorithm result υπονοούν ότι δε θα αποσταλεί κάποια τιμή για αλλαγή στον λαμπτήρα που είναι σε κατάσταση on. Επίσης, παρατηρούμε ότι θα πρέπει να αντισταθμίσουμε τη χειροκίνητη αλλαγή της φωτεινότητας στον αισθητήρα '03'. Το σύστημα επανήλθε σε σταθερή κατάσταση έπειτα από 4 κύκλους λειτουργίας, δηλαδή περίπου 10 δευτερόλεπτα.

```
[20:43:00] Executing task:request_readings with args:[...]
[20:43:03] sensor 03, received 443
[20:43:03] sensor 04, received 422
[20:43:04] Executing task:adjust_lights with args:[]
[20:43:04] Algorithm result: [3.65, 0]
...
```

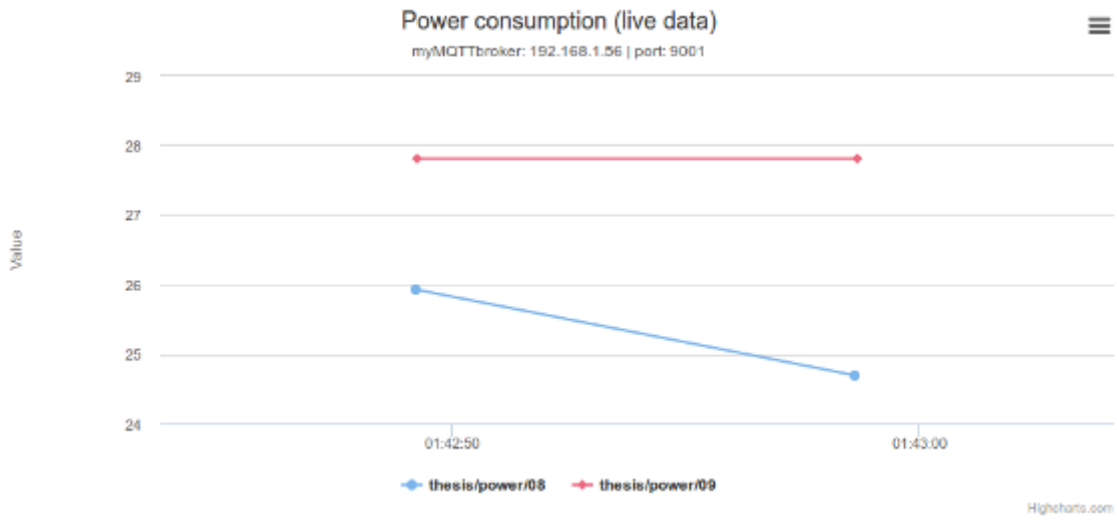
6.2 Επαλήθευση ορθής λειτουργίας του WebGUI

Η επαλήθευση της επικοινωνίας μεταξύ του GUI και του κυρίως προγράμματος έχει ήδη επαληθευτεί, μέσω των προηγούμενων δοκιμών. Σε αυτή την ενότητα θα εξετάσουμε αν το GUI λειτουργεί σωστά όσον αφορά τις υπηρεσίες παρουσίασης δεδομένων.

Real time data

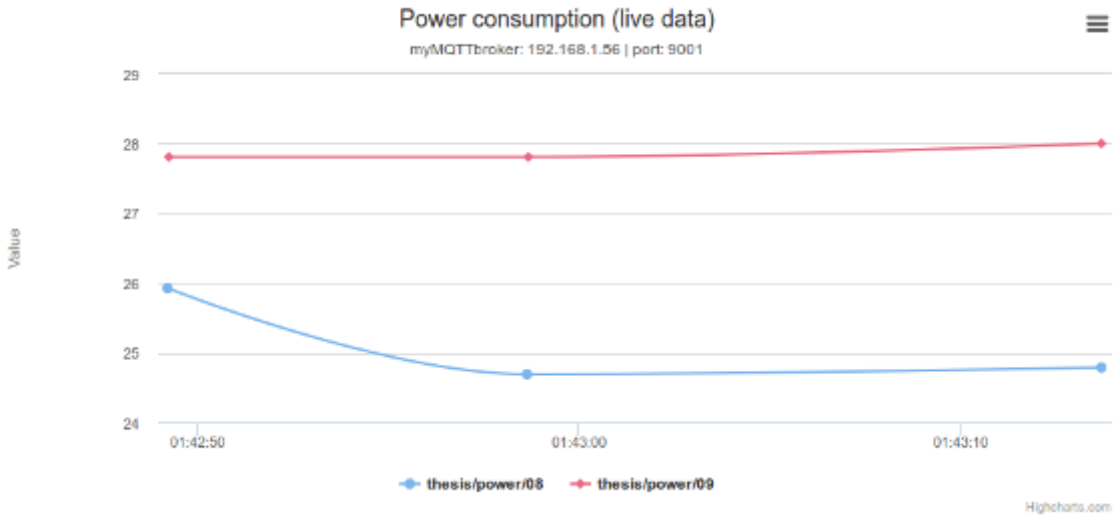
Έχοντας συνδέσει το WebGUI στον τοπικό MQTT broker που έχει εγκατασταθεί στο Raspberry Pi, και χρησιμοποιώντας websockets για την επικοινωνία του broker με τον web browser και τα client-side scripts, επαληθεύουμε ότι τα δεδομένα στέλνονται από το κυρίως πρόγραμμα στο GUI, και παρουσιάζονται με σωστό τρόπο:

```
[01:42:51] Executing task:request_readings with args:[...]
[01:42:52] sensor 08, received 3276
[01:42:52] SCT reading: 3276
[01:42:52] Converted to: 24.6923530405
[01:42:54] sensor 09, received 3778
[01:42:54] SCT reading: 3778
[01:42:54] Converted to: 27.8101810928
[01:42:55] Executing task:_db_insert with args:[..., addr 08]
[01:42:55] Executing task:_mqtt_publish with args:[..., addr 08]
[01:42:55] Executing task:_db_insert with args:[..., addr 09]
[01:42:55] Executing task:_mqtt_publish with args:[..., addr 09]
...
```



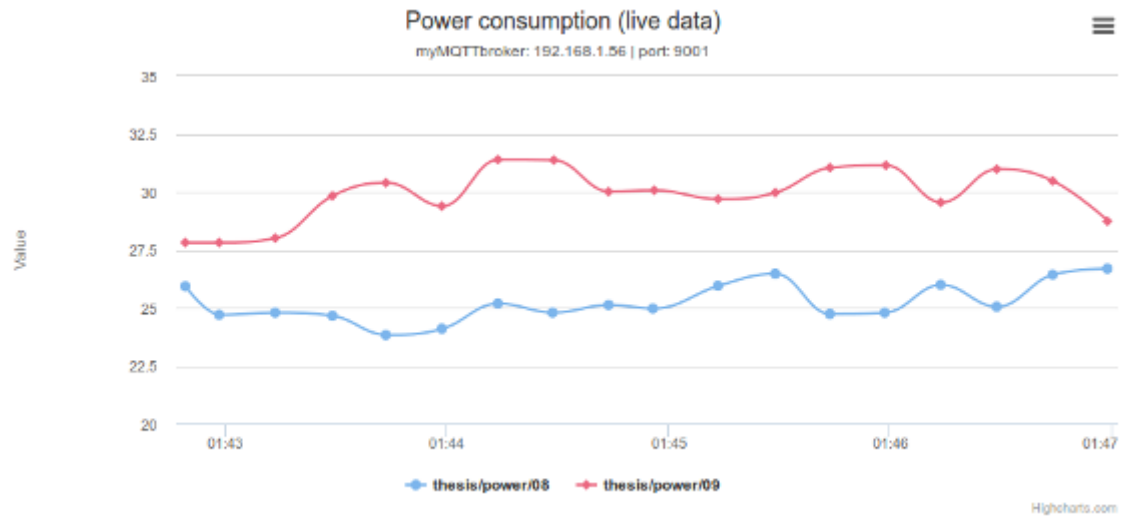
Εικόνα 6.3 - Δοκιμή real time data (1)

```
...
[01:43:06] Executing task:request_readings with args:[...]
[01:43:08] sensor 08, received 3281
[01:43:08] SCT reading: 3281
[01:43:08] Converted to: 24.7897268825
[01:43:10] sensor 09, received 3788
[01:43:10] SCT reading: 3788
[01:43:10] Converted to: 28.0047171177
[01:43:11] Executing task:_db_insert with args:[..., addr 08]
[01:43:11] Executing task:_mqtt_publish with args:[..., addr 08]
[01:43:11] Executing task:_db_insert with args:[..., addr 09]
[01:43:11] Executing task:_mqtt_publish with args:[..., addr 09]
...
```



Εικόνα 6.4 - Δοκιμή real time data (2)

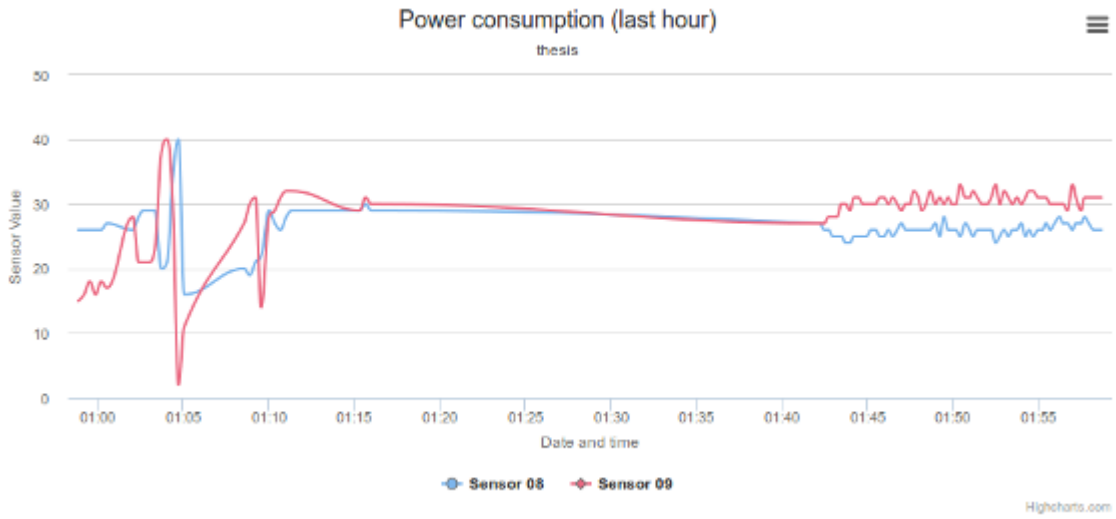
Και μετά από μερικούς κύκλους λειτουργίας:



Εικόνα 6.5 - Δοκιμή real time data (3)

Historical data

Επιλέγοντας την αναπαράσταση δεδομένων προηγούμενων μετρήσεων, το GUI δίνει τα charts που φαίνονται στις παρακάτω εικόνες.



Εικόνα 6.6 - Δοκιμή historical data

Κεφάλαιο 7 Σύνοψη

Σε αυτό το τελευταίο κεφάλαιο θα γίνει αρχικά μια αποτίμηση των αποτελεσμάτων, κατά την οποία θα συγκριθεί το τελικό προϊόν με αυτό που περιγράφηκε στο κεφάλαιο 2. Έπειτα θα απαριθμηθούν κάποιες δυνατές επεκτάσεις που θα μπορούσαν να γίνουν στο μέλλον, καθώς και κάποιες προσαρμογές που θα μπορούσαν να είχαν γίνει, αλλά παραλείφθηκαν με σκοπό την εξοικονόμηση χρόνου και πόρων.

7.1 Συμπεράσματα

Μπορούμε να ισχυριστούμε ότι το σύστημα έχει επιτύχει τον κυριότερο σκοπό του: την εξοικονόμηση ενέργειας, χωρίς να παρεμβαίνει στην αισθητική του χώρου, ή στην αποδοτικότητα, αν πρόκειται για χώρο γραφείου. Στο 6ο κεφάλαιο είδαμε ότι το σύστημα ανταποκρίνεται κατάλληλα στις αλλαγές του φυσικού φωτισμού, μειώνοντας την ένταση των λαμπτήρων που χρειάζεται για να επανέλθει ο φωτισμός σε ένα σταθερό επίπεδο.

Αυτό βέβαια δε σημαίνει ότι το σύστημα θα έχει πάντα ως αποτέλεσμα την ομοιομορφία του φωτισμού: αυτό εξαρτάται κατά πολύ από το φυσικό φως. Αν το φυσικό φως έχει αρκετά υψηλή ένταση, τότε υπάρχει η πιθανότητα όλοι οι λαμπτήρες να τεθούν σε ένταση 0%, και ακόμα η φωτεινότητα να είναι υψηλότερη από την επιθυμητή. Το να έχουμε υψηλότερη ένταση από το threshold όμως δεν είναι πρόβλημα, αρκεί να υπάρχει η βέλτιστη κατανάλωση ενέργειας. Αντίστοιχα, αν ο χρήστης έχει θέσει το επιθυμητό επίπεδο σε αρκετά υψηλή τιμή, υπάρχει περίπτωση να μην ικανοποιηθεί αυτή η τιμή, ακόμα και με όλους τους λαμπτήρες στο 100%.

Όσον αφορά την ομαλότητα στις μεταβάσεις των ρυθμίσεων, οι δοκιμές έδειξαν πως το χρονικό διάστημα μεταξύ τους είναι επαρκές, ώστε να μην είναι αντιληπτές από τον χρήστη. Το σημείο-κλειδί εδώ είναι το εξής: επειδή σπάνια θα υπάρχουν πολύ απότομες αλλαγές στην ένταση, το σύστημα θα καταφέρνει να εξισορροπήσει μια Χ αλλαγή στην ένταση σταδιακά, καθώς αυτή η αλλαγή συμβαίνει. Δηλαδή, αν το επίπεδο φωτεινότητας στον χώρο μεταβληθεί από μια ένταση Α σε μια ένταση Β με πολύ μικρά βήματα, τότε το σύστημα θα ανταποκρίνεται γρήγορα σε αυτά τα βήματα, αλλά ο χρήστης δε θα αντιλαμβάνεται αυτές τις πολύ μικρές αλλαγές. Έτσι, μπορεί να έχουμε αλλαγή στο επίπεδο φωτισμού ακόμα ακόμα και κάθε 2 δευτερόλεπτα, η οποία θα είναι επαρκώς μικρή.

Μια άλλη απαίτηση που ικανοποιήθηκε είναι η προσαρμογή του συστήματος σε αλλαγές. Χάρη στο setup stage και στον αλγόριθμο μηχανικής μάθησης που εφαρμόσθηκε στους μετρητές κατανάλωσης, μπορούμε να αλλάξουμε την τοπολογία και τα χαρακτηριστικά του συστήματος, και με μια επανεκκίνηση να ρυθμιστούν όλες οι κρίσιμες παράμετροι καταλλήλως. Επίσης, ακόμα και αν προστεθεί κάποιο επιπλέον τερματικό, χάρη στην object oriented υλοποίηση των τερματικών στον κώδικα του gateway, αρκεί μόνο μια γραμμή κώδικα για να είναι το σύστημα up to date με την προσθήκη.

Επιπλέον, επιτεύχθηκε η ανάπτυξη ενός πλήρως λειτουργικού GUI, το οποίο προσφέρει ακριβώς τις υπηρεσίες που αναφέρθηκαν στην εισαγωγή. Έγινε επίσης δυνατή η διασύνδεση του συστήματος με το GUI και με τη βάση δεδομένων σε τοπικό επίπεδο, χωρίς να είναι αναγκαία η σύνδεση στο internet.

Ένα μειονέκτημα του συστήματος που είναι αισθητό κατά τη λειτουργία του είναι το γεγονός ότι ο αλγόριθμος επιλογής φωτισμού δεν είναι σε "συνεννόηση" με τα dimmer. Αυτό σημαίνει ότι δε μπορεί να αντιληφθεί αν ένας λαμπτήρας είναι στο 0 ή 100% της φωτεινότητας και υπάρχει περίπτωση να συνεχίζει να ζητά περαιτέρω αύξηση ή μείωση της φωτεινότητας, πέρα από τα δυνατά όρια. Αυτό μπορεί να επιλυθεί αν ζητάμε σε κάθε κύκλο από το dimmer να στείλει την τρέχουσα τιμή που εφαρμόζει, ή να ρυθμίσουμε το dimmer έτσι ώστε να ειδοποιεί στην περίπτωση που ζητάμε ρύθμιση πέρα από τα όριά του. Αυτή η ιδέα επεκτείνεται περισσότερο στην επόμενη ενότητα.

Τέλος, όσον αφορά την απαίτηση για χαμηλό κόστος, έχουμε τα εξής:

Κοστος υλοποίησης (CAPEX)

Το κόστος υλοποίησης του συστήματος αποτελείται από το κόστος αγοράς των απαραίτητων εξαρτημάτων και το κόστος εγκατάστασης, το οποίο είναι πολύ μικρό. Το κόστος των εξαρτημάτων δεν είναι συγκεκριμένο, και εξαρτάται από τον χώρο που θα γίνει η εγκατάσταση και τις εκάστοτε απαιτήσεις όσον αφορά τον αριθμό λαμπτήρων, αισθητήρων, την ακρίβεια κλπ. Σε γενικές γραμμές, τα εξαρτήματα που απαιτούνται είναι τα τερματικά, τυπικά 6-8 μικροελεγκτές για τους αισθητήρες και τα dimmer, το gateway, και λοιπά ηλεκτρονικά εξαρτήματα. Επίσης, δεν είναι ανάλογο μεγέθους του συστήματος. Για παράδειγμα, μεγάλο μέρος του κόστους προέρχεται από το gateway, το οποίο είναι μοναδικό στο σύστημα.

Κόστος λειτουργίας (OPEX)

Για το κόστος λειτουργίας, το σύστημά μας προδιαγράφει ότι θα είναι πολύ μικρότερο ή και μηδαμινό σε σχέση με το κόστος που εξοικονομείται από τη μείωση της κατανάλωσης. Όπως και το κόστος υλοποίησης, το κόστος λειτουργίας εξαρτάται από το πλήθος των τερματικών που χρησιμοποιούνται, καθώς και από τα περιφερειακά που χρησιμοποιεί το κάθε τερματικό. Σε γενικές γραμμές, αξίζει να σημειωθεί ότι ένας μικροελεγκτής Arduino στα 5V παρέχει 20mA σε κάθε pin. Η λειτουργία προσαρμόσθηκε κατάλληλα μέσω κώδικα έτσι ώστε όταν ένα τερματικό δεν χρησιμοποιείται, να καταναλώνει όσο το δυνατόν λιγότερη ισχύ. Όταν, για παράδειγμα, χρειαστεί να στείλουμε μια μέτρηση στο gateway, το εκάστοτε τερματικό ενεργοποιεί τα απαραίτητα περιφερειακά του, κάνει ανάγνωση της μέτρησης, την στέλνει και έπειτα μπαίνει σε idle mode μέχρι να χρειαστεί να τεθεί πάλι σε λειτουργία.

7.2 Επεκτάσεις

Υπάρχουν πολλές δυνατές επεκτάσεις που μπορούν να εφαρμοσθούν, έτσι ώστε να αυξηθεί η απόδοση και η ακρίβεια του συστήματος, καθώς και η περαιτέρω εξοικονόμηση ενέργειας.

Κατ'αρχάς, είναι δυνατόν να μειώσουμε την πολυπλοκότητα και να αυξήσουμε την ακρίβεια του συστήματος αν αφαιρέσουμε τους μετρητές κατανάλωσης. Έπειτα, το ποσοστό κατανάλωσης του κάθε λαμπτήρα θα το μετράμε ως εξής: με τον ίδιο τρόπο που αιτούμε μια μέτρηση από κάποιον αισθητήρα, θα αιτούμε το τρέχον dim level από το dimmer. Το dimmer θα αποστέλλει πληροφορία σχετικά με το ποσοστό της τάσης που παρέχει στον λαμπτήρα, δηλαδή τον αριθμό που δίνει την αναμονή του timer μέχρι να στείλει παλμό στο TRIAC. Έτσι, εφαρμόζοντας μια διαδικασία αντίστροφη από αυτή που μας έδωσε τιμές του dimmer για ποσοστά κατανάλωσης 0-100%, θα μπορούμε να έχουμε μια κατά πολύ ακριβέστερη εκτίμηση

σχετικά με τη δαπάνη ενέργειας από τον κάθε λαμπτήρα. Όπως έχουμε αναφέρει όμως, το Arduino που υλοποιεί το dimmer τρέχει ήδη πολλές διεργασίες, και αν του αναθέσουμε περαιτέρω υπάρχει περίπτωση να μειωθεί η απόδοσή του (π.χ. flickering). Για να γίνουν τα παραπάνω υλοποιήσιμα, θα πρέπει να χρησιμοποιήσουμε έναν πιο ισχυρό μικροελεγκτή (π.χ. Intel Galileo). Από τη μία έχουμε σημαντική διαφορά στην τιμή, αλλά και μείον έναν μετρητή κατανάλωσης ανά λαμπτήρα.

Μια ακόμα επέκταση θα μπορούσε να γίνει με την εισαγωγή περαιτέρω μηχανικής μάθησης στο setup stage. Μπορούμε για παράδειγμα να εξάγουμε τα βάρη για διάφορα configurations, δίνοντας τα σε έναν αλγόριθμο, μαζί με σχετικές πληροφορίες (αποστάσεις μεταξύ λαμπτήρων - αισθητήρων, τοποθεσία πηγής φυσικού φωτισμού, εμπόδια, προδιαγραφές του κάθε λαμπτήρα κ.ά.). Ο αλγόριθμος θα μπορεί να εφαρμόζει παλινδρόμηση με όλες αυτές τις μεταβλητές, έτσι ώστε να δώσει μια καλή εκτίμηση για τα βάρη της τρέχουσας διάταξης. Αυτό εξοικονομεί χρόνο, καθώς δε χρειάζεται σε κάθε δοκιμή διάταξης να γίνεται εκτέλεση του setup, μέχρι να βρεθεί η κατάλληλη διάταξη, αλλά αντίθετα θα απαιτούνται μόνο κάποιες αλλαγές σε παραμέτρους.

Ακόμα μια σημαντική βελτίωση στο σύστημα θα ήταν η αντικατάσταση της RF επικοινωνίας μέσω modules με επικοινωνία μέσω τοπικού δικτύου, δηλαδή να αντικαταστήσουμε τα NRF με WiFi ή Ethernet shields στα Arduino, ενώ στο Raspberry Pi να χρησιμοποιήσουμε ethernet ή ένα δέκτη WiFi. Αυτό έχει ως αποτέλεσμα να μη χρειάζεται να περιμένει το σύστημα την εκτέλεση μιας αποστολής ή λήψης από το NRF, αλλά αντίθετα να στέλνει συγχρόνως σε πολλά τερματικά. Η δυνατότητα αυτή μπορεί εύκολα να υλοποιηθεί χρησιμοποιώντας multithreading ή multiprocessing, ή ακόμα καλύτερα με event loops και ασύγχρονες ρουτίνες, κάτι το οποίο προσφέρει πλέον η Python 3, και μοιάζει με τον τρόπο που διαχειρίζεται η JavaScript τα events σε μια ιστοσελίδα. Αυτήη βελτίωση έχει όμως και μερικά μειονεκτήματα: υψηλότερο κόστος, δυσκολία στην υλοποίηση, καθώς και εξάρτηση από τη συνδεσιμότητα στο internet.

Τέλος, πολύ χρήσιμη θα ήταν η μετατροπή του συστήματος, ώστε να εκτελεί αυτόματη διευθυνσιοδότηση. Το παρόν σύστημα έχει hard-coded τις διευθύνσεις σε κάθε τερματικό, κάτι που σημαίνει ότι για να προσθέσουμε ένα τερματικό πρέπει να αλλάξουμε τη διεύθυνση τον κώδικα και μετά να τον "γράψουμε" σε αυτό. Η λύση που προτείνεται είναι, κατά την εκκίνηση τα τερματικά να δημιουργούν μια τυχαία ακολουθία αριθμών και να τη στέλνουν στο gateway (σαν προσωρινή διεύθυνση). Έπειτα, το gateway απαντά σε κάθε τερματικό, στέλνοντας πίσω την ακολουθία αυτή μαζί με μια διεύθυνση. Το τερματικό αποθηκεύει αυτή τη διεύθυνση και την χρησιμοποιεί σαν δική του.

Κεφάλαιο 8 Βιβλιογραφία - Παραπομπές

[1] Arduino – Home
https://www.arduino.cc/

[2] NRF24L01(+) datasheet
http://www.nordicsemi.com/eng/content/download/2730/34105/file/nRF24L01_Product_Specification_v2_0.pdf

[3] Arduino Playground: nRF24L01+
http://playground.arduino.cc/InterfacingWithHardware/Nrf24L01

[4] GitHub: TMRh20 / RF24
https://github.com/tmrh20/RF24

[5] GitHub: stanleyseow / RF24
https://github.com/stanleyseow/RF24

[6] Dimming 230V AC with Arduino
http://alfadex.com/2014/02/dimming-230v-ac-with-arduino-2/

[7] Stackexchange forum: Arduino 230v Light bulb dimming
http://electronics.stackexchange.com/questions/59615/arduino-230v-light-bulb-dimming

[8] Instructables: Arduino controlled light dimmer
http://www.instructables.com/id/Arduino-controlled-light-dimmer-The-circuit/

[9] Arduinotechniq: AC light dimmer with Arduino
http://arduinotehniq.blogspot.gr/2014/10/ac-light-dimmer-with-arduino.html

[10] Bristolwatch: Arduino Hardware Interrupts Control AC Power
http://www.bristolwatch.com/arduino/arduino_power_control.htm

[11] Bristolwatch: In Depth Look at AC Power Control with Arduino
http://www.bristolwatch.com/arduino/arduino1.htm

[12] Bristolwatch: Zero-Crossing Detectors Circuits and Applications
http://www.bristolwatch.com/ele2/zero_crossing.htm

[13] Arduino forums: 230v AC Triac Dimmer. Works, but flickers on "fade-up"
http://forum.arduino.cc/index.php?topic=111196.0

[14] Arduino&stuff: Timer interrupts
https://arduinodiy.wordpress.com/2012/02/28/timer-interrupts/

[15] Arduino 101: Timers and Interrupts
http://letsmakerobots.com/content/arduino-101-timers-and-interrupts

[16] Arduino Playground: PhotoResistor
http://playground.arduino.cc/Learning/PhotoResistor

[17] OpenEnergyMonitor: CT sensors - Interfacing with an Arduino
https://openenergymonitor.org/emon/buildingblocks/ct-sensors-interface

[18] OpenEnergyMonitor: Installation and Calibration
https://openenergymonitor.org/emon/buildingblocks/ct-and-ac-power-adaptor-installation-and-calibration-theory

[19] GitHub: openenergymonitor / EmonLib
https://github.com/openenergymonitor/EmonLib

[20] Raspberry Pi - Teach, Learn, and Make with Raspberry Pi
https://www.raspberrypi.org/

[21] Instructables: Raspberry Pi: Python scripting the GPIO
http://www.instructables.com/id/Raspberry-Pi-Python-scripting-the-GPIO/

[22] Tomas Holderness: Using NodeJS to develop a temperature server with the Raspberry Pi
https://tomholderness.wordpress.com/2013/01/03/raspberry-pi-temperature-server/

[23] Instructables: Uber Home Automation w/ Arduino & Pi
http://www.instructables.com/id/Uber-Home-Automation-w-Arduino-Pi/

[24] BlueflameSoftware: Column chart with data from MySQL using Highcharts
http://blueflame-software.com/column-chart-with-data-from-mysql-using-highcharts/

[25] Raspberry Pi forums: NRF24L01+ in Python
https://www.raspberrypi.org/forums/viewtopic.php?f=45&t=85504

[26] GitHub: jpbarraca / pynrf24
https://github.com/jpbarraca/pynrf24

[27] xAppSoftware Blog: Six Steps to install mosquitto 1.4.2 with websockets on debian wheezy
http://www.xappsoftware.com/wordpress/2015/05/18/six-steps-to-install-mosquitto-1-4-2-with-websockets-on-debian-wheezy/comment-page-1/

[28] Paho MQTT: Python Client
https://eclipse.org/paho/clients/python/

[29] MySQLdb Python library User's Guide
http://mysql-python.sourceforge.net/MySQLdb.html

[30] Python Watchdog Documentation
http://pythonhosted.org/watchdog/

[31] NumPy and SciPy Documentation
http://docs.scipy.org/doc/

[32] Matplotlib Documentation

http://matplotlib.org/contents.html

[33] PyMOTW: threading: Manage concurrent threads
https://pymotw.com/2/threading/index.html

[34] PyMOTW: Queue – A thread-safe FIFO implementation
https://pymotw.com/2/Queue/index.html

[35] Flask Documentation
http://flask.pocoo.org/docs/0.10/

[36] Flask-Bootstrap Documentation
https://pythonhosted.org/Flask-Bootstrap/

[37] Grinberg, Miguel. Flask Web Development: Developing Web Applications with Python. 2014, O'Reilly Media

[38] Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. 1995, Addison-Wesley

Παράρτημα Α Λίστα - περιγραφή εξοπλισμού

Τερματικά

Αισθητήρας φωτισμού

- Arduino Micro
  https://www.arduino.cc/en/Main/ArduinoBoardMicro
- NRF24L01
  http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01

Μετρητής κατανάλωσης

- Arduino Nano
  https://www.arduino.cc/en/Main/ArduinoBoardNano
- NRF24L01
  http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01
- YHDC SCT-013 (30A/1V)
  http://www.yhdc.com/en/product/320/

Dimmer

- Arduino UNO
  https://www.arduino.cc/en/Main/ArduinoBoardUno
- NRF24L01
  http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01
- DB105 Rectifier
  http://www.rectron.com/data_sheets/db101-107.pdf
- 4N35 Opto-coupler
  http://www.vishay.com/docs/81181/4n35.pdf
- MOC3062 Opto-TRIAC
  http://pdf.datasheetcatalog.com/datasheet/motorola/MOC3061.pdf
- MAC97A8 TRIAC
  http://www.nxp.com/documents/data_sheet/MAC97A8.pdf

Gateway

- Raspberry Pi 2 Model B
  https://www.raspberrypi.org/products/raspberry-pi-2-model-b/
- NRF24L01+
  https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P

Παράρτημα Β Πηγαίος κώδικας

Τερματικά

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include "printf.h"

char SendPayload[31] = "temp";
char ReceivePayload[32];
char SendValue[31] = " ";
char myAddr[] = "03";
char gwAddr[] = "01";
char bcAddr[] = "99";
char srcType[] = "01";
char dataCommand[] = "01";
int role = 0;

#define ARRAY_SIZE(array) (sizeof((array))/sizeof((array[0])))

//DATA
typedef struct {
  // id of the device -> max is 32 characters
  char id[16];
  // type of the device -> a simple int
  int16_t type;
  // table of value -> 8 values max
  int16_t val[7];
}
Payload;
Payload p;

int msg[1];
float sensor;
const int sensorPin = A1;
int sensorMax = 1000;
int sensorMin = 10;
long unsigned sensorValue;

// Set up nRF24L01 radio on SPI pin for CE, CSN
RF24 radio(9,10);
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

char receivePayload[32];
uint8_t counter=0;

void setup() {

  Serial.begin(115200);
  printf_begin();
  pinMode(13, OUTPUT);
  digitalWrite(13, LOW);
```

```
  //CONFIGURE RADIO
  radio.begin();
  radio.enableDynamicPayloads();
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setChannel(0x4c);
  radio.setCRCLength(RF24_CRC_8);
  radio.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1,pipes[1]);
  radio.powerUp();
  radio.startListening();
  //radio.printDetails();

  for (int i=0; i < 8; i++) {
    p.val[i] = 0;
  }
}

void loop() {
    while (true) {
    while (!(radio.available())) {
        //wait!
    }
    bool done = false;
    while (!done) {
        // Fetch the payload, and see if this was the last one.
        uint8_t len = radio.getDynamicPayloadSize();
        done = radio.read( &ReceivePayload, len );
    }
    char destAddr[] = {ReceivePayload[0], ReceivePayload[1], '\0'};
    if (strcmp(destAddr, myAddr) != 0) { continue; }
    char role_char[] = {ReceivePayload[2], ReceivePayload[3], '\0'};
    role = atol(role_char);
    if (role == 1) {
      readSensor();
      sendOverRadio();
    }
    radio.startListening();
  }
}

void sendOverRadio() {
  uint8_t data1 = 0; ;
  uint16_t nodeID = 2;

  char outBuffer[32]="";
  unsigned long send_time, rtt = 0;

  Serial.print("NodeID: ");
  Serial.println(nodeID);
```

```
  unsigned long time = micros();

  strncpy(SendPayload, gwAddr, 31);
  strcat(SendPayload, myAddr);
  strcat(SendPayload, srcType);
  strcat(SendPayload, dataCommand);
  dtostrf(sensorValue,1,0,SendValue);
  if ((sensorValue>=0) && (sensorValue<10)) {
    strcat(SendPayload, "000");
  }
  else if ((sensorValue>=10) && (sensorValue<100)){
    strcat(SendPayload,"00");
  }
  else if ((sensorValue>=100) && (sensorValue<1000)) {
    strcat(SendPayload, "0");
  }
  strcat(SendPayload, SendValue);
  radio.stopListening();
  digitalWrite(13, HIGH);
  if ( !radio.write(&SendPayload,strlen(SendPayload))) {
    printf("Send failed!!!\n\r");
  }
  else {
    printf("Send successful\n\r");
  }
  digitalWrite(13, LOW);
  delay(20);
}

void readSensor() {
  sensorValue = analogRead(sensorPin);
  sensorValue = map(sensorValue, sensorMin, sensorMax, 0, 800);
  sensorValue = constrain(sensorValue, 0, 800);

  p.val[3] = round(sensorValue)*100;
  sensor = p.val[3]/100;
  Serial.println(sensorValue);
}
```

Μετρητής κατανάλωσης

```
#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include "printf.h"
#include "EmonLib.h"

EnergyMonitor emon1;

char SendPayload[31] = "temp";
```

```
char ReceivePayload[32];
char SendValue[31] = " ";
char myAddr[] = "09";
char gwAddr[] = "01";
char bcAddr[] = "99";
char srcType[] = "02";
char dataCommand[] = "01";
double Rb = 1322.5; //resistance of 40W light bulb
bool action = false;
int role = 0;

#define ARRAY_SIZE(array) (sizeof((array))/sizeof((array[0])))

//DATA
typedef struct {
  // id of the device -> max is 32 characters
  char id[16];
  // type of the device -> a simple int
  int16_t type;
  // table of value -> 8 values max
  int16_t val[7];
}
Payload;
Payload p;

int msg[1];
double sensor;
const int CTpin = 1; // Analog Pin 0
double sensorValue;

// Set up nRF24L01 radio on SPI pin for CE, CSN
RF24 radio(9,10);
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };
char receivePayload[32];

void setup() {
  delay(7000);
  Serial.begin(115200);
  printf_begin();

  // Setup CT Sensor, calibration values:
  // 10 kOhm ---> 1.24
  // 330 Ohm ---> 6.0606
  emon1.current(CTpin, 60.6060); // Current: input pin, calibration.

  radio.begin();
  radio.enableDynamicPayloads();
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setChannel(0x4c);
  radio.setRetries(5,15);
```

```
  radio.setCRCLength(RF24_CRC_8);
  radio.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1,pipes[1]);

  radio.powerUp();
  radio.startListening();
  //radio.printDetails();

  for (int i=0; i < 8; i++) {
    p.val[i] = 0;
  }
}

void loop() {
    while (true) {
    while (!(radio.available())) {
        //wait!
    }
    bool done = false;
    while (!done) {
        // Fetch the payload, and see if this was the last one.
        uint8_t len = radio.getDynamicPayloadSize();
        done = radio.read( &ReceivePayload, len );
    }
    char destAddr[] = {ReceivePayload[0], ReceivePayload[1], '\0'};
    if (strcmp(destAddr, myAddr) != 0) { continue; }
    char role_char[] = {ReceivePayload[2], ReceivePayload[3], '\0'};
    role = atol(role_char);
    if (role == 1) {
      readSensor();
      sendOverRadio();
    }
    radio.startListening();
  }
}

void sendOverRadio() {
  uint8_t data1 = 0; ;
  uint16_t nodeID = 2;

  char outBuffer[32]="";
  unsigned long send_time, rtt = 0;

  Serial.print("NodeID: ");
  Serial.println(nodeID);
  unsigned long time = micros();

  strncpy(SendPayload, gwAddr, 31);
  strcat(SendPayload, myAddr);
  strcat(SendPayload, srcType);
  strcat(SendPayload, dataCommand);
  dtostrf(sensorValue,1,0,SendValue);
```

```
  if ((sensorValue>=0) && (sensorValue<10)) {
    strcat(SendPayload, "000");
  }
  else if ((sensorValue>=10) && (sensorValue<100)){
    strcat(SendPayload,"00");
  }
  else if ((sensorValue>=100) && (sensorValue<1000)) {
    strcat(SendPayload, "0");
  }
  strcat(SendPayload, SendValue);
  radio.stopListening();
  digitalWrite(13, HIGH);
  if ( !radio.write(&SendPayload,strlen(SendPayload))) {
    printf("Send failed!!!\n\r");
  }
  else {
    printf("Send successful\n\r");
  }
  digitalWrite(13, LOW);
  delay(20);
}

void readSensor() {

  sensorValue = emon1.calcIrms(10000);
  sensorValue = sensorValue*sensorValue*Rb*100;
  sensorValue = int (sensorValue);
  Serial.println(sensorValue);

  p.val[3] = (sensorValue)*100;
  sensor = p.val[3]/100;
}
```

Dimmer

```
#include <SPI.h>
#include <stdio.h>
#include <stdlib.h>
#include "nRF24L01.h"
#include "RF24.h"
#include "printf.h"

// NRF24L01
// Set up nRF24L01 radio on SPI pin for CE, CSN
RF24 radio(9,10);
// Example below using pipe5 for writing
//const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0x7365727631LL };
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

char ReceivePayload[32];
```

```
char myAddr[] = "07";
char gwAddr[] = "01";
void dim_routine_increment (int& dim_level, int& role, int& incr_val);
void dim_routine_fade_value (int& dim_level, int& next_level);
void dim_routine_value (int& dim_level, int& next_level);
int incr_val = 0;
int next_level;
int role = 0;
bool action = false;

#define DETECT 0  // interrupt 0 = pin 2
#define GATE   8  // triac gate

#define PULSE  8 // triac gate trigger pulse width (timer counts)

#define MAX_LEVEL 600
#define MIN_LEVEL 50
#define FADE_DELAY 10

int dim_level;
boolean increment_pause = false;
boolean start_low_fix = true;
char *ptr;

void setup()
{

  Serial.begin(115200);
  dim_level = 300;
  next_level = dim_level;
  printf_begin();
  //CONFIGURE RADIO
  radio.begin();
  radio.enableDynamicPayloads();
  radio.setAutoAck(false);
  radio.setDataRate(RF24_250KBPS);
  radio.setPALevel(RF24_PA_MAX);
  radio.setChannel(0x4c);
  radio.setCRCLength(RF24_CRC_8);
  radio.openWritingPipe(pipes[0]);
  radio.openReadingPipe(1,pipes[1]);
  radio.powerUp();
  radio.startListening();
  //moc3022 setup:
  pinMode(GATE, OUTPUT);   // triac gate control
  digitalWrite(GATE, HIGH); // turn off triac gate

/* Timer Stuff */
cli();
  TCCR1A = 0x00;
  TCCR1B = 0x00;
  TIMSK1 = 0x03;
```

```
  OCR1A  = dim_level;
sei();

attachInterrupt(DETECT, zero_cross_detect, RISING);
}

ISR(TIMER1_OVF_vect) {
  TCCR1B = 0x00;           // stop timer
  digitalWrite(GATE, LOW); // turn off triac gate
}

ISR(TIMER1_COMPA_vect) {
  digitalWrite(GATE, HIGH);  // set triac gate to high
  TCNT1 = 65536-PULSE;       // trigger pulse width
}

void zero_cross_detect() {
   start_low_fix = false;
   TCNT1  = 0;          // reset timer - count from zero
   OCR1A  = dim_level;
   TCCR1B = 0x04;       // start timer with 256 prescale
}



void loop()
{
  if (start_low_fix) // waiting for first ZC
      return;

  while (true){
  while (!(radio.available())) {
    }
      bool done = false;
      while (!done) {
        uint8_t len = radio.getDynamicPayloadSize();
        done = radio.read( &ReceivePayload, len );
      }
        char destAddr[] = {ReceivePayload[0], ReceivePayload[1], '\0'};
        if (strcmp(destAddr, myAddr) != 0) { continue; }
        char role_char[] = {ReceivePayload[2], ReceivePayload[3], '\0'};
        char next_level_char[] = {ReceivePayload[4], ReceivePayload[5],
                                  ReceivePayload[6], '\0'};
        role = atol(role_char);
        if  (role == 1)
        {
          incr_val = atol(next_level_char);
          if (incr_val != 0) {
            dim_routime_increment(dim_level, role, incr_val);
          }
        }
        else if (role == 2)
```

```
        {
          incr_val = atol(next_level_char);
          if (incr_val != 0) {
            dim_routime_increment(dim_level, role, incr_val);
          }
        }
        else if (role == 4)
        {
          next_level = atol(next_level_char);
          if (next_level != dim_level) {
            dim_routine_value(dim_level, next_level);
          }
        }
        else if (role == 3)
        {
          next_level = atol(next_level_char);
          if (next_level != dim_level) {
            dim_routine_fade_value(dim_level, next_level);
          }
        }
        role = 0;
        strncpy(ReceivePayload, gwAddr, 32);
  }
}

void dim_routime_increment(int& dim_level, int& role, int& incr_val) {

  if (role == 1) {
    int next = dim_level + incr_val;
    if (next > MAX_LEVEL) {
      next = MAX_LEVEL;
    }
    while (dim_level < next) {
      dim_level++;
      delay(50);
    }
  }
  else if (role == 2) {
    int next = dim_level - incr_val;
    if (next < MIN_LEVEL) {
      next = MIN_LEVEL;
    }
    while (dim_level > next) {
      dim_level--;
      delay(50);
    }
  }
}

void dim_routine_fade_value (int& dim_level, int& next_level) {

 if (next_level > MAX_LEVEL) { next_level = MAX_LEVEL; }
```

```
 else if (next_level < MIN_LEVEL) { next_level = MIN_LEVEL; }

 if (dim_level > next_level) {
   while (dim_level > next_level) {
     dim_level--;
     delay(20);
   }
 } else if (dim_level < next_level) {
   while (dim_level < next_level) {
     dim_level++;
     delay(20);
   }
 }
}

void dim_routine_value (int& dim_level, int& next_level) {

  if (next_level > MAX_LEVEL) { next_level = MAX_LEVEL; }
  else if (next_level < MIN_LEVEL) { next_level = MIN_LEVEL; }

  dim_level = next_level;
}
```

Gateway

Δομή

```
energy_efficient_lights/
  |-- __init__.py
  |-- setup.py
  |-- config.py
  |__ components/
        |-- __init__.py
        |-- devices.py
        |-- functions.py
        |-- servers.py
  |__ extern/
        |-- nrf24.py
  |__ plots/
        |-- __init__.py
        |-- power_estimate.py
        |-- dim_table.json
        |-- poly_fit.py
        |-- dimmer_auto.py
```

/config.py

```
import os
```

```
import paho.mqtt.client as mqtt
import MySQLdb
import json
from .extern import NRF24

"""
The DIM_TABLE constant holds the value that the dimmer has to
be set in order to achieve a specific light intensity level.
e.g. for i=20% we need DIM_TABLE[i/10]=460

"""

basedir = os.path.abspath(os.path.dirname(__file__))
dim_dir = os.path.join(basedir, 'plots', 'dim_table.json')
with open(dim_dir, 'r') as f:
    dim_table_ints = json.loads(f.read())
DIM_TABLE = [str(i) for i in dim_table_ints]
GET_READ = '01' # command to request reading from a sensor
GATEWAY_ADDR = '01' # our address
DIM_MIN = 50
DIM_MAX = 600
DIM_INCR_UP = '02' # command to request an intensity change by increment
up..
DIM_INCR_DOWN = '01' # .. and increment down
DIM_TO_VAL = '04' # command to dim to a specific value
THRESH_MIN = 0
THRESH_MAX = 800
NODE_TIMEOUT = 1.5 # max wait time to receive reading in seconds
OPERATION_CYCLE = 20 # duration of each cyle of operation in seconds
MAX_INCR = 40

CONFIGPATH = os.path.join(basedir, 'thesis.conf')
GUI_CONF_DIR = os.path.join(basedir, 'web_server', 'data')
GUI_CONF_PATH = os.path.join(GUI_CONF_DIR, 'gui.conf')
SCT_TRAIN_DATA_DIR = os.path.join(basedir, 'sct_train_data.txt')
SEND_DELAY = 1.0/2

# MySQL setup
SQL_HOST = 'localhost'
SQL_USER = 'root'
SQL_PW = os.environ.get('MYSQL_PW')
SQL_DB = 'thesis'
SQL_TABLE = 'power'
# instantiation
db = MySQLdb.connect(host=SQL_HOST, user=SQL_USER,
                     passwd=SQL_PW, db=SQL_DB)
cur = db.cursor()

# MQTT setup
MQTT_HOST = 'localhost'
MQTT_PORT = 1884
MQTT_BASE = 'thesis/power/'
```

```
# instantiation
mqttc = mqtt.Client()
mqttc.connect(MQTT_HOST, MQTT_PORT, 60)
mqttc.loop_start()

# NRF24 setup-instantiation
pipes = [[0xf0, 0xf0, 0xf0, 0xf0, 0xe1],
        [0xf0, 0xf0, 0xf0, 0xf0, 0xd2]]
radio = NRF24()
radio.begin(0, 0, 25, 18) #set gpio 25 as CE pin
radio.setRetries(5, 15)
radio.setPayloadSize(32)
radio.enableDynamicPayloads()
radio.setChannel(0x4c)
radio.setDataRate(NRF24.BR_250KBPS)
radio.setPALevel(NRF24.PA_MAX)
radio.setAutoAck(False)
radio.setCRCLength(NRF24.CRC_8)
radio.openWritingPipe(pipes[0])
radio.openReadingPipe(1, pipes[1])
radio.startListening()
radio.stopListening()
pipe = [0]
```

/setup.py

```
import json
import time
from .config import *
from .components.devices import *


def read_init_data():
    """
    Read the current data provided by the web interface,
    store them accordingly and return a dict with references
    to each instance or class attribute

    :rtype: dict

    """
    with open(GUI_CONF_PATH, 'r') as f:
        init_data_json = f.read()
    init_data = json.loads(init_data_json)

    gui_data = init_data

    for key, val in init_data.items():
        if key=='thresh':
            gui_data[key] = Light.thresh = int(val)
```

```
        elif key[9:]=='state':
            gui_data[key] = Light.mapping.get(key[6:8]).state = val
        elif key[9:]=='int':
            gui_data[key] = Light.mapping.get(key[6:8]).intens = int(val)
if val is not None else 50

    return gui_data

def setup():
    """
    For each light that has been instantiated, increment its
    intensity level by 10% (the next value in DIM_TABLE), while
    each other light is at 0%. For each incrementation, request
    a reading from each of the LightSensors that have been
    instantiated, and store it to its setup_table (the index
    depends on the Light instance). Finally, store each sensor
    setup_table in a file in JSON format.

    """
    for bulb_i, bulb in enumerate(Light.instances):
        for b in Light.instances:
            b.dim_to_val(0)
        for i in xrange(len(DIM_TABLE)):
            bulb.dim_to_val(i*10)
            for sensor in LightSensor.instances:
                sensor.get_reading()
                reading = sensor.current_read
                print reading
                sensor.setup_table[bulb_i].append(reading)
                print sensor.setup_table

    config = {}
    for i, sensor in enumerate(LightSensor.instances):
        config[sensor.addr] = sensor.setup_table
    config_json = json.dumps(config)
    print config

    with open(CONFIGPATH, 'w') as f:
        f.write(config_json)

def setup_from_file():
    """
    Instead of setting up the system from scratch, it can
    be set up by a previous saved configuration. This conf
    is stored in each LightSensor's setup_table.

    """
    with open(CONFIGPATH, 'r') as f:
        config_json = f.read()
    config = json.loads(config_json)

    for key in config:
```

```
        try:
            LightSensor.mapping.get(key).setup_table = config[key]
        except KeyError:
            raise KeyError('Invalid config file')
```

/components/devices.py

```
import time
from scipy.interpolate import interp1d
from numpy import clip, array
from numpy.polynomial import Polynomial
from ..config import *
try:
    from colorize import clr_ret as clr
except ImportError:
    def clr(text, *args):
        return text


class Timer(object):
    def __init__(self, count):
        self.start = time.time()
        self.count = count

    def __call__(self):
        return time.time()-self.start <= self.count


class Sensor(object):
    """
    Base class for all types of sensors. It implements a
    method to request a reading from a sensor.

    """
    def __init__(self, addr):
        self.addr = addr
        self.instances.append(self)
        self.mapping[addr] = self
        # setup_table length equal to no. of lights
        self.setup_table = [[] for _ in xrange(len(Light.instances))]
        self.current_read = Light.thresh # most recent reading

    def get_reading(self):
        """
        Request a reading from a sensor.
        The process is:
            send the request
            wait for response
            if a response is detected, store it
            else, if there is a timeout while waiting,
```

```
                continue with the previous stored value

        :rtype: int or None

        """
        time.sleep(SEND_DELAY)

        send_data = ''.join([self.addr, GET_READ, '        '])
        radio.write(send_data)

        timer = Timer(NODE_TIMEOUT)
        radio.startListening()
        while not radio.available(pipe, True) and timer():
            time.sleep(1000/1000000.0)
        recv_buffer = []
        radio.read(recv_buffer, radio.getDynamicPayloadSize())
        recv_data = ''.join(chr(i) for i in recv_buffer)
        radio.stopListening()

        dest = recv_data[:2]
        source = recv_data[2:4]
        if (source == self.addr) and (dest == GATEWAY_ADDR):
            self.current_read = int(recv_data[-4:])
            print clr('sensor {}, received {}'
                      .format(self.addr, self.current_read),
                      'cyan')

    def __repr__(self):
        return '{}, addr {}'.format(self.__class__, self.addr)



class Light(object):
    """
    Class for all lights. It implements a method to dim by an increment,
    and another one to dim to a specific DIM_TABLE value (only used in the
    setup stage). It also keeps track of every Light instance, in the
    instances table.

    """
    instances = []
    mapping = {}
    mean_percentage = None
    thresh = THRESH_MAX

    def __init__(self, addr):
        self.addr = addr
        self.instances.append(self)
        self.mapping[addr] = self
        self.state = 'auto'
        self.intens = self.thresh
        self._sum = 0
```

```
    def dim_increment(self, dim):
        """
        Dim by an increment. The increment is provided
        as a percent value, and it is modified to a
        value in dimmer units through multiplication by the
        mean_percentage value

        :type dim: integer
        :rtype: bool

        """
        incr_send = abs(int(self.mean_percentage*dim))
        if incr_send>MAX_INCR:
            incr_send = MAX_INCR
        if dim>0:
            send_data = ''.join([self.addr,
                                 DIM_INCR_UP,
                                 str((incr_send)),
                                 '       '])
        else:
            send_data = ''.join([self.addr,
                                 DIM_INCR_DOWN,
                                 str((incr_send)),
                                 '       '])
        time.sleep(SEND_DELAY)
        radio.write(send_data)

    def dim_to_val(self, val):
        """
        Dim to a value (0-100%)

        :type val: integer
        :rtype: bool
        """
        send_data = ''.join([self.addr,
                             DIM_TO_VAL,
                             DIM_TABLE[val/10],
                             '       '])
        time.sleep(SEND_DELAY)
        radio.write(send_data)

    def dim_real_value(self, val):
        """
        Dim to a value (DIM_MIN-DIM_MAX)

        :type val: integer
        :rtype: bool
        """
        val = clip(val, DIM_MIN, DIM_MAX)
        send_data = ''.join([self.addr,
                             DIM_TO_VAL,
```

```
                            str(val),
                            '       '])
        time.sleep(SEND_DELAY)
        radio.write(send_data)

    def update_sum(self, val):
        self._sum += val
        abs_sum = abs(self._sum)
        if abs_sum>600 and self.state=='auto':
            self.state = 'nauto'
        elif abs_sum<=600 and self.state=='nauto':
            self.state = 'auto'
        self._sum = clip(self._sum, -600, 600)




class LightSensor(Sensor):
    """
    Inherits from the Sensor class. Here, we use the 'instances'
    table to keep track of all instances of this class. This way
    we can request readings simply by iterating through this table,
    and invoking LightSensor.instances[i].get_reading()

    """
    instances = []
    mapping = {}


class DissipationSensor(Sensor):
    """
    Inherits from the Sensor class. Here, we use the 'instances'
    table to keep track of all instances of this class. This way
    we can request readings simply by iterating through this table,
    and invoking DissipationSensor.instances[i].get_reading()

    """
    instances = []
    mapping = {}

    def __init__(self, addr, max_power, power_recv=None):
        Sensor.__init__(self, addr)
        power_expected = array([max_power*i/100
                                for i in xrange(0, 110, 10)])
        if power_recv is None:
            power_recv = power_expected[:]
        self._poly = Polynomial.fit(power_recv, power_expected, 2)
        self.least_squares(power_recv, power_expected)
        self._conv = interp1d(*self._poly.linspace())
        self._min_recv = min(power_recv)
        self._max_recv = max(power_recv)
        self._min_pow = min(power_expected)
```

```
        self._max_pow = max(power_expected)

    def get_reading(self):
        Sensor.get_reading(self)
        print 'SCT reading:', self.current_read
        self.current_read = self.remove_error(self.current_read)
        print 'Converted to:', self.current_read

    def get_raw_reading(self):
        """
            For setup and testing purposes
            NOTE: also returns the current reading

        """
        Sensor.get_reading(self)
        return self.current_read

    def remove_error(self, p_recv):
        p_recv = p_recv/100.0
        p_recv_clipped = clip(p_recv, self._min_recv, self._max_recv)
        return clip(self._conv(p_recv_clipped),
                    self._min_pow,
                    self._max_pow)

    def least_squares(self, p_rec, p_exp):
        plt.figure(figsize=(5*0.75, 5*0.75), dpi=96)
        plt.plot(*self._poly.linspace(), linewidth=1.2)
        plt.plot(p_rec, p_exp, 'ro', alpha=0.8)
        plt.title('$Sensor\,addr:\,{}$'.format(self.addr))
        plt.xlabel('$Measured\,power\,(W)$')
        plt.ylabel('$Expected\,power\,(W)$')
        plt.ylim(-5, 45)
        plt.grid(True)
        plt.tight_layout(True)
        plt.savefig('plot_{}.png'.format(self.addr), dpi=96)
```

/components/servers.py

```
import time
from math import floor, ceil
import os
from ..config import *


def mqtt_publish(topic, data):
    mqttc.publish(''.join([MQTT_BASE, topic]), "%s" % data, 1, retain=True)


def db_insert(sensor_id, sensor_value):
    min_val = floor(sensor_value)
```

```
    max_val = ceil(sensor_value)
    sensor_value = min_val if (sensor_value-min_val<=max_val-sensor_value)
                           else max_val
    sensor_value = int(sensor_value)
    sql_date = (time.strftime("%Y-%m-%d ") + time.strftime("%H:%M:%S"))
    sql = """INSERT INTO power\
             (datetime,sensor_id,sensor_value)\
             VALUES (%s,%s,%s)"""

    try:
        #print "Writing to database..."
        cur.execute(sql, (sql_date, sensor_id, sensor_value))
        db.commit()
        #print "Write complete!"

    except:
        # Rollback in case there is any error
        db.rollback()
        #print "Failed writing to database"
```

/components/functions.py

```
import numpy

def get_diff(v_list):
    """
    Takes an array containing multiple arrays as an
    argument, and constructs a new array, with the
    same number of sub-arrays, each one with an element
    less, that contains the difference between consecutive
    elements of each sub-array of the original argument.
    It is used as a step to create the weights for each
    LightSensor, corresponding to each light. v_list is
    each LightSensor's setup_table

    :type v_list: NxM list
    :rtype: (N-1)xM list

    """
    d_list = [range(10), range(10)]
    for i in xrange(len(v_list)):
        for j in xrange(len(v_list[i])-1):
            diff = v_list[i][j+1] - v_list[i][j]
            d_list[i][j] = diff if (diff >= 0) else 1

    return d_list

def get_mean(d_list):
    """
    Takes an array containing multiple arrays as an
```

```
    argument, and constructs a new array, with the
    same number of sub-arrays, each one containing a
    single value, the mean value of each sub-array.
    It is used to construct the weights for each
    LightSensor

    :type d_list: NxM list
    :rtype: 1xM list

    """
    m_list = []
    for i in xrange(len(d_list)):
        m_list.append(numpy.mean((d_list[i])))
    return m_list

def get_next_dim(m_list, s_list_func, thresh):
    """
    This function takes the arguments:
    m_list: contains the mean value of v_list
    s_list: current sensor values
    thresh: threshold light value
    and returns an array for the value to add
    to the light intensity. For a negative value,
    we subtract.

    :type m_list: 1xM list
    :type s_list_func: 1xM list
    :type thresh: int
    :rtype: 1xM list

    """

    s_list_lin = [(10.0*(thresh - i)) for i in s_list_func]
    try:
        l_add = list(numpy.linalg.solve(m_list, s_list_lin))
        return l_add
    except numpy.linalg.linalg.LinAlgError:
        # This exception is raised in case of Singular Matrix.
        # In this case we treat each sensor as if it only
        # depends on the nearest bulb.
        l_add = [(s_list_lin[i]/m_list[i][i])
                 for i in range(len(s_list_lin))]
        return l_add
```

app_setup.py

```
from lights import *

def test(light, sensor):
    cur_reads = []
```

```
    for i in xrange(0, 110, 10):
        light.dim_to_val(i)
        sensor.get_reading()
        cur_reads.append(sensor.current_read)
    return cur_reads

light_1 = Light('06')
light_2 = Light('07')
ldr_1 = LightSensor('03')
ldr_2 = LightSensor('04')

setup()
```

app_sct_learn.py

```
from lights import *
from collections import defaultdict
import json, time

light_1 = Light('06')
light_2 = Light('07')

sct_1 = DissipationSensor(addr='08', max_power=40)
sct_2 = DissipationSensor(addr='09', max_power=40)

train_data = defaultdict(list)
assoc = {sct_1: light_1, sct_2: light_2}

for sct, light in assoc.items():
    for lum in xrange(0, 110, 10):
        light.dim_to_val(lum)
        time.sleep(4)
        reading = sct.get_raw_reading()
        train_data[sct.addr].append(reading/100.0)

with open('lights/sct_train_data.txt', 'w') as f:
    f.write(json.dumps(train_data))
```

app_run.py

```
from RPi.GPIO import cleanup as GPIOcleanup
from scipy.interpolate import interp1d
from numpy import clip
from watchdog.events import FileSystemEventHandler
from watchdog.observers import Observer
from threading import Thread
import Queue
import re
import sys
```

```
import operator
from time import strftime

try:
    from colorize import clr_ret as clr
except ImportError:
    def clr(text, *args):
        return text

from lights import *


def close_all_lights():
    for light in Light.instances:
        light.dim_to_val(0)

def request_readings(sensor_type):
    for sensor in sensor_type.instances:
        sensor.get_reading()

def adjust_lights():
    states = [light.state for light in Light.instances]
    print 'BEFORE:', states
    if 'auto' not in states and 'nauto' not in states:
        print 'no auto light found'
        return

    thresh = Light.thresh
    cur_reads = [thresh if abs(int(sensor.current_read)-thresh)<=20
                 else int(sensor.current_read)
                 for sensor in LightSensor.instances]

    print cur_reads

    for reading in cur_reads:
        if abs(reading-thresh)>60:
            Light.mean_percentage *= 4
        elif abs(reading-thresh)>30:
            Light.mean_percentage *= 2
            break
    else:
        Light.mean_percentage = abs(m_dim/10.0)

    # If we have one light at auto and one at on or off, then
    # the send value for the auto light is calculated using a
    # weighted average.
    # ONLY IN CASE OF 2 LIGHT INSTANCES!
    send_vals = get_next_dim(m_list, cur_reads, thresh)
    print "DEBUG", send_vals
    for i, light in enumerate(Light.instances):
        light.update_sum(send_vals[i])
        states[i] = light.state
```

```
    print 'AFTER:', states

    if 'auto' not in states:
        print 'no auto light found'
        return

    if 'auto' in states and ('on' in states \
                             or 'off' in states \
                             or 'nauto' in states):
        auto_index = [light.state
                      for light in Light.instances].index('auto')

        mean_index, mean = max(enumerate(zip(*m_list)[auto_index]),
                                          key=operator.itemgetter(1))
        send_vals = [0 if light.state!='auto'
                     else (10/mean)*(thresh-cur_reads[mean_index])
                     for light in Light.instances]


    print 'Algorithm result:', send_vals
    for i, light in enumerate(Light.instances):
        if light.state=='auto' and send_vals[i]!=0:
            light.dim_increment(send_vals[i])

def init_lights():
    """
    Initial dimming to a value that results to the desired
    threshold. If the state from the conf file is 'on' or 'off',
    dim accordingly. If it is 'auto', dim to the closest multiple
    of 10, which is bigger than the calculated value.

    """
    from math import ceil

    v_send = get_next_dim(m_list,
                          [sensor.setup_table[0][0] for sensor in
LightSensor.instances],
                          Light.thresh)
    print v_send
    for i, light in enumerate(Light.instances):
        if light.state=='on':
            light.dim_to_val(light.intens)
        elif light.state == 'off':
            light.dim_to_val(0)
        else:
            intens = 10*ceil(v_send[i]/10.0)
            light.dim_to_val(clip(int(intens), 0, 100))

def _db_insert(sct):
    db_insert(int(sct.addr), sct.current_read)

def _mqtt_publish(sct):
```

```
    mqtt_publish(sct.addr, sct.current_read)

class GUIDataChangeHandler(FileSystemEventHandler):
    def __init__(self, queue):
        FileSystemEventHandler.__init__(self)
        self.queue = queue

    def on_modified(self, event):
        """
        the json file has the following format:
        {'light_xy_state': 'on' or 'off' or 'auto',
         'light_xy_int'  : '0' or '10' .... or '100',
         .
         .
         'thresh'        : '0' to '800'
        }

        """
        global gui_data
        print clr('Received command from WebGUI', 'green')
        with open(GUI_CONF_PATH, 'r') as f:
            gui_new = json.loads(f.read())
        # update light instance and class attributes
        for key, val in gui_new.items():
            if key=='thresh':
                Light.thresh = int(val)
            elif key[9:]=='state':
                Light.mapping.get(key[6:8]).state = val
            elif key[9:]=='int':
                Light.mapping.get(key[6:8]).intens = \
                                int(val) if val is not None else 50
        # queue any dimming tasks, if necessary
        for light in Light.instances:
            state = ''.join(['light_', light.addr, '_state'])
            intens = ''.join(['light_', light.addr, '_int'])

            if gui_new.get(state) == 'auto':
                continue
            if gui_new.get(state) == 'off' and gui_new.get(state) != \
            gui_data.get(state):
                task = [light.dim_to_val, [0]]
                self.queue.put(task)
                continue
            if gui_new.get(state) != gui_data.get(state) or
            gui_new.get(intens) != gui_data.get(intens):
                task = [light.dim_to_val, [light.intens]]
                self.queue.put(task)

        gui_data = gui_new


class CommHandler(Thread):
```

```
    """
    This class handles all communications during the
    regular stage of the system. It runs in a dedicated
    thread, and monitors the global queue. When there is
    a task to be executed, it calls the passed function
    with the arguments provided.

    """
    def __init__(self, queue):
        Thread.__init__(self)
        self.queue = queue

    def run(self):
        while True:
            task = self.queue.get()
            self._execute(task)
            self.queue.task_done()

    def _execute(self, task):
        """
        A task is a list, formatted like this:
        [callable, [args]]

        The [args] list can have any length, and
        it doesn't matter, since we use the star (*)
        to unpack it.

        """
        routine = task[0]
        args = task[1]
        print clr('Executing task:{} with args:{}'
                  .format(routine.__name__, args),
                    'red', False)
        routine(*args)


class PrintWithTime:
    """
    Every print statement gets printed
    to stdout with the current date and
    time appended.
    Usage:
        std_out = sys.stdout
        sys.stdout = PrintWithTime()

    """
    n = True
    def write(self, s):
        if s == '\n':
            std_out.write(s)
            self.n = True
        elif self.n:
```

```
            now = strftime('%H:%M:%S')
            std_out.write('[%s] %s' % (now, s))
            self.n = False
        else:
            std_out.write(s)

# instantiate with an address
light_1 = Light('06')
light_2 = Light('07')
lsensor_1 = LightSensor('03')
lsensor_2 = LightSensor('04')

with open(SCT_TRAIN_DATA_DIR, 'r') as f:
    sct_train_data = json.loads(f.read())

sct_1 = DissipationSensor(addr='08', max_power=40,
    power_recv=sct_train_data.get('08', None)
    )
sct_2 = DissipationSensor(addr='09', max_power=40,
    power_recv=sct_train_data.get('09', None)
    )

#TODO: arg parsing
setup_from_file()
gui_data = read_init_data()
thresh = int(gui_data.get('thresh'))

v_list = [[] for i in LightSensor.instances]
for i, sensor in enumerate(LightSensor.instances):
    v_list[i] = sensor.setup_table

d_list = [[] for i in LightSensor.instances]
for i, sublist in enumerate(v_list):
    d_list[i] = get_diff(sublist)

m_list = [[] for i in LightSensor.instances]
for i, sublist in enumerate(d_list):
    m_list[i] = get_mean(sublist)

v_dim = [int(i) for i in DIM_TABLE]
d_dim = [v_dim[i+1] - v_dim[i] for i in xrange(len(v_dim)-1)]
m_dim = sum(d_dim)/len(d_dim)
Light.mean_percentage = abs(m_dim/10.0)

recv_reads = [thresh, thresh] # storage for sensor readings

init_lights()

std_out = sys.stdout
sys.stdout = PrintWithTime()

queue = Queue.Queue(1)
```

```
handler = CommHandler(queue)
handler.setDaemon(True)

gui_data_handler = GUIDataChangeHandler(queue)
observer = Observer()
observer.schedule(gui_data_handler, GUI_CONF_DIR, recursive=False)

handler.start()
observer.start()

try:
   timer_sct = Timer(60) # get SCT reading every minute
    while True:
        if not timer_sct():
            task = [request_readings, [DissipationSensor]]
            queue.put(task)

            for sct in DissipationSensor.instances:
            task = [_db_insert, [sct]]
            queue.put(task)
            task = [_mqtt_publish, [sct]]
            queue.put(task)

            timer_sct = Timer(60)

        task = [request_readings, [LightSensor]]
        queue.put(task)

        task = [adjust_lights, []]
        queue.put(task)

        #OPERATION_CYCLE = 1 # debugging
        time.sleep(OPERATION_CYCLE)

except KeyboardInterrupt:
    print clr('Wait for all tasks to be completed...', 'purple')
    queue.join()
    prompt = raw_input('Close all lights? (y/[n]) ')
    if prompt == 'y':
        close_all_lights()

    cur.close()
    db.close()
    mqttc.disconnect()
    GPIOcleanup()
    print 'EXITING'
```

WebGUI

Δομή

```
web_interface/
  |-- app.py
  |__ templates/
        |-- base.html
        |-- index.html
  |__ static/
        |__ js/
              |-- ... (JavaScript frameworks-extensions)
              |-- scripts.js
```

app.py

```
from flask import (Flask, render_template,
                   request, flash, redirect,
                   url_for, g)
from flask.ext.bootstrap import Bootstrap
import MySQLdb
import time
import json


SQL_DB = 'thesis'
SQL_TABLE = 'power'

app = Flask(__name__)
app.config['SECRET_KEY'] = 'null'
bootstrap = Bootstrap(app)

@app.before_request
def before_request():
    g.db = MySQLdb.connect(host='localhost', user='root',
                           passwd='raspberry', db=SQL_DB)
    g.cur = g.db.cursor()

@app.teardown_request
def teardown_request(exception):
    cur = getattr(g, 'cur', None)
    if cur is not None:
        cur.close()
    db = getattr(g, 'db', None)
    if db is not None:
        db.close()


@app.route('/', methods=['GET', 'POST'])
def index():
    req = request.args.get('time', None)
    if req:
```

```
        print req
        g.cur.execute("""SELECT * FROM %s WHERE \
                      datetime >= (NOW() - INTERVAL %s)""",
                      (SQL_TABLE, req))
        data = g.cur.fetchall()
        arr_send = [[str(time.mktime(i[0].timetuple())),
                     str(int(i[1])),
                     str(int(i[2]))]
                    for i in data]
        data_send = json.dumps(arr_send)
        return data_send
    else:
        flash('Current threshold: {}'
              .format(int(json.loads(init_gui())['thresh'])))
        return render_template('index.html', number=666)

@app.route('/setvals', methods=['GET', 'POST'])
def set_config():
    recv_conf = {key: val for key, val in request.form.items()}
    print recv_conf
    with open('data/gui.conf', 'r') as f:
        prev_data = json.loads(f.read())
    if prev_data != recv_conf:
        with open('data/gui.conf', 'w') as f:
            f.write(json.dumps(recv_conf))
    return ('', 204)

@app.route('/initvals', methods=['GET', 'POST'])
def init_gui():
    with open('data/gui.conf', 'r') as f:
        ret_conf = json.loads(f.read())
    return json.dumps(ret_conf)

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

templates/base.html

```
{% extends "bootstrap/base.html" %}


{% block navbar %}
<div class="navbar navbar-inverse" role="navigation">
<div class="container-fluid" style="margin-right: 80px; margin-left:80px;">
<div class="navbar-header">
<button type="button" class="navbar-toggle"
data-toggle="collapse" data-target=".navbar-collapse">
<span class="sr-only">Toggle navigation</span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
```

```
<span class="icon-bar"></span>
</button>
<a class="pull-left"><img src="../static/control.png" width=50 height=50
    style="opacity: 0.65; margin-right:5px;"></a>
<a class="navbar-brand" href="/">Thesis WebGUI</a>
</div>
<div class="navbar-collapse collapse">
</div>
</div>
</div>
{% endblock %}

{% block content %}
<div class="container-fluid" style="margin-right: 80px; margin-left:80px;">
    {% for message in get_flashed_messages() %}
    <div class="alert alert-info fade in">
        <button type="button" class="close"
                data-dismiss="alert">&times;</button>
        {{ message }}
    </div>
    {% endfor %}

    {% block page_content %}{% endblock %}
</div>
{% endblock %}
```

templates/index.html

```
{% extends "base.html" %}
{% block title %}Admin panel{% endblock %}
{% block head %}
{{super()}}
{% block favicon %}<link rel="icon" type="image/png"
href="../static/bulb.png">{% endblock %}
{% endblock %}
{% block scripts %}
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">
</script>
<script src="{{ url_for('static',
                filename='js/mqttws31_new.js') }}">
</script>
<script src="{{ url_for('static',
                filename='js/scripts.js') }}">
</script>
<script src="http://code.highcharts.com/highcharts.js"></script>
<script src="http://code.highcharts.com/modules/exporting.js"></script>
{{super()}}
{% endblock %}
```

```
{% block page_content %}
<div id="header">
<h1>Control panel</h1>
</div>
<div style="width:100%; vertical-align:middle;">

<div style="float:left; width:20%; margin-top:5px;">

<fieldset>
    <form onsubmit="return false"
          oninput="level.value = thresh.valueAsNumber">

        Threshold:
          <input id="thresh" type="range" min="0" max="800" value="0">
          <output for="thresh" name="level" id="showselect"></output>
        </form><br>

        Light 6:
        <select type="text" id="light_06_state" name="light_06_state">
            <option value="auto">Auto</option>
            <option value="on">On</option>
            <option value="off">Off</option>
        </select>


        <select type="text" id="light_06_int" name="light_06_int">
            <option value="0">0%</option>
            <option value="10">10%</option>
            <option value="20">20%</option>
            <option value="30">30%</option>
            <option value="40">40%</option>
            <option value="50">50%</option>
            <option value="60">60%</option>
            <option value="70">70%</option>
            <option value="80">80%</option>
            <option value="90">90%</option>
            <option value="100">100%</option>
        </select>
        <br>

        Light 7:
        <select type="text" id="light_07_state" name="light_07_state">
            <option value="auto">Auto</option>
            <option value="on">On</option>
            <option value="off">Off</option>
        </select>

        <select type="text" id="light_07_int" name="light_07_int">
            <option value="0">0%</option>
            <option value="10">10%</option>
            <option value="20">20%</option>
            <option value="30">30%</option>
```

```
            <option value="40">40%</option>
            <option value="50">50%</option>
            <option value="60">60%</option>
            <option value="70">70%</option>
            <option value="80">80%</option>
            <option value="90">90%</option>
            <option value="100">100%</option>
        </select>
        <br><br>

        <button type="button" class="btn btn-primary btn-md"
                id="txtsub" value="Send">
        <span class="glyphicon glyphicon-ok"></span> Send</button>
        </select>

<br><br>

    <label>Current threshold</label>
    <span class="label label-info" id="showthresh"></span>

    </label>
</fieldset>
<br>

<fieldset>
    Historical data:
    <select id="timeinterval" name="time">
    <option value="">Select interval:</option>
    <option value="1 hour" name="last hour">1 hour</option>
    <option value="5 hour" name="last 5 hours">5 hours</option>
    <option value="1 day" name="last day">1 day</option>
    <option value="5 day" name="last 5 days">5 days</option>
    <option value="2 week" name="last 2 weeks">2 weeks</option>
    <option value="1 month" name="last month">1 month</option>
    <option value="1 year" name="last year">1 year</option>
    </select>
    <br>
    </select><br>

    <input type="button" class="btn btn-default"
           id="realtime" value="Real Time Data">
</fieldset>

</div>

<div style="float:right; width:75%; margin-right:40px; margin-top:20px;">
<div id="chart_hist" style="height: 400px; margin: 0 auto"></div>
<div id="chart_realtime" style="height: 400px; margin: 0 auto"></div>
</div>

</div>
{% endblock %}
```

static/js/scripts.js

```
$(document).ready(function() {

    $.getJSON("initvals", function(init) {
        $.each(init, function(key, value){
            if (key !== 'thresh') {
                $("#"+key).val(value);
            }
        });
        th = init["thresh"]
        $("#thresh").prop("value", th);
        $("#showthresh").text(th);
        $("#showselect").val(th);
    });

    $('#light_06_state').change(function() {
        if (($(this).val() == "on")) {
            $("#light_06_int").prop("disabled", false);
        } else {
            $("#light_06_int").val("50");
            $("#light_06_int").prop("disabled", true);
        }
    });

    $('#light_07_state').change(function() {
        if (($(this).val() == "on")) {
            $("#light_07_int").prop("disabled", false);
        } else {
            $("#light_07_int").val("50");
            $("#light_07_int").prop("disabled", true);
        }
    });

    $('#txtsub').click(function() {

        $.get("/test.py?thresh="+option, function(data) {
            alert( "Data Loaded: ");*/
        thresh = $("#thresh").val();
        //alert(thresh);
        if ($.isNumeric(thresh)) {
            $('#showthresh').text(thresh);
        } else {
            thresh = "";
        }


        light_06_state = $("#light_06_state").val();

        light_06_int = $("#light_06_int").val();
```

```
        light_07_state = $("#light_07_state").val();

        light_07_int = $("#light_07_int").val();

        $.post("/setvals",
            {thresh: thresh,
            light_06_state: light_06_state,
            light_06_int: light_06_int,
            light_07_state: light_07_state,
            light_07_int: light_07_int
        });
    });

    $('#realtime').click(function() {

        var MQTTbroker = '192.168.1.56';
        var MQTTport = 9001;
        var MQTTsubTopic1 = 'thesis/power/08';
        var MQTTsubTopic2 = 'thesis/power/09';

        var chart_h;
        var chart_r;
        var dataTopics = new Array();

        var client = new Messaging.Client(MQTTbroker, MQTTport,
                    "myclientid_" + parseInt(Math.random() * 100, 10));
        client.onMessageArrived = onMessageArrived;
        client.onConnectionLost = onConnectionLost;
        var options = {
            timeout: 3,
            onSuccess: function () {
                console.log("mqtt connected");
                // Connection succeeded; subscribe to our topics
                client.subscribe(MQTTsubTopic1, {qos: 1});
                client.subscribe(MQTTsubTopic2, {qos: 1});
                },
            onFailure: function (message) {
                console.log("Connection failed, ERROR: " +
message.errorMessage);
            }
        };

    //can be used to reconnect on connection lost
        function onConnectionLost(responseObject) {
            console.log("connection lost: " + responseObject.errorMessage);
        };

    //what is done when a message arrives from the broker
        function onMessageArrived(message) {
            console.log(message.destinationName, '',message.payloadString);
```

```
        //check if it is a new topic, if not add it to the array
        if (dataTopics.indexOf(message.destinationName) < 0){

            dataTopics.push(message.destinationName);
            var y = dataTopics.indexOf(message.destinationName);
            if (message.destinationName === 'thesis/power/09'){
                var color = '#EB6881';
            }
            else {
                var color;
            }
            var newseries = {
                    id: y,
                    name: message.destinationName,
                    data: [],
                    color: color
                    };

            chart_r.addSeries(newseries); //add the series

            };

        var y = dataTopics.indexOf(message.destinationName);
        var myEpoch = new Date().getTime();
        var timezone = new Date().getTimezoneOffset();
        var thenum = message.payloadString.replace( /^\D+/g, '');
        var plotMqtt = [myEpoch-timezone*60000, Number(thenum)];
        if (isNumber(thenum)) {
            plot(plotMqtt, y);
        };
    };

//check if a real number
    function isNumber(n) {
      return !isNaN(parseFloat(n)) && isFinite(n);
    };
//function that is called once the document has loaded
    function init() {

        Highcharts.setOptions({
            global: {
            }
        });
        client.connect(options);
    };

    function plot(point, chartno) {
        var series = chart_r.series[0],
        shift = series.data.length > 20;
        chart_r.series[chartno].addPoint(point, true, shift);
   };
            init();
```

```
            chart_r = new Highcharts.Chart({
                chart: {
                    renderTo: 'chart_hist',
                    defaultSeriesType: 'spline'
                },
                title: {
                    text: 'Power consumption (live data)'
                },
                subtitle: {
                          text: 'myMQTTbroker: '
                                + MQTTbroker + ' | port: '
                                + MQTTport
                             },
                xAxis: {
                    type: 'datetime',
                    tickPixelInterval: 150,
                    maxZoom: 20 * 1000
                },
                yAxis: {
                    minPadding: 0.2,
                    maxPadding: 0.2,
                    title: {
                        text: 'Value',
                        margin: 80
                    }
                },
                series: []
            });
        });

$('#timeinterval').change(function() {
    var option = $(this).find('option:selected').val();
    var name = $(this).find('option:selected').attr("name");
   var datetime = [];
    var sensor_one = [];
    var sensor_two = [];
    var series;
    var sensornum;
    var sensorval;
    var timestamp;
    var assoc_sens = {};


    $.getJSON("?time="+option, function(data) {
        timezone = new Date().getTimezoneOffset();
        //console.log(timezone);
        $.each(data, function( index, value ) {
            timestamp = parseFloat(value[0])*1000 - timezone*60*1000;
            sensornum = value[1];
            sensorval = parseFloat(value[2]);
            if (sensornum == 8) {sensor_one.push({
                                        x: timestamp,
```

```
                                    y: sensorval
                                });
                            }
        if (sensornum == 9) {sensor_two.push({
                                    x: timestamp,
                                    y: sensorval
                                });
                            }

    });

    chart_h = new Highcharts.Chart({
    chart : {
    renderTo: 'chart_hist',
    type : 'spline'
    },
    title : {
    text : 'Power consumption (' + name + ')'
    },
    subtitle : {
    text : 'thesis'
    },
    xAxis : {
    type: 'datetime',
    dateTimeLabelFormats: { // don't display the dummy year
                month: '%e. %b',
                year: '%b'
    },
    title : {
    text : 'Date and time'
    },
    },
    yAxis : {
    title : {
    text : 'Sensor Value'
    },
    labels : {
    formatter : function() {
    return this.value
    }
    }
    },
    tooltip : {
    crosshairs : true,
    shared : true,
    valueSuffix : ''
    },
    plotOptions : {
    spline : {
    turboThreshold: 0,
    marker : {
    radius : 4,
```

```
            lineColor : '#666666',
            lineWidth : 1,
            }
            }
            },
            series : [{

            name : 'Sensor 08',
            data : sensor_one
            },{

            name : 'Sensor 09',
            data : sensor_two,
            color: '#EB6881'

            }]
            });
        });
    });

$("#realtime").click()
});
```

Άλλα

dimmer_vis.py

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider

def pulse_zero(x, k, fract):
    """
    This function returns True if x is equal
    to k*pi + fract. It provides the boundaries
    inside which the waveform is not equal to 0.
    The parameter k is incremented with each
    period of the waveform.

    """
    cnd1 = x >= k
    cnd2 = x < (fract + k)
    cnd3 = x >= (np.pi + k)
    cnd4 = x < (np.pi + fract + k)
    return ((cnd1 and cnd2) or (cnd3 and cnd4))

def make_pulse(fract):
    """
    Function to create the pulse and the relevant
```

```
    chopped sin, according to the slider value.
    TODO: document this mess

    """
    repeats = 2 # more repeats create ugly display
    xf = np.linspace(0, repeats, 250*repeats)
    lf = len(xf)/repeats
    zf = [1 for i in xrange(len(xf))]
    yt = 2*np.pi*xf
    for k in xrange(0, 2*repeats, 2):
        for i in xrange(lf*k/2, lf + lf*k/2):
            if pulse_zero(yt[i], k*np.pi, fract):
                zf[i] = 0
            else:
                zf[i] = 1
    pulse = [0 for i in xrange(len(zf))]
    trig = 0
    for i in range(len(zf)):
        if trig<10 and zf[i]==1 :
            pulse[i] = 1
            trig += 1
        if zf[i] == 0 :
            trig = 0

    y = np.sin(2*np.pi*x)
    for i in xrange(len(y)):
        y[i] *= zf[i]

    return y, pulse

def update(val):
    """
    Triggered when the slider changes value.
    It then regenerates the plot, creating
    the visualization of the value change.

    """
    new_dim = np.pi - sl.val
    new1, new2 = make_pulse(new_dim)
    p1.set_ydata(new1)
    p2.set_ydata(new2)
    fig.canvas.draw_idle()

def automate():
    """
    Placeholder to implement a method that
    creates an automated visualization.

    """
    print 'auto'

fract = np.pi/2
```

```
repeats = 2
x = np.linspace(0, repeats, 250*repeats)
fig, ax = plt.subplots()
plt.subplots_adjust(left=0.1, bottom=0.25)
y1, y2 = make_pulse(fract)

p1, = plt.plot(x, y1, linewidth=1.5)
#plt.plot(x, z, 'g')
p2, = plt.plot(x, y2, 'r', alpha=0.73)
p3, = plt.plot(x, np.sin(2*np.pi*x), 'b--', alpha=0.37)
plt.axis([0, repeats, -2, 2])
plt.gca().axes.get_xaxis().set_ticklabels([])
plt.gca().axes.get_yaxis().set_ticklabels([])
plt.grid(True)
axdim = plt.axes([0.2, 0.1, 0.65, 0.03], axisbg='lightgoldenrodyellow')
#plt.ylim(-2, 2)
fig.canvas.set_window_title('Dimmer visualization')
sl = Slider(axdim, 'dim value', 0.05, np.pi-0.05, valinit=fract)
sl.on_changed(update)
plt.show()
automate()
```

poly_fit.py

```
from numpy.polynomial import Polynomial
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
import sys

try:
    num = float(sys.argv[1])
    num = np.clip(num, 2.6, 42)
except:
    num = 42

x = np.array([2.6, 16.4, 20.9, 24.6, 26.7,
              29.5, 32, 33.85, 35.5, 36.5, 42])
y = np.array([0, 4, 8, 12, 16,
              20, 24, 28, 32, 36, 40])

p = Polynomial.fit(x, y, 2)
plt.plot(*p.linspace(), linewidth=1.5)
plt.plot(x, y, 'ro')
conv = interp1d(*p.linspace())
print np.clip(conv(num), 0, 40)
plt.title('$Least\,squares\,fitting\,of\,power$\n$measurement,\,based\,on\,
error$')
plt.xlabel('$Measured\,power\,(W)$')
plt.ylabel('$Expected\,power\,(W)$')
```

```
plt.show()
```

power_estimate.py

```
"""
This module is used to determine the values that the dimmer
must be set to, in order to achieve various levels of power
dissipation (in this case, the levels 0,10...90,100%).

Because P(t)=V(t)*I(t), it is true that we can determine the
power dissipation in a specific amount of time by integrating
this equation. Here, we integrate for half a period of the
mains supply, because that is the period of the power (the
period of sin squared).

Then, for each value we want to determine, we brute-force the
solution by comparing the desired result to the result of each
iteration (the precision is user-determined). The mean value
is mapped through the MIN and MAX values of the dimmer, thus
providing us with a result - of course with an error.

"""

from numpy import sin, linspace, pi
from scipy.integrate import quad
from scipy.interpolate import interp1d # C's 'map' function
import json


f = 50          # mains frequency
V = 325         # Vrms*sqrt(2)
I = 0.3         # 100/V because we work with percentages
HALF_P = 0.5/f  # half period of the wave
DIM_MAX = 600   # max value dictated by the dimmer..
DIM_MIN = 50    # .. and min value
PREC = 2        # precision for calculations
# Absolute tolerance for comparisons. If one value
# yields no results, we move on to a greater one,
# thus to a larger error.
tolerance = [i/1000.0 for i in xrange(10)]


def isclose(a, b, rel_tol=1e-09, abs_tol=0.0):
    """
    Implementation of the `math.isclose` method
    which is provided in python 3.5, as described
    in the documentation.

    """
    return abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)
```

```
def integrand(x):
    return V*I*sin(2*pi*f*x)**2

def construct_table():
    """:rtype: list"""
    solutions = []
    MAX, _ = quad(integrand, 0, HALF_P)
    for i in [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]:
        power = MAX*i
        results = []

        for tol in tolerance:
            if len(results) > 0:
                break
            for start in linspace(0, HALF_P, 1000):
                res, _ = quad(integrand, start, HALF_P)
                #print res
                if isclose(res, power, abs_tol=tol):
                    results.append(start)

        mean = sum(results)/len(results)
        solutions.append(mean)

    conv = interp1d([min(solutions), max(solutions)], [DIM_MIN, DIM_MAX])
    mapped_sol = conv(solutions)
    return [int(i) for i in mapped_sol] # = DIM_TABLE


if __name__ == '__main__':
    DIM_TABLE = construct_table()
    print DIM_TABLE

    with open('dim_table.json', 'w') as f:
        f.write(json.dumps(DIM_TABLE))
```