ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΑΡΑΓΩΓΗΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

F1TENTH ΚΑΤΑΣΚΕΥΗ ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Ντομένικο Μάλι

ΑΜ: 2021/009

ΕΠΙΒΛΕΠΩΝ

ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

ΔΗΜΗΤΡΙΟΣ ΜΠΕΧΤΣΗΣ

ΕΠΙΚΟΥΡΙΚΟ ΕΡΓΟ

Αυτόνομα οχήματα με χρήση ROS

ΘΕΣΣΑΛΟΝΙΚΗ 2024

Ευχαριστίες

Η παρούσα διπλωματική εργασία πραγματοποιήθηκε στο Διεθνές Πανεπιστήμιο της Ελλάδος, στο τμήμα Μηχανικών Παραγωγής και Διοίκησης για το ετος 2023-2024. Σε αυτό το σημείο θα ήθελα να ευχαριστήσω τους γονείς μου και τους φίλους μου για την αμέριστη συμπαράσταση και υπομονή τους κατά την ολοκλήρωση του μεταπτυχιακού μου διπλώματος και όχι μόνο. Τον Αναπληρωτή Καθηγητή Μπεχτσή Δημήτριο, για την ουσιαστική καθοδήγηση του ως επιβλέπον της διπλωματικής εργασίας και τον διδακτορικό φοιτητή Καραμήτσος Γιώργος για την πολύτιμη βοήθεια που μου παρείχε κατά την εκπόνηση της εργασίας . Επίσης θα ήθελα να ευχαριστήσω τους καθηγητές του τμήματος και τους συμφοιτητές για την άψογη συνεργασία μας.

Περίληψη

Η αποδοτική διαχείριση μεγάλου όγκου δεδομένων είναι ένα πολύ σημαντικό κεφάλαιο καθώς ο όγκος δεδομένων που παράγεται καθημερινά είναι τεράστιος και προσπελάσιμος από πάρα πολλές πηγές. Συνεπώς για την διαχείριση του πρέπει να προβλεφθεί τόσο η ανάλυση των δεδομένων (καθαρισμός δεδομένων, ταξινόμηση και ομαδοποίηση δεδομένων) όσο και η αποθήκευση τους για την βέλτιστη χρήση τους. Σκοπός της πτυχιακής εργασίας είναι η ολοκληρωμένη προσέγγιση ενός προβλήματος διαχείρισης μεγάλου όγκου δεδομένων, από την λήψη των δεδομένων, την επεξεργασία τους, την δημιουργία αλγορίθμου μηχανικής μάθησης, την αποθήκευση τους σε μια βάση δεδομένων και την δημιουργία ενός user interface για την οπτικοποίηση τους. Για την επίτευξη του σκοπού της εργασίας χρησιμοποιήθηκε το λογισμικό Linux 20.04 και η γλώσσα προγραμματισμού python και C++ μέσω του Qt creator, Το Qt Creator είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης πολλαπλών πλατφορμών (IDE) προσαρμοσμένο για μέγιστη παραγωγικότητα προγραμματιστών. Το Qt Creator υποστηρίζει τη χρήση βοηθών κωδικοποίησης όπως το GitHub Copilot κατά τη διάρκεια του προγραμματισμού. Βοηθά τους προγραμματιστές να δημιουργήσουν λογισμικό για επιτραπέζιους υπολογιστές, κινητές συσκευές και ενσωματωμένες πλατφόρμες , επιπλέον χρησιμοποιήθηκε το ROS , οπού είναι ένα πρόγραμμα η αλλιώς Software που έχει μια μεγάλη γκάμα από βιβλιοθήκες και μας βοηθάει να χτίσουμε το ρομπότ μας στην συγκεκριμένη περίπτωση το αμάξι μας έτσι όπως ακριβώς θέλουμε. Βέβαια σε αυτό το σημείο να επισημαίνουμε ότι δεν υπάρχει μόνο το **f1tenth** απλά εμείς θα ασχοληθούμε με το συγκεκριμένο . Εδώ να επισημαίνουμε ακόμη ότι ο σκοπός αυτού το μεταπτυχιακού είναι να χτίσει 4 βασικούς πυλώνες:

1. ***Το χτίσιμο***: Σχεδιάζουμε και διατηρούμε το f1tenth αυτόματο όχημα , σε ένα δυνατό και open-source πλατφόρμα για ερευνητικό και εκπαιδευτικό σκοπό
2. ***Την μάθηση***: Μαθαίνουμε πως να προγραμματίσουμε στο ROS και βλέπουμε πως είναι η αλληλεπίδραση του ανθρώπου μηχανής στο συγκεκριμένο περιβάλλον
3. ***Αγώνας***: Το f1tenth αναπτύσσεται δραστικά και ανά τακτικά διαστήματα δημιουργεί αγώνες ανά τον κόσμο και έτσι μας κεντράρει ακόμα περισσότερο το ενδιαφέρον
4. ***Την Ερευνά***: Η πλατφόρμα του f1tenth είναι μια δυνατή πλατφόρμα για χρήση σε ερευνητικό τομέα, για μαθηση και για ρομποτική και πολλά άλλα.

Επιπλέον καθώς το κεφάλαιο της ενέργειας είναι ένα πολύ σημαντικό ζήτημα αλλά και μια τεράστια πρόκληση την οποία αντιμετωπίζει η ανθρωπότητα ,τα δεδομένα που χρησιμοποιήθηκαν για την πτυχιακή εργασία αφορούσαν την παραγωγή και διανομή ενέργειας.

Αφηρημένη

Η αποτελεσματική διαχείριση μεγάλου όγκου δεδομένων είναι ένα πολύ σημαντικό κεφάλαιο καθώς ο όγκος των δεδομένων που παράγονται σε καθημερινή βάση είναι τεράστιος και προσβάσιμος από πολλές πηγές. Επομένως για τη διαχείρισή του πρέπει να προβλεφθεί τόσο η ανάλυση των δεδομένων (καθαρισμός, ταξινόμηση, ομαδοποίηση) όσο και η αποθήκευσή τους για προαιρετική χρήση τους. Σκοπός της διπλωματικής εργασίας είναι η ολοκληρωμένη προσέγγιση ενός προβλήματος μεγάλου όγκου δεδομένων από τη λήψη των δεδομένων, την επεξεργασία τους, τη δημιουργία αλγορίθμου μηχανικής μάθησης, την αποθήκευσή τους σε βάση δεδομένων και τη δημιουργία διεπαφής χρήστη για οπτικοποίηση. Για την επίτευξη του σκοπού της εργασίας χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python μέσω QT creator, η βάση δεδομένων ROS για την αποθήκευση όλων αυτών των δεδομένων και η γλώσσα προγραμματισμού C# μέσω του περιβάλλοντος του visual studio για τη δημιουργία της διεπαφής χρήστη για την οπτικοποίηση δεδομένων. Επιπλέον, καθώς το κεφάλαιο της ενέργειας είναι ένα πολύ σημαντικό ζήτημα αλλά και μια τεράστια πρόκληση που αντιμετωπίζει η ανθρωπότητα, τα δεδομένα που χρησιμοποιήθηκαν για τη διατριβή αφορούσαν την παραγωγή και διανομή ενέργειας.

Περιεχόμενα

Εισαγωγή

Ο όγκος των δεδομένων που παράγεται καθημερινά είναι τεράστιος και επιπλέον αυξάνεται συνεχώς, αυτό έχει σαν αποτέλεσμα την όλο και δυσκολότερη απόκτηση, αποτελεσματική επεξεργασία και αποθήκευση πληροφοριών. Σύμφωνα με μια έκθεση της International Data Corporation που πραγματοποιήθηκε το 2011 αποκαλύφθηκε πως σε δύο μέρες ελέγχου είχαν δημιουργηθεί 1,82ZB δεδομένα, αριθμός ο οποίος διπλασιάζεται κάθε το πολύ δύο χρόνια. Κάπως έτσι δημιουργήθηκε η ανάγκη για την ανθρωπότητα ώστε να αντιμετωπίσει αυτό το φαινόμενο, οπότε ανοίξανε νέα πεδία για έρευνα και μελέτη αλλά και νέες θέσεις εργασίας. Σε αυτό το σημείο και η έννοια του Big Data απόκτησε επιστημονική οντότητα καθώς πλέον ορίζεται για την περιγραφή ενός τεράστιου συνόλου δεδομένων. Με τον όρο περιγραφή δεδομένων νοείται η κάλυψη τριών βασικών απαιτήσεων

- ⇨ Ταχύτητα αφορά το πόσο εύκολα και γρήγορα δίνεται πρόσβαση στα δεδομένα (σχεδόν σε πραγματικό χρόνο)
- ⇨ Όγκος είναι το τεράστιο σύνολο δεδομένων για οποιοδήποτε αντικείμενο
- ⇨ Ποικιλία τα δεδομένα μπορούν αν έχουν δομημένη ή αδόμητη φύση

Όπως διαπιστώνεται, ο όρος big data δεν αναφέρεται μόνο στα δεδομένα αυτούσια αλλά και στο σύνολο των τεχνολογιών που αλληλεπιδρούν μαζί τους. Ουσιαστικά μέσω όλων αυτών των λειτουργιών δημιουργείται μια μεγάλη συσχέτιση δεδομένων η οποία προκαλεί την δημιουργία ενός ευρύτερου δικτύου πληροφοριών. Όλη αυτή η διαδικασία έχει βοηθήσει στην εξέλιξη της ανθρωπότητας αφού βοηθάει σημαντικά στην βέλτιστη μάθηση απαντώντας σε ερωτήσεις και απορίες οι οποίες δεν ήταν εύκολα προσβάσιμες.

Σκοπός της πτυχιακής εργασίας είναι η δημιουργία μιας ολοκληρωμένης προσέγγισης απόκτησης, ανάλυσης ,οπτικοποίησης και αποθήκευσης ενός μεγάλου όγκου δεδομένων. Για την επίτευξη του σκοπού θα χρησιμοποιηθούν μια πληθώρα από εργαλεία τα οποία θα εκτελέσουν όλες τις απαραίτητες λειτουργίες για να επιφέρουν τα βέλτιστα αποτελέσματα. Τα σημαντικότερα από τα εργαλεία που θα χρησιμοποιηθούν είναι οι γλώσσες προγραμματισμού Python η οποία θα είναι υπεύθυνη για την απόκτηση και ανάλυση των δεδομένων, η γλώσσα C# και η έκδοση .net core 3.1 μέσω της οποίας θα γίνει η περαιτέρω ανάλυση αλλά και οπτικοποίηση των δεδομένων και η βάση δεδομένων MySQL στην οποία θα γίνει η αποθήκευση των δεδομένων.

Ένα ακόμη πολύ σημαντικό κεφάλαιο για την ανθρωπότητα είναι η ενέργεια. Η ηλεκτρική ενέργεια είναι ίσως η σπουδαιότερη ανθρώπινη ανάγκη καθώς είναι απαραίτητη για εργασιακούς και όχι μόνο σκοπούς. Η ανθρωπότητα έχει στηριχθεί πλήρως στην ενέργεια και κάθε μορφή μείωσης ή απουσίας της δημιουργεί τεράστια προβλήματα. Επιπλέον σύμφωνα με την διακυβερνητική έκθεση του 2018 ο κόσμος

θα αντιμετωπίσει πολύ σοβαρά ζητήματα εάν δεν μειωθεί ριζικά η ατμοσφαιρική ρύπανση η οποία προκαλεί το γνωστό σε όλους φαινόμενο του θερμοκηπίου. Οπότε

ένα μεγάλο βήμα για την αντιμετώπιση αυτού του φαινομένου θα ήταν η αλλαγή των συστημάτων παραγωγής ηλεκτρικής ενέργειας.

Πάνω σε αυτήν την ανάγκη και με δεδομένη την εξέλιξη της επιστήμης της πληροφορικής αποφασίστηκε ότι το θέμα του συνόλου δεδομένων που θα καλυφθεί στην πτυχιακή εργασία θα αφορά τον ενεργειακό τομέα και η ανάλυση των δεδομένων θα αφορά τον καθαρισμό τους από κακές τιμές, την επεξεργασία τους (ενδεχόμενη ομαδοποίηση, αλλαγή τύπου μεταβλητών) και τέλος την δημιουργία αλγορίθμου ο οποίος θα εκπαιδευτεί ώστε να προβλέπει τιμές ενέργειας.

1 Κεφάλαιο : Βιβλιοθήκες/ Προγράμματα που χρησιμοποιήθηκαν

1.1 ROS

Το Robot Operating System (ROS ή ros) είναι μια σουίτα ρομποτικού ενδιάμεσου λογισμικού ανοιχτού κώδικα. Αν και το ROS δεν είναι λειτουργικό σύστημα (OS), αλλά ένα σύνολο πλαισίων λογισμικού για την ανάπτυξη λογισμικού ρομπότ, παρέχει υπηρεσίες σχεδιασμένες για ένα ετερογενές σύμπλεγμα υπολογιστών, όπως αφαίρεση υλικού, έλεγχος συσκευών χαμηλού επιπέδου, υλοποίηση λειτουργιών που χρησιμοποιούνται συνήθως, μετάδοση μηνυμάτων μεταξύ διαδικασιών και διαχείρισης πακέτων. Εκτελούμενα σύνολα διεργασιών που βασίζονται σε ROS αντιπροσωπεύονται σε μια αρχιτεκτονική γραφήματος όπου η επεξεργασία πραγματοποιείται σε κόμβους που μπορούν να λαμβάνουν, να δημοσιεύουν και να πολυπλέκουν δεδομένα αισθητήρα, έλεγχο, κατάσταση, σχεδιασμό, ενεργοποιητή και άλλα μηνύματα. Παρά τη σημασία της αντιδραστικότητας και της χαμηλής καθυστέρησης στον έλεγχο ρομπότ, το ROS δεν είναι ένα λειτουργικό σύστημα σε πραγματικό χρόνο (RTOS). Ωστόσο, είναι δυνατή η ενσωμάτωση ROS με υπολογιστικό κώδικα σε πραγματικό χρόνο. Η έλλειψη υποστήριξης για συστήματα σε πραγματικό χρόνο έχει αντιμετωπιστεί κατά τη δημιουργία του ROS 2 μια σημαντική αναθεώρηση του ROS API που θα αξιοποιήσει τις σύγχρονες βιβλιοθήκες και τεχνολογίες για βασικές λειτουργίες ROS και θα προσθέσει υποστήριξη κώδικα σε πραγματικό χρόνο και ενσωματωμένο υλικό συστήματος.

Το λογισμικό στο οικοσύστημα ROS μπορεί να χωριστεί σε τρεις ομάδες:

- εργαλεία ανεξάρτητα από γλώσσα και πλατφόρμα που χρησιμοποιούνται για την κατασκευή και τη διανομή λογισμικού που βασίζεται σε ROS
- Εφαρμογές βιβλιοθήκης πελατών ROS όπως roscpp, rospy, και roslisp
- πακέτα που περιέχουν κώδικα που σχετίζεται με την εφαρμογή που χρησιμοποιεί μία ή περισσότερες βιβλιοθήκες πελατών ROS

Τόσο τα ανεξάρτητα από τη γλώσσα εργαλεία όσο και οι κύριες βιβλιοθήκες πελατών (C++, Python και Lisp) κυκλοφορούν υπό τους όρους της άδειας BSD και ως εκ τούτου είναι λογισμικό ανοιχτού κώδικα και δωρεάν τόσο για εμπορική όσο και για ερευνητική χρήση. Η πλειοψηφία των άλλων πακέτων αδειοδοτείται με μια ποικιλία αδειών ανοιχτού κώδικα. Αυτά τα άλλα πακέτα υλοποιούν λειτουργίες και εφαρμογές που χρησιμοποιούνται συνήθως, όπως

προγράμματα οδήγησης υλικού, μοντέλα ρομπότ, τύπους δεδομένων, σχεδιασμός, αντίληψη, ταυτόχρονη τοπική προσαρμογή και χαρτογράφηση (SLAM), εργαλεία προσομοίωσης και άλλους αλγόριθμους.

Οι κύριες βιβλιοθήκες πελατών ROS είναι προσανατολισμένες προς ένα σύστημα παρόμοιο με το Unix, κυρίως λόγω της εξάρτησής τους από μεγάλα σύνολα εξαρτήσεων λογισμικού ανοιχτού κώδικα. Για αυτές τις βιβλιοθήκες πελατών, το Ubuntu Linux αναφέρεται ως "Υποστηριζόμενο", ενώ άλλες παραλλαγές όπως το Fedora Linux, το macOS και τα Microsoft Windows χαρακτηρίζονται ως "πειραματικές" και υποστηρίζονται από την κοινότητα. Η εγγενής βιβλιοθήκη προγράμματος-πελάτη Java ROS, rosjava, ωστόσο, δεν μοιράζεται αυτούς τους περιορισμούς και έχει ενεργοποιήσει το λογισμικό που βασίζεται σε ROS να γραφτεί για το λειτουργικό σύστημα Android. Η rosjava επέτρεψε επίσης την ενσωμάτωση του ROS σε μια επίσημα υποστηριζόμενη εργαλειοθήκη MATLAB που μπορεί να χρησιμοποιηθεί σε Linux, macOS και Microsoft Windows. Μια βιβλιοθήκη πελάτη JavaScript, roslibjs έχει επίσης αναπτυχθεί, η οποία επιτρέπει την ενσωμάτωση λογισμικού σε ένα σύστημα ROS μέσω οποιουδήποτε προγράμματος περιήγησης ιστού συμβατό με πρότυπα. Το ROS σχεδιάστηκε για να είναι ανοιχτού κώδικα, με σκοπό οι χρήστες να μπορούν να επιλέγουν τη διαμόρφωση των εργαλείων και των βιβλιοθηκών που αλληλεπιδρούν με τον πυρήνα του ROS, έτσι ώστε οι χρήστες να μπορούν να αλλάζουν τις στοίβες λογισμικού τους ώστε να ταιριάζουν με το ρομπότ και την περιοχή εφαρμογών τους. Ως εκ τούτου, υπάρχουν πολύ λίγα στοιχεία που είναι ο πυρήνας των ROS, πέρα από τη γενική δομή εντός της οποίας πρέπει να υπάρχουν και να επικοινωνούν τα προγράμματα. Κατά μία έννοια, το ROS είναι η υποκείμενη υδραυλική εγκατάσταση πίσω από τους κόμβους και τη μετάδοση μηνυμάτων. Ωστόσο, στην πραγματικότητα, το ROS δεν είναι μόνο αυτό το υδραυλικό, αλλά ένα πλούσιο και ώριμο σύνολο εργαλείων, ένα ευρύ φάσμα ρομποτικών ικανοτήτων που παρέχονται από πακέτα και ένα μεγαλύτερο οικοσύστημα προσθηκών σε ROS

1.2 Qt Creator

Το Qt Creator είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) C++, JavaScript, Python και QML που απλοποιεί την ανάπτυξη εφαρμογών GUI. Αποτελεί μέρος του SDK για το πλαίσιο ανάπτυξης εφαρμογών Qt GUI και χρησιμοποιεί το Qt API, το οποίο ενσωματώνει τις κλήσεις λειτουργίας του κεντρικού λειτουργικού συστήματος GUI.[3] Περιλαμβάνει ένα οπτικό πρόγραμμα εντοπισμού σφαλμάτων και μια ενσωματωμένη διάταξη WYSIWYG GUI και σχεδιαστή φορμών. Το πρόγραμμα επεξεργασίας διαθέτει χαρακτηριστικά όπως επισήμανση σύνταξης και αυτόματη συμπλήρωση. Το Qt Creator χρησιμοποιεί τον μεταγλωττιστή C++ από τη συλλογή GNU Compiler

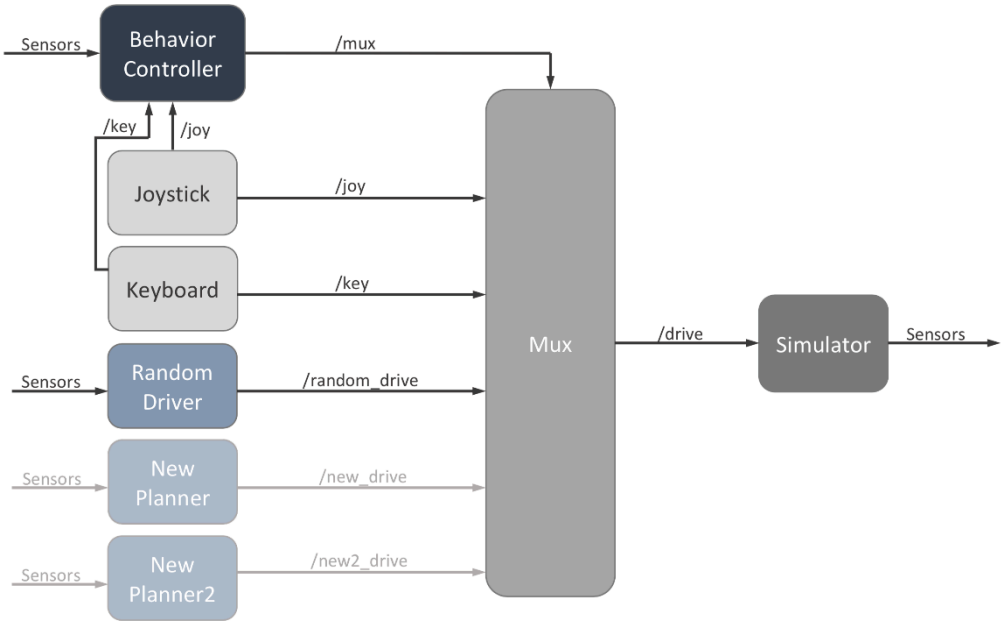
στο Linux. Στα Windows μπορεί να χρησιμοποιήσει MinGW ή MSVC με την προεπιλεγμένη εγκατάσταση και μπορεί επίσης να χρησιμοποιήσει το Microsoft Console Debugger όταν μεταγλωττίζεται από τον πηγαίο κώδικα. Το Clang υποστηρίζεται επίσης. Το Qt Creator περιλαμβάνει έναν διαχειριστή έργου που μπορεί να χρησιμοποιήσει μια ποικιλία μορφών έργου όπως .pro, CMake, Autotools και άλλα. Ένα αρχείο έργου μπορεί να περιέχει πληροφορίες όπως ποια αρχεία περιλαμβάνονται στο έργο, προσαρμοσμένα βήματα κατασκευής και ρυθμίσεις για την εκτέλεση των εφαρμογών. Το Qt Creator περιλαμβάνει ένα πρόγραμμα επεξεργασίας κώδικα και ενσωματώνει το Qt Designer για το σχεδιασμό και τη δημιουργία γραφικών διεπαφών χρήστη (GUI) από γραφικά στοιχεία Qt. Ο επεξεργαστής κώδικα στο Qt Creator υποστηρίζει επισήμανση σύνταξης για διάφορες γλώσσες. Επιπλέον, ο επεξεργαστής κώδικα μπορεί να αναλύει τον κώδικα σε γλώσσες C++ και QML και ως αποτέλεσμα παρέχεται η συμπλήρωση κώδικα, η βοήθεια που είναι ευαίσθητη στο περιβάλλον, η σημασιολογική πλοήγηση. Το Qt Designer είναι ένα εργαλείο για το σχεδιασμό και τη δημιουργία γραφικών διεπαφών χρήστη (GUI) από γραφικά στοιχεία Qt. Είναι δυνατό να συνθέσετε και να προσαρμόσετε τα γραφικά στοιχεία ή τα παράθυρα διαλόγου και να τα δοκιμάσετε χρησιμοποιώντας διαφορετικά στυλ και αναλύσεις απευθείας στο πρόγραμμα επεξεργασίας. Τα γραφικά στοιχεία και οι φόρμες που δημιουργούνται με το Qt Designer ενσωματώνονται με προγραμματισμένο κώδικα, χρησιμοποιώντας το μηχανισμό σημάτων και θυρίδων Qt. Το Qt Quick Designer είναι ένα εργαλείο για την ανάπτυξη κινούμενων εικόνων χρησιμοποιώντας μια δηλωτική γλώσσα προγραμματισμού QML. Το Qt Creator παρέχει υποστήριξη για τη δημιουργία και την εκτέλεση εφαρμογών Qt για επιτραπέζια περιβάλλοντα (Windows, Linux, FreeBSD και macOS), κινητές συσκευές (Android, BlackBerry, iOS, Maemo και MeeGo) και ενσωματωμένες συσκευές Linux. Οι ρυθμίσεις κατασκευής επιτρέπουν στο χρήστη να κάνει εναλλαγή μεταξύ στόχων κατασκευής, διαφορετικών εκδόσεων Qt και διαμορφώσεων κατασκευής. Για στόχους φορητών συσκευών, το Qt Creator μπορεί να δημιουργήσει ένα πακέτο εγκατάστασης, να το εγκαταστήσει σε μια κινητή συσκευή που είναι συνδεδεμένη στον υπολογιστή ανάπτυξης και να το εκτελέσει εκεί.

1.3 ROSLAUNCH

Το roslaunch είναι ένα εργαλείο για την εύκολη εκκίνηση πολλών κόμβων ROS τοπικά και απομακρυσμένα μέσω SSH, καθώς και για τη ρύθμιση παραμέτρων στον διακομιστή παραμέτρων. Περιλαμβάνει επιλογές για αυτόματη επανεκκίνηση διεργασιών που έχουν ήδη πεθάνει. Το roslaunch λαμβάνει ένα ή περισσότερα αρχεία διαμόρφωσης XML (με την επέκταση .launch) που καθορίζουν τις παραμέτρους που θα οριστούν και τους κόμβους που θα εκκινήσουν, καθώς και τα μηχανήματα στα οποία πρέπει να εκτελούνται.

1.4 RVIZ

Το rviz (Εργαλείο οπτικοποίησης ρομπότ) είναι ένας τρισδιάστατος οπτικοποιητής που χρησιμοποιείται για την οπτικοποίηση των ρομπότ, των περιβαλλόντων στα οποία εργάζονται και των δεδομένων αισθητήρων. Είναι ένα εργαλείο υψηλής παραμετροποίησης, με πολλούς διαφορετικούς τύπους οπτικοποιήσεων και προσθηκών. Το Unified Robot Description Format (URDF) είναι μια μορφή αρχείου XML για περιγραφή μοντέλου ρομπότ. Με τον προσομοιωτή σε λειτουργία, ανοίξτε το rviz. Στο αριστερό πλαίσιο στο κάτω μέρος, κάντε κλικ στο κουμπί Προσθήκη και, στη συνέχεια, στην καρτέλα Κατά θέμα προσθέστε το θέμα /map και το θέμα /σάρωση. Στη συνέχεια, στην καρτέλα By display type, προσθέστε τον τύπο RobotModel. Στο αριστερό πλαίσιο κάτω από την ενότητα LaserScan που προστέθηκε πρόσφατα, αλλάξτε το μέγεθος σε 0,1 μέτρα για πιο ξεκάθαρη απεικόνιση του lidar (εμφανίζεται στο ουράνιο τόξο). Μπορείτε να χρησιμοποιήσετε ένα πληκτρολόγιο ή ένα joystick USB για να μετακινήσετε το αυτοκίνητο ή μπορείτε να τοποθετήσετε το αυτοκίνητο χειροκίνητα κάνοντας κλικ στο κουμπί Εκτίμηση πόζας 2D στο επάνω μέρος της οθόνης και σύροντας το ποντίκι σας στην επιθυμητή στάση. Ο προσομοιωτής δημιουργήθηκε με δύο κύριους στόχους: την ομοιότητα με το πραγματικό αυτοκίνητο και τη γρήγορη δημιουργία πρωτοτύπων αγωνιστικών αλγορίθμων. Ο κόμβος προσομοιωτή γράφτηκε έτσι ώστε να μπορεί να αντικατασταθεί με το ίδιο το αυτοκίνητο F1/10 και αν όλα τα ονόματα των θεμάτων παραμείνουν ίδια, μπορεί να εκτελεστεί ο ίδιος ακριβής κωδικός για την οδήγηση του αυτοκινήτου. Οι υπόλοιποι κόμβοι ROS είναι οργανωμένοι έτσι ώστε να μπορούν να προστεθούν γρήγορα νέοι αλγόριθμοι σχεδιασμού και να εναλλάσσονται μεταξύ τους κατά την οδήγηση.



Ο δημόσιος προσομοιωτής μας περιλαμβάνει έναν απλό τυχαίο κόμβο προγράμματος οδήγησης ως παράδειγμα για το πώς πρέπει να μοιάζει ένας κόμβος σχεδιασμού. Κάθε σχεδιαστής μπορεί να ακούσει τα δεδομένα του αισθητήρα που δημοσιεύονται από τον προσομοιωτή και στη συνέχεια να δημοσιεύσει μηνύματα AckermannDrive στο δικό του συγκεκριμένο θέμα (π.χ. /random_drive). Ο κόμβος mux ακούει όλα αυτά τα θέματα, στη συνέχεια παίρνει το μήνυμα από όποιο πρόγραμμα σχεδιασμού είναι ενεργοποιημένο και το δημοσιεύει στο κύριο θέμα / μονάδα δίσκου, το οποίο ακούει ο προσομοιωτής. Σημειώστε ότι χρησιμοποιούνται μόνο η ταχύτητα και η γωνία διεύθυνσης που καθορίζονται στο μήνυμα. Ο κόμβος mux ακούει επίσης μηνύματα joystick και πληκτρολογίου, για χειροκίνητη οδήγηση. Ο κόμβος του ελεγκτή συμπεριφοράς λέει στον κόμβο mux σε ποιον προγραμματιστή βρίσκεται μέσω του θέματος /mux. Από προεπιλογή, κάθε προγραμματιστής (συμπεριλαμβανομένου του πληκτρολογίου και του joystick) αντιστοιχίζεται σε ένα κουμπί joystick και ένα πλήκτρο πληκτρολογίου και απλώς ενεργοποιούνται και απενεργοποιούνται χειροκίνητα. Επιπλέον, κατά τη σύγκρουση, το αυτοκίνητο θα σταματήσει και όλα τα κανάλια mux θα είναι καθαρά - τίποτα δεν θα είναι υπό έλεγχο μέχρι τη χειροκίνητη επέμβαση. Για να μετακινήσετε αμέσως το αυτοκίνητο σε νέα κατάσταση, δημοσιεύστε μηνύματα Pose στο θέμα /pose. Αυτό μπορεί να είναι χρήσιμο για το σενάριο του αυτοκινήτου μέσω μιας σειράς αυτοματοποιημένων δοκιμών. Το προσομοιωμένο lidar δημοσιεύεται στο θέμα /scan ως μηνύματα LaserScan. Η στάση του αυτοκινήτου μεταδίδεται ως μετασχηματισμός μεταξύ του πλαισίου του χάρτη και του πλαισίου βάσης_σύνδεσης. base_link είναι το κέντρο του πίσω άξονα. Το πλαίσιο λέιζερ ορίζει το πλαίσιο από το οποίο λαμβάνεται η σάρωση lidar και ένας άλλος μετασχηματισμός μεταδίδεται μεταξύ αυτού και της βάσης_σύνδεσης.

1.5 CATKIN

Το catkin είναι το σύστημα κατασκευής ROS, το οποίο έχει αντικαταστήσει το rosbuild από το ROS Groovy. Το catkin βασίζεται στο CMake και είναι ομοίως cross-platform, open-source και ανεξάρτητο από τη γλώσσα.

1.6 CMAKE

Στην ανάπτυξη λογισμικού, το CMake είναι δωρεάν και ανοιχτού κώδικα λογισμικό πολλαπλών πλατφορμών για αυτοματοποίηση κατασκευής, δοκιμή, συσκευασία και εγκατάσταση λογισμικού χρησιμοποιώντας μια μέθοδο ανεξάρτητη από μεταγλωττιστή.Το CMake δεν είναι ένα σύστημα κατασκευής από μόνο του. δημιουργεί τα αρχεία κατασκευής ενός άλλου συστήματος. Υποστηρίζει ιεραρχίες καταλόγων και εφαρμογές που εξαρτώνται από πολλές βιβλιοθήκες. Χρησιμοποιείται σε συνδυασμό με εγγενή περιβάλλοντα κατασκευής όπως το Make, το Qt Creator, το Ninja, το Android Studio, το Xcode της Apple και το Microsoft Visual Studio. Έχει ελάχιστες εξαρτήσεις, απαιτώντας μόνο έναν μεταγλωττιστή C++ στο δικό του σύστημα κατασκευής. Το CMake διανέμεται ως ελεύθερο λογισμικό ανοιχτού κώδικα με άδεια BSD-3-Clause

2 Κεφάλαιο : Μοντέλο

2.1 ΤΙ ΘΑ ΧΡΕΙΑΣΤΟΥΜΕ ΓΙΑ ΤΟ ΟΧΗΜΑ ΜΑΣ

Για να φτιάξουμε το f1tenth θα πρέπει πρώτα από όλα να έχουμε το ROS εγκατεστημένο στο λαπτοπ μας. Για να το εγκαταστήσουμε θα πρέπει να έχουμε το λογισμικό Linux 20.04 γιατί με αυτό το λογισμικό το ROS δουλεύει πιο ήπια αντί με τα Windows. Για να εγκαταστήσουμε το ROS θα σας έχω το λινκ εδώ: (αυτό είναι το λινκ για να το εγκαταστήσουμε (https://wiki.ros.org/noetic/Installation/Ubuntu)). Εγώ έχω εγκαταστήσει το noetic διότι το f1tenth δουλεύει καλύτερα με αυτό το ROS. Το noetic είναι το δεκατοτριτο κατά σειρά από το ROS που βγήκε και είναι από την σειρά ROS 1, έχει και βγει και η σειρά ROS 2 που σε εκείνο εμπεριέχονται τα ardent apalone , bouncy bolson, foxy fitzroy, humble hawksbill και πολλά άλλα . Για το αυτοκινητάκι για να το φτιάξουμε εκτός από αυτά που ανέφερα επίσης θα χρειαστούμε :

Μηχανικά μέρη για το αυτοκινητάκι :

1. **Traxxas RC rally car (συνίσταται για το μοντέλο: Ford Fiesta ST)**

2. **(1 κομμάτι) Velineon 3351R/3500 χωρίς ψήκτρες DC motor**
3. **(1 κομμάτι) σασί που θα είναι κομμένο με λέιζερ για να τοποθετήσουμε όλα τα υλικά**
4. **(1 κομμάτι) βάση LIDAR με 2 βραχίονες σε σχήμα C**
5. **(2 κομμάτια) σπειρώματα 14mm M3 standoffs**
6. **(4 κομμάτια) 19 mm M3 standoffs**
7. **(8 κομμάτια) 35mm M3 standoffs**
8. **(4 κομμάτια) 45 mm M3 standoffs**
9. **(35 κομμάτια) 10 mm M3 βίδες**
10. **(4 κομμάτια) 20 mm M3 βίδες**
11. **(4 κομμάτια) 8 mm αποστάτες στρογγυλοί νάιλον**
12. **Ρολό ταινίας διπλής όψεως (για τοποθέτηση διανομέων USB και προαιρετικά FOCbox)**

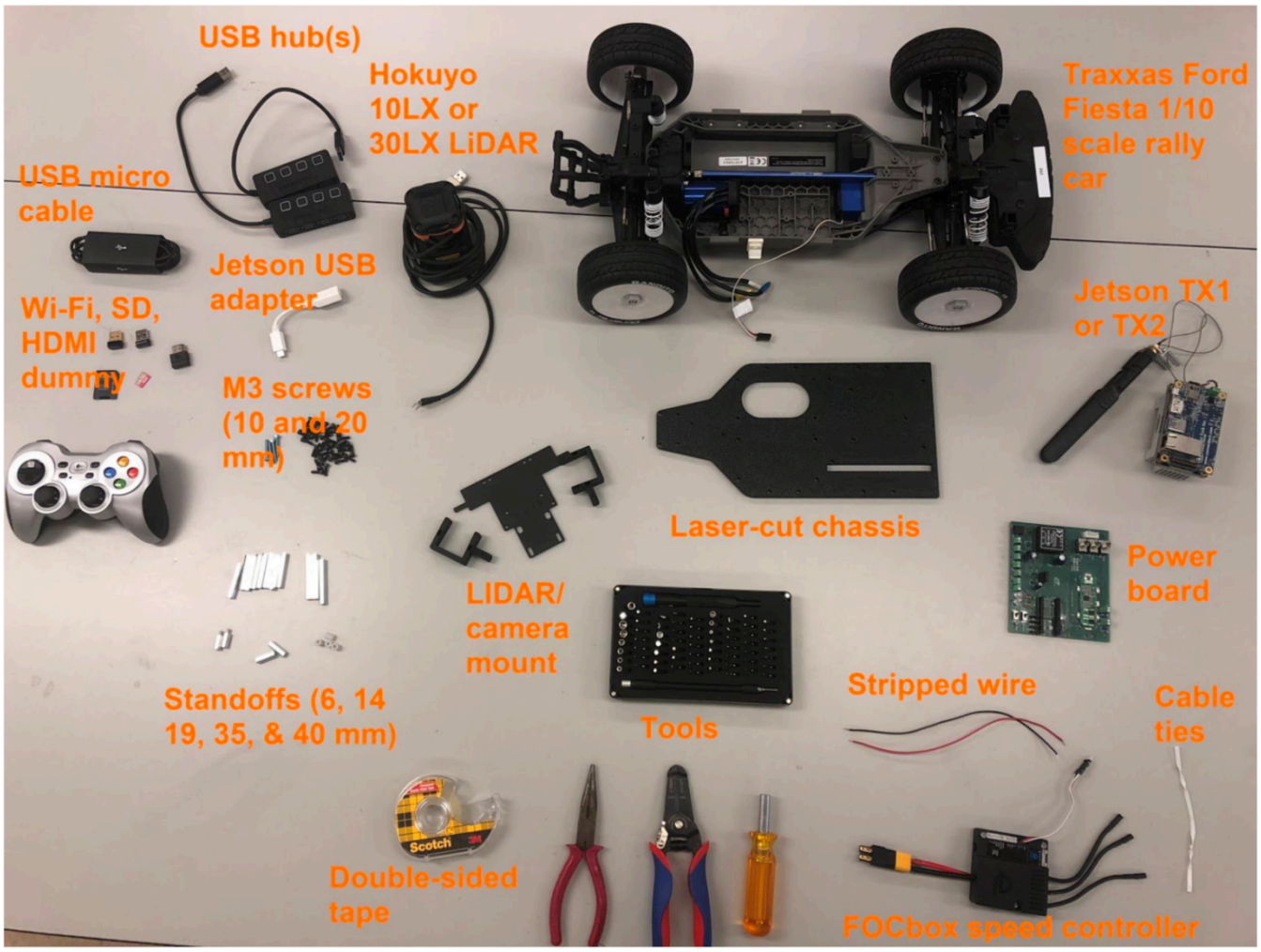
Ηλεκτρονικά μέρη για το αυτοκινητάκι:

1. **(1 κομμάτι) πλακέτα τροφοδοσίας (που ελήφθη από την ομάδα του Πανεπιστημίου της Πενσυλβάνια. Σχεδιάστηκε στο Penn, τυπώθηκε από www.4pcb.com)**
2. **(1 κομμάτι) Nvidia Jetson TX1 ή TX2 with Wi-Fi Αντέννα (περιλαμβάνονται στον πίνακα ανάπτυξης)**
3. **(1 κομμάτι) Τροχιά για σανίδα μεταφοράς**
4. **(1 κομμάτι) Hokuyo 10LX ή 30LX LIDAR**
5. **(1 κομμάτι) Traxxas LiPo ή NiMH battery (τουλάχιστον 9 Βολτ)**
6. **(1 κομμάτι) FOCbox ή VESC 4.12 ηλεκτρονικός ελεγκτής για ταχύτητα**
7. **(1 κομμάτι) USB hub (τουλάχιστον 6 θυρών)**
8. **(1 κομμάτι) το πολύ 1 μέτρου USB micro cable**
9. **(2 κομμάτι) καρούλια/λωρίδες σύρματος 22 AWG διαφορετικών χρωμάτων (κατά προτίμηση κόκκινο και μαύρο)**
10. **USB πληκτρολόγιο και ποντίκι**
11. **1 HDMI για την οθόνη**
12. **Προαιρετικά: (1) εικονικό βύσμα HDMI και (1) προσαρμογέας USB Wi-Fi για σύνδεση στο αυτοκίνητο μέσω VNC**

Εργαλεία που θα χρειαστούμε για το αυτοκινητάκι:

1. **Ένα μετρό ή χάρακας**
2. **Μια πένσα**
3. **Έναν απογυμνωτή καλωδίων**
4. **Ένα κατσαβίδι ίσιο πλάτους 2mm ή και μικρότερο**
5. **Ένα εξαγωνικό κλειδί οδηγού διαμέτρου 5/64 ιντσών ή μικρό κατσαβίδι Philips (1) Εξαγωνικός οδηγός ή κλειδί υποδοχής (για να συγκρατεί τις βάσεις στη θέση τους)**
6. **Ένα Torx σαν κατσαβίδι για να μπορέσουμε να σφίξουμε**

Αυτά θα χρειαστούμε για να μπορέσουμε να φτιάξουμε το αυτοκινητάκι μας. Στην παρακάτω φωτογραφία φαίνεται ακριβώς τα παραπάνω εργαλεία και μέρη :



Σχ. 1..0 (Τι ακριβώς χρειαζόμαστε για το αυτοκινητάκι)

2.2 ΚΑΤΑΣΚΕΥΗ ΤΟΥ ΟΧΗΜΑΤΟΣ ΜΑΣ

Λήψη της πλακέτας τροφοδοσίας :

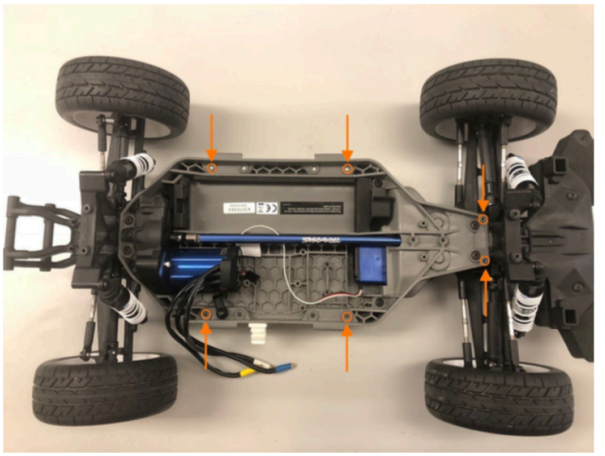
1. Οδηγίες για την απόκτηση της πλακέτας τροφοδοσίας από την Penn,
2. και έναν σύνδεσμο προς αρχεία που θα στέλνατε σε μια εταιρεία PCB όπως το 4PCB ή το PCBWay. Αυτήν τη στιγμή στο github.

Σημείωση σχετικά με το γιατί έχουμε μια πλακέτα τροφοδοσίας: Η πλακέτα τροφοδοσίας χρησιμοποιείται για να παρέχει μια σταθερή πηγή τάσης για το αυτοκίνητο και τα περιφερειακά του, καθώς η τάση της μπαταρίας πέφτει καθώς η μπαταρία τελειώνει. Η πλακέτα δεν κάνει καμία φόρτιση της μπαταρίας, επομένως θα χρειαστείτε έναν φορτιστή Traxxas EZ-Peak για να τη φορτίσετε (μπορείτε να τους

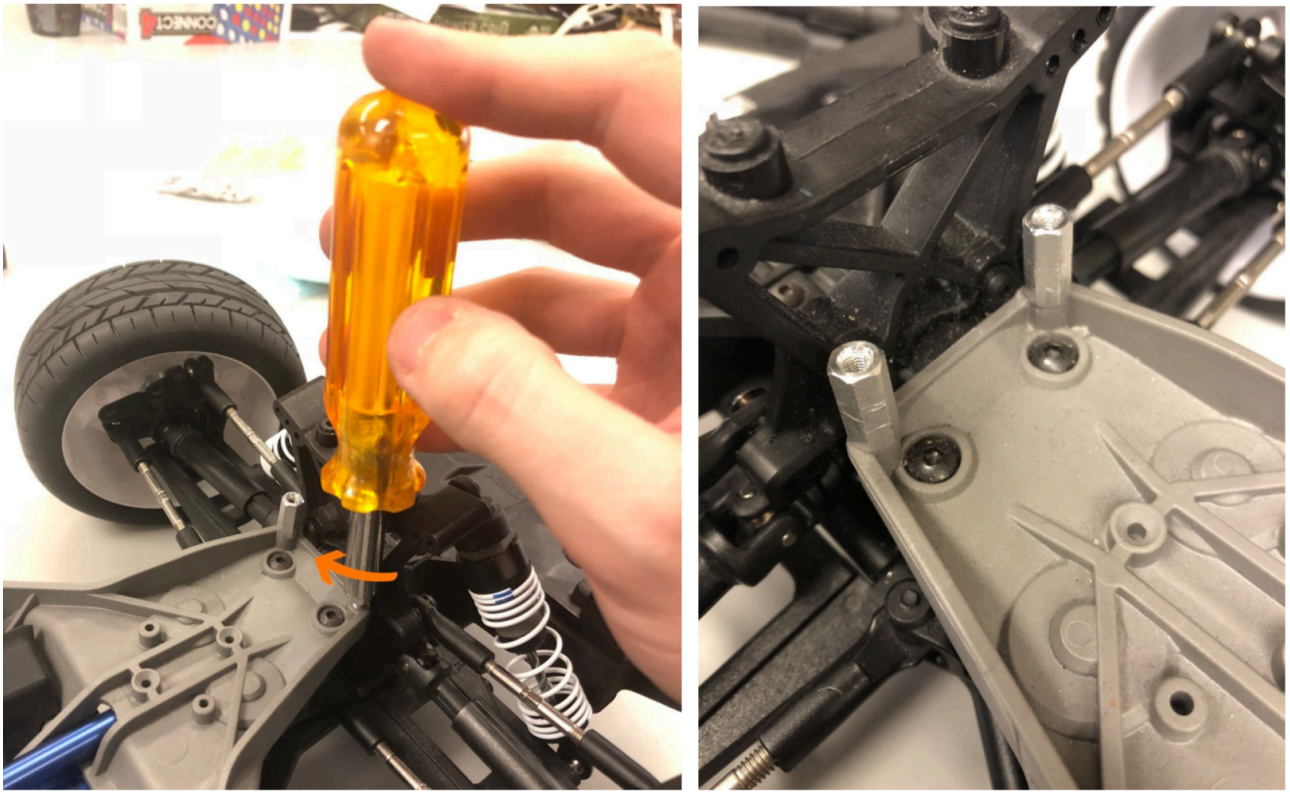
βρείτε στον ιστότοπο της Traxxas). Προς το παρόν, δεν υπάρχει τρόπος να γνωρίζουμε το επίπεδο φόρτισης της μπαταρίας εκτός από τη μέτρησή της με ένα πολύμετρο ή το εργαλείο BLDC καθώς λειτουργεί, αλλά θα μπορούσαμε να σκεφτούμε να προσθέσουμε μια οθόνη LED χαμηλής τάσης ή οθόνη LCD επτά τμημάτων (για να δείξει την τάση). Η μονάδα προστασίας LIPO και οι πράσινοι σύνδεσμοι δεν χρησιμοποιούνται αυτήν τη στιγμή και αποτελούν κληρονομιά από προηγούμενες επαναλήψεις αυτοκινήτων F1/10 που χρησιμοποιούσαν τον μικροελεγκτή Teensy ως οδηγό κινητήρα.

Εγκαταστάσεων των αντιστάσεων του σώματος :

Θα ξεκινήσουμε τοποθετώντας τα τέσσερα 45 mm standoffs M3 στις οπές του κύριου αμαξώματος του αυτοκινήτου που απεικονίζεται παρακάτω. Θα στερεωσουμε τα στηρίγματα στο κάτω μέρος του αυτοκινήτου χρησιμοποιώντας βίδες M3 10 mm και θα χρησιμοποιήσουμε είτε ρύγχους είτε εξάγωνο οδηγό για να κρατήσουμε τη βάση στη θέση της ενώ γυρίζουμε τη βίδα στην άλλη πλευρά. Να προσέξουμε στο που θα τοποθετήσουμε τα standoffs αφού υπάρχουν αρκετές οπές στερέωσης στη βάση του αυτοκινήτου. Δες την παρακάτω εικόνα για διευκρίνιση. Στη συνέχεια, θα τοποθετήσουμε δύο σπειρώματα 14mm M3 standoffs στις μπροστινές οπές στη βάση του αυτοκινήτου, χρησιμοποιώντας την πένσα ή το εξάγωνο οδηγώ για να βιδώσουμε τα standoffs στις οπές. Να μην τοποθετηθεί ακόμα το πλαίσιο στο αυτοκίνητο, καθώς πρέπει ακόμα να τοποθετήσουμε το Jetson και την πλακέτα ισχύος στο σασί.



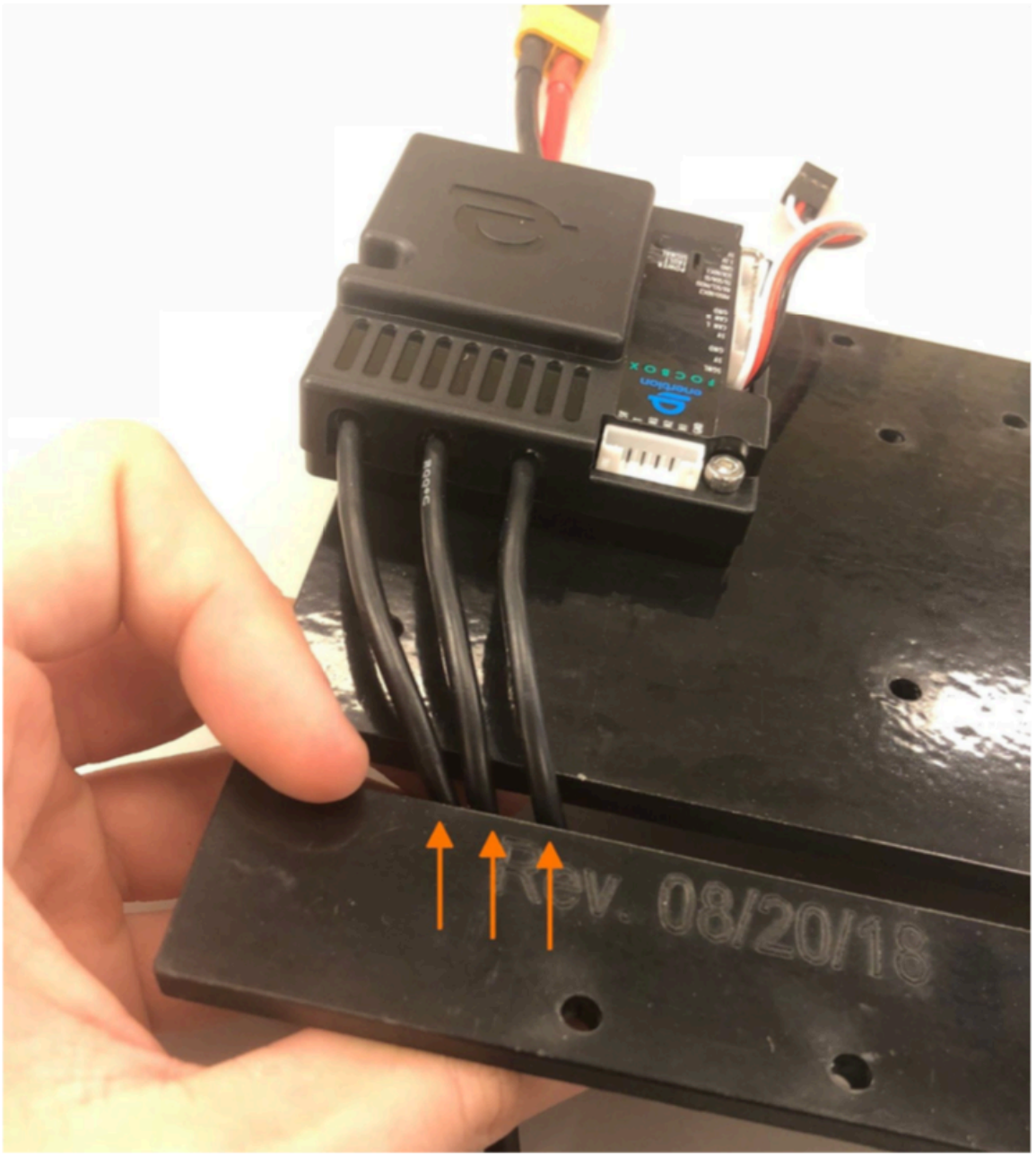
Σχ. 1.1 (Στερέωση των στηριγμάτων)



Σχ. 1.2

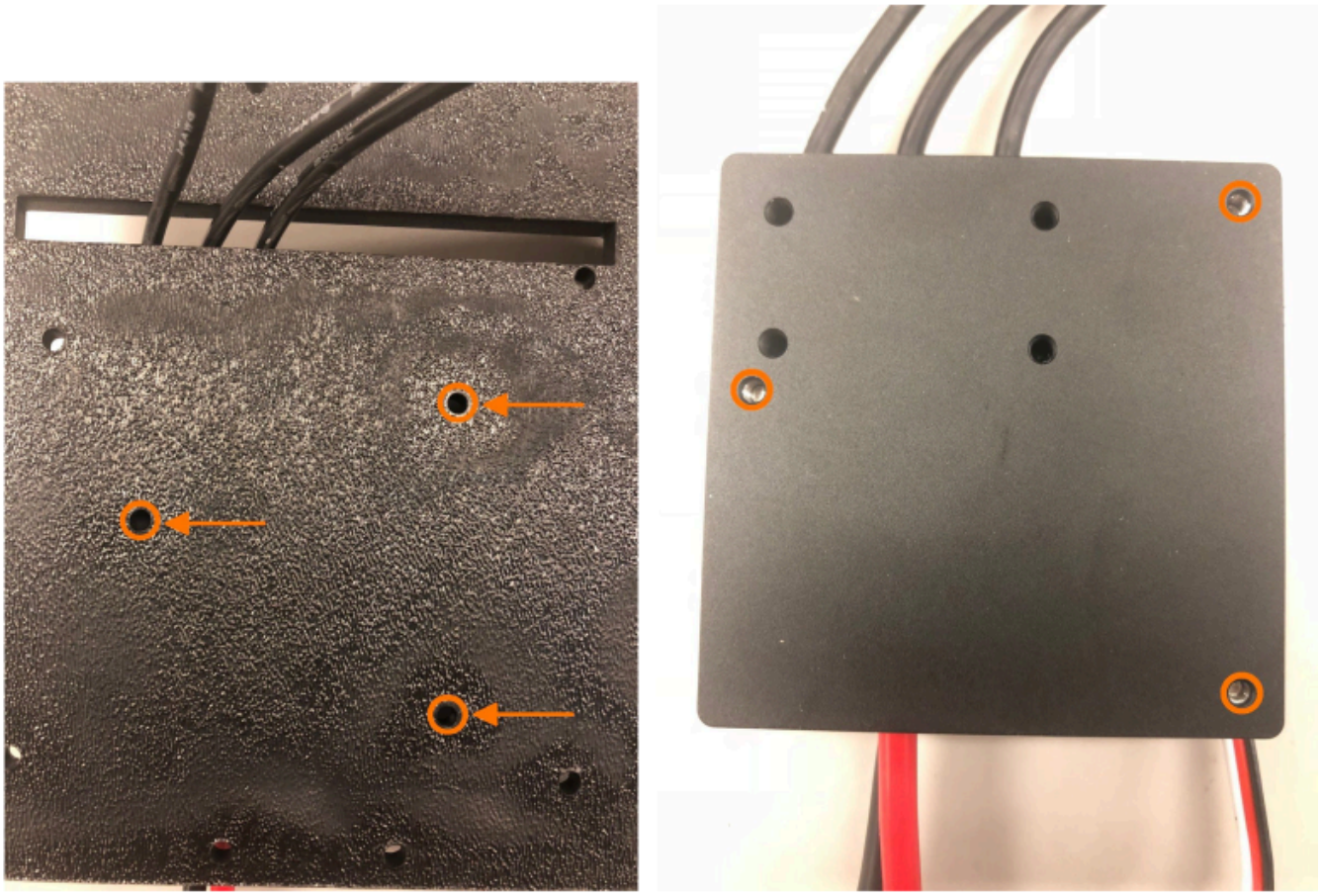
Τοποθέτηση του FOCbox στο πλαίσιο:

Θα περάσουμε και τα τρία καλώδια του κινητήρα για το FOCbox μέσα από την ορθογώνια υποδοχή στο πλαίσιο όπως φαίνεται παρακάτω.



Σχ. 1.3

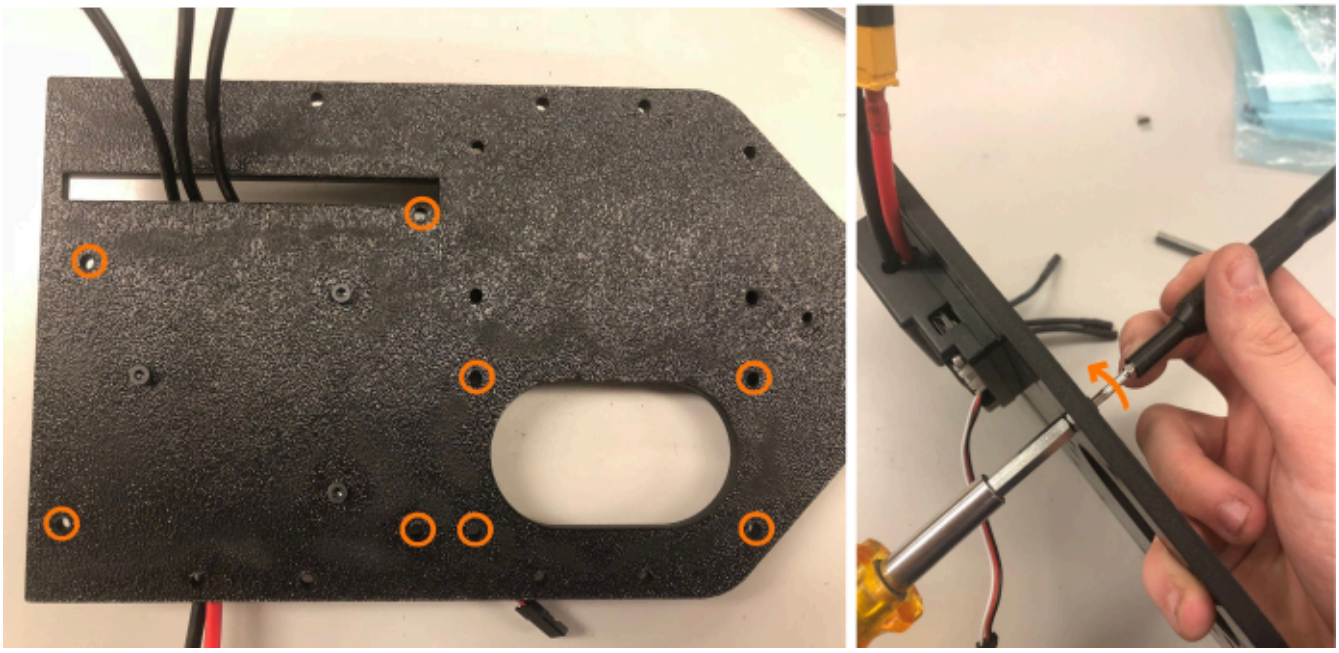
Εναλλακτικά θα κάνουμε στερέωση του κουτιού FOC εάν οι βίδες δεν ταιριάζουν: Ορισμένα κουτιά FOC (τα πιο πρόσφατα κατασκευασμένα) έχουν μικρότερες οπές για βίδες που δεν εισέρχονται βίδες μεγέθους M3. Εάν αυτό ισχύει για το FOCbox σας, μπορείτε να χρησιμοποιήσετε μερικά κομμάτια ταινίας διπλής όψης για να ασφαλίσετε το FOCbox ή μπορούμε να βάλουμε και κόλλα για να στειρωθεί καλά απλά θα πρέπει να προσέχουμε τα καλώδια και FOCbox.



Σχ. 1.4

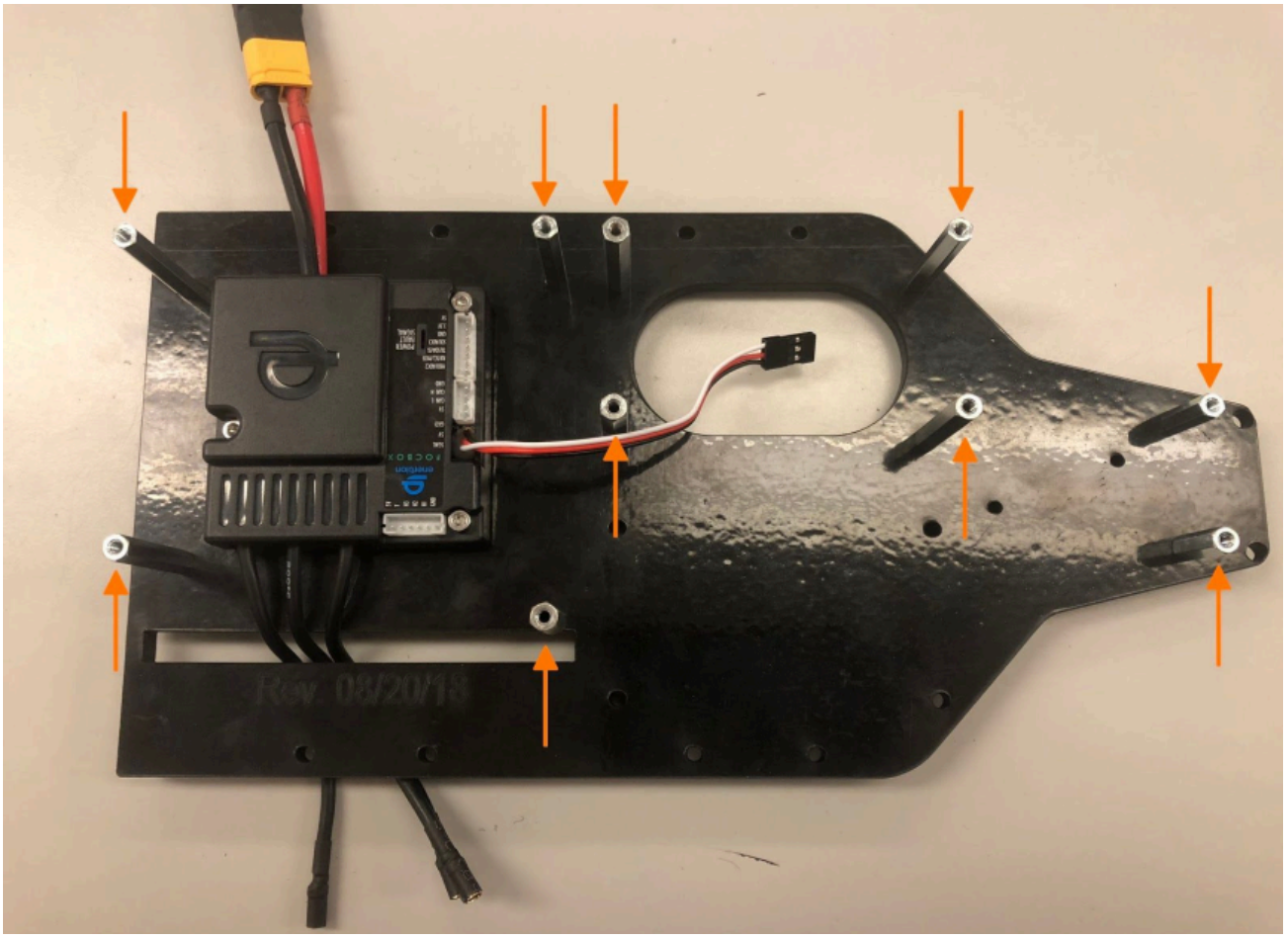
Τώρα θα τοποθετήσουμε τα στηρίγματα του πλαισίου:

Θα τοποθετήσουμε οκτώ βάσεις M3 35 mm στη γυαλιστερή πλευρά του μαύρου πλαισίου κοπής με λέιζερ στις θέσεις που φαίνονται παρακάτω. Θα περάσουμε τις βίδες M3 10 mm μέσα από τις οπές διάτρησης και θα βιδώσουμε τις στις βάσεις για να τις θα στερεωσουμε. (Σημαντικό: Εάν δεν τοποθετήσουμε τις βάσεις στη γυαλιστερή πλευρά, η πλακέτα τροφοδοσίας δεν θα χωρέσει θα είναι λάθος δηλαδή και έτσι οι οπές των βιδών θα είναι λάθος ευθυγραμμισμένες). Σημειώστε ότι υπάρχουν πολλές τρύπες στο σασί, οπότε θα πρέπει να χρησιμοποιήσουμε τις κατάλληλες τρύπες. Στην παρακάτω εικόνα, οι 4 βίδες στα αριστερά (που είναι το πίσω μέρος του αυτοκινήτου) θα συγκρατήσουν τελικά την πλακέτα ισχύος, οπότε αυτός είναι ένας καλός τρόπος για να δείτε αν είναι σωστά τοποθετημένες. Οι 4 βίδες στα δεξιά (2 σε κάθε πλευρά της οβάλ εγκοπής) θα συγκρατήσουν τελικά τον φορέα Orbitty συνδεδεμένο στο Jetson.



Σχ. 1.5 (Αυτές είναι οι σωστές τρύπες που θα χρησιμοποιηθούν)

Θα τοποθετήσουμε δύο ακόμη βάσεις (19 mm M3) στο μπροστινό μέρος του πλαισίου για το LIDAR και θα το στερεώσουμε με βίδες 10 mm. Το επάνω μέρος του πλαισίου πρέπει να μοιάζει με την παρακάτω εικόνα.

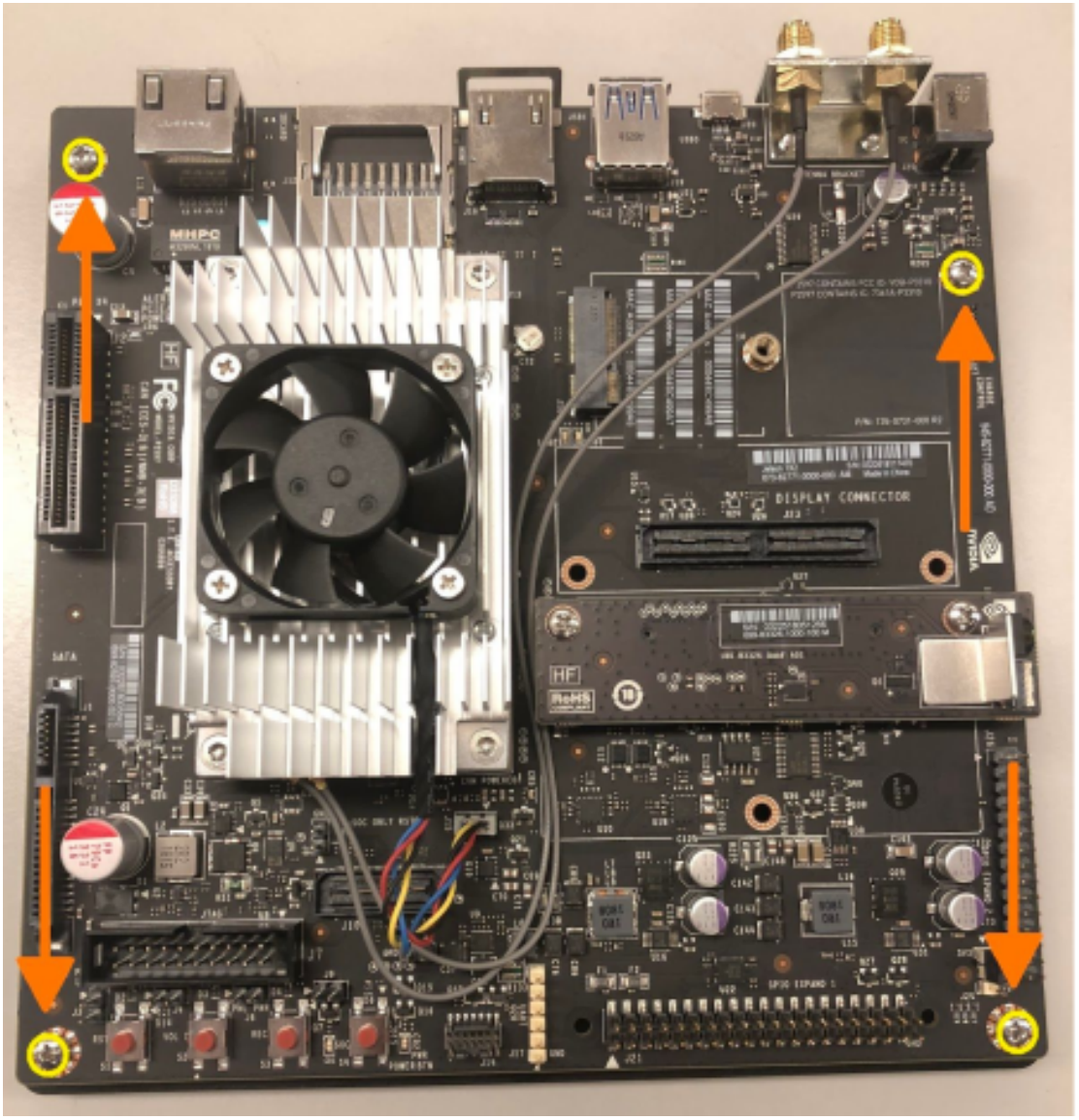


Σχ. 1.6

Αποσύνδεση του Jetson από την πλακέτα:

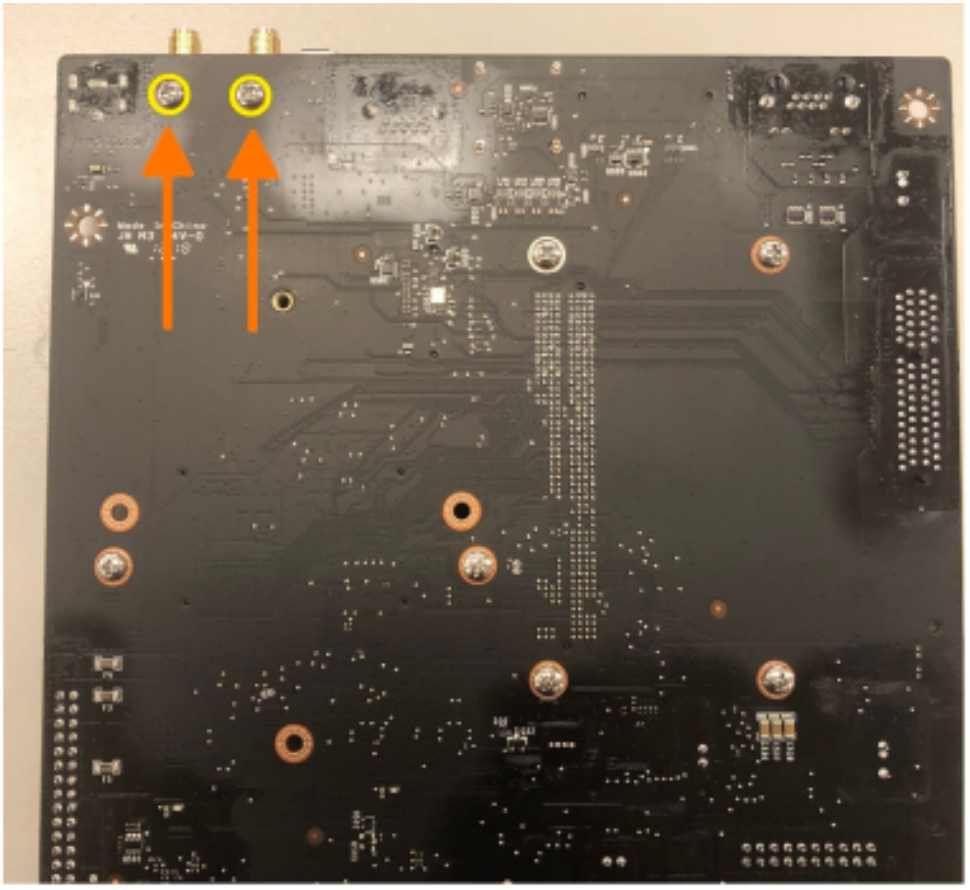
Όταν αγοράζουμε ένα Jetson, είναι προσαρτημένο σε μια πλακέτα ανάπτυξης. Για να το χρησιμοποιήσουμε στο αυτοκίνητο, θα χρειαστεί να ξεβιδώσουμε το Jetson και την κεραία Wi-Fi του από την πλακέτα ανάπτυξης.

Πριν αφαιρέσουμε την κεραία, θα χρειαστεί να αφαιρέσουμε την κάτω πλάκα από την πλακέτα ανάπτυξης. Αφαιρούμε τις τέσσερις βίδες που σημειώνονται παρακάτω και σηκώνουμε την πλακέτα ανάπτυξης μακριά από την πλάκα. Όπως φαίνεται στην παρακάτω φωτογραφία:



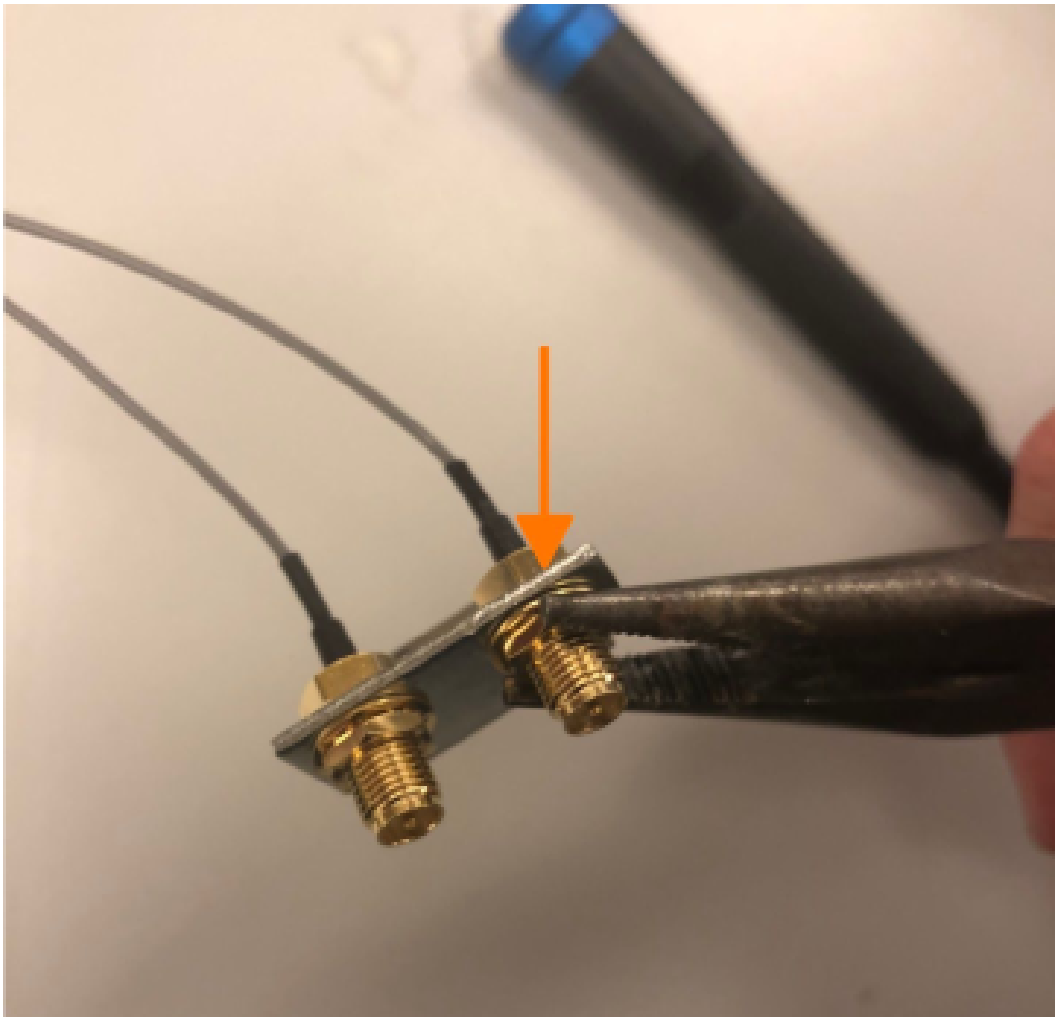
Σχ. 1.7

Στη συνέχεια, αφαιρούμε την κεραία Wi-Fi ξεβιδωνοντας τις δύο βίδες που φαίνονται παρακάτω. Κρατάμε τις βίδες σε ασφαλές μέρος, καθώς θα τις χρησιμοποιήσουμε ξανά σε λίγο για να στερεώνουμε τις κεραίες στις βάσεις.



Σχ. 1.8

Αφαιρούμε τα δύο παξιμάδια στο χρώμα του ορείχαλκου που συγκρατούν τις κεραίες στο στήριγμα σχήματος L και, στη συνέχεια, αφαιρούμε τις δύο κεραίες από το στήριγμα. βοηθούσε αν είχαμε δύο ζευγάρια πένσα: ένα για να συγκρατεί τα πίσω παξιμάδια στη θέση τους και ένα άλλο για να ξεβιδώνει τα παξιμάδια στο άκρο με τους συνδέσμους της κεραίας.

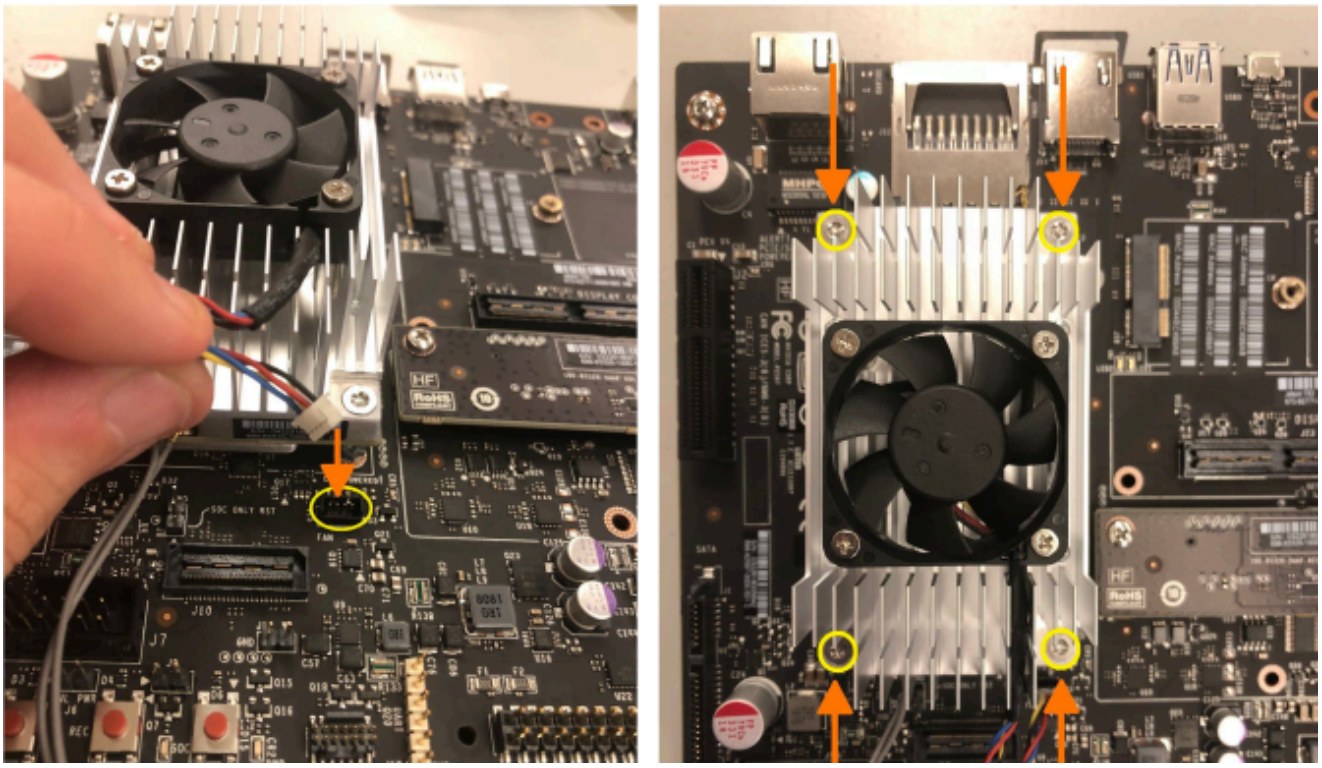


Σχ. 1.9

Θα χρησιμοποιήσουμε ένα κατσαβίδι Phillips, θα περάσουμε εντελώς τις δύο βίδες που της κρατήσαμε νωρίτερα στο στήριγμα όπως απεικονίζεται παρακάτω. Στερεώνουμε δύο βάσεις στα απέναντι άκρα των βιδών και σφίγγουμε με το χέρι μέχρι να μην γυρίζουν πια. Χρησιμοποιώντας την πένσα για να σφίξουμε περισσότερο τις βάσεις ενώ κρατάμε την κεφαλή της βίδας στη θέση της χρησιμοποιώντας το κατσαβίδι. Αφού ολοκληρώσουμε αυτά τα βήματα, τοποθετούμε ξανά τις κεραίες και τις ροδέλες στο στήριγμα και σφίγγουμε ξανά τα ορειχάλκινα παξιμάδια στους συνδετήρες με σπείρωμα.

Σχ. 2.0

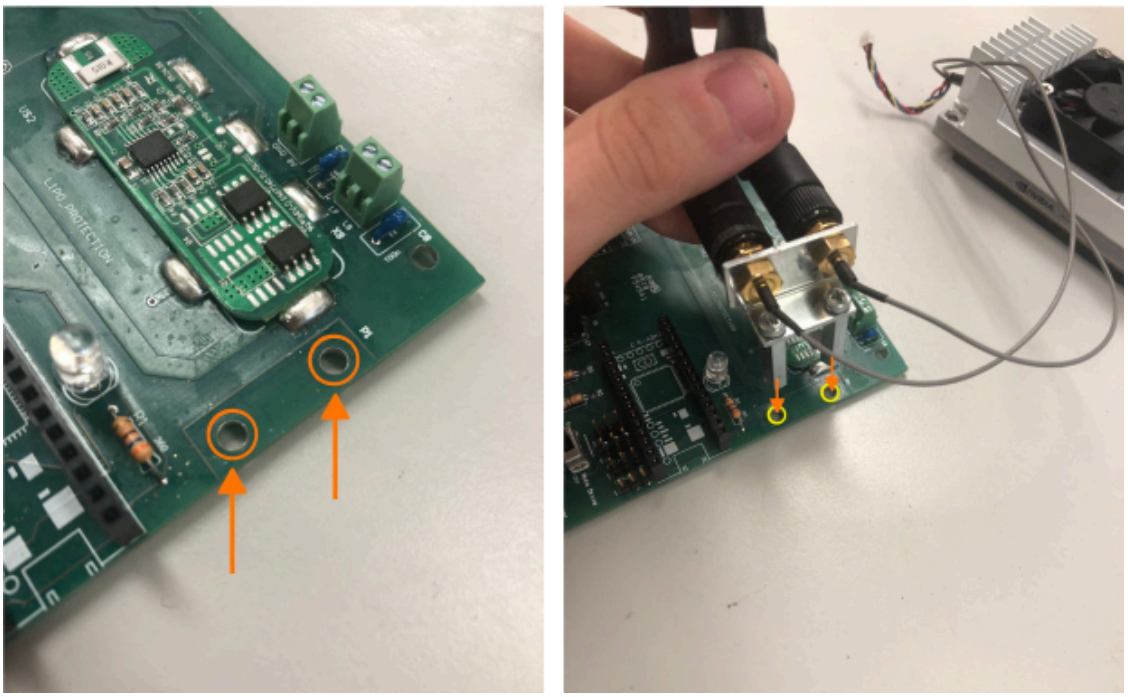
Σε αυτήν την φάση θα αποσυνδέουμε τον ανεμιστήρα του Jetson και θα αφαιρέσουμε το Jetson από την πλακέτα ανάπτυξης χρησιμοποιώντας ένα κατσαβίδι T3 Torx για να ξεβιδωσετε το Jetson (τη μεγάλη ασημένια ψήκτρα) και, στη συνέχεια, τραβώντας το απαλά προς τα πάνω για να το αποσυνδέσουμε από την πλακέτα ανάπτυξης. Ετσι με αυτό το τρόπο θα κρατήσουμε το Jetson σε ασφαλές μέρος ενώ συνδέουμε τις κεραίες στην πλακέτα τροφοδοσίας.



Σχ. 2.1

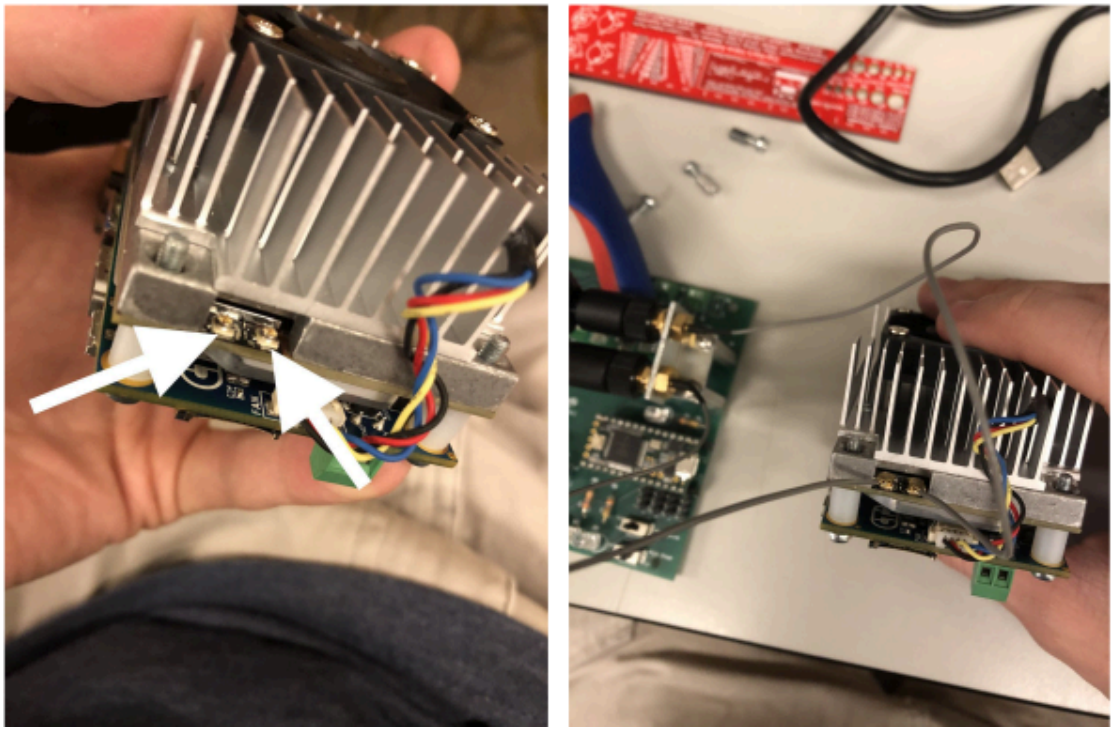
Τοποθέτηση των κεραιών Wi-Fi στην πλακέτα τροφοδοσίας:

Τώρα θα συνδέσουμε τις δύο βάσεις για το στήριγμα της κεραίας Wi-Fi στην πλακέτα τροφοδοσίας, θα πρέπει πρώτα να βεβαιωθούμε ότι οι κεραίες, όταν εγκατασταθούν και επεκταθούν, βρίσκονται πάνω από την πλακέτα και τώρα μακριά από αυτήν. Στο τέλος θα πρέπει να τοποθετήσουμε τις δύο μαύρες κεραίες στους συνδετήρες με σπείρωμα, εάν δεν είναι ήδη ενεργοποιημένοι.



Σχ. 2.2

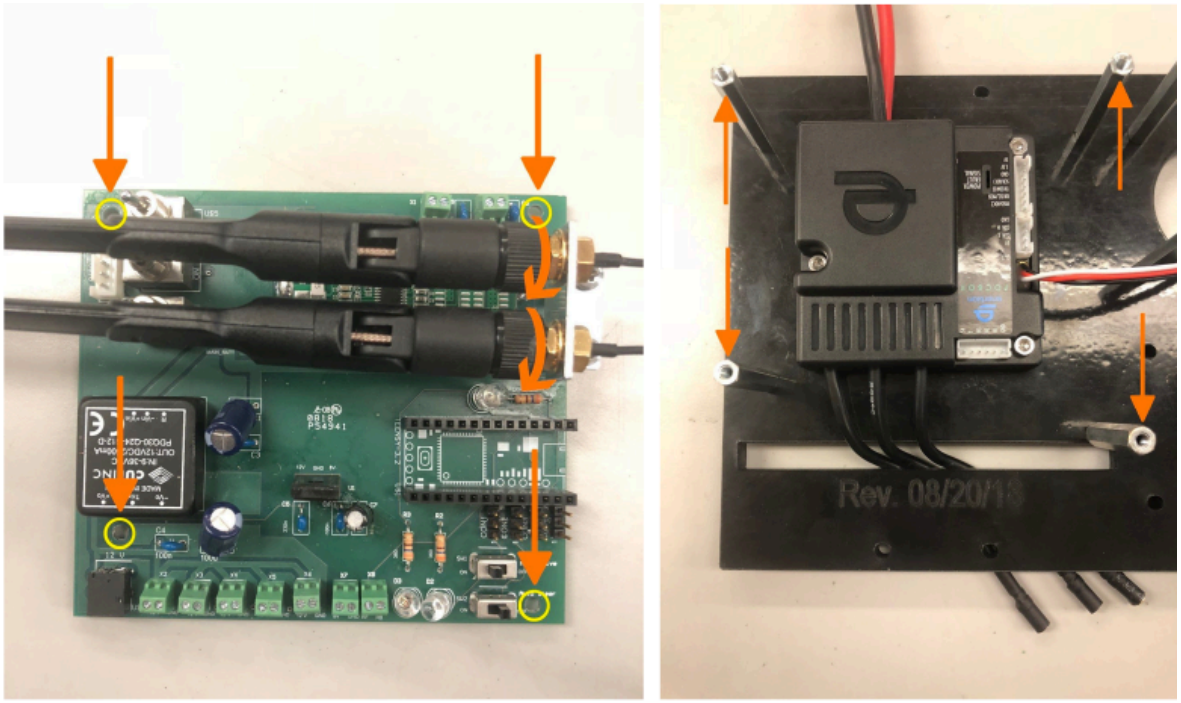
Μετά θα πρέπει να συνδέσουμε τα δύο καλώδια για την κεραία Wi-Fi Jetson στις δύο χρυσές υποδοχές κοντά στην υποδοχή ανεμιστήρα της ψύκτρας (η σειρά των καλωδίων δεν έχει σημασία). Αυτό μπορεί να είναι λίγο δύσκολο, επομένως ίσως θα έπρεπε να χρησιμοποιήσουμε ένα κατσαβίδι επίπεδης κεφαλής για να βεβαιωθούμε ότι οι συνδέσεις είναι σφιχτές. Δεν πρέπει να πιέσουμε πολύ δυνατά, ωστόσο, καθώς μπορεί εύκολα να καταστρέφουν οι σύνδεσμοι εάν χρησιμοποιήσουμε υπερβολική δύναμη.



Σχ.2.3

Τοποθέτηση της πλακέτας στο πλαίσιο:

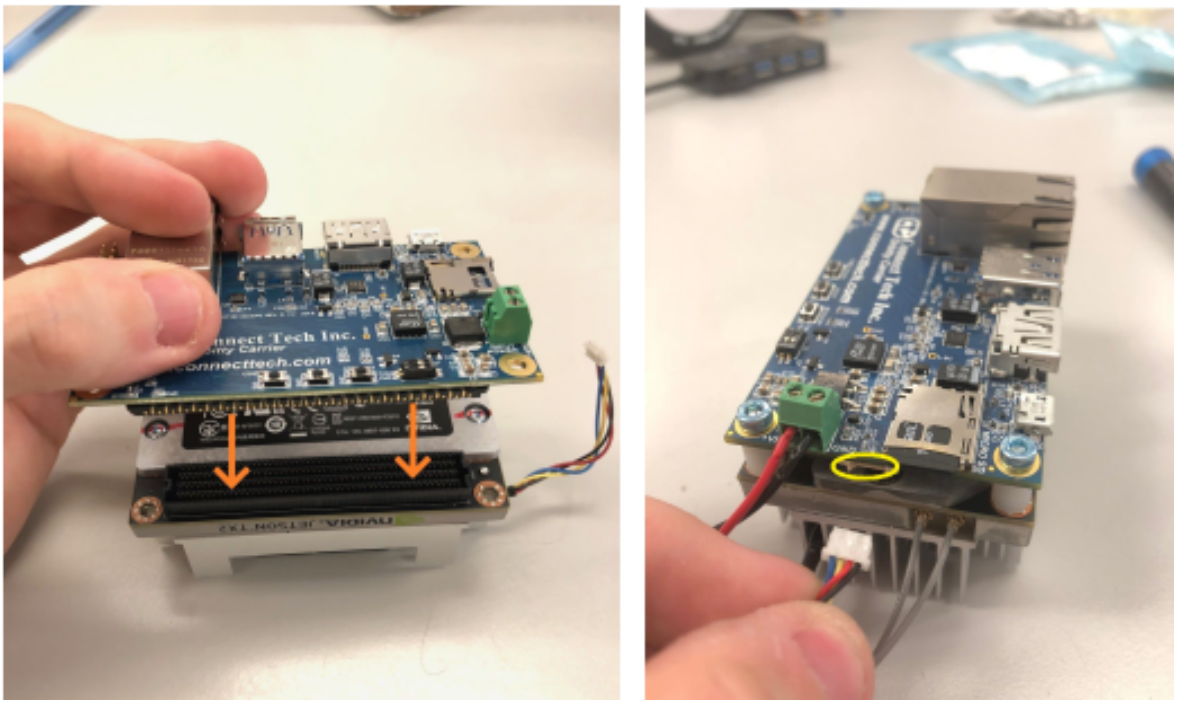
Σε αυτό το σημείο θα βιδώσουμε την πλακέτα ισχύος στις βάσεις του πλαισίου και θα χρησιμοποιήσουμε τις βίδες M3 10 mm. Οι θέσεις των βιδών φαίνονται με βέλη παρακάτω:



Σχ. 2.4

Προσάρτηση της τροχιάς του Jetson στην πλακέτα:

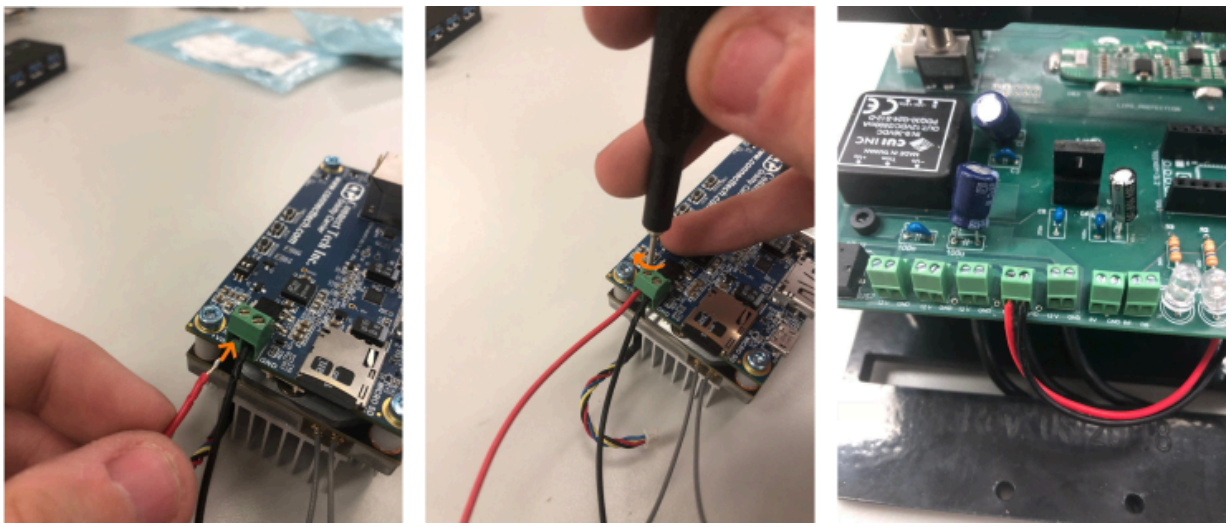
Συνδέουμε την πλακέτα με την τροχιά στο Jetson συνδέοντας τις δύο μακριές μαύρες θύρες και μετά συνδέουμε τον ανεμιστήρα του Jetson στην υποδοχή ανεμιστήρα της πλακέτα με την τροχιά όπως φαίνεται στις παρακάτω εικόνες:



Σχ. 2.5

Οι συνδεσμολογία των Jetson και της πλακέτας:

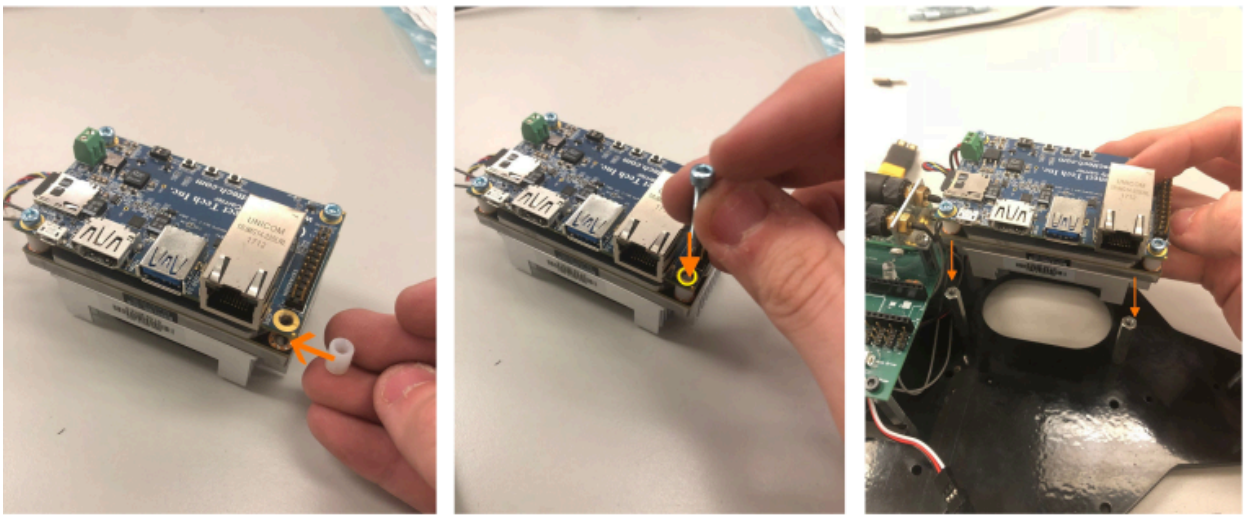
Θα χρειαστούμε δύο κομμάτια καλώδιού οπού και θα κόψουμε περίπου 15 εκ. διαφορετικών χρωμάτων (κατά προτίμηση κόκκινο και μαύρο ή καφέ και μαύρο) και θα τα απογυμνώσουμε. Θα τα συνδέσουμε στην κλεμα που έχει οι πλακέτα στο Jetson και έπειτα θα συνδέσουμε το ένα άκρο ενός καλωδίου στον ακροδέκτη + Vin και το άλλο σε μια από τις δυο κλεμες 2V στην πλακέτα τροφοδοσίας. (Οποιοσδήποτε από τους ακροδέκτες των 1 ή 2 volt είναι αποδεκτός. Για να συνδέσουμε, θα πρέπει να τοποθετήσουμε το απογυμνωμένο άκρο στον ακροδέκτη και βιδώσουμε την κλέμα με ένα μικρό κατσαβίδι). Μετά το άλλο απογυμνωμένο καλώδιο στην κλεμα που λέει GND στο Jetson και στο τον ακροδέκτη GND στο αντίστοιχο μπλοκ ακροδεκτών στην πλακέτα τροφοδοσίας:



Σχ. 2.6

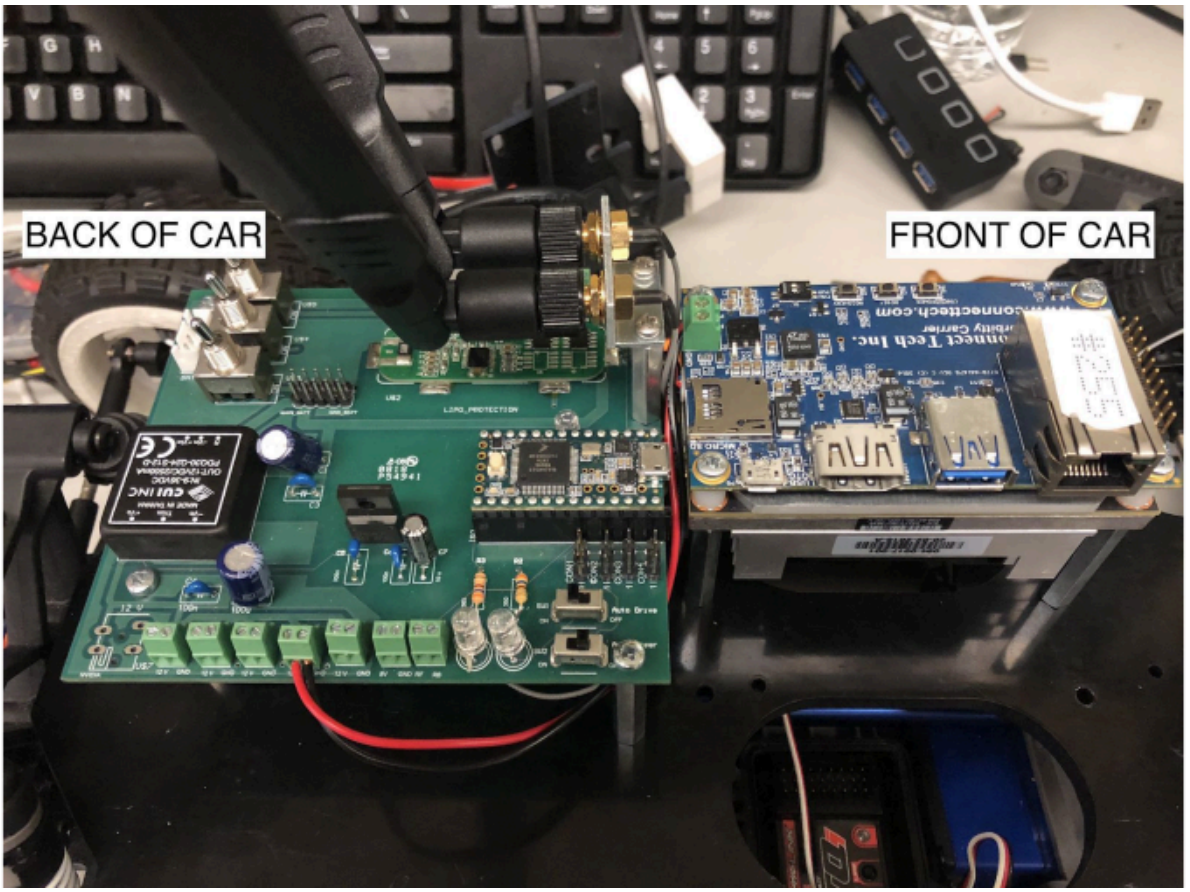
Τοποθέτηση του Jetson στο πλαίσιο:

Στο κιτ που έχουμε συνοδεύεται από τέσσερις λευκές πλαστικές βάσεις. Τοποθετώντας τα προσέχτηκα μεταξύ του Jetson PCB και της ψύκτρας (βλ. εικόνα) πριν να βιδώσουμε τις βίδες. Διαφορετικά, κινδυνεύουμε να λυγίσουμε την πλακετα της τροχιάς ενώ το βιδώνουμε. Χρησιμοποιώντας τις βίδες 20 mm για να στερεωσουμε το Jetson στις βάσεις του πλαισίου του:



Σχ. 2.7

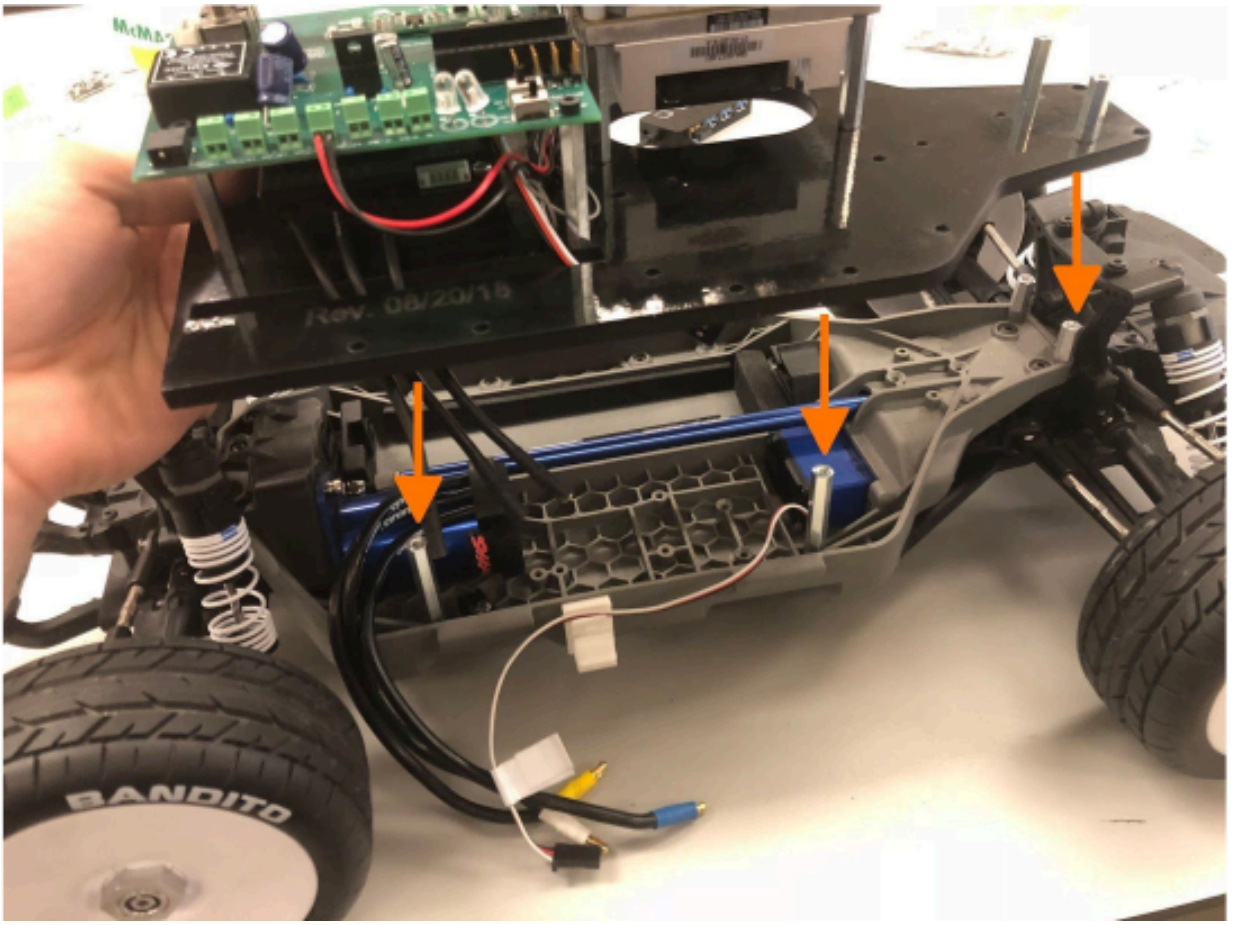
Θα πρέπει να βεβαιωθούμε ότι όταν βάζουμε το Jetson στην πλακέτα δεν τσιμπάνε τα καλώδια ούτε για την κεραία Wi-Fi ούτε για τις συνδέσεις τροφοδοσίας του Jetson. Μπορεί να μας βοηθήσει αν κολλήσουμε και τα δύο σετ καλωδίων κάτω από την πλακέτα τροφοδοσίας ή αν βάλουμε κάτι να τα συγκρατεί όπως παραδείγματος χάριν μικρό δεματικό. (Μην τα βάζετε κάτω από το Jetson γιατί μπορεί να περιορίσουν τη ροή του αέρα ή να εμποδίσουν τα πτερύγια του ανεμιστήρα.) Η διαμόρφωσή σας θα πρέπει τώρα να μοιάζει κάπως έτσι:



Σχ. 2.8

Τοποθέτηση του πλαισίου στο κάτω αμάξωμα του αυτοκινήτου:

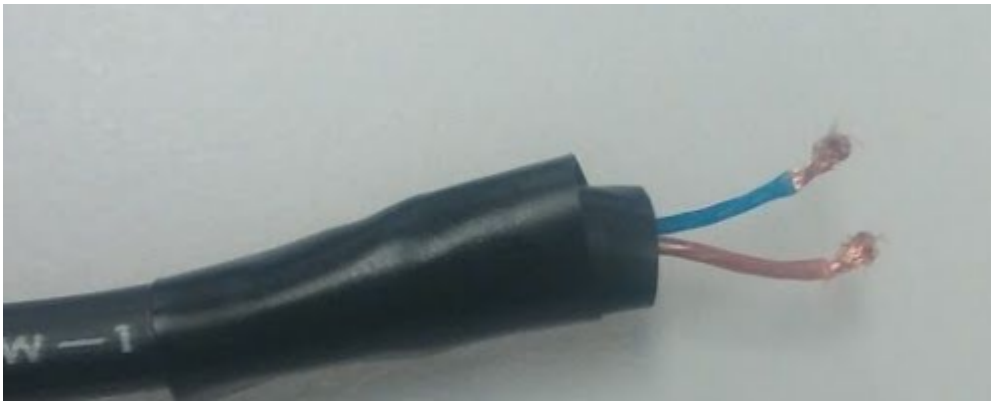
Θα πρέπει να τοποθετήσουμε το σασί στις πέντε βάσεις στη βάση του αυτοκινήτου και ευθυγραμμίσουμε τις οπές διάτρησης του πλαισίου με τις βάσης του αυτοκινήτου που συνδέσαμε νωρίτερα, όπως φαίνεται παρακάτω. Εδώ θα χρησιμοποιήσουμε έξι βίδες M3 των 10 mm για να στερεωσουμε το πλαίσιο στις βάσεις:



Σχ. 2.9

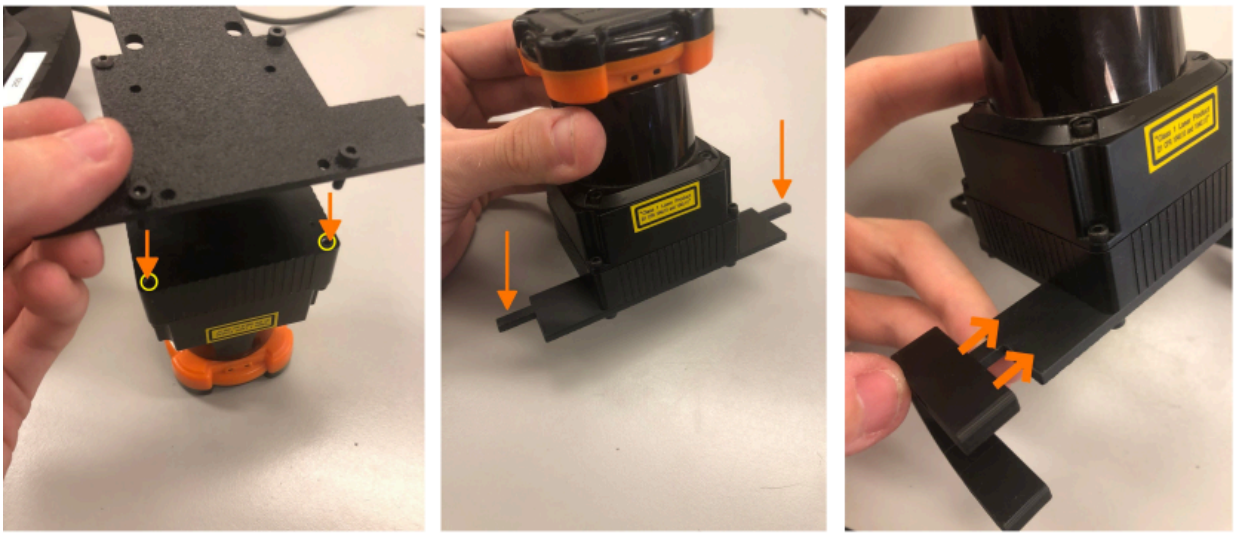
Τοποθέτηση του LIDAR:

Στο LIDAR θα πρέπει να έχουμε δύο καλώδια: ένα για τροφοδοσία και ένα είτε Ethernet είτε για USB. Κόβουμε το άκρο του καλωδίου τροφοδοσίας, αφήνοντας 1-2 χιλ. καλώδιο. Απογυμνώνουμε το άκρο, κόβουμε όλα τα καλώδια εκτός από τα μπλε και καφέ και απογυμνώνουμε αυτά τα δύο καλώδια σε 3 χιλ. όπως φαίνεται παρακάτω:



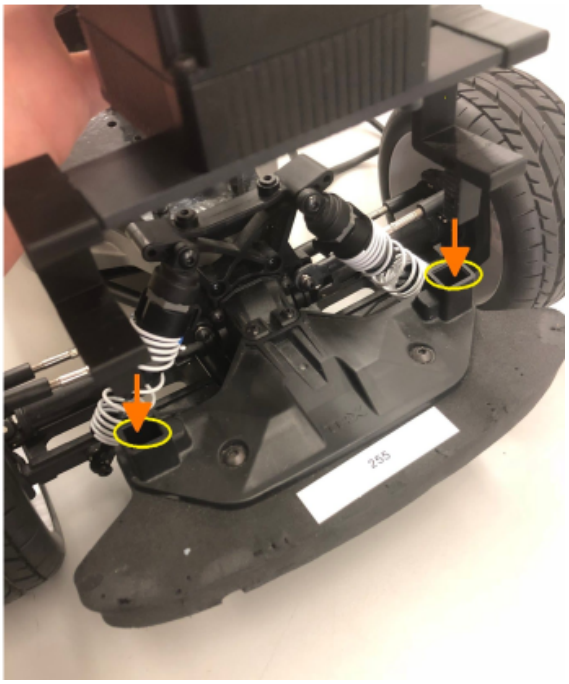
Σχ. 3.0

Θα στερεωσουμε το LIDAR στη βάση σε σχήμα δέντρου χρησιμοποιώντας δύο (10LX) ή τέσσερις (30LX) βίδες, έτσι ώστε τα καλώδια που προεξέχουν από το LIDAR να πηγαίνουν προς το πίσω μέρος του αυτοκινήτου. (Για το 30LX, οι δύο λυχνίες LED στο πάνω μέρος του LIDAR θα πρέπει να κοιτούν προς το μπροστινό μέρος του αυτοκινήτου.) Στη συνέχεια, τοποθετούμε τα στηρίγματα σε σχήμα C στη μαύρη βάση σε σχήμα δέντρου, όπως φαίνεται στις παρακάτω εικόνες. Σημείωση: εάν είναι μαύρο. Η βάση δεν χωράει στις οπές των στηριγμάτων C, τρίψτε τα ένθετα μέχρι να χωρέσουν ακόλουθη η εικόνα παρακάτω:



Σχ. 3.1

Τοποθετούμε και το LIDAR στο αυτοκίνητο τοποθετώντας τα δύο χοντρά προεξέχοντα μέρη των βραχιόνων στήριξης στις οπές. Το ανοιχτό μέρος του "C" στις αγκύλες πρέπει να κοιτάζει προς τα εμπρός όπως φαίνεται παρακάτω.



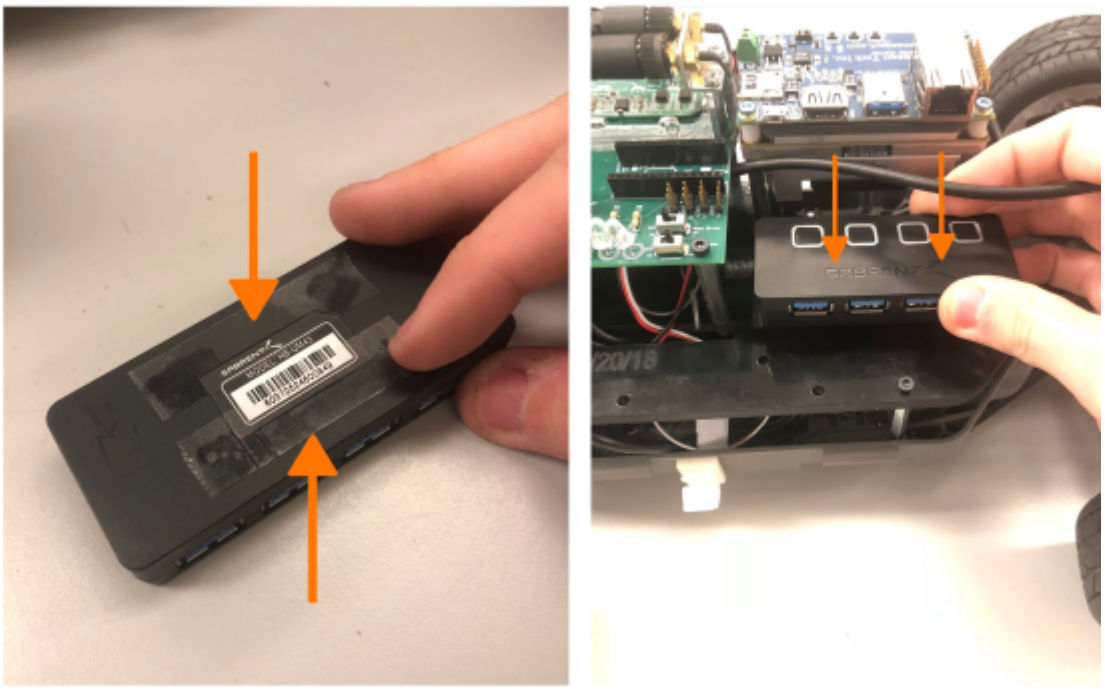
Σχ. 3.2

Εάν τα κάναμε όλα σωστά μέχρι εδώ τότε , δύο από τις οπές στο στενό άκρο της βάσης LIDAR θα πρέπει να ταιριάζουν με τις δύο βάσεις 19 mm που βάλαμε νωρίτερα. Τυλίγουμε τα καλώδια τροφοδοσίας και USB του LIDAR γύρω από τη βάση του, ώστε να υπάρχει αρκετό διαθέσιμο για να συνδέσουμε την πλακέτα τροφοδοσίας και το Jetson, με λίγη χαλαρότητα, ώστε να μην είναι πολύ σφιχτό. Θα περάσουμε και τα δύο κορδόνια κάτω από την πλάκα στήριξης LIDAR ανάμεσα στις δύο ασημένιες βάσεις και στερεώνουμε τη βάση LIDAR στις βάσεις του πλαισίου χρησιμοποιώντας δύο βίδες M3 10 mm:

Σχ. 3.3

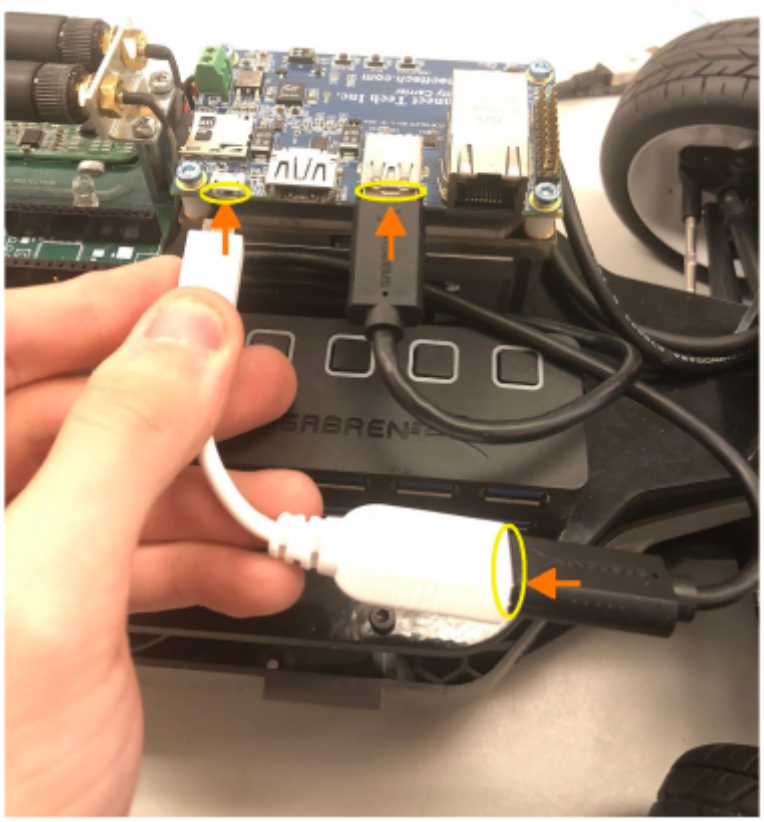
Τοποθέτηση των διανομέων USB:

Θα πρέπει να προσθέσουμε δύο κομμάτια ταινίας διπλής όψης σε έναν διανομέα USB. Μετά τοποθετούμε την πλήμνη στον κενό χώρο του πλαισίου ακριβώς κάτω από το Jetson και στα δεξιά της πλακέτας ισχύος, όπως φαίνεται. Εάν ο διανομέας σας διαθέτει κουμπιά λειτουργίας, θα πρέπει να βεβαιωθούμε ότι είναι όλα ενεργοποιημένα:



Σχ. 3.4

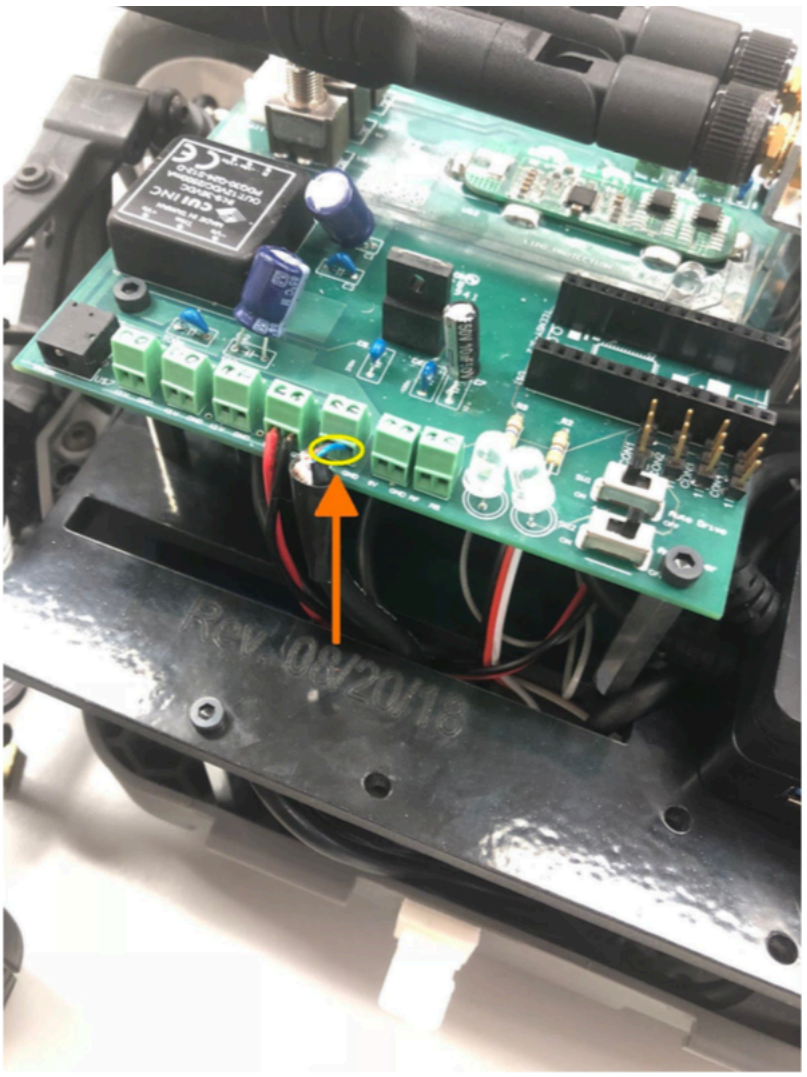
Εάν θα χρειαστούμε περισσότερες θύρες USB (απαιτείται εάν το LiDAR σας χρησιμοποιεί USB), μπορούμε να στοιβαξουμε τον ένα διανομέα πάνω στον άλλον. Και πάλι, θα πρέπει να χρησιμοποιήσουμε ταινία διπλής όψης για να ασφαλίσουμε το δεύτερο κέντρο και θα πρέπει να βεβαιωθούμε ότι όλα τα κουμπιά λειτουργίας είναι ενεργοποιημένα. Εφόσον γίνουν όλα τα παραπάνω τότε συνδέουμε τον πρώτο διανομέα USB στην πλακέτα τροχιάς. Εάν χρησιμοποιήσουμε δεύτερο διανομέα, θα πρέπει να χρησιμοποιήσουμε τον λευκό προσαρμογέα micro USB που συνοδεύει το Jetson για να συνδέσουμε τον δεύτερο διανομέα.



Σχ. 3.5

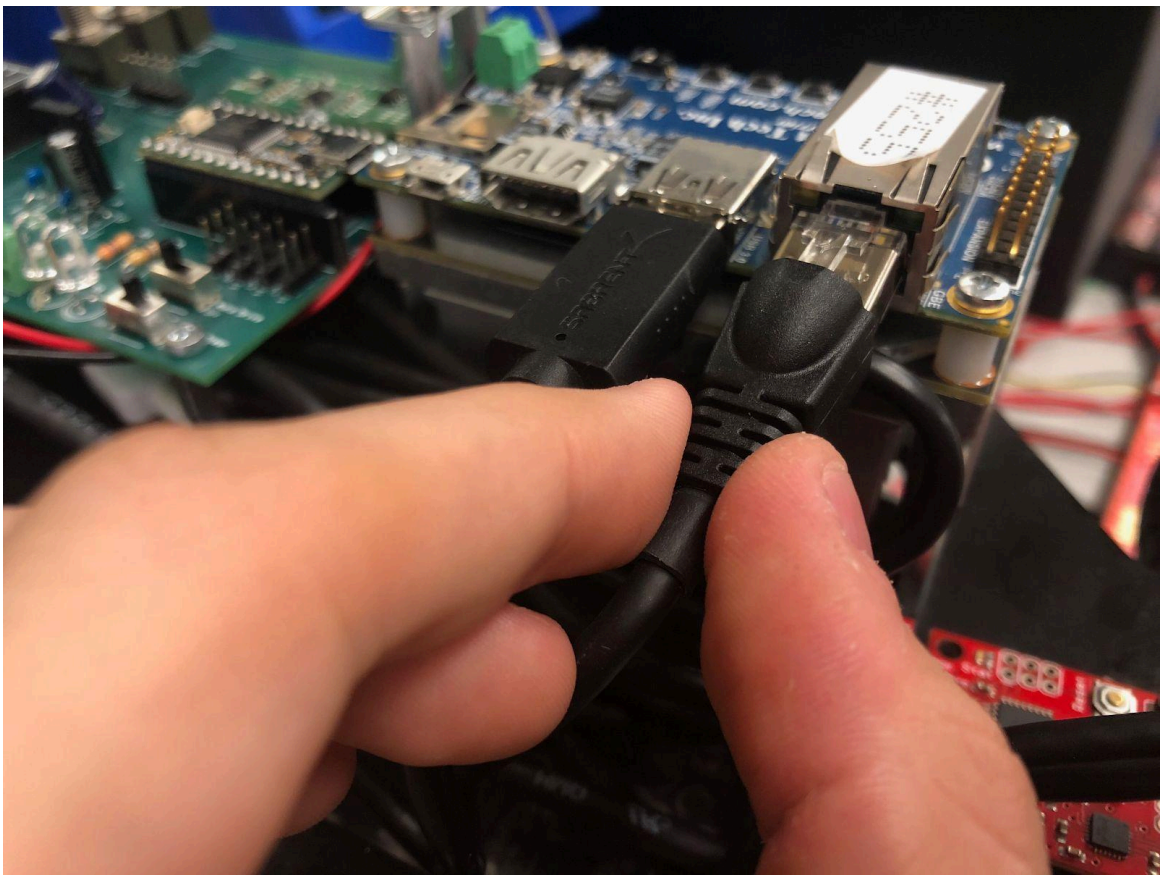
Σύνδεση του LIDAR:

Τώρα θα συνδέσουμε το καλώδιο τροφοδοσίας του LIDAR σε ένα ελεύθερο μπλοκ ακροδεκτών 12 V στην πλακέτα τροφοδοσίας. Το καφέ καλώδιο πρέπει να πάει στον ακροδέκτη 12 V και το μπλε καλώδιο πρέπει να πάει στον αντίστοιχο ακροδέκτη GND. Η πλευρά του LIDAR έχει και pinout .



Σχ. 3.6

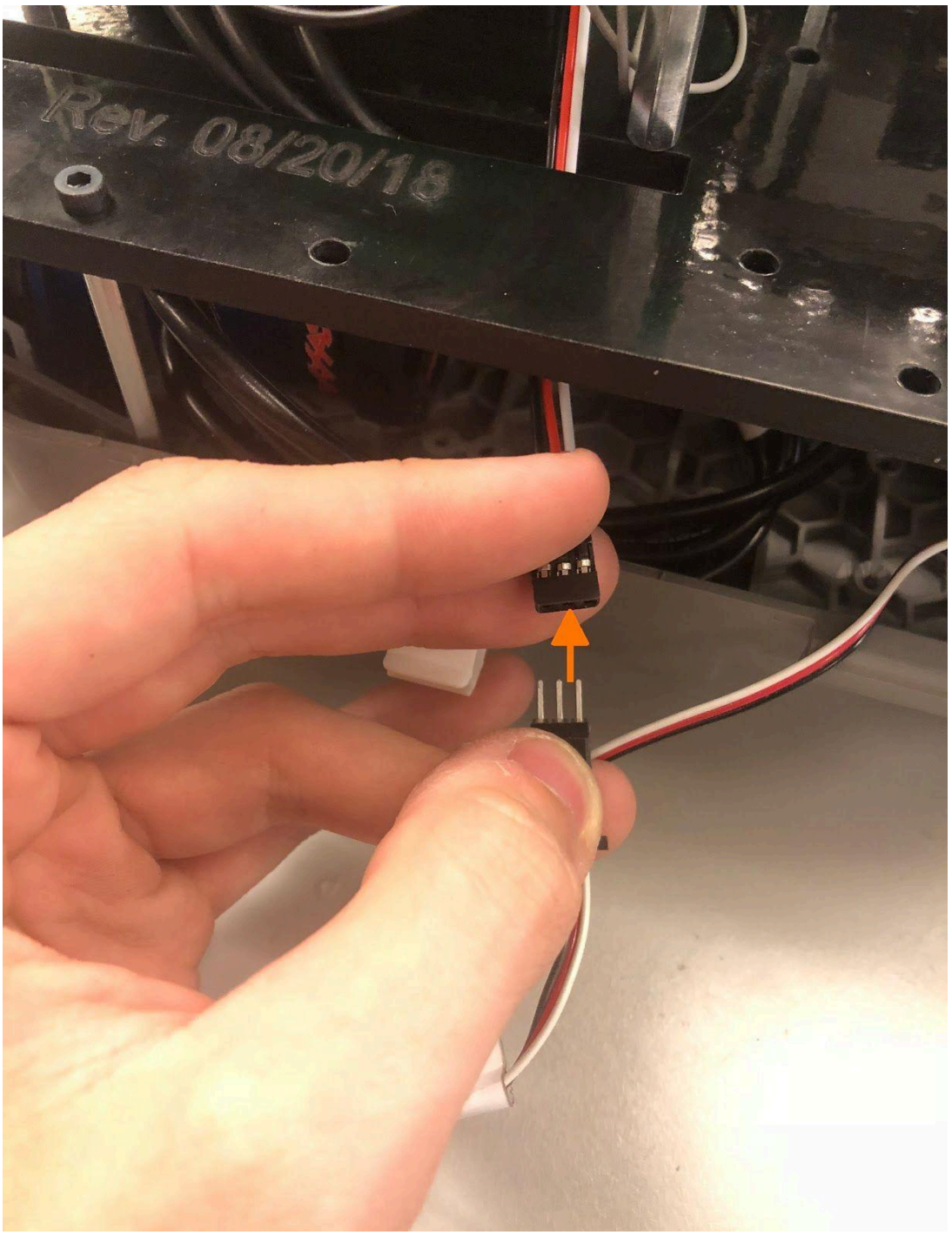
Εάν το LIDAR διαθέτει καλώδιο Ethernet, τοτε συνδέουμε το συγκεκριμένο στη θύρα Ethernet του Jetson. Εάν έχει καλώδιο USB, τότε συνδέουμε το συγκεκριμένο στον διανομέα USB. Θα περάσουμε τα καλώδια που περισσεύουν πίσω από τους διανομείς USB όπως φαίνεται στην εικόνα.



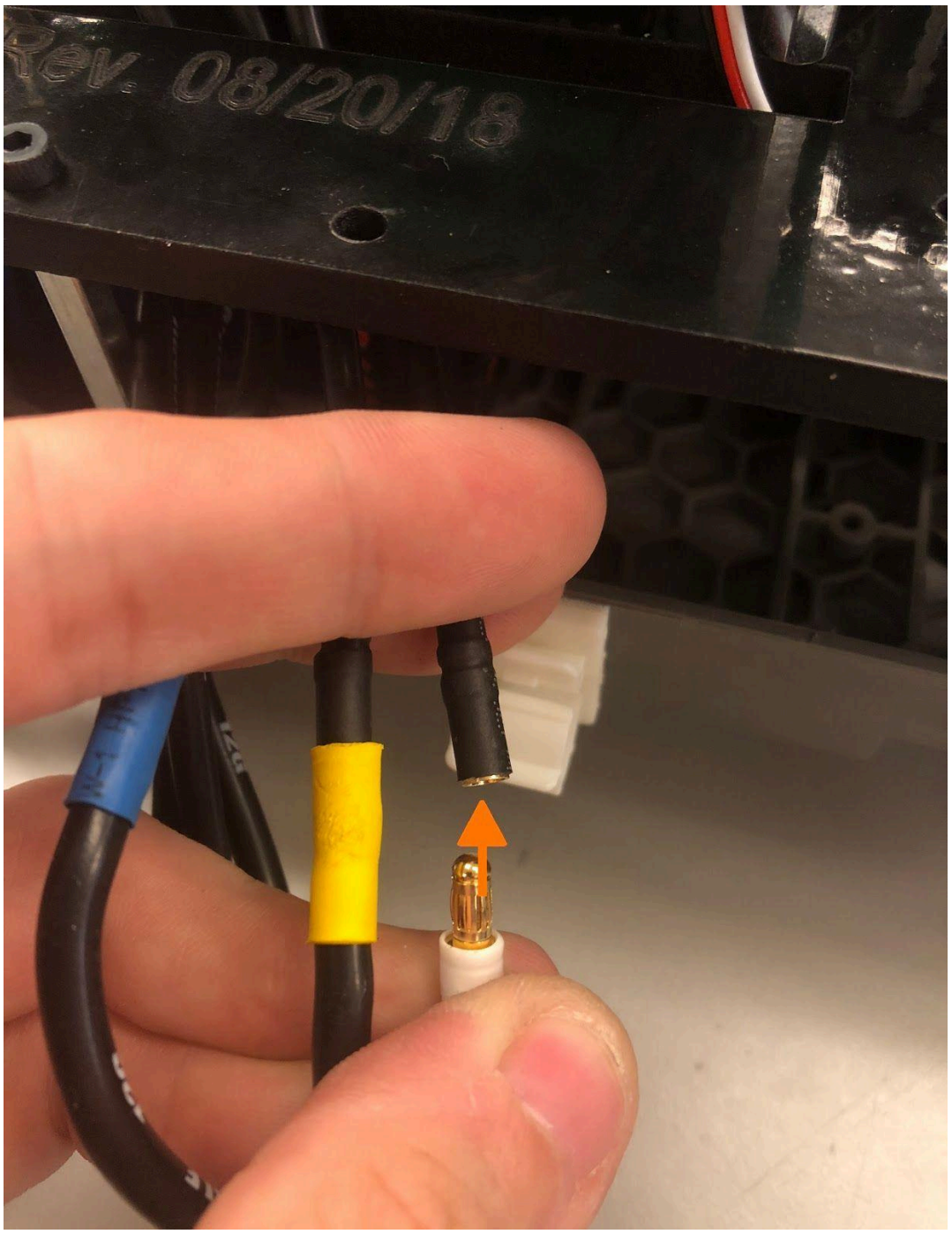
Σχ. 3.7

Σύνδεση του FOCbox:

Θα πρέπει να περάσουμε τα 3 στρογγυλά καλώδια FOCbox μέσα από την ορθογώνια σχισμή στο πλαστικό πλαίσιο και, στη συνέχεια, συνδέουμε τους 3 κυκλικούς συνδέσμους σφαιρών στα τρία καλώδια του κινητήρα. (Η σειρά με την οποία συνδέουμε τα καλώδια δεν έχει καμία σημασία (ηλεκτρικά μιλώντας). Εάν τρέχετε το αυτοκίνητο και πηγαίνει προς τα πίσω όταν πρέπει να πάει μπροστά, μπορούμε φυσικά να αλλάξουμε οποιαδήποτε από τα τρία καλώδια.) Συνδέοντας τα 3 καλώδια καλώδιο σέρβο κορδέλας επίσης, φροντίζοντας τα χρώματα να ταιριάζουν.

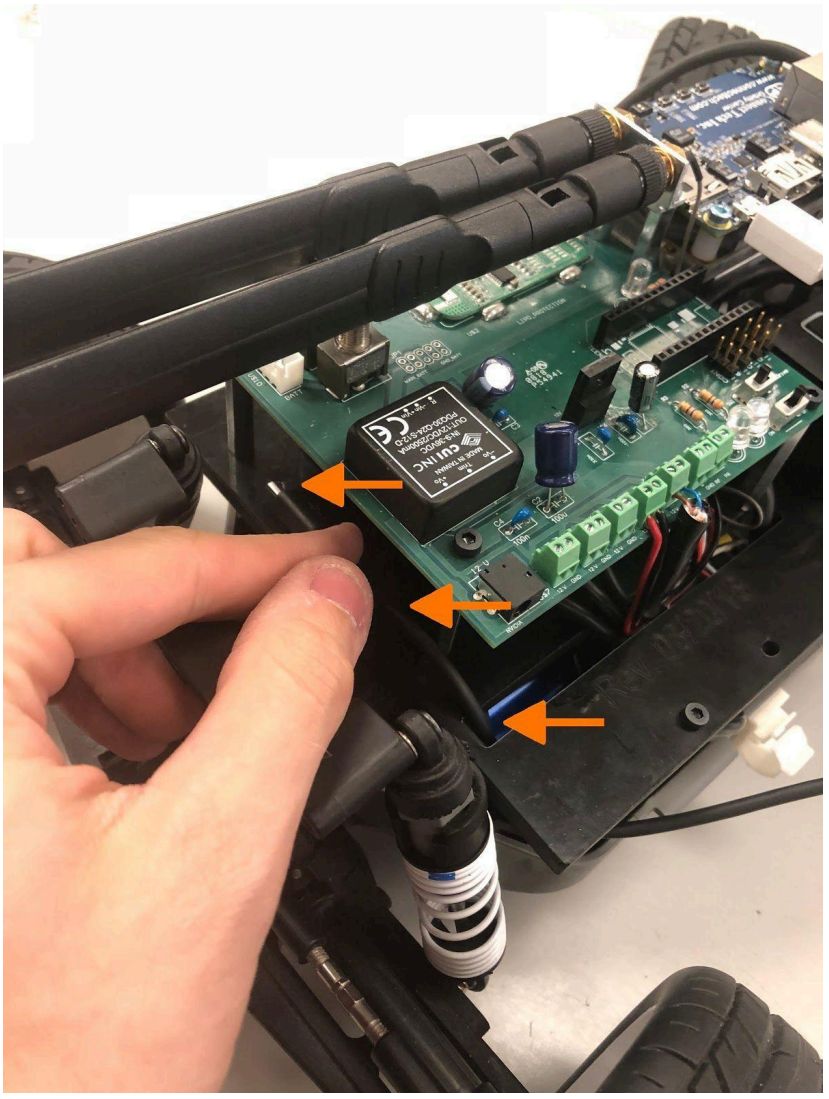


Σχ. 3.8

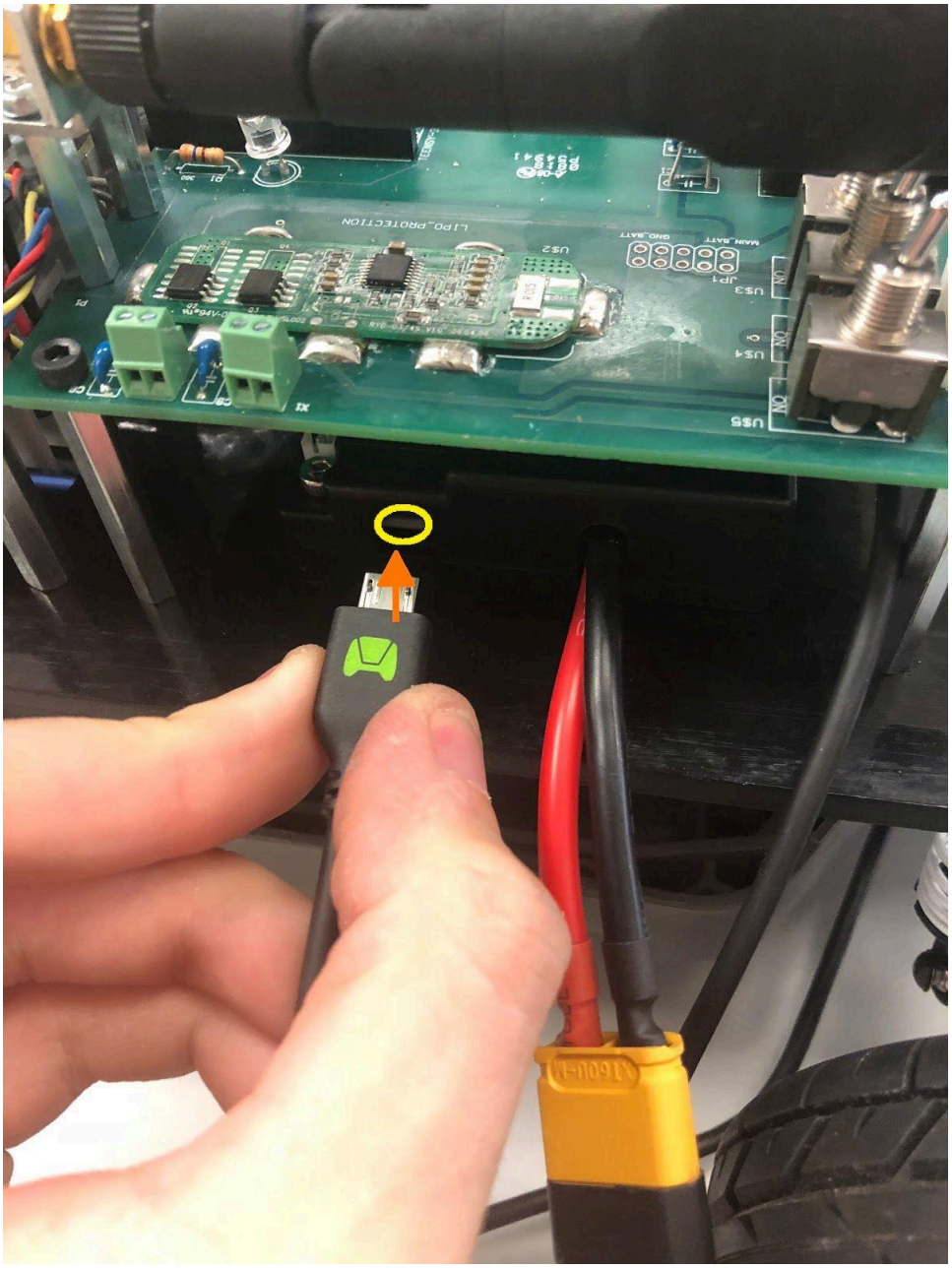


Σχ. 3.9

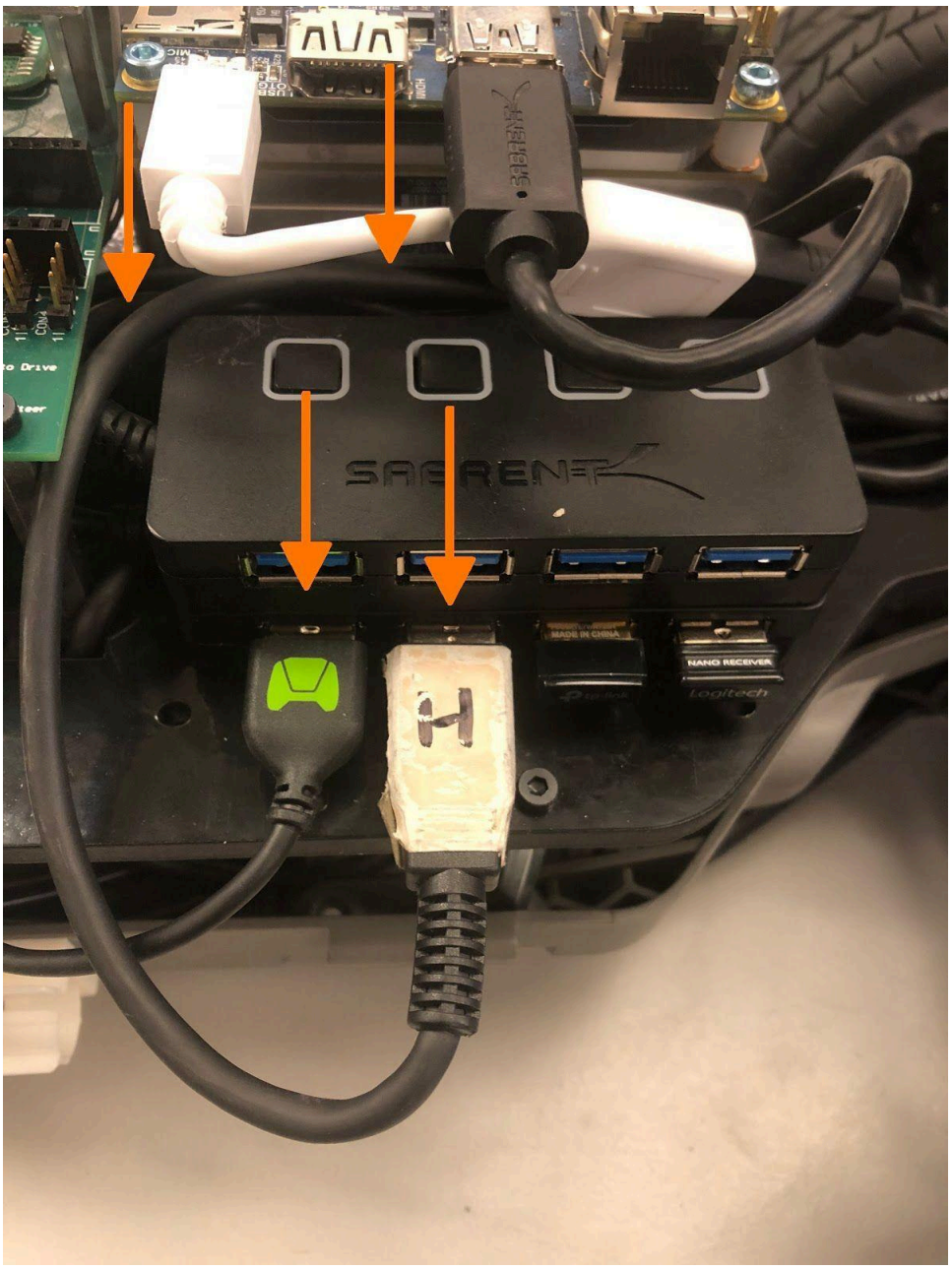
Εάν το καλώδιο micro USB είναι αρκετά λεπτό, θα πρέπει να περάσουμε το μέσα από την ορθογώνια υποδοχή καλωδίου και γύρω από το FOCbox στην υποδοχή USB επίσης φαίνεται παρακάτω ή θα πρέπει να το περάσουμε το γύρω από το πίσω άκρο του πλαισίου, εάν δεν είναι. Συνδέουμε το καλώδιο στην υποδοχή USB του FOCbox και σε μια ελεύθερη θύρα στον διανομέα USB. Θα πρέπει να δέσουμε το καλώδιο USB χρησιμοποιώντας ένα δέσιμο καλωδίου και τυλίγουμε όλα τα καλώδια κάτω από το πλαίσιο. Μπορούμε επίσης να χρησιμοποιήσουμε αυτήν τη φορά για να συνδέσουμε το LIDAR (αν είναι USB), τον εξωτερικό προσαρμογέα Wi-FI και τον δέκτη για το gamepad F1/10.



Σχ. 4.0



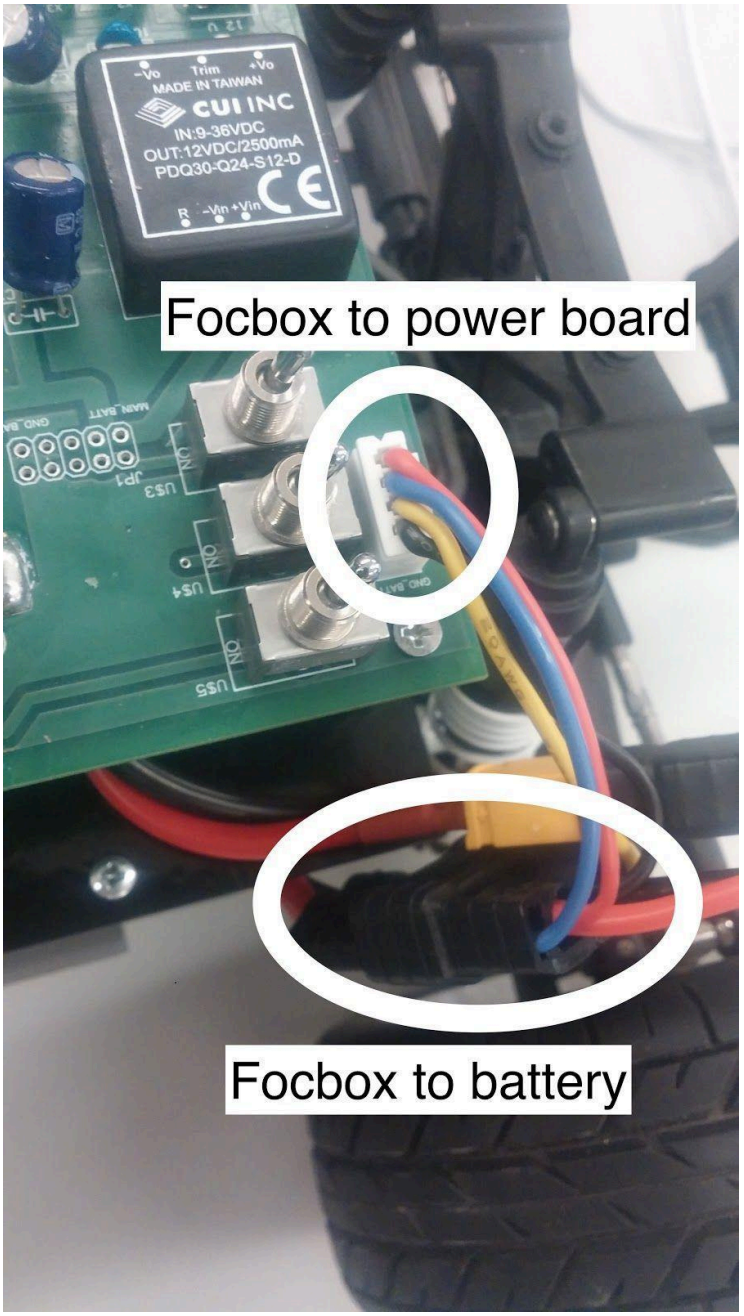
Σχ. 4.1



Σχ. 4.2

Σύνδεση του αυτοκινήτου με την μπαταρία:

Τέλος συνδέουμε την μπαταρία στο FOCbox χρησιμοποιώντας την υποδοχή της μπαταρίας. Σε αυτό το σημείο θα πρέπει να βεβαιωθούμε ότι το κόκκινο είναι ευθυγραμμισμένο με το κόκκινο και το μαύρο με το μαύρο διαφορετικά τα πράγματα θα γίνουν καυτά και δυσάρεστα. Στη συνέχεια συνδέουμε το FOCbox στην πλακέτα τροφοδοσίας χρησιμοποιώντας τη λευκή θύρα που φαίνεται στην παρακάτω εικόνα.



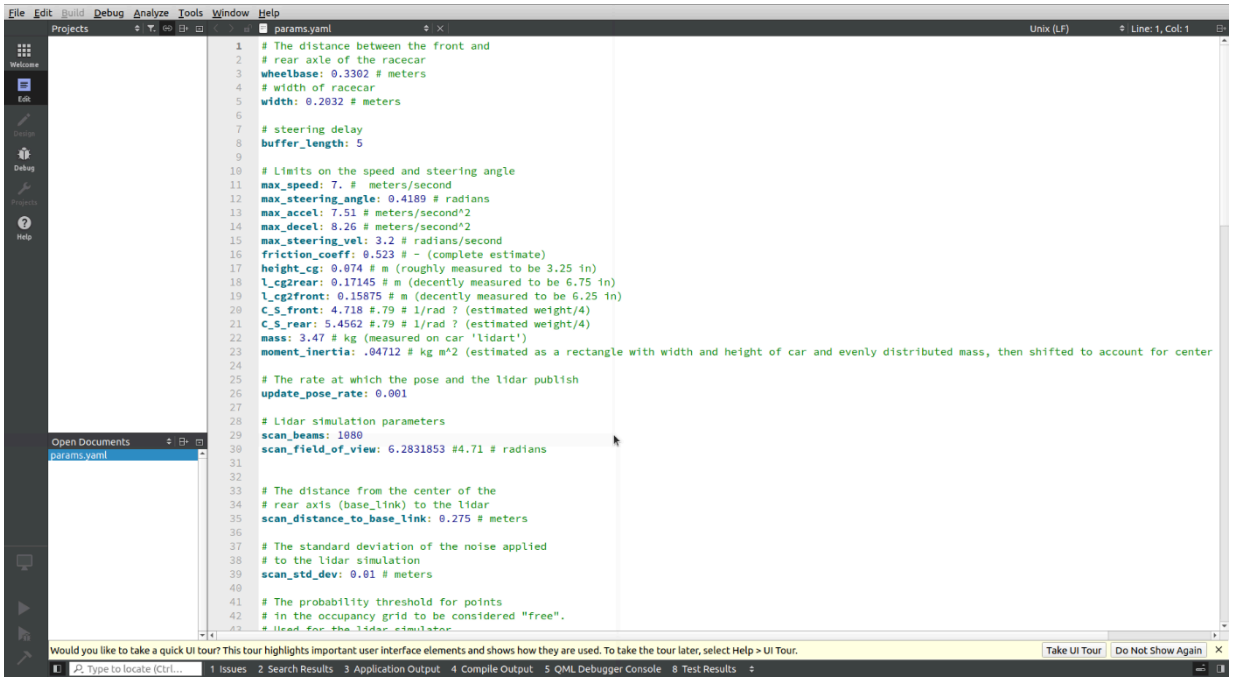
Σχ. 4.3

Σε αυτό το σημείο, το αυτοκίνητό μας θα πρέπει να συναρμολογηθουν, τα φώτα Jetson θα πρέπει να αναβοσβήνουν όταν γυρναμε τον διακόπτη του ρεύματος και το αυτοκίνητο είναι έτοιμο για λειτουργία!Σημείωση: Προς το παρόν, δεν υπάρχει τρόπος να γνωρίζουμε το επίπεδο φόρτισης της μπαταρίας εκτός από τη μέτρησή της με ένα πολύμετρο καθώς λειτουργεί. Σε αυτό το σημείο μπορούμε να βάλουμε είτε οθόνη LED χαμηλής τάσης είτε οθόνη LCD επτά τμημάτων (για να εμφανίσουμε την τάση).

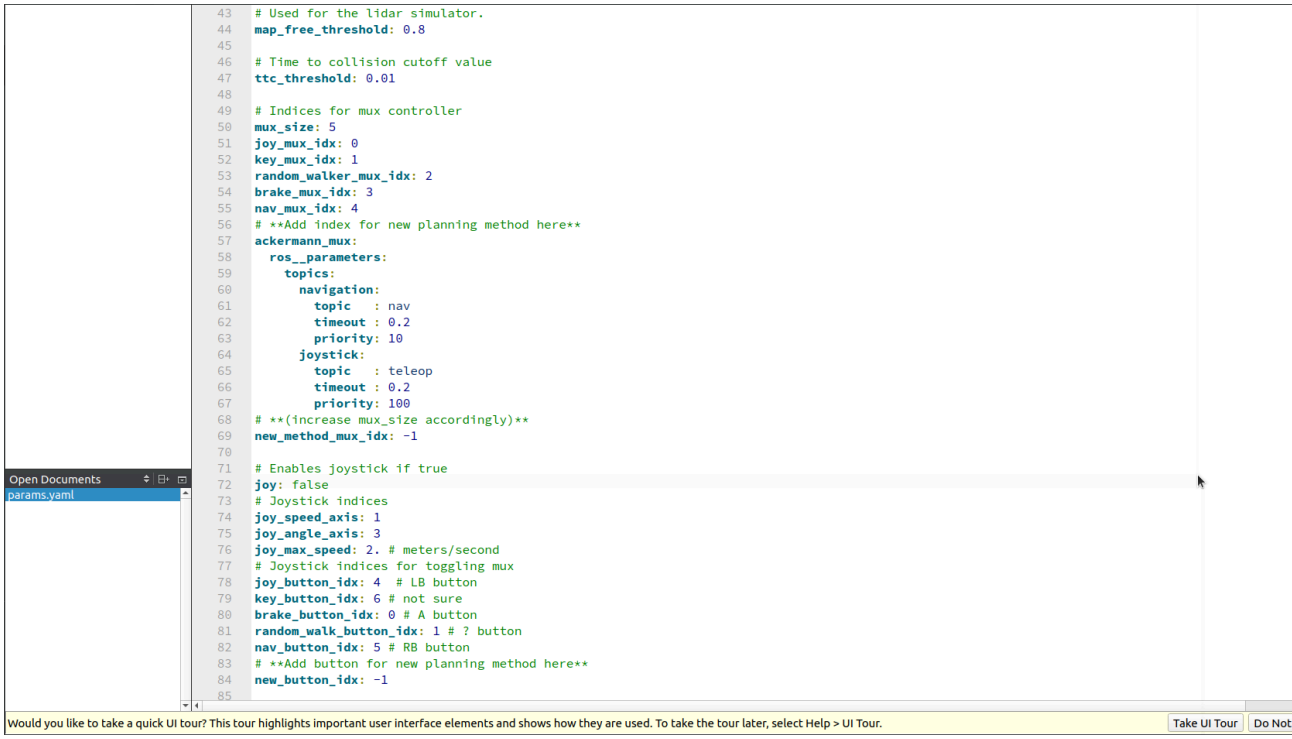
3 Κεφάλαιο : Simulation

3.1 Simulation

Αρχικά έφτιαξα το πρόγραμμα για το RVIZ το συγκεκριμένο πρόγραμμα ήθελε μετατροπές για να λειτουργήσει με την βοήθεια του Google. Το πρόγραμμα είχε 135 σειρές κώδικα και λέγεται parames.yaml. Στην αρχή θέλαμε να το κάνουμε με να πληκτρολογούμε εμείς της συντεταγμένες αλλά αυτό δεν γινόταν και έτσι φτιάξαμε ένα άλλο αρχείο που ουσιαστικά με την βοήθεια του PID μπορεί το όχημα μας να κινείται στον χώρο αυτόματα. Μετά με διαφορά τεστ καταφέραμε και βρήκαμε το κατάλληλο PID που κάνει μια ομαλή και σχετικά γρήγορη προσέγγιση στον χώρο. Το όχημα σκανάρει συνέχεια τον χώρο και στην συνέχεια υπολογίζει πότε πρέπει να αποφύγει το αντικείμενο αν δεν μπορεί τότε τρακάρει και σταματάει. Παρακάτω έχω διάφορες φωτογραφίες που έβγαλα από τον προγραμματισμό έβγαλα μόνο τα απαραίτητα και φυσικά έβγαλα και το PID : εδώ να σημειωθεί ότι υπάρχουν συνολικά 5 αρχεία για το συγκεκριμένο ένα για το κοντρόλ του δυο για ποιος θα κινηθεί στην μέση η όχι του δρόμου και ένα για το που θα πάει και ένα για να σκανάρει το περιβάλλον.



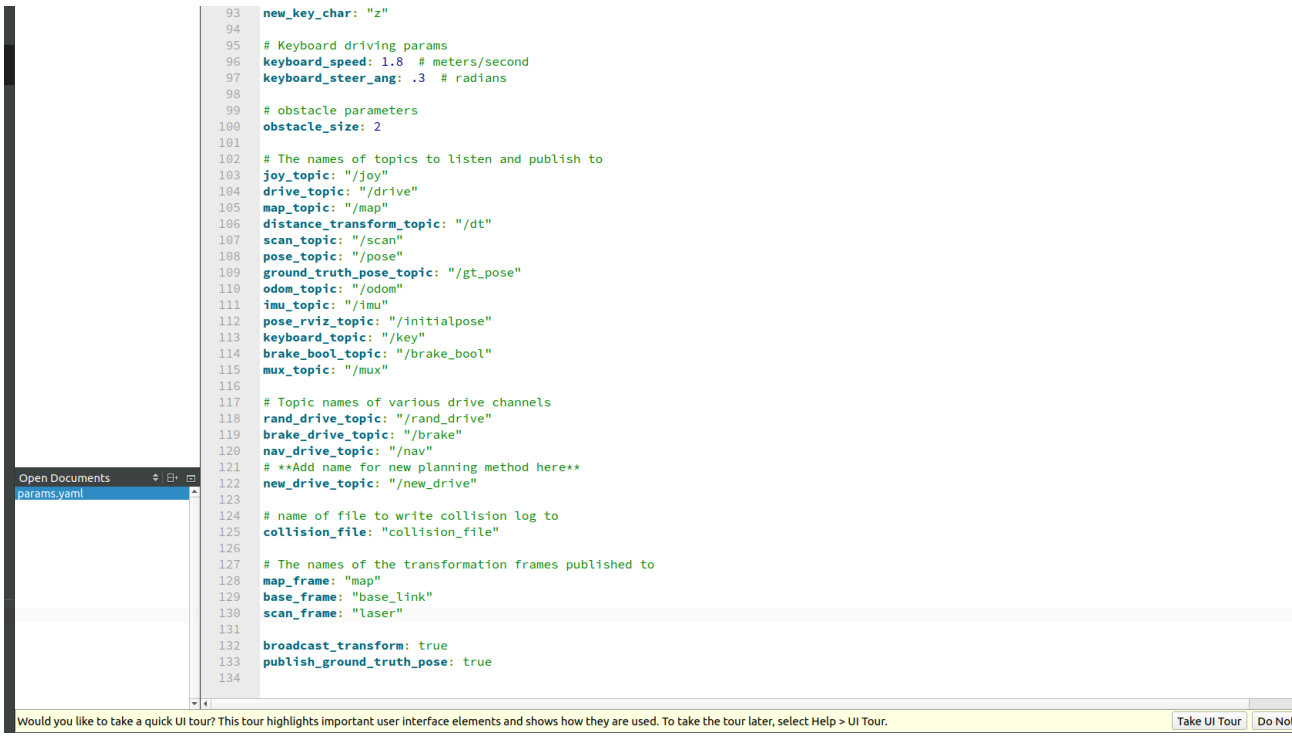
Σχ. 4.4 (οι παράμετροι του αμαξιού)


Open Documents
params.yaml
Used for the lidar simulator.
map_free_threshold: 0.8
Time to collision cutoff value
ttc_threshold: 0.01
Indices for mux controller
mux_size: 5
joy_mux_idx: 0
key_mux_idx: 1
random_walker_mux_idx: 2
brake_mux_idx: 3
nav_mux_idx: 4
Add index for new planning method here
ackermann_mux:
ros__parameters:
topics:
navigation:
topic : nav
timeout : 0.2
priority: 10
joystick:
topic : teleop
timeout : 0.2
priority: 100
(increase mux_size accordingly)
new_method_mux_idx: -1
Enables joystick if true
joy: false
Joystick indices
joy_speed_axis: 1
joy_angle_axis: 3
joy_max_speed: 2. # meters/second
Joystick indices for toggling mux
joy_button_idx: 4 # LB button
key_button_idx: 6 # not sure
brake_button_idx: 0 # A button
random_walk_button_idx: 1 # ? button
nav_button_idx: 5 # RB button
Add button for new planning method here
new_button_idx: -1
Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour.
Take UI Tour
Do Not

Σχ. 4.5

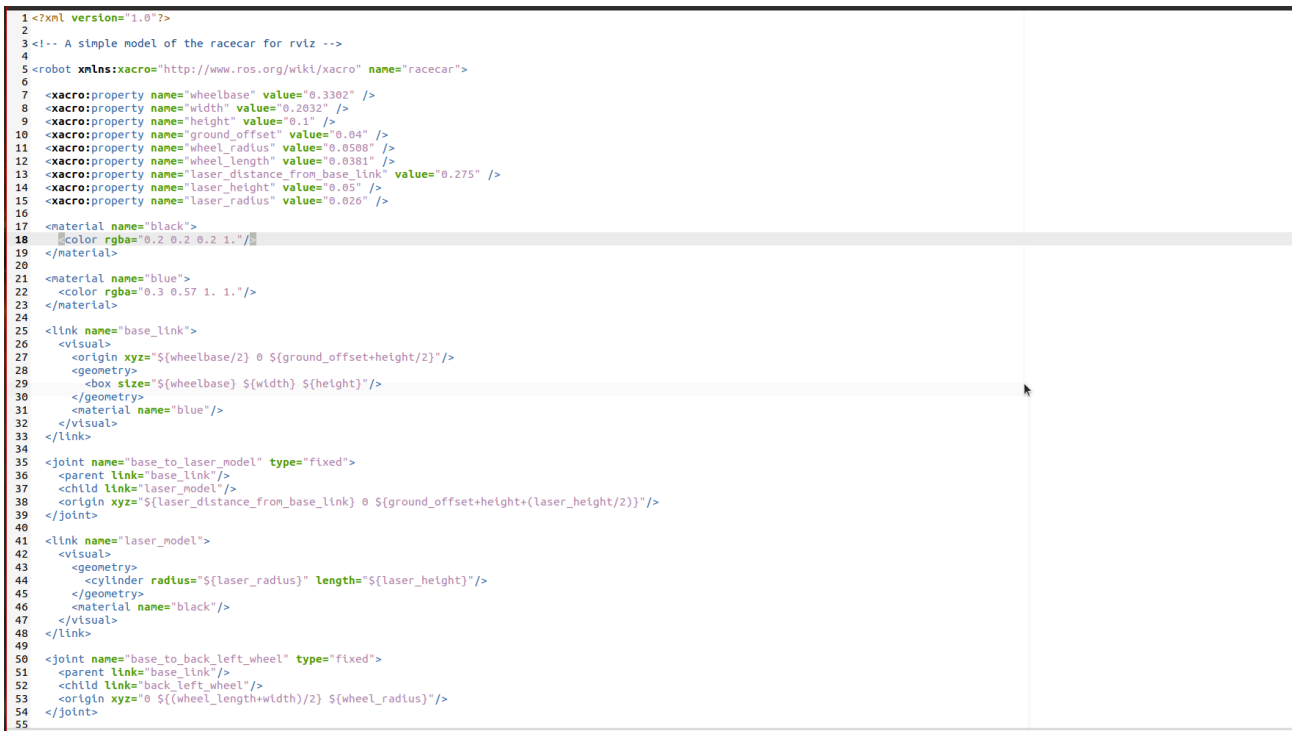

Open Documents
params.yaml
Keyboard characters for toggling mux
joy_key_char: "j"
keyboard_key_char: "k"
brake_key_char: "b"
random_walk_key_char: "r"
nav_key_char: "n"
Add button for new planning method here
new_key_char: "z"
Keyboard driving params
keyboard_speed: 1.8 # meters/second
keyboard_steer_ang: .3 # radians
obstacle parameters
obstacle_size: 2
The names of topics to listen and publish to
joy_topic: "/joy"
drive_topic: "/drive"
map_topic: "/map"
distance_transform_topic: "/dt"
scan_topic: "/scan"
pose_topic: "/pose"
ground_truth_pose_topic: "/gt_pose"
odom_topic: "/odom"
imu_topic: "/imu"
pose_rviz_topic: "/initialpose"
keyboard_topic: "/key"
brake_bool_topic: "/brake_bool"
mux_topic: "/mux"
Topic names of various drive channels
rand_drive_topic: "/rand_drive"
brake_drive_topic: "/brake"
nav_drive_topic: "/nav"
Add name for new planning method here
new_drive_topic: "/new_drive"
name of file to write collision log to
collision_file: "collision_file"
The names of the transformation frames published to
Would you like to take a quick UI tour? This tour highlights important user interface elements and shows how they are used. To take the tour later, select Help > UI Tour.
Take UI Tour
Do Not Show Ag

Σχ. 4.6



Σχ. 4.7

Στις προηγούμενες φωτογραφίες δείχνω τον κώδικα για όλο το simulation σε αυτό το φάκελο θα μπορούμε να αλλάξουμε την ταχύτητα του αμαξιού την πιστά το σκανάρισμα στο σημείο αυτό να σας ενημερώσω ότι το συγκεκριμένο simulation μπορούμε να το χειριστούμε και με το πληκτρολόγιο απλά είναι το θέμα δεν θα έχουμε το PID. Αρά σε αυτό τον αρχείο μπορούμε να αλλάξουμε ότι θέλουμε .



```
<?xml version="1.0"?>

<!-- A simple model of the racecar for rviz -->

<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="racecar">

  <xacro:property name="wheelbase" value="0.3302" />
  <xacro:property name="width" value="0.2032" />
  <xacro:property name="height" value="0.1" />
  <xacro:property name="ground_offset" value="0.04" />
  <xacro:property name="wheel_radius" value="0.0508" />
  <xacro:property name="wheel_length" value="0.0381" />
  <xacro:property name="laser_distance_from_base_link" value="0.275" />
  <xacro:property name="laser_height" value="0.05" />
  <xacro:property name="laser_radius" value="0.026" />

  <material name="black">
    <color rgba="0.2 0.2 0.2 1."/>
  </material>

  <material name="blue">
    <color rgba="0.3 0.57 1. 1."/>
  </material>

  <link name="base_link">
    <visual>
      <origin xyz="${wheelbase/2} 0 ${ground_offset+height/2}"/>
      <geometry>
        <box size="${wheelbase} ${width} ${height}"/>
      </geometry>
      <material name="blue"/>
    </visual>
  </link>

  <joint name="base_to_laser_model" type="fixed">
    <parent link="base_link"/>
    <child link="laser_model"/>
    <origin xyz="${laser_distance_from_base_link} 0 ${ground_offset+height+(laser_height/2)}"/>
  </joint>

  <link name="laser_model">
    <visual>
      <geometry>
        <cylinder radius="${laser_radius}" length="${laser_height}"/>
      </geometry>
      <material name="black"/>
    </visual>
  </link>

  <joint name="base_to_back_left_wheel" type="fixed">
    <parent link="base_link"/>
    <child link="back_left_wheel"/>
    <origin xyz="0 ${(wheel_length+width)/2} ${wheel_radius}"/>
  </joint>

```

Σχ. 4.8 (εδώ είναι το αρχείο με το σκανάρισμα)

```
  <link name="back_left_wheel">
    <visual>
      <geometry>
        <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
      </geometry>
      <material name="black"/>
      <origin rpy="${pi/2} 0 0"/>
    </visual>
  </link>

  <joint name="base_to_back_right_wheel" type="fixed">
    <parent link="base_link"/>
    <child link="back_right_wheel"/>
    <origin xyz="0 ${-(wheel_length+width)/2} ${wheel_radius}"/>
  </joint>

  <link name="back_right_wheel">
    <visual>
      <geometry>
        <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
      </geometry>
      <material name="black"/>
      <origin rpy="${pi/2} 0 0"/>
    </visual>
  </link>

  <joint name="base_to_front_left_hinge" type="fixed">
    <parent link="base_link"/>
    <child link="front_left_hinge"/>
    <origin xyz="${wheelbase} ${(wheel_length+width)/2} ${wheel_radius}"/>
  </joint>

  <link name="front_left_hinge"/>

  <joint name="front_left_hinge_to_wheel" type="continuous">
    <parent link="front_left_hinge"/>
    <child link="front_left_wheel"/>
  </joint>

  <link name="front_left_wheel">
    <visual>
      <geometry>
        <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
      </geometry>
      <material name="black"/>
      <origin rpy="${pi/2} 0 0"/>
    </visual>
  </link>

  <joint name="base_to_front_right_hinge" type="fixed">
    <parent link="base_link"/>
    <child link="front_right_hinge"/>
    <origin xyz="${wheelbase} ${-(wheel_length+width)/2} ${wheel_radius}"/>
  </joint>

```

Σχ. 4.9

,

```
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
    </geometry>
    <material name="black"/>
    <origin rpy="${pi/2} 0 0"/>
  </visual>
</link>

<joint name="base_to_front_left_hinge" type="fixed">
  <parent link="base_link"/>
  <child link="front_left_hinge"/>
  <origin xyz="${wheelbase} ${(wheel_length+width)/2} ${wheel_radius}"/>
</joint>

<link name="front_left_hinge"/>

<joint name="front_left_hinge_to_wheel" type="continuous">
  <parent link="front_left_hinge"/>
  <child link="front_left_wheel"/>
</joint>

<link name="front_left_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
    </geometry>
    <material name="black"/>
    <origin rpy="${pi/2} 0 0"/>
  </visual>
</link>

<joint name="base_to_front_right_hinge" type="fixed">
  <parent link="base_link"/>
  <child link="front_right_hinge"/>
  <origin xyz="${wheelbase} ${-(wheel_length+width)/2} ${wheel_radius}"/>
</joint>

<link name="front_right_hinge"/>

<joint name="front_right_hinge_to_wheel" type="continuous">
  <parent link="front_right_hinge"/>
  <child link="front_right_wheel"/>
</joint>

<link name="front_right_wheel">
  <visual>
    <geometry>
      <cylinder radius="${wheel_radius}" length="${wheel_length}"/>
    </geometry>
    <material name="black"/>
    <origin rpy="${pi/2} 0 0"/>
  </visual>
</link>

</robot>
```

Σχ. 5.0

package.xml simulator.launch

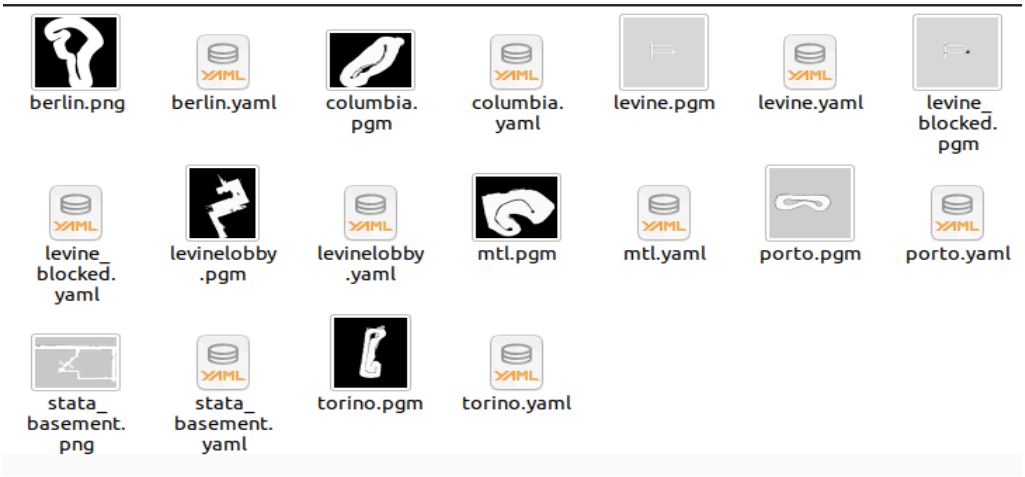
```
<?xml version="1.0"?>
<launch>
  <!-- Listen to messages from joysicks -->
  <node pkg="joy" name="joy_node" type="joy_node"/>

  <!-- Launch a map from the maps folder-->
  <arg name="map" default="$(find f1tenth_simulator)/maps/levine.yaml"/>
  <node pkg="map_server" name="map_server" type="map_server" args="$(arg map)"/>

  <!-- Launch the racecar model -->
  <include file="$(find f1tenth_simulator)/launch/racecar_model.launch"/>

  <!-- Begin the simulator with the parameters from params.yaml -->
  <node pkg="f1tenth_simulator" name="f1tenth_simulator" type="simulator" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

  <!-- Launch the mux node with the parameters from params.yaml -->
  <node pkg="f1tenth_simulator" name="mux_controller" type="mux" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

  <!-- Launch the behavior controller node with the parameters from params.yaml -->
  <node pkg="f1tenth_simulator" name="behavior_controller" type="behavior_controller" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

  <!-- Launch the Random Walker Node -->
  <node pkg="f1tenth_simulator" name="random_walker" type="random_walk" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

  <!-- Launch the Keyboard Node -->
  <node pkg="f1tenth_simulator" name="keyboard" type="keyboard" output="screen">
    <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
  </node>

  <!-- ***Put launch command for new planner here:-->
  <!-- Launch the New Planner Node -->
  <!--   <node pkg="f1tenth_simulator" name="new node's name" type="new file name" output="screen">
          <rosparam command="load" file="$(find f1tenth_simulator)/params.yaml"/>
        </node>    -->


  <!-- Launch RVIZ -->
    <node pkg="rviz" type="rviz" name="rviz" args="-d $(find f1tenth_simulator)/launch/simulator.rviz" output="screen"/>
</launch>
```

Σχ. 5.1 (εδώ είναι ο κώδικας για το simulation)

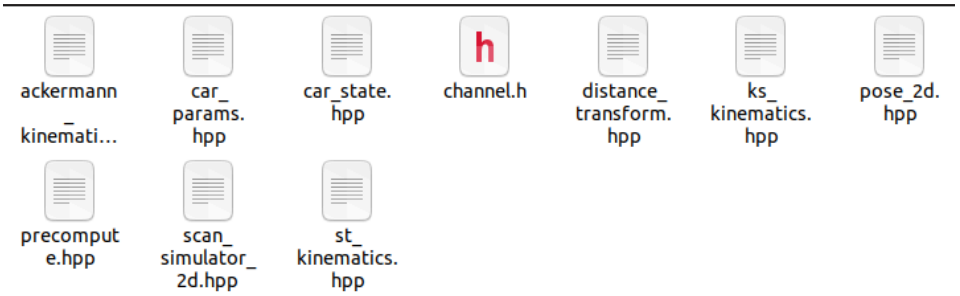
Στης επόμενες φωτογραφίες βλέπουμε τι περιέχει όλο το αρχείο με το simulation δεν έβγαλα εδώ screenshot διότι δεν χρειάζεται

Σχ. 5.2

Σχ. 5.3



Σχ. 5.4 (εδώ βλέπουμε της πίστες που έχω κατεβάσει εγώ χρησιμοποιώ το levine_blocked.pgm)



Σχ. 5.5

Στις παρακάτω φωτογραφίες θα σας παρουσιάσω το PID σαν κώδικα:

Center_wall_control :

```
#!/usr/bin/env python3

import rospy
import math
from race.msg import pid_input
from ackermann_msgs.msg import AckermannDriveStamped

kp = 5# 0.27 for porto
kd = 0.9#0.001 #0.09
ki = 0.6 #0.5
kp_vel = 10 #9 is using keep in middle
kd_vel = 0.01
ki_error = -3
prev_error = -1
vel_input = 2.5        # base velocity
angle = 360.0   # initial steering angle

pub = rospy.Publisher("/drive", AckermannDriveStamped, queue_size=5)


def control(data):
        global prev_error
        global vel_input
        global kp
        global kp_vel
        global kd
        global kd_vel
        global ki
        global angle

        dist_in_front = data.pid_vel
        front_obs = 1
        e = data.pid_error
        # Calculating deviation for lateral deviation from path
        kp_error = kp * e
        kd_error = kd * (e - prev_error)

        # Calculating error for velocity
        kp_vel_er = kp_vel * e
        kd_vel_er = kd * (e - prev_error)
        # ki_error = ki_error + (ki * e)

        vel_error = kp_vel_er + kd_vel_er
        pid_error = kp_error + kd_error


        min_angle=-10
        max_angle= 10

        # Heigher error results in lower velocity
        # while lower error results in heigher velocity
        if dist_in_front <= 5:
                front_obs = 3#abs(- ( (math.log10(dist_in_front/5)/math.log10(2)) +1 ))
        velo = vel_input + 1/ ( (abs(vel_error)*front_obs ))
        print("velo :", velo)
```

Σχ. 5.6

```
32          front_obs = 1
33          e = data.pid_error
34          # Calculating deviation for lateral deviation from path
35          kp_error = kp * e
36          kd_error = kd * (e - prev_error)
37
38          # Calculating error for velocity
39          kp_vel_er = kp_vel * e
40          kd_vel_er = kd * (e - prev_error)
41          # ki_error = ki_error + (ki * e)
42
43          vel_error = kp_vel_er + kd_vel_er
44          pid_error = kp_error + kd_error
45
46
47          min_angle=-10
48          max_angle= 10
49
50          # Heigher error results in lower velocity
51          # while lower error results in heigher velocity
52          if dist_in_front <= 5:
53                  front_obs = 3#abs(- ( (math.log10(dist_in_front/5)/math.log10(2)) +1 ))
54          velo = vel_input + 1/ ( (abs(vel_error)*front_obs ))
55          print("velo :", velo)
56          #corrected steering angle
57          angle = pid_error
58
59          #print("raw velo:", velo) # Testing
60
61          #Speed limit
62          if velo > 50 :
63                  velo = 50
64
65          # Filtering steering angle for Out-of-Range values
66          if angle < min_angle:
67                  angle = min_angle
68          elif angle > max_angle:
69                  angle = max_angle
70
71          # print("filtered angle :" , angle) # Testing
72
73
74          # Sending Drive information to Car
75          msg = AckermannDriveStamped()
76          msg.header.stamp = rospy.Time.now()
77          msg.header.frame_id = ''
78          msg.drive.speed = velo
79          msg.drive.steering_angle = angle
80          pub.publish(msg)
81
82  if __name__ == '__main__':
83          print("Starting control...")
84          rospy.init_node('pid_controller', anonymous=True)
85          rospy.Subscriber("error", pid_input, control)
86          rospy.spin()
```

Σχ. 5.7

Center_wall_follow:

```python
#!/usr/bin/env python3

import rospy
import math
from sensor_msgs.msg import LaserScan
from race.msg import pid_input


angle_range = 360              # sensor angle range of the lidar
car_length = 5  # projection distance we project car forward.
vel = 1.0              # used for pid_vel (not much use).
error = -1

pub = rospy.Publisher('error', pid_input, queue_size=10)

def getRange(data,theta):

    angle_range = data.angle_max - data.angle_min
    angle_increment  = data.angle_increment
    scan_range = []
    rad2deg_factor = 57.296
    angle_range *= rad2deg_factor
    angle_increment *= rad2deg_factor

    for element in data.ranges:
        if math.isnan(element) or math.isinf(element):
            element = 100
        scan_range.append(element)

    index = round(theta / angle_increment)

    return scan_range[index]

def callback(data):

    dist_in_front = getRange(data, 180)
    theta = 30
    left_dist = getRange(data, 270)
    right_dist = getRange(data,90)     # Ray perpendicular to right side of car
    a_right = getRange(data,(90 + theta)) # Ray at degree theta from right_dist
    a_left = getRange(data,(270 - theta))
    theta = math.radians(theta)

    # Calculating the deviation of steering(alpha) from right
    alpha_r = math.atan( (a_right * math.cos(theta) - right_dist)/ a_right * math.sin(theta) )
    curr_pos_r = right_dist * math.cos(alpha_r) # Present Position
    fut_pos_r = curr_pos_r + car_length * math.sin(alpha_r) # projection in Future Position

    # Calculating the deviation of steering(alpha) from left
    alpha_l = math.atan( (a_left * math.cos(theta) - left_dist)/ a_left * math.sin(theta) )
    curr_pos_l = left_dist * math.cos(alpha_l) # Present Position
    fut_pos_l = curr_pos_l + car_length * math.sin(alpha_l) # projection in Future Position

    error = - (fut_pos_r - fut_pos_l) # total error
    # print('error: ', error) #Testing
```

Σχ. 5.8

```
pub = rospy.Publisher('error', pid_input, queue_size=10)

def getRange(data,theta):

    angle_range = data.angle_max - data.angle_min
    angle_increment  = data.angle_increment
    scan_range = []
    rad2deg_factor = 57.296
    angle_range *= rad2deg_factor
    angle_increment *= rad2deg_factor

    for element in data.ranges:
        if math.isnan(element) or math.isinf(element):
            element = 100
        scan_range.append(element)

    index = round(theta / angle_increment)

    return scan_range[index]

def callback(data):

    dist_in_front = getRange(data, 180)
    theta = 30
    left_dist = getRange(data, 270)
    right_dist = getRange(data,90)     # Ray perpendicular to right side of car
    a_right = getRange(data,(90 + theta)) # Ray at degree theta from right_dist
    a_left = getRange(data,(270 - theta))
    theta = math.radians(theta)

    # Calculating the deviation of steering(alpha) from right
    alpha_r = math.atan( (a_right * math.cos(theta) - right_dist)/ a_right * math.sin(theta) )
    curr_pos_r = right_dist * math.cos(alpha_r) # Present Position
    fut_pos_r = curr_pos_r + car_length * math.sin(alpha_r) # projection in Future Position

    # Calculating the deviation of steering(alpha) from left
    alpha_l = math.atan( (a_left * math.cos(theta) - left_dist)/ a_left * math.sin(theta) )
    curr_pos_l = left_dist * math.cos(alpha_l) # Present Position
    fut_pos_l = curr_pos_l + car_length * math.sin(alpha_l) # projection in Future Position

    error = - (fut_pos_r - fut_pos_l) # total error
    # print('error: ', error) #Testing

    # Sending PID error to Control
    msg = pid_input()
    msg.pid_error = error
    msg.pid_vel = dist_in_front # pid_vel used for distance in front
    pub.publish(msg)


if __name__ == '__main__':
    print("Laser node started")
    rospy.init_node('dist_finder',anonymous = True)
    rospy.Subscriber("scan",LaserScan,callback)
    rospy.spin()
```

Σχ. 5.9

Control :

```
#!/usr/bin/env python3

import rospy
from race.msg import pid_input
from ackermann_msgs.msg import AckermannDriveStamped

kp = 0.1 #45
kd = 0.085#0.001 #0.09
ki = 0.5#0.5
kp_vel = 5 #9 is using keep in middle
kd_vel = 0.15
ki_error = 60
prev_error = -1
vel_input = 3.5        # base velocity
angle = 360.0  # initial steering angle

pub = rospy.Publisher("/drive", AckermannDriveStamped, queue_size=5)


def control(data):
        global prev_error
        global vel_input
        global kp
        global kp_vel
        global kd
        global kd_vel
        global ki
        global angle

        e = data.pid_error
        # Calculating deviation for lateral deviation from path
        kp_error = kp * e
        kd_error = kd * (e - prev_error)

        # Calculating error for velocity
        kp_vel_er = kp_vel * e
        kd_vel_er = kd * (e - prev_error)
        # ki_error = ki_error + (ki * e)

        vel_error = kp_vel_er + kd_vel_er
        pid_error = kp_error + kd_error


        min_angle=-10
        max_angle= 10

        # Heigher error results in lower velocity
        # while lower error results in heigher velocity
        velo = vel_input + 1/(abs(vel_error))

        #corrected steering angle
        angle = pid_error

        #print("raw velo:", velo) # Testing

```

Σχ. 6.0

```
        global kd
        global angle

        e = data.pid_error
        # Calculating deviation for lateral deviation from path
        kp_error = kp * e
        kd_error = kd * (e - prev_error)

        # Calculating error for velocity
        kp_vel_er = kp_vel * e
        kd_vel_er = kd * (e - prev_error)
        # ki_error = ki_error + (ki * e)

        vel_error = kp_vel_er + kd_vel_er
        pid_error = kp_error + kd_error


        min_angle=-10
        max_angle= 10

        # Heigher error results in lower velocity
        # while lower error results in heigher velocity
        velo = vel_input + 1/(abs(vel_error))

        #corrected steering angle
        angle = pid_error

        #print("raw velo:", velo) # Testing

        #Speed limit
        if velo > 15 :
                velo = 10

        # Filtering steering angle for Out-of-Range values
        if angle < min_angle:
                angle = min_angle
        elif angle > max_angle:
                angle = max_angle

        # print("filtered angle :" , angle) # Testing


        # Sending Drive information to Car
        msg = AckermannDriveStamped()
        msg.header.stamp = rospy.Time.now()
        msg.header.frame_id = ''
        msg.drive.speed = velo
        msg.drive.steering_angle = angle
        pub.publish(msg)

if __name__ == '__main__':
        print("Starting control...")
        rospy.init_node('pid_controller', anonymous=True)
        rospy.Subscriber("error", pid_input, control)
        rospy.spin()
```

Σχ. 6.1

Dist_finder:

```
#!/usr/bin/env python3

import rospy
import math
from sensor_msgs.msg import LaserScan
from race.msg import pid_input


angle_range = 360              # sensor angle range of the lidar
car_length = 5  # projection distance we project car forward.
vel = 2.0               # used for pid_vel (not much use).
error = -1
dist_from_wall = 0.65

pub = rospy.Publisher('error', pid_input, queue_size=10)

def getRange(data,theta):

        angle_range = data.angle_max - data.angle_min
        angle_increment  = data.angle_increment
        scan_range = []
        rad2deg_factor = 57.296
        angle_range *= rad2deg_factor
        angle_increment *= rad2deg_factor

        for element in data.ranges:
                if math.isnan(element) or math.isinf(element):
                        element = 100
                scan_range.append(element)

        index = round(theta / angle_increment)

        return scan_range[index]

def callback(data):
        theta = 55
        left_dist = float(getRange(data, 270))
        a = getRange(data,(90 + theta)) # Ray at degree theta from right_dist
        right_dist = getRange(data,90)         # Ray perpendicular to right side of car
        theta_r = math.radians(theta)
        # dist_from_wall = (left_dist + right_dist)/2 # keep in middle

        # Calculating the deviation of steering(alpha)
        alpha = math.atan( (a * math.cos(theta_r) - right_dist)/ a * math.sin(theta_r) )
        AB = right_dist * math.cos(alpha) # Present Position
        CD = AB + car_length * math.sin(alpha) # projection in Future Position

        error = dist_from_wall - CD # total error
        # print('error: ', error) #Testing

        # Sending PID error to Control
        msg = pid_input()
        msg.pid_error = error
        msg.pid_vel = vel
        pub.publish(msg)
```

Σχ. 6.2

```
 8
 9 angle_range = 360             # sensor angle range of the lidar
10 car_length = 5  # projection distance we project car forward.
11 vel = 2.0              # used for pid_vel (not much use).
12 error = -1
13 dist_from_wall = 0.65
14
15 pub = rospy.Publisher('error', pid_input, queue_size=10)
16
17 def getRange(data,theta):
18
19         angle_range = data.angle_max - data.angle_min
20         angle_increment  = data.angle_increment
21         scan_range = []
22         rad2deg_factor = 57.296
23         angle_range *= rad2deg_factor
24         angle_increment *= rad2deg_factor
25
26         for element in data.ranges:
27                 if math.isnan(element) or math.isinf(element):
28                         element = 100
29                 scan_range.append(element)
30
31         index = round(theta / angle_increment)
32
33         return scan_range[index]
34
35 def callback(data):
36         theta = 55
37         left_dist = float(getRange(data, 270))
38         a = getRange(data,(90 + theta)) # Ray at degree theta from right_dist
39         right_dist = getRange(data,90)         # Ray perpendicular to right side of car
40         theta_r = math.radians(theta)
41         # dist_from_wall = (left_dist + right_dist)/2 # keep in middle
42
43         # Calculating the deviation of steering(alpha)
44         alpha = math.atan( (a * math.cos(theta_r) - right_dist)/ a * math.sin(theta_r) )
45         AB = right_dist * math.cos(alpha) # Present Position
46         CD = AB + car_length * math.sin(alpha) # projection in Future Position
47
48         error = dist_from_wall - CD # total error
49         # print('error: ', error) #Testing
50
51         # Sending PID error to Control
52         msg = pid_input()
53         msg.pid_error = error
54         msg.pid_vel = vel
55         pub.publish(msg)
56
57
58 if __name__ == '__main__':
59         print("Laser node started")
60         rospy.init_node('dist_finder',anonymous = True)
61         rospy.Subscriber("scan",LaserScan,callback)
62         rospy.spin()
```

Σχ. 6.3

Scan_test:

```
#! /usr/bin/env python3

import rospy
from sensor_msgs.msg import LaserScan

def callback(data):
    print(data.ranges[270])

rospy.init_node("scan_test", anonymous=False)
sub = rospy.Subscriber("/scan", LaserScan, callback)
rospy.spin()
```

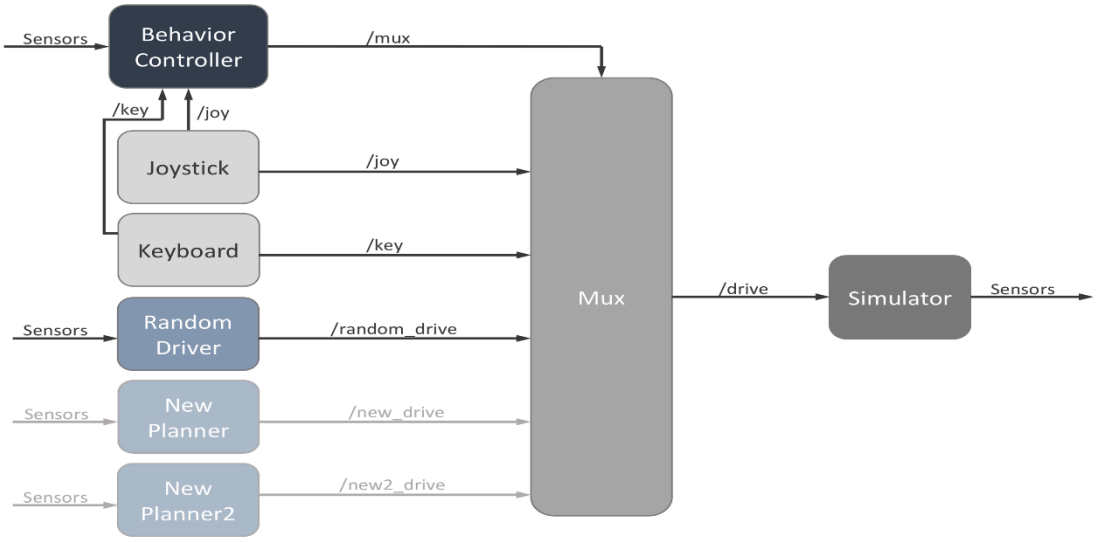
Σχ. 6.4

4 Κεφάλαιο : Σύνοψη

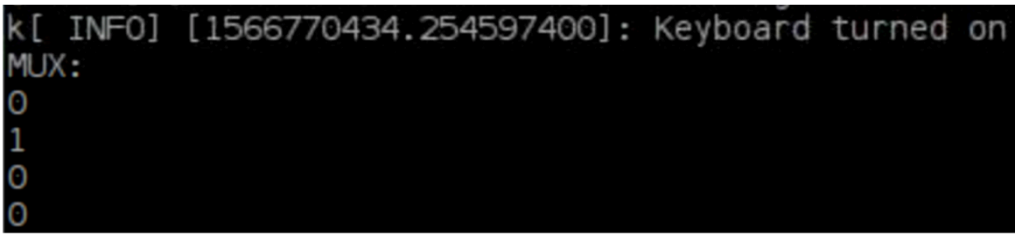
Σε αυτό το κεφάλαιο συνοψίζεται όλη η διαδικασία της πτυχιακής, αναλύονται δηλαδή όλα τα βήματα που πραγματοποιήθηκαν για την δημιουργία και την εκτέλεση των προγραμμάτων. Αρχικά έπρεπε να διαβάσω το βιβλίο του ROS που ήταν ευανάγνωστο και κάλυπτε με σαφήνεια ότι χρειάζεται ένας φοιτητής για ένα ξεκίνημα στο ROS. Έπειτα έπρεπε να κατεβάσω το LINUX UBUNTU 20.04 διότι σε αυτό το λογισμικό μπορούσε να απόδοση με αρκετά καλή ακρίβεια το ROS. Εδώ να σημειωθεί ότι είχα κατεβάσει και στα Windows 10 που έχω ένα visual LINUX αλλά δεν έτρεχε σωστά το ROS και έτσι εγκατέστησα το λογισμικό κανονικά. Μετά άρχισα να κατεβάζω το ROS και το εγκατέστησα. Ξεκίνησα να ψάχνω στο διαδίκτυο πως να χρησιμοποιώ το ROS διότι δεν το έχω ξανά ακούσει ούτε το ξέρω σαν πρόγραμμα μετά από 2 μήνες σχεδόν και ώρες ψαξίματος ήμουν έτοιμος να ξεκινήσω να γραφώ τον κώδικα. Στην αρχή άρχισα να κάνω το περιβάλλον με το RVIZ έψαχνα στο διαδίκτυο να έχει καμία πιστά έτοιμη γιατί απ’ ότι διάβαζα θα έπρεπε να σκανάρω τον χώρο στον οποίο τρέχει το αμάξι αλλά αφού δεν είχαμε ακόμα το αμάξι έπρεπε να αυτοσχεδιάσω και έτσι και έγινε για καλή μου τύχει είχε πάρα πολλά στο διαδίκτυο και έτσι μπόρεσα να βρω μερικούς χάρτες και να τα κατεβάσω. Μετά άρχισα να σχεδιάζω το αμάξι στο διαδίκτυο έχει βιντεάκια πως να το κάνεις απλά θέλει χρόνο όταν τελείωσα και αυτό ξεκίνησα να κάνω το πρόγραμμα την βοήθεια του διαδικτύου. Τέλος για το πρώτο για το πρώτο μέρος της διπλωματικής έτρεξα το RVIZ και αφού διαπίστωσα ότι όλα τρέχουν σωστά και ο χάρτης και το αμάξι φόρτωναν στο πρόγραμμα άρχισα να ψάχνω πως μπορώ να κάνω να κινηθεί το αμάξι στην αρχή με το πληκτρολόγιο και έπειτα με συντεταγμένες.

Έπειτα ακολουθεί η δημιουργία του δεύτερου μέρους της διπλωματικής, πριν από αυτό θα ήθελα να προσθέσω ότι μαζί με τον επιβλέποντα καθηγητή μου κάναμε ακόμα ένα simulation μικρό γιατί είχαμε πρόβλημα στο κύριο κώδικα αλλά αυτό θα το εξηγήσουμε παρακάτω. Άρχισα να γράφω τον κώδικα για το πως θα ξεκινήσει το

αμάξι με το πληκτρολόγιο και πως θα σταματάει εδώ καλό είναι να εξηγήσω κάτι με ένα διάγραμμα ροής:



Σε αυτό το διάγραμμα ροής φαίνεται πως ξεκίνησε το σκεπτικό με το κώδικα και πως λειτουργεί με το πληκτρολόγιο (μπορούμε να βάλουμε και joystick). Αυτό ήταν σαν κύρια ιδέα πως θα προχωρήσω. Σε αυτό το σημείο να τονιστεί πως στην περίπτωση που το αμάξι είχε τρακάρει, το πληκτρολόγιο θα απενεργοποιηθεί και θα πρέπει να πατήσω ξανά το K για να το ενεργοποιήσω ξανά. Επίσης, μπορεί να χειριστεί μόνο ένα πάτημα πλήκτρων κάθε φορά, επομένως το να κρατηθούν πατημένα πολλά πλήκτρα ταυτόχρονα δεν λειτουργεί. Τέλος, το πάτημα του A ή του D θα οδηγήσει σε μια σταθερή γωνία και ο μόνος τρόπος για να ισιώσετε είναι με το πλήκτρο spacebar. Εδώ είναι μια φωτογραφία πως φαινόταν ότι κινούνταν:



Στην συνέχεια στο τρίτο κατά σειρά μέρος της διπλωματικής, ξεκίνησα να κάνω το πρόγραμμα με συντεταγμένες και εδώ υπήρχε ένα πρόβλημα που δεν μπορούσα να το βρω και με καθυστέρησε αρκετό καιρό. Το πρόβλημα ήταν ότι με βάση το πρόγραμμα δεν μπορούσε να αλλάξει στα δεδομένα που ήθελα εγώ και μετά από αρκετό χρόνο βρήκαμε τρόπο να το κάνουμε. Φτιάξαμε ένα άλλο κώδικα που έτρεχε παράλληλα με το κύριο πρόγραμμα και στο δευτερεύον πρόγραμμα κάναμε PID οπού ουσιαστικά εμείς ρυθμίζουμε τον ελεγκτή μας.

Κάπου εδώ ολοκληρώθηκε όλο το κομμάτι της δημιουργίας, και το πρόγραμμα έτρεχε σωστά και το αμάξι με βάση τον αλγόριθμο άρχισε να λειτουργεί σωστά. Χρησιμοποιώντας τα εργαλεία της RVIZ, του ROS_API και γλώσσα προγραμματισμού c# ένα καινούργιο πρόγραμμα το οποίο θα αποτελεί το γραφικό περιβάλλον για το αμάξι και το οποίο λειτουργεί με ελεγκτή που μπορούμε να το προσομοιώσουμε στις δικές μας ανάγκες. Για την οπτικοποίηση η εφαρμογή αλληλεπιδρά με δυο προγράμματα έναν για την κίνηση του αμαξιού και αν θέλουμε

να το κινούμαι με το πληκτρολόγιο και ένα για την αυτοματοποίηση του αμαξιού με ελεγκτή. Το γραφικό περιβάλλον αποτελείται από 3 σελίδες, την αρχική και άλλες 4 στις οποίες βλέπουμε την αυτοματοποίηση του οχήματος μας , επιπλέον γίνεται έλεγχος για να μην τρακάρει το όχημα μας.

Σαν επόμενα βήματα της εργασίας θα μπορούσε να γίνει ένα αντίστοιχο user interface το οποίο όμως θα αφορά mobiles και θα έδινε πρόσβαση στις προβλέψεις σε ακόμη περισσότερο πληθυσμό. Ακόμα μια πρόταση θα μπορούσε να είναι η δημιουργία του μοντέλου από την αρχή χρησιμοποιώντας όμως κάποιον άλλο πρόγραμμα με σκοπό να μην έχει το PID αλλά να λειτουργεί με τους αισθητήρες του η ακόμα μπορούμε να κάνουμε την προσομοίωση σε άλλη πλατφόρμα παραδείγματος χάριν στο GAZEBO .

Αναφορές:

- ROS Robot Programming, YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim (link : https://drive.google.com/file/d/1sS5KEbsuj5wVwT3y7SVtrHu0YbMr90Qd/view)

Χρήσιμοι σύνδεσμοι

- https://github.com/f1tenth
- https://www.youtube.com/watch?v=eeV9XHYXMlI
- https://dev.to/admantium/robot-operating-system-creating-a-robot-simulation-45f1
- https://f1tenth.readthedocs.io/en/stable/going_forward/simulator/sim_use.html
- https://f1tenth.readthedocs.io/en/foxy_test/getting_started/build_car/all_together.html
- https://traxxas.com/products/models/electric/6804Rslash4x4platinum
- https://answers.ros.org/question/347214/how-to-load-a-map-in-rviz/
- https://www.youtube.com/watch?v=391fBiJeUkQ
- https://www.ros.org/
- https://wiki.ros.org/Documentation
- https://wiki.ros.org/pid
- https://www.youtube.com/watch?v=-GJRhU1a3es&list=PL868twsx7OjdnroeAUFVBGlKGnFGi9txc
- https://www.youtube.com/@RealTimemLAB
- chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://web2.qatar.cmu.edu/~gdicaro/16311-Fall17/slides/Lab-1-ROS-Intro.pdf