ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΣΕΡΡΩΝ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόγραμμα Μεταπτυχιακών Σπουδών Ρομποτικής

Ολοκληρωμένα πακέτα αυτόνομης οδήγησης βασισμένα στο ROS

Διπλωματική Εργασία του:

Παπαδόπουλου Αθανάσιου (22)

Επιβλέπων: Σταύρος Βολογιαννίδης

ΣΕΡΡΕΣ, ΦΕΒΡΟΥΑΡΙΟΣ 2019

Περίληψη

Στην παρούσα διπλωματική εργασία αναλύουμε το σύστημα αυτόνομης οδήγησης Autoware σε συνδυασμό με το πακέτο εφαρμογών του ROS. Η αυτόνομη οδήγηση αποτελεί βασικό παράγοντα για τη μελλοντική κινητικότητα. Η σωστή αντίληψη του περιβάλλοντος από τα οχήματα είναι απαραίτητη για την ασφαλή οδήγηση, η οποία απαιτεί την ακριβή γεωμετρική και σημασιολογική πληροφόρηση σε πραγματικό χρόνο. Ένα αυτόνομο αυτοκίνητο θεωρείται ένα όχημα που μπορεί να ανιχνεύσει το περιβάλλον του και να πλοηγηθεί χωρίς την ανάγκη της ανθρώπινης επέμβασης. Το software της Autoware έχει σημειώσει πολύ μεγάλη πρόοδο, καθώς ο χρήστης μπορεί με το κατάλληλο υλικό όπως : ραντάρ, laser scans, GPS, οδομετρία κτλ όχι μόνο να προσομοιώσει μία αυτόνομη οδήγηση, αλλά να την πραγματοποιήσει κιόλας με ένα ρεαλιστικό όχημα σε ένα ελεγχόμενο περιβάλλον, ΠΡΟΣ ΤΟ ΠΑΡΟΝ. Αυτή τη στιγμή με το κατάλληλο hardware και software αυτό είναι εφικτό με το πακέτο του Autoware. Παρακάτω, θα δούμε αναλυτικά:

- τη δομή του software
- τρόπους εγκατάστασης
- ανάλυση των αλγορίθμων
- προσομοιώσεις σε διάφορες οδικές καταστάσεις

Τέλος θα αναλύσουμε το μέλλον της αυτόνομης οδήγησης καθώς επίσης και μία θεωρητική προσέγγιση δημιουργίας ενός αυτόνομου οχήματος.

Περιεχόμενα

Εισαγωγή

Η αυτόνομη οδήγηση αποτελεί βασικό παράγοντα για τη μελλοντική κινητικότητα. Η σωστή αντίληψη του περιβάλλοντος των οχημάτων είναι απαραίτητη για την ασφαλή οδήγηση, η οποία απαιτεί την ακριβή γεωμετρική και σημασιολογική πληροφόρηση σε πραγματικό χρόνο. Ένα αυτόνομο αυτοκίνητο είναι ένα όχημα που μπορεί να ανιχνεύσει το περιβάλλον του και να πλοηγηθεί χωρίς την ανάγκη της ανθρώπινης επέμβασης. Τα αυτόνομα οχήματα συνδυάζουν μία ποικιλία τεχνολογιών προκειμένου να μπορούν να αντιληφθούν το περιβάλλον τους. Εξαρτήματα που βοηθούν στην «αντίληψη» του οχήματος είναι: ραντάρ, leaser scans, GPS, οδομετρία, κάμερες κ.α. Όλα τα παραπάνω εξαρτήματα καταφέρνουν να «διαβάζουν» τις διάφορες πληροφορίες γύρω τους, με τελικό αποτέλεσμα την αναγνώριση ανθρώπων, σημάνσεων, διαβάσεων, εμποδίων κτλ, όπου εν τέλει βγάζουν και μία εφικτή και ασφαλή οδική διαδρομή. Τα οφέλη των αυτόνομων αυτοκινήτων περιλαμβάνουν:

- μειωμένη κινητικότητα
- μειωμένο κόστος υποδομής
- αύξηση της ασφάλειας
- μειωμένη εγκληματικότητα
- δυνητική σημαντική μείωση των ατυχημάτων και των τραυματισμών

Αναμένεται ότι τα αυτόνομα οχήματα: θα αυξήσουν τη ροή της κυκλοφορίας, θα βελτιώσουν την κινητικότητα των παιδιών (θα μπορούν να κυκλοφορούν πιο ασφαλή στους δρόμους), των ηλικιωμένων και των ατόμων με ειδικές ανάγκες. Επίσης, θα απαλλάσσει τους ταξιδιώτες από την ανάγκη της οδήγησης. Παρ' όλα τα παραπάνω οφέλη υπάρχουν ακόμα ανεπίλυτα προβλήματα όπως:

- Η πλήρης ασφάλεια. Δεν υπάρχει ακόμα η τεχνολογία που απαιτείται.
- Η αποδοχή της απώλειας ελέγχου των αυτοκινήτων από τους πολίτες
- Εφόσον υπάρξει η απαραίτητη τεχνολογία, έχουμε την ανησυχία των πολιτών σχετικά με την ασφάλεια αυτών των οχημάτων.

Εδώ εμπλέκεται και το νομικό πλαίσιο στη θέσπιση κυβερνητικών κανονισμών για την προστασία της ιδιωτικής ζωής και της ασφάλειας, (χάκερ , τρομοκρατία κτλ). Από την πλευρά της πολιτικής υπάρχει και ανησυχία σχετικά με την απώλεια θέσεων εργασίας που συνδέονται με την οδήγηση στον τομέα των οδικών μεταφορών.

Παράρτημα Εικόνων

Παράρτημα πινάκων

Κεφάλαιο 1ο

ROS & Αυτοκινούμενα Οχήματα

1.1 Η εξέλιξη του Robot Operating system

Το ROS [2] είναι ένα μεγάλο project με πολλούς ‘προγόνους’ και συνεισφέροντες. Η ανάγκη για ένα ανοικτό πλαίσιο συνεργασίας έγινε αισθητή από πολλούς ανθρώπους στην ερευνητική κοινότητα ρομποτικής και πολλά έργα έχουν δημιουργηθεί για αυτό τον σκοπό. Οι διάφορες προσπάθειες στο πανεπιστήμιο του Στάνφορντ στα μέσα της δεκαετίας του 2000, που αφορούσαν artificial intelligence, όπως το STanford AI Robot (STAIR) και το πρόγραμμα Personal Robots (PR) [3], δημιούργησαν εσωτερικά πρωτότυπα ευέλικτων και δυναμικών συστημάτων λογισμικού που προορίζονται για ρομποτική χρήση.
Το 2007, η Willow Garage [4], παρείχε σημαντικούς πόρους για να επεκτείνει περαιτέρω αυτές τις έννοιες και να δημιουργήσει καλά δοκιμασμένες υλοποιήσεις. Η προσπάθεια ενισχύθηκε από αμέτρητους ερευνητές που συνέβαλαν με τον χρόνο και την εμπειρία τους τόσο στις βασικές ιδέες του ROS όσο και στα βασικά πακέτα λογισμικού. Καθ’ όλη τη διάρκεια του προγράμματος, το λογισμικό αναπτύχθηκε ανοικτά με τη χρήση της αδειοδότησης BSD* ανοιχτού κώδικα και σταδιακά έχει γίνει μια ευρέως χρησιμοποιούμενη πλατφόρμα στην ερευνητική κοινότητα ρομποτικής.

Από την αρχή το ROS αναπτύχθηκε σε πολλά ιδρύματα και για πολλαπλά ρομπότ όπως, το PR2 της Willow Garage [4]. Αν και θα ήταν πολύ απλούστερο για όλους τους συνεισφέροντες να τοποθετήσουν τον κώδικα τους στους ίδιους servers, με τα χρόνια το "ομοσπονδιακό" – αποκεντρωμένο μοντέλο, έχει αναδειχθεί ως ένα από τα μεγαλύτερα πλεονεκτήματα του οικοσυστήματος του ROS. Οποιαδήποτε ομάδα μπορεί να φτιάξει το δικό της αποθετήριο (repository) στους δικούς της servers και να διατηρεί πλήρη έλεγχο αυτού. Εάν επιλέξουν να δημοσιοποιήσουν το αποθετήριό τους, στην συνέχεια θα μπορούν να λάβουν την αναγνώριση και την πιστοποίηση που τους αξίζει για τα επιτεύγματά τους και να επωφεληθούν από ανατροφοδοτήσεις και βελτιώσεις από άλλα προγράμματα «λογισμικού ανοιχτού κώδικα».

Το οικοσύστημα ROS αποτελείται σήμερα από δεκάδες χιλιάδες χρήστες παγκοσμίως, που εργάζονται σε τομείς που κυμαίνονται από απλό χόμπι έως μεγάλα βιομηχανικά συστήματα αυτοματισμού.

**Οι άδειες BSD είναι μια οικογένεια επιτρεπόμενων αδειών ελεύθερου λογισμικού, επιβάλλοντας ελάχιστους περιορισμούς στη χρήση και την ανακατανομή του καλυπτόμενου λογισμικού*

Το ROS [2] είναι ένα μετα-λειτουργικό σύστημα (meta-operating system) ανοιχτού κώδικα για ρομπότ. Παρέχει τις υπηρεσίες που μπορεί να περιμένει κάποιος από ένα λειτουργικό σύστημα, συμπεριλαμβανομένου καθώς επίσης:

- hardware abstraction
- έλεγχος συσκευών χαμηλού επιπέδου της εφαρμογής κοινώς χρησιμοποιούμενων λειτουργιών
- μετάδοση μηνυμάτων μεταξύ των διαδικασιών
- διαχείριση πακέτων

Παρέχει επίσης εργαλεία και βιβλιοθήκες για την απόκτηση, κατασκευή, γραφή και εκτέλεση κώδικα σε πολλούς υπολογιστές. Το ROS είναι παρόμοιο σε ορισμένα σημεία με άλλα «robot-frameworks» όπως:

- Player [5]
- YARP [6]
- Orocos [7]
- CARMEN [8]
- Orca [9]
- MOOS [10]
- Microsoft Robotics Studio [11]

Το ROS runtime graph είναι ένα peer-to-peer δίκτυο διεργασιών, όπου τα ρομποτικά συστήματα επικοινωνούν μεταξύ τους χρησιμοποιώντας την υποδομή επικοινωνίας του ROS. Το ROS υλοποιεί διάφορες μορφές επικοινωνίας, όπως σύγχρονη επικοινωνία τύπου RPC [12] (remote procedure call) πάνω σε υπηρεσίες, ασύγχρονη ροή δεδομένων πάνω σε topics και αποθήκευση δεδομένων σε έναν server.

(για περισσότερες πληροφορίες δείτε: http://wiki.ros.org/ROS/Introduction).

1.2 Στόχοι του ROS

Στόχος του ROS είναι να υποστηρίξει την επαναχρησιμοποίηση κώδικα στην έρευνα και ανάπτυξη ρομποτικής και όχι να προσφέρει απλά περισσότερες λειτουργίες. Το ROS έχει ένα κατανεμημένο πλαίσιο διαδικασιών (γνωστά και ως Nodes) το οποίο επιτρέπει την εκπόνηση ξεχωριστών εκτελέσιμων προγραμμάτων καθώς και τη σύνδεση τους κατά το χρόνο εκτέλεσης. Αυτές οι διαδικασίες μπορούν να ομαδοποιηθούν σε Packages και Stacks. Το ROS υποστηρίζει επίσης ένα «κοινό» αποθετήριο κώδικα που επιτρέπει την κοινή συνεργασία πολλών ομάδων.

Για την υποστήριξη του πρωταρχικού στόχου (ανταλλαγή και συνεργασία), υπάρχουν διάφοροι άλλοι στόχοι στα πλαίσια του ROS όπως:

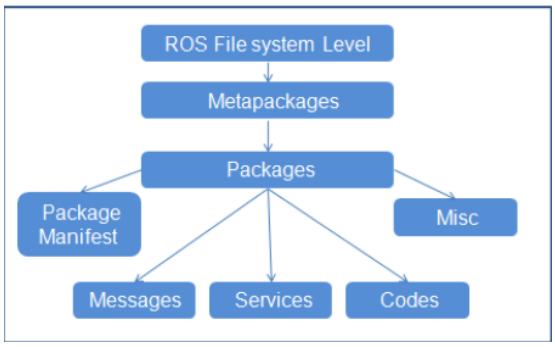
- *Thin:* Το ROS έχει σχεδιαστεί ώστε να είναι όσο το δυνατόν «ελαφρύτερο». Όπως λένε και οι ίδιοι : *we won't wrap your main()*. Επίσης, το ROS είναι εύκολο να ενσωματωθεί και σε

άλλα πλαίσια λογισμικού ρομπότ. Πράγμα που έχει γίνει στα πχ OpenRAVE, Orocos και Player.

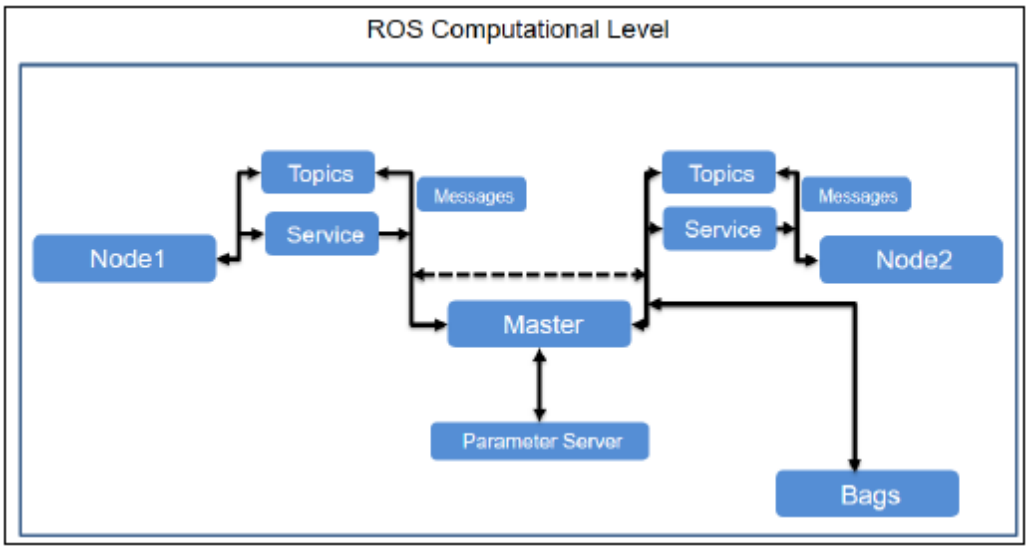
- *ROS-agnostic libraries:* το αναπτυξιακό μοντέλο που προτιμάτε περισσότερο είναι να γράφουμε ROS-agnostic βιβλιοθήκες με καθαρές λειτουργικές διεπαφές.
- *Language independence:* το ROS είναι εύκολο να χρησιμοποιηθεί σε οποιαδήποτε σύγχρονη γλώσσα προγραμματισμού. Υπάρχουν ήδη βιβλιοθήκες σε Python, C++ και Lisp. Επίσης, υπάρχουν πειραματικές βιβλιοθήκες σε Java και Lua.
- *Easy testing:* Το ROS έχει ένα ενσωματωμένο πλαίσιο δοκιμής μονάδας / ολοκλήρωσης, το οποίο ονομάζεται rostest, το οποίο διευκολύνει τον έλεγχο των δοκιμαστικών εκδόσεων διαφόρων πακέτων.
- Scaling: Το ROS είναι κατάλληλο για δημιουργία μεγάλων συστημάτων και για μεγάλες διαδικασίες ανάπτυξης.

1.3 Διαφορά του ROS από τα άλλα ρομποτικά software

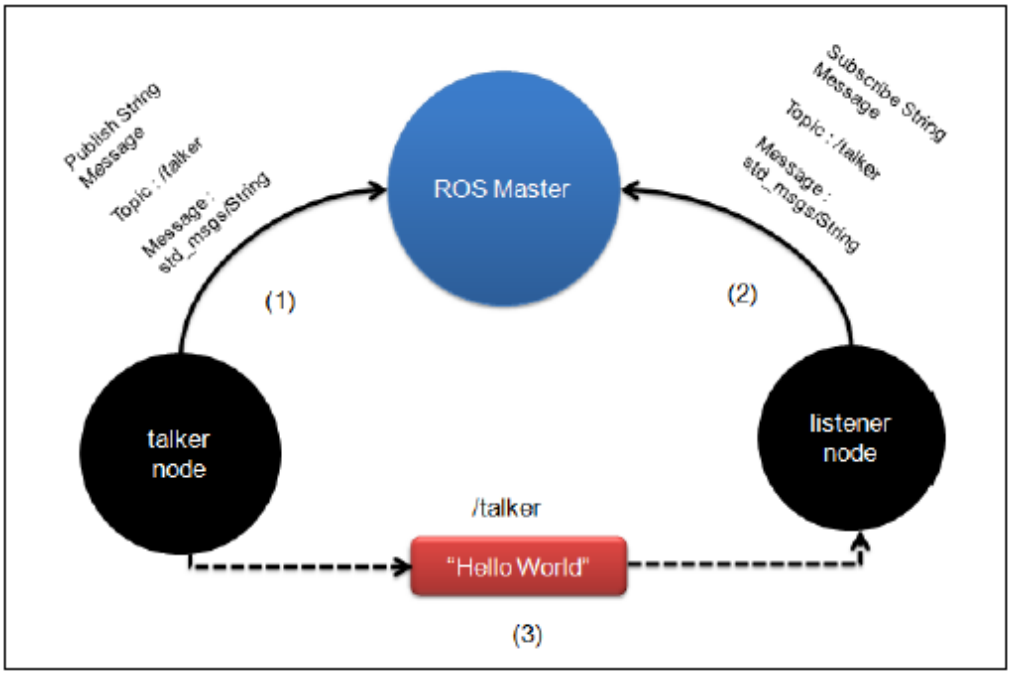
Η επιλογή ενός framework είναι δύσκολη και συνήθως εξαρτάται από το τι προσπαθούμε να κάνουμε και που θέλουμε να φτάσουμε. Το Player είναι ιδανικό για απλές, non-articulated mobile platforms. Σχεδιάστηκε για να παρέχει εύκολη πρόσβαση σε αισθητήρες και κινητήρες. Το ROS, από την άλλη πλευρά, σχεδιάζεται γύρω από περίπλοκες πλατφόρμες χειρισμού, με ενεργοποιημένη ανίχνευση (λέιζερ, κεφαλές αισθητήρων πανοραμικής / κλίσης, αισθητήρες που είναι προσαρτημένοι σε βραχίονες) κλπ. Σε σύγκριση με το Player, το ROS διευκολύνει την αξιοποίηση ενός κατανεμημένου περιβάλλοντος υπολογιστών, ενώ το Player προσφέρει περισσότερα προγράμματα οδήγησης υλικού. Το ROS προσφέρει περισσότερες εφαρμογές αλγορίθμων. Επίσης, το ROS είναι ισχυρότερο και πιο ευέλικτο από τον Player, αλλά, ως συνήθως, η μεγαλύτερη δύναμη και η ευελιξία έρχονται με το κόστος της μεγαλύτερης πολυπλοκότητας. Αξίζει να σημειωθεί ότι το ROS αξιοποιεί πολύ κώδικα από το πρόγραμμα Player. Υπάρχουν κόμβοι του ROS που επαναχρησιμοποιούν κώδικα από πολλούς οδηγούς προγράμματος οδήγησης (πχ Stage και Gazebo), τα οποία υποστηρίζονται και χρησιμοποιούνται ευρέως στην κοινότητα ROS. Τα παρακάτω διαγράμματα παρουσιάζουν την βασική αρχιτεκτονική του ROS:



Εικόνα 1: ROS file System Level

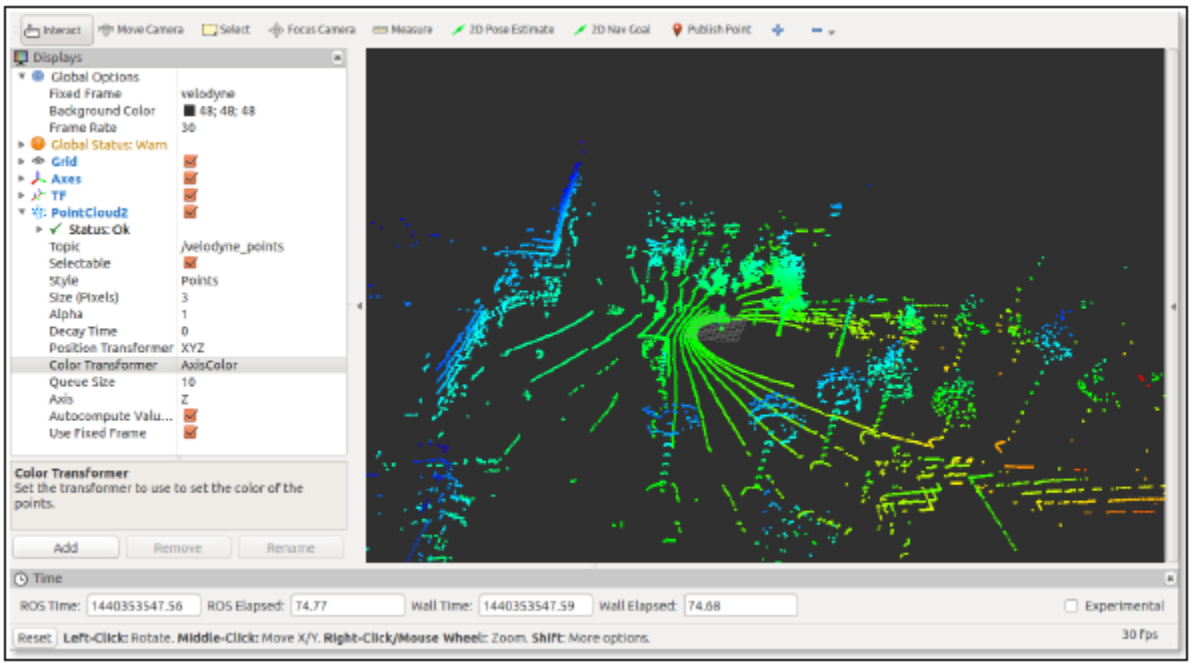


Εικόνα 2: ROS computational graph concept



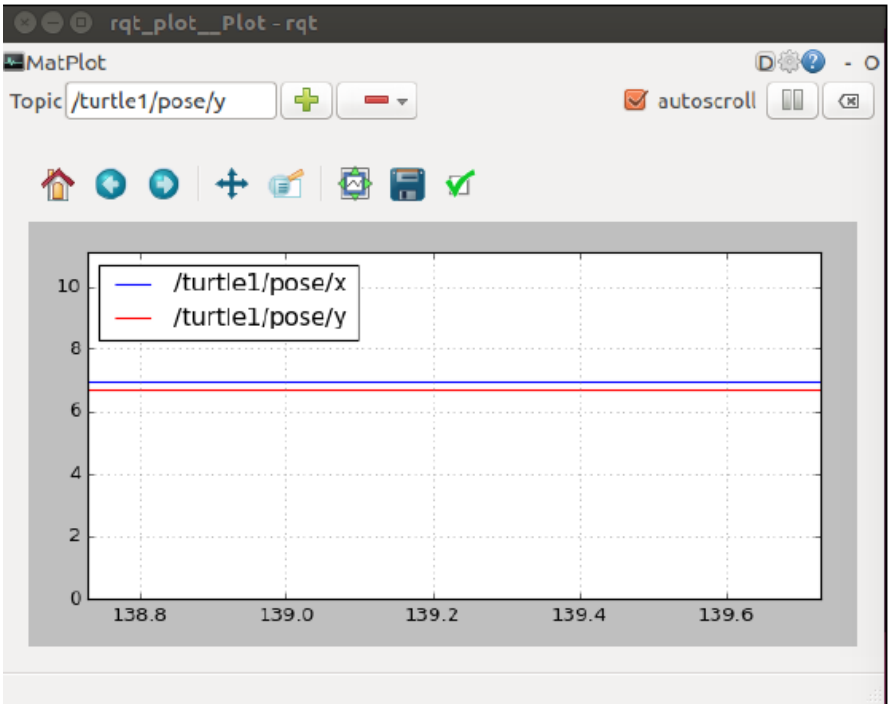
Εικόνα 3: Communication between ROS node

1.4 Υποστηριζόμενα γραφικά περιβάλλοντα από το ROS package



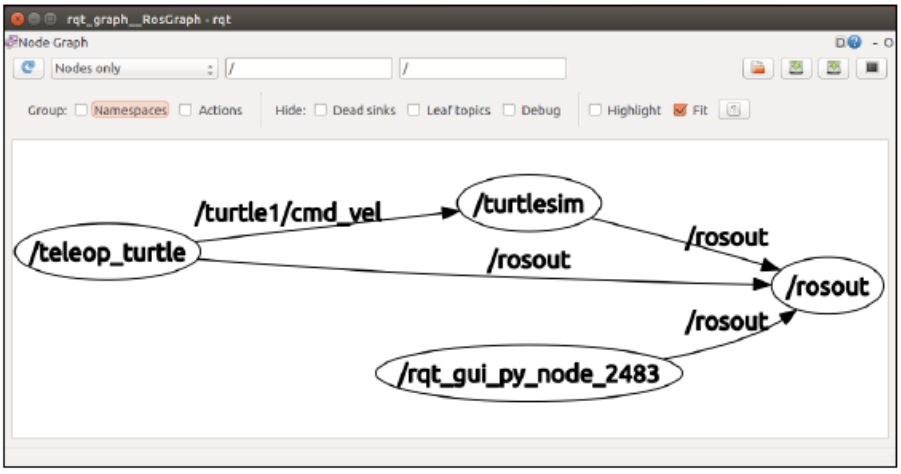
Εικόνα 4: Rviz (ROS visualizer)

Rviz: είναι ένας από τους τρισδιάστατους οπτικοποιητές που διατίθενται στο ROS για να απεικονίσουν σε 2D και 3D τα topics και τις παραμέτρους του ROS. Το Rviz βοηθά στην απεικόνιση δεδομένων όπως: ρομποτικά μοντέλα, δεδομένα τρισδιάστατου μετασχηματισμού του ρομπότ, σύννεφα σημείων, δεδομένα λέιζερ και εικόνας, καθώς και μια ποικιλία από διαφορετικά δεδομένα αισθητήρων.



Εικόνα 5 : rqt_plot

rqt_plot : είναι ένα εργαλείο που βοηθά στον σχεδιασμό και στην απεικόνιση scalar values που προέρχονται από τα topics του ROS. Ο χρήστης μπορεί να δώσει ένα όνομα στο topic και να το ανιχνεύει στο rqt_plot με βάση αυτό.



Εικόνα 6 : rqt_graph

rqt_graph: είναι ένα interface το οποίο βοηθά τον χρήστη να απεικονίσει το γράφημα διασύνδεσης μεταξύ των κόμβων του ROS.

Εικόνα 7 : Gazebo

Gazebo: είναι ένα software ανοιχτού κώδικα που έχει ενσωματωθεί με το ROS. Το Gazebo είναι ένας δυναμικός ρομποτικός προσομοιωτής με μεγάλη ποικιλία μοντέλων ρομπότ και εκτενή υποστήριξη αισθητήρων Οι λειτουργίες του Gazebo μπορούν να προστεθούν μέσω plugins. Οι τιμές αισθητήρων μπορούν να προσπελάσουν στο ROS μέσω topics, παραμέτρων και services.

1.5 Δημιουργία ενός Self –Driving αυτοκινήτου χρησιμοποιώντας το ROS

Έχοντας αυτή τη στιγμή στην αγορά μία ποικιλία από software (όπως είδαμε παραπάνω) που μπορούν να συνεισφέρουν άμεσα στο concept των self driving οχημάτων, ερχόμαστε στην θέση που θα πρέπει κρίνουμε το κατάλληλο. Τα βασικά θέματα που πρέπει να τονιστούν εδώ σχετικά με τα αυτόνομα οχήματα και τον τρόπο υλοποίησής τους με το ROS είναι :

- Βασικές γνώσεις για τα αυτόνομα οχήματα
- Software block διάγραμμα ενός τυπικού αυτόνομου οχήματος
- Προσομοίωση και διασύνδεση αισθητήρων αυτόνομου οχήματος στο ROS
- Προσομοίωση ενός αυτοκινήτου με αισθητήρες στο Gazebo
- Διεπαφή ενός αυτοκινήτου DBW σε ROS

Με βάση τα παραπάνω, ένας χρήστης θα πρέπει να έχει στα υπ όψιν του όχι μόνο το software αλλά και το ανάλογο hardware.

Παρακάτω μπορούμε να δούμε βασικά παραδείγματα αυτόνομων οχημάτων.

Εικόνα 8 : Tartan

Ο *Boss*, όπως τον έχουν επονομάσει, της Tartan Racing, το ρομποτικό SUV που κέρδισε το DARPA Urban Challenge του 2007 (http://www.tartanracing.org/).

Το 2009, η Google άρχισε να αναπτύσσει το δικό της αυτόνομο όχημα, γνωστό ως Waymo (https://waymo.com/).

Εικόνα 9 : Google αυτόνομο όχημα

Επίσης,αυτόνομα υπάρχουν και λεωφορεία. (http://www.auro.ai/)

Εικόνα 10 : Auro 1

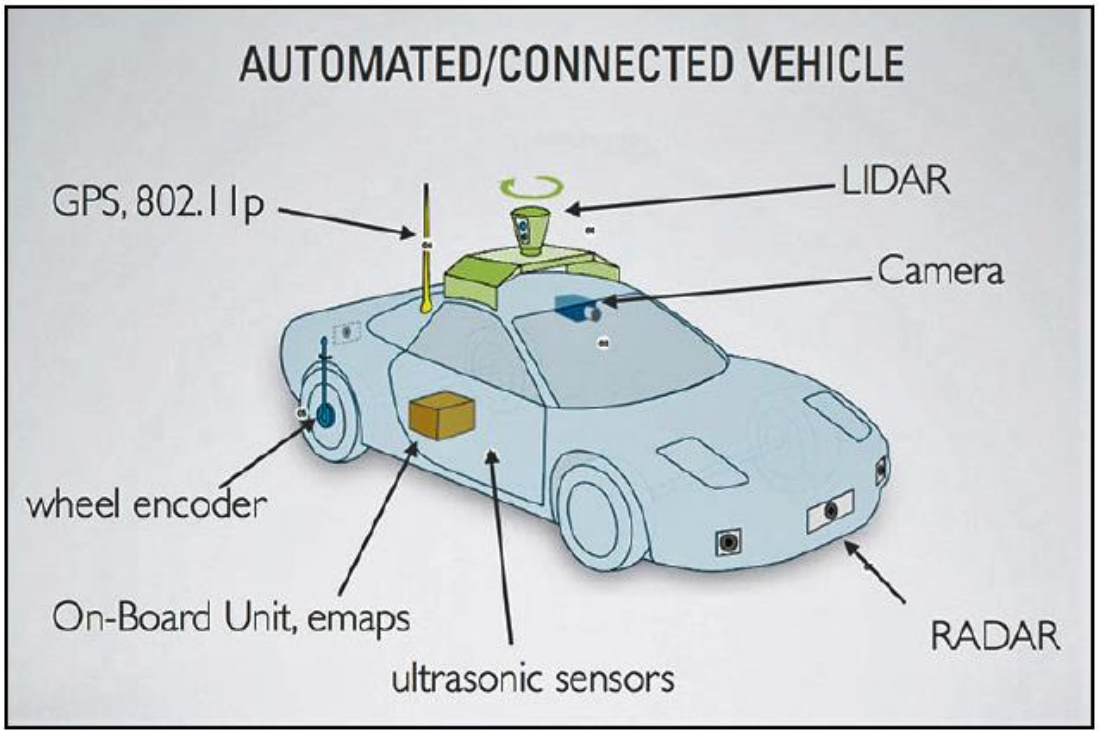
Υπάρχουν διάφορα επίπεδα τα οποία ορίζουν ένα όχημα ως αυτόνομο:

- **Επίπεδο 0** : Τα οχήματα με αυτονομία επιπέδου 0 είναι εντελώς χειροκίνητα, με έναν άνθρωπο οδηγό. Τα περισσότερα αυτοκίνητα ανήκουν σε αυτή την κατηγορία.
- **Επίπεδο 1**: Τα οχήματα με αυτονομία επιπέδου 1 έχουν έναν άνθρωπο οδηγό, αλλά έχουν επίσης ένα σύστημα υποστήριξης που μπορεί είτε να ελέγξουν αυτόματα το σύστημα επιτάχυνσης / επιβράδυνσης με τη χρήση πληροφοριών από το περιβάλλον. Όλες οι άλλες λειτουργίες πρέπει να ελέγχονται από τον οδηγό.
- **Επίπεδο 2**: Σ αυτό το επίπεδο, το όχημα μπορεί να ελέγξει το τιμόνι και επιτάχυνση / επιβράδυνση. Όλες οι άλλες εργασίες πρέπει να ελέγχονται από τον οδηγό. Εδώ μπορούμε να πούμε ότι το όχημα είναι εν μέρει αυτοματοποιημένο σε αυτό το επίπεδο.
- **Επίπεδο 3**: Σε αυτό το επίπεδο, αναμένεται όλες οι εργασίες να εκτελούνται αυτόνομα, αλλά ταυτόχρονα, αναμένεται ότι ένας άνθρωπος θα παρέμβει όποτε απαιτείται. Αυτό το επίπεδο ονομάζεται αυτοματοποίηση υπό όρους.
- **Επίπεδο 4**: Σε αυτό το επίπεδο, δεν υπάρχει ανάγκη για οδηγό. Όλα διαχειρίζονται από ένα αυτοματοποιημένο σύστημα. Αυτό το είδος αυτόνομου συστήματος θα λειτουργήσει σε ένα συγκεκριμένο περιβάλλον υπό συγκεκριμένες καιρικές συνθήκες.
- **Επίπεδο 5:** Αυτό το επίπεδο ονομάζεται πλήρης αυτοματοποίηση. Σε αυτό το επίπεδο, τα πάντα είναι αυτοματοποιημένα και μπορεί να λειτουργήσουν σε οποιονδήποτε δρόμο και οποιαδήποτε καιρική κατάσταση. Δεν υπάρχει ανάγκη για έναν άνθρωπο οδηγό.

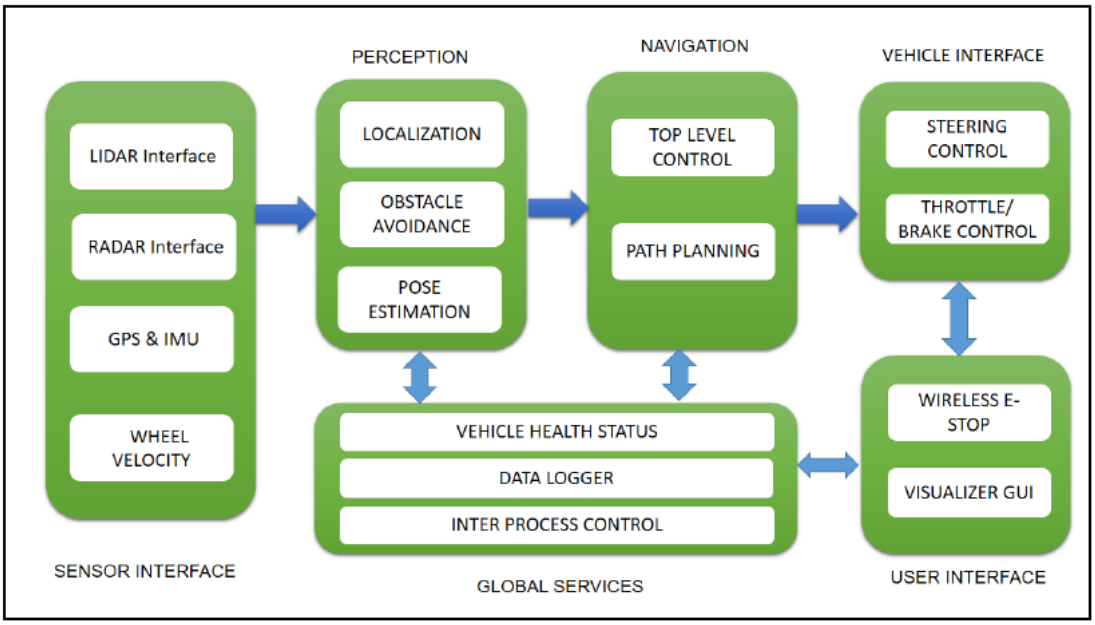
1.6 Ελάχιστα εξαρτήματα που χρειαζόμαστε για ένα αυτόνομο όχημα

Για την υλοποίηση ενός αυτόνομου οχήματος χρειάζονται γενικά, κάποια πολύ βασικά εξαρτήματα τα οποία προσφέρουν τις απαραίτητες πληροφορίες που χρειάζεται το όχημα για να λειτουργήσει αυτόνομα. Τέτοια εξαρτήματα είναι τα παρακάτω:

- GPS: ένα παγκόσμιο δορυφορικό σύστημα ραδιοπλοήγησης που παρέχει πληροφορίες γεωγραφικής κατανομής και χρόνου σε ένα δέκτη GPS οπουδήποτε στη Γη. Σημαντικό: πρέπει να υπάρχει οπτική επαφή με τέσσερις ή περισσότερος δορυφόρους.
- IMU: είναι μία ηλεκτρονική συσκευή που μετράει και αναφέρει τον προσανατολισμό, την ταχύτητα, την βαρυντική δύναμη και το μαγνητικό πεδίο που περιβάλλει το σώμα. Αυτό πραγματοποιείται με ένα συνδυασμό επιταχυνσιόμετρων και γυροσκοπίων.
- Wheel encoders: είναι μια ηλεκτρομηχανική συσκευή που μετατρέπει τη γωνιακή θέση ή την κίνηση ενός άξονα σε αναλογικά ή ψηφιακά σήματα εξόδου.
- Camera: το μέσο με το οποίο «διαβάζουμε» τον εξωτερικό χώρο και το προβάλουμε στην οθόνη.
- Ultrasonic sensors: είναι αισθητήρες υπερήχων που μετρούν την απόσταση χρησιμοποιώντας υπερηχητικά κύματα
- LIDAR: είναι συσκευές που μετράνε αποστάσεις και χαρτογραφούν χώρους. Η μέθοδος που χρησιμοποιεί είναι να μετρά την απόσταση από έναν στόχο φωτίζοντάς τον με παλλόμενο φως λέιζερ. Στη συνέχεια μετρώνται οι παλμοί που ανακλώνται με έναν αισθητήρα. Οι διαφορές στους χρόνους επιστροφής σήματος και τα μήκη κύματος μπορούν στη συνέχεια να χρησιμοποιηθούν για να γίνουν ψηφιακές 3-D αναπαραστάσεις του στόχου.
- RADAR: είναι μία συσκευή ανίχνευσης που χρησιμοποιεί ραδιοκύματα για τον προσδιορισμό του εύρους, της γωνίας ή της ταχύτητας των αντικειμένων. Ένα σύστημα ραντάρ αποτελείται από έναν πομπό που παράγει ηλεκτρομαγνητικά κύματα (μήκη κύματος ραδιοφώνου), μια κεραία μετάδοσης, μια κεραία λήψης (συχνά χρησιμοποιείται η ίδια κεραία για τη μετάδοση και τη λήψη) και ένα δέκτη και επεξεργαστή για τον προσδιορισμό των ιδιοτήτων του αντικειμένου (αντικειμένων). Τα ραδιοκύματα (παλμικά ή συνεχή) από τον πομπό αντανακλούν το αντικείμενο και επιστρέφουν στον δέκτη, δίνοντας πληροφορίες σχετικά με τη θέση και την ταχύτητα του αντικειμένου.
- On-board computer: μπορεί να είναι οποιοδήποτε φορητό laptop υψηλών επιδόσεων, καθώς οι απαιτήσεις επεξεργασίας δεδομένων είναι αρκετά υψηλές.



Εικόνα 11 : Σημαντικά εξαρτήματα ενός αυτόνομου οχήματος

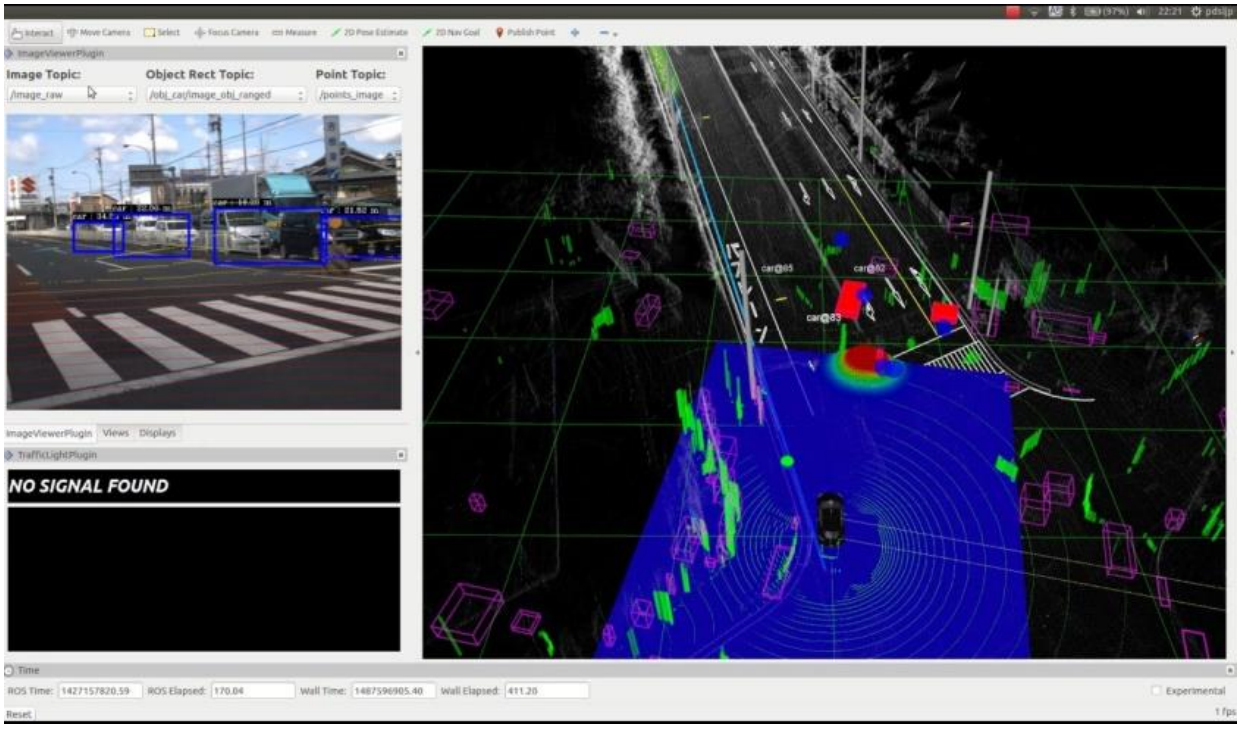


Εικόνα 12 : Software Block διάγραμμα αυτόνομου οχήματος

Πιο αναλυτικά τι κάνει το κάθε πεδίο στο διάγραμμα :

- **Sensor interface modules** : Όπως υποδεικνύει και το όνομα της μονάδας, όλη η επικοινωνία μεταξύ των αισθητήρων και του οχήματος γίνεται σε αυτό το μπλοκ. Μας επιτρέπει να παρέχουμε τα διάφορα δεδομένα που προέρχονται από τους αισθητήρες σε όλα τα άλλα μπλοκ. Οι κύριοι αισθητήρες είναι: το LIDAR, η κάμερα, το ραντάρ, το GPS, η IMU και οι wheel encoders.
- **Perception modules** : Εδώ επεξεργάζονται τα δεδομένα που έρχονται από τους αισθητήρες όπως: το LIDAR, η camera, το radar κτλ. Στην πορεία ταξινομεί τα δεδομένα σε στατικά και δυναμικά (κινούμενα και ακίνητα αντικείμενα). Επίσης, βοηθάει να εντοπιστεί το όχημα σε σχέση με το ψηφιακό του περιβάλλον.
- **Navigation modules** : Αυτή η μονάδα καθορίζει τη συμπεριφορά του οχήματος. Έχει motion planners και μηχανές πεπερασμένων καταστάσεων για διαφορετικές συμπεριφορές στο ρομπότ.
  - **Vehicle interface** : Τα σύγχρονα οχήματα είναι όλα εφοδιασμένα με τεχνολογία Drive By Wire (DBW). Το DBW ορίζει τη χρήση ηλεκτρικών ή ηλεκτρομηχανικών συστημάτων προκειμένου να εκτελεστούν διάφορες λειτουργίες του οχήματος. Αυτή η τεχνολογία αντικαθιστά τα παραδοσιακά μηχανικά συστήματα ελέγχου με ηλεκτρονικά συστήματα ελέγχου χρησιμοποιώντας ηλεκτρομηχανικούς αισθητήρες και ενεργοποιητές. Έτσι, πληροφορίες όπως γωνία τιμονιού, θέσεις πεντάλ κτλ, πλέον ελέγχονται ηλεκτρονικά. Το DBW είναι ευρέως γνωστό στα οχήματα και ως CAN[13], controller area network. Ουσιαστικά, δεν έχουμε μία μονάδα που λαμβάνει όλα τα σήματα, αλλά πολλές μονάδες όπου επικοινωνούν μεταξύ τους. Με το σύστημα CAN έχουμε το απαραίτητο δίκτυο για την αλληλοεπικοινωνία των εξαρτημάτων μας.
- **User interface** : Μπορεί να είναι π.χ. οθόνη αφής για να βλέπουμε τους χάρτες και να ορίζουμε τον προορισμό. Επίσης, υπάρχει και ο έλεγχος έκτακτης ανάγκη stop από τον χρήστη.
- **Global services** : βοηθάει στην καταγραφή των δεδομένων. Χρησιμεύει κυρίως για την αποφυγή υποκλοπής μηνυμάτων προκειμένου να διατηρείται η αξιοπιστία του λογισμικού.

Πάνω στο τομέα των αυτοκινούμενων οχημάτων και των open-source προγραμμάτων μεγάλη πρόοδο έχει σημειώσει η Ιαπωνική εταιρία Autoware.



Εικόνα 13 : Παράδειγμα λειτουργίας του Autoware

Κεφάλαιο 2ο

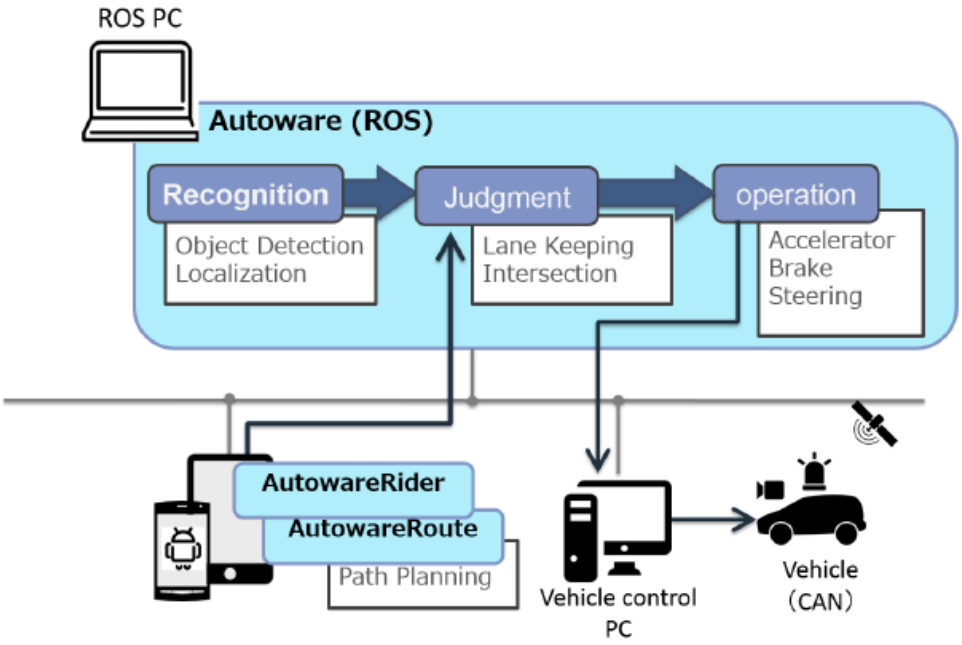
AUTOWARE – Θεωρητική προσέγγιση

2.1 Το AUTOWARE

Το Autoware είναι λογισμικό ανοιχτού κώδικα που βασίζεται στο ROS. Ο κώδικας του Autoware δημοσιεύεται στο Github. Το μεγαλύτερο μέρος του κώδικα χαρακτηρίζεται από:

- αναγνώριση (object recognition)
- κρίση (judgement)
- εκτέλεση εντολών (action)

Το Autoware παρέχει απαραίτητες λειτουργίες, όπως η δημιουργία χαρτών 3-D, εντοπισμός, αναγνώριση αντικειμένων και έλεγχος του οχήματος για αυτόνομη οδήγηση.



Εικόνα 14 : Autoware overview

Το Autoware χρησιμοποιεί LIDAR (Light Detection and Ranging), κάμερες και GNSS (Global Navigation Satellite System) στο όχημα για την «περιφερειακή του όραση». Μέσω αυτών μπορεί να εντοπίσει πεζούς, οχήματα, φωτεινούς σηματοδότες κ.λπ.

Αναλόγως τις λειτουργίες, η επεξεργασία αναθέτετε στην GPU ή στην CPU. Παραδείγματος χάριν: για να την αναγνώριση λωρίδας και διασταυρώσεων χρησιμοποιείται η GPU, ενώ για την διάγνωση ασφαλείας χρησιμοποιείται η CPU. Οι πληροφορίες του χώρου καθώς επίσης, τι συμπεριλαμβάνεται σ' αυτόν, μεταδίδεται με την μορφή 3-D χαρτών, τα οποία επεξεργάζονται με διάφορους αλγορίθμους που θα δούμε παρακάτω.

2.2 3-D Map Generation & Sharing

Το πιο αξιοσημείωτο χαρακτηριστικό του Autoware είναι το γεγονός ότι χρησιμοποιεί 3-D χάρτες. Ο χάρτης 3-D, σε αντίθεση με τους χάρτες 2-D που χρησιμοποιούνται σε ένα συμβατικό σύστημα πλοήγησης αυτοκινήτων, είναι ένας χάρτης που περιλαμβάνει πολύ περισσότερες πληροφορίες σχετικά με τον γύρο χώρο. Ο 3-D χάρτης παίζει κρίσιμο ρόλο στην ανάπτυξη αυτόνομης οδήγησης στον πραγματικό κόσμο. Το Autoware εντοπίζει τη θέση του οχήματος ταιριάζοντάς τον στον 3-D χάρτη που έχει φορτωθεί στον υπολογιστή με τις γύρω πληροφορίες που συλλέγονται από το LIDAR στην οροφή. Η ακρίβεια του εντοπισμού είναι περίπου 10cm. (αριθμός που αντιστοιχεί σε πολύ μεγαλύτερη ακρίβεια από το GPS)

Από τον Σεπτέμβριο του 2015, ο 3-D χάρτης καλύπτει μόνο την πλήρως απαραίτητη περιοχή που χρειάζεται να γνωρίζει το όχημα. Το όχημα μπορεί να εισέλθει σε μία περιοχή και το Autoware με τη σειρά του μπορεί να δημιουργήσει έναν 3-D χάρτη σε πραγματικό χρόνο από τις πληροφορίες που συλλέγει το LIDAR. Αυτός ο μηχανισμός επιτρέπει τη δημιουργία του 3-D χάρτη που περιλαμβάνει κάθε γωνιά της πόλης, όπως μικρά σοκάκια που δεν μπορούν να

εισέλθουν κάποια οχήματα. Επιπλέον, τα γεωγραφικά δεδομένα από τον 3-D χάρτη μπορούν μετά από χειροκίνητη επεξεργασία να δημιουργήσουν έναν διανυσματικό 3-D χάρτη.

2.3 Localization (NDT : Normal Distributions Transform)

Η θέση του οχήματος μπορεί να εντοπιστεί με scan matching με βάση τον αλγόριθμο NDT [14]. Ο αλγόριθμος NDT είναι ένας αλγόριθμος που χρησιμοποιεί τυποποιημένες τεχνικές βελτιστοποίησης από στατιστικά μοντέλα τρισδιάστατων σημείων για τον εντοπισμό και καταχώρηση μεταξύ δύο σημείων. Αυτόν τον εφαρμόζουμε στους 3-D χάρτες, που είναι της μορφής PCD (Point Cloud Data). Το σφάλμα θέσης εντοπισμού είναι περίπου 10cm.

2.4 Object Detection

Οι αλγόριθμοι DPM [16] (Deformable Part Models) μπορούν να ανιχνεύσουν αυτοκίνητα, πεζούς, και φανάρια από εικόνες κάμερας. Η παρακολούθηση με χρήση φίλτρου Kalman μπορεί να εφαρμοστεί και να βελτιώσει την ακρίβεια ανίχνευσης. Η συγχώνευση των δεδομένων 3-D LIDAR μπορεί να λάβει τις αποστάσεις των εντοπισμένων αντικειμένων.

2.5 Path Generation

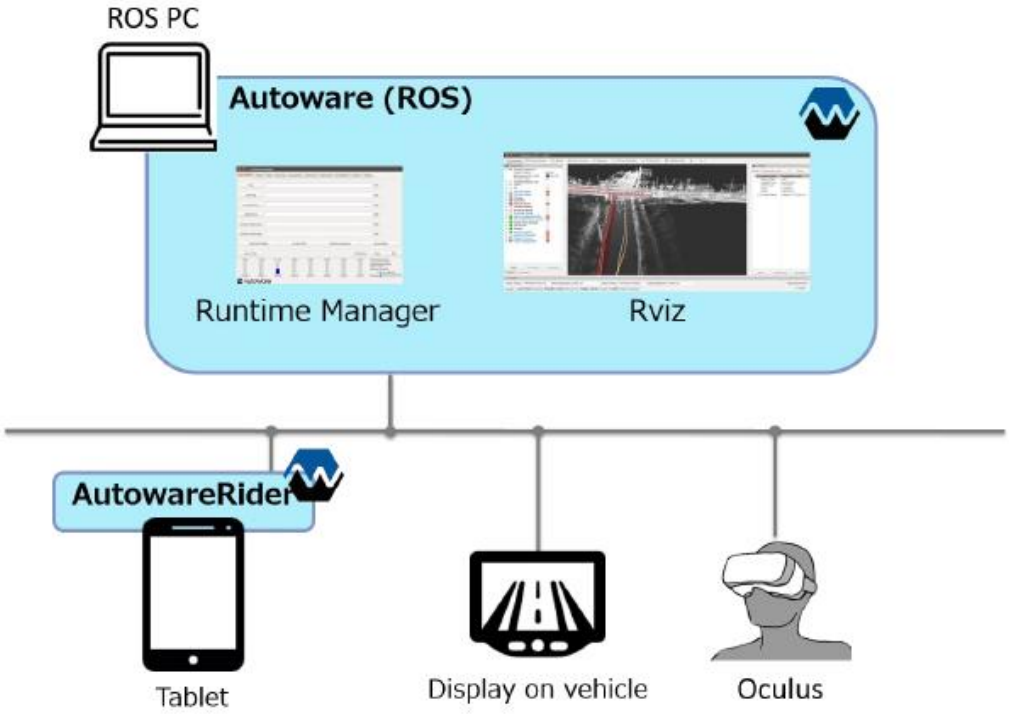
Το AutowareRoute (εφαρμογή δημιουργίας δεδομένων διαδρομής με χρήση του MapFan) δημιουργεί μια διαδρομή προς τον επιλεγμένο προορισμό. Στη συνέχεια, το αυτόνομο σύστημα οδήγησης καθορίζει τις διαθέσιμες λωρίδες για την υπολογισμένη διαδρομή. Οι τροχιές στις λωρίδες υπολογίζονται κινηματικά.

2.6 Autonomous Driving

Η παραγόμενη διαδρομή περιλαμβάνει τις κατάλληλες πληροφορίες ταχύτητας. Το αυτόνομο σύστημα οδήγησης χρησιμοποιεί αυτή την πληροφορία της ταχύτητας ως στόχο. Επιπλέον, η διαδρομή περιλαμβάνει landmarks, “way point” ,σε διαστήματα 1m. Το αυτόνομο σύστημα οδήγησης υλοποιεί το “path following” ακολουθώντας τα σημεία της διαδρομής. Εάν το όχημα αποκλίνει από τη διαδρομή, το σύστημα στοχεύει στο σημείο γειτνίασης και μετά πάλι πίσω στη στοχευμένη διαδρομή.

2.7 User Interface

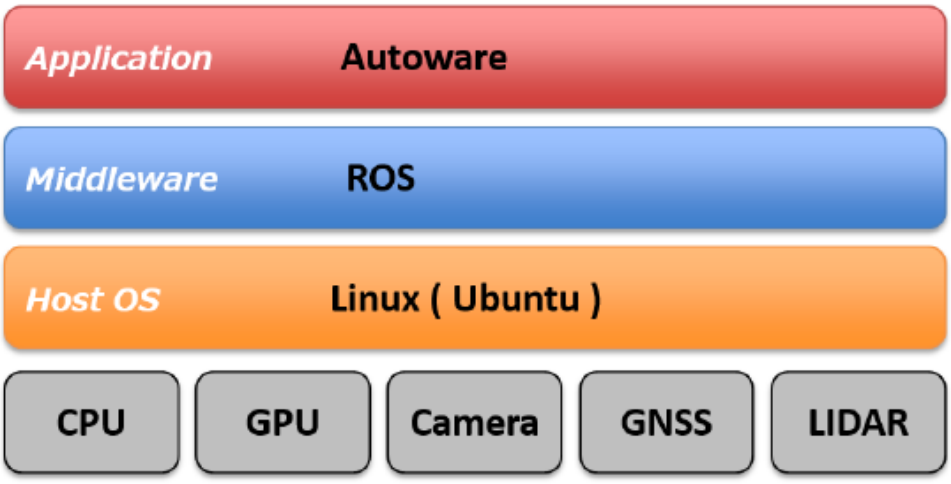
Ο "Runtime Manager" του Autoware επιτρέπει στους χρήστες να ελέγχουν και να μεταβάλλουν διάφορες καταστάσεις του προγράμματος όπως: εντοπισμός, ανίχνευση αντικειμένων και παρακολούθηση διαδρομών κτλ. Επιπλέον, το RViz μπορεί να ενσωματώσει και να απεικονίσει τον εντοπισμό σε χάρτη 3-D, την ανίχνευση αντικειμένων, τον σχεδιασμό διαδρομής και την ακολουθία διαδρομής. Επιπλέον, το "AutowareRider" του Autoware, επιτρέπει την εύκολη πλοήγηση, τον προγραμματισμό διαδρομής, τη μετάβαση σε αυτόνομο τρόπο οδήγησης κ.λπ. Επιπλέον, το Autoware μπορεί να απεικονίσει τον 3-D χάρτη που χρησιμοποιείται σε αυτόνομο σύστημα οδήγησης στις οθόνες του οχήματος και σε συσκευές Oculus.



Εικόνα 15 : User interface

2.8 Platform structure for Autoware

Το Autoware είναι μία εφαρμογή που χρησιμοποιεί το ROS. Οπότε μπορεί να εγκατασταθεί μόνο σε Linux.



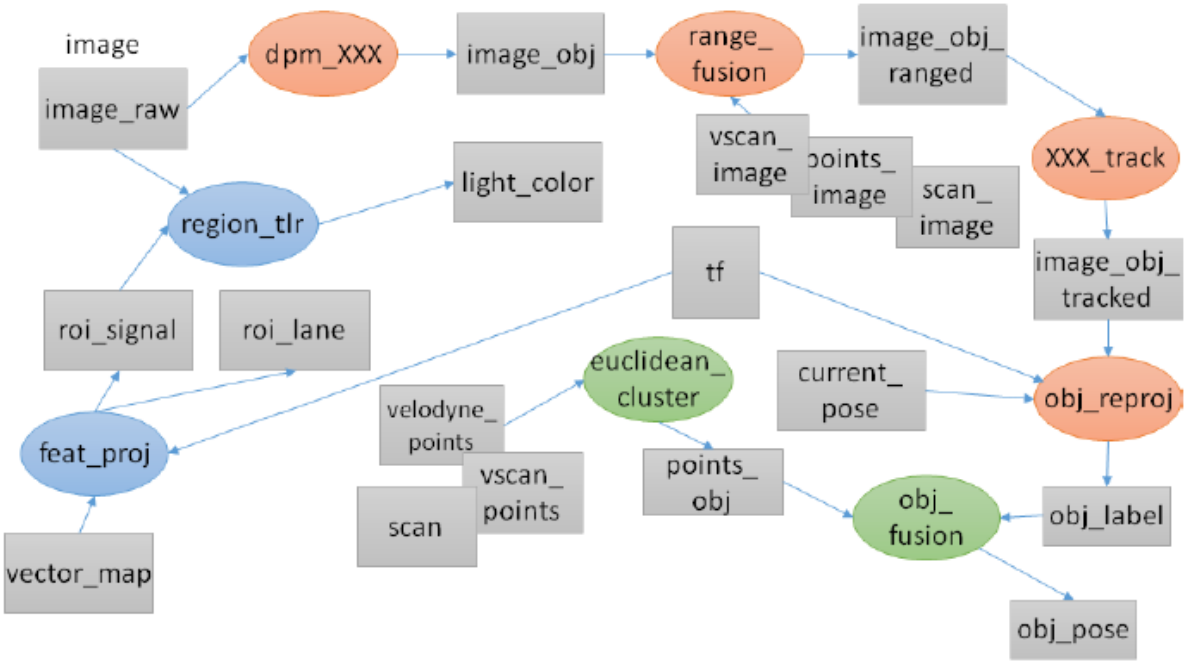
Εικόνα 16 : Πλατφόρμα καταστευής του Autoware

2.9 Perception/Recognition

Εγκαθιστώντας το πρόγραμμα, δημιουργούνται διάφορα πακέτα που χρειαζόνται για την πλήρη και σωστή λειτουργία του Autoware. Παρακάτω βλέπουμε το directory της κάθε λειτουργίας και πως λειτουργεί σχηματικά:

Perception/Recognition	
Directory	/Autoware/ros/src/computing/perception/detection
Used by/for	road_wizard,cv_tracker , lidar_tracker

Πίνακας 1: Perception/Recognition

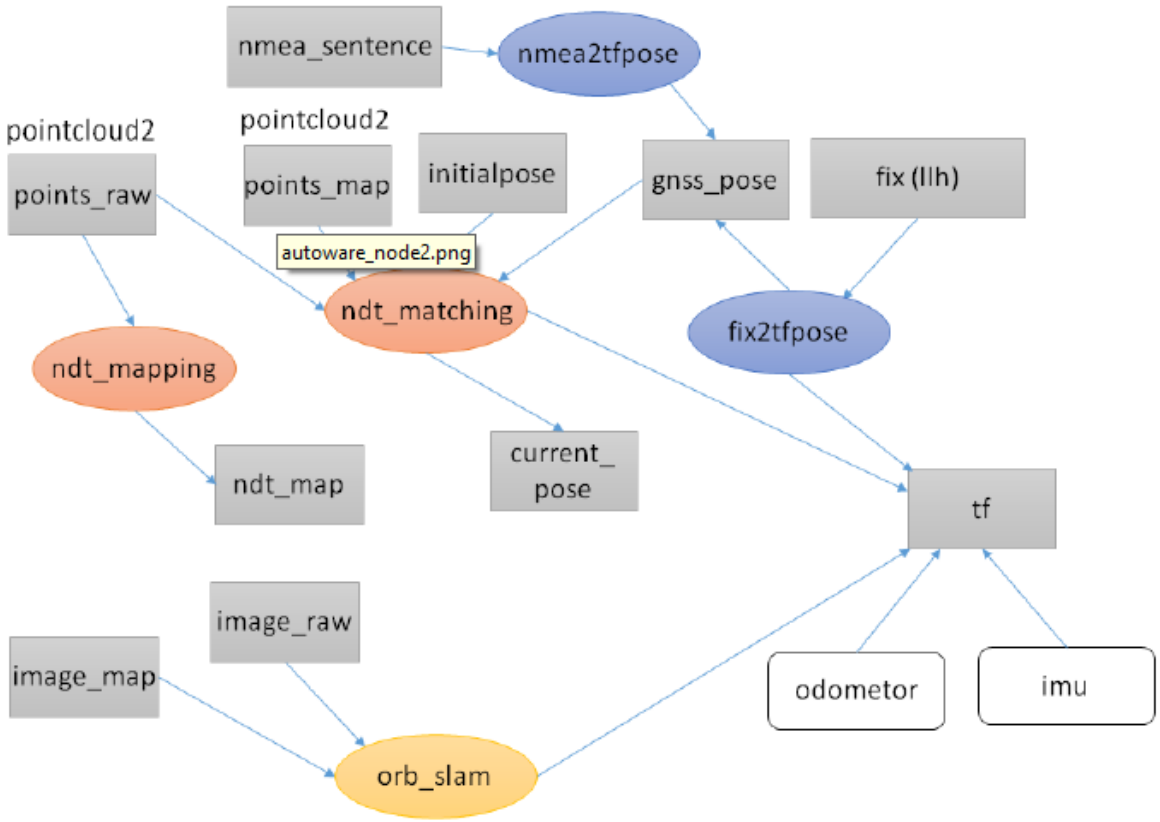


Εικόνα 17 : Perception & recognition

Με βάση το παραπάνω διάγραμμα βλέπουμε όλες τις πληροφορίες που χρειάζεται να ληφθούν υπ όψιν (road_wizard,cv_tracker , lidar_tracker) προκειμένου να υπολογιστεί η ακριβής θέση του οχήματος (obj_pose). Επίσης, βλέπουμε στο διάγραμμα, την ακολουθία που τηρείται για να καταλήξουμε στην ακριβής θέση του obj_pose.

Directory	/Autoware/ros/src/computing/perception/localization
Used by/for	ndt_localizer, orb_localizer, gnss_localizer

Πίνακας 2 Judgement/Operation/Localization

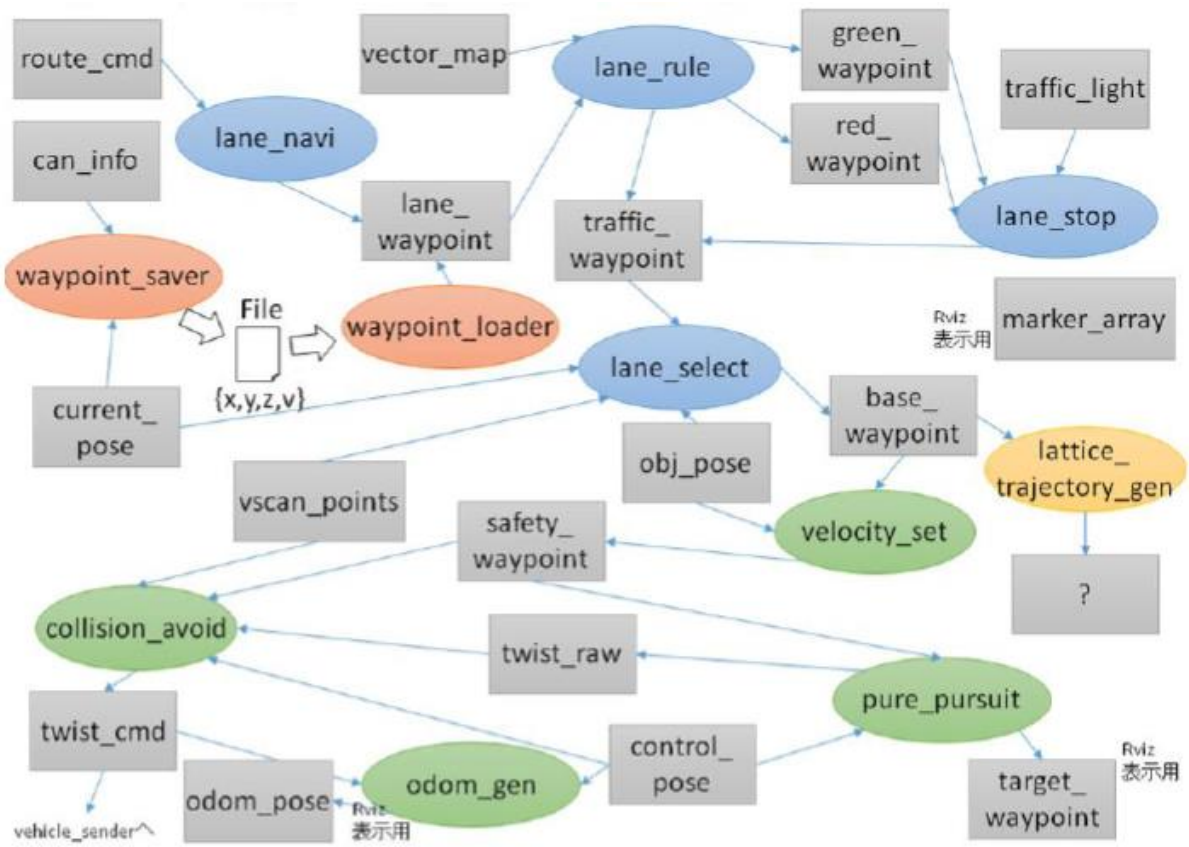


Εικόνα 18 : Judgment, operation and localization

Με βάση το παραπάνω διάγραμμα βλέπουμε όλες τις πληροφορίες που χρειάζεται να ληφθούν υπ όψιν (ndt_localizer, orb_localizer, gnss_localizer) προκειμένου να υπολογιστεί η ακριβής θέση του συστήματος συντεταγμένων (tf). Η πληροφορία αυτή χρειάζεται για να επαληθεύεται η θέση του οχήματος με τον δορυφόρο GPS. Επίσης, βλέπουμε στο διάγραμμα, την ακολουθία που τηρείται για να καταλήξουμε στην ακριβής θέση του tf.

Path Planning	
Directory	/Autoware/ros/src/computing/planning
Used by/for	lane_planner, driving_planner, waypoint_maker, waypoint_follower

Πίνακας 3 Path Planning

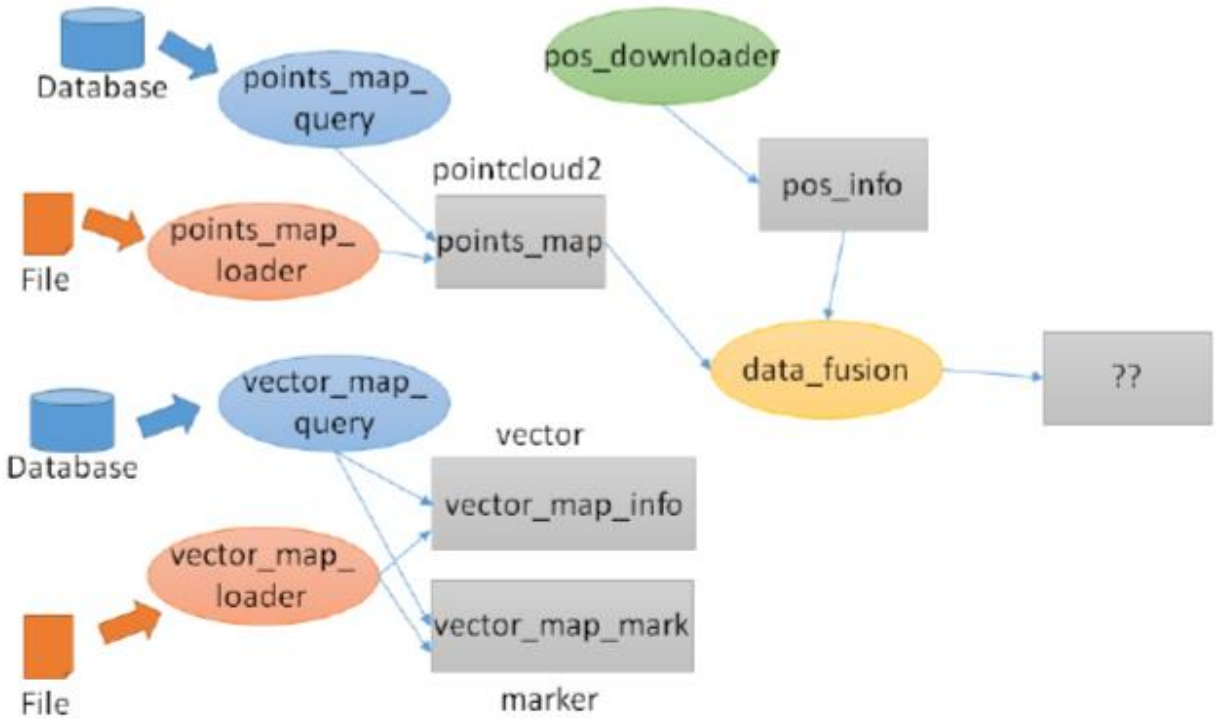


Εικόνα 19 : Path planning

Με βάση το παραπάνω διάγραμμα βλέπουμε όλες τις πληροφορίες που χρειάζεται να ληφθούν υπ όψιν (lane_planner, driving_planner, waypoint_maker, waypoint_follower) προκειμένου να υπολογιστεί το pose της οδομετρίας (odom_pose). Εάν έχουμε pure_pursuit υπολογίζεται το αντίστοιχο target_waypoint. Εάν το όχημα βρίσκεται σε κατάσταση αποφυγής εμποδίου υπολογίζεται νέα διαδρομή (twist_cmd).

Data Loading (3-D Map, Database, Files)	
Directory	/Autoware/ros/src/data
Used by/for	map_db, map_file, pos_db, dynamic_map

Πίνακας 4 : Data Loading (3-D Map, Database, Files)

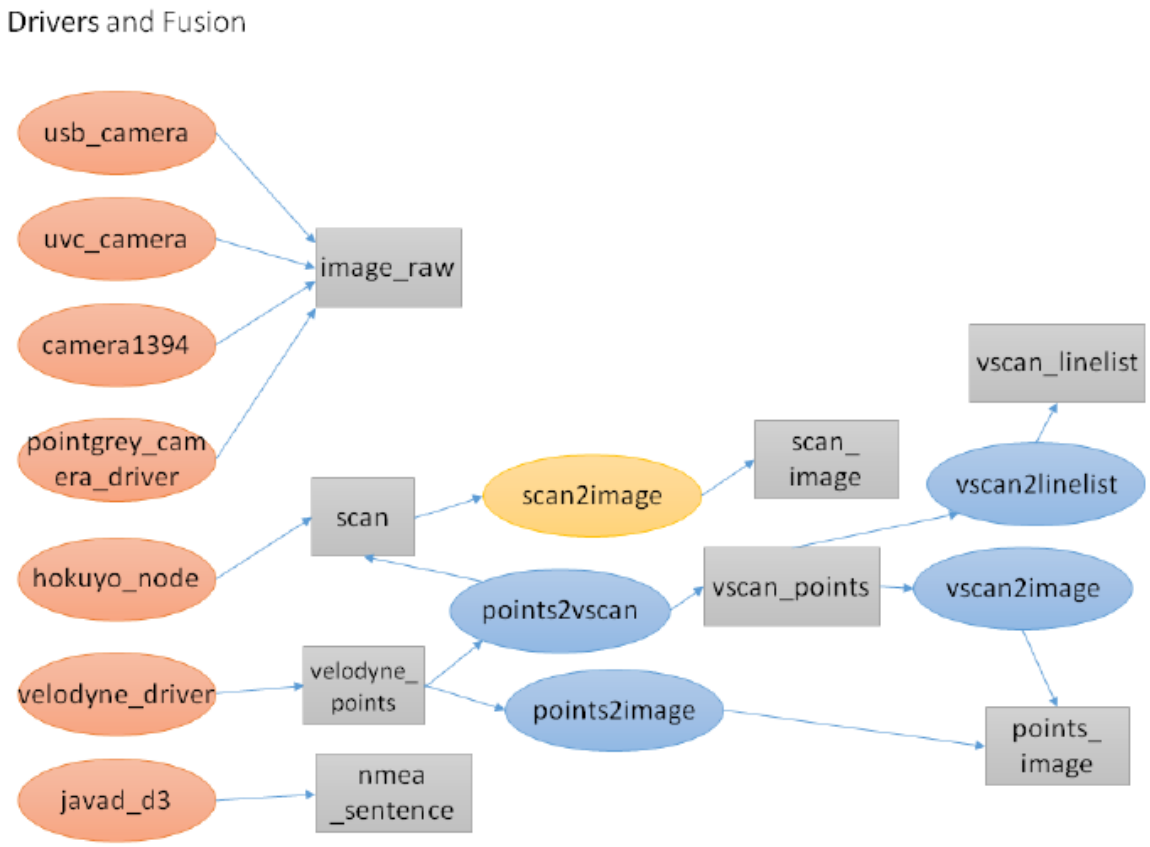


Εικόνα 20 : Data Loading

Με βάση το παραπάνω διάγραμμα βλέπουμε πως επεξεργάζονται τα point cloud data και πως επεξεργάζονται τα vector data.

Device Drivers and Sensor Fusion	
Directory	/Autoware/ros/src/sensing/drivers & ros/src/sensing/fusion
Used by/for	-

Πίνακας 5 Device Drivers and Sensor Fusion

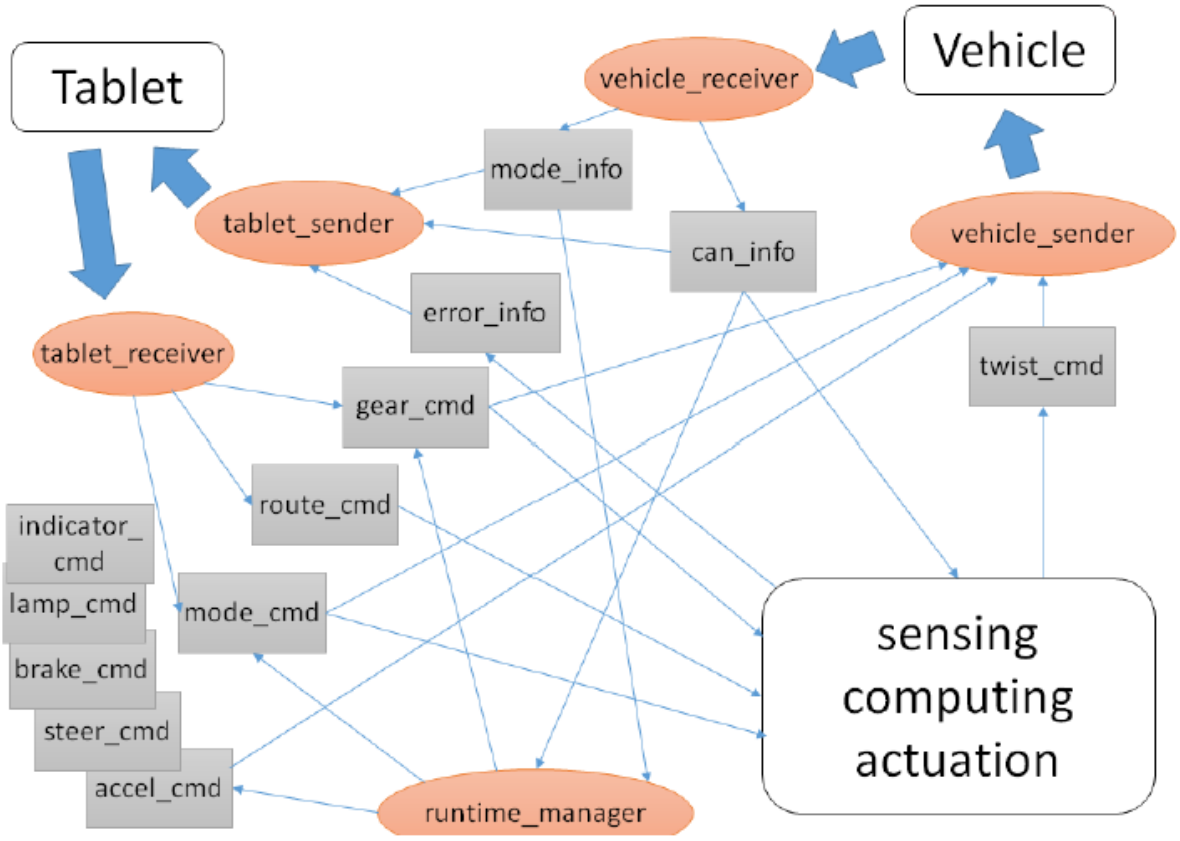


Εικόνα 21 : Device drivers and sensor fusion

Με βάση το παραπάνω διάγραμμα βλέπουμε ότι η camera (usb_camera, uvc_camera, camera1394, point_camera-driver) παράγει την εικόνα του γύρω χώρου και με το laserscanner (hokuyo_node, velodyne_driver, javad_d3) πραγματοποιείται το σκανάρισμα του χώρου.

Interface for Smart Phone Applications	
Directory	/Autoware/ros/src/socket
Used by/for	-

Πίνακας 6 Interface for Smart Phone Applications



Εικόνα 22 : Interface for smart phone applications

Στο παραπάνω διάγραμμα βλέπουμε την ακολουθία των πληροφοριών που έχουμε μεταξύ του οχήματος, του υπολογιστή (laptop ή tablet) και του software.

Utilities and Others	
Directory	/Autoware/ros/src/util/
Directory	/Autoware/ui/tablet/
Directory	/Autoware/vehicle/

Πίνακας 7 Utilities and Others

Κεφάλαιο 3ο

Εγκατάσταση του Autoware

Το Autoware είναι ένα αρκετά απαιτητικό λογισμικό, από πλευράς hardware, λόγο του πολύ υψηλού επιπέδου υπολογισμών που έχει να κάνει on-the-fly καθώς τρέχουμε ένα simulation ή ακόμα και ότι καταγράφουμε πειραματικά δεδομένα για να τα επεξεργαστούμε αργότερα. Σύμφωνα με τον κατασκευαστή τα προτεινόμενα minimum requirements είναι τα παρακάτω:

- Intel Core i7 (preferred), Core i5, Atom
- 16GB to 32GB of main memory
- More than 30GB of SSD
- NVIDIA GTX GeForce GPU (980M or higher performance)

Όπως έχουμε προαναφέρει το λογισμικό μπορεί να εγκατασταθεί μόνο σε Ubuntu και κατά προτίμηση την έκδοση 16.04.

Θα πρέπει να εγκατασταθεί σε Physical Machine, όχι σε Virtual Machine. Πέρα από τις προσομοιώσεις, ακόμα και στην εγκατάσταση καλούνται οι drivers της nvidia και άρα πρέπει να υπάρχει πρόσβαση στην πραγματική κάρτα γραφικών

3.1 Τρόπος εγκατάστασης

Υπάρχουν δύο τρόποι εγκατάστασης (https://autoware.ai/). Πριν όμως ξεκινήσουμε την εγκατάσταση του Autoware θα πρέπει πρώτα να εγκαταστήσουμε και κάποια άλλα προαπαιτούμενα πακέτα που χρειάζονται στο σύνολο της διαδικασίας.

OpenCV 2.4.10 or higher	Υποχρεωτικό
Qt 5.2.1 or higher	Υποχρεωτικό
CUDA	Υποχρεωτικό (latest)
FlyCapture2	Επιλογής
Armadillo	Επιλογής

Πίνακας 8 : Προαπαιτούμενα πακέτα

Τώρα, το Autoware μπορεί να εγκατασταθεί με δύο τρόπους.

Με την βοήθεια του εργαλείου **Docker** ή με **catkin_make**

3.2 Εντολές εγκατάστασης του Autoware με Docker

```
sudo apt-get remove docker docker-engine docker.io
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
sudo apt-key fingerprint 0EBFCD88
```

pub 4096R/0EBFCD88 2017-02-22 *[Report message]*

Key fingerprint = 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

uid Docker Release (CE deb)

sub 4096R/F273FCD8 2017-02-22

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce
sudo docker run hello-world
wget https://github.com/NVIDIA/nvidia-docker/releases/download/v1.0.1/nvidia-docker_1.0.1-1_amd64.deb

sudo dpkg -i nvidia-docker_1.0.1-1_amd64.deb
systemctl list-units --type=service | grep -i nvidia-docker
sudo nvidia-docker run --rm nvidia/cuda nvidia-smi
git clone https://github.com/CPFL/Autoware.git

cd Autoware/docker/generic

Ανοίγουμε το αρχείο run.sh και την αλλάζουμε από

autoware-$1 *σε* ***autoware/autoware:1.7.0-kinetic***

Αλλάζουμε την γραμμή, δεν κάνουμε comment απλά

sudo sh run.sh kinetic

*Autoware με Docker (*https://github.com/CPFL/Autoware/wiki/Generic-x86-Docker

3.3 Εντολές εγκατάστασης του Autoware με common installation

sudo apt-get update
sudo apt-get install -y python-catkin-pkg python-rosdep python-wstool ros-$ROS_DISTRO-catkin libmosquitto-dev
cd $HOME
git clone https://github.com/CPFL/Autoware.git --recurse-submodules
cd ~/Autoware/ros/src

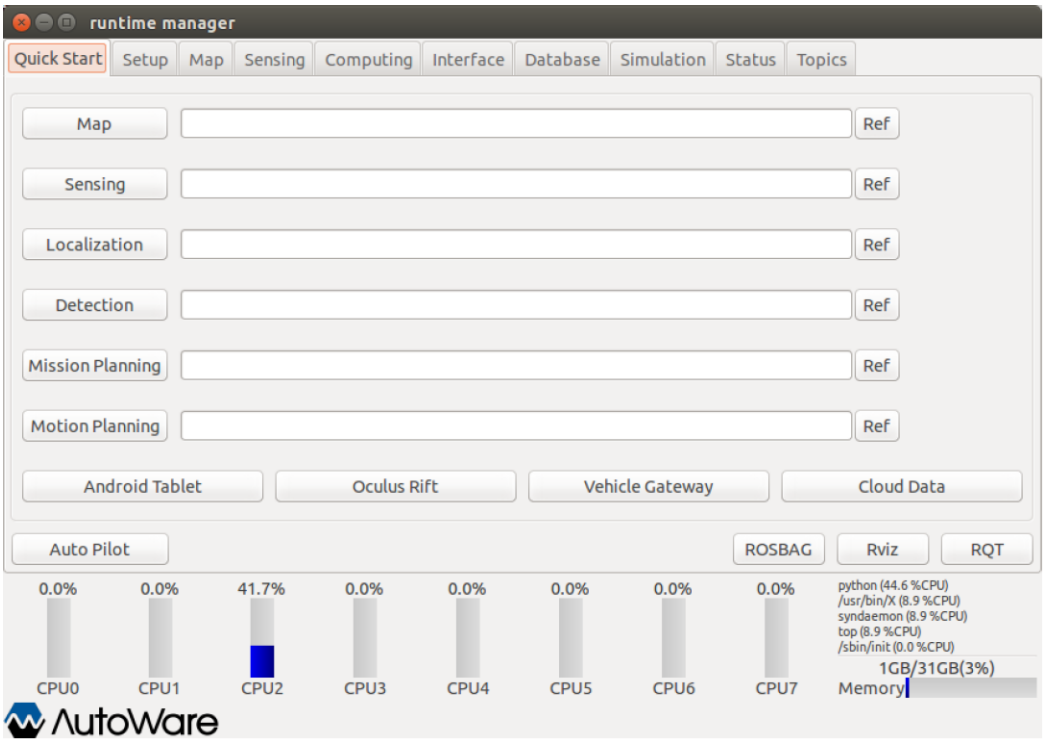
catkin_init_workspace
cd ../
rosdep install -y --from-paths src --ignore-src --rosdistro $ROS_DISTRO
./catkin_make_release

*Autoware με source (*https://github.com/CPFL/Autoware/wiki/Source-Build*)*

3.4 Περιγραφή του Runtime Manager

Runtime Manager - Quick Start tab

Στο πρώτο tab του Runtime Manager κατά βάση μπορεί ο χρήστης να ορίσει όλα τα απαραίτητα αρχεία που χρειάζεται για να τρέξει ένα simulation. Όλα τα αρχεία που θα εισαχθούν εδώ είναι τύπου *.launch και έχουν *.xml format. Παρακάτω βλέπουμε αναλυτικά τι χρειάζεται να ορίσουμε σε κάθε πεδίο.



Εικόνα 23 : Runtime Manager - Quick Start Tab

Map: Επιλογή ενός map launch αρχείου από το παράθυρο διαλόγου, πατώντας το κουμπί [Ref]. Επίσης, φόρτωση αυτού του αρχείου στο simulation
Sensing: Επιλογή ενός sensing launch αρχείου από το παράθυρο διαλόγου, πατώντας το κουμπί [Ref]. Επίσης, φόρτωση αυτού του αρχείου στο simulation.

Localization : Επιλογή ενός localization launch αρχείου από το παράθυρο διαλόγου, πατώντας το κουμπί [Ref]. Επίσης, φόρτωση αυτού του αρχείου στο simulation.

Detection : Επιλογή ενός detection launch αρχείου από το παράθυρο διαλόγου, πατώντας το κουμπί [Ref]. Επίσης, φόρτωση αυτού του αρχείου στο simulation.

Mission Planning: Επιλογή ενός mission planning launch αρχείου από το παράθυρο διαλόγου, πατώντας το κουμπί [Ref]. Επίσης, φόρτωση αυτού του αρχείου στο simulation.

Motion Planning : Επιλογή ενός motion planning launch αρχείου από το παράθυρο διαλόγου, πατώντας το κουμπί [Ref]. Επίσης, φόρτωση αυτού του αρχείου στο simulation.

**Πιο αναλυτικά θα δούμε τα παραπάνω πεδία στο κεφάλαιο 6*

Android Tablet : Start / End του script runtime_manager/tablet_socket.launch για επικοινωνία με τα Tablets.

Oculus Rift : Προσεχώς

Vehicle Gateaway : Start/end το script runtime_manager/vehicle_socket.launch για επικοινωνία με ZMP Robocar.

Cloud Data : Start/end των obj_db/obj_downloader nodes

Auto Pilot : Publish /mode_cmd topics σύμφωνα με το status των buttons.

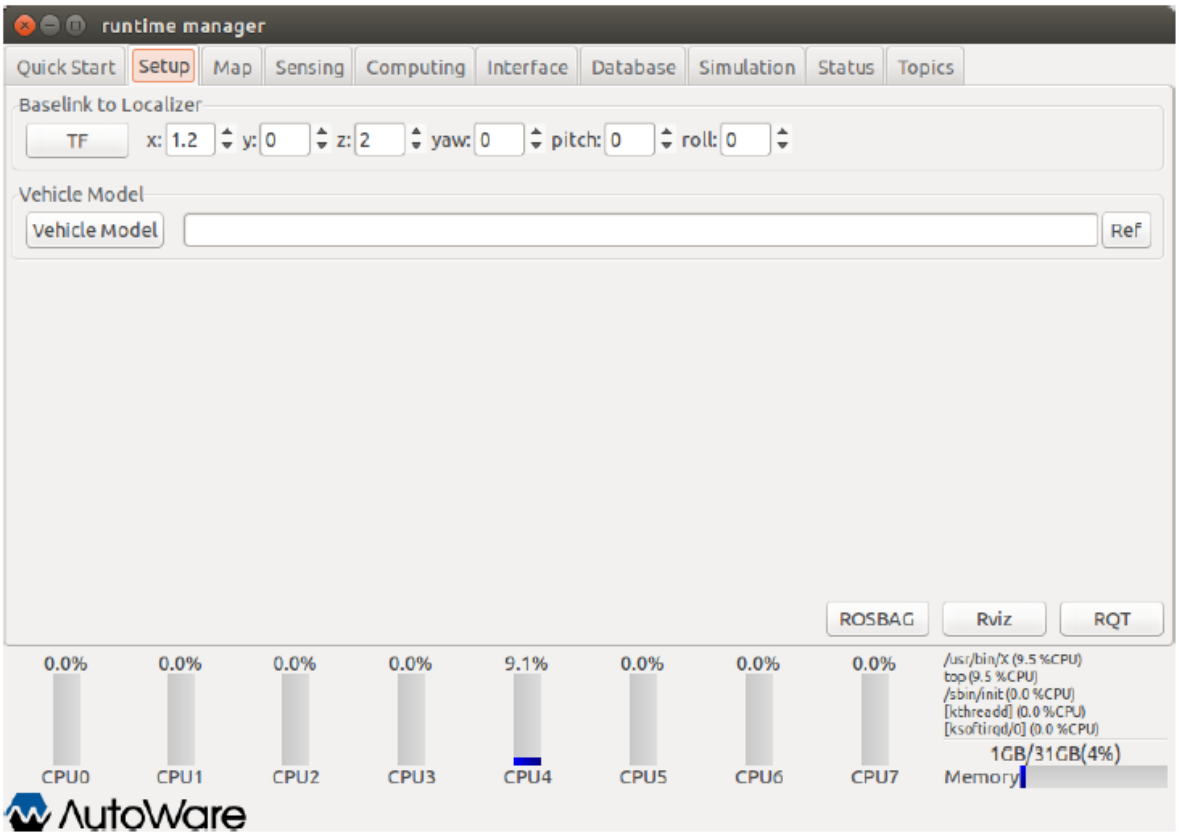
ROSBAG : προβάλει το ROSBAG Record παράθυρο διαλόγου του ROSBAG

RViz : Start/End RViz

RQT : Start/End Rqt

Runtime Manager – Setup Tab

Στο δεύτερο tab του Runtime Manager μπορεί ο χρήστης αρχικές πληροφορίες για το σύστημα συντεταγμένων καθώς επίσης μα φορτώσει ξεχωριστά οχήματα στο current simulation που τρέχει. Το είδος του αρχείου που μπορούμε να φορτώσουμε είναι *.yaml. Παρακάτω βλέπουμε αναλυτικά τι χρειάζεται να ορίσουμε σε κάθε πεδίο.

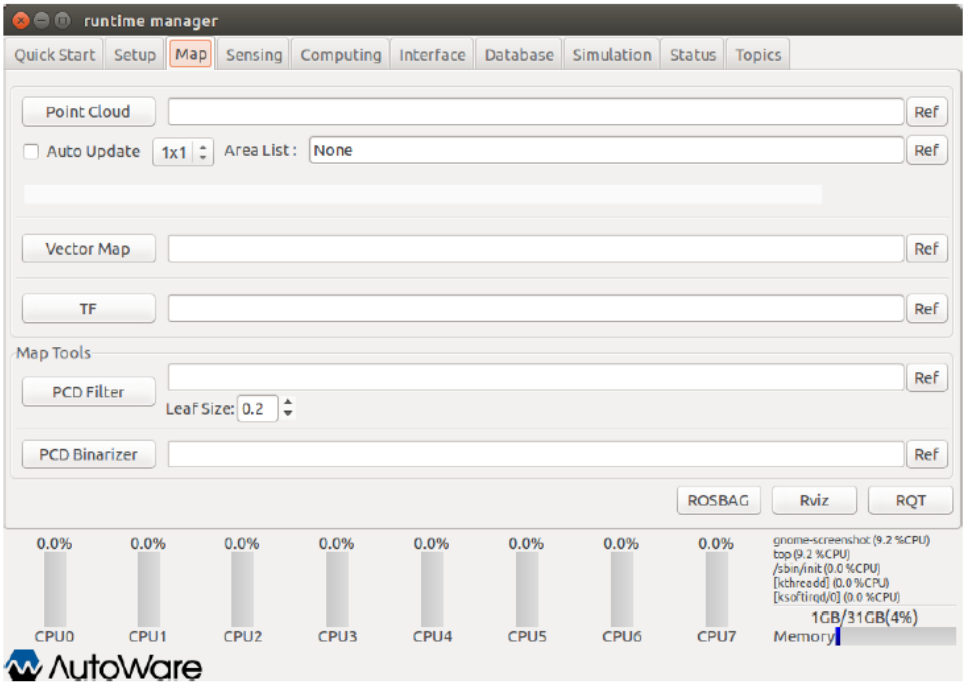


Εικόνα 24 : Runtime Manager - Setup Tab

TF : πατώντως το TF κουμπί ένα topic Vehicle control position θα γίνει publish.
x, y, z, yaw, pitch, roll : εισάγουμε την θέση του velodyne στο base_link
Vehicle Model : Start/end ένα script model_publisher/vehicle_model.launch. Με αυτό το αρχείο ένα πλήρες όχημα προβάλεται στο RViz
ROSBAG : Προβολή του ROSBAG Record.
RViz : Start/End RViz
RQT : Start/End Rqt

Runtime Manager – Map Tab

Στο τρίτο tab του Runtime Manager κατά βάση μπορεί ο χρήστης να ορίσει όλα τα απαραίτητα αρχεία που χρειάζεται για να τρέξει ένα manual simulation. Σε σχέση με το πρώτο tab, εδώ ο χρήστης θα πρέπει να ορίσει όλες τις παραμέτρους χειροκίνητα πχ. ταχύτητα, διαδρομή που θα ακολουθήσει κτλ. Σ' αυτό το tab ορίζουμε δεδομένα όπως point cloud data, vector data, tf data κτλ. Στα επόμενα tabs θα δούμε πως ορίζονται και τα υπόλοιπα δεδομένα του simulation.



Εικόνα 25 : Runtime Manager - Map Tab

Point Cloud Start/end το map_file/points_map_loader όπου είναι το full path του 3-D map (PCD files) Εναλλακτικά, ένας 3-D map μπορεί να επιλεγεί πατώντας **Ref**.
Auto Update : Αυτόματο update status (ενεργοποίηση ή όχι) των launching map_file/points_map_loader. Ο αριθμός των σκηνών που θα λαμβάνουν update μπορεί να δηλωθεί επιλέγοντας items στο drop box. Αυτό το configuration επιδρά μόνο όταν το Auto Update είναι on. Δήλωση του map_file/point_map_loader. Ref.
Vector Map : Start/end ενός map_file/vector_map_loader nodes (csv format)
TF : Δήλωση αρχείου με TF. Αν δεν δηλωθεί θα χρησιμοποιηθεί το Autoware/data/tf/tf.launch

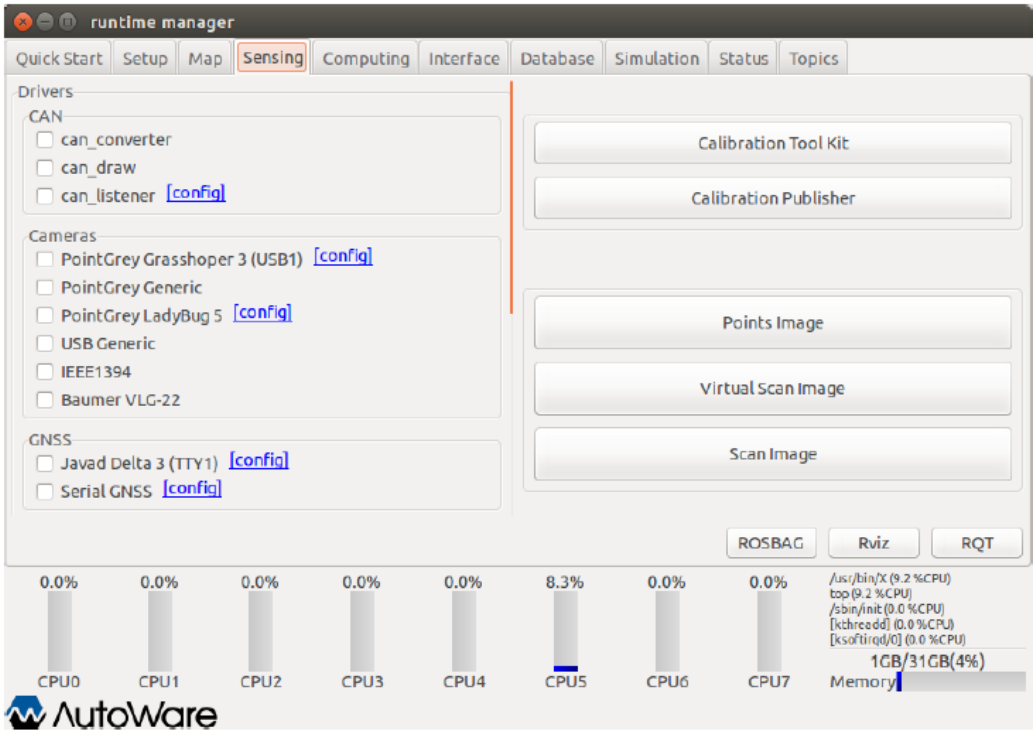
Map Tools

PCD Filter: Start/end map_tools/pcd_filter nodes όπου είναι το full path του 3-D map (PCD files).
PCD Binarizer: Start/end map_tools/pcd_binalizer nodes όπου είναι το full path ενός ASCII format PCD file.

ROSBAG : προβάλει το ROSBAG Record παράθυρο διαλόγου του ROSBAG
RViz : Start/End RViz
RQT : Start/End Rqt

Runtime Manager – Sensing Tab

Το tab Sensing το χρησιμοποιούμαι όταν θέλουμε να συνδέσουμε το software με το όχημα. Προκειμένου να επικοινωνήσουμε το σύστημα CAN του οχήματος ,τους αισθητήρες κτλ.



Εικόνα 26 : Runtime Manager - Sensing Tab

Drivers

CAN

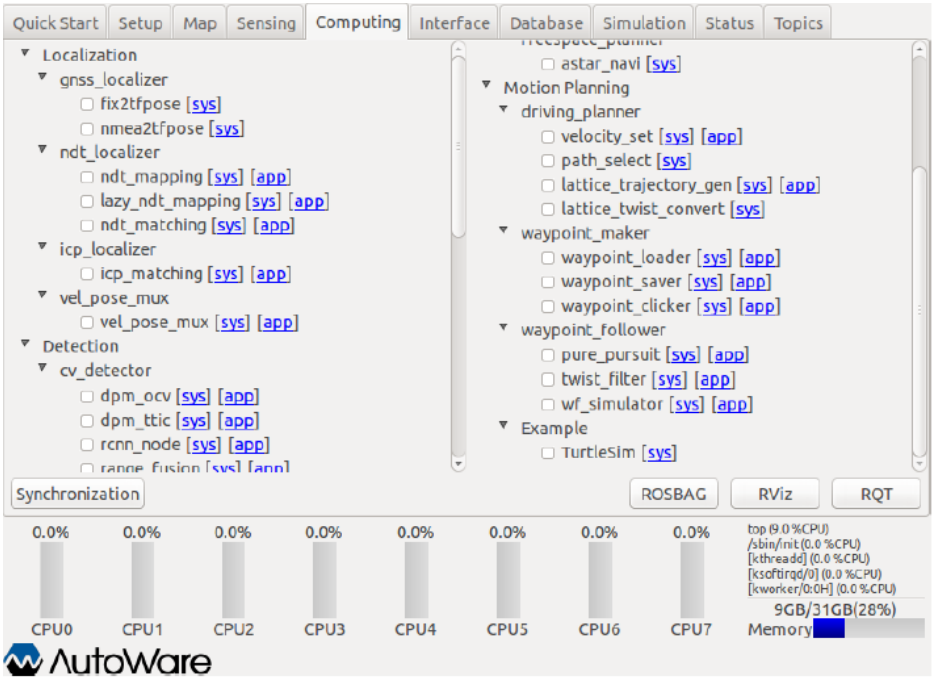
can_converter : Start/end kvaser/can_converter nodes.
can_draw : Start/end kvaser/can_draw nodes.
can_listener : Start/end kvaser/can_listener nodes.
can_listener : Προβάλει έναν can_listener dialog. Δήλωση των καναλιών των launching nodes.
Cameras

PointGrey Grasshoper 3 (USB1) : Start/end ένα pointgrey/grasshopper3.launch script.
PointGrey Grasshoper 3 (USB1): Προβάλει το calibration_path_grasshopper3 dialog.
PointGrey Generic: Start/end ένα pointgrey_camera_driver/camera.launch script.
PointGrey PointGray LadyBug 5: Start/end ένα pointgrey/ladybug.launch script.
PointGrey PointGray LadyBug 5: Προβολή ενός calibration_path_ladybug dialog
USB Generic : Start/end runtime_manager/uvc_camera_node
IEEE1394 :Προσεχώς
Baumer VLG-22: Start/end ένα vlg22c_cam/baumer.launch script.

GNSS
Javad Delta 3(TTY1) : Start/end ενός javad_navsat_driver/gnss.sh script.
Javad Delta 3(TTY1) : Προβάλει ένα serial dialog. Δήλωση παραμέτρων του RS232C.
Serial GNSS: Start/end ενός nmea_navsat/nmea_navsat.launch script.
Serial GNSS : Προβάλει ένα serial dialog. Δήλωση παραμέτρων RS232C.
IMU
Crossbow vg440: Προσεχώς
LIDARs
Velodyne HDL-64e-S2: Start/end ένα velodyne_pointcloud/velodyne_hdl64e_s2.launch script.
Velodyne HDL-64e-S2: προβολή ενός calibration_path.
Velodyne HDL-64e-S3 : Start/end ένα velodyne_pointcloud/velodyne_hdl64e_s3.launch script.
Velodyne HDL-64e-S3: προβολή ενός calibration_path.
Velodyne HDL-32e : Start/end ενός velodyne_pointcloud/velodyne_hdl32e.launch script.
Velodyne HDL-32e : προβολή ενός calibration_path
Velodyne VLP-16 : Start/end ενός velodyne_pointcloud/velodyne _vlp16.launch scripts.
Velodyne VLP-16: : προβολή ενός calibration_path
Hokuyo TOP-URG: Start/end a hokuyo/top_urg.launch script.
Hokuyo 3D-URG : Start/end ένα hokuyo/hokuyo_3d_urg.launch script.
SICK LMS511: Προσεχώς
IBEO 8L Single: Προσεχώς

Points Filter
Calibration Tool Kit: Start/end Calibration_camera_lidar/calibration_toolkit nodes
Calibration Publisher: Start/end του runtime_manager calibration_publisher.launch script.
Points Image : Start/end runtime_manager/points2image.launch script.
Virtual Scan Image: Start/end runtime_manager/vscan.launch script.
Scan Image : Start/end scan2image/scan2image nodes
ROSBAG : προβάλει το ROSBAG Record παράθυρο διαλόγου του ROSBAG
RViz : Start/End RViz
RQT : Start/End Rqt
Runtime Manager – Computing Tab

Στο tab Computing ορίζομε όλες τις απαραίτητες παραμέτρους που χρειαζόμαστε όταν τρέχουμε manual simulation (tab Setup).



Εικόνα 27 : Runtime Manager - Computing Tab

Localization
gnss_lozalizer
fix2tfpose : Start/end gnss_localizer/fix2tfpose nodes
nmea2tfpose : Start/end gnss_localizer/nmea2tfpose.launch script.

ndt_lozalizer
ndt_mapping: Start/end ndt_localizer/ndt_mapping.launch script.
ndt_matching : Start/end ndt_localizer/ndt_matching.launch script.
vel_pose_mux
vel_pose_mux : Start/end vel_pose_mux/vel_pose_mux.launch script.

Detection
cv_detector
dpm_ocv : Start/end ένα cv_tracker/dpm_ocv.launch script.
dpm_ttic : Start/end ένα cv_tracker/dpm_ttic.launch script.
rcnn_node : Start/end ένα cv_tracker/rcnn.launch script.
range_fusion: Start/end ένα cv_tracker/ranging.launch script.
klt_track: Start/end ένα cv_tracker/klt_tracking.launch script.
kf_track : Start/end ένα cv_tracker/kf_tracking.launch script.

obj_reproj: Start/end ένα cv_tracker/reprojection.launch script.

lidar_detector
euclidean_cluster : Start/end ένα lidar_tracker/euclidean_clustering.launch scritp.
obj_fusion : Start/end ένα lidar_tracker/obj_fusion.launch script
road_wizard
feat_proj : Start/end ένα road_wizard/feat_proj.launch script.
region_tlr : Start/end ένα road_wizard/traffic_light_recognition.launch script.

Viewers
image_viewer: Start/end ένα viewers/viewers.launch script.
image_d_viewer : Start/end ένα viewers/viewers.launch script.
points_image_viewer :Start/end ένα viewers/viewers.launch script.
points_image_d_viewer : Start/end ένα viewers/viewers.launch script.
vscan_image_viewer : Start/end a viewers/viewers.launch script.
vscan_image_d_viewer : Start/end a viewers/viewers.launch script
traffic_light_viewer: Start/end ένα viewers/viewers.launch script.

Semantics
laserscan2costmap : Start/end ένα object_map/laserscan2costmap.launch script.
points2costmap : Start/end ένα object_map/points2costmap.launch script.

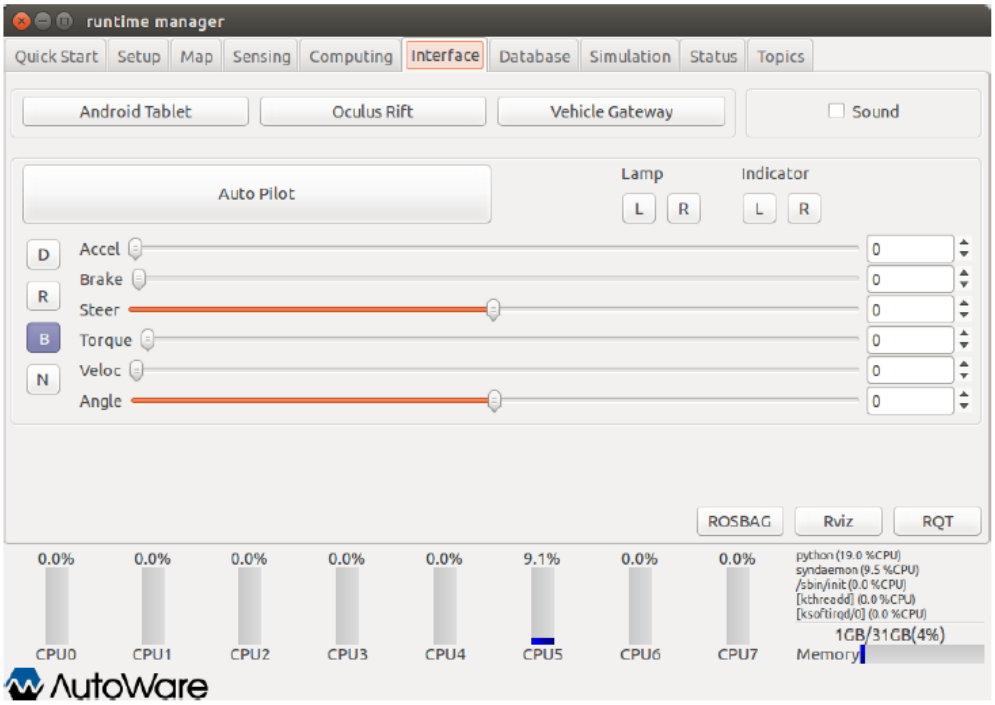
Mission Planning
lane_planner
lane_navi : Start/end ένα lane_planner/lane_navi.launch script.
lane_rule : Start/end ένα lane_planner/lane_rule script.
lane_stop : Start/end ένα lane_planner/lane_stop script.
lane_select : Start/end ένα lane_planner/lane_script.
freespace_planner
astar_navi: Start/end ένα freespace_planner/astar_navi script.
Motion Planning
driving_planner
velocity_set : Start/end ένα driving_planner/velocity_set.launch script.
path_select: Start/end ένα driving_planner/path_select script.
lattice_trajectory_gen: Start/end ένα driving_planner/lattice_trajectory_gen.launch script.
lattice_twist_convert : Start/end ένα driving_planner/lattice_twist_convert nodes.

waypoint_marker
waypoint_loader: Start/end ένα waypoint_maker/waypoint_loader.launch script.
waypoint_saver : Start/end ένα waypoint_maker/waypoint_saver.launch script.
waypoint_clicker : Start/end ένα waypoint_maker/waypoint_clicker nodes.

waypoint_follower
pure_pursuit : Start/end ένα waypoint_follower/pure_pursuit.launch script.
twist_filter : Start/end ένα waypoint_follower/twist_filter.launch script.
wf_simulator : Start/end ένα waypoint_follower/wf_simulator.launch script.

Runtime Manager – Interface Tab

Στο Interface tab έχουμε διάφορες επιλογές να διαχειριστούμε συσκευές όπως : tablet, VR γυαλιά κλτ.



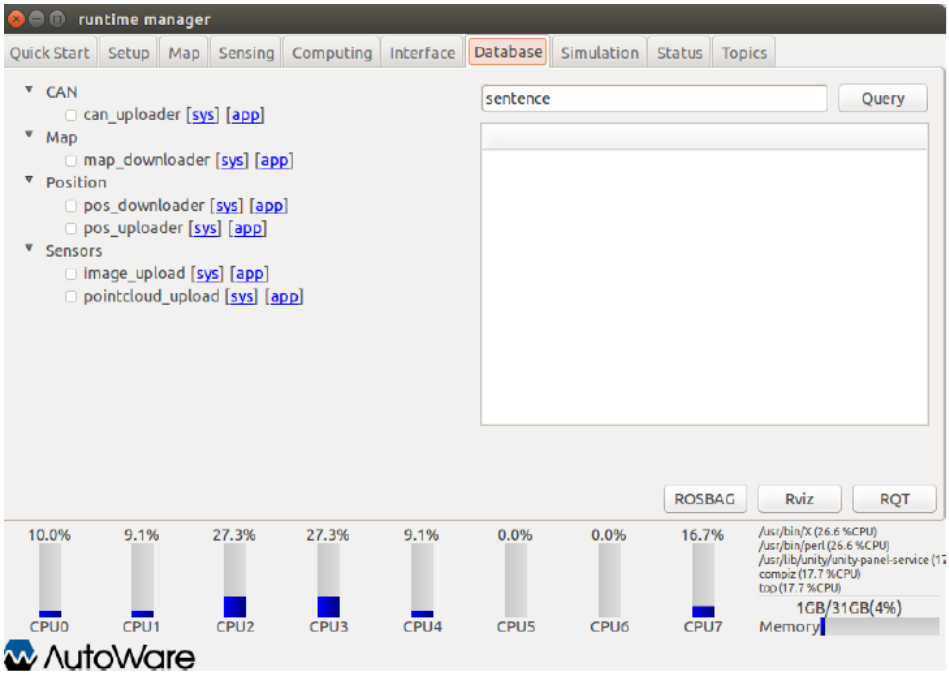
Εικόνα 28 : Runtime Manager - Interface Tab

Android Tablet: Start/end runtime_manager/tablet_socket.launch για υπολογισμούς από tablet
Oculus Rift : Προσεχώς
Vehicle Gateway : Start/end runtime_manager/vehicle_socket.launch για επικοινωννία με ZMP Robocar.
Sound : Start/end ένα sound_player/sound_player.py.
Auto Pilot : Publish a topic (/mode_cmd) σύμφωνα με το button status.
Lamp : Publish ένα topic (/lamp_cmd) σύμφωνα με το button status.
Indicator : Publish a topic (/indicator_cmd) σύμφωνα με το button status.
D: Publish ένα topic (/gear_cmd).
R : Publish ένα topic (/gear_cmd)

B : Publish ένα topic (/gear_cmd).
N : Publish ένα topic (/gear_cmd)
Accel : Publish ένα topic (/accel_cmd) από το slider status.
Brake : Publish ένα topic (/brake_cmd) από το slider status.
Steer : Publish ένα topic (/steer_cmd) από το slider status
Torque : Προσεχώς
Veloc : Publish ένα topic (/twist_cmd) από το slider status.
Angle : Publish ένα topic (/twist_cmd) από το slider status.

Runtime Manager – Database Tab

Στο tab Database έχουμε ένα monitor όπου μπορούμε να τυπώνουμε δεδομένα όπως : πληροφορίες από το σύστημα CAN, δεδομένα των χαρτών κτλ



Εικόνα 29 : Runtime Manager - Database Tab

CAN
can_uploder : Start/end obj_db/can_uploader nodes.

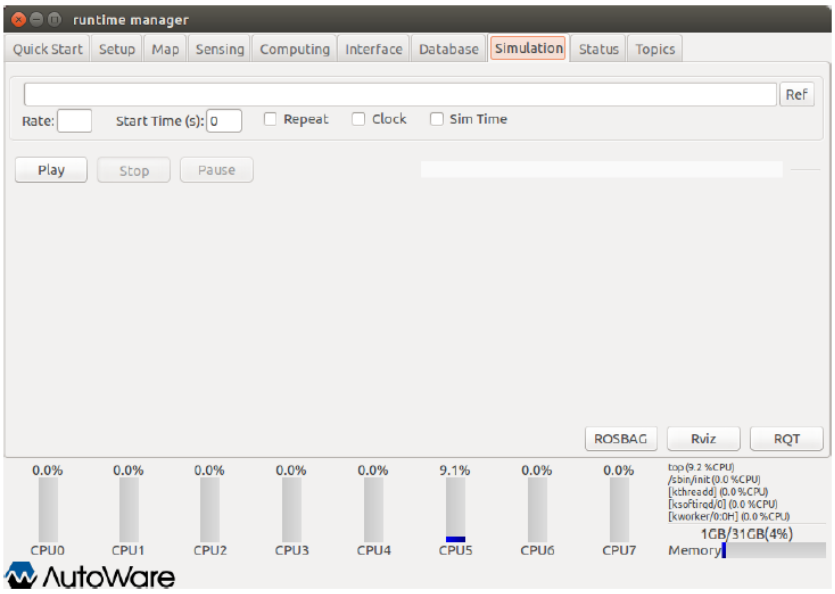
Map
map_downloader : Start/end το map_file/map_downloader.launch scripts.

Position
pos_downloader : Start/end pos_db/pos_downloader nodes.

pos_uploder : Start/end pos_db/pos_uploader nodes.
Sensors
image_upload : Προσεχώς
pointcloud_uplod : Προσεχώς
Query: Προσεχώς

Runtime Manager – Simulation Tab

Εδώ ορίζουμε το *rosbag αρχείο, τον χρόνο που θέλουμε να τρέξει η προσομοίωση καθώς επίσης και πότε ξεκινάει και πότε σταματάει. Τις πληροφορίες που φέρει το rosbag αρχείο θα τις δούμε σε παρακάτω κεφάλαια.

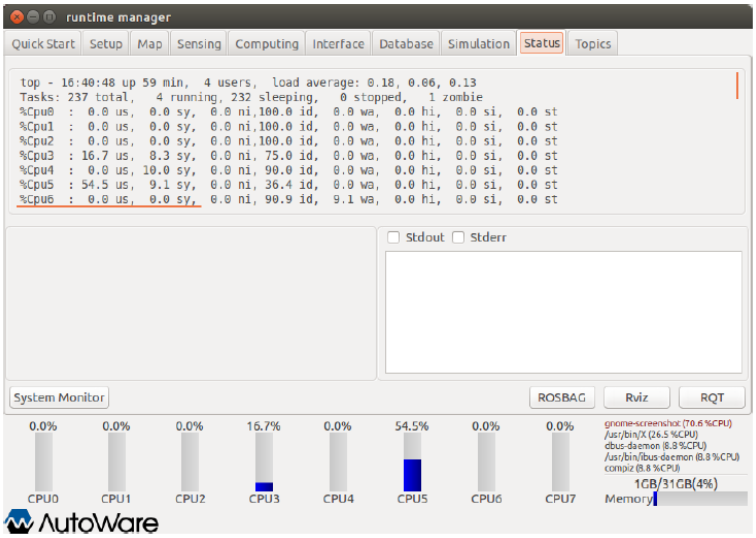


Εικόνα 30 : Runtime Manager - Simulation Tab

ROSBAG: Ξεκινά ένα ROSBAG δηλώνοντας το full path.
Rate : Δηλώνει μία αριθμητική τιμή στο ROSBAG με το option –r. Είναι optional
State Time(s) : Δήλωση του χρονικού διαστήματος του simulation που θέλουμε να τρέξουμε
Repeat: Αφού ολοκληρωθεί ο χρόνος του State time, άμα είναι επιλεγμένο τότε η προσομοίωση επαναλάμβανεται
Play : Ξεκινά την προσομοίωση
Stop : Σταματά την τρέχουσα προσομοίωση
Pause : Παγώνει την τρέχουσα προσομοίωση
RViz : Start/End RViz
RQT : Start/End Rqt
Gazebo : Σε νεότερες εκδόσεις υπάρχει και επιλογή Start/End του Gazebo

Runtime Manager – Status Tab

Σ αυτό το tab λαμβάνουμε πληροφορίες για τον status του simulation.

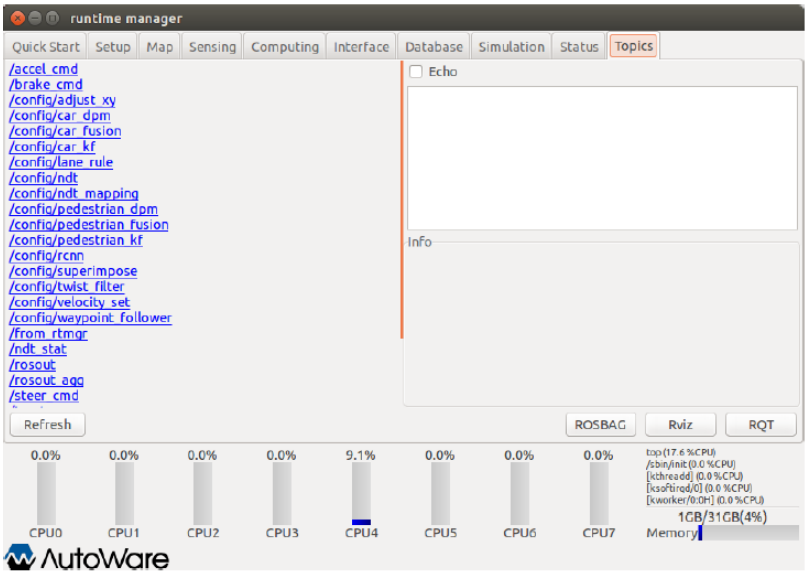


Εικόνα 31 : Runtime Manager - Status Tab

Top window: Προβάλει το αποτέλεσμα της current εντολής
Left-bottom window: Εμφάνιση του χρόνου περιοδικής εκτέλεσης που δημοσιεύεται από τον σχετικό node.
Right-bottom window: Εμφάνιση launched nodes, standard outputs και standard outputs erros των scripts

Runtime Manager - Topics Tab

Σ αυτό το tab λαμβάνουμε πληροφορίες για τα topics.



Εικόνα 32 : Runtime manager - Topics Tab

Κεφάλαιο 4ο

Αρχιτεκτονική του Hardware

Το Autoware μπορεί να χρησιμοποιηθεί για να παράξουμε δεδομένα πεδίου, πραγματοποιώντας μία διαδρομή με το DBW όχημά μας και να τα αποθηκεύσουμε ως ένα ROSBAG database. Αυτό το πακέτο μπορεί να χρησιμοποιηθεί στην συνέχεια προκειμένου να τρέξουμε διάφορες προσομοιώσεις με αυτή τη βάση δεδομένων. Τώρα, προκειμένου να παράξουμε όλα αυτά τα δεδομένα θα πρέπει να εφοδιαστούμε με επιπλέον hardware όπως : κάμερες, laserscan, αισθητήρες, ραντάρ , IMU, ενεργοποιητές κτλ.

Το Autoware υποστηρίζει κάμερες, LIDAR, IMU και GPS ως πρωτεύοντες αισθητήρες. Παρακάτω θα αναφέρουμε όλα τα εξαρτήματα που έχουν επαληθευτεί από το Autoware μέσω field testing.

4.1 Τύποι καμερών που υποστηρίζονται και έχουν επαληθευτεί από το Autoware

- PointGrey (FLIR) Grasshopper 3 (USB/GigE)
  - https://www.ptgrey.com/grasshopper3-usb3-vision-cameras
- PointGrey (FLIR) Flea 2/3 (USB/GigE)
  - https://www.ptgrey.com/flea3-gige-vision-cameras
- PointGrey (FLIR) Blackfly (USB3/GigE)
  - https://www.ptgrey.com/blackfly-usb3-vision-cameras
- Baumer VLG-22C (USB3/GigE)
  - https://www.baumer.com/sg/en/product-overview/image-processing-identification/industrial-cameras/long-term-availability/visiline-series/ams-cmosis-sensors/vlg-22c/p/24060
- Baumer VCXU-24C (USB3/GigE)
  - https://www.baumer.com/us/en/product-overview/image-processing-identification/industrial-cameras/cxseries/usb-3-0-interface/vcxu-24c/p/23796
- Allied Vision Camera Mako G-319C (PoE GigE)
  - https://www.alliedvision.com/en/products/cameras/detail/Mako%20G/G-319.html
- Generic UVC Webcam (USB2/3)

4.2 Τύποι LIDAR που υποστηρίζονται και έχουν επαληθευτεί στο Autoware

- VELODYNE HDL-64E (S1/S2/S3)
  - http://velodynelidar.com/hdl-64e.html
- VELODYNE HDL-32E
  - http://velodynelidar.com/hdl-32e.html
- VELODYNE VLP-32C
  - http://velodynelidar.com/vlp-32c.html
- VELODYNE VLP-16
  - http://velodynelidar.com/vlp-16.html
- VELODYNE VLP-16 Lite
  - http://velodynelidar.com/vlp-16-lite.html

- VELODYNE VLP-16 Hi-Res
  - http://velodynelidar.com/vlp-16-hi-res.html
- HOKUYO YVT-35LX (3D-URG)
  - https://www.hokuyo-aut.co.jp/search/single.php?serial=165
- HOKUYO UTM-30LX (TOP-URG)
  - https://en.manu-systems.com/HOK-UTM-30LX.shtml
- SICK LMS511
  - https://www.sick.com/us/en/detection-and-ranging-solutions/2d-lidar-sensors/lms5xx/lms511-10100-pro/p/p215941
- PIONEER 3D LiDAR (yet to be released)
  - http://global.pioneer/en/news/press/2017/pdf/1130-1.pdf

Μπορούν να συνδυάστουν πολλαπλές μονάδες των παραπάνω LiDAR, παρέχοντας περισσότερα δεδομένα σύντηξης pointcloud για ακριβέστερη ανίχνευση και εντοπισμό αντικειμένων.

4.3 Τύποι RADAR που υποστηρίζονται και έχουν επαληθευτεί στο Autoware

- Delphi ESR
  - https://autonomoustuff.com/product/delphi-esr-2-5-24v/

Η ανίχνευση που κάνει το Autoware βασίζεται κυρίως σε σαρωτές LIDAR. Τα RADAR είναι χρήσιμα για σκοπούς εντοπισμού αντικειμένων σε μεγάλες αποστάσεις. Ωστόσο, η ένταξή τους στον τομέα της «αντίληψης» εξακολουθεί να είναι υπό εξέλιξη.

4.4 Τύποι IMU που υποστηρίζονται και έχουν επαληθευτεί στο Autoware

- Memsic VG440
  - http://www.aceinna.com//VG440CA-200%7C400
- Xsens MTi-300
  - https://www.xsens.com/products/mti-100-series/
- MicroStrain 3DM-GX5-15
  - http://www.microstrain.com/inertial/3dm-gx5-15
- Novatel IGM S1 IMU
  - https://www.novatel.com/products/span-gnss-inertial-systems/span-imus/span-mems-imus/imu-igm-s1/

Γενικώς ο εντοπισμός SLAM [16] είναι αρκετά αξιόπιστος χωρίς τη χρήση ενός IMU. Ωστόσο, πιστεύουμε ότι το IMU εξακολουθεί να είναι χρήσιμο. Επομένως, το Autoware υποστηρίζει ακόμα το IMU και την ενσωμάτωση δεδομένων στις ενότητες εντοπισμού.

4.5 Τύποι GPS/GNSS που υποστηρίζονται και έχουν επαληθευτεί στο Autoware

- Javad DELTA-3
  - https://www.javad.com/jgnss/products/receivers/delta-3.html
- MITSUBISHI AQLOC (only available in Japan)
  - http://www.mitsubishielectric.co.jp/news/2017/1129.html
- Trimble NetR9
  - http://www.trimble.com/Infrastructure/Trimble-NetR9.aspx
- Leica Viva GS25
  - https://leica-geosystems.com/products/gnss-systems/receivers/leica-viva-gs10-gs25
- Applanix APX-15 UAV
  - https://www.applanix.com/products/dg-uavs.htm

Οι δέκτες GPS / GNSS τυπικά παράγουν NMEA-compliant sentences (text strings), όπου και υποστηρίζονται εξ ολοκλήρου από το Autoware, μέσω της σειριακής διασύνδεσης.

4.6 Πακέτα και λειτουργίες που παρέχει το Autoware στον τομέα αντίληψης

Localization

Το **lidar_localizer** υπολογίζει τη θέση (x, y, z, roll, pitch, yaw) του οχήματος στο σύστημα αξόνων του κόσμου, χρησιμοποιώντας τα σαρωμένα δεδομένα από το LiDAR και τις προεγκατεστημένες πληροφορίες του 3D χάρτη. Συνίσταται ο αλγόριθμος NDT [14] (Normal Distributions Transform) για τη αντιστοίχιση της σάρωσης LiDAR με τον τρισδιάστατο χάρτη. Υποστηρίζεται επίσης και ο αλγόριθμος ICP [20] (Iterative Closet Point).

Το **gnss_localizer** μετατρέπει το μήνυμα NMEA από ένα δέκτη GNSS στη θέση (x, y, z, roll, pitch, yaw). Αυτό το αποτέλεσμα μπορεί να χρησιμοποιηθεί ως η θέση του οχήματος, ενώ μπορεί επίσης να χρησιμοποιηθεί για την αρχικοποίηση και συμπλήρωση του αποτελέσματος του **lidar_localizer**.

Το **dead_reckoner** χρησιμοποιεί κυρίως έναν αισθητήρα IMU προκειμένου να προβλέπει κάθε φορά την επόμενη θέση του οχήματος. Επίσης, πραγματοποιεί μία παρεμβολή μεταξύ των αποτελεσμάτων του **lidar_localizer** και του **gnss_localizer**.

Detection

Το **lidar_detector** παρέχει δυνατότητες ανίχνευσης αντικειμένων βασισμένες στο LiDAR. Η βασική απόδοση προέρχεται από τον Ευκλείδειο αλγόριθμο ομαδοποίησης. Για την ταξινόμηση των συμπλεγμάτων, υποστηρίζονται επίσης αλγόριθμοι που βασίζονται στο DNN[21], όπως το VoxelNet [22] και το LMNet [23].

Το **lidar_tracker** εντοπίζει τα κινούμενα αντικείμενα που εντοπίζονται από το **lidar_detector**. Το αποτέλεσμα της παρακολούθησης μπορεί να χρησιμοποιηθεί για την πρόβλεψη της συμπεριφοράς του αντικειμένου και την εκτίμηση της ταχύτητας του αντικειμένου. Ο αλγόριθμος παρακολούθησης βασίζεται στα φίλτρα Kalman[24].

Ο **vision_detector** παρέχει και αυτός δυνατότητες ανίχνευσης αντικειμένων. Οι κύριοι αλγόριθμοι είναι ο SSD [25] και Yolo [25], τα οποία έχουν σχεδιαστεί για να εκτελούν μόνο DNN (για απόδοση σε πραγματικό χρόνο). Υποστηρίζονται πολλές κατηγορίες ανίχνευσης, όπως τα αυτοκίνητα, οι επιβάτες κ.α.

Το **vision_tracker** δεν χρησιμοποιείται επί του παρόντος, αλλά εφαρμόζει τα φίλτρα Kalman για να προβλέψει τη θέση του επόμενου πλαισίου των κινούμενων αντικειμένων που ανιχνεύεται από το **vision_detector**.

Ο **fusion_detector** χρησιμοποιεί ταυτόχρονα δεδομένα cloud point, από LiDAR, και δεδομένα εικόνας από κάμερες για να επιτύχει πιο ακριβή ανίχνευση αντικειμένων σε 3D συντεταγμένες. Οι θέσεις των καμερών και των LiDAR πρέπει να βαθμονομηθούν εκ των προτέρων. Η τρέχουσα υλοποίηση βασίζεται στον αλγόριθμο MV3D [26].

Το **fusion_tracker** χρησιμοποιεί, είτε το αποτέλεσμα του **fusion_detector,** είτε τα αποτελέσματα των **lidar_detector** και **vision_detector**. Σε κάθε περίπτωση, τα κινούμενα αντικείμενα ταυτοποιούνται άμεσα για την πρόβλεψη της συμπεριφοράς του αντικειμένου και την εκτίμηση της ταχύτητας του αντικειμένου.

Prediction

Ο **moving_predictor** χρησιμοποιεί το αποτέλεσμα του object tracking που περιγράφεται παραπάνω για να προβλέψει τις μελλοντικές τροχιές των γειτονικών κινούμενων αντικειμένων, όπως τα αυτοκίνητα και οι επιβάτες.

Ο **collision_predictor** χρησιμοποιεί το αποτέλεσμα του **moving_predictor** για να προβλέψει εάν το όχημα εμπλέκεται σε πιθανή σύγκρουση με τα κινούμενα αντικείμενα. Τα waypoint και η ταχύτητα πληροφοριών του οχήματος απαιτούνται ως δεδομένα εισόδου.

Ο **cutin_predictor** χρησιμοποιεί τις ίδιες πληροφορίες όπως ο **collision_predictor** για να προβλέψει εάν τα γειτονικά κόβονται στο μπροστινό μέρος του οχήματος.

Computing/Decision

Το **decision module** του Autoware γεφυρώνει τις ενότητες **perception** και **planning**. Με το αποτέλεσμα του **perception**, το Autoware αποφασίζει μια συμπεριφορά οδήγησης, που αντιπροσωπεύεται από μια μηχανή πεπερασμένης κατάστασης, έτσι ώστε να μπορεί να επιλεγεί μια κατάλληλη λειτουργία προγραμματισμού. Η σημερινή προσέγγιση στη λήψη αποφάσεων είναι ένα σύστημα βασισμένο σε κανόνες.

Intelligence

Ο **decision_maker** συντάσσει ένα μεγάλο σύνολο θεμάτων που σχετίζονται με τα αποτελέσματα: της αντίληψης, τις πληροφορίες του χάρτη και την τρέχουσα κατάσταση. Τελικό στόχο έχει να δημοσιεύσει την current κατάσταση του topic.

State

Το **state_machine** αλλάζει την κατάσταση μέσα σε προκαθορισμένους κανόνες, μαζί με τον **decision_maker**.

Computing/Planning

Το τελευταίο κομμάτι υπολογισμού στο Autoware είναι το planning module. Ο ρόλος της ενότητας αυτής είναι να καταρτίσει δύο σχέδια : **global mission** και **local motion**, με βάση τα αποτελέσματα των ενοτήτων του **perception** και του **decision**. Ένα **global mission** καθορίζεται όταν το όχημα ξεκινήσει ή επανεκκινηθεί, ενώ ένα **local motion** ενημερώνεται σύμφωνα με τις αλλαγές των states. Για παράδειγμα, η ταχύτητα του οχήματος προγραμματίζεται να γίνει μηδέν μπροστά από ένα αντικείμενο με «Χ» περιθώριο ασφαλείας ή σε μια γραμμή στάσης εάν η κατάσταση του Autoware είναι ρυθμισμένη να το σταματήσει. Ένα άλλο παράδειγμα είναι η τροχιά του οχήματος να έχει σκοπό να παρακάμψει ένα εμπόδιο εάν η κατάσταση του Autoware έχει ρυθμιστεί να "αποφεύγει". Τα κύρια πακέτα που περιλαμβάνονται στη μονάδα προγραμματισμού είναι τα ακόλουθα:

Mission

Το **route_planner** αναζητά μια συνολική διαδρομή προς τον προορισμό. Η διαδρομή αντιπροσωπεύεται από ένα σύνολο διασταυρώσεων στο οδικό δίκτυο.

Το **lane_planner** καθορίζει τις λωρίδες που θα χρησιμοποιηθούν μαζί με τη διαδρομή που δημοσιεύει το **route_planner**. Οι λωρίδες αντιπροσωπεύονται από πίνακες των waypoints (πολλαπλά waypoints αντιπροσωπεύουν μία λωρίδα).

Το **waypoint_planner** μπορεί εναλλακτικά να χρησιμοποιηθεί για τη δημιουργία waypoints του προορισμού.

Το **waypoint_maker** είναι ένα βοηθητικό εργαλείο για την αποθήκευση και φόρτωση των χειροκίνητων waypoints. Για να αποθηκευτούν waypoints σε ένα καθορισμένο αρχείο, θα πρέπει να οδηγηθεί το όχημα χειροκίνητα ενώ θα είναι ενεργοποιημένος ο εντοπισμός. Το Autoware καταγράφει τα waypoints της διαδρομής με προσθέτοντας και την πληροφορία της ταχύτητας. Τα καταγεγραμμένα waypoints μπορούν να φορτωθούν αργότερα από το καθορισμένο αρχείο, ώστε να έχουν στη διάθεσή τους το motion planning path.

Motion

Ο **velocity_planner** ενημερώνει ένα σχέδιο ταχύτητας στα waypoints που έχουν εγγραφεί είτε από το **lane_planner**, είτε από **waypoints_planner** ή από **waypoints_maker**, έτσι ώστε να επιταχύνεται/επιβραδύνεται η κίνηση προς τα γύρω οχήματα καθώς και τα οδικά χαρακτηριστικά, όπως οι γραμμές στάσης και τα φανάρια. Ο πληροφορίες ταχύτητας που είναι ενσωματωμένες στα συγκεκριμένα waypoints είναι στατικές.

Ο **astar_planner** εφαρμόζει τον αλγόριθμο αναζήτησης **hybrid A***[27], ο οποίος παράγει μια εφικτή τροχιά από το current position στο specified position. Αυτό το πακέτο μπορεί να χρησιμοποιηθεί για την αποφυγή εμποδίων και τις απότομες στροφές στα προκαθορισμένα waypoints καθώς και τη δρομολόγηση σε ελεύθερο χώρο, όπως χώρους στάθμευσης.

Ο **adas_lattice_planner** υλοποιεί τον αλγόριθμο **State Lattice planning** [28], ο οποίος παράγει πολλαπλές εφικτές τροχιές μπροστά από την τρέχουσα θέση βάση:

- καμπυλών spline
- έναν προκαθορισμένο πίνακα παραμέτρων
- πληροφορίες ADAS Map (aka, Vector Map).

Αυτό το πακέτο μπορεί να χρησιμοποιηθεί κυρίως για αποφυγή εμποδίων και αλλαγών λωρίδων.

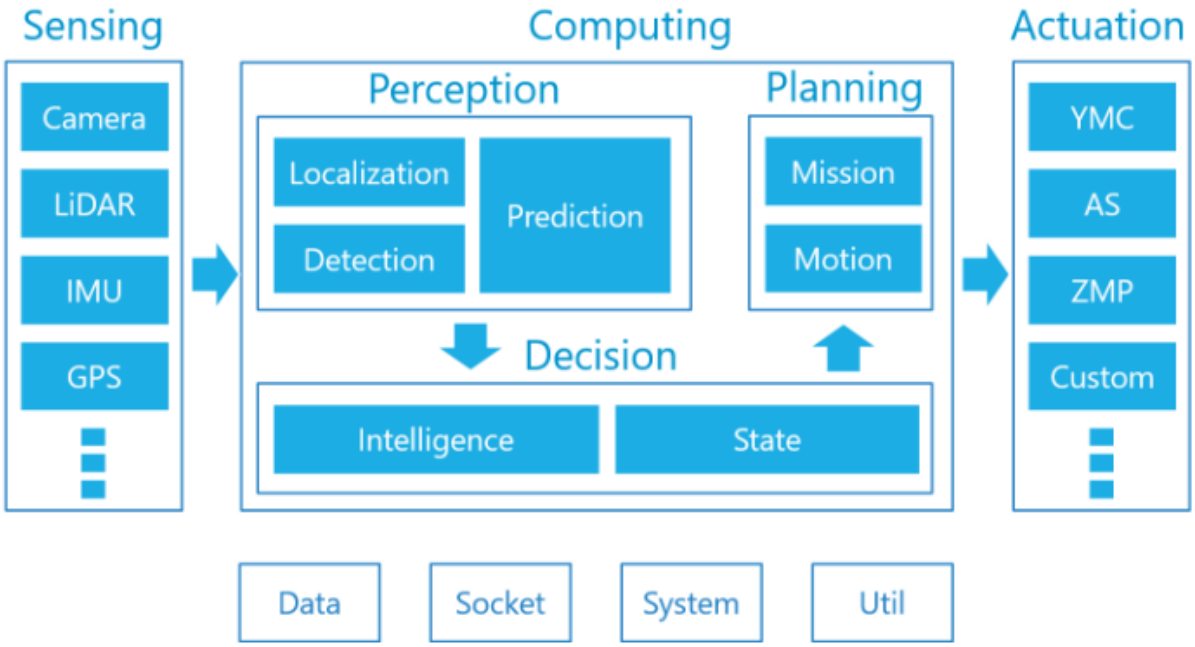
Ο **waypoint_follower** εφαρμόζει τον αλγόριθμο **Pure Pursuit** [29] που δημιουργεί ένα twisted set γραμμικής και γωνιακής ταχύτητας για να μετακινήσει το όχημα με ομοιόμορφη κυκλική κίνηση σε ένα target waypoint πάνω από τα συγκεκριμένα σημεία διέλευσης. Αυτό το πακέτο θα πρέπει να χρησιμοποιείται σε συνδυασμό με:

- το **velocity_planner**
- το **astar_planner**
- το **adas_lattice_planner**.

To published twisted set θα διαβαστεί από έναν controller του οχήματος ή από **by-wire interface** και το όχημα θα ελέγχεται αυτόνομα.

Actuation

Το Autoware έχει εγκατασταθεί και έχει δοκιμαστεί με διάφορα οχήματα: Yamaha Golf Cart, Suzuki Senior Car, Toyota Estima, Toyota Prius PHV, Toyota Prius, Ford Lincoln MKZ, Ford Fusion, Lexus RX 450h.



Εικόνα 33 : Overview of Autoware

Στην παραπάνω εικόνα βλέπουμε σε μορφή μπλοκ διαγράμματος τον τρόπο επικοινωνίας του hardware και software του Autoware. Όπως είναι αναμενόμενο έχουμε ένα κλειστό σύστημα όπου έχουμε είσοδο-επεξεργασία-έξοδο, όπου σαν είσοδο έχουμε όλα τα δεδομένα που έρχονται από τους διάφορους αισθητήρες του συστήματος, αυτά μεταδίδονται στην μονάδα επεξεργασίας όπου τα επεξεργάζεται και παίρνει τις ανάλογες αποφάσεις και εξάγει εντολές για στους ενεργοποιητές.

Κεφάλαιο 5^{ο}

Object recognition algorithm

5.1 Ανίχνευση των αντικειμένων

Ένας από τα πιο εξελιγμένα και σύγχρονα πακέτα ανίχνευσης που υπάρχουν γενικά στο ROS και ειδικά στο Autoware είναι το YOLO v3 package [30]. Τα ακρωνύμια του πακέτου δηλώνουν το εξής : You Only Look Once.

Αρχικά λαμβάνοντας υπ όψιν διάφορα άλλα ανταγωνιστικά συστήματα (πχ. SSD, R-CNN κτλ) ανίχνευσης, μπορούμε να δούμε την ακολουθία που έχουν, όπου:

- τοποθετούν classifiers ή detectors για να γίνει η ανίχνευση
- εφαρμόζετε ένα πρότυπο δείγμα σε μια εικόνα, σε πολλαπλές όμως τοποθεσίες
- τέλος, οι περιοχές με την υψηλότερη βαθμολογία ορίζουν τι ανιχνεύθηκε

Στο YOLO χρησιμοποιείτε μία εντελώς διαφορετική προσέγγιση:

- Για αρχή εφαρμόζεται ένα ενιαίο νευρωνικό δίκτυο σε όλη την εικόνα
- Αυτό με τη σειρά του διαιρεί την εικόνα σε μικρές περιοχές και ορίζει τα όρια των πλαισίων
- Το αναγνωρισμένο αντικείμενο στο κάθε πλαίσιο ορίζεται προσεγγιστικά (π.χ. άνθρωπος: 85%, ποδήλατο: 95% κτλ)

Το YOLO έχει πολλά πλεονεκτήματα σε σχέση με τα άλλα κλασικά συστήματα που βασίζονται σε classifiers. Στον YOLO αλγόριθμο, κατά την διαδικασία εκπαίδευσης, τα υποπλαίσια πάντα επαληθεύονται με το συνολικό πλαίσιο της εικόνας. Επίσης, αξιολογεί την εικόνα μόνο μία φορά σε σχέση με άλλους αλγορίθμους, όπως ο R-CNN, που το κάνει χιλιάδες μόνο για μία εικόνα. Αυτό το καθιστά εξαιρετικά γρήγορο, περισσότερο από 1000 φορές ταχύτερο από το R-CNN και 100 φορές ταχύτερο από το Fast R-CNN [31].

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights

Εικόνα 34 : Επιδόσεις με COCCO dataset

Παρακάτω έχουμε μία αρκετά ενδιαφέρουσα εργασία όπου περιγράφει και συγκρίνει τον αλγόριθμο YOLO με διάφορους άλλους αλγορίθμους. [32]

Γενική περιγραφή

Το ανθρώπινο μάτι αντικρίζοντας μία εικόνα αναγνωρίζει αμέσως ποια αντικείμενα είναι στην εικόνα. Το ανθρώπινο οπτικό σύστημα είναι γρήγορο και ακριβές, επιτρέποντάς μας να εκτελούμε σύνθετα καθήκοντα, όπως π.χ. να οδηγούμε με λιγότερη σκέψη. Οι γρήγοροι και ακριβείς αλγόριθμοι ανίχνευσης αντικειμένων μπορούν να επιτρέπουν στους υπολογιστές να οδηγούν αυτοκίνητα χωρίς πολύ εξειδικευμένους αισθητήρες και να μεταδίδουν σε πραγματικό χρόνο πληροφορίες σκηνής στους χρήστες ξεκλειδώνοντας δυνατότητες για αυτόνομα ρομποτικά συστήματα.

Τα τρέχοντα συστήματα ανίχνευσης αναπροσαρμόζουν τους ταξινομητές για ανίχνευση και εκτέλεση. Για να ανιχνεύσουμε ένα αντικείμενο, τα συστήματα αυτά παίρνουν ένα ταξινομητή, το αξιολογούν σε διάφορες τοποθεσίες και τα «ζυγίζουν» σε μια δοκιμαστική εικόνα. Αλγόριθμοι όπως ο DPM [16] χρησιμοποιούν μια προσέγγιση ολισθαίνων παραθύρων όπου ο ταξινομητής ολισθαίνει το παράθυρο σε ομοιόμορφα τοποθετημένες θέσεις σε ολόκληρη την περιοχή.

Ένας άλλος αλγόριθμος ο R-CNN [31], στην αρχή δημιουργεί πλαίσια σε μια εικόνα και στη συνέχεια εκτελεί έναν ταξινομητή σε αυτά τα προτεινόμενα πλαίσια. Μετά την ταξινόμηση, το post-processing τελειοποιεί τα όρια στα κουτιά προκειμένου να εξαλειφθούν οι διπλές ανιχνεύσεις. Σ αυτή τη μέθοδο χρησιμοποιούνται pipelines, τα οποία είναι αρκετά σύνθετα και αργά, με αποτέλεσμα να είναι δύσκολο να βελτιστοποιηθούν, καθώς κάθε ένα μεμονωμένο πλαίσιο πρέπει να εκπαιδεύεται ξεχωριστά.

Ερχόμενοι στο YOLO, ο αλγόριθμος χρειάζεται να αντικρίσει μόνο μία φορά μια εικόνα για να προβλέψει ποια αντικείμενα είναι παρόντες και που βρίσκονται. Το YOLO είναι αρκετά απλό convolutional network, καθώς ορίζει ταυτόχρονα πολλαπλά πλαίσια και πιθανότητες κλάσης για αυτά τα πλαίσια. Το YOLO εκπαιδεύεται σε πλήρης εικόνες και βελτιστοποιεί άμεσα την απόδοση ανίχνευσης.

Τα πλεονεκτήματα του YOLO συγκριτικά είναι:

- είναι συγκριτικά γρηγορότερο από τις άλλες μεθόδους
- Δεν χρειάζεται complex pipelines. Απλά τρέχει το νευρωνικό δίκτυο σε μια νέα εικόνα σε δοκιμαστικό χρόνο, για να προβλέψει τι μπορεί να ανιχνευθεί. Το βασικό δίκτυο λειτουργεί στα 45 fps (χωρίς batch mode σε μία Titan X κάρτα) και μια πιο γρήγορη έκδοση ξεπερνά τα 150 fps. Αυτό σημαίνει ότι μπορεί να επεξεργαστεί streaming βίντεο σε πραγματικό χρόνο με λιγότερο από 25 χιλιοστά του δευτερολέπτου.
- Σε αντίθεση με τον DPM και R-CNN, το YOLO βλέπει ολόκληρη την εικόνα κατά τη διάρκεια της εκπαίδευσής του, αλλά και κατά τον χρόνο των δοκιμών. Έτσι κωδικοποιεί άμεσα τις «συμφραζόμενες» πληροφορίες καθώς και το που βρίσκονται στην εικόνα.

Επίσης, ο YOLO μαθαίνει γενικευμένες παραστάσεις αντικειμένων. Όταν εκπαιδεύεται σε φυσικές εικόνες και δοκιμάζεται, ξεπερνά τις μεθόδους ανίχνευσης κορυφών όπως το DPM και R-CNN. Δεδομένου ότι ο YOLO είναι εξαιρετικά γενικεύσιμος, είναι λιγότερο πιθανό να καταρρεύσει όταν εφαρμοστεί σε νέους τομείς ή απρόοπτα inputs. Παρ όλα αυτά, το YOLO εξακολουθεί να υστερεί σε ακρίβεια σε συστήματα ανίχνευσης τελευταίας τεχνολογίας.

Unified Detection

Τα στοιχεία ανίχνευσης αντικειμένων βρίσκονται ενοποιημένα σε ένα μόνο νευρωνικό δίκτυο. Το δίκτυό χρησιμοποιεί χαρακτηριστικά από ολόκληρη την εικόνα για να προβλέψει την οριοθέτηση κάθε πλαισίου. Επίσης, προβλέπει όλα τα όρια των πλαισίων σε όλες τις κατηγορίες για μια εικόνα ΤΑΥΤΟΧΡΟΝΑ.

Ο αλγόριθμος χωρίζει την εικόνα σε εισόδους $S\ X\ S$ grid. Εάν το κέντρο ενός αντικειμένου πέσει σε ένα grid cell, αυτό το grid cell είναι υπεύθυνο για την ανίχνευση αυτού του αντικειμένου. Κάθε grid cell προβλέπει τα bounding boxes και το confidence score για αυτά τα κουτιά. Τα confidence score αντικατοπτρίζουν την βεβαιότητα ότι το αυτό που βλέπει στο κουτί είναι το σωστό αντικείμενο. Εάν δεν υπάρχει αντικείμενο σ αυτό το cell θα πρέπει το score να είναι μηδέν.

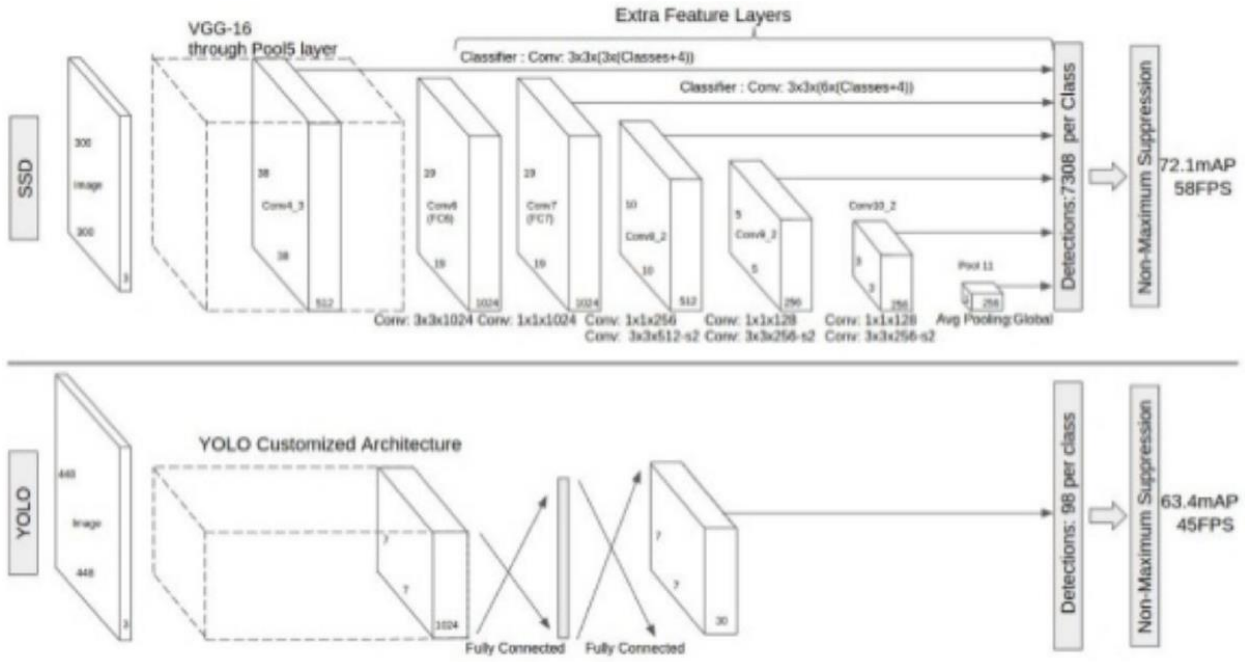
Κάθε bounding box αποτελείται από 5 προβλέψεις: x, y, w, h, και confidence. Οι συντεταγμένες (x, y) αντιπροσωπεύουν το κέντρο του κουτιού σχετικά με τα όρια του grid cell. Το πλάτος και το ύψος προβλέπεται σε σχέση με ολόκληρη την εικόνα. Το confidence prediction αντιπροσωπεύει το IOU μεταξύ του προβλεπόμενου πλαισίου και του ground truth.

Κάθε grid cell προβλέπει και τα conditional class probabilities C, Pr (ClassijObject). Αυτές οι πιθανότητες είναι κλιμακωτές στο grid cell που περιέχει ένα αντικείμενο. Προβλέπεται μόνο ένα σύνολο πιθανών κλάσεων ανά grid cell, ανεξάρτητα από τον αριθμό των κιβωτίων B. Στο χρόνο δοκιμής πολλαπλασιάζονται τα conditional class probabilities και τα individual box confidence predictions:

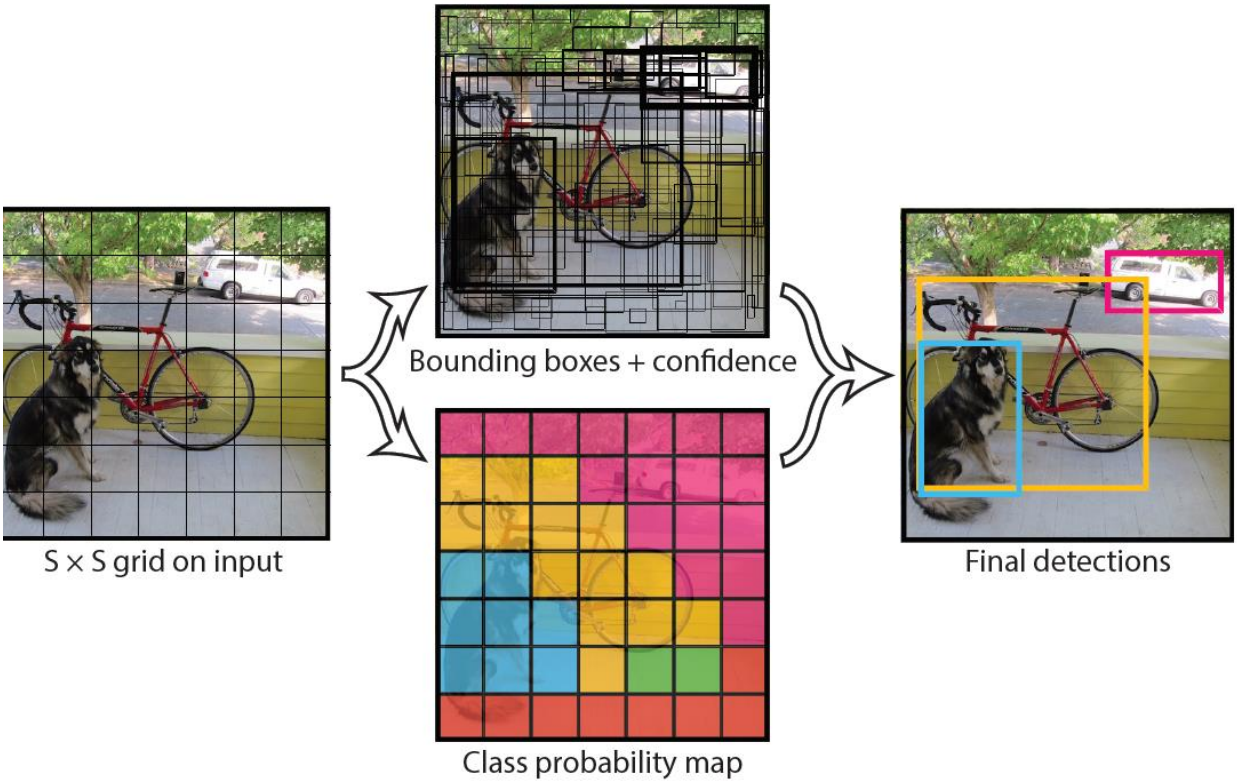
$$\Pr(\text{Class}_i \mid \text{Object}) * \Pr(\text{Object}) * \text{IOU}^{truth}_{pred} = \Pr(\text{Class}_i) * \text{IOU}^{truth}_{pred}$$

που μας δίνουν βαθμολογίες αξιοπιστίας για κάθε κατηγορία κάθε κουτιού. Αυτές οι βαθμολογίες κωδικοποιούν τόσο την πιθανότητα αυτής της κλάσης που εμφανίζεται στο κουτί καθώς επίσης και πόσο καλά ταιριάζει στο προβλεπόμενο αντικείμενο.

Π.χ. με S=7, B=2, C=20. Τελικό αποτέλεσμα έχουμε έναν *tensor 7X7X30.*



Εικόνα 35 : Σύγκριση SSD-YOLO



Εικόνα 36 : Εικόνα δοκιμής του αλγορίθμου. Παρακάτω αναλύεται η συνολική διαδικασία.

5.2 Πραγματοποίηση training

Αρχικά προεκπαιδεύουμαι το convolutional layers στο ImageNet των 1000 κλάσεων. Στην προ-εκπαίδευση χρησιμοποιούνται αρχικά τα πρώτα 20 convolutional layers. Εκπαιδεύεται αυτό το δίκτυο για περίπου μια εβδομάδα σε ποσοστό επιτυχίας ~88% . Το Darknet framework χρησιμοποιείται για την εξαγωγή των συμπερασμάτων.

Στη συνέχεια, διακριτοποιείται το μοντέλο για να γίνει η ανίχνευση.

Το YOLO προβλέπει πολλά bounding boxes ανά grid cell. Κατά τη διάρκεια του training χρειάζεται μόνο ένα predictor boundary box να είναι υπεύθυνο για κάθε αντικείμενο. Αυτό οδηγεί σε εξειδίκευση του κάθε predictor boundary box. Κάθε predictor βελτιώνεται καθώς προβλέπει ορισμένων μεγέθη, αναλογίες διαστάσεων ή κατηγορίες αντικειμένων, βελτιώνοντας έτσι την συνολική του ανάκληση.

Κατά τη διάρκεια της εκπαίδευσης βελτιστοποιείται η παρακάτω συνάρτηση:

$$\lambda\text{coord}\sum_{i=0}^{s2}\sum_{j=0}^{B}1_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]$$

$$+\lambda\text{coord}\sum_{i=0}^{s2}\sum_{j=0}^{B}1_{ij}^{obj}[(\sqrt{\omega_i}-(\sqrt{\widehat{\omega}_i})^2+\sqrt{h_i}-(\sqrt{\widehat{h}_i})^2]$$

$$+\sum_{i=0}^{s2}\sum_{j=0}^{B}1_{ij}^{obj}(C_i-\hat{C}_i)^2+\lambda\text{coord}\sum_{i=0}^{s2}\sum_{j=0}^{B}1_{ij}^{obj}(C_i-\hat{C}_i)^2$$

$$+\sum_{j=0}^{B}1_{ij}^{obj}\sum_{c\epsilon\,classes}(p_i(c)-\hat{p}_i(c))^2$$

Όπου 1_i^{obj} δηλώνει εάν ένα αντικείμενο εμφανίζεται σε ένα cell i και το 1_{ij}^{obj} δηλώνει ποιος predictor είναι υπεύθυνος για αυτό το cell.

Το συγκεκριμένο δίκτο εκπαιδεύεται με 135 εποχές κατά την εκπαίδευση και χρησιμοποιούνται validation data από PASCAL VOC 2007 και 2012.

Ακριβώς όπως στην εκπαίδευση, έτσι και στην πρόβλεψη ανίχνευσης, πάνω στις δοκιμαστικές εικόνες απαιτείται μόνο μία αξιολόγηση δικτύου. Στο PASCAL VOC το δίκτυο προβλέπει 98 οριοθετημένα κουτιά ανά εικόνα και class probabilities για κάθε κιβώτιο. Το YOLO είναι εξαιρετικά γρήγορο στη δοκιμή δεδομένων καθώς απαιτεί single network evaluation, σε αντίθεση με τις classifier-based μεθόδους.

Ο σχεδιασμός του πλέγματος επιβάλλει τη χωρική ποικιλομορφία στο όριο πρόβλεψης του κουτιού. Συχνά είναι σαφές σε πιο grid cell πέφτει το αντικείμενο και το δίκτυο προβλέπει μόνο ένα πλαίσιο για κάθε αντικείμενο. Ωστόσο, ορισμένα μεγάλα αντικείμενα ή αντικείμενα κοντά στα όρια των πολλαπλών cells μπορούν να εντοπιστούν και μέσα σε άλλα cells.

5.3 Περιορισμοί του YOLO

Ο YOLO επιβάλλει ισχυρούς χωρικούς περιορισμούς στην οριοθέτηση πρόβλεψης του κουτιού, δεδομένου ότι κάθε κυψέλη πλέγματος προβλέπει μόνο δύο πλαίσια και μπορεί να έχει μόνο μία κλάση. Αυτός ο περιορισμός του χώρου περιορίζει και τον αριθμό των κοντινών αντικειμένων που μπορεί να προβλέψει σε ένα μοντέλο.

Με βάση το συγκεκριμένο παράδειγμα εικόνας, η loss function αντιμετωπίζει τα ίδια σφάλματα είτε σε μικρά είτε σε μεγάλα κουτιά. Ένα μικρό σφάλμα σε ένα μεγάλο κουτί είναι γενικά αμελητέο, αλλά το μικρό σφάλμα σε ένα μικρό κουτί έχει πολύ μεγαλύτερη επίδραση στο IOU.

5.4 Σύγκριση με άλλα Detection systems

Deformable parts model: το DPM [16] χρησιμοποιεί μια προσέγγιση συρόμενου παραθύρου για την ανίχνευση αντικειμένων. Χρησιμοποιεί disjoint pipelines όπου: κάνει εξαγωγή στατικών χαρακτηριστικών, ταξινομεί περιοχές, και προβλέπει πλαίσια με high scoring regions. Το YOLO αντικαθιστά όλα αυτά τα διαφορετικά μέρη με ένα μόνο convolutional neural network. Το δίκτυο εκτελεί : εξαγωγή χαρακτηριστικών, bounding box prediction, nonmaximal suppression και contextual reasoning ταυτόχρονα. Αντί για στατικά χαρακτηριστικά, το δίκτυο εκπαιδεύει τα χαρακτηριστικά in-line και τα βελτιστοποιεί για την εργασία ανίχνευσης.

R-CNN: ο R-CNN [31] χρησιμοποιεί συρόμενα παράθυρα για να βρει αντικείμενα στις εικόνες. Η επιλεκτική αναζήτηση δημιουργεί:

- πιθανά πλαίσια οριοθέτησης
- ένα convolutional network extracts features
- ένα SVM να βαθμολογεί τα πλαίσια
- ένα γραμμικό μοντέλο να ρυθμίζει τα κιβώτια οριοθέτησης
- Τέλος ένα non-max suppression eliminates duplicate detections.

Ο YOLO χρησιμοποιεί μία παρόμοια τεχνική αλλά με λιγότερα bound boxes.

Deep MultiBox: Σε αντίθεση με το R-CNN, το Deep MultiBox [33] χτίζει ένα convolutional network για να προβλέψει περιοχές ενδιαφέροντος, αντί να χρησιμοποιήσει την επιλεκτική αναζήτηση. Το MultiBox μπορεί επίσης να εκτελέσει ανίχνευση μεμονωμένων αντικειμένων αντικαθιστώντας το confidence prediction με μία class prediction. Ωστόσο, το MultiBox δεν μπορεί να πραγματοποιήσει general object detection με αποτέλεσμα να εξακολουθεί να είναι απλά μία επέκταση ενός detection pipeline , όπου απαιτεί περαιτέρω patch classification.

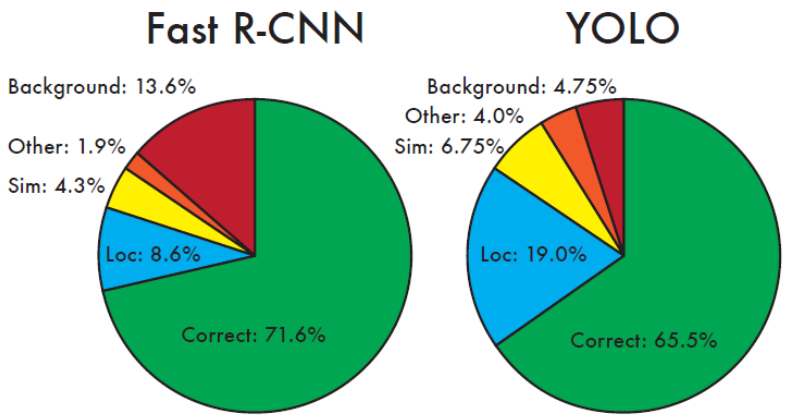
Τόσο το YOLO όσο και το MultiBox χρησιμοποιούν ένα convolutional network για την πρόβλεψη οριοθέτησης πλαισίων σε μια εικόνα, το YOLO όμως θεωρείται ένα πλήρες σύστημα ανίχνευσης.

5.4 Σύγκριση των διαφόρων μεθόδων με πειραματικά δεδομένα

Παρακάτω μπορούμε να δούμε πειραματικές συγκρίσεις μεταξύ του YOLO και του DPM. Επίσης, λήφθηκε υπ όψιν και το mAP (mean Average Precision) του κάθε αλγορίθμου ούτως ώστε να ελεγχθεί και η ακρίβεια του αποτελέσματος του κάθε αλγορίθμου για κάθε αντικείμενο.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Εικόνα 37 : Real-time με PASCAL VOC 2007



Εικόνα 38 Error Analysis - Fast R-CNN vs. YOLO

Το YOLO κάνει πολύ λιγότερα λάθη στο background σε σχέση με το Fast-RCNN. Χρησιμοποιώντας το YOLO για την ανίχνευση αντικειμένων στο background ο χρόνος που κερδίζεται είναι ξεκάθαρα πολύ μεγαλύτερος . Το καλύτερο δυνατό αποτέλεσμα που μπορούσε να δώσει ο Fast-RCNN είναι mAP=71.8% με το YOLO=75%.

	mAP	Combined	Gain
Fast R-CNN	71.8	-	-
Fast R-CNN (2007 data)	**66.9**	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	63.4	**75.0**	**3.2**

Εικόνα 39 : Σύγκριση αποτελεσμάτων

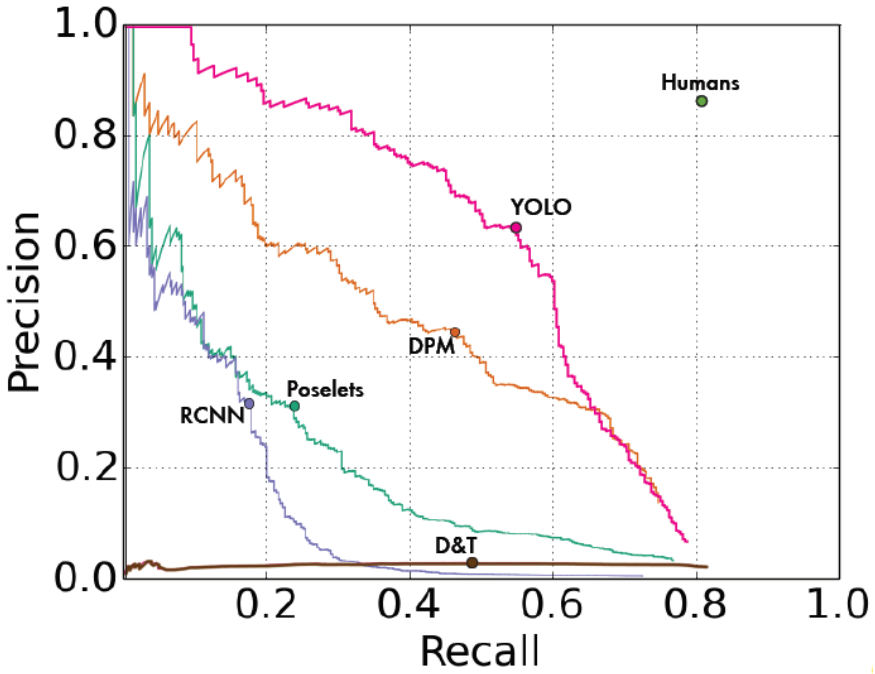
Παρακάτω έχουμε μπορούμε να δούμε και πιο αναλυτικά για κάθε αντικείμενο:

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Εικόνα 40 : Σύγκριση αποτελεσμάτων – κάθε αντικειμένου

5.5 Real-time detection με το YOLO

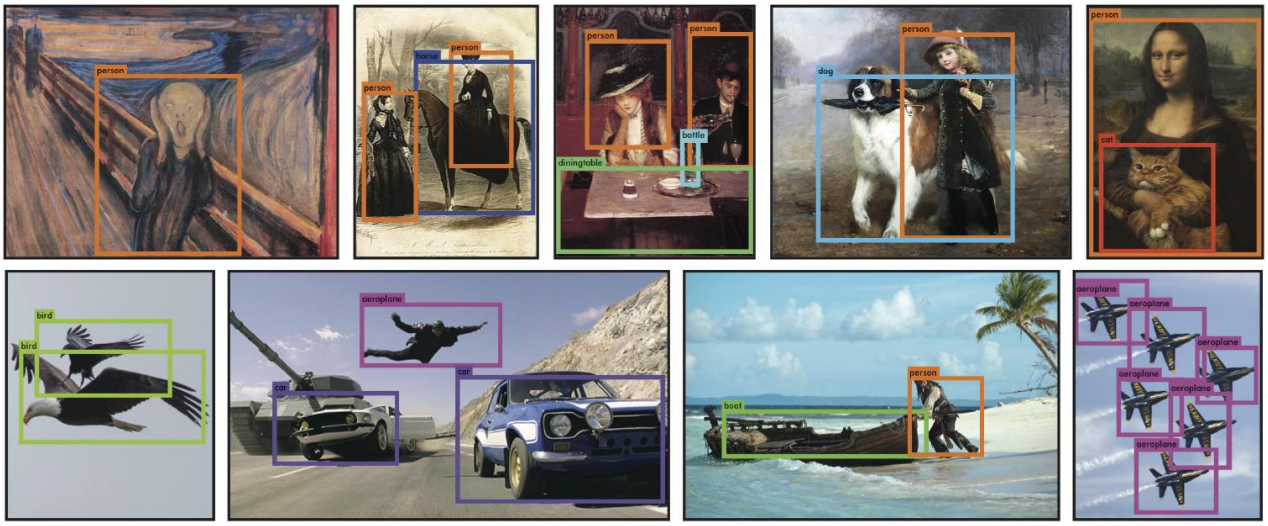
Το YOLO είναι ένας γρήγορος και ακριβής object detector, κάνοντάς τον ιδανικό για εφαρμογές ρομποτικής όρασης. Συνδέοντας το YOLO με μία webcam μπορούμε να διακρίνουμε το realtime performance που έχει:



Εικόνα 41 : Ακρίβεια αναγνώρισης προς την απόκριση

	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [2]	36.5	17.8	0.271	
D&T [4]	-	1.9	0.051	

Εικόνα 42 : Ποσοτική αναγνώριση με VOC 2007



Εικόνα 43 : Αποτελέσματα αναγνώρισης με webcam

5.6 Σύνοψη του testing

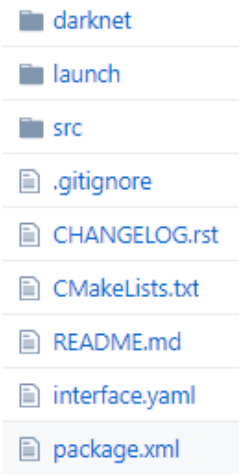
Το YOLO είναι ο ταχύτερος ανιχνευτής αντικειμένων γενικής χρήσης και κυρίως σε πραγματικό χρόνο. Πράγμα που τον καθιστά ιδανικό για καταστάσεις όπως το Autoware όπου έχουμε on-the-fly αναγνώριση και επεξεργασία εικόνας. Τα frame της εικόνας καθώς οδηγούμε προφανώς και είναι αρκετά υψηλά (επίπεδα ενός βίντεο στα 50fps), με αποτέλεσμα να χρειαζόμαστε έναν αλγόριθμο αρκετά γρήγορο και ακριβή όπως ο YOLO.

5.7 YOLO & Autoware

Όπως είπαμε και παραπάνω το YOLO έχει εισαχθεί στο Autoware. Με βάση το repository του Autoware όλα τα απαραίτητα αρχεία είναι αποθηκευμένα στο :

Autoware/ros/src/computing/perception/detection/vision_detector/packages/vision_darknet_detect/

Ερχόμενοι στο folder vision_darnet_detect έχουμε τα παρακάτω αρχεία:



Εικόνα 44 : Vision_darknet_detect αρχεία

Μέσα στο folder *launch* έχουμε όλα *.launch αρχεία για yolo2 και yolo3. Προφανώς εμείς συνεχίζουμε το yolo3.

```
<launch>
    <arg name="gpu_device_id" default="0"/>
    <arg name="score_threshold" default="0.30"/>
    <arg name="nms_threshold" default="0.30"/>
    <arg name="network_definition_file" default="$(find
vision_darknet_detect)/darknet/cfg/yolov3.cfg"/>
    <arg name="pretrained_model_file" default="$(find
vision_darknet_detect)/darknet/data/yolov3.weights"/>
    <arg name="names_file" default="$(find
vision_darknet_detect)/darknet/cfg/coco.names"/>
    <arg name="camera_id" default="/"/>
    <arg name="image_src" default="/image_raw"/>
    <node pkg="vision_darknet_detect" name="vision_darknet_detect"
type="vision_darknet_detect" output="screen">
        <param name="network_definition_file" type="str" value="$(arg
network_definition_file)"/>
        <param name="pretrained_model_file" type="str" value="$(arg
pretrained_model_file)"/>
        <param name="score_threshold" type="double" value="$(arg
score_threshold)"/>
        <param name="nms_threshold" type="double" value="$(arg nms_threshold)"/>
        <param name="gpu_device_id" type="int" value="$(arg gpu_device_id)"/>
        <param name="image_raw_node" type="str" value="$(arg camera_id)$(arg
image_src)"/>
        <param name="names_file" type="str" value="$(arg names_file)"/>
    </node>
</launch>
```

Εφόσον θέλουμε να τρέξουμε το YOLO ξεχωριστά μέσα από το Autoware για να κάνουμε ένα testing στο visio tracking, μπορούμε να τρέξουμε την παρακάτω εντολή (για να το κάνουμε αυτό προφανώς έχουμε και το αντίστοιχο hardware, όπως κάμερες):

```
roslaunch vision_darknet_detect vision_yolo3_detect.launch
```

Παρακάτω μπορούμε να δούμε μία βιντεοσκόπηση μιας τέτοιας διαδικασίας:

https://www.youtube.com/watch?v=pO4vM4ehI98

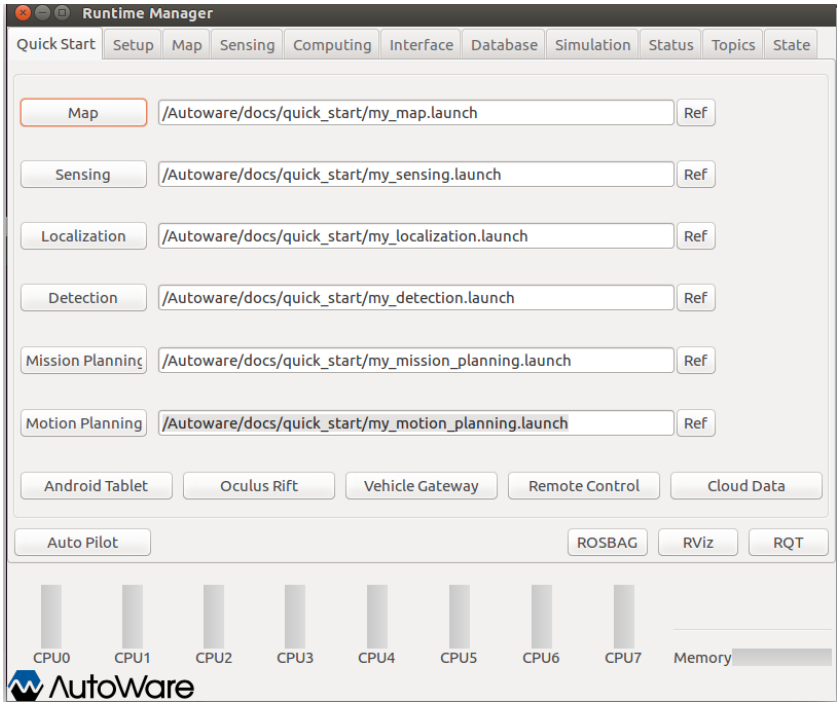
Κεφάλαιο 6ο

Αρχιτεκτονική του Software

Όπως έχει αναφερθεί και παραπάνω ένα αυτόνομο όχημα χρειάζεται κάποιες αρχικές, πολύ βασικές πληροφορίες, προκειμένου να μπορεί να κινηθεί αυτόνομα. Για αρχή χρειάζεται να γνωρίζει τον χώρο γύρω του. Στη συνέχεια, χρειάζεται να διακρίνει από τι απαρτίζεται αυτός ο χώρος : ανθρώπους, τετράποδα, διαβάσεις, φανάρια, άλλα οχήματα κτλ. Επόμενο είναι να γνωρίζει την αρχική του θέση καθώς επίσης την τωρινή θέση του σε κάθε frame αναμετάδοσης. Τέλος, πρέπει να γνωρίζει την αποστολή του: πού είναι, πού θέλει να πάει, καθώς επίσης και δεδομένα πλοήγησης: μέγιστες – ελάχιστες ταχύτητες, αν επιτρέπεται αλλαγής λωρίδας κτλ.

Όλα αυτά είναι δεδομένα που θα πρέπει να τροφοδοτήσουμε το όχημα τόσο στην πραγματικότητα όσο και σε εξομοίωση, όπως θα δούμε και παρακάτω.

Σύμφωνα με τον gui του Runtime Manager στο αρχικό tab έχουμε το Quick start tab, όπου έχει πεδία που θα πρέπει να συμπληρώσουμε με όλα τα παραπάνω δεδομένα ένα-ένα.



Εικόνα 45 :Quick start gui

Όλα μας τα δεδομένα στον Quick start ορίζονται ως *.launch αρχεία. Ουσιαστικά αυτά τα αρχεία είναι xml format αρχεία όπου καλούν ένα σύνολο άλλων αρχείων που χρειάζεται να φορτωθούν. Επίσης, στα ίδια αρχεία μπορούμε να ορίσουμε και παραμέτρους αρχικοποίησης π.χ. αρχική ταχύτητα του οχήματος, αρχική επιτάχυνση καθώς επίσης και boolean operations σε κρίση αποφάσεων. Παρακάτω θα αναλύσουμε ένα-ένα τα αρχεία που φορτώνουμε σειριακά:

6.1 Launch αρχείο για τους χάρτες

```
<launch>
  <!-- TF -->
  <!-- include file="$(env HOME)/.autoware/data/tf/tf.launch"/ -->
  <include file="/Autoware/data/tf/tf.launch"/>
  <!-- Point Cloud -->
  <node pkg="map_file" type="points_map_loader" name="points_map_loader"
args="noupdate
  /Autoware/data/map/pointcloud_map/bin_Laser-00167_-00864.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00153_-00852.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00159_-00859.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00160_-00861.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00148_-00849.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00168_-00866.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00147_-00851.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00149_-00847.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00156_-00854.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00158_-00858.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00147_-00847.pcd
```

/Autoware/data/map/pointcloud_map/bin_Laser-00154_-00852.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00158_-00857.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00154_-00851.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00167_-00866.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00168_-00865.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00169_-00868.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00147_-00849.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00153_-00850.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00154_-00853.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00161_-00861.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00168_-00867.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00151_-00849.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00153_-00851.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00152_-00850.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00150_-00848.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00156_-00856.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00167_-00865.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00157_-00856.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00148_-00848.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00152_-00851.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00156_-00855.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00168_-00868.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00161_-00860.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00159_-00857.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00150_-00847.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00164_-00863.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00157_-00857.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00162_-00861.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00151_-00850.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00165_-00864.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00149_-00848.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00155_-00852.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00160_-00859.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00163_-00861.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00160_-00858.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00158_-00856.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00155_-00854.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00160_-00860.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00159_-00858.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00147_-00850.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00148_-00847.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00163_-00862.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00151_-00848.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00155_-00853.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00152_-00849.pcd
/Autoware/data/map/pointcloud_map/bin_Laser-00149_-00846.pcd

```
  /Autoware/data/map/pointcloud_map/bin_Laser-00166_-00865.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00147_-00846.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00167_-00867.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00165_-00863.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00166_-00864.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00155_-00855.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00162_-00862.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00150_-00846.pcd
  /Autoware/data/map/pointcloud_map/bin_Laser-00164_-00862.pcd"/>

  <!-- Vector Map -->
  <node pkg="map_file" type="vector_map_loader" name="vector_map_loader"
  args="Autoware/data/map/vector_map/lane.csv
  /Autoware/data/map/vector_map/line.csv
  /Autoware/data/map/vector_map/utilitypole.csv
  /Autoware/data/map/vector_map/curb.csv
  /Autoware/data/map/vector_map/node.csv
  /Autoware/data/map/vector_map/gutter.csv
  /Autoware/data/map/vector_map/point.csv
  /Autoware/data/map/vector_map/pole.csv
  /Autoware/data/map/vector_map/vector.csv
  /Autoware/data/map/vector_map/zebrazone.csv
  /Autoware/data/map/vector_map/streetlight.csv
  /Autoware/data/map/vector_map/whiteline.csv
  /Autoware/data/map/vector_map/road_surface_mark.csv
  /Autoware/data/map/vector_map/area.csv
  /Autoware/data/map/vector_map/idx.csv
  /Autoware/data/map/vector_map/dtlane.csv
  /Autoware/data/map/vector_map/signaldata.csv
  /Autoware/data/map/vector_map/poledata.csv
  /Autoware/data/map/vector_map/roadsign.csv
  /Autoware/data/map/vector_map/roadedge.csv
  /Autoware/data/map/vector_map/crosswalk.csv
  /Autoware/data/map/vector_map/stopline.csv"/>

</launch>
```

Αρχικά το αρχείο καλεί το tf.launch αρχείο όπου ορίζει στον χάρτη μία στατική μεταβλητή συστήματος συντεταγμένων:

```
<!---->
<launch>
<!-- worldからmapへのtf -->
<node pkg="tf"  type="static_transform_publisher" name="world_to_map" args="14771
84757 -39 0 0 0 /world /map 10" />
<!-- mapからmobilityへのtf -->
<node pkg="tf"  type="static_transform_publisher" name="map_to_mobility" args="0
0 0 0 0 0 /map /mobility 10" />
</launch>
```

Στη συνέχεια ξεκινάει ο ορισμός των *.pcd αρχείων που θα πρέπει να διαβαστούν. Αυτά είναι τα κυριότερα αρχεία που χρειαζόμαστε προκειμένου να γνωρίσει τον χώρο του το όχημα. Αυτά τα αρχεία έχουν προέλθει από ένα lidar velodyne και μας προσφέρουν 3D point cloud data, όπου ο συνδυασμός αυτών μας δίνει την περιγραφή του χώρου σε σημεία.

Ανοίγοντας ένα *.pcd αρχείο έχουμε:

 # .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 990454
HEIGHT 1
VIEWPOINT 0 0 0 1 0 0 0
POINTS 990454
DATA ascii
-15163.647 -84919.289 -16.642422 65280
-15162.334 -84920.797 -15.375725 65280
-15162.222 -84920.57 -15.405148 65280
-15162.963 -84920.125 -13.565678 65280
-15163.104 -84919.82 -12.090225 65280
.....
VERSION 0.7 : αρχικά ορίζεται το version format του αρχείου
FIELDS x y z rgb : ορίζει τι δηλώνουν οι στήλες του αρχείου. Εδώ έχουμε xyz συντεταγμένες και το rgb του χρώματος.
SIZE: ορίζει τις μέγεθος της κάθε διάστασης σε bytes. (unsigned int/int/float has 4 bytes)
TYPE: τύπος της διάστασης . F = float

COUNT: ορίζει πόσα στοιχεία έχει η κάθε διάσταση. Για παράδειγμα, ένα x data έχει ένα στοιχείο ενώ ένα feature description έχει κοντά στα 308 στοιχεία.
WIDTH: το πλήθος των σημείων που υπάρχουν στο αρχείο.
HEIGHT: ορίζει το μήκος των σημείων του cloud dataset.

Παράδειγμα:

WIDTH 640 HEIGHT 480
640*480=307200 συνολικά σημεία στο cloud dataset
WIDTH 307200 HEIGHT 1
307200 συνολικά σημεία στο cloud dataset

POINTS: το συνολικό πλήθος των points στο χώρο
DATA: ο τύπος του αρχείου ascii ή binary.
Από εκεί και πέρα ξεκινάει η δήλωση των σημείων και rgb

Αυτά όπως είπαμε είναι 3D point cloud data. Σαν πληροφορία θα πρέπει λοιπόν να τα επεξεργαστούμε προκειμένου να μπορούμε να ορίσουμε τα αντικείμενα στον χώρο. Στο ίδιο *.launch αρχείο παρακάτω έχουμε πλέον και τα vector αρχεία τα οποία δέχονται σαν input τα point cloud data προκειμένου να σχηματίσουν τα διάφορα αντικείμενα που αναγνωρίστηκαν στον χώρο.

Το κάθε *.csv αρχείο χρησιμοποιείτε για να περιγραφεί και ένα διαφορετικό είδος αντικειμένων. Από το όνομα των αρχείων μπορούμε να καταλάβουμε τι είναι υπεύθυνο να σχηματίσει το κάθε αρχείο πχ,

lane.csv, line.csv, utilitypole.csv, curb.csv, node.csv, gutter.csv, point.csv, pole.csv, vector.csv, zebrazone.csv, streetlight.csv, area.csv, idx.csv, dtlane.csv, signaldata.csv, poledata.csv, roadsign.csv, roadedge.csv, crosswalk.csv, stopline.csv.

Αυτά τα *.csv αρχεία παράγονται μόνο μέσω κάποιων γεννητριών που έχουν αναλάβει διάφορες εταιρίες. Μία από αυτές τις εταιρίες είναι η Tier IV . Η συγκεκριμένη εταιρία παρέχει στην ιστοσελίδα της μία γεννήτρια που τροφοδοτείται με τα *.pcd αρχεία και στο τέλος παράγει όλα τα απαραίτητα .csv που έχουν ζητηθεί από τον χρήστη.

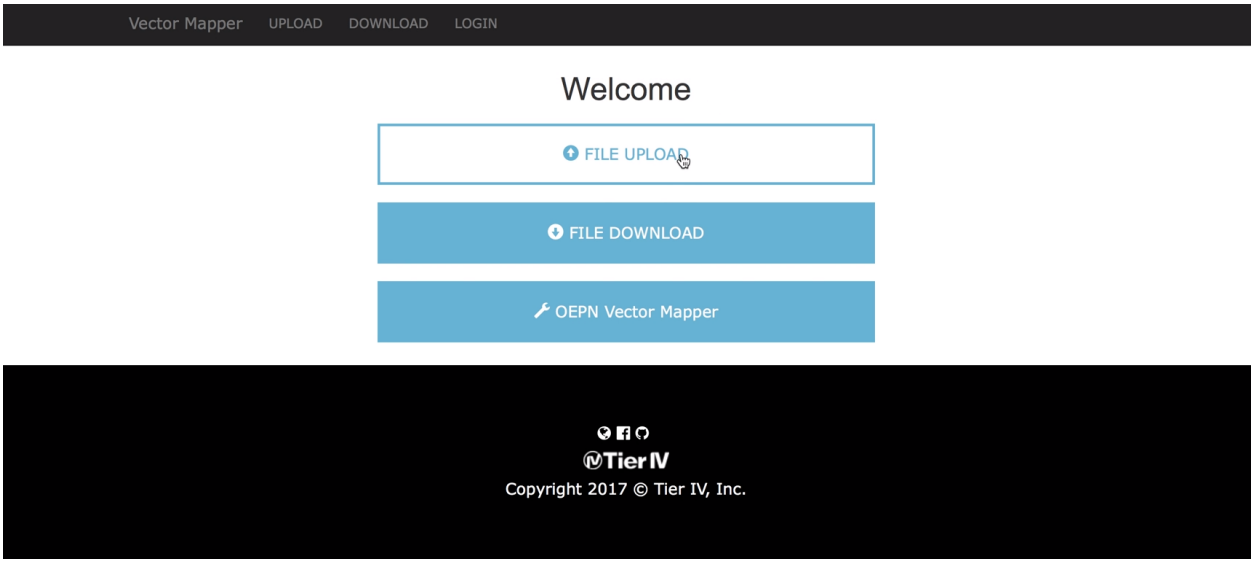
Κάτι πολύ σημαντικό που πρέπει να τονιστεί είναι ότι η διαδικασία δεν είναι πλήρως αυτοματοποιημένη. Αφού διαβαστούν τα *.pcd αρχεία προβάλλεται το αποτέλεσμα αυτών και αρχίζει σταδιακά ο χρήστης να βοηθάει στην δήλωση μαζικών σημείων προκειμένου να δηλωθούν (κομμάτια του χώρου) ως πεζοδιάβαση, εμπόδιο, διαχωριστική κτλ.

Παρακάτω μπορούμε να δούμε ένα χαρακτηριστικό παράδειγμα μιας τέτοιας διαδικασίας :

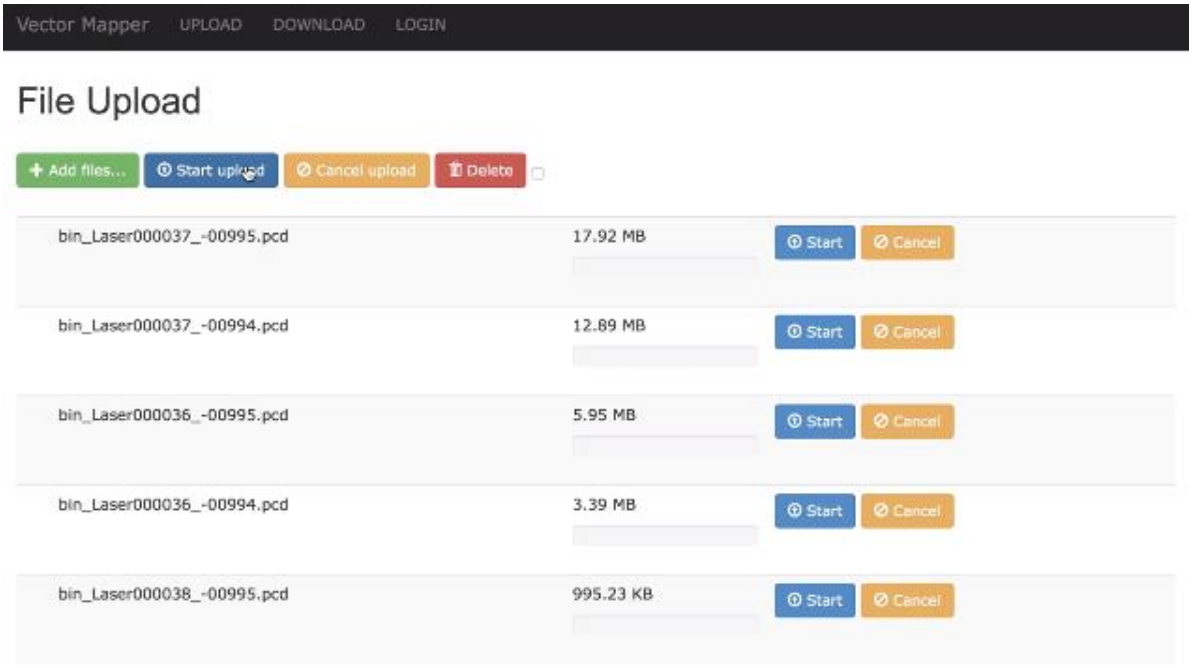
Πρώτα θα πρέπει να εισέλθουμε στην σελίδα :

https://maptools.tier4.jp/vector_mapper_description/

Στην συνέχεια τροφοδοτούμε την γεννήτρια με τα *.pcd αρχεία:

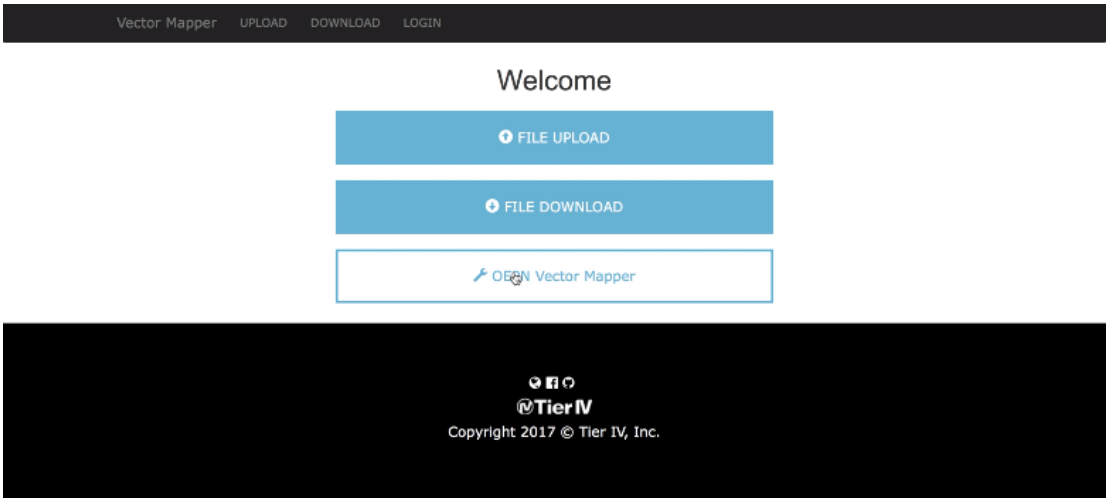


Εικόνα 46 : Upload *.pcd



Εικόνα 47 : Start upload *.pcd

Εφόσον έχουν φορτωθεί τα αρχεία ανοίγουμε τον Vector Mapper



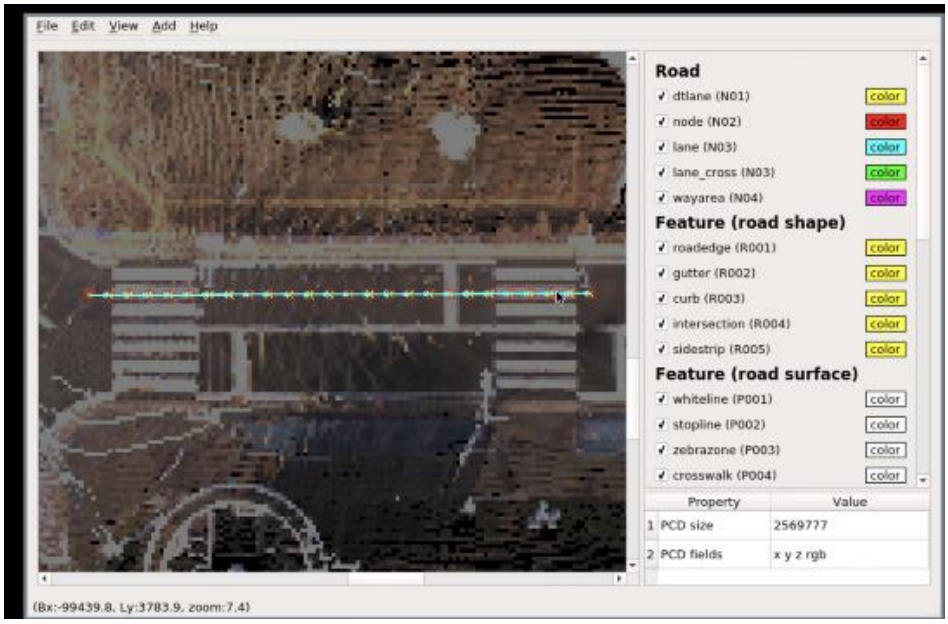
Εικόνα 48 : Open vector mapper

Ανοίγοντας τον verctor mapper, μας προβάλλεται ένα παράθυρο όπως φαίνεται παρακάτω, όπου είναι το σύννεφο από τα *.pcd αρχεία. Από αυτό το σημείο και μετά πλέον ο χρήστης, αρχίζει να ορίζει τις διάφορες οντότητες όπως lanes, waylanes, stoplines, crosswalk κτλ.

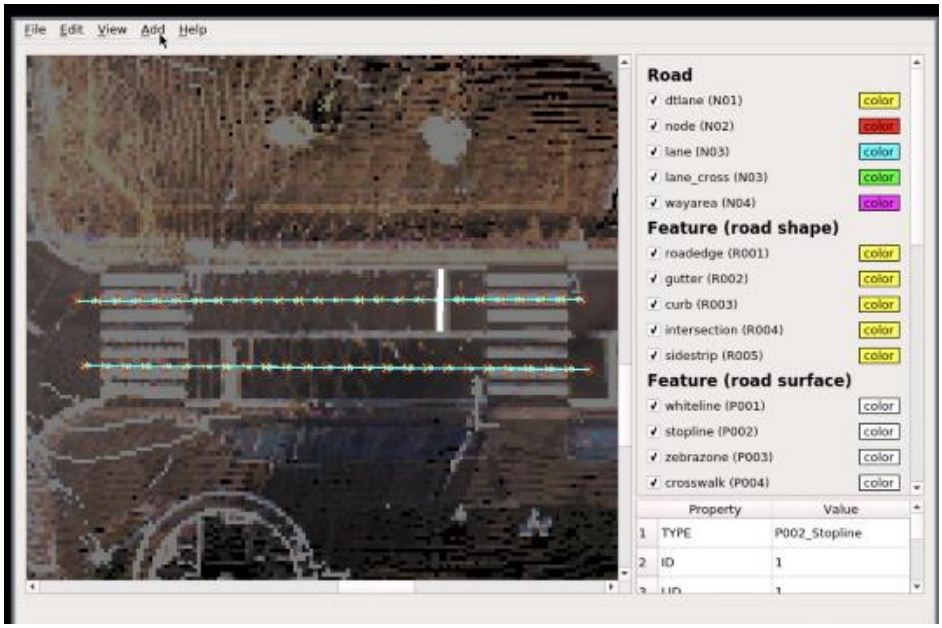
Εικόνα 49 : Open vector mapper

Όλες αυτές οι οντότητες μπορούν να οριστούν από το *Main pull down menu > Add*. Η διαδικασία είναι semi-automatic καθώς δεν είναι ικανές ακόμα οι γεννήτριες να γνωρίζουν ακριβώς αυτό που θέλει να παράξει ο χρήστης.

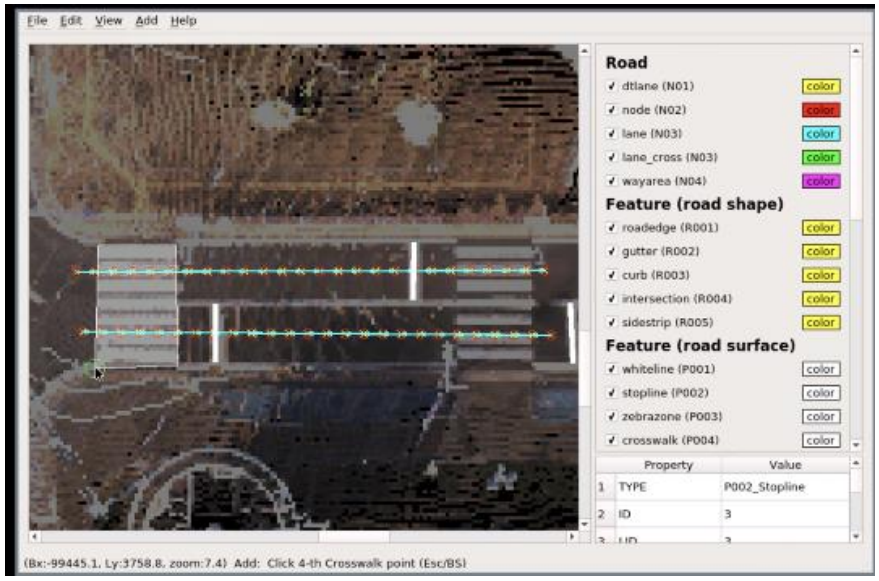
Παρακάτω βλέπουμε πως ορίζουμε ένα *Lane*.



Εικόνα 50 :Ορισμός ενός Lane



Εικόνα 51 : Ορισμός ενός StopLane



Εικόνα 52 : Ορισμός μιας πεζοδιάβασης

Αφού βοηθήσουμε στη δημιουργία της κάθε οντότητας που θα χρειαστούμε να έχουμε, στο τέλος τα αποθηκεύουμε και τα κατεβάζουμε από το site όπου χρησιμοποιήσαμε την γεννήτρια. Αφού τα κατεβάσουμε, τα ορίζουμε σε ένα καθορισμένο directory όπου και θα τα δηλώσουμε στο αντίστοιχο map.launch αρχείο.

Παραδείγματα μορφής των vector αρχείων βλέπουμε παρακάτω:

```
AID,SLID,ELID
1,8480,8583
2,8584,8676
3,8677,8744
4,8745,8969
5,8970,9032
6,9033,9479
7,9480,9651
8,9652,9952
9,9953,10247
10,10248,10411
11,10412,10550
12,10551,10734
13,10735,10809
14,10810,10888
15,10889,10958
```

```
ID,AID,Type,BdID,LinkID
1,408,0,0,11078
2,409,0,0,11065
3,410,0,0,11049
4,411,0,0,11017
5,412,0,0,11007
6,413,0,0,10982
7,414,0,0,10948
8,415,0,0,10938
9,416,0,0,10907
10,417,0,0,10896
11,418,0,0,10871
12,419,0,0,10861
13,420,0,0,10834
14,421,0,0,10825
```

```
DID,Dist,PID,Dir,Apara,r,slope,cant,LW,RW
1,0,10001,0.96680603,0,-313.432,1.5619,1,1.4598,1.4941
2,1,10002,0.96361554,0,-313.432,1.5619,1,1.4545,1.4843
3,2,10003,0.96042505,0,-313.432,1.5619,1,1.4491,1.4745
4,3,10004,0.95723456,0,-313.432,1.5619,1.2007,1.4434,1.4647
5,4,10005,0.95404408,0,-313.432,1.6963,1.4795,1.4408,1.4549
6,5,10006,0.95085359,0,-313.432,1.7157,1.6729,1.4395,1.4419
7,6,10007,0.94766310,0,-313.432,1.7346,1.8605,1.4402,1.4258
8,7,10008,0.94447261,0,-313.432,1.793,2.0564,1.4442,1.412
9,8,10009,0.94128212,0,-313.432,1.91,2.2865,1.4514,1.4269
10,9,10010,0.93809164,0,-313.432,1.9201,2.4358,1.4523,1.4386
11,10,10011,0.93490115,0,-313.432,1.9275,2.5813,1.4524,1.4472
12,11,10012,0.93171066,0,-313.432,1.7229,2.7259,1.4557,1.4526
13,12,10013,0.92852017,0,-313.432,1.8154,2.7236,1.4622,1.4552
14,13,10014,0.92532969,0,-313.432,1.9038,2.5822,1.4634,1.4547
15,14,10015,0.92213920,0,-313.432,2.1075,2.3852,1.463,1.451
```

Εικόνα 53 : area.csv, crosswalk.csv, dtlane.csv

6.2 Launch αρχείο για το sensing

Το sensing αρχείο είναι αυτό με το οποίο το όχημά μας ενεργοποιεί την ικανότητα αναγώρισης :

```
<launch>
  <!-- calibration file path -->
  <arg name="velodyne_calib"
default="Autoware/ros/src/sensing/drivers/lidar/packages/velodyne/velodyne_pointc
loud/params/32db.yaml"/>
  <arg name="camera_calib"
default="Autoware/data/calibration/camera_lidar_3d/prius/nic-150407.yml"/>
  <!-- HDL-32e -->
  <include
file="Autoware/ros/src/sensing/drivers/lidar/packages/velodyne/velodyne_pointclou
d/launch/velodyne_hdl32e.launch">
    <arg name="calibration" value="$(arg velodyne_calib)"/>
  </include>
  <!-- Javad Delta 3 -->
  <!-- <node pkg="javad_navsat_driver" type="gnss.sh" name="javad_driver"/> -->
  <!-- PointGrey Grasshopper3 -->
  <include
file="Autoware/ros/src/sensing/drivers/camera/packages/pointgrey/scripts/grasshop
per3.launch">
    <arg name="CalibrationFile" value="$(arg camera_calib)"/>
  </include>
</launch>
```

Σύμφωνα με τον παραπάνω κώδικα, καλούμε τα δεδομένα από το lidar velodyne καθώς επίσης και την camera. Καθώς τα φορτώνουμε στον launcher τα συγκεκριμένα αρχεία, ακόμα και σε επίπεδο εξομοίωσης, προσπαθούν να ανιχνεύσουν αυτά τα δύο hardware. Αν δεν τα βρουν, απλά ενημερώνουν τον χρήστη με ένα μήνυμα στο terminal.

6.3 Δόμηση του sensing αρχείου

Το αρχείο αυτό είναι υπεύθυνο προκειμένου το όχημα να γνωρίζει την current θέση του ανα πάσα στιγμή με ένα συγκεκριμένο σύστημα συντεταγμένων. Το αρχείο αυτό καλεί τα εξής :

setup_tf.launch
vehicle_model.launch
points_downsample.launch
nmea2tfpose.launch
ndt_matching.launch

Μέσω του setup_tf ορίζουμε ένα σύστημα συντεταγμένων στον χώρο καθώς επίσης και που βρίσκεται στον χώρο. Από το vehicle_model αρχείο καλούμε το όχημα που θέλουμε να φορτώσουμε. Στο points_downsample αρχείο αρχικοποιούμε κάποιες μεταβλητές. Και με τα αρχεία nmea2tfpose και ndt_matching κάνουμε matching τα δεδομένα cameras και lidar.

6.4 Launch αρχείο για το detection

Εφόσον πλέον έχουμε τον χώρο μέσα στον οποίο μπορεί να κινηθεί το όχημά μας, καθώς επίσης το όχημά μας έχει αντίληψη του χώρου γύρω του μέσω του sensing file και το που βρίσκεται μέσω του localization file, σειρά έχει να ενεργοποιήσουμε το 'detecting' του οχήματός μας, την ικανότητα δηλαδή να μπορεί να αντιλαμβάνεται τα διάφορα αντικείμενα γύρω του. Αυτό μπορούμε να περιγράψουμε μέσω του αρχείου my_detection.launch.

Αναλύοντας το παραπάνω αρχείο ξεκινάμε από κάποιες αρχικοποιήσεις που χρειάζονται:

```
<!-- setting of this launch file -->
  <arg name="car_detection" default="true" />
  <arg name="pedestrian_detection" default="false" />
  <arg name="is_use_gpu" default="true" />
  <arg name="is_register_lidar2camera_tf" default="true" />
  <arg name="is_publish_projection_matrix" default="true" />
  <arg name="is_publish_camera_info" default="true" />
  <arg name="camera_calib"
default="Autoware/data/calibration/camera_lidar_3d/prius/nic-150407.yml"/>
```

Στη συνέχεια θα πρέπει να καλιμπράρουμε τα topics που θα κάνουμε publish:

```
<!-- calibration_publisher -->
  <include
file="Autoware/ros/src/util/packages/runtime_manager/scripts/calibration_publishe
r.launch">
    <arg name="file" value="$(arg camera_calib)" />
    <arg name="register_lidar2camera_tf" value="$(arg
is_register_lidar2camera_tf)" />
    <arg name="publish_extrinsic_mat" value="$(arg is_publish_projection_matrix)"
/>
    <arg name="publish_camera_info" value="$(arg is_publish_camera_info)" />
  </include>
```

Σειρά έχει η ανίχνευση οχημάτων και πεζών:

```
<!-- car and pedestrian detection -->
  <!-- dpm_ttic -->
  <include
file="Autoware/ros/src/computing/perception/detection/vision_detector/packages/vi
sion_dpm_ttic_detect/launch/vision_dpm_ttic_detect.launch">
    <arg name="car" value="$(arg car_detection)" />
    <arg name="pedestrian" value="$(arg pedestrian_detection)" />
    <arg name="use_gpu" value="$(arg is_use_gpu)" />
  </include>
```

Προφανώς από αυτό το κομμάτι δεν φαίνονται πολλά. Πολύ απλά καθώς την κυρίως δουλειά την κάνει το vision_dpm_ttic_detect.launch αρχείο.

Αναλύοντας το παραπάνω αρχείο μπορούμε να δούμε ότι το κάθε όχημα και πεζός ορίζονται ως οντότητες στο σύστημα κατόπιν επεξεργασίας των ανάλογων *.csv αρχείων. Π.χ

Ορισμός αυτοκινήτων:

```
<group if="$(arg car)">
    <group ns="obj_car">

      <!-- arguments list -->
      <arg name="comp_model_car" default="$(find
vision_dpm_ttic_detect)/data/car_comp.csv"/>
      <arg name="root_model_car" default="$(find
vision_dpm_ttic_detect)/data/car_root.csv"/>
      <arg name="part_model_car" default="$(find
vision_dpm_ttic_detect)/data/car_part.csv"/>
      <arg name="image_src_car" default="/image_raw"/>
```

Ορισμός των οχημάτων ως οντότητες

Διάβασμα των αρχείων

Δημιουργία μίας εικόνας

Τα *.csv αποτελέσματα τα προωθούμε στο dmp detection function:

```
<!-- dpm detection -->
      <node pkg="vision_dpm_ttic_detect" name="vision_dpm_ttic_detect"
type="vision_dpm_ttic_detect">
        <remap from="/config/obj_car/dpm" to="/config/car_dpm"/>
        <param name="detection_class_name" type="str" value="car"/>
        <param name="comp_model_path" type="str" value="$(arg comp_model_car)"/>
        <param name="root_model_path" type="str" value="$(arg root_model_car)"/>
        <param name="part_model_path" type="str" value="$(arg part_model_car)"/>
        <param name="use_gpu" type="bool" value="$(arg use_gpu)"/>
        <param name="image_raw_topic" type="str" value="$(arg camera_id)$(arg
image_src_car)"/>
        <remap from="/image_raw" to="/sync_drivers/image_raw" if="$(arg sync)" />
      </node>
    </group>
```

Συνεχίζουμε αντίστοιχα και με τον ορισμό των πεζών όπου δίνουμε τα *.csv αρχεία και τα προωθούμε στο dmp detection function:

```
<group if="$(arg pedestrian)">
    <group ns="obj_person">
```

Ορισμός των πεζών ως οντότητες

```
      <!-- arguments list -->
      <arg name="comp_model_pedestrian" default="$(find
vision_dpm_ttic_detect)/data/person_comp.csv"/>
      <arg name="root_model_pedestrian" default="$(find
vision_dpm_ttic_detect)/data/person_root.csv"/>
      <arg name="part_model_pedestrian" default="$(find
vision_dpm_ttic_detect)/data/person_part.csv"/>
      <arg name="image_src_pedestrian" default="/image_raw"/>
```

Διάβασμα των αρχείων

Δημιουργία μίας εικόνας

```
<!-- dpm detection -->
      <node pkg="vision_dpm_ttic_detect" name="vision_dpm_ttic_detect"
type="vision_dpm_ttic_detect">
        <remap from="/config/obj_person/dpm" to="/config/pedestrian_dpm"/>
        <param name="detection_class_name" type="str" value="person"/>
        <param name="comp_model_path" type="str" value="$(arg
comp_model_pedestrian)"/>
        <param name="root_model_path" type="str" value="$(arg
root_model_pedestrian)"/>
        <param name="part_model_path" type="str" value="$(arg
part_model_pedestrian)"/>
        <param name="use_gpu" type="bool" value="$(arg use_gpu)"/>
        <param name="image_raw_topic" type="str" value="$(arg camera_id)$(arg
image_src_pedestrian)"/>
        <remap from="/image_raw" to="/sync_drivers/image_raw" if="$(arg sync)" />
      </node>
    </group>
```

Στην πορεία θα πρέπει για τις παραπάνω οντότητες να δηλωθεί: ένα range fusion, ένα vision tracking και ένα object reprojection.

```
<!-- range_fusion -->
  <include
file="Autoware/ros/src/computing/perception/detection/lidar_tracker/packages/rang
e_fusion/launch/range_fusion.launch">
    <arg name="car" value="$(arg car_detection)" />
    <arg name="pedestrian" value="$(arg pedestrian_detection)" />
  </include>
  <!-- vision_klt_track -->
```

```
  <include
file="Autoware/ros/src/computing/perception/detection/vision_tracker/packages/vis
ion_klt_track/launch/vision_klt_track.launch">
    <arg name="car" value="$(arg car_detection)" />
    <arg name="pedestrian" value="$(arg pedestrian_detection)" />
  </include>
  <!-- obj_reproj -->
  <include
file="Autoware/ros/src/computing/perception/detection/lidar_tracker/packages/obj_
reproj/launch/obj_reproj.launch">
    <arg name="car" value="$(arg car_detection)" />
    <arg name="pedestrian" value="$(arg pedestrian_detection)" />
  </include>
```

Λαμβάνοντας υπ όψιν τα παραπάνω αρχεία ξεκινάμε τους απαραίτητους υπολογισμούς ελαχίστων αποστάσεων χάρις την Ευκλείδια διαίρεση :

```
 <!-- lidar_euclidean_cluster_detect -->
  <include
file="Autoware/ros/src/computing/perception/detection/lidar_detector/packages/lid
ar_euclidean_cluster_detect/launch/lidar_euclidean_cluster_detect.launch">
  </include>
  <!-- obj_fusion -->
  <include
file="Autoware/ros/src/computing/perception/detection/lidar_tracker/packages/obj_
fusion/launch/obj_fusion.launch">
    <arg name="car" value="$(arg car_detection)" />
    <arg name="pedestrian" value="$(arg pedestrian_detection)" />
  </include>
```

Το light traffic recognition είναι ακόμα σε βασικό στάδιο εξέλιξης οπότε δεν το λαμβάνουμε υπ όψιν.

6.5 Launch αρχείο για την αποστολή της διαδρομής

Εφόσον πλέον το όχημά μας γνωρίζει που βρίσκεται στον χώρο και τα αντικείμενα που έχει γύρω του, σειρά έχει να του ορίσουμε τη διαδρομή που έχει να κάνει με βάση τον χάρτη που έχουμε φορτώσει. Σαν πρώτη κίνηση αρχικοποιούμε την θέση (pose) του οχήματος όπως θα κάναμε για οποιοδήποτε ρομπότ :

```
<launch>
  <!-- setting path parameter -->
  <arg name="topic_pose_stamped" default="/ndt_pose" />
  <arg name="topic_twist_stamped" default="/estimate_twist" />
  <!-- Tablet UI -->
  <!--
```

```
  <include
file="Autoware/ros/src/util/packages/runtime_manager/scripts/tablet_socket.launch
"/>
  -->
  <!-- vel_pose_mux -->
  <include
file="Autoware/ros/src/computing/perception/localization/packages/autoware_connec
tor/launch/vel_pose_connect.launch">
    <arg name="topic_pose_stamped" value="$(arg topic_pose_stamped)" />
    <arg name="topic_twist_stamped" value="$(arg topic_twist_stamped)" />
  </include>
```

Στην πορεία πλέον θα πρέπει να του ορίσουμε την διαδρομή που έχει να ακολουθήσει με βάσει κάποια σημεία. Αυτό πραγματοποιείται μέσω ενός *.csv αρχείου, όπου πέρα από τα σημεία έχουμε και μία τέταρτη στήλη όπου είναι η επιθυμητή ταχύτητα που θέλουμε να κρατήσει το όχημα. Πέρα όμως από αυτά τα δεδομένα, επειδή το όχημα κατά την διαδρομή του θα συναντά εμπόδια, αλλαγές κατεύθυνσης, πεζοδιαβάσεις κτλ, θα χρειαστεί να επαναυπολογίζει ξανά και ξανά δεδομένα όπως : ονομαστική ταχύτητα, μέγιστη ταχύτητα, μέγιστη και ελάχιστη επιτάχυνση κτλ. Όλα αυτά τα ορίζουμε μέσα σε ένα dictionary της python όπου και θα ληφθούν υπ όψιν κατά το simulation :

```
<!-- waypoint_loader -->
  <node pkg="rostopic" type="rostopic" name="config_waypoint_loader_rostopic"
        args="pub -l /config/waypoint_loader autoware_msgs/ConfigWaypointLoader
        '{multi_lane_csv: 'Autoware/data/path/moriyama_path.txt',
replanning_mode: false, velocity_max: 20.0, velocity_min: 4.0, accel_limit: 0.98,
decel_limit: 0.98, radius_thresh: 20.0, radius_min: 6.0, resample_mode: true,
resample_interval: 1.0, velocity_offset: 4, end_point_offset: 5}' "
  />
  <include
file="Autoware/ros/src/computing/planning/motion/packages/waypoint_maker/launch/w
aypoint_loader.launch" />
```

Καθώς θα υπάρχουν εναλλαγές συνεχώς, συνεχώς θα πρέπει να υπολογίζουμε και τα καινούργια waypoints : *waypoint_loader.launch*

6.6 Launch αρχείο για το motion

Τέλος, εφόσον έχουμε ορίζει τον τρόπο μετακίνησης του οχήματός μας, θα πρέπει να ορίζουμε και τον τρόπο υπολογισμού την καλύτερης δυνατής απόστασης και ταχύτητας. Εδώ παίρνει θέση ο αρκετά γνωστός εδώ και χρόνια αλγόριθμος A* (alpha_star_planner). Κατά το simulation έχουμε πάντα δύο συστήματα συντεταγμένων: αυτό που ακολουθεί την ιδανική κατάσταση (που έχουμε ορίσει) και το «πραγματικό» όπου δηλώνει την current *κατάσταση* του οχήματος (*κατάσταση* = θέση και ταχύτητα).

```
<launch>
  <!-- Vehicle Contorl -->
  <include
file="Autoware/ros/src/util/packages/runtime_manager/scripts/vehicle_socket.launc
h"/>
  <!-- obstacle_avoid -->
  <include
file="Autoware/ros/src/computing/planning/motion/packages/astar_planner/launch/ob
stacle_avoid.launch"/>
  <!-- velocity_set -->
  <include
file="Autoware/ros/src/computing/planning/motion/packages/astar_planner/launch/ve
locity_set.launch"/>
  <!-- pure_pursuit -->
  <node pkg="rostopic" type="rostopic" name="config_waypoint_follower_rostopic"
        args="pub -l /config/waypoint_follower
autoware_msgs/ConfigWaypointFollower
        '{ header: auto, param_flag: 1, velocity: 5.0, lookahead_distance: 4.0,
lookahead_ratio: 2.0, minimum_lookahead_distance: 6.0, displacement_threshold:
0.0, relative_angle_threshold: 0.0 }' "
  />
  <include
file="Autoware/ros/src/computing/planning/motion/packages/waypoint_follower/launc
h/pure_pursuit.launch"/>
  <!-- twist_filter -->
  <include
file="Autoware/ros/src/computing/planning/motion/packages/waypoint_follower/launc
h/twist_filter.launch"/>
</launch>
```

Κεφάλαιο 7ο

A* planner Αλγόριθμος

7.1 Περιγραφή του α* planner

Ο A* [37] είναι ένας αλγόριθμος ηλεκτρονικού υπολογιστή που χρησιμοποιείται ευρέως στο pathfinding και στο graphtraversal, όπου είναι η διαδικασία εύρεσης διαδρομής μεταξύ πολλαπλών σημείων, που ονομάζονται "κόμβοι". Έχει ευρεία χρήση λόγω της απόδοσης και της ακρίβειάς του. Οι Peter Hart, Nils Nilsson και Bertram Raphael του Ινστιτούτου Ερευνών του Stanford δημοσίευσαν για πρώτη φορά τον αλγόριθμο το 1968. Μπορεί να θεωρηθεί ως επέκταση του αλγορίθμου του 1959 του Edsger Dijkstra. Ο A* επιτυγχάνει καλύτερη απόδοση χρησιμοποιώντας ευρετικά για να καθοδηγήσει την έρευνά του.

7.2 Ανάλυση του α* planner

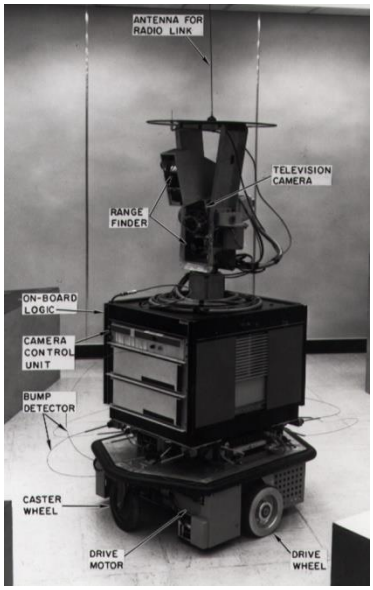
Ο A* είναι ένας αλγόριθμος αναζήτησης ή αλλιώς best-find-search, που σημαίνει ότι είναι διατυπωμένος με όρους σταθμισμένων γραφημάτων: ξεκινώντας από έναν συγκεκριμένο κόμβο ενός γραφήματος, στοχεύει να βρει μια διαδρομή προς τον συγκεκριμένο κόμβο που έχει το μικρότερο κόστους (ελάχιστη απόσταση που διανύθηκε, συντομότερη διάρκεια κ.λπ.). Αυτό επιτυγχάνεται διατηρώντας ένα δέντρο μονοπατιών, που προέρχονται από τον κόμβο εκκίνησης και επεκτείνοντας αυτές τις διαδρομές μία άκρη κάθε φορά, μέχρι να ικανοποιηθεί το κριτήριο τερματισμού.

Σε κάθε επανάληψη του κύριου βρόχου, ο A* πρέπει να καθορίσει ποια από τα μονοπάτια του να επεκταθούν. Αυτό γίνεται με βάση το κόστος της διαδρομής και μια εκτίμηση του κόστους που απαιτείται για την επέκταση της διαδρομής μέχρι το στόχο. Συγκεκριμένα, ο A* επιλέγει τη διαδρομή που ελαχιστοποιεί το:

f (n) = g (n) + h (n)

όπου n είναι ο επόμενος κόμβος στη διαδρομή, το g (n) είναι το κόστος της διαδρομής από τον κόμβο έναρξης ως το n και το h (n) είναι μια ευρηστική συνάρτηση που υπολογίζει το κόστος της ελάχιστης διαδρομής από το n στο στόχο. Ο A* τερματίζεται όταν η διαδρομή που επιλέγει να ακολουθήσει είναι: μια διαδρομή από την αρχή προς το στόχο, ή εάν δεν υπάρχουν διαδρομές που είναι επιλέξιμες για επέκταση. Η ευρηστική λειτουργία (heuristic function) είναι problem-specific.

Οι τυπικές υλοποιήσεις του A* χρησιμοποιούν μια ουρά προτεραιότητας για να εκτελέσουν την επαναλαμβανόμενη επιλογή των ελάχιστων (εκτιμώμενων) κόμβων κόστους για επέκταση. Αυτή η ουρά προτεραιότητας είναι γνωστή ως *open-set* ή *fringe*. Σε κάθε βήμα του αλγορίθμου, ο κόμβος με την χαμηλότερη τιμή f (x) αφαιρείται από την ουρά, οι τιμές f και g των γειτόνων του ενημερώνονται αναλόγως και αυτοί οι γείτονες προστίθενται στην ουρά. Ο αλγόριθμος συνεχίζεται έως ότου ένας target κόμβος έχει την χαμηλότερη τιμή f από οποιονδήποτε κόμβο στην ουρά (ή έως ότου η ουρά είναι κενή). Η τιμή f του στόχου είναι τότε το κόστος της μικρότερης διαδρομής, δεδομένου ότι το h στο στόχο είναι μηδέν σε αποδεκτή ευρετική (*heuristic*).



Εικόνα 54 : SRI's Shakey, το πρώτο robot όπου μπορούσε να πάρει αποφάσεις κίνησης μόνο του

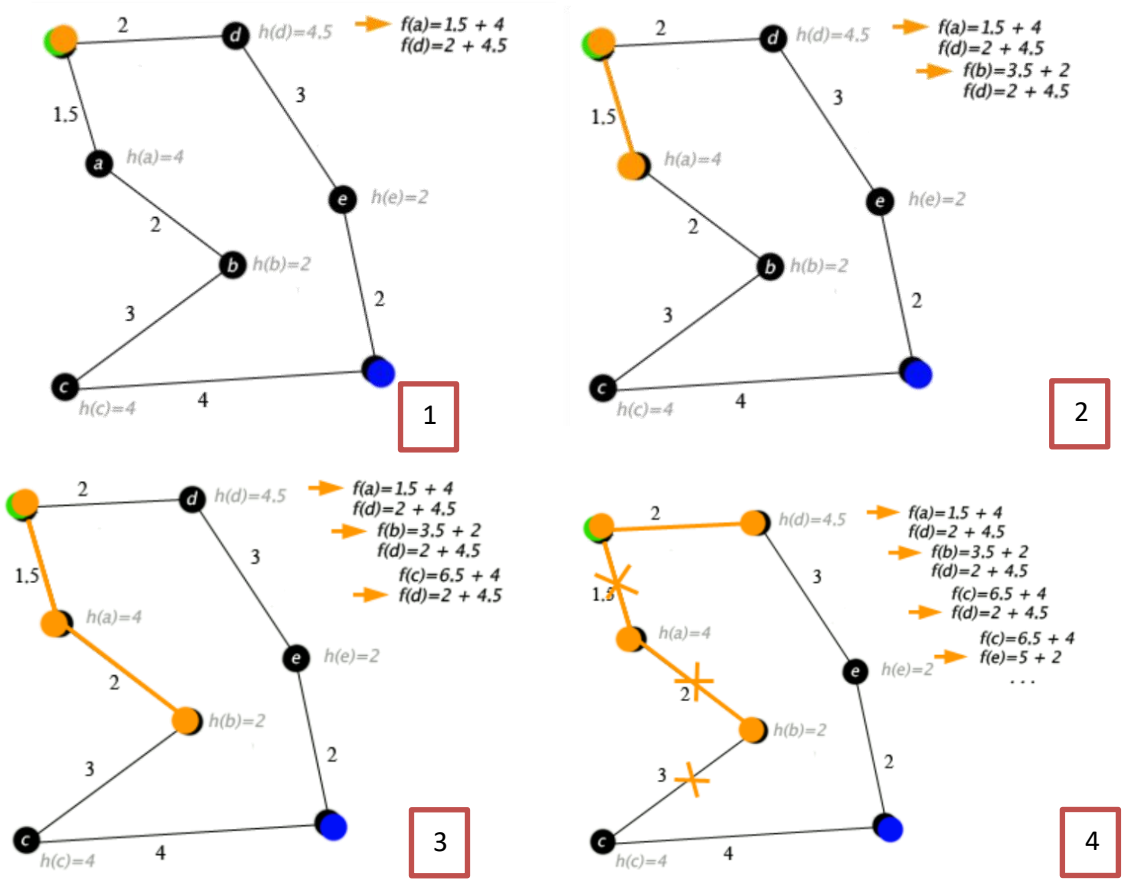
Ο αλγόριθμος που περιγράψαμε μέχρι τώρα μας δίνει μόνο το μήκος της μικρότερης διαδρομής. Για να βρούμε την συνολική ακολουθία βημάτων (κόμβων) είναι:

- ο κάθε νέος κόμβος λαμβάνει υπ όψιν του και τους προκατόχους του
- Αφού εκτελεστεί η κίνηση του συγκεκριμένου κόμβου θα δείχνει τις πληροφορίες του στον επόμενο κόμβο
- Μέχρι που ο καθορισμένος κόμβος να είναι ο βέλτιστος

Για παράδειγμα: κατά την αναζήτηση της συντομότερης διαδρομής σε έναν χάρτη, το h (x) μπορεί να αντιπροσωπεύει την ευθεία απόσταση από το στόχο, αφού αυτή είναι φυσικά η μικρότερη δυνατή απόσταση μεταξύ οποιωνδήποτε δύο σημείων.

Αν το h ικανοποιεί την επιπρόσθετη συνθήκη h (x) ≤ d (x, y) + h (y) για κάθε ακμή (x, y) του γραφήματος (όπου d δηλώνει το μήκος αυτής της ακμής) τότε το h ονομάζεται monotone ή consistent. Σε μια τέτοια περίπτωση, ο A* μπορεί να εφαρμοστεί πιο αποτελεσματικά, καθώς κανένας κόμβος δεν χρειάζεται να επεξεργαστεί περισσότερες από μία φορά. Εδώ ο A* ισοδυναμεί με τον αλγόριθμο του Dijkstra : d '(x, y) = d (x, y) + h (y) -h (x).

Παρακάτω βλέπουμε ένα χαρακτηριστικό παράδειγμα του A* όπου βρίσκει την κοντινότερη διαδρομή μεταξύ κάποιων πόλεων:



Εικόνα 55 : Παράδειγμα του A*

Όπως και στην πρώτη αναζήτηση, ο A * θα βρει πάντα μια λύση (εάν υπάρχει) :

d(x,y)>ε >0 για σταθερό ε

Εάν η h είναι αποδεκτή, δηλαδή δεν υπερτιμά ποτέ πραγματικό ελάχιστο κόστος επίτευξης του στόχου, τότε ο A* είναι αποδεκτός (εάν δεν χρησιμοποιούμε close set). Εάν χρησιμοποιείται close set, τότε το h πρέπει επίσης να είναι motone ή consistent για να είναι ο A* βέλτιστος. Αυτό σημαίνει ότι για οποιοδήποτε ζεύγος γειτονικών κόμβων x και y, όπου:

d (x, y) δηλώνει το μήκος της άκρης μεταξύ τους, πρέπει να έχουμε:

$h(x) \leq d(x,y) + h(y)$

Αυτό μας επιβεβαιώνει ότι για κάθε path X από τον αρχικό κόμβο, το x:

$L(X) + h(x) \leq L(X) + d(x,y) + h(y) = L(Y) + h(y)$

όπου L είναι μια συνάρτηση που υποδηλώνει το μήκος μιας διαδρομής και το Y είναι η διαδρομή X που έχει επεκταθεί για να συμπεριλάβει το y. Με άλλα λόγια, είναι αδύνατο να μειωθεί (η συνολική απόσταση μέχρι τώρα συν η εκτιμώμενη υπόλοιπη απόσταση) επεκτείνοντας μια διαδρομή για να συμπεριλάβει έναν γειτονικό κόμβο. (Αυτό είναι ανάλογο με

τον περιορισμό στα μη αρνητικά βάρη άκρων στον αλγόριθμο Dijkstra.) Η μονοτονικότητα συνεπάγεται παραδεκτότητα όταν η ευρετική εκτίμηση σε οποιοδήποτε κόμβο είναι μηδέν. [αφήνοντας P = (f, v1, v2, ..., vn, g) είναι μια συντομότερη διαδρομή από οποιοδήποτε κόμβο f στον πλησιέστερο στόχο g].

$h(f) \leq d(f,u_1) + h(v_1) \leq d(f,v_1) + d(v_1,v_2) + h(v_2) \leq \ldots \leq L(P) + h(g) = L(P)$

7.3 Εφαρμογή του A* planner αλγόριθμου

Σύμφωνα με το αρχείο **motion_planning.launch* που χρησιμοποιούμε στον Runtime Manager, προσδιοριορίζεται η κίνηση του οχήματος καθώς και ο επαναυπολογισμός του σε κάθε time frame. Ο προσδιορισμός της κίνησής του όπως είπαμε παραπάνω ορίζεται από ένα *.csv αρχείο ενταγμένο σαν argument σε ένα dictionary, μαζί με διάφορα άλλα arguments. Στο ίδιο αρχείο όμως καλούνται και δύο άλλα *.launch files όπου είναι σχετισμένα με την αποφυγή εμποδίων και ορισμού της ταχύτητας του οχήματος για κάθε περίπτωση όπου θα βρει εμπόδιο. Ξεκινώντας με το αρχείο * *obstacle_avoid.launch* παρατηρούμε ότι είναι ένα αρχείο όπου αρχικοποιούνται όλες οι τιμές που χρειαζόμαστε σχετικά με τον επαναυπολογισμό οποιουδήποτε εμποδίου που λαμβάνουμε υπ όψιν π.χ. : avoid_distance, avoid_velocity_limit_mps, minimum_turning_radius κτλ.

```
<launch>

  <arg name="use_2dnav_goal" default="false" />
  <arg name="map_frame" default="map" />
  <arg name="angle_size" default="42" />
  <arg name="minimum_turning_radius" default="5.5" />
  <arg name="obstacle_threshold" default="70" />
  <arg name="robot_length" default="5.0" />
  <arg name="robot_width" default="4.0" />
  <arg name="base2back" default="0.8" />
  <arg name="curve_weight" default="1.00" />
  <arg name="reverse_weight" default="2.50" />
  <arg name="distance_heuristic_weight" default="1.0" />
  <arg name="potential_weight" default="10.0" />
  <arg name="lateral_goal_range" default="0.5" /> <!-- meter -->
  <arg name="longitudinal_goal_range" default="2.0" /> <!-- meter -->
  <arg name="goal_angle_range" default="48.0" /> <!-- degree -->
  <arg name="time_limit" default="5.0" />
  <arg name="use_back" default="false" />
  <arg name="use_wavefront_heuristic" default="false" />
  <arg name="use_potential_heuristic" default="true" />
  <arg name="publish_marker" default="true" />

  <!-- params for search_info -->
  <arg name="obstacle_detect_count" default="8" />
  <arg name="avoid_distance" default="13" />
```

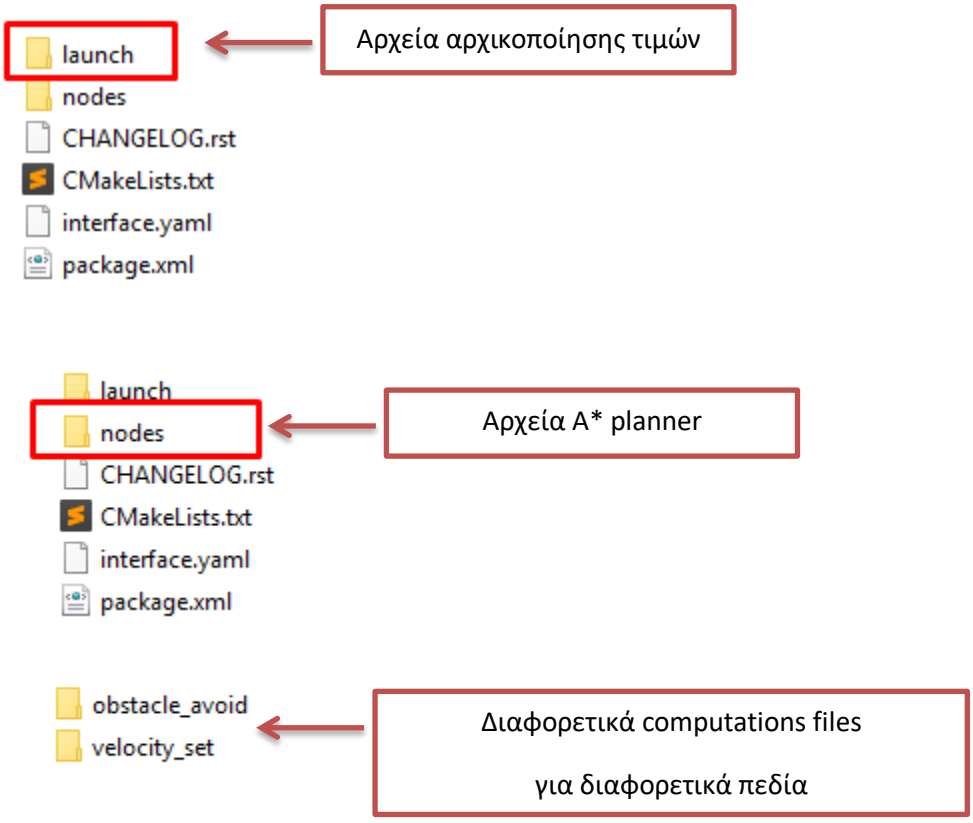
```
  <arg name="avoid_velocity_limit_mps" default="4.5" />
  <arg name="upper_bound_ratio" default="1.20" />
  <arg name="avoidance" default="true" /> <!-- always avoid regardless of current
state -->
  <arg name="switch_path" default="true" /> <!-- generate avoiding path and
follow it -->
  <node pkg="astar_planner" type="obstacle_avoid" name="obstacle_avoid"
output="log">
    <param name="use_2dnav_goal" value="$(arg use_2dnav_goal)" />
    <param name="map_frame" value="$(arg map_frame)" />
    <param name="angle_size" value="$(arg angle_size)" />
    <param name="minimum_turning_radius" value="$(arg minimum_turning_radius)" />
    <param name="obstacle_threshold" value="$(arg obstacle_threshold)" />
    <param name="use_back" value="$(arg use_back)" />
    <param name="robot_length" value="$(arg robot_length)" />
    <param name="robot_width" value="$(arg robot_width)" />
    <param name="base2back" value="$(arg base2back)" />
    <param name="curve_weight" value="$(arg curve_weight)" />
    <param name="reverse_weight" value="$(arg reverse_weight)" />
    <param name="distance_heuristic_weight" value="$(arg
distance_heuristic_weight)" />
    <param name="potential_weight" value="$(arg potential_weight)" />
    <param name="time_limit" value="$(arg time_limit)" />
    <param name="lateral_goal_range" value="$(arg lateral_goal_range)" />
    <param name="longitudinal_goal_range" value="$(arg longitudinal_goal_range)"
/>
    <param name="goal_angle_range" value="$(arg goal_angle_range)" />
    <param name="use_wavefront_heuristic" value="$(arg use_wavefront_heuristic)"
/>
    <param name="use_potential_heuristic" value="$(arg use_potential_heuristic)"
/>
    <param name="publish_marker" value="$(arg publish_marker)" />

    <param name="obstacle_detect_count" value="$(arg obstacle_detect_count)" />
    <param name="avoid_distance" value="$(arg avoid_distance)" />
    <param name="avoid_velocity_limit_mps" value="$(arg
avoid_velocity_limit_mps)" />
    <param name="upper_bound_ratio" value="$(arg upper_bound_ratio)" />
    <param name="avoidance" value="$(arg avoidance)" />
    <param name="change_path" value="$(arg switch_path)" />
  </node>

  <!-- obstacle simulation node -->
  <arg name="obstacle_height" default="1.0" />
  <arg name="obstacle_width" default="1.0" />
  <arg name="points_interval" default="0.1" />
```

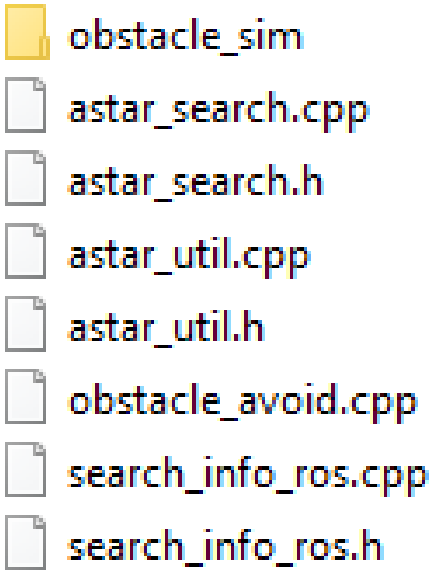
```
  <arg name="obstacle_frame" default="/velodyne" />
  <node pkg="astar_planner" type="obstacle_sim" name="obstacle_sim"
output="screen">
    <param name="obstacle_height" value="$(arg obstacle_height)" />
    <param name="obstacle_width" value="$(arg obstacle_width)" />
    <param name="points_interval" value="$(arg points_interval)" />
    <param name="obstacle_frame" value="$(arg obstacle_frame)" />
  </node>

</launch>
```

Και αντίστοιχα το **velocity_set.launch:*

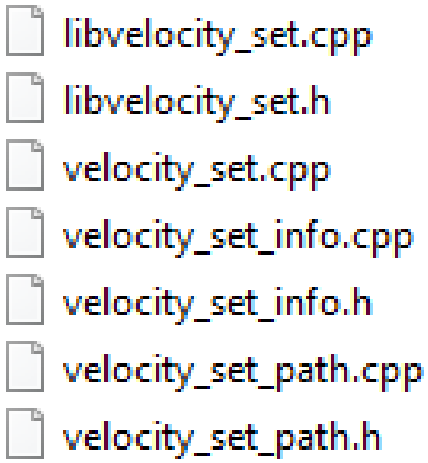
```
<!-- -->
<launch>
  <arg name="use_crosswalk_detection" default="true" />
  <arg name="points_topic" default="points_no_ground" />
  <arg name="velocity_offset" default="1.2" />
  <arg name="decelerate_vel_min" default="1.3" />
  <arg name="remove_points_upto" default="2.3" />
  <arg name="enable_multiple_crosswalk_detection" default="true" />
  <arg name="enablePlannerDynamicSwitch" default="false" />

  <node pkg="astar_planner" type="velocity_set" name="velocity_set"
output="screen">
    <param name="use_crosswalk_detection" value="$(arg use_crosswalk_detection)"
/>
    <param name="enable_multiple_crosswalk_detection" value="$(arg
enable_multiple_crosswalk_detection)" />
    <param name="points_topic" value="$(arg points_topic)" />
    <param name="velocity_offset" value="$(arg velocity_offset)" />
    <param name="decelerate_vel_min" value="$(arg decelerate_vel_min)" />
    <param name="remove_points_upto" value="$(arg remove_points_upto)" />
    <param name="enablePlannerDynamicSwitch" value="$(arg
enablePlannerDynamicSwitch)" />
  </node>

</launch>
```

Οπότε μέχρι τώρα έχουμε αρχικοποιήσει απλά τις παραμέτρους μας. Όλοι οι υπολογισμοί με βάση τον A* planner πραγματοποιούνται από ένα πακέτο αρχείων όπου βρίσκονται σ'ένα φάκελο πριν τα launch files :





Εικόνα 56 : C++ αρχεία για τον υπολογισμό κατά των εμποδίων



Εικόνα 57 : C++ αρχεία για τον υπολογισμό της ταχύτητας

Κεφάλαιο 8ο

Προσομοίωση

Προφανώς, πέρα από αυτή τη βασική προσομοίωση, ο κάθε χρήστης μπορεί να βρει διάφορα video στο διαδίκτυο για να τρέξει διάφορες προσομοιώσεις:

https://github.com/CPFL/Autoware/wiki/videos

Εφόσον πλέον έχουμε αναλύσει σε διάφορα στάδια τον κώδικα, προχωράμε στο επόμενο βήμα, όπου είναι και το κυριότερο : πως πραγματοποιείται μία προσομοίωση και πως μπορεί να παραμετροποιηθεί.

8.1 Αρχεία προσομοίωσης

Τα demo αρχεία που χρησιμοποιήθηκαν είναι τα παρακάτω:
Quick start : https://github.com/CPFL/Autoware/tree/master/docs/quick_start
Προκειμένου να τρέξουμε μία ήδη υπάρχουσα προσομοίωση θα πρέπει να έχουμε τα εξής αρχεια:

- 3D map and ROSBAG data
  - https://www.autoware.ai/sample/sample_moriyama_data.tar.gz
- ROSBAG sample data : LiDAR: VELODYNE HDL-32E, Camera: PointGrey Grasshopper 3, GNSS: JAVAD GPS RTK Delta 3
  - https://www.autoware.ai/sample/sample_moriyama_150324.tar.gz

Για περισσότερα δεδομένα μπορούμε να χρησιμοποιήσουμε το site ROSBAG STORE.

https://rosbag.tier4.jp/

8.2 Εκτέλεση ενός Simulation

Προκειμένου να τρέξουμε την προσομοίωση εκτελούμε τις παρακάτω εντολές σε ένα terminal:

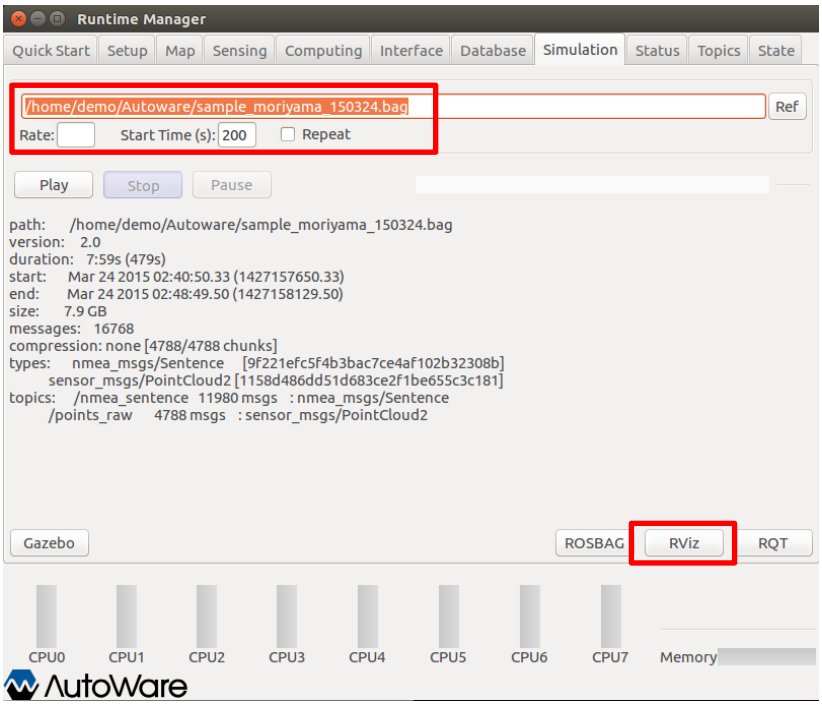
```
$ cd $HOME/Autoware/ros
$ ./run
```

Σ αυτό το σημείο ανοίγει ο **Runtime Manager**

Εδώ πλέον για να τρέξουμε τα demo αρχεία, δηλώνουμε στο tab **Simulation** το αρχείο *sample_moriyama_150324.bag*

Αυτού του είδους τα αρχεία εμπεριέχουν όλα τα απαραίτητα nodes, topicss, serialized messages data που χρειάζονται να τρέχουν στο background καθώς θα τρέχει η προσομοίωση. Όλα αυτά τα nodes τροφοδοτούνται on-the-fly με την αντίστοιχη πληροφορία και ο κάθε χρήστης μπορεί να την αντλήσει, προκειμένου να ελέγξει την προσομοίωση ή ακόμα και για να την χρησιμοποιήσει.

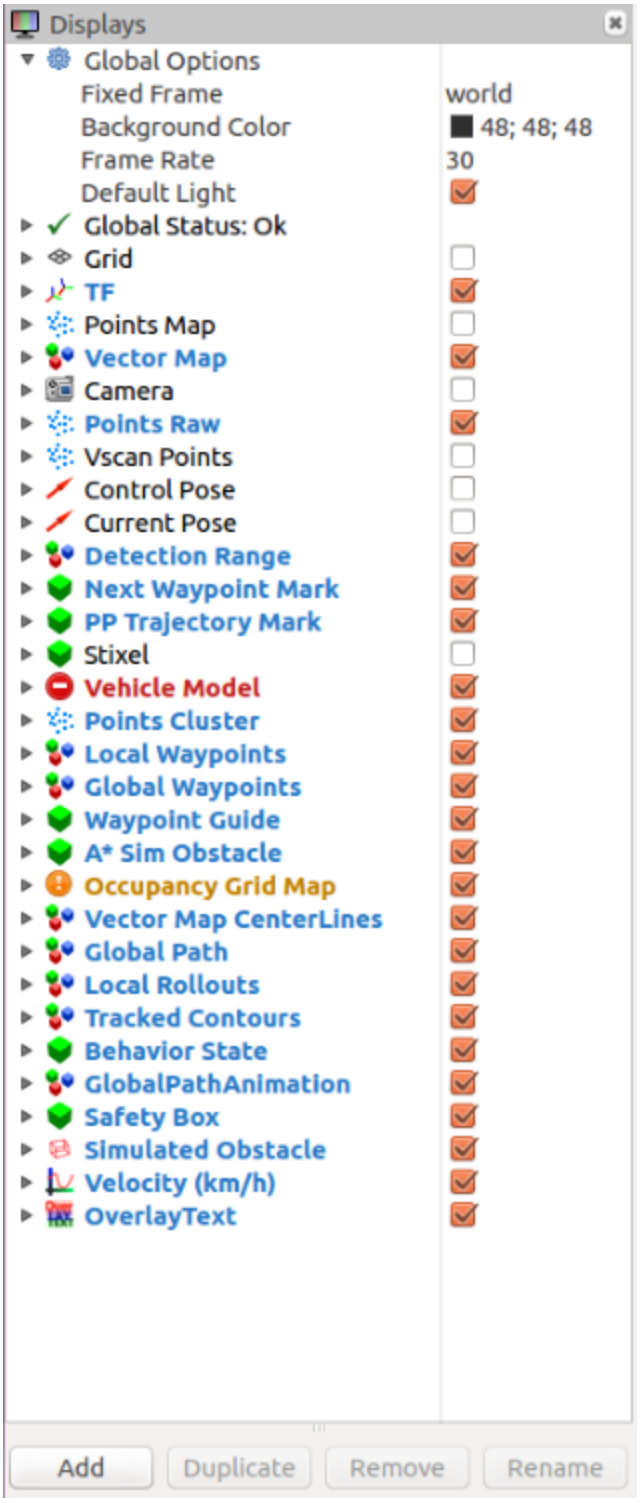
Πριν φορτώσουμε το αρχείο πατώντας το "Play" θα πρέπει να ορίσουμε και τον χρόνο προσομοίωσης. Πχ. 200 sec.



Εικόνα 58 : Εκκίνηση του Simulation στο Rviz

Αφού τα ορίσουμε τα παραπάνω, πατάμε το "Play" και ξεκινάει να τρέχει στο back ground το simulation. Το interface όπου θα χρησιμοποιήσουμε είναι το RViz. Αφού ξεκινήσει το RViz μπορούμε να ξεκινήσουμε να ορίζουμε διάφορα objects που θα μας χρειαστούν όπως: *coordinates, grids, robot, Point Map, Vector Map, Global Path* κτλ. ή μπορούμε άμεσα να χρησιμοποιήσουμε ένα έτοιμο configuration file όπου υπάρχουν στο repository του Autoware. Συγκεκριμένα : το αρχείο */Autoware/ros/srv/.config/rviz/default.rviz* περιέχει όλα τα objects που

χρειαζόμαστε για την προσομοίωση που θα κάνουμε. Οπότε φορτώνουμε κατευθείαν το default.rviz αρχείο.

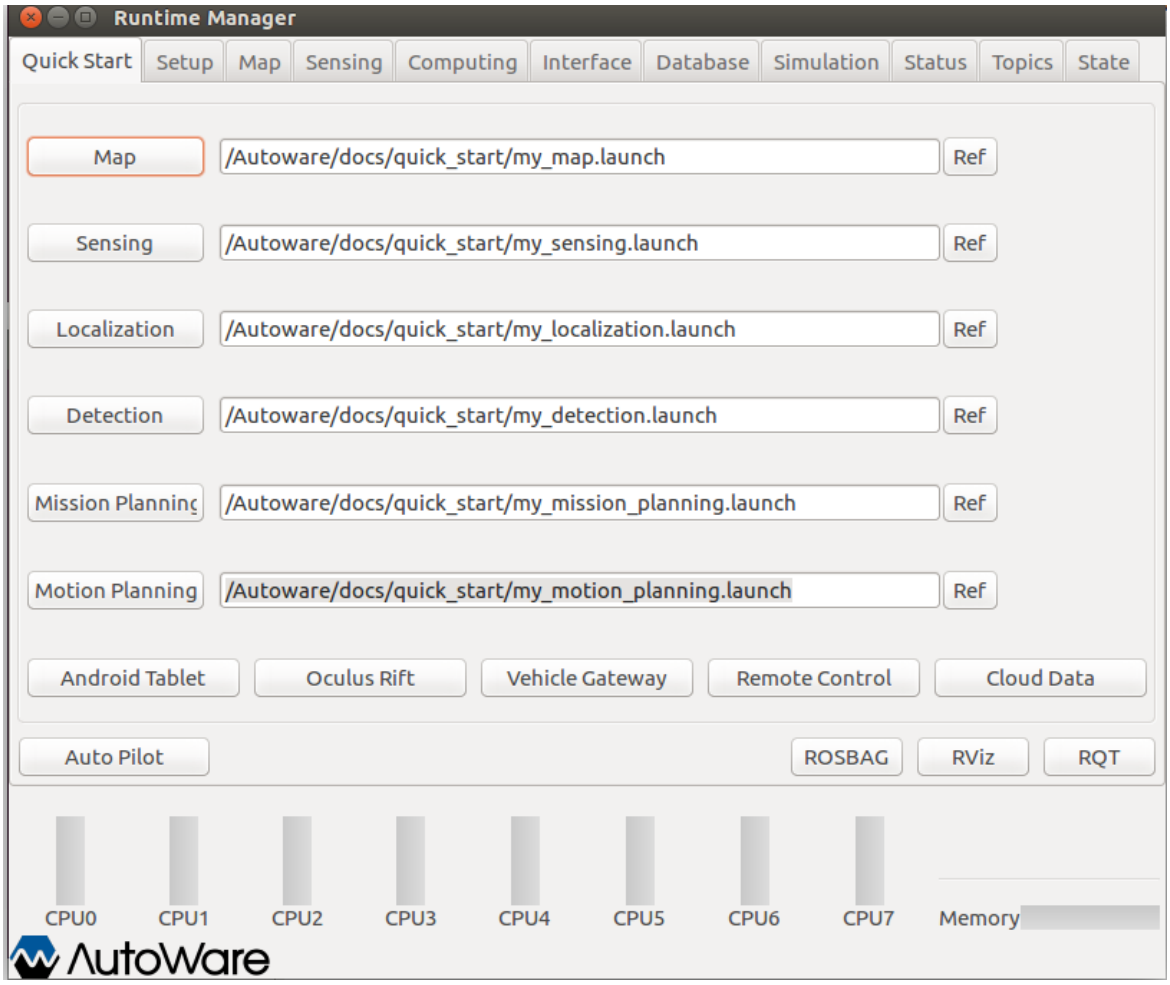


Εικόνα 59 : Objects του default.rviz αρχείου

Όπως βλέπουμε παραπάνω, φορτώνοντας το *default.rviz* αρχείο, έχουμε έτοιμα όλα τα παραπάνω objects, τα οποία θα γεμίσουν στην πορεία με τις αντίστοιχες οντότητες που θα στείλουμε.

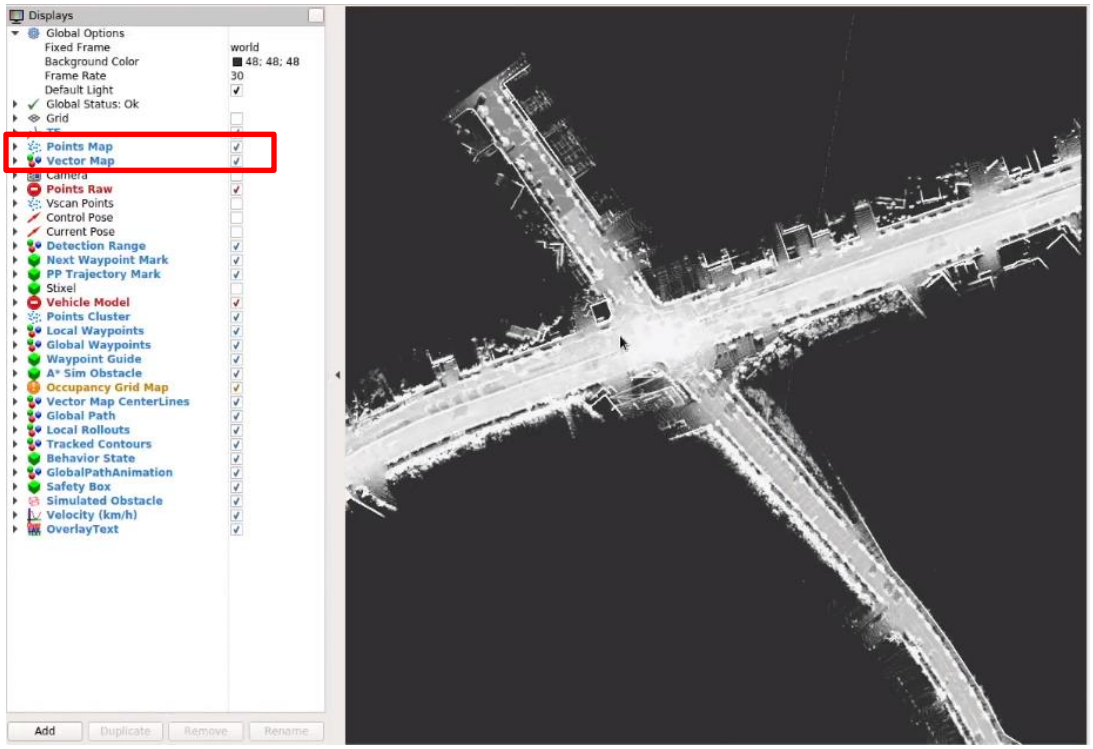
Όπως προαναφέρθηκε και στην θεωρία, ένα αυτόνομο όχημα, προκειμένου να είναι αυτόνομο θα πρέπει να διαχειρίζεται αρκετές πληροφορίες όπως : map, sensing, localization και detection. Επίσης, χρειαζόμαστε και πληροφορίες όπως mission Planning και motion Planning όπου βοηθάνε το όχημα να γνωρίζει τι διαδρομή πρέπει να ακολουθήσει και πως θα την ακολουθήσει. Όλα αυτά τα αρχεία δηλώνονται στο tab Quick start και φορτώνονται στα σταδιακά.

Σειρά έχει πλέον να πάμε στο tab 'Quick Start' και να δηλώσουμε ένα-ένα τα *.launch αρχεία με τη σειρά:

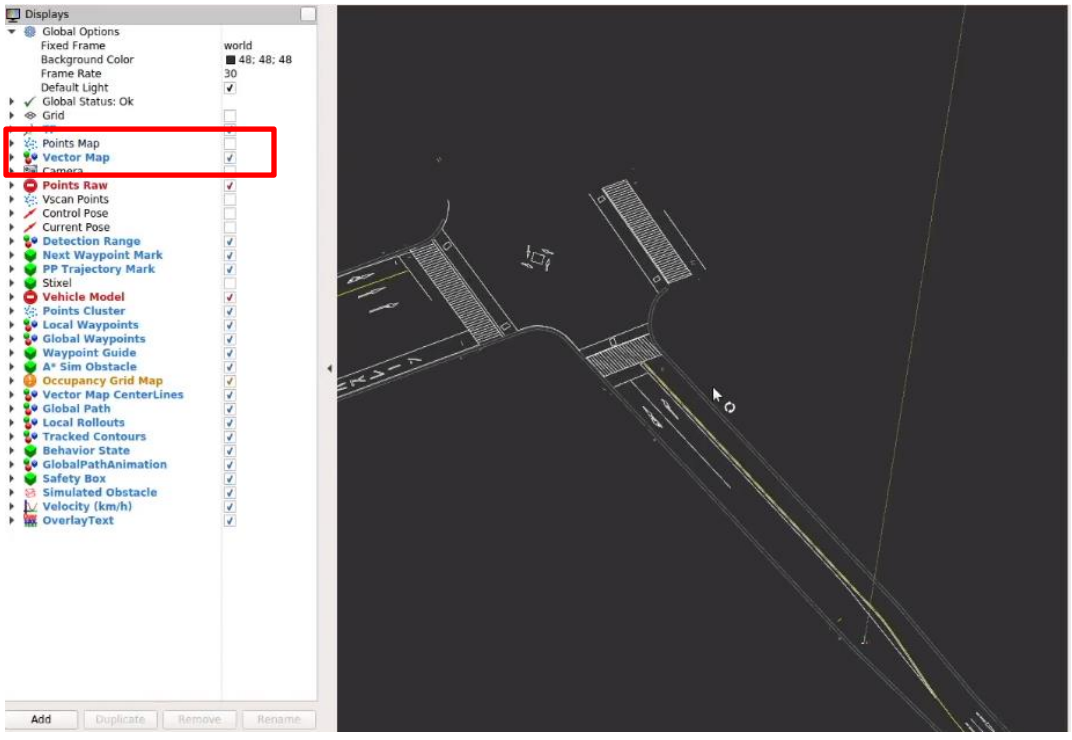


Εικόνα 60 : Runtime Manager – Φόρτωση των *.launch αρχείων

Φορτώνοντας το Map αρχείο θα δούμε στο RViz τον χάρτη όπου θα κινηθεί το όχημά μας. Ο χάρτης έχει διακριτοποιηθεί σε points και vectors.

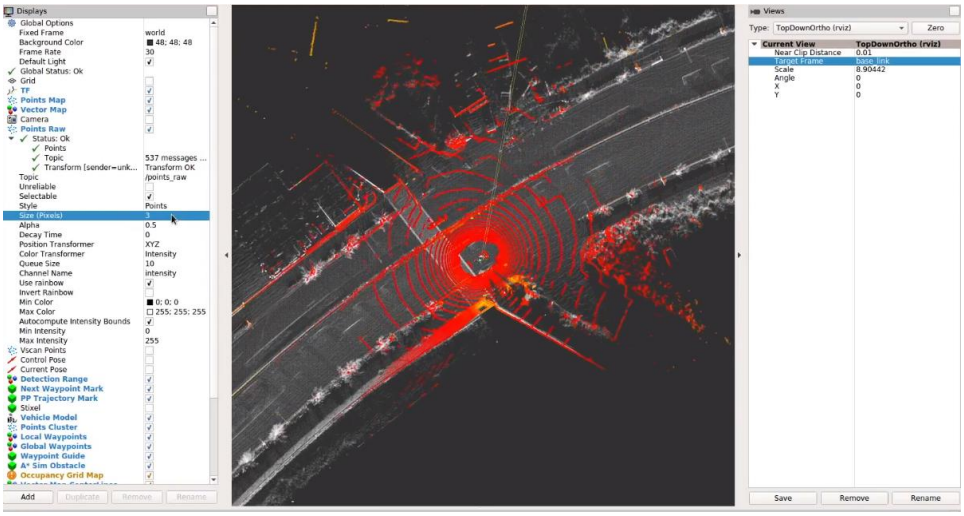


Εικόνα 61 : Simulation – Προβολή του χάρτη ως point



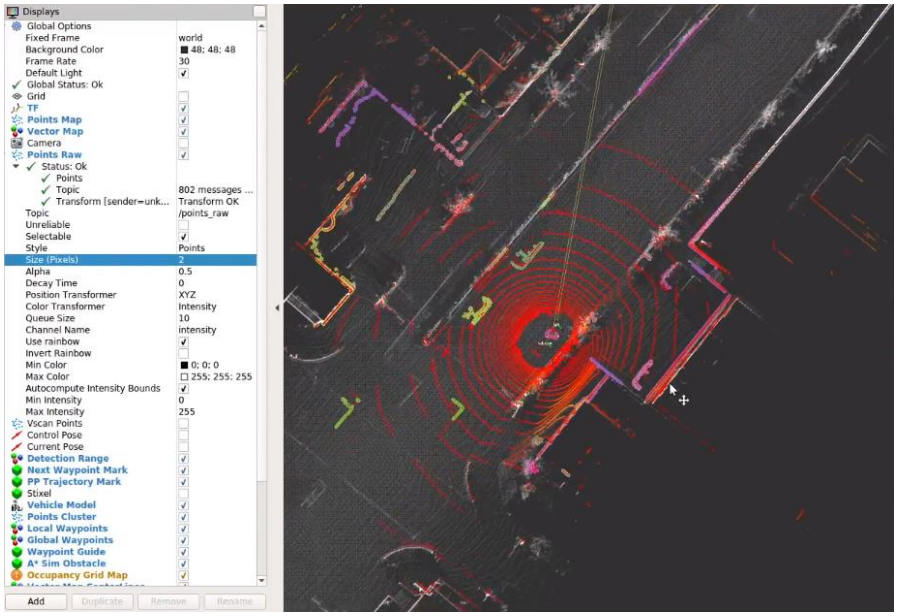
Εικόνα 62 : Simulation – Προβολή του χάρτη ως vector

Όπως βλέπουμε παραπάνω, μπορούμε να προβάλουμε τον χάρτη μας είτε με points είτε με vectors, είτε και με τα δύο. Αυτά τα entities τα ελέγχουμε μέσα από την Displa list. Στην πορεία φορτώνουμε το sensing και localization:



Εικόνα 63 : Simulation – Sensing και Localization

Φορτώνοντας και τα παραπάνω αρχεία πλέον έχουμε το όχημά μας φορτωμένο στο χώρο. Μέσω του sensing αρχείου το όχημά μας πλέον έχει την αίσθηση του χώρου. Πληροφορία που έχει παραχθεί με τη βοήθεια του laser scanner και την κάμερας. Το πάχος των γραμμών γύρω από το όχημα μπορεί να ελεγχθεί από την Display list μέσα στο entity *Points Raw> Size(pixels)*. Εφόσον πλέον έχουμε τον χάρτη, έχουμε το όχημα και το sensing φορτωμένο, περνάμε στο επόμενο στάδιο, όπου φορτώνουμε το object detection. Φορτώνουμε πλέον το αρχείο όπου εισάγει τα διάφορα objects στο χώρο μας (γειτονικά οχήματα και εμπόδια.)

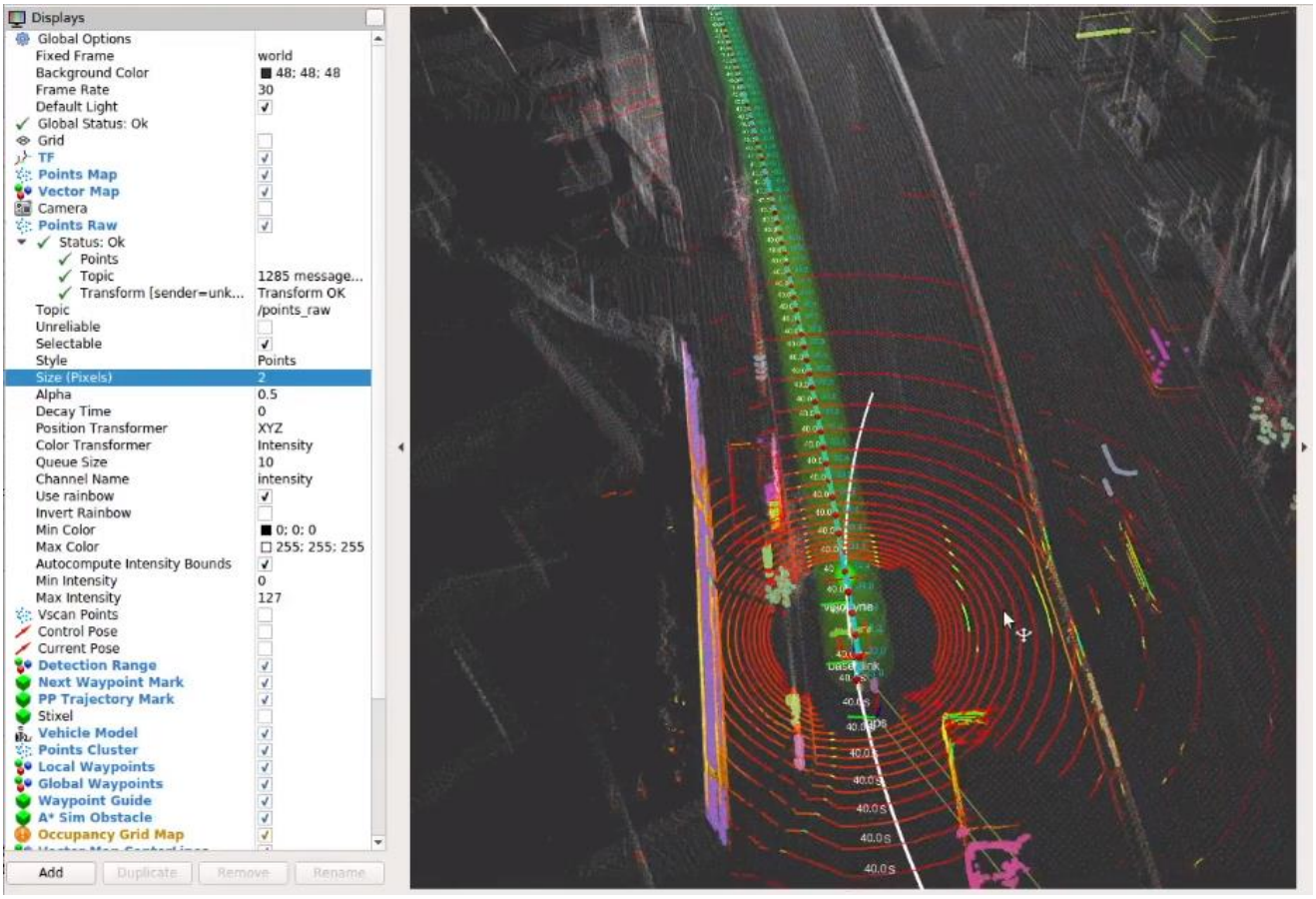


Εικόνα 64 : Simulation – Detection

Βλέποντας την *Εικόνα 66* έχουνε φορτωθεί τα διάφορα objects στον γύρω χώρο και προβάλλονται ως points στο χώρο με διαφορετικά χρώματα για να τα ξεχωρίζουμε (το κάθε object έχει το δικό του χρωματισμό). Παρακάτω θα δούμε πως μπορούμε να αλλάξουμε το σχήμα με το οποίο προβάλλονται τα objects γύρω από από το όχημά μας.

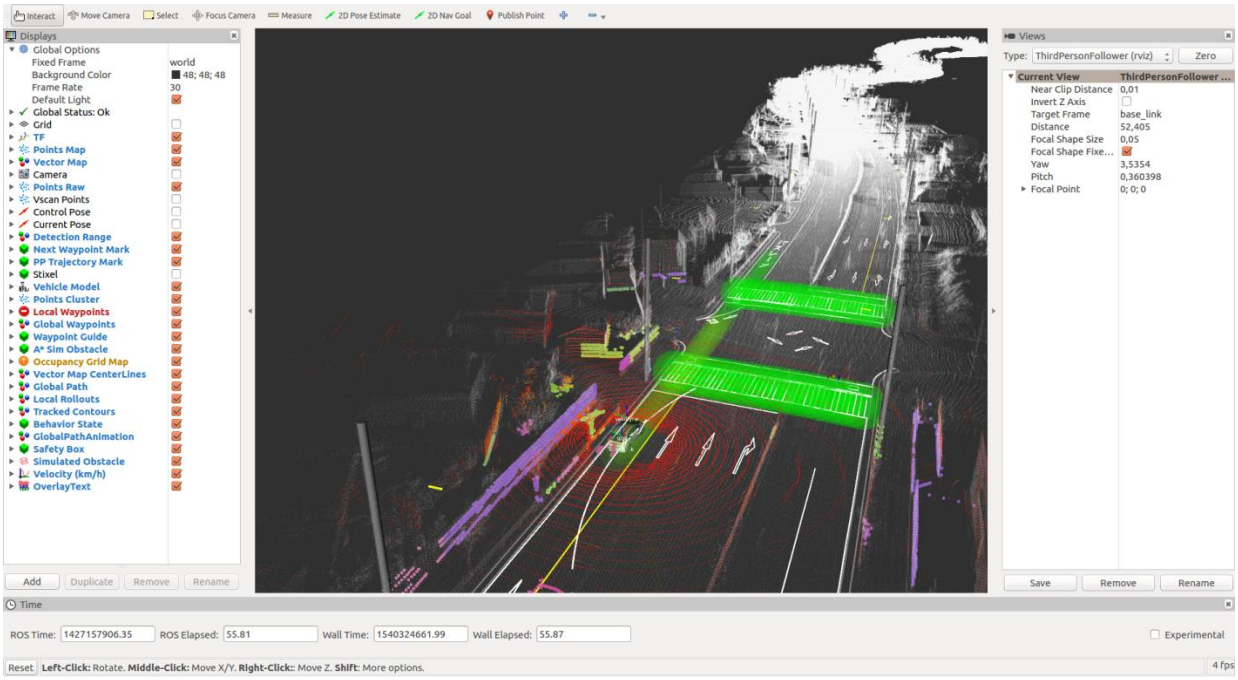
Φορτώνοντας πλέον και τα τελευταία δύο αρχεία, όπου είναι το mission και motion planning μπορούμε να δούμε πλέον:

- το path όπου θα ακολουθήσει το όχημά μας
- την ταχύτητα όπου χρειάζεται να κρατήσει
- την ακτίνα ελέγχου κατά την διαδρομή
- ποιες διαβάζεις έχουν οριστεί ελεύθερες για να τις περάσει και σε ποιες πρέπει να σταματήσει για έλεγχο

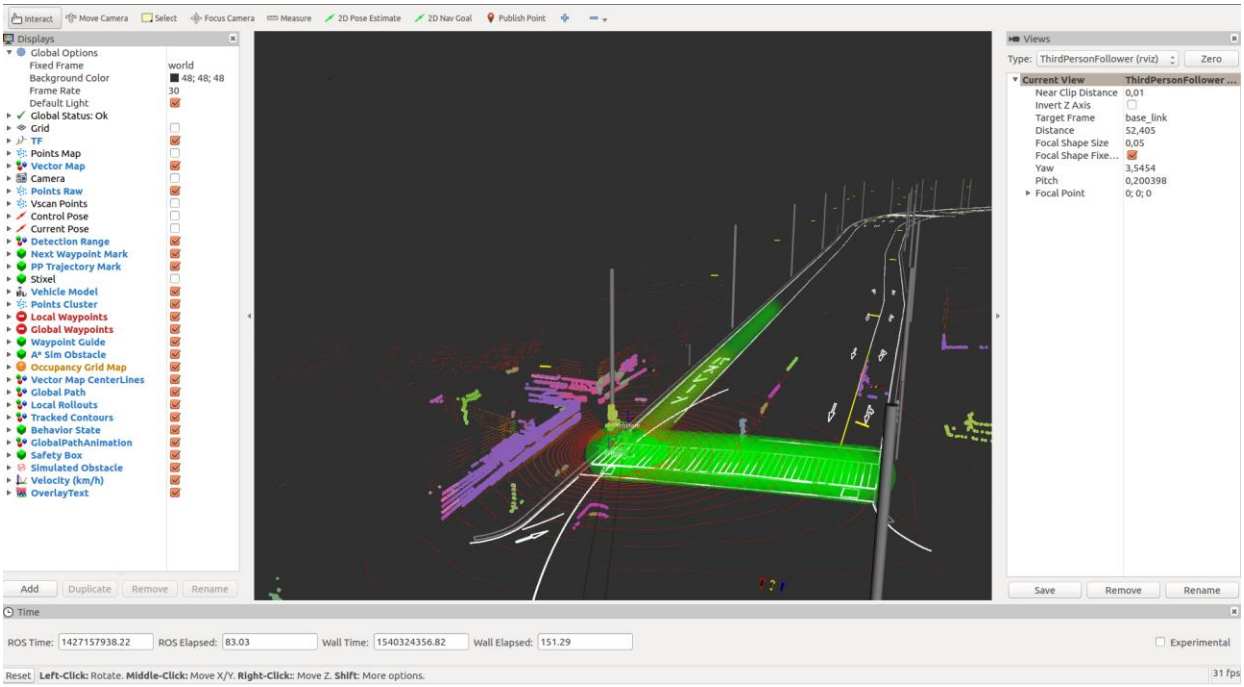


Εικόνα 65 : Simulation – Mission & Motion

Παρακάτω βλέπουμε το όχημα να πλησιάζει πεζοδιαβάσεις. Όπως βλέπουμε οι πεζοδιαβάσεις έχουν green illumination. Γεγονός που σημαίνει ότι ο χώρος έχει ελεγχθεί και το όχημα μπορεί να περάσει ελεύθερα.

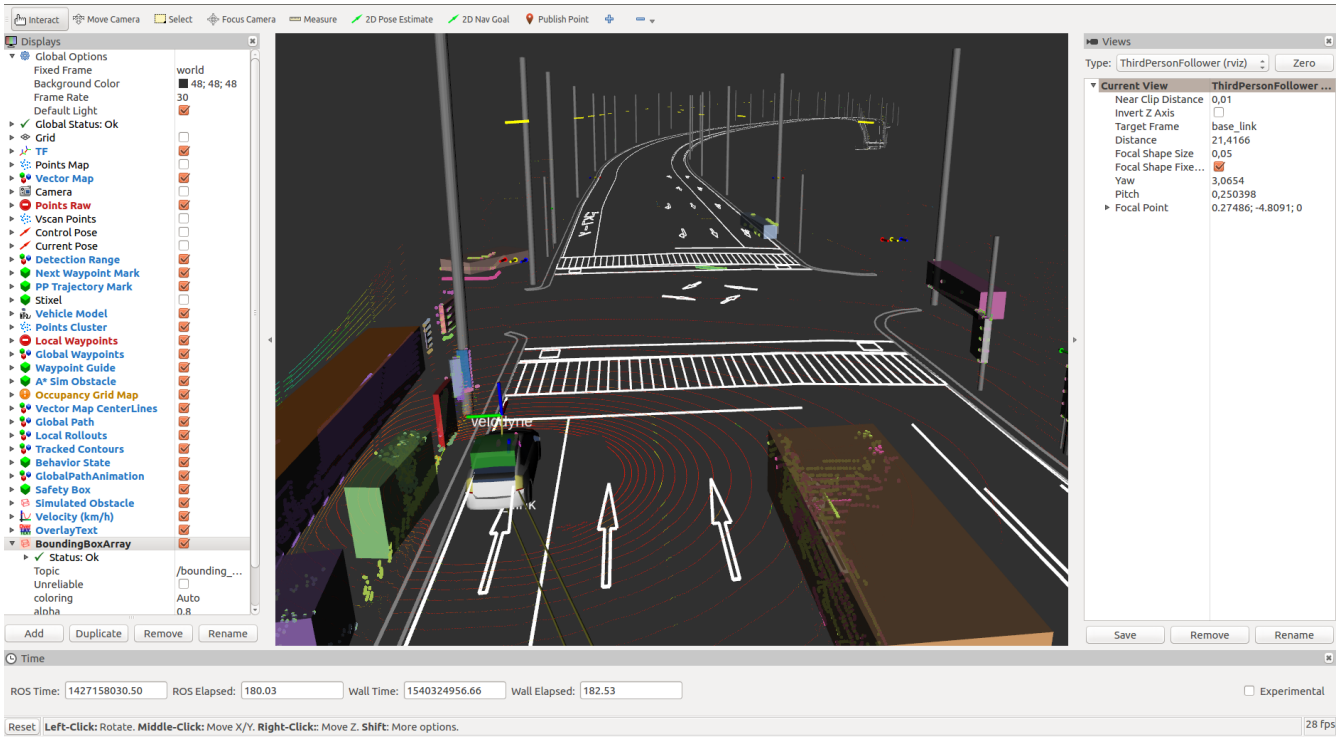


Εικόνα 66 : Simulation – Gren pedestrian cross (ο χώρος προβάλεται ως points και ως vectors)

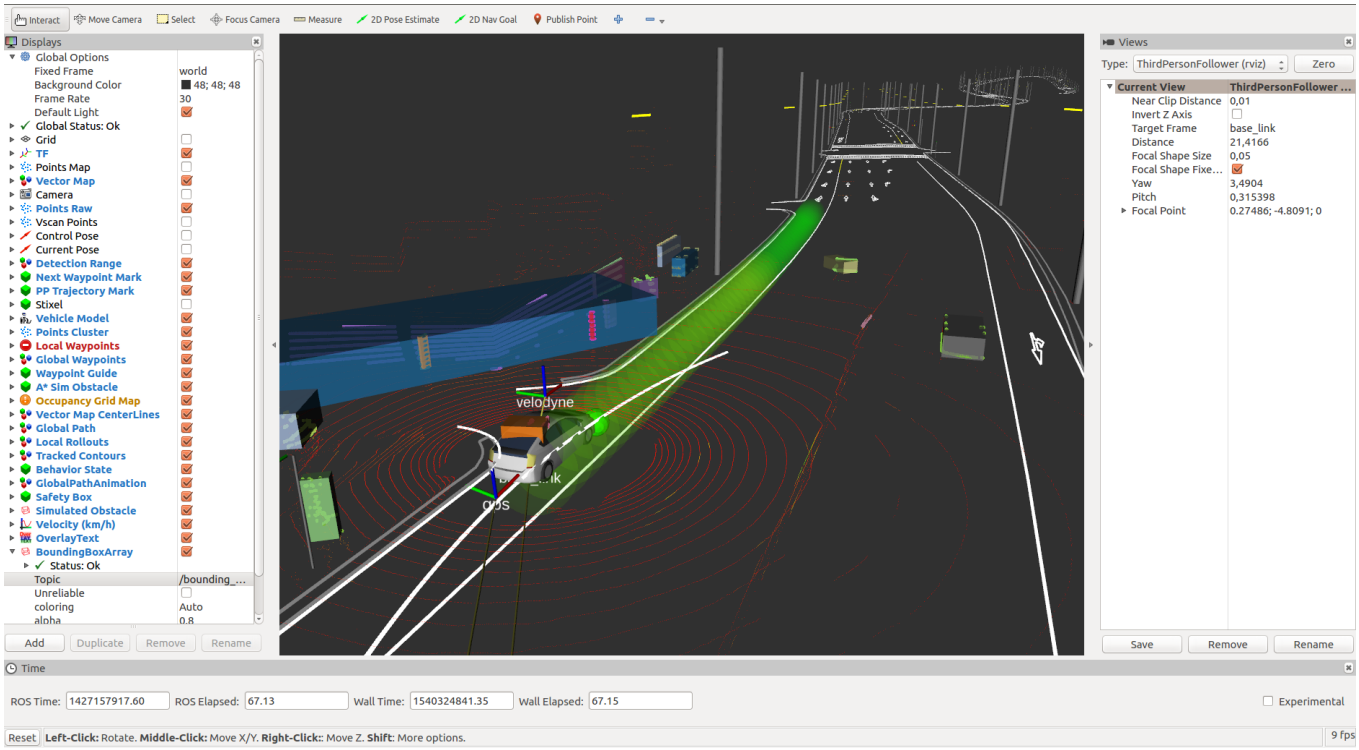


Εικόνα 67 : Simulation – Gren pedestrian cross (ο χώρος προβάλεται μόνο ως vectors)

Παρακάτω έχουμε ένα παράδειγμα όπου το όχημά μας συνάντησε red access σε πεζοδιάβαση. Εδώ καθώς τρέχουμε την προσομοίωση το όχημα σταματάει και περιμένει την έγκριση για να ξεκινήσει.



Εικόνα 68 : Simulation – Red access for Pedestrian Cross



Εικόνα 69 : Simulation – Obstacles Box view

Στην εικόνα 71 βλέπουμε πως μπορούμε να αλλάζουμε την εμφάνιση των εμποδίων από απλά σημεία σε κουτιά. Για να το κάνουμε αυτό φορτώνουμε ένα ακόμα entity στο Display list όπου είναι το *BoundaryBoxArray* και δηλώνουμε από την λίστα του το *topic>boundary_box*.

Όλα τα παραπάνω, πραγματοποιήθηκαν με την βοήθεια των αρχείων *.rosbag και *.launch.

Πέρα από το κομμάτι της προσομοίωσης το software μπορεί να χρησιμοποιηθεί και για real case πάνω σε όχημα, όπου κατά τη διάρκεια της οδήγησης θα καταγράφονται δεδομένα όπως: χάρτης, object detection, ταχύτητα διαδρομής κτλ. Στη συνέχεια τα αρχεία αυτά αποθηκεύονται και μπορούμε να πραγματοποιήσουμε την ίδια ακριβώς διαδρομή με το όχημά μας, αυτόνομα όμως. Επίσης, μπορούνε να χρησιμοποιηθούν και για ένα αντίστοιχο (όπως αναλύσαμε παραπάνω) simulation τα νέα αρχεία.

8.3 Field testing

Ένα από τα σημαντικότερα πλεονεκτήματα του Autoware είναι ότι ο οποιοσδήποτε μπορεί να ξεκινήσετε ένα field test με το δικό του όχημα αφού πρώτα τοποθετήσει ένα LiDAR στο όχημά του. Μπορεί να δημιουργήσει έναν τρισδιάστατο χάρτη, να κάνει μια διαδρομή και σε επόμενη φάση (αν διαθέτει τις κατάλληλες υποδομές) να την ακολουθήσει αυτόνομα, χρησιμοποιώντας το Autoware.

8.4 Δημιουργία Point Cloud Map

Το Autoware παρέχει μια τρισδιάστατη ενότητα χαρτογράφησης βασισμένη στον ταυτόχρονο εντοπισμό και χαρτογράφηση SLAM [16]. Καθώς οδηγείται το όχημα χτίζεται ένας τοπικός χάρτης συντεταγμένων (pointcloud). Για περισσότερες πληροφορίες σχετικά με τον SLAM αλγόριθμο :
http://ais.informatik.uni-freiburg.de/teaching/ss12/robotics/slides/12-slam.pdf

8.5 Διαδρομή με υπάρχων point cloud map

Μόλις ο point cloud map είναι έτοιμος, μπορεί ο χρήστης να δημιουργήσει μια διαδρομή την οποία θέλει να ακολουθήσετε αυτόνομα. Ο πιο απλός τρόπος για να γίνει αυτό είναι: οδηγούμε την πρώτη φορά τη διαδρομή και καταγράφονται οι θέσεις του οχήματος. Αυτό το ίχνος θέσεων του οχήματος μπορεί να αποθηκευτεί ως waypoints . Το μόνο που πρέπει να κάνει ο χρήστης είναι να ακολουθήσει αυτά τα σημεία. Ένας άλλος τρόπος, είναι να χρησιμοποιηθεί ο VectorMapper για να προστεθούν και χαρακτηριστικά του δρόμου στο pointcloud map. Έτσι, ακολουθούνται τα σημεία που αποτελούν την κεντρική γραμμή της λωρίδας της διαδρομής.

8.5 Waypoints μίας διαδρομής

Μόλις δημιουργηθεί μια διαδρομή με waypoints, το μόνο που χρειάζεται μετά είναι να ακολουθηθούν αυτά τα σημεία, ελέγχοντας την ταχύτητα και τη γωνία του οχήματος με βάση τα αποτελέσματα εντοπισμού και ανίχνευσης. Πηγαίνοντας ένα βήμα παραπέρα, για να κυκλοφορήσουμε σε δημόσιο δρόμο θα πρέπει να ενεργοποιήσουμε περισσότερες μονάδες όπως: object tracker, traffic light recognizer, και decision maker. Υποθέτουμε κιόλας ότι στην

πράξη, τα waypoints παράγονται από κάποια υπηρεσία πλοήγησης υψηλής ακρίβειας, τα οποία θα υποστηρίζονται από το Autoware.AI.

Public road testing: https://www.youtube.com/watch?v=npQMzH3j_d8

Private course testing: https://www.youtube.com/watch?v=zujGfJcZCpQ

Κεφάλαιο 9ο

Μέλλον στην αυτόνομη οδήγηση

9.1 Θεωρητική προσέγγιση

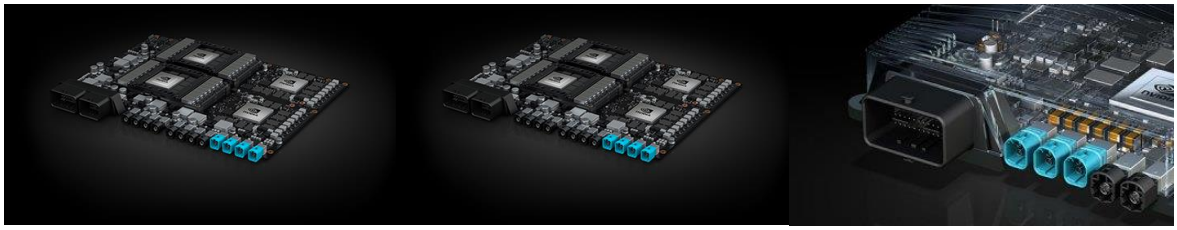
Προφανώς και η αυτόνομη οδήγηση των οχημάτων ανήκει στα μελλοντικά σχέδια διαφόρων αυτοκινητοβιομηχανιών. Τη σημερινή εποχή υπάρχουν ήδη αυτοκινητοβιομηχανίες όπως η Tesla και η Volvo, όπου προσφέρουν ήδη στο κοινό την αυτόνομη οδήγηση κάποιου οχήματος. Προφανώς και είναι περιορισμένος ο χώρος όπου λαμβάνουν ευθύνη οι αντίστοιχες βιομηχανίες καθώς οι χώροι όπου μπορεί να κινείται το όχημα σε αυτόνομη κατάσταση θα πρέπει να έχει και τις αντίστοιχες υποδομές του χώρου, όπως: σωστή διαγράμμιση και επαρκή σήμανση.

Αυτά όμως είναι παροντικά προβλήματα, αλλά και μελλοντικά. Δεν πρόκειται ποτέ να αντιμετωπιστούν βελτιώνοντας απλά τους αλγορίθμους αναγνώρισης. Προφανώς και θα πρέπει να βελτιωθούν σε κάθε χώρα, που θέλει να «ασπαστεί» την αυτόνομη οδήγηση, και η διαγράμμιση και η σήμανση.

Μία από τις μεγαλύτερες δυσκολίες που έχουμε να λύσουμε ακόμα και εμείς ως άνθρωποι κατά της οδήγηση είναι οι ηθικές αποφάσεις. Δεν αναφερόμαστε σε καταστάσεις όπως: «Πορτοκαλί φανάρι, περνάω δεν περνάω». Αναφερόμαστε σε καταστάσεις όπου η αποφυγή ατυχήματος θα είναι αδύνατο να αποφευχθεί και θα πρέπει να αποφασίσουμε αν θα τραυματίσουμε τους επιβάτες του οχήματος ή τους πεζούς που βρίσκονται μπροστά μας.

Προφανώς μία τέτοια κατάσταση δεν έχει ευτυχές τέλος, οπότε θα πρέπει να αρχίσουμε να αντιμετωπίζουμε το πρόβλημα πηγαίνοντας τουλάχιστον ένα βήμα πίσω, δηλαδή: ΝΑ ΜΗΝ ΒΡΕΘΕΙ ΤΟ ΟΧΗΜΑ ΣΕ ΤΕΤΟΙΑ ΚΑΤΑΣΤΑΣΗ. Βελτιώνοντας στο σύνολο όλα τα παραπάνω πεδία προφανώς και είναι εφικτό να λυθούν διάφορα καθημερινά προβλήματα.

Η τεχνητή νοημοσύνη είναι ένα αναπόσπαστο κομμάτι της αυτόνομης οδήγησης. Προκειμένου να βελτιωθούν οι αλγόριθμοί, θα πρέπει να έρθουν αντιμέτωποι με όλες τις πιθανές καταστάσεις. Η εικονική πραγματικότητα μπορεί να βοηθήσει σ αυτό δημιουργώντας διάφορα σενάρια και καθημερινές καταστάσεις των δρόμων και μέσα από αυτά τα συμβάντα να βελτιώνονται οι αλγόριθμοί και με μηδενικό κόστος υλικών. Εδώ μεγάλη πρόοδο έχει σημειώσει η NVIDIA με τις κάρτες NVIDIA DRIVE AGX PEGASUS & NVIDIA DRIVE AGX XAVIER όπου είναι έτοιμες πλακέτες που μπορούν να χρησιμοποιηθούν για να προσομοιώσουν ένα οδικό σενάριο.



Εικόνα 70 : NVIDIA DRIVE AGX PEGASUS & NVIDIA DRIVE AGX XAVIER

9.2 Πρακτική προσέγγιση

Προκειμένου να υλοποιήσουμε ένα όχημα με βάσης τις προδιαγραφές του Autoware θα χρειαστούν κάποια βασικά εξαρτήματα από πλευράς hardware:

Hardware	**Κόστος σε ευρώ**
Όχημα που συμπεριλαμβάνει σύστημα CAN (controller area network), π.χ. *TOYOTA PRIUS*	~25000
PointGrey (FLIR) Grasshopper 3 (USB/GigE)	200
VELODYNE HDL-64E (S1/S2/S3)	3500
Delphi ESR	3600
MicroStrain 3DM-GX5-15	1195
Trimble NetR9	19130
Laptop	3000
Σύνολο	55625

Πίνακας 9 : Κοστολόγιο υλοποίησης ενός αυτόνομου οχήματος με Autoware

Όπως βλέπουμε παραπάνω η υλοποίηση αυτή τη στιγμή ενός οχήματος έτοιμου να λειτουργήσει αυτόνομα με την βοήθεια του λειτουργικού Autoware φτάνει τα ~55625 ευρώ. Λαμβάνοντας υπ όψιν τον πρόχειρο αυτό υπολογισμό μπορούμε να πούμε ότι είναι μία λογική τιμή στα πλαίσια ενός αυτόνομου οχήματος καθώς αυτή τη στιγμή ένα Tesla Model 3 με το feature Autonomus driving φτάνει τα 60000 ευρώ. Και αναφερόμαστε σε ένα όχημα μαζικής παραγωγής.

Βιβλιογραφία

[1] *Autoware* – Quick Start . 2016

[2] ROS. Διαθέσιμο στην διεύθυνση : http://www.ros.org/

[3] STAIR, the STanford AI Robot. Διαθέσιμο στη διεύθυνση: http://ai.stanford.edu/~asaxena/stairmanipulation/

[4] Willow Garage. Διαθέσιμο στην διεύθυνση: http://www.willowgarage.com/

[5] Player. Διαθέσιμο στη διεύθυνση: http://wiki.ros.org/player

[6] YARP. Διαθέσιμο στη διεύθυνση: http://www.yarp.it/

[7] OROCOS. Διαθέσιμο στη διεύθυνση: http://www.orocos.org/

[8] CARMEN. Διαθέσιμο στη διεύθυνση: http://carmen.sourceforge.net/

[9] Orca. Διαθέσιμο στη διεύθυνση: http://orca-robotics.sourceforge.net/

[10] MOOS. Διαθέσιμο στη διεύθυνση: https://sites.google.com/site/moossoftware/

[11] Microsoft Robotics Studio. Διαθέσιμο στη διεύθυνση: https://en.wikipedia.org/wiki/Microsoft_Robotics_Developer_Studio

[12] Remote produce call. Διαθέσιμο στη διεύθυνση: https://en.wikipedia.org/wiki/Remote_procedure_call

[13] CAN bus. Διαθέσιμο στη διεύθυνση: https://en.wikipedia.org/wiki/CAN_bus

[14] Peter Biber. The Normal Distributions Transform: A New Approach to Laser Scan Matching. University of Tübingen

[15] Martin Magnusson. The Three_Dimensional Normal-Distribution Transform-an Efficient Representation for Registration, Surface Analysis and Loop Detection. ÖREBRO UNIVERSITY

[16] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester and Deva Ramanan. Object Detection with Discriminatively Trained Part Based Models.

[16] Wolfram Burgard, Cyrill Stachniss, Kai Arras, Maren Bennewitz. SLAM: Simultaneous Localization and Mapping.

[17] Steven M. LaValle. *Planning Algorithms*. University of Illinnois. Copyright of Steven M. LaValle 2006. Published by Cambridge University Press

[18] Dave Ferguson, Maxim Likhachev, and Anthony Stentz. *A Guide to Heuristic-based Path Planning*. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA

[19] Carol Fairchild, Dr. Thomas L. Harman. *ROS Robotics By Example*. PACKT Publishing. Open source community experience distilled

[20] Wolfram Burgard, Cyrill Stachniss, Maren Bennewitz, Kai Arras. Iterative Closest Point Algorithm.

[21] Vivienne Sze, Senior Member, IEEE, Yu-Hsin Chen, Student Member, IEEE, Tien-Ju Yang, StudentMember, IEEE, Joel Emer, Fellow, IEEE. Efficient Processing of Deep Neural Networks: A Tutorial and Survey

[22] VoxelNet. Cornell University. Διαθέσιμο στη διεύθυνση: https://arxiv.org/abs/1711.06396

[23] LMNet. Cornel University. Διαθέσιμο στη διεύθυνση: https://arxiv.org/abs/1805.04902

[24] Greg Welch, Gary Bishop. An Introduction to the Kalman Filter.Copyright 2001 by ACM, Inc.

[25] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy,Scott Reed, Cheng-Yang Fu1, Alexander C. Berg. SSD: Single Shot MultiBox Detector

[26] Xinxin Du, Marcelo H. Ang Jr., Sertac Karaman and Daniela Rus. A General Pipeline for 3D Detection of Vehicles

[27] Janko Petereit, Thomas Emter, Christian W. Frey. Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments

[28] Shuiying Wang. State Lattice-based Motion Planning for Autonomous On-Road Driving

[29] R. Craig Conlter. Implementation of the Pure Pursuit Path 'hcking Algorithm.

[30] YOLO. Διαθέσιμο στη διεύθυνση: https://pjreddie.com/darknet/yolo/

[31] Vincent Chen and Edward Chou. Practical Object Detection

and Segmentation

[32] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection

[33] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable Object Detection using Deep Neural Networks

[34] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks

[35] Joseph Redmon, Anelia Angelova. Real-Time Grasp Detection Using Convolutional Neural Networks

[36] MapTools. VectorMapper. Διαθέσιμο στη διεύθυνση : https://maptools.tier4.jp/vector_mapper_description/

[37] Steven M. LaValle. University of Illinois. PLANNING ALGORITHMS.

[38] Σ.Βολογιαννίδης. Σημειώσεις του μαθήματος Σχεδίαση και προσομοίωση των ρομποτικών συστημάτων.Τήμα Μηχανικών Πληροφορικής. Π.Μ.Σ. Ρομποτικής.

[39] *Autoware Hands-on Exercises*. version 1.1 .

[39] *Autoware Use's Manual*. Nagoya University. version 1.1. Autoware ver.2016.Sep.12

[41] Lentin Joseph. *ROS Robotics Projects*. April 4th 2017.

[42] Autoware. Διαθέσιμο στη διεύθυνση : https://autoware.ai/