*Αλεξάνδρειο Τεχνολογικό Εκπαιδευτικό Ίδρυμα Θεσσαλονίκης (Α.Τ.Ε.Ι.Θ.)*

*Σχολή Τεχνολογικών Εφαρμογών (Σ.Τ.ΕΦ.)*

**Τμήμα Μηχανικών Αυτοματισμού Τ.Ε.**


**Πτυχιακή Εργασία**

**Θέμα:**

# Χρήση του ROS σε εφαρμογές διαχείρισης αυτόνομων οχημάτων

**Μωυσιάδης Βασίλειος**

**Α.Μ. 123054**

**Επιβλέπων Καθηγητής:**

**Δημήτριος Μπεχτσής**
**Καθηγητής Εφαρμογών**

**Θεσσαλονίκη 2017**

**Περίληψη**

Οι εργασίες σε βιομηχανικό περιβάλλον αποθήκης με χρήση οχημάτων έχουν υψηλό βαθμό επικινδυνότητας και θέτουν σε κίνδυνο τόσο τους χειριστές των μηχανημάτων όσο και το προσωπικό. Επιπλέον κατά τη διάρκεια της μεταφοράς μπορεί να προκληθούν ζημιές με αποτέλεσμα τη μείωσης της παραγωγής και την απώλεια χρημάτων. Σκοπός της παρούσας έρευνας είναι να συμβάλλει στη αυτοματοποιημένη διαχείριση μιας βιομηχανικής αποθήκης (intra-logistics), παρέχοντας ένα καινοτόμο εργαλείο προγραμματισμού και ελέγχου σε πραγματικό χρόνο, για την πλοήγηση των έξυπνων αυτόνομων οχημάτων ή intelligent autonomous vehicles (IAV). Τα IAV είναι απόγονοι καθοδηγούμενων οχημάτων (AGVs). Το νέο εργαλείο παρέχει ανεξάρτητη χαρτογράφηση του χώρου σε πραγματικό χρόνο και εντοπίζει τη βέλτιστη διαδρομή που θα ακολουθήσει το IAV κατά την εκτέλεση των εργασιών του. Αυτό έχει ως αποτέλεσμα την αύξηση της ροής εργασιών, την μείωση της ενεργειακής κατανάλωσης και την αύξηση των επιπέδων ασφαλείας. Σύμφωνα λοιπόν με όλα τα παραπάνω τα IAVs είναι μια πολλά υποσχόμενη εναλλακτική λύση για την αποτελεσματική εκτέλεση των εργασιών σε επιχειρησιακό επίπεδο, δεδομένου ότι: (i) παρέχουν επιχειρησιακές πληροφορίες σε πραγματικό χρόνο σχετικά με τις παραμέτρους και τις συνθήκες της αποθήκης, (ii) μπορούν να λειτουργούν 24/7 με αξιοπιστία, (iii) καταναλώνουν λιγότερη ενέργεια σε σχέση με την συμβατική ανθρώπινη οδήγηση των μηχανημάτων, (iv) αποτελούν βασικό κομμάτι της τέταρτης βιομηχανικής φάσης (Industry 4.0) για τη δημιουργία έξυπνων βιομηχανικών χώρων (v) βελτιώνουν τα επίπεδα ασφαλείας χώρο εργασίας. Τα AGVs οδηγούν στη χρήση αυτοματοποιημένων συστημάτων και είναι σε θέση να διαχειριστούν καταστάσεις αβεβαιότητας λαμβάνοντας αποφάσεις κυρίως σε επιχειρησιακό επίπεδο με δυναμικό τρόπο και ως εκ τούτου επηρεάζουν και το επίπεδο του προγραμματισμού των εργασιών. Στην παρούσα εργασία παρουσιάστηκε αναλυτικά ο τρόπος εγκατάστασης του λογισμικού καθώς και ο οδηγός εκμάθησης του ROS στα Ελληνικά (Κεφάλαιο 1 και 2). Για τους σκοπούς της εργασίας δημιουργήθηκε σε περιβάλλον προσομοίωσης ένα IAV που ικανοποιούσε της ανάγκες του περιβάλλοντος προσομοίωσης(Κεφάλαιο 3). Στη συνέχεια εξηγείται το Navigation Stack, ο ρόλος του, η θεωρία, οι μέθοδοι και οι αλγόριθμοι για την χαρτογράφηση, τον εντοπισμό και την κίνηση του IAV στον εκάστοτε χώρο με την χρήση σχετικών συντεταγμένων(Κεφάλαιο 4). Τέλος αναπτύσσονται δύο αλγόριθμοι όπως και ένα μοντέλο προσομοίωσης μιας εργοστασιακής αποθήκης. Ο πρώτος αλγόριθμος αφορά την διαδικασία της χαρτογράφησης και τον τρόπο αυτοματοποίησης της ώστε να μην είναι πλέον απαραίτητη η παρέμβαση του ανθρώπινου παράγοντα. Ο δεύτερος αποτελεί μια σύνδεση στο Navigation Stack καθώς αρχικά κατευθύνει το IAV στην παραλαβή του ζητούμενου αποθέματος στην αποθήκη στη συνέχεια το παραδίδει στην έξοδο της αποθήκεη, τέλος κατευθύνει το IAV στο σημείο στάθμευσης για την αναμονή της επόμενης αποστολής(Κεφάλαιο 5).

## Abstract

Material handling activities in an industrial warehouse environment are considered of high risk and lead to major injuries, both for the people who operate the vehicles and the rest of the staff. In addition accidents in and industrial environments are slowing down the production line and lead to money loss. The purpose of this research is to contribute to the automated management of an industrial warehouse (intra-logistics) providing an innovative real-time programming and control tool for managing intelligent autonomous vehicles (IAVs) (autonomous ground vehicle descendant). The proposed tool provides a fully automated real time mapping of the industrial environment. As a result, work flow and security levels are increased while energy consumption decreases due to the selection of the optimized route. According to the above, IAVs are a promising solution for the efficient execution of any process that involves transportation at operational level. Our research concludes that IAVs: (I) Provide real time information about warehouse parameters and conditions, (ii) Reliably operate 24/7, (iii) consume less energy than a conventional human driving machine, (iv) are an essential part of the fourth industrial phase (industry 4.0), (v) improve security levels in the warehouse. AGVs can be used to automate industrial processes and they are capable of managing real time disruptions at operational level by taking dynamic decisions and therefore affect the production rate of the facility.

In this paper presented the installation, as well as the learning guild of ROS in Greek language (Chapter 1 and 2). We describe how to create in a simulation environment a model of a robot that meets the needs of an industrial warehouse(Chapter 3). Then we explain the theory and the role of Navigation Stack as well as the methods and the algorithms for mapping, localization, and movement of the robot to each of environments using relative coordinates(Chapter 4). In the last Chapter(5) we developed two algorithms and a model of an industrial warehouse. This first algorithm is about the procedure of SLAM and how it could be automated so it wont be necessary the innervation of human factor. The second algorithm is a simple action client to Navigation Stack. Fist drives the robot to the location of a specific location to pick up a box, then drives the robot at the exit of the warehouse to deliver the box, finally drives the robot to the parking spot to wait the next assignment.

**Ευχαριστίες**
Για την διεκπεραίωση της πτυχιακής εργασία ευχαριστώ θερμά

- Τον κύριο Δημήτριο Μπεχτσή, Καθηγητή Εφαρμογών του Τμήματος Μηχανικών Αυτοματισμού για την συνεχή καθοδήγηση του.
- Τον κύριο Διονύση Μπόχτη διευθυντή του Ινστιτούτου Έρευνας και Τεχνολογίας Θεσσαλίας (Ι.Ε.ΤΕ.Θ/Ε.Κ.Ε.Τ.Α) για τις πολύτιμες συμβουλές του.
- Την οικογένεια και τους φίλους μου για την συνεχή συμπαράσταση τους στη μέχρι τώρα σταδιοδρομία μου.

# Περιεχόμενα

## Κεφάλαιο 1 Ξεκινώντας με το ROS

### 1.1 Εισαγωγή

Το 1950 εμφανίζονται για πρώτη φορά στη βιομηχανία τα αυτοματοποιημένα καθοδηγούμενα οχημάτα (AGVs). Αρχικά τα AGV χρησιμοποιούνταν για την εκτέλεση απλών διαδρομών με την χρήση αισθητήρων αφής για την αναγνώριση ανεπιθύμητων αντικειμένων(Günter, 2015(1)). Στην σύγχρονη εποχή ανθίζει η τάση αυτοματοποίησης των εργασιών. Πλέον διανύουμε την περίοδο της τέταρτης βιομηχανικής φάσης (Industry 4.0) και τα AGVs συνεργάζονται είτε με τον άνθρωπο είτε μεταξύ τους για την διεκπεραίωση πολύπλοκων εργασιών. Πλέον τα AGVs χρησιμοποιούν πολύπλοκες μηχανικές κινήσεις με μεγάλη ακρίβεια. Με την χρήση της τεχνητής νοημοσύνης είναι σε θέση να αναγνωρίζουν πρόσωπα και αντικείμενα όπως και να λαμβάνουν αποφάσεις για την λύση ενός προβλήματος. Η προσθήκη των AGVs στο εργασιακό περιβάλλον συμβάλει στην μείωση των ανθρώπινων ατυχημάτων, στην μείωση του εργασιακού κόστους και στην αύξηση της παραγωγικότητας και της αξιοπιστίας.Στόχος της εργασίας είναι να συμβάλλει στη αυτοματοποιημένη διαχείριση μιας βιομηχανικής αποθήκης (intra-logistics), παρέχοντας ένα καινοτόμο εργαλείο προγραμματισμού και ελέγχου σε πραγματικό χρόνο για την πλοήγηση των αυτοματοποιημένων καθοδηγούμενων οχημάτων (AGVs).Για την υλοποίηση αυτού το εγχειρήματος χρησιμοποιείται το λογισμικού ανοιχτού κώδικα ROS (Robot Operating System).

### 1.2 Εγκατάσταση του ROS kinetic

Βασική προϋπόθεση για την εγκατάσταση και την λειτουργία του ROS είναι να έχουμε εγκαταστήσει το ανοιχτό λογισμικό linux με την τελευταία έκδοση Ubuntu.

#### 1.2.1 Προσθήκη βιβλιοθηκών στο αρχείο source.list

Το πρώτο βήμα για την εγκατάσταση του ROS είναι να έχουμε εγκαταστήσει το αρχείο source.list, ανοίγουμε ένα τερματικό (Ctrl + Alt+t ή δεξή κλικ στην επιφάνεια εργασίας και επιλογή του “νέο τερματικό’’) και εισάγουμε την παρακάτω εντολή καθώς και τον προσωπικό μας κωδικό:

**$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'**

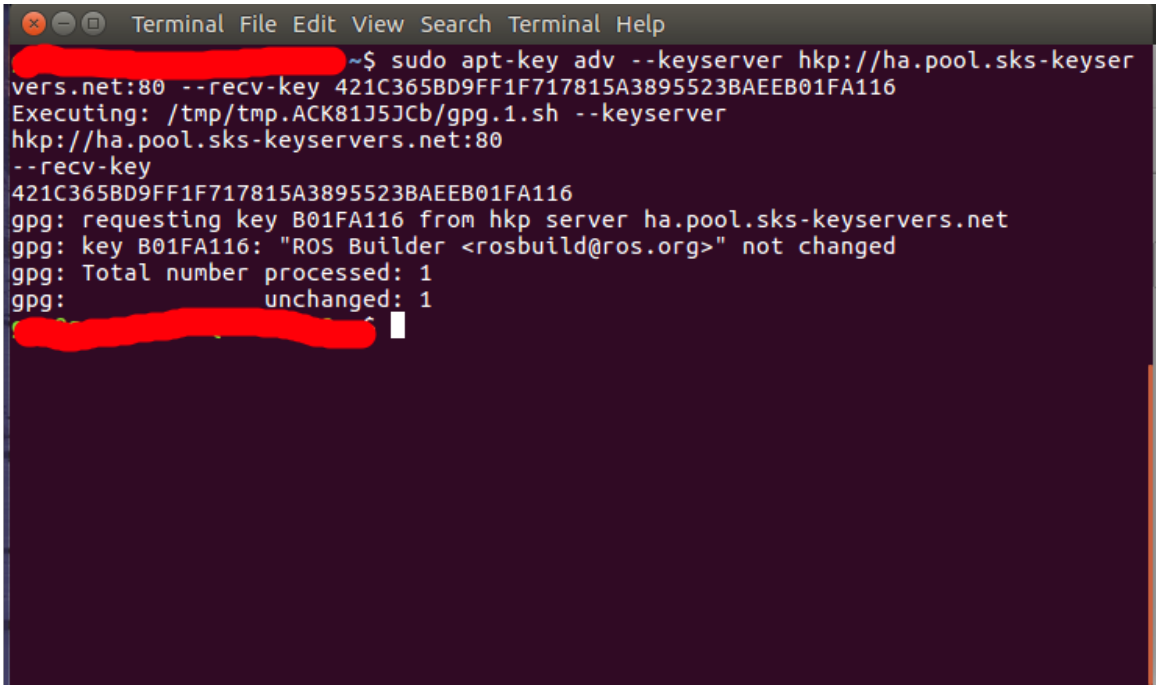
Με αυτόν τον τρόπο ο υπολογιστής μας θα δέχεται το λογισμικό από package.ros.org, εφόσον η διαδικασία ήταν επιτυχής στο τερματικό δε θα μας εμφανίσει κάποιο μήνυμα. Η παραπάνω εντολή είναι μόνο για τις εκδόσεις Wily (Ubuntu 15.10), Xenial (Ubuntu 16.04) και Jessie (Debian 8).

#### 1.2.2 Εγκατάσταση προσωπικού κλειδιού

Εγκαθιστώντας το το προσωπικό κλειδί εξασφαλίζουμε ότι ο πηγαίος κώδικας που λαμβάνουμε είναι σωστός και ότι δε θα αλλάξει κάποιος κώδικας ή πρόγραμμα χωρίς την έγκριση του ιδιοκτήτη. Στο τερματικό που έχουμε ήδη ανοιχτό πληκτρολογούμε την παρακάτω εντολή καθώς και τον προσωπικό μας κωδικό :

**$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116**

Μετά την εκτέλεση θα πρέπει να μας εμφανίσει ένα μήνυμα παρόμοιο με την εικόνα 1.1.1



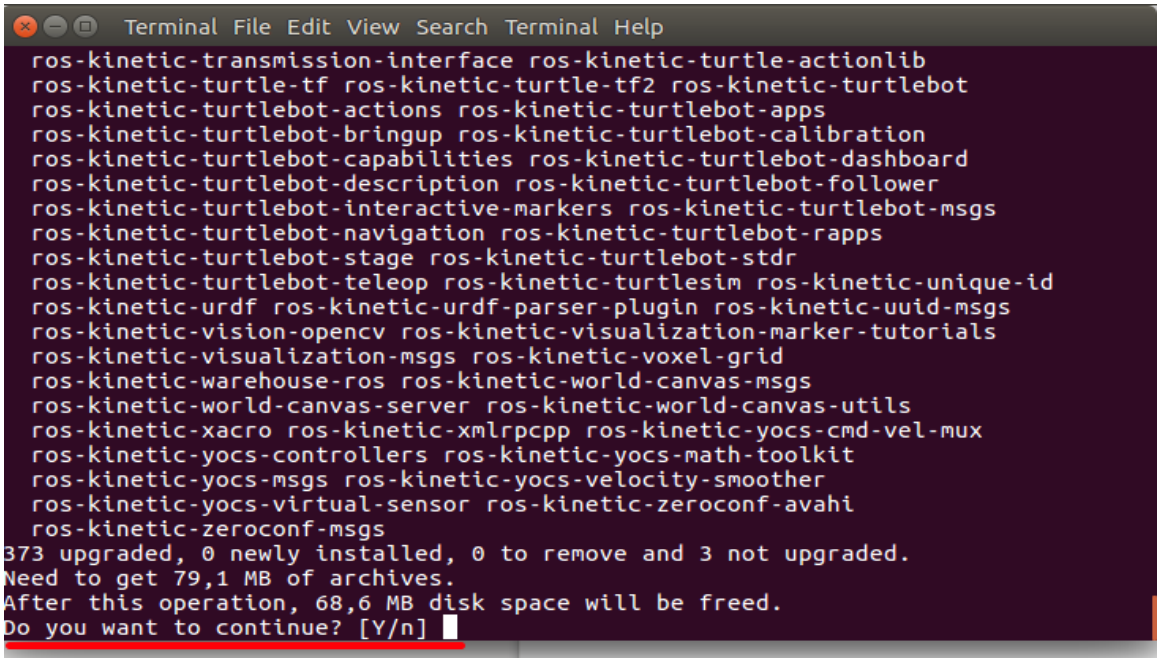

*Εικόνα 1.1.1 Δημιουργία κλειδιού*

### 1.2.3 Εγκατάσταση του ROS kinetic

Πριν την εγκατάσταση του ROS πρέπει να είμαστε σίγουροι ότι ο υπολογιστής διαθέτει τις τελευταίες ενημερώσεις του λειτουργικού μας συστήματος (Ubuntu 16.04) , ανοίγοντας ένα τερματικό πληκτρολογούμε:

**$ sudo apt-get update**
**$ sudo apt-get upgrate**

Εισάγουμε τον προσωπικό μας κωδικό, σε περίπτωση μου μας εμφανίσει το μήνυμα της εικόνας 1.1.2 σε οποιαδήποτε από τις παραπάνω εντολές πληκτρολογούμε “y” .




*Εικόνα 1.1.2 Εγκατάσταση του πιο πρόσφατου λογισμικού*

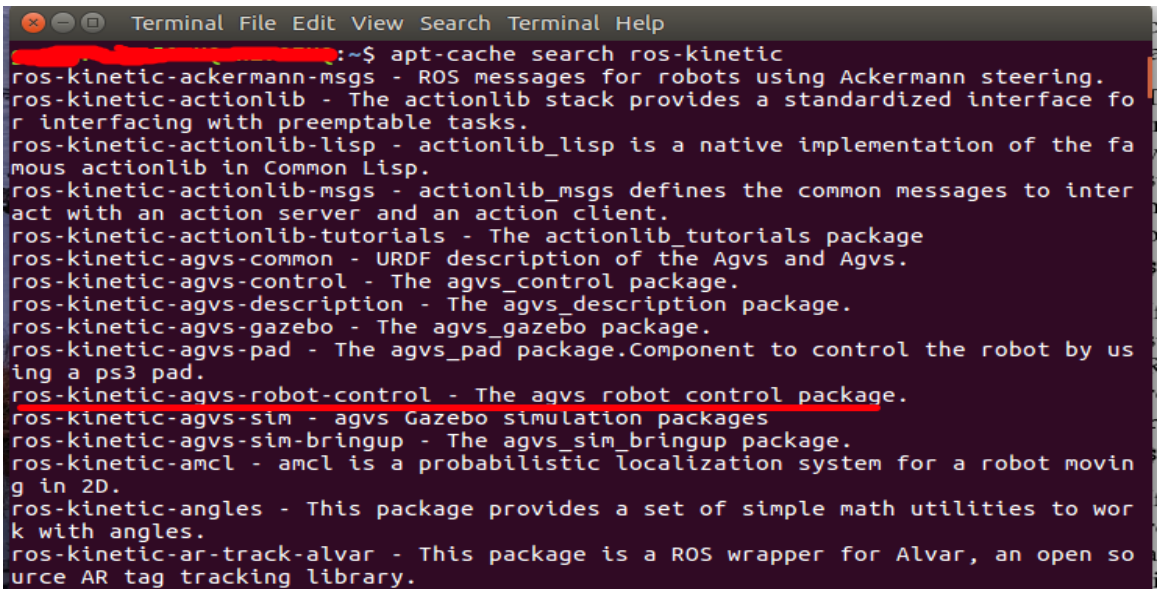
Για την εγκατάσταση του ROS υπάρχουν πολλές εκδόσεις και ο χρήστης πρέπει να επιλέξει εκείνη που καλύπτει της ανάγκες του Η τελευταία έκδοση που αποτελεί την προτεινόμενη και πιο ολοκληρωμένη επιλογή είναι το ROS Kinetic Desktop Full και για την εγκατάσταση του εκτελούμε στο τερματικό την παρακάτω εντολή:

**$ sudo apt-get install ros-kinetic-desktop-full**

Στο μήνυμα που θα μας εμφανίσει πληκτρολογούμε “y”. Με την εγκατάσταση του ROS Kinetic Desktop Full εγκαθίστανται κάποια βασικά πακέτα του ROS που είναι απαραίτητα για την λειτουργία του και την συνεργασία του με άλλα προγράμματα, όμως το ROS έχει μια αρκετά μεγάλη αποθήκη πακέτων που είναι άμεσα διαθέσιμα. Για να δούμε αυτήν την αποθήκη καθώς και ποια πακέτα της είναι διαθέσιμα για εγκατάσταση πληκτρολογούμε στο τερματικό:

**$ apt-cache search ros-kinetic**

Θα μας εμφανίσει το παράθυρο στην εικόνα 1.1.3




*Εικόνα 1.1.3 Διαθέσιμα πακέτα*

Όπως παρατηρούμε στην παραπάνω εικόνα αποτελείται από δύο στήλες, στην πρώτη είναι το όνομα του πακέτου και στην δεύτερη μια μικρή περιγραφή του, έτσι αν για παράδειγμα εμείς θέλουμε να εγκαταστήσουμε (δεν είναι απαραίτητη προς το παρόν αυτή η εγκατάσταση) το πακέτο “ros-kinetic-agvs-robot-control πληκτρολογούμε την παρακάτω εντολή:

**$ sudo apt-get install ros-kinetic-agvs-robot-control**

### 1.2.4 Αρχικοποίηση του rosdep

Για να είμαστε σε θέση να χρησιμοποιήσουμε το ROS πρέπει πρώτα να έχουμε αρχικοποιήσει το rosdep. Το rosdep είναι ακρωνύμιο από το ROS και το dependencies και μας επιτρέπει να εύκολη

εγκατάσταση και σύνταξη πηγαίου κώδικα που χρειάζονται για να τρέξουν βασικά στοιχεία του ROS, ανοίγουμε ένα τερματικό και εκτελούμε τις παρακάτω εντολές:

**$ sudo rosdep init**
**$ rosdep update**

### 1.2.5 Εγκατάσταση του περιβάλλοντος εργασίας

Αφού έχει γίνει επιτυχής εγκατάσταση του ROS και για να ξεκινήσει σωστά η χρήση του πρέπει να ορίσουμε μια διαδρομή μέσω του αρχείου bashrc για να ξέρει το λογισμικό που είναι εγκατεστημένο το ROS. Ανοίγουμε ένα τερματικό και πληκτρολογούμε τις παρακάτω εντολές :

**$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc**
**$ source ~/.bashrc**

Η διαδικασία αυτή πρέπει να επαναλαμβάνεται κάθε φορά που ανοίγουμε ένα καινούργιο τερματικό.

### 1.2.6 Εγκατάσταση του rosinstall

Το rosinstall είναι ένα πολύ σημαντικό πακέτο το οποίο δεν περιλαμβάνεται στην αρχική εγκατάσταση του ROS και μας βοηθάει στο κατέβασμα πολλαπλών πακέτων με την χρήση μόνο μίας εντολής:

**$ sudo apt-get install python-rosinstall**

## Κεφάλαιο 2 Αρχιτεκτονική του ROS

Η αρχιτεκτονική του ROS βασίζεται πάνω σε τρεις βασικά τμήματα:

- **Το τμήμα αρχείων ή filesystem,** είναι μια σειρά από έννοιες οι οποίες χρησιμοποιούνται για να εξηγήσουν τη δομή και τον τρόπο λειτουργίας του, από την διάταξη των φακέλων, μέχρι τον ελάχιστο αριθμό αρχείων που χρειάζεται το ROS για να λειτουργήσει.

- **Το τμήμα της επεξεργασίας,** μας δείχνει τους τρόπου με τους οποίους το ROS στέλνει, λαμβάνει και επεξεργάζεται δεδομένα ,πως ταξινομεί και δίνει προτεραιότητα στις διεργασίες του και πως μπορεί και επικοινωνεί με παραπάνω από έναν υπολογιστή.

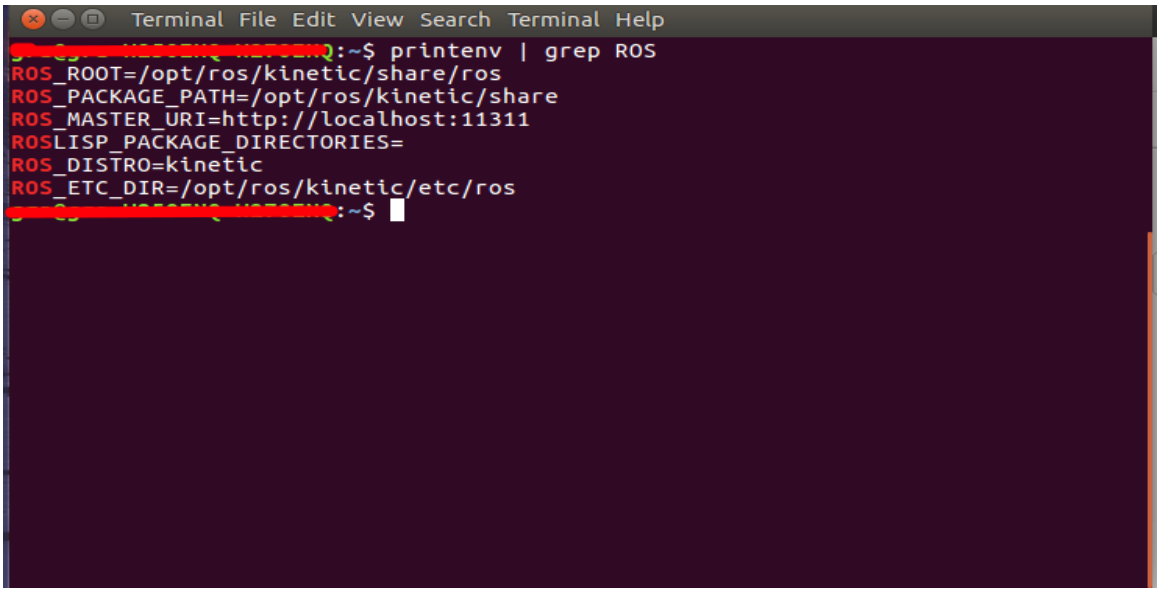
- **Το τμήμα της επικοινωνίας,** είναι το μέρος αυτό που εξηγεί ιδέες και έργα που σχετίζονται με το ROS. Επιπλέον υπάρχει η δυνατότητα να μοιραστείς τις ιδέες σου καθώς και του κοινόχρηστους αλγορίθμους με άλλους προγραμματιστές. Αυτό το τμήμα είναι πάρα πολύ σημαντικό για την ανάπτυξη και διάδοση του ROS.

### 2.1 Δημιουργία του περιβάλλοντος εργασίας

Πριν ξεκινήσουμε ξεκινήσουμε την διαδικασία δημιουργίας περιβάλλοντος εργασίας πρέπει να βεβαιωθούμε ότι έχει γίνει σωστά η εγκατάσταση του ROS , ανοίγουμε ένα νέο τερματικό και πληκτρολογούμε:

**$ printenv | grep ROS**

Εάν μας εμφανίσει την είκονα 1.2.1 τότε η διαδικασία εγκατάστασης ήταν επιτυχής ,διαφορετικά επαναλαμβάνουμε ξανά την διαδικασία




*Εικόνα 2. 1.1 ROS root paths*

Για την δημιουργία workspace , ανοίγουμε ένα τερματικό και πληκτρολογούμε:

**$ source /opt/ros/kinetic/setup.bash**
**$ mkdir -p ~/catkin_ws/src**

**$ cd ~/catkin_ws/src**
**$ catkin_init_workspace**

Με την εντολή catkin-init-workspace δημιουργούμε ένα CmakeLists.txt σε αυτό το αρχείο περιγράφουμε πως και σε ποια κομμάτια κώδικα του πακέτου θέλουμε να μεταγλωττίσουμε , όπως επίσης δηλώνουμε ποια αρχεία του πακέτου μας είναι εκτελέσιμα. Στην συνέχεια για να μεταγλωττίσουμε το προς το παρόν άδειο περιβάλλον εργασίας, στο ήδη ανοιχτό τερματικό εκτελούμε:

**$ cd ~/catkin_ws/**
**$ catkin_make**

Με την εντολή catkin_make ουσιαστικά μεταγλωττίσουμε το περιβάλλον εργασίας. Αν δούμε στο αρχείο που δημιουργήσαμε πλέον υπάρχουν δυο ακόμα φάκελοι , build και devel. Ο φάκελος build περιέχει μια λίστα με τα εκτελέσιμα πακέτα του περιβάλλοντος εργασίας και ο φάκελος devel περιέχει setup αρχεία κάνοντας source σε αυτό δείχνουμε στο ROS την νέα διαδρομή που πρέπει να ακολουθήσει έτσι ώστε να εκτελεί αρχεία μέσω του περιβάλλοντος εργασίας. Για να το πετύχουμε αυτό εκτελούμε:

**$ source devel/setup.bash**

Για να βεβαιωθούμε ότι η διαδικασία ήταν επιτυχής πληκτρολογούμε:

**$ echo $ROS_PACKAGE_PATH**

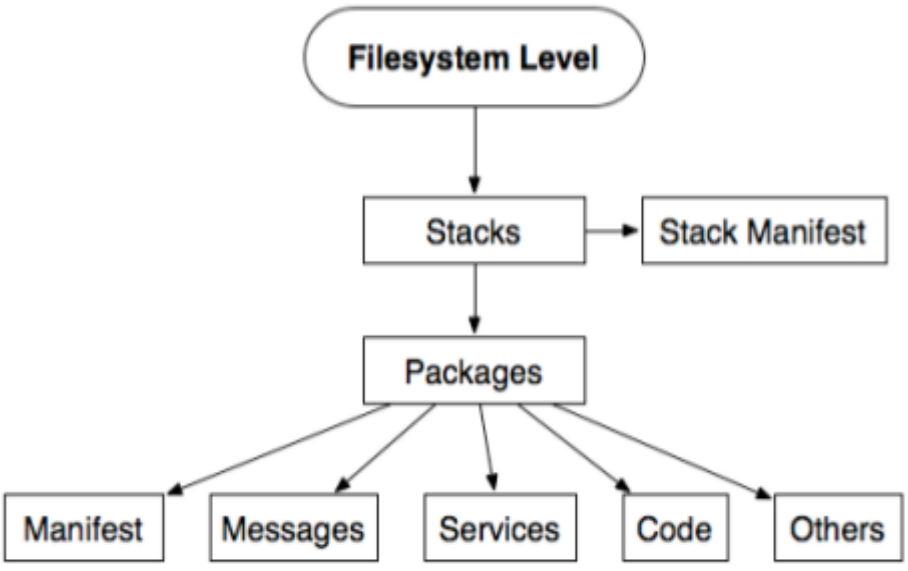
Αν μας εμφανίσει την ακόλουθη διαδρομή, η διαδικασία ήταν επιτυχής

**/home/όνομα χρήση /catkin_ws/src:/opt/ros/kinetic/share**

## 2.2 Κατανόηση και πλοήγηση στο σύστημα αρχείων του ROS

### 2.2.1 Κατανόηση του συστήματος αρχείων του ROS

Στην είκονα 2.2.1 φαίνεται η τυπική μορφή ενός filesystem που βρίσκεται εγκατεστημένο στο περιβάλλον εργασίας μας (workspace).




Εικόνα 2.2.1 Δομή του filesystem, πηγή Learning ROS for Robotics Programming

- Τα **πακέτα(packages)** περιέχουν τον ελάχιστη δομή και περιεχόμενο αρχείων που χρειάζεται το ROS για να λειτουργήσει, επίσης σε αυτά εμπεριέχονται βιβλιοθήκες, εκτελέσιμα αρχεία, srcipts, καθώς και κόμβοι στους οποίου γίνεται η επεξεργασία πληροφοριών.

- Το **μανιφέστο(manifest)** είναι η περιγραφή του πακέτου. Στο αρχείο αυτό αναγράφονται, η άδεια (license), οι εξαρτήσεις(dependencies) και πληροφορίες του πακέτου στο οποίο εμπεριέχεται. Στην έκδοση kinetic του ROS το αρχείο αυτό ονομάζεται package.xml.

- Η **στοίβα(stack)** δημιουργείται στην περίπτωση που έχουμε παραπάνω από ένα πακέτο που εκτελούν διαφορετικές λειτουργίες όμως εξυπηρετούν έναν κοινό σκοπό.

- Το **μανιφέστο στοίβας(stuck manifest)** περιέχει ακριβώς τις ίδιες πληροφορίες με το μανιφέστο, μόνο που στην συγκεκριμένη περίπτωση αφορούν ολόκληρη την στοίβα και όχι μόνο ένα πακέτο.

- **Τα μηνύματα(message)** είναι η πληροφορία την οποία το ROS λαμβάνει, επεξεργάζεται και στέλνεται ανάλογα με το είδος της διεργασίας που έχει να διεκπεραιώσει.

- Οι **υπηρεσίες(services)** είναι τα αρχεία με κατάληξη .src και προσδιορίζουν αίτηση(request) για διεργασία από το ROS και την απάντηση(response) αυτού σε σχέση με την διεργασία που του ζητήθηκε.

### 2.2.2 Πλοήγηση στο σύστημα αρχείων του ROS

Προτού ξεκινήσουμε θα πρέπει να εγκαταστήσουμε ένα πακέτο που δεν εμπεριέχεται στην αρχική εγκατάσταση, ανοίγουμε ένα τερματικό και πληκτρολογούμε:

**$ sudo apt-get install ros-kinetic-ros-tutorials**

Σε περιπτώσεις που έχουμε στοίβες πολλών πακέτων η εναλλαγή της διαδρομή στο τερματικό μας μέσω των εντολών ls και cd που μας παρέχει το Ubuntu μπορεί είτε να μας μπερδέψει είτε να μας καθυστερήσει. Για αυτόν τον λόγο το ROS μας παρέχει μια σουίτα εντολών που μπορούν να μας διευκολύνουν. Παρακάτω αναφέρονται μερικές από αυτές

- Η εντολή pospack μας επιτρέπει να λάβουμε πληροφορίες για το πακέτο το οποίο μας ενδιαφέρει, η σύνταξη της εντολής αυτής είναι “rospack (εντολή) (όνομα πακέτου)” ,για παράδειγμα πληκτρολογούμε στον τερματικό:

  **$ rospack find roscpp**

  και εφόσον η διαδικασία ήταν επιτυχής στον τερματικό θα πρέπει να μας εμφανίσει:

  **/opt/ros/kinetic/share/roscpp**

- Η εντολή roscd μας επιτρέπει να αλλάξουμε διαδρομή στο τερμετικό απευθείας στο πακέτο ή στην στοίβα που επιθυμούμε. Η σύνταξη της εντολής αυτής είναι “roscd (το όνομα του πακέτου)”, για παράδειγμα:

  **$ roscd roscpp**

  Εφόσον η διαδικασία ήταν επιτυχής στον τερματικό θα πρέπει να μας εμφανίσει την διαδρομή του πακέτου που επιθυμούμε.

- Η εντολή rosls μας επιτρέπει να βλέπουμε τα αρχεία που εμπεριέχονται σε κάποιον φάκελο. Η σύνταξη της εντολή αυτής είναι “rosls (το όνομα του φακέλου), για παράδειγμα:

  **$ rosls roscpp_tutorials**

και εφόσον η διαδικασία ήταν επιτυχής στον τερματικό θα πρέπει να μας εμφανίσει:

**cmake launch package.xml  srv**

- Τέλος το ROS μας δίνει την δυνατότητα της αυτόματης συμπλήρωσης εντολών με χρήση του πλήκτρου tab, για παράδειγμα αν πληκτρολογήσουμε την παρακάτω εντολή:

  **$ roscd roscpp_tut**( και το πλήκτρο tab)

  θα πρέπει να μας εμφανίσει:

  **$ rosls roscpp_tutorials**

## 2.3 Τα πακέτα του ROS

### 2.3.1 Δημιουργία ενός πακέτου

Για να δημιουργήσουμε ένα πακέτο αρχικά θα πρέπει να μεταβούμε μέσω ενός τερματικού στον φάκελο src στο χώρο εργασίας που έχουμε δημιουργήσει (φάκελος catkin_ws), για να γίνει αυτό ανοίγουμε ένα νέο τερματικό και πληκτρολογούμε:

**$ cd ~/catkin_ws/src**

στην συνέχεια η εντολή που θα μας δημιουργήσει το πακέτο είναι η catkin_create_pkg η οποία έχει την εξής σύνταξη “catkin_create_pkg (όνομα πακέτου) (τα αρχεία τα οποία εξαρτάται), οπότε πληκτρολογούμε:

**$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp**

Δηλαδή το πακέτο μας θα λέγεται beginner_tutorials το ποίο θα εξαρτάται (δηλαδή εμπεριέχει) αρχεία rospy (ros python) και roscpp (ros cpp).

### 2.3.2 Μεταγλώττιση ενός πακέτου (building a package)

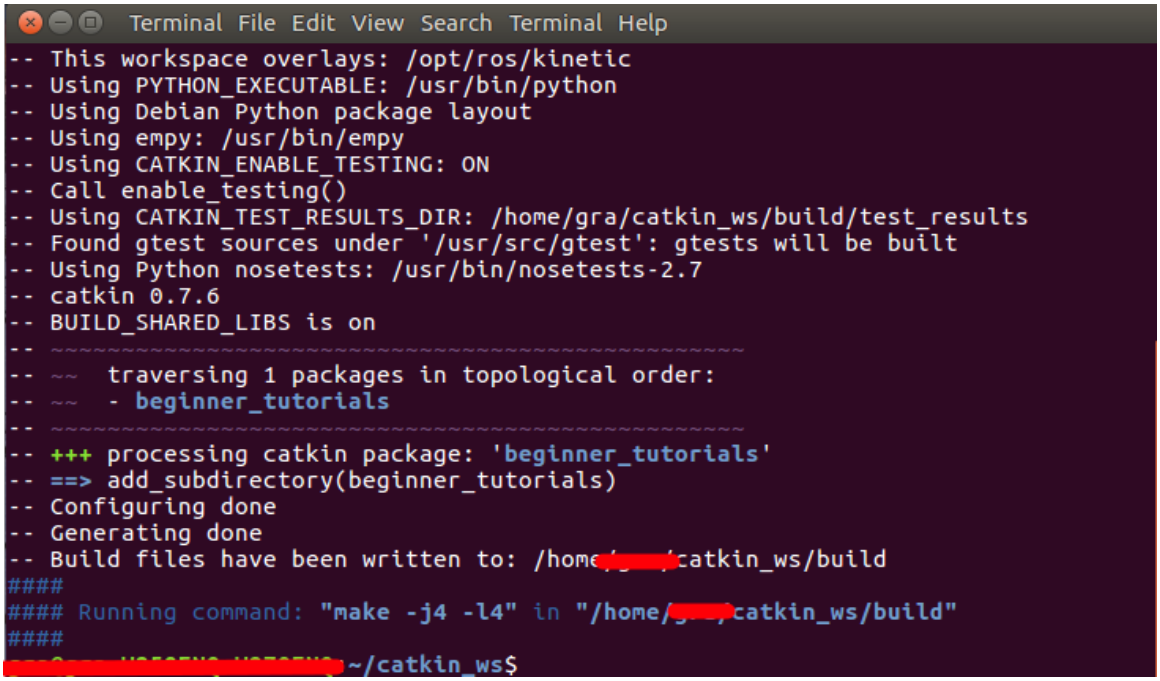
Τέλος για να  μεταγλωττίσουμε το περιβάλλον εργασίας μας αρκεί να μεταφερθούμε στη διαδρομή του, άρα στο ήδη ανοιχτό τερματικό πληκτρολογούμε:

**$ cd ..**

Στη συνέχεια πληκτρολογούμε:

**$ catkin_make**

Εφόσον η διαδικασία ήταν επιτυχής το τερματικό μας θα πρέπει να έχει περίπου μια μορφή σαν της εικόνας 2.3.1




*Εικόνα 2.3.1 Μεταγλώττιση στο περιβάλλον εργασίας*

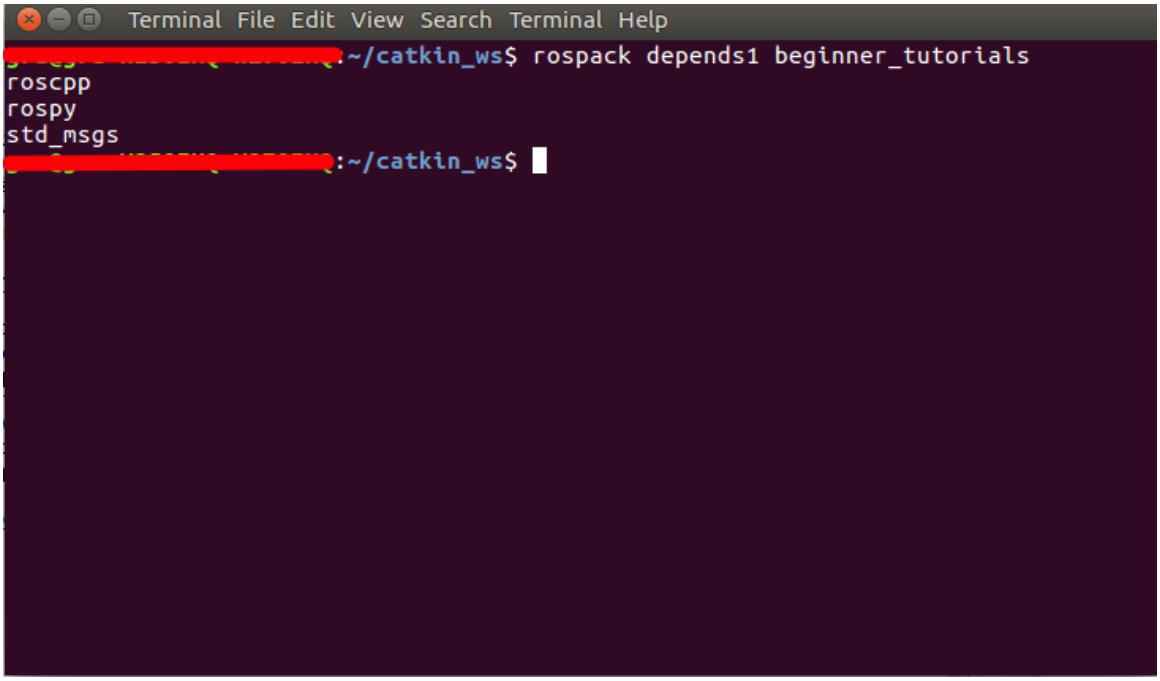
Αν στο πακέτο μας είχαμε είχαμε και εκτελέσιμα αρχεία, όπως για παράδειγμα .py ή .cpp ο μεταγλωττιστής ROS θα έβλεπε αν υπήρχαν τυχόν λάθη στον κώδικα, αλλά προς το παρόν το πακέτο μας είναι άδειο. Το τελευταίο βήμα για να ολοκληρώσουμε την δημιουργία του πακέτου μας είναι να προσθέσουμε το περιβάλλον εργασίας που έχουμε δημιουργήσει(συμπεριλαμβανομένου του πακέτου) στο περιβάλλον του ROS , για να γίνει αυτό αρκεί να κάνουμε source το αρχείο setup.bash που βρίσκεται στον φάκελο devel. Οπότε στον τερματικό που έχουμε ήδη ανοιχτό εκτελούμε την παρακάτω εντολή:

**$ source devel/setup.bash**

Όπως αναφέραμε στην προηγούμενη παράγραφό κάθε πακέτο βασίζεται και εξαρτάται από dependencies οι οποίες μπορούν να μας δώσουν πληροφορίες για το πακέτο μας, όπως για παράδειγμα τι γλώσσα προγραμματισμού την οποία χρησιμοποιεί (ή c++ ή python ή και τις δύο) αλλά και ποιος είναι ο ρόλος του, για παράδειγμα τι τύπο μηνυμάτων επεξεργάζεται. Για να είμαστε σε θέση να βρούμε αυτού του είδους τις πληροφορίες θα χρειαστούμε το rospack από την εργαλειοθήκη του ROS.Για να δούμε απο ποία dependencies αποτελείται το πακέτο που μόλις δημιουργήσαμε, δεν έχουμε παρά να πληκτρολογήσουμε σε ένα τερματικό την παρακάτω εντολή:

**$ rospack depends1 beginner_tutorials**

Μετά την εκτέλεση της εντολής το τερματικό μας θα πρέπει να έχει την μορφή της εικόνας 3.3.2




*Εικόνα 3.3.2 Beginner_tutorials dependencies*

Όπως παρατηρούμε τα dipendencies του πακέτου μας είναι τρία, τα δύο τα οποία τα είχαμε ορίσει εμείς (roscpp rospy) και το τρίτο (std_msgs) που το όρισε το ROS από μόνο του διότι ο τύπος std_msgs (standard messages) είναι βασικό κάθε πακέτου.

## 2.4 Μηνύματα και υπηρεσίες σου ROS

### 2.4.1 Μηνύματα του ROS

Τα μηνύματα του είναι απλά .txt αρχεία και εμπεριέχουν τους τύπους δεδομένων που μπορεί να επεξεργαστεί το ROS. Εξ αρχής το ROS έχει αρκετά προκαθορισμένα μηνύματα όμως μας δίνει επίσης την δυνατότητα να αναπτύξουμε και εμείς τον δικό μας τύπο μηνυμάτων.Η δομή ενός τέτοιο μηνύματος (είτε προκαθορισμένο από το ROS είτε δικό μας ) πρέπει να αποτελείται από δύο κομμάτια. Πρώτον τον τύπο του μηνύματος για παράδειγμα bool, int8, float32 ή έναν νέο τύπο που έχουμε δημιουργήσει εμείς, για παράδειγμα type1, type2 και δεύτερον από όνομα που θα ορίσουμε για αυτό το μήνυμα, για παράδειγμα:

- bool flag
- int8 counter
- float32 vel
- type1 mytype

Οι βασικοί τύποι που εμπεριέχει το ROS για τις δύο βασικές γλώσσες προγραμματισμού που χρησιμοποιεί, δηλαδή c++ και python  είναι οι εξής:

| Primitive type | Serialization | C++ | Python |
|---|---|---|---|
| bool | Unsigned 8-bit int | uint8_t | bool |
| int8 | Signed 8-bit int | int8_t | int |
| uint8 | Unsigned 8-bit int | uint8_t | int |
| int16 | Signed 16-bit int | int16_t | int |
| uint16 | Unsigned 16-bit int | uint16_t | int |
| int32 | Signed 32-bit int | int32_t | int |
| uint32 | Unsigned 32-bit int | uint32_t | int |
| int64 | Signed 64-bit int | int64_t | long |
| uint64 | Unsigned 64-bit int | uint64_t | long |
| float32 | 32-bit IEEE float | float | float |
| float64 | 64-bit IEEE float | double | float |
| string | ASCII string (4-bit) | std::string | string |
| time | Secs/nsecs signed 32-bit ints | ros::Time | rospy.Time |
| duration | Secs/nsecs signed 32-bit ints | ros::Duration | rospy.Duration |

Εικόνα 2.4.1 Τύποι μηνυμάτων, πηγή Learning ROS for Robotics Programming

Το ROS μαζί με την αποστολή του μηνύματος στέλνει και μια επικεφαλίδα ή header στην οποία τοποθετούμε χρόνο ή την πραγματική ώρα το όνομα του μηνύματος ποίο είναι ο αποστολέας του και ποίος είναι ο σειριακός αριθμός του μηνύματος, μια τυπική μορφή επικεφαλίδας είναι η εξής:

- unit32 seq
- time stamp
- sting frame_id

Το τμήμα της επικεφαλίδας στα μηνύματα του ROS είναι πολύ σημαντικό διότι περιλαμβάνει την μεταβλητή του χρόνου και γνωρίζοντας με τέτοια πληροφορία είμαστε σε θέση να γνωρίζουμε πληροφορίες για την εκάστοτε κατάσταση του IAV.

**Δημιουργία μηνυμάτων**

Για να δημιουργήσουμε ένα νέο μήνυμα στο πακέτο που δημιουργήσαμε στην προηγούμενη ενότητα ανοίγουμε ένα τερματικό και πληκτρολογούμε:

```
$ roscd beginner_tutorials
$ mkdir msg
$ echo "int64 num" > msg/Num.msg
```

Επίσης πρέπει να βεβαιωθούμε ότι το μήνυμα θα μετατραπεί σε πηγαίο κωδικά για c++ και python για να γίνει αυτό αρχικά ανοίγουμε το αρχείο package.xml που βρίσκεται μέσα στο φάκελο beginner_tutorials και προσθέτουμε στο τέλος του αρχείο της παρακάτω δύο γραμμές

```
<build_depend>message_generation</build_depend>
<run_depend>message_runtime</run_depend>
```

Στη συνέχεια για να μπορεί το ROS να αναγνωρίσει το μήνυμα που δημιουργήσαμε θα πρέπει να το προσθέσουμε στο αρχείο CmakeLists.txt. Το αρχείο αυτό βρίσκεται στον ίδιο φάκελο με το αρχείο package.xml. Αφού το ανοίξουμε τοποθετούμε το message_generation σαν dependency στο find_package , η τελική μορφή αυτού θα είναι η εξής:

```
find_package(catkin REQUIRED COMPONENTS

roscpp
rospy
std_msgs
 message_generation
)
```

Επίσης πρέπει να εξάγουμε τις εξαρτήσεις του πακέτου, βρίσκουμε το catkin_package() στο αρχείο CmakeLists.txt και το μορφοποιούμε έτσι ώστε η τελική του μορφή να είναι η παρακάτω:

```
catkin_package(
        #  INCLUDE_DIRS include
        #  LIBRARIES beginner_tutorials

                CATKIN_DEPENDS roscpp rospy std_msgs message_runtime

        #  DEPENDS system_lib
        )
```

Τέλος μορφοποιούμε το add_massage_files() kai generate_messages() ώστε η τελική τους μορφή να είναι η εξής:

```
## Generate messages in the 'msg' folder

add_message_files(
        FILES
         Num.msg
        )

## Generate added messages and services with any dependencies listed here

 generate_messages(
        DEPENDENCIES
        std_msgs
)
```

Αφού μεταβούμε στο φάκελο του περιβάλλοντος εργασίας μας μέσω ενός τερματικού, δηλαδή catkin_ws/ εκτελούμε την εντολή:

```
$ catkin_make
```

Εφόσον η μεταγλώττιση ήταν επιτυχής θα έχουμε δημιουργήσει το μήνυμα που επιθυμούμε, διαφορετικά βλέπουμε το λάθος που θα μας εμφανίσει στο τερματικό και το διορθώνουμε.

**Εύρεση πληροφοριών των μηνύματων του ROS**

Πολύ συχνά χρειαζόμαστε πληροφορίες σχετικά με τα μηνύματα που δεχόμαστε ή στέλνουμε για να μπορέσουμε να τα επεξεργαστούμε,Η σουίτα εργαλείων του ROS μας προφέρει ένα τέτοιο εργαλείο, την εντολή rosmsg**.**
Για να μάθουμε τον τύπο ενός μηνύματος ,για παράδειγμα του μηνύματος που δημιουργήσαμε παραπάνω, πληκτρολογούμε:

**$ rosmsg show beginner_tutorials/Num**

Η απάντηση που θα μας επιστρέψει το τερματικό θα είναι η εξής:

**int64 num**

Για να μάθουμε σε ποίο πακέτο έχουμε τοποθετήσει το μήνυμα πληκτρολογούμε:

**$ rosmsg show Num**

Η απάντηση που θα μας επιστρέψει το τερματικό θα είναι η εξής:

**[beginner_tutorials/Num]:**

**int64 num**

### 2.4.2 Υπηρεσίες του ROS

**Δημιουργία μας υπηρεσίας(service)**

Αρχικά πρέπει να δημιουργήσουμε τον φάκελο στον οποίο θα τοποθετήσουμε την υπηρεσία, ανοίγουμε ένα νέο τερματικό και μεταφερόμαστε στο πακέτο beginner_tutorials και δημιουργούμε τον φάκελο:

**$ roscd beginner_tutorials**
**$ mkdir srv**

Δε θα δημιουργήσουμε εμείς μια, θα αντιγράψουμε μια ήδη υπάρχουσα υπηρεσία του ROS με την εντολή roscp η σύνταξη της οποίας είναι, " roscp (το όνομα του πακέτου απο το οποίο θα την αντιγράψουμε) (το όνομα του πακέτο στο οποίο επιθυμούμε να το αντιγράψουμε) (τη διαδρομή της υπηρεσίας)" Στο ήδη ανοιχτό τερματικό πληκτρολογούμε:

**Εύρεση πληροφοριών των μηνύματων του ROS**

Πολύ συχνά χρειαζόμαστε πληροφορίες σχετικά με τα μηνύματα που δεχόμαστε ή στέλνουμε για να μπορέσουμε να τα επεξεργαστούμε,Η σουίτα εργαλείων του ROS μας προφέρει ένα τέτοιο εργαλείο, την εντολή rosmsg**.**
Για να μάθουμε τον τύπο ενός μηνύματος ,για παράδειγμα του μηνύματος που δημιουργήσαμε παραπάνω, πληκτρολογούμε:

**$ rosmsg show beginner_tutorials/Num**

Η απάντηση που θα μας επιστρέψει το τερματικό θα είναι η εξής:

**int64 num**

Για να μάθουμε σε ποίο πακέτο έχουμε τοποθετήσει το μήνυμα πληκτρολογούμε:

**$ rosmsg show Num**

Η απάντηση που θα μας επιστρέψει το τερματικό θα είναι η εξής:

**[beginner_tutorials/Num]:**

**int64 num**

### 2.4.2 Υπηρεσίες του ROS

**Δημιουργία μας υπηρεσίας(service)**

Αρχικά πρέπει να δημιουργήσουμε τον φάκελο στον οποίο θα τοποθετήσουμε την υπηρεσία, ανοίγουμε ένα νέο τερματικό και μεταφερόμαστε στο πακέτο beginner_tutorials και δημιουργούμε τον φάκελο:

**$ roscd beginner_tutorials**
**$ mkdir srv**

Δε θα δημιουργήσουμε εμείς μια, θα αντιγράψουμε μια ήδη υπάρχουσα υπηρεσία του ROS με την εντολή roscp η σύνταξη της οποίας είναι:

**$ roscp rospy_tutorials AddTwoInts.srv srv/AddTwoInts.srv**

Για να μπορέσουμε να χρησιμοποιήσουμε την υπηρεσία που μόλις αντιγράψαμε πρέπει να επαναλάβουμε μια παρόμοια διαδικασία με αυτήν της δημιουργίας μηνυμάτων. Ανοίγουμε το αρχείο CmakeLists.txt
και αντικαθιστούμε το add_service_files() με το παρακάτω:

```
add_service_files(
  FILES
  AddTwoInts.srv
  )
```

Στη συνέχεια πρέπει να μεταβούμε στον φάκελο catkin_ws μέσω του τερματικού μας και να μεταγλωττίσουμε τον χώρο εργασίας μας με την εντολή catkin_make.
Τέλος αφού η μεταγλώττιση ήταν επιτυχής θα χρησιμοποιήσουμε την εντολή rossrv με την οποία μπορούμε να πάρουμε πληροφορίες για την εκάστοτε υπηρεσία. Η σύνταξη αυτής είναι, “rossrv (εντολή) (το όνομα του πακέτο/το όνομα της υπηρεσίας). Παραδείγματος χάριν:

**$ rossrv show beginner_tutorials/AddTwoInts**

Στο τερματικό θα πρέπει να μας εμφανίσει

**int64 a**

**int64 b**

**---**

**int64 sum**

Ωστόσο στην περίπτωση που δε γνωρίζουμε το πακέτο αλλά μόνο το όνομα της υπηρεσίας μπορούμε να πληκτρολογήσουμε:

**$ rossrv show AddTwoInts**
Θα πάρουμε ως απάντηση:

**[beginner_tutorials/AddTwoInts]:**

**int64 a**

**int64 b**

**---**

**int64 sum**

**[rospy_tutorials/AddTwoInts]:**

**int64 a**

**int64 b**

**---**

**int64 sum**

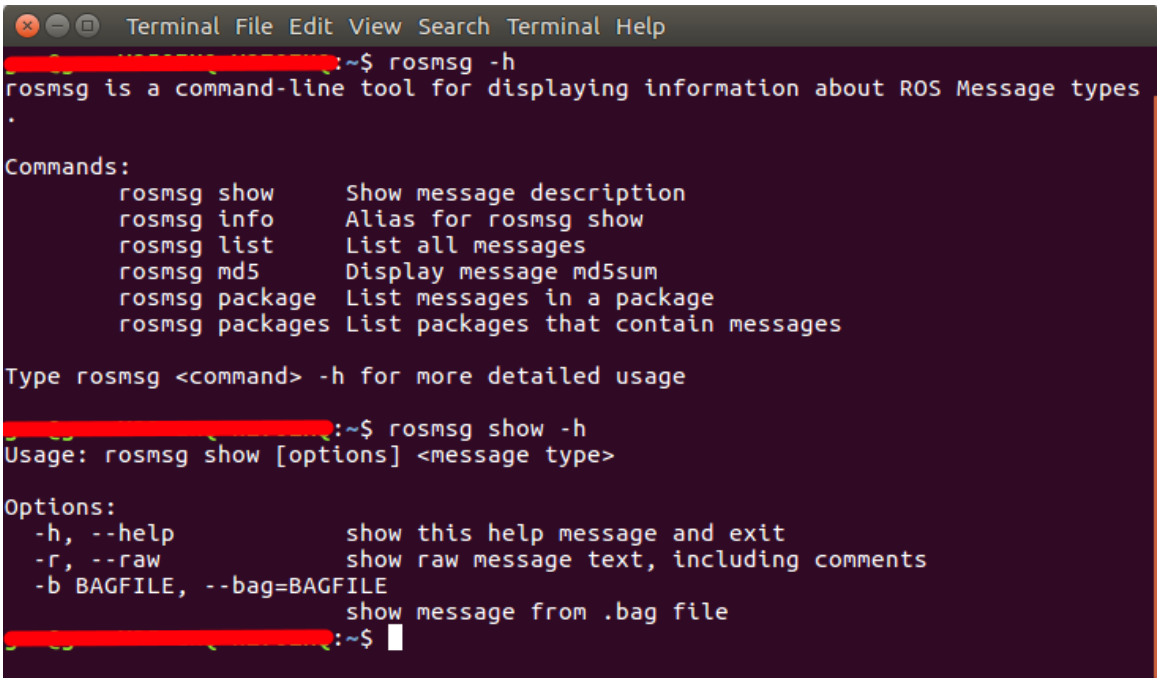
Παρατηρούμε ότι η υπηρεσία χωρίζεται σε δύο τμήματα. Το πρώτο τμήμα είναι του αιτήματος, δηλαδή ποιόν τύπο δεδομένων καθώς και τον αριθμό των μεταβλητών που δέχεται σαν αίτηση, και το δεύτερο τμήμα είναι της απάντησης , δηλαδή τον αριθμό και το είδος τον μεταβλητών που θα πάρουμε ως απάντηση.

**Βοήθεια μετης εντολές του ROS**

Εάν δε θυμόμαστε την ακριβή σύνταξη μιας εντολής του ROS μπούμε να πληκτρολογήσουμε την εντολή και το **-**h και το ROS θα μας δείξει τον τρόπο σύνταξή της. Για παράδειγμα αν πληκτρολογήσουμε:

**$ rosmsg -h**

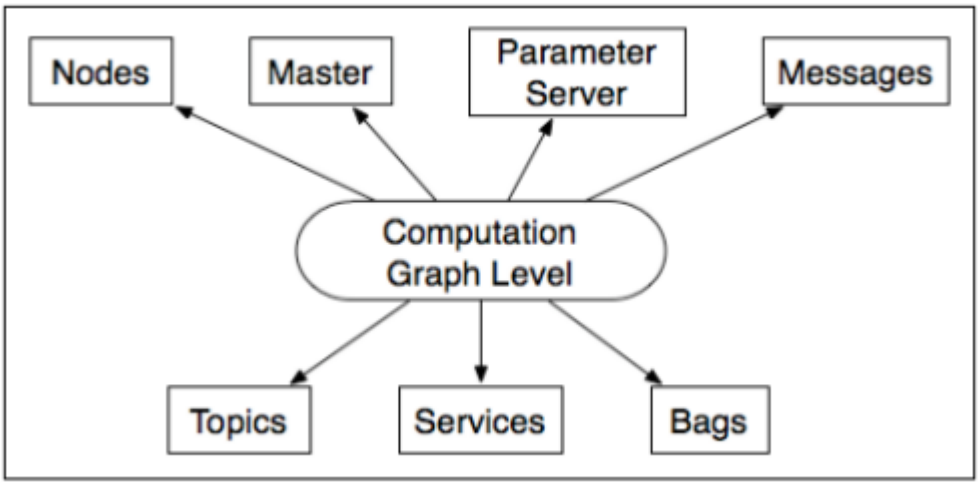
Θα πρέπει στον τερματικό να μας εμφανιστεί  το μήνυμα της εικόνας 2.4.2




*Εικόνα 2.4.2 Βοήθεια εντολών*

## 2.5 Κατανόηση του τμήματος επεξεργασίας του ROS

Το Ros στο κομμάτι της επεξεργασίας δημιουργεί ένα δίκτυο στο οποίο κάθε κομμάτι αυτού μπορεί να αλληλεπιδρά με τα υπόλοιπα βλέποντας τα δεδομένα τα οποία μπορεί να λαμβάνονται, επεξεργάζονται ή στέλνονται, στην εικόνα 2.5.1 φαίνεται μια τυπική μορφή αυτό του δικτύου.




Εικόνα 2.5.1 Τμήμα επεξεργασίας του πηγή ROS Learning ROS for Robotics Programming

Τα βασικά στοιχεία αυτού το δικτύου είναι οι κόμβοι ,ο κυρίαρχος κόμβος, ο διακομιστής παραμέτρου τα μηνύματα, τα θέματα (θεμάτων), οι υπηρεσίες και τα bags.

- Οι **κόμβοι**(**Nodes**) είναι τα σημεία στα οποία γίνεται η επεξεργασία των δεδομένων. Οι κόμβου μπορούν να επικοινωνούν μέσω των υπηρεσιών και των θεμάτων, δηλαδή κάθε κόμβος μπορεί να λαμβάνει ή να στέλνει δεδομένα σε έναν άλλον κόμβο με μεγάλη ευκολία αφού βρίσκονται στο ίδιο δίκτυο. Κάθε κόμβος πρέπει να έχει ξεχωριστό όνομα και ένα node-handle ώστε να διασφαλίζεται η μοναδικότητα και η αποφυγή απώλειας δεδομένων. Σε μεγάλα συστήματα προτιμάται η ύπαρξη πολλών και μικρών κόμβων όπου ο καθένας με ξεχωριστές λειτουργίες που εξυπηρετούν όμως έναν κοινό σκοπό παρά την από την ύπαρξη ενός μεγάλου κόμβου.

- Ο **κυρίαρχος κόμβος**(**Master**) είναι ο κεντρικός κόμβος του συστήματος. Κάθε σύστημα για να μπορεί να λειτουργεί σωστά είναι απαραίτητο να έχει έναν και μοναδικό κυρίαρχο κόμβο, διότι αυτός είναι υπεύθυνος για να ονοματοθεσία τον υπόλοιπων κόμβων καθώς και για τον ορισμό των node-handle. Χωρίς αυτόν θα ήταν αδύνατη η δυνατότητα επικοινωνίας των υπόλοιπων κόμβων.

- Ο **διακομιστής παραμέτρου** (**Parameter Server**) μας δίνει την δυνατότητα να αποθηκεύουμε δεδομένα σε μια κεντρική τοποθεσία. Έτσι μας δίνετε η μπορούμε να διαμορφώνουμε τους κόμβους καθώς βρίσκονται σε λειτουργία καθώς και να αλλάζουμε την λειτουργία τους.
- **Μηνύματα(Masseges)**

- Τα **topics(θέματα)** είναι οι ροές που μεταφέρουν τα μηνύματα από τον έναν κόμβο στον άλλον. Όταν ένας κόμβος στέλνει μηνύματα τότε εκδίδει(publishing) ένα θέμα και για να λάβει ένας άλλος κόμβος αυτό το μήνυμα πρέπει να κάνει εγγραφή (subscribe) στο συκγεκριμένο θέμα. Επιπλέον τα θέματα μπορούν να εκδίδονται χωρίς απαραίτητα να οδηγούν απευθείας σε έναν κόμβο. Τέλος είναι σημαντικό κάθε θέμα να διαθέτει μοναδικό όνομα.

- Οι **υπηρεσίες**(**Services**) μας βοηθούν να μεταφέρουμε δεδομένα από τον έναν κόμβο στον άλλον, είναι παρόμοιες με τα θεμάτα όμως διαφέρουν στην δομή. Τα θεμάτα βασίζονται στην peer to peer επικοινωνία. Ενώ στις περιπτώσεις που χρειαζόμαστε αίτηση(request) ή απάντηση από έναν κόμβο χρησιμοποιήσουμε τις υπηρεσίες. Επίσης το όνομα κάθε υπηρεσίας πρέπει να είναι ξεχωριστώ για τους ίδιους λόγος. Τέλος αν ένας κόμβος διαθέτει μια υπηρεσία τότε όλοι οι υπόλοιποι κόμβοι μπορούν να επικοινωνήσουν μαζί του.

- Τα **bags** είναι μια διαμόρφωση αρχείων η οποία μας επιτρέπει να αποθηκεύουμε και να αναπαράγουμε δεδομένα. Τα ποίο σύνηθες δεδομένα αποθήκευσης είναι είναι από αισθητήρες laser τα οποία δύσκολα καταγράφονται αλλά είναι πολύ σημαντικά για την λειτουργία των IAVs.

## 2.6 Οι κόμβοι του ROS

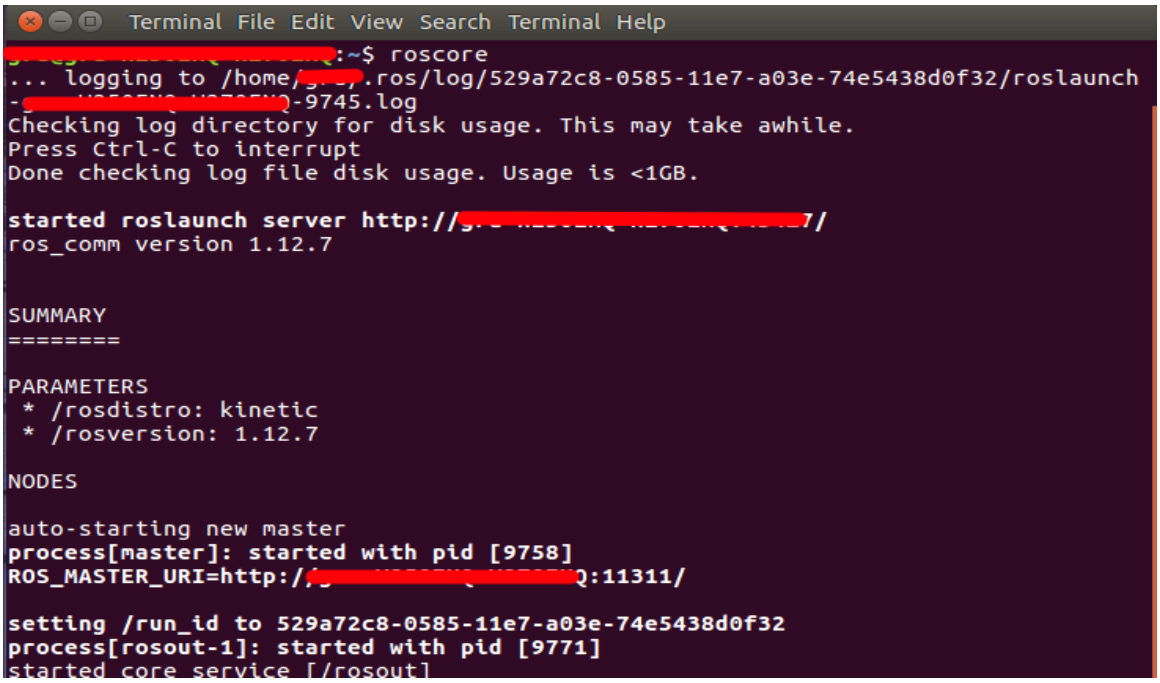
Στην πράξη ένας κόμβος είναι ένα εκτελέσιμο πρόγραμμα γραμμένο είτε με c++ μέσω της roscpp είτε με python μέσω της rospy. Το ROS μας προμηθεύει με διάφορες βιβλιοθήκες που κάνουν την επικοινωνία μεταξύ των κόμβων πάρα πολύ εύκολη. Η roscpp κα ηι rospy είναι δύο από αυτές.

### 2.6.1 Αρχικοποίηση και διαχείριση

Για να δημιουργήσαμε ένα σύστημα σαν αυτό της εικόνας 2.5.1 πρέπει να αρχικοποιήσουμε τον κυρίαρχο κόμβο ή master node διότι είναι αυτός που θα μας επιτρέψει την επικοινωνία με τους υπόλοιπους κόμβους. Για να αρχικοποιήσουμε τον κυρίαρχο κόμβο ανοίγουμε ένα τερματικό και πληκτρολογούμε:

**$ roscore**

Μετά την εκτέλεση της εντολή στον τερματικό θα πρέπει να μας εμφανίζει ένα παρόμοιο μήνυμα με αυτό της εικόνας 2.6.1.

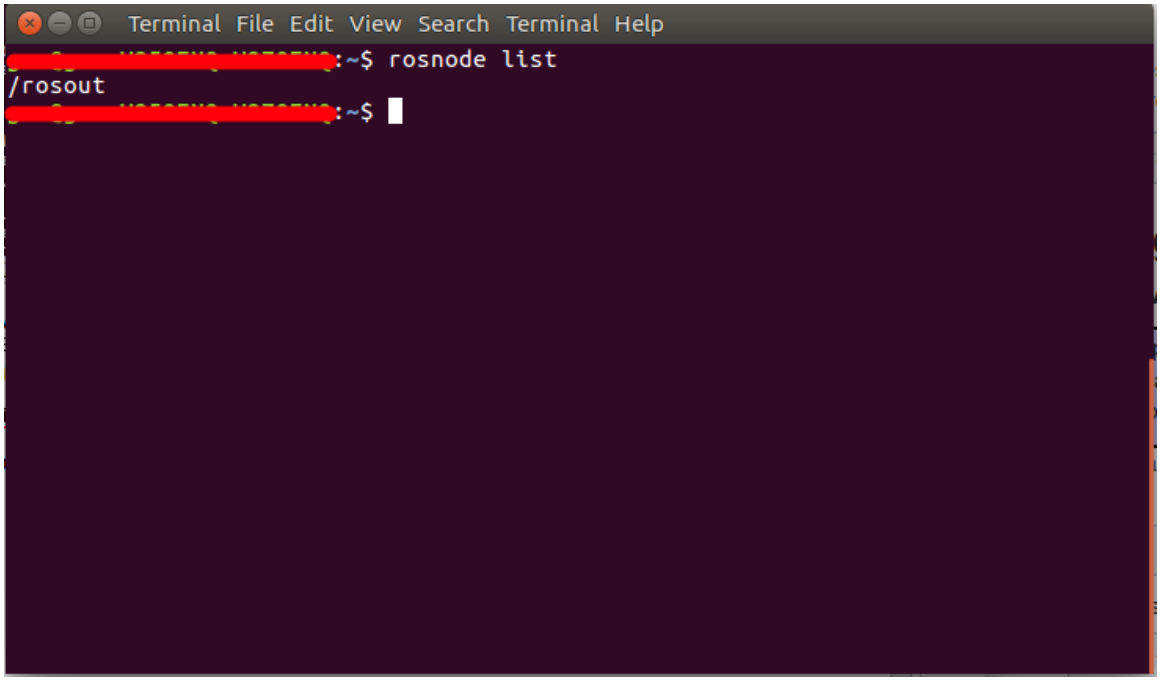



*Εικόνα 2.6.1 Αρχικοποίηση του roscore*

Το ROS μας δίνει την δυνατότητα να παρακολουθούμε ή να αναζητούμε πληροφορίες για τους ενεργούς μας κόμβους με την βοήθεια της εντολής rosnode. Για να βρούμε ποίους και πόσους ενεργούς κόμβους έχει το σύστημα μας ανοίγουμε ένα νέο τερματικό χωρίς όμως να κλείσουμε το προηγούμενο πληκτρολογούμε:

**$ rosnode list**

Και εφόσον δεν έχουμε κλείσει τον τερματικό στον οποίο τρέχει ο κυρίαρχος κόμβος θα μας εμφανίσει:

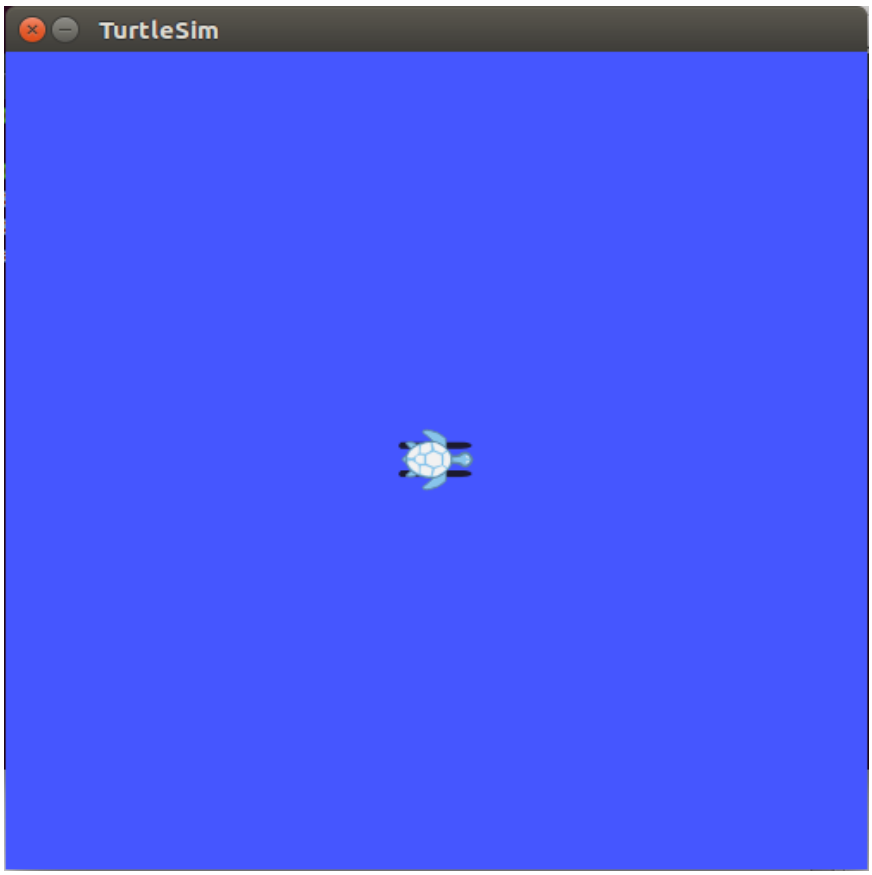



*Εικόνα 2.6.2 Λίστα με τους ενεργούς κόμβους*

Για να δημιουργήσουμε έναν νέο κόμβο χρησιμοποιήσουμε την εντολή rosrun. Η εντολή αυτή χρησιμοποιείται για αρχεία c++ και python και η σύνταξη αυτής είναι, “rosrun (το όνομα του πακέτου) (το όνομα του αρχείου), για παράδειγμα:

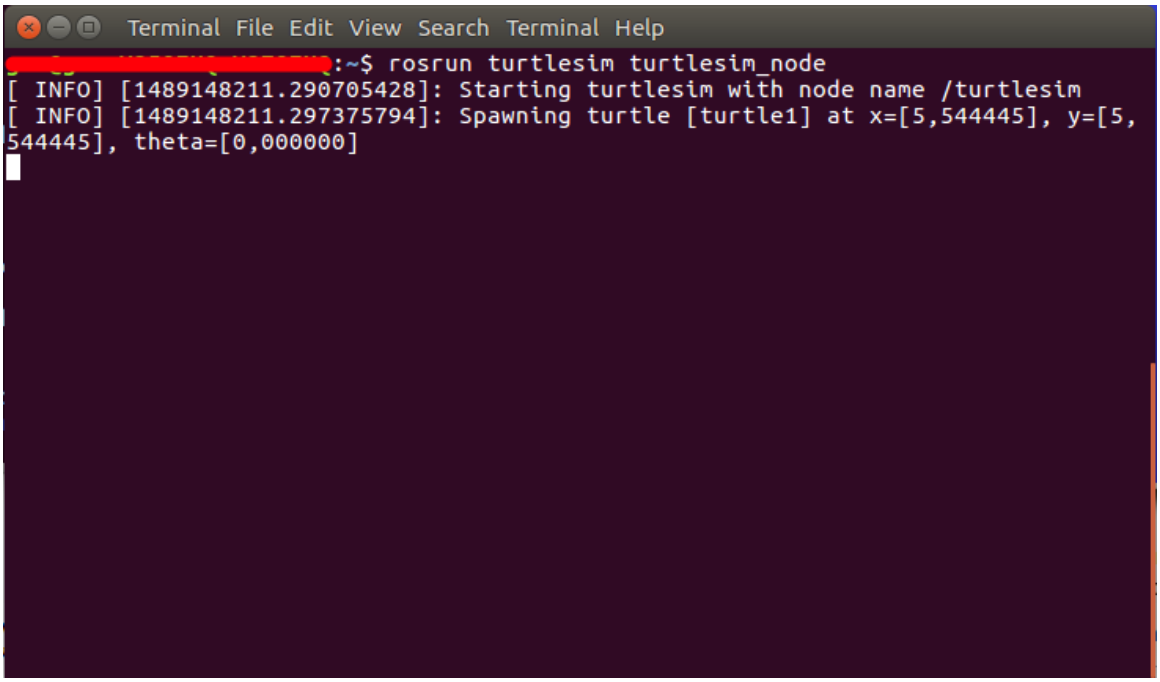
**$ rosrun turtlesim turtlesim_node**

Πληκτρολογώντας την παραπάνω εντολή θα μας ανοίξει το το παράθυρο της εικόνας 2.6.3.




*Εικόνα 2.6.3 Κόμβους TurtleSim*

Δεν είναι αναγκαίο να είναι η ίδια ακριβώς χελώνα διότι το ROS διαθέτει μεγάλη ποικιλία, Επίσης στο τερματικό που εκτελέσαμε την εντολή μας εμφανίζει πληροφορίες σχετικά με τον κόμβο και την θέση της εικόνας.

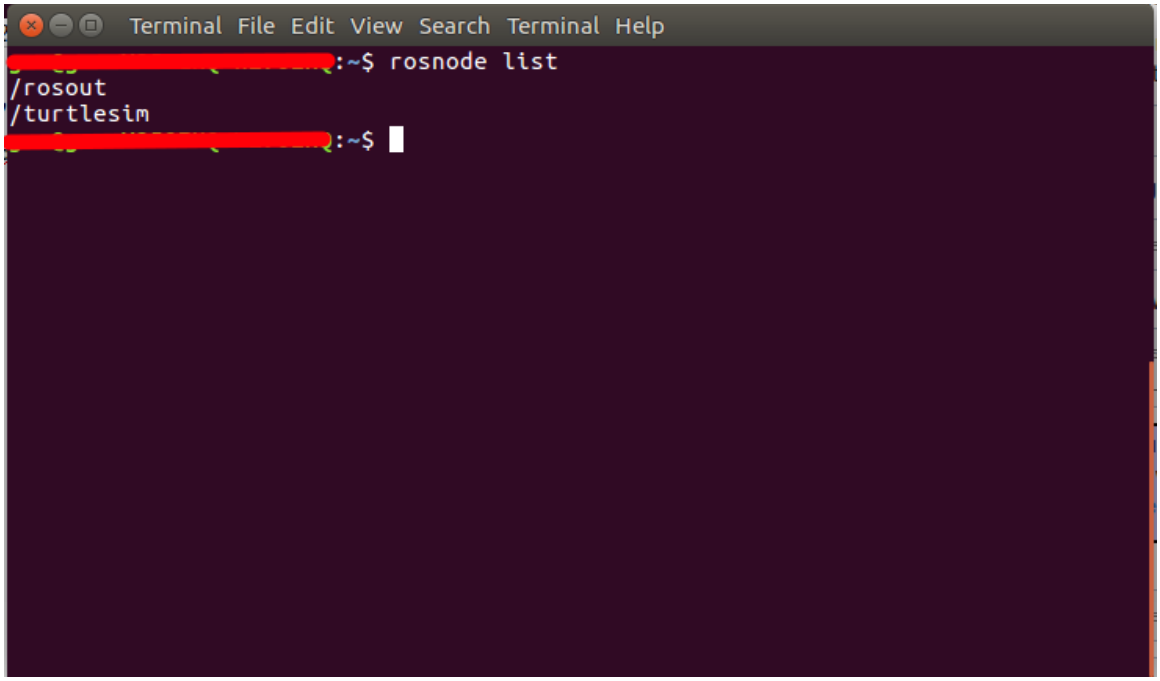



*Εικόνα 2.6.4 Κόμβος TurtleSim*

Ανοίγοντας ένα νέο τερματικό χωρίς όμως να κλείσουμε τα προηγούμενα δύο πληκτρολογούμε:

**$ rosnode list**

Στην εικόνα 2.6.4 παρατηρούμε ότι πλέον υπάρχουν δυο ενεργοί κόμβοι.




*Εικόνα 2.6.5 Λίστα με τους ενεργούς κόμβους*

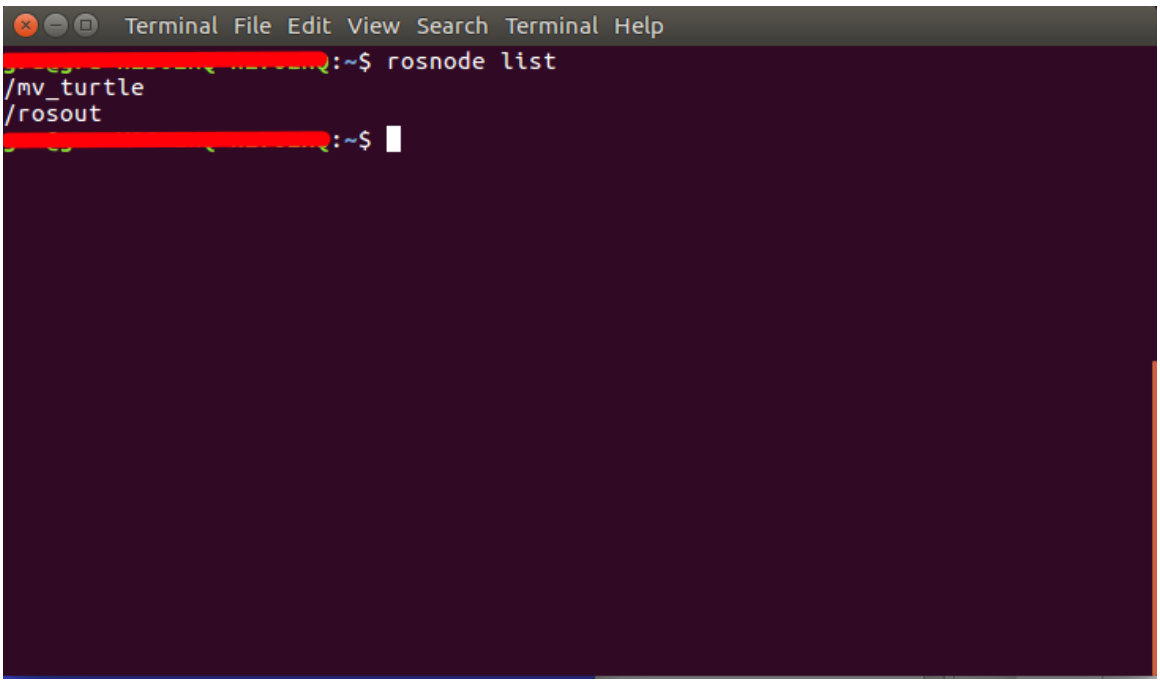
Μια ακόμα πολύ χρήσιμη δυνατότητα που μας προσφέρει το ROS είναι ότι μπορούμε να ονομάσουμε τον κόμβο με την χρήση της ίδιας εντολής με αυτήν την οποία τον αρχικοποιούμε, Αρχικά πρέπει να τερματίσουμε τον ήδη ανοιχτό κόμβο, οπότε στο τερματικό που τρέχει ο κόμβος μας πληκτρολογούμε Ctrl +c και στην συνέχεια εκτελούμε την παρακάτω εντολή:

**$ rosrun turtlesim turtlesim_node __name:=mv_turtle**

Πληκτρολογώντας σε ένα νέο τερματικό:

**$ rosnode list**

Όπως φαίνεται στην εικόνα 2.6.6 παρατηρούμε ότι όντως έχει αλλάξει το όνομα του κόμβου.




*Εικόνα 2.6.5 Λίστα με τους ενεργούς κόμβους*

Τέλος τερματίζουμε όλους τους ενεργούς κόμβους.

## 2.7 ROS topics

Τα θεμάτα(topics) είναι βασισμένα στον τύπο μηνυμάτων του ROS και προκειμένου ένας κόμβος να μπορεί να λάβει ένα μήνυμα από οποιοδήποτε θέμα πρέπει να εμπεριέχει τον ίδιο τύπου μηνύματος με αυτόν του θέματος.Τα θέματα για να μεταδίδονται χρησιμοποιούν TCP/IP μέσω του TCPROS και UDP μέσω του UDPROS σαν τρόπους επικοινωνίας, με τον TCPROS να είναι ο προκαθορισμένος, ενώ ο UDPROS χρησιμοποιείται περισσότερο για τον χειρισμό του IAV από απόσταση.

### 2.7.1 Κατανόηση των θεμάτων

Αρχικά, για την καλύτερη κατανόηση των θεμάτων θα χρειαστούμε ένα εργαλείο το οποίο δεν περιλαμβάνεται στην αρχική εγκατάσταση του ROS, για αυτό τον λόγο θα πρέπει να εγκαταστήσουμε δυο παρακάτω πακέτα:

**$ sudo apt-get install ros-kinetic-rqt**

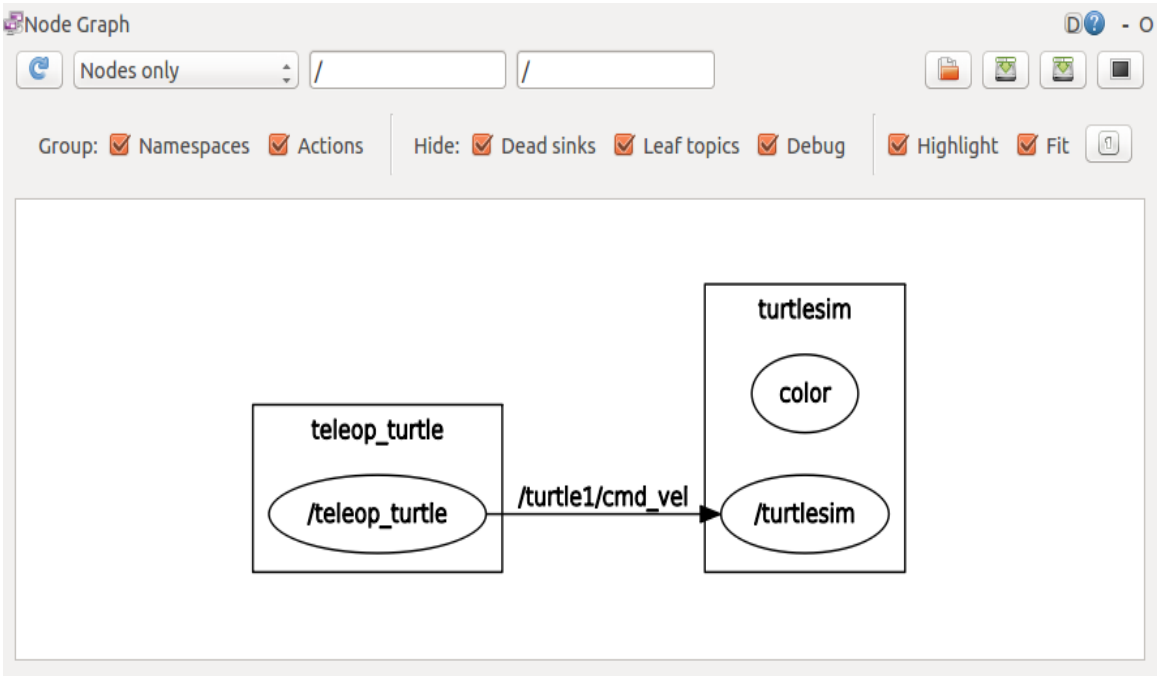
**$ sudo apt-get install ros-kinetic-rqt-common-plugins**

Έπειτα πρέπει να ακολουθήσουμε την ίδια διαδικασία με αυτήν της προηγούμενης ενότητας μόνο αυτή τη φορά θα ενεργοποιήσουμε ακόμα έναν κόμβο μέσω του οποίο θα έχουμε την δυνατότητα να κινούμε την χελώνα εκδίδοντας μηνύματα στον κόμβο της χελώνας μέσω ενός θέματος. Αφού έχουμε αρχικοποιήσει τον κυρίαρχο κόμβο και τον κόμβο της χελώνας(παράγραφος 2.6) ανοίγουμε ένα νέο τερματικό για να αρχικοποιούμε και τον κόμβο με τον οποίο θα κινούμε την χελώνα:

**$ rosrun turtlesim turtle_teleop_key**

Για να μπορέσουμε να κινήσουμε την χελώνα, δηλαδή να στείλουμε μηνύματα μέσω ενός θέματος με την χρήση των βέλων (όπως αναγράφεται και στον τερματικό) πρέπει έχουμε επιλεγμένο τον τελευταίο τερματικό. Για να μπορέσουμε να δούμε μέσω γραφήματος το παραπάνω σύστημα και την επικοινωνία η οποία πραγματοποιήθηκε ανοίγουμε ένα νέο τερματικό και πληκτρολογούμε:

**$ rosrun rqt_graph rqt_graph**




*Εικόνα 2.7.1 Γράφημα κόμβων/θεμάτων*

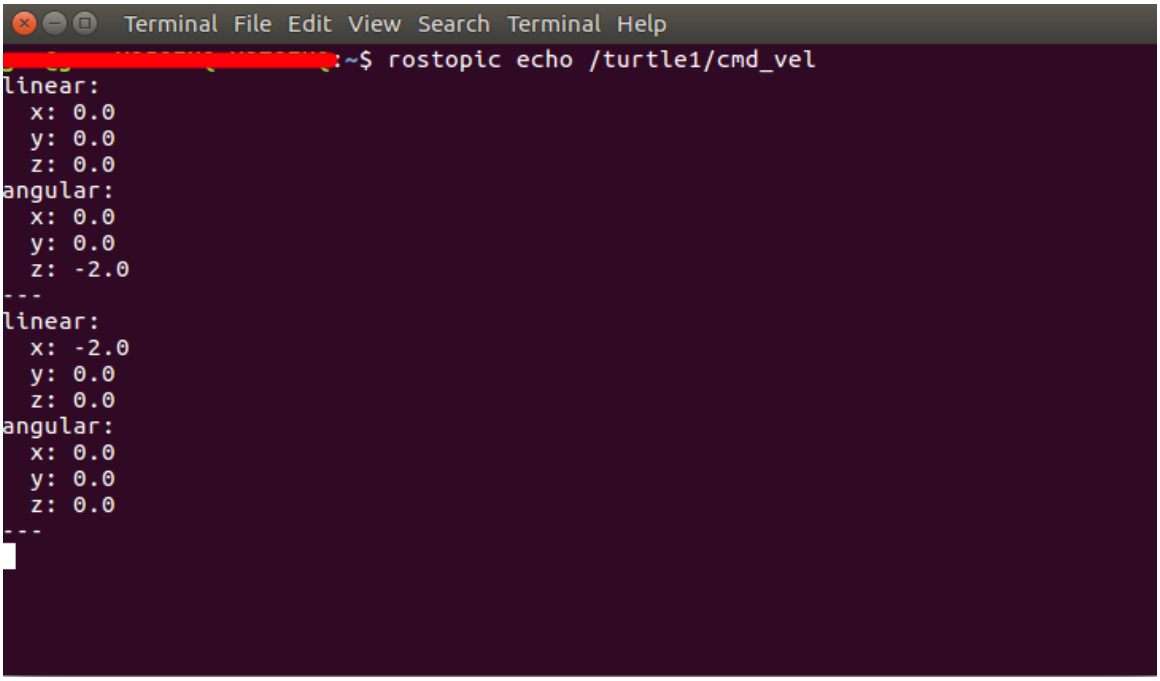
Στην εικόνα 2.7.1 με τετράγωνο απεικονίζονται οι κόμβοι και με βέλη τα θεμάτα. Όπως παρατηρούμε υπάρχουν δύο κόμβοι (τρεις στο σύνολο διότι δεν απεικονίζεται ο κυρίαρχος κόμβος), ο ένας είναι ο κόμβος της χελώνας ο όπου δέχεται μηνύματα από τον δεύτερο κόμβο του τηλεχειρισμού ο οποίο στέλνει τα μηνύματα μέσω ενός θέματος που ονομάζεται /turtle1/cmd_vel.

### 2.7.2 Διαχείριση των θέματων

Το εργαλείο που μας παρέχει το ROS για την διαχείριση των θεμάτων είναι το rostopic με το οποίο όχι μόνο έχουμε την δυνατότητα να λάβουμε πληροφορίες όπως τον τύπο μηνυμάτων αλλά μπορούμε να δούμε και τα μηνύματα που στέλνονται ή λαμβάνονται καθώς και να εκδώσουμε και τα δικά μας μηνύματα σε οποιονδήποτε κόμβο.Για να μπορέσουμε να δούμε ένα μήνυμα το οποίο μεταδίδεται χρησιμοποιούμε το rostopic με την εξής σύνταξη **rostopic echo** (το όνομα του θέματος), δηλαδή:

**$ rostopic echo /turtle1/cmd_vel**

Ανοίγοντας ένα νέο τερματικό πληκτρολογούμε την παραπάνω εντολή, στη συνέχεια πρέπει να μεταβούμε στον τερματικό στον οποίο έχουμε ενεργοποιήσει τον κόμβο teleop_turtle και να μετακινήσουμε την χελώνα. Τέλος όταν μεταβούμε στον τερματικό στον οποίο έχουμε εκτελέσει την παραπάνω εντολή θα παρατηρήσουμε ένα ανάλογό μήνυμα με αυτό της εικόνας 2.7.2.




*Εικόνα 2.7.2 Απεσταλμένο μήνυμα μέσω topic*

Η λογική και η λειτουργία που κρύβεται πίσω από αυτήν την εντολή είναι ότι δημιουργείται ένας προσωρινός κόμβος οποίο κάνει εγγραφή τον κόμβο στον οποίο στέλνει τα δεδομένα και έτσι μπορεί να έχει πρόσβαση σε αυτά.Για να μπορέσουμε να να δούμε τον αριθμό καθώς και τον τύπο μηνυμάτων ο οποίο αποστέλνεται αρκεί να πληκτρολογήσουμε της παρακάτω δύο εντολές:

**$ rostopic list**

Αφού επιλέξουμε ένα θέμα από αυτήν την λίστα:

**$ rostopic type /turtle1/cmd_veλ**

θα πάρουμε ως απάντηση.

**geometry_msgs/Twist**

Τέλος η πιο σημαντική λειτουργία του rostopic είναι ότι μας δίνει την δυνατότητα να εκδώσουμε ένα μήνυμα μέσω ενός θέματος χωρίς την χρήση κώδικα, το μόνο που χρειαζόμαστε είναι το όνομα του θέματος τον τύπο του μηνύματος και φυσικά τα δεδομένα, Ο τρόπος με τον οποίο μπορούμε να πάρουμε αυτές τις πληροφορίες έχει καλυφθεί σε προηγούμενες ενότητες.Η σύνταξη της εντολής είναι “rostopic pub (εντολή) (το όνομα του θέματος) (δεδομένα)”.Σε ένα νέο τερματικό πληκτρολογούμε:

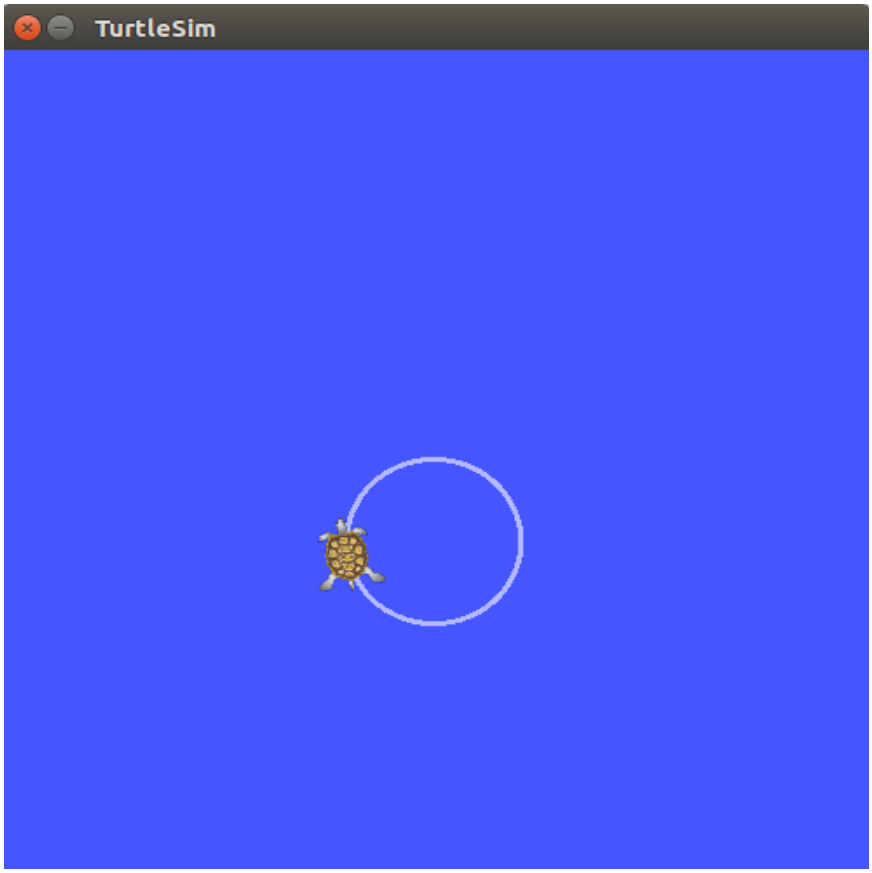
**$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'**

Οι δύο κάθετες παύλες στην εντολή “--“ τοποθετούνται για να δείξουν ότι η δήλωση αυτή δεν είναι υποχρεωτική και την τοποθετούμε σε περιπτώσεις που μπορεί να υπάρξουν αρνητική αριθμοί στα δεδομένα. Επιπλέον το “-1” δηλώνει στο ROS ότι πρέπει να φτιάξει έναν προσωρινό κόμβο να μεταδώσει μια φορά το

μήνυμα και τέλος να διαγράψει τον κόμβο. Αν μεταβούμε στο παράθυρο που απεικονίζεται η χελώνα θα δούμε ότι έχει μετακινηθεί.
Για να στέλνουμε συνεχή μηνύματα αρκεί να δηλώσουμε μια συχνότητα αποστολής μηνύματος, 1Hz για παράδειγμα. Αυτό γίνεται με την εντολή:

**$ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'**

Η δήλωση “-r 1” είναι αυτή που ρυθμίζει συχνότητα 1Hz. Αν μεταβούμε στο παράθυρο με την χελώνα θα δούμε ότι εκτελεί μια μόνιμη κυκλική κίνηση.




*Εικόνα 2.7.3 Συνεχής έκδοση μηνυμάτων*

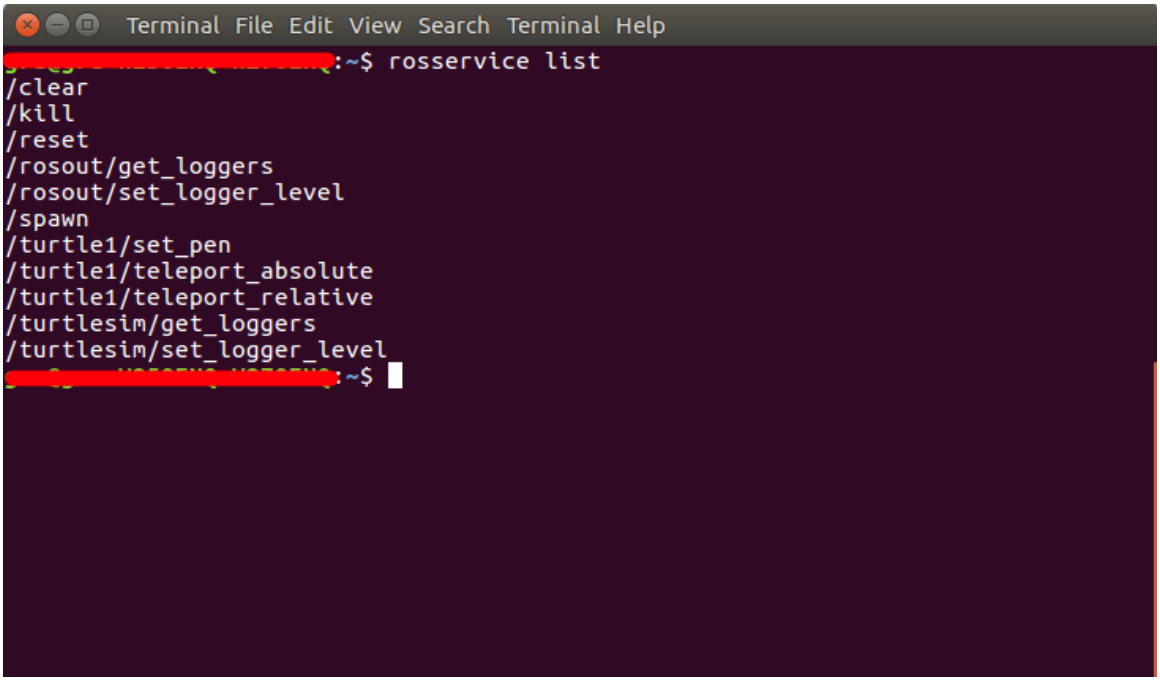
## 2.8 Υπηρεσίες και διακομιστής παραμέτρου του ROS

### 2.8.1 Υπηρεσίες του ROS

Όπως έχουμε ήδη αναφέρει οι υπηρεσίες είναι ένας άλλος τρόπος που χρησιμοποιεί το ROS ώστε οι κόμβοι να επικοινωνούν μεταξύ τους. Μέσω μια υπηρεσίας ο κόμβος που θέλει να επικοινωνήσει με έναν άλλον κόμβο στέλνει μια αίτηση και ο δεύτερος στέλνει μια απάντηση στην αίτηση του πρώτου.Το εργαλείο που χρησιμοποιεί το ROS για να διαχειρίζεται της υπηρεσίες είναι το rosservice. Για να χρησιμοποιήσουμε μια υπηρεσία αρχικά θα πρέπει να γνωρίζουμε ποίες είναι η διαθέσιμες υπηρεσίες κάθε κόμβου. Για να αποκτήσουμε αυτές της πληροφορίες πληκτρολογούμε:

**$ rosservice list**

Υποθέτοντας ότι δεν έχουμε κλειστούς κόμβους από τις προηγούμενες ενότητες θα μας εμφανιστεί το μήνυμα της εικόνας 2.8.1.




*Εικόνα 2.8.1 Λίστα με διαθέσιμες υπηρεσίες*

Το αμέσως επόμενο βήμα για να μπορέσουμε μια υπηρεσία είναι να ξέρουμε ποία είναι τα ορίσματα της και τι τύπο δεδομένων θα μας επιστρέψει,παίρνουμε για παράδειγμα την υπηρεσία /clear κα πληκτρολογούμε:

**$ rosservice type /clear| rossrv show**

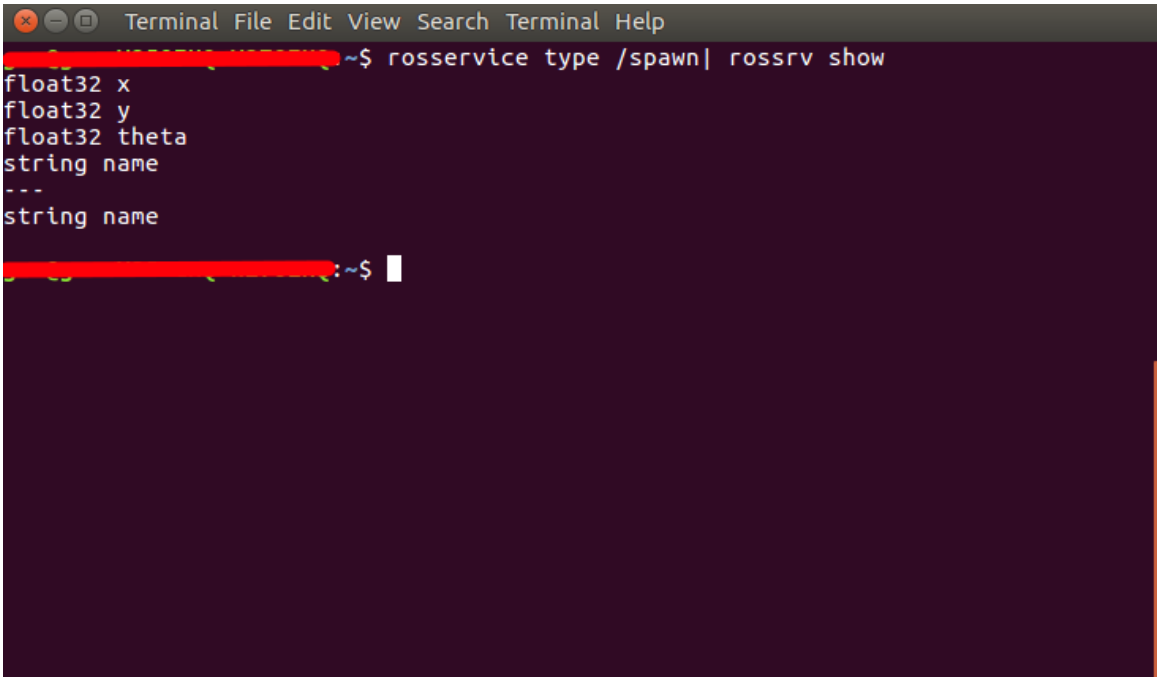
Θα πρέπει να πάρουμε ως απάντηση:

**---**

Αυτό σημαίνει ότι η συγκεκριμένη υπηρεσία δεν έχει κάποιο όρισμα και δεν επιστρέφει δεδομένα ως απάντηση.
Ωστόσο αν επιλέξουμε την υπηρεσία /spawn:

**$ rosservice type /spawn| rossrv show**

Παίρνουμε ως απάντηση το μήνυμα της εικόνας 2.8.2




*Εικόνα 2.8.2 Τύπος δεδομένων υπηρεσίας*

Το πάνω μέρος του μηνύματος δηλώνει τον τύπο δεδομένων που δέχεται σαν όρισμα η αίτηση ενώ το κάτω τον τύπο δεδομένων που δέχεται σαν όρισμα η απάντηση.
Πλέον γνωρίζουμε ότι χρειαζόμαστε για να καλέσουμε μια υπηρεσία πληκτρολογώντας:

**$ rosservice call /clear**
**$ rosservice call /spawn 2 2 0.2 ""**

Εάν τώρα μεταφερθούμε στο παράθυρο με την χελώνα αφενός μεν θα παρατηρήσουμε ότι η άσπρη γραμμή που αντιπροσωπεύει με πορεία της από τότε που ενεργοποιήσαμε τον κόμβο έχει σβηστεί εξαιτίας της πρώτης υπηρεσίας, αφετέρου δε έχει εμφανιστεί μια νέα χελώνα. Μια σημαντική παρατήρηση είναι ότι η απάντηση στην αίτηση που στείλαμε λήφθηκε στον τερματικό τον οποίο στάλθηκε η υπηρεσία και είναι η ακόλουθος.

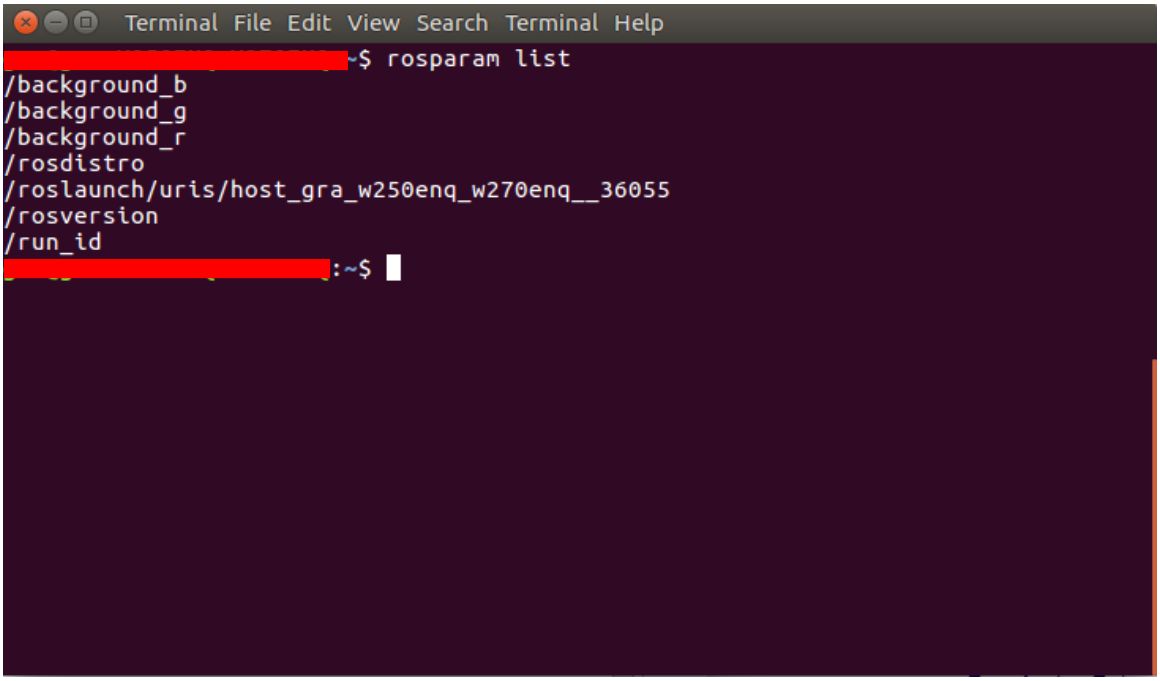
**name: turtle2**

### 2.8.2 Διακομιστής παραμέτρου του ROS

Όπως έχουμε προαναφέρει ο διακομιστής παραμέτρου χρησιμοποιείται για την καταγραφή και αποθήκευση δεδομένων που όμως είναι προσβάσιμα από όλους τους κόμβους στο δικτύου. Το εργαλείο για την διαχείριση αυτών τον δεδομένων είναι το rosparam**.** Για να είμαστε σε θέση να το χρησιμοποιήσουμε όπως και με το rosservice θα πρέπει να γνωρίζουμε την λίστα με της διαθέσιμους παραμέτρους, πληκτρολογούμε:

**$ rosparam list**

Επομένως η λίστα με της διαθέσιμες παραμέτρους είναι η ακόλουθος.



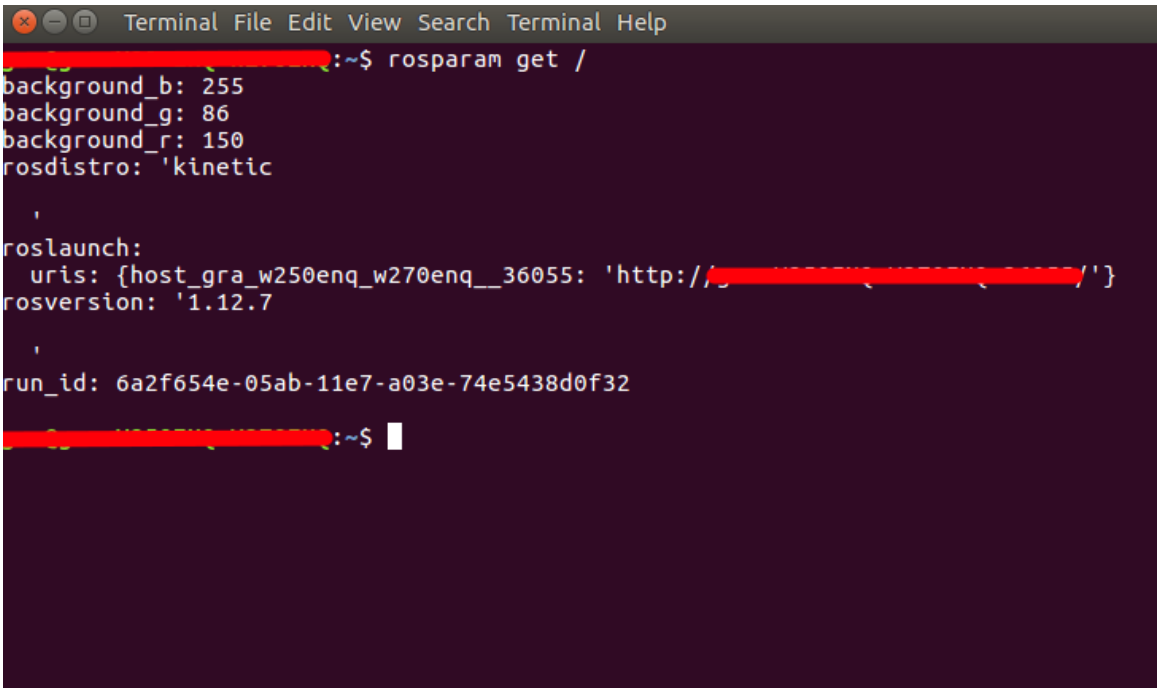

*Εικόνα 2.8.3 Διαθέσιμες παράμετροι*

Με ανάλογο τρόπο με αυτόν των υπηρεσιών μπορούμε να πάρουμε πληροφορίες σχετικές με τα ορίσματα της παραμέτρου, Επιλέγοντας την παράμετρο /background_r μπορούμε να αλλάξουμε το χρώμα στο φόντο του παραθύρου:

**$ rosparam set /background_r 150**
**$ rosservice call /clear**

Για να αλλάξει η παράμερος πρέπει να καλέσουμε την αίτηση /clear Παρατηρούμε ότι το φόντο του παραθύρου έγινε μοβ. Τέλος για να πάρουμε πληροφορίες για τις τιμές των παραμέτρων πληκτρολογούμε:

**$ rosparam get /**




*Εικόνα 2.8.4 Τιμές των ενεργών παράμετρων*

## 2.9 Εκδότες και Συνδρομητές (Publishers-Subscribers)

Έχοντας αναλύσει τον βασικό κορμό του ROS ήρθε η ώρα να υλοποιήσουμε έναν συνδρομητή και έναν εκδότη με την χρήση κώδικα γραμμένο σε c++. Το εγχείρημα αυτό δεν είναι παρά ένας κόμβος που εκδίδει μηνύματα μέσω ενός θέματος και ένας δεύτερος κόμβος ο που εγγράφεται στο συγκεκριμένο θέμα και λαμβάνει τα μηνύματα.

### 2.9.1 Δημιουργία του κόμβου εκδότη

Ο κώδικας για τον εκδότη είναι:

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv)
{

 ros::init(argc, argv, "publisher");

 ros::NodeHandle n;

 ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);

 ros::Rate loop_rate(10);

int count = 0;

while (ros::ok())
 {

   std_msgs::String msg;

   std::stringstream ss;

   ss << "hello world " << count;

   msg.data = ss.str();

   ROS_INFO("%s", msg.data.c_str());

   chatter_pub.publish(msg);

   ros::spinOnce();

   loop_rate.sleep();

    ++count;

 }

return 0;

}
```

Αποθηκεύουμε τον κώδικα στον φάκελο src του πακέτου beginner_tutorials με το όνομα publisher.cpp.

Η συγγραφή του αρχείου μπορεί να γίνει και με την χρήση ενός άδειο αρχείου, μόνο που στο τέλος θα πρέπει να αλλάξουμε την κατάληξη του σε c++.

**Εξήγηση του κώδικα**

- Εισάγουμε την βασική επικεφαλίδα του ROS και την βιβλιοθήκη που περιέχει τους βασικούς τύπους μηνυμάτων.

  **#include "ros/ros.h"**
  **#include "std_msgs/String.h"**

- Αρχικοποιούμε τον κόμβο χρησιμοποιώντας για όρισμα ένα μοναδικό όνομα καθώς και τα argc, argv

  **ros::init(argc, argv, "publisher");**

- Δημιουργούμε μια λαβή(node handle) για τον κόμβο μας διότι αυτή ουσιαστικά είναι υπεύθυνη για την αρχικοποίηση του και έτσι διασφαλίζεται η μοναδικότητά του.

  **ros::NodeHandle n;**

- Δημιουργούμε ένα αντικείμενο της κλάσης publisher και ενημερώνουμε τον κυρίαρχο κόμβο ότι θα εκδώσουμε μηνύματα τύπου std μέσω ενός θέματος που ονομάζεται chatter το οποίο θα αποθηκεύει χίλια μηνύματα πριν αρχίσει να διαγράφει τα πρώτα.

  **ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);**

  Επιλέγουμε την συχνότητα με την οποία θα εκδίδονται τα μηνύματα

  **ros::Rate loop_rate(10);**

- Επιλέγουμε η διαδικασία αυτή να επαναλαμβάνεται μέχρι να κλείσει ο τερματικός και αρχικοποιούμε έναν μετρητή για να γνωρίζουμε τον αριθμό των μηνυμάτων τα οποία έχουν σταλθεί.

  **int count = 0;**
  **while (ros::ok())**

- Δημιουργούμε ένα αλφαριθμητικό αντικείμενο στο οποίο θα αποθηκεύσουμε το μήνυμα που θα σταλθεί καθώς και τον σειριακό αριθμό του.

  **std_msgs::String msg;**

  **std::stringstream ss;**

  **ss << "hello world " << count;**

  **msg.data = ss.str();**

- Στέλνουμε το μήνυμα.

  **chatter_pub.publish(msg);**

- Γράφουμε στο τερματικό το μήνυμα που θα στείλουμε.

  **ROS_INFO("%s", msg.data.c_str());**

- Η παρακάτω εντολή χρησιμοποιείται στις callback συναρτήσεις, χωρίς αυτήν δεν γίνεται η διακοπή της ροής του προγράμματος και εξυπηρέτηση της συνάρτησης, Στο συγκεκριμένο πρόγραμμα η συνάρτηση αυτή είναι προαιρετική.

  **ros::spinOnce();**

- Αφού έχει σταλθεί το μήνυμα περιμένουμε το χρονικό διάστημα που έχουμε δηλώσει παραπάνω και η διαδικασία επαναλαμβάνεται.

  **loop_rate.sleep();**

### 2.9.2 Δημιουργία του κόμβου συνδρομητή

Ο κώδικας για συνδρομητή είναι:

```
# include "ros/ros.h"

# include "std_msgs/String.h"

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{

  ROS_INFO("I heard: [%s]", msg->data.c_str());

}

int main(int argc, char **argv)
{

   ros::init(argc, argv, "subscriber");

   ros::NodeHandle n;

  ros::Subscriber sub = n.subscribe("publisher", 1000, chatterCallback);

  ros::spin();

  return 0;

}
```

Η αποθήκευση του κώδικα θα γίνει στον ίδιο φάκελο με το όνομα subsrciber.cpp.

**Εξήγηση του κώδικα**

- Η συνάρτηση λειτουργεί σαν διακοπή της ροής του προγράμματός κάθε φορά που λαμβάνεται ένα μήνυμα**.**

  **void subsrciberCallback(const std_msgs::String::ConstPtr& msg)**

- Δημιουργούμε ένα αντικείμενο της κλάσης Subsrciber με το οποίο δηλώνουμε στον κυρίαρχο κόμβο ότι επιθυμούμε να εγγραφούμε στο θέμα, να αποθηκεύουμε χίλια μηνύματα προτού αρχίσουν να διαγράφονται τα παλιά και κάθε φορά που υπάρχει ένα νέο μήνυμα να καλείται η συνάρτηση subsrciberCallback.

  **ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);**

Ενώ έχουμε τελειώσει με την συγγραφή του κώδικα πρέπει να κάνουμε το ROS να αναγνωρίσει ως εκτελέσιμα αυτά αρχεία έτσι ώστε να μπορέσει να τα συμπεριλάβει στην μεταγλώττιση. Ανοίγουμε το αρχείο CmakeLists.txt και προσθέτουμε τις παρακάτω σειρές στο τέλος του αρχείου

**include_directories(include ${catkin_INCLUDE_DIRS})**
**add_executable(publisher src/publisher.cpp)**
**target_link_libraries(publisher ${catkin_LIBRARIES})**
**add_dependencies(publisher beginner_tutorials_generate_messages_cpp)**

**add_executable(subsrciber src/subsrciber.cpp)**
**target_link_libraries(subsrciber ${catkin_LIBRARIES})**
**add_dependencies(subsrciber beginner_tutorials_generate_messages_cpp)**

Ανοίγουμε ένα τερματικό, μεταβαίνουμε στο περιβάλλον εργασίας μας και εκτελούμε την παρακάτω εντολή για να μεταγλωττίσουμε τα αρχεία του περιβάλλοντος εργασίας μας.

**$ catkin_make**

Εφόσον δεν μας εμφανιστεί κάποιο λάθος η μεταγλώττιση ήταν επιτυχές**.**

### 2.9.3 Εκτέλεση εκδότη και συνδρομητή

Αρχικά πρέπει να ενεργοποιήσαμε τον κυρίαρχο κόμβο πληκτρολογώντας:

**$ roscore**

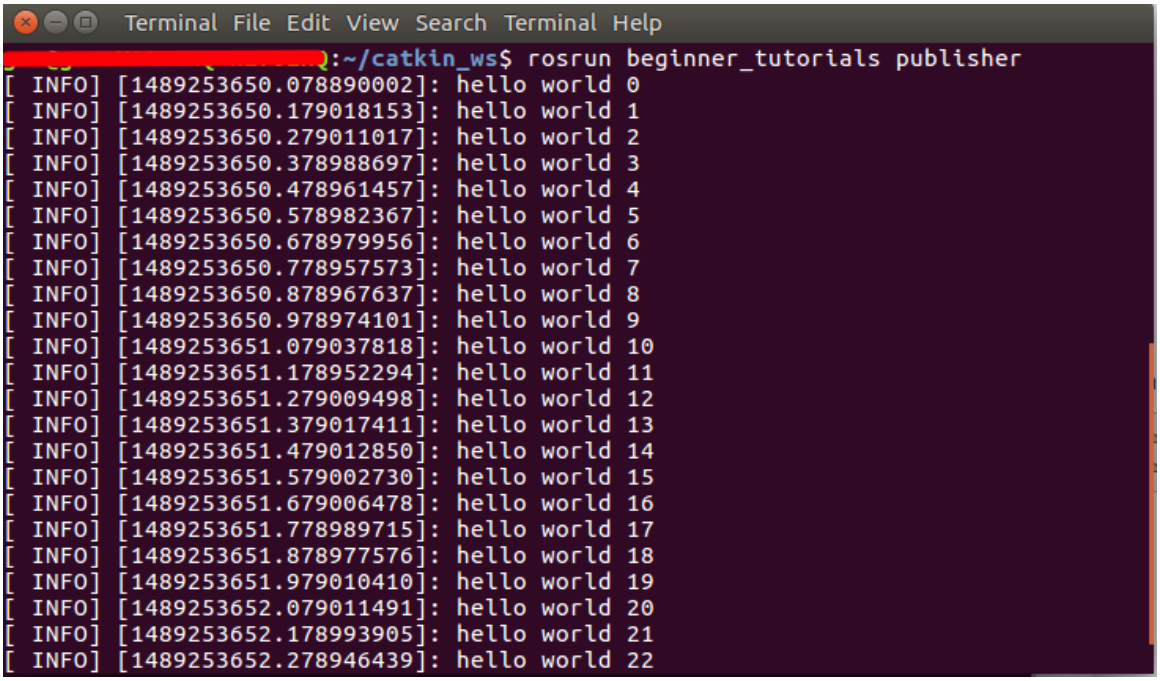
Αφού έχουμε μεταβεί στο περιβάλλον εργασίας μας εκτελούμε την εντολή:

**$ source ./devel/setup.bash**

Και για να ενεργοποιήσαμε τον κόμβο του εκδότη:

**$ rosrun beginner_tutorials publisher**

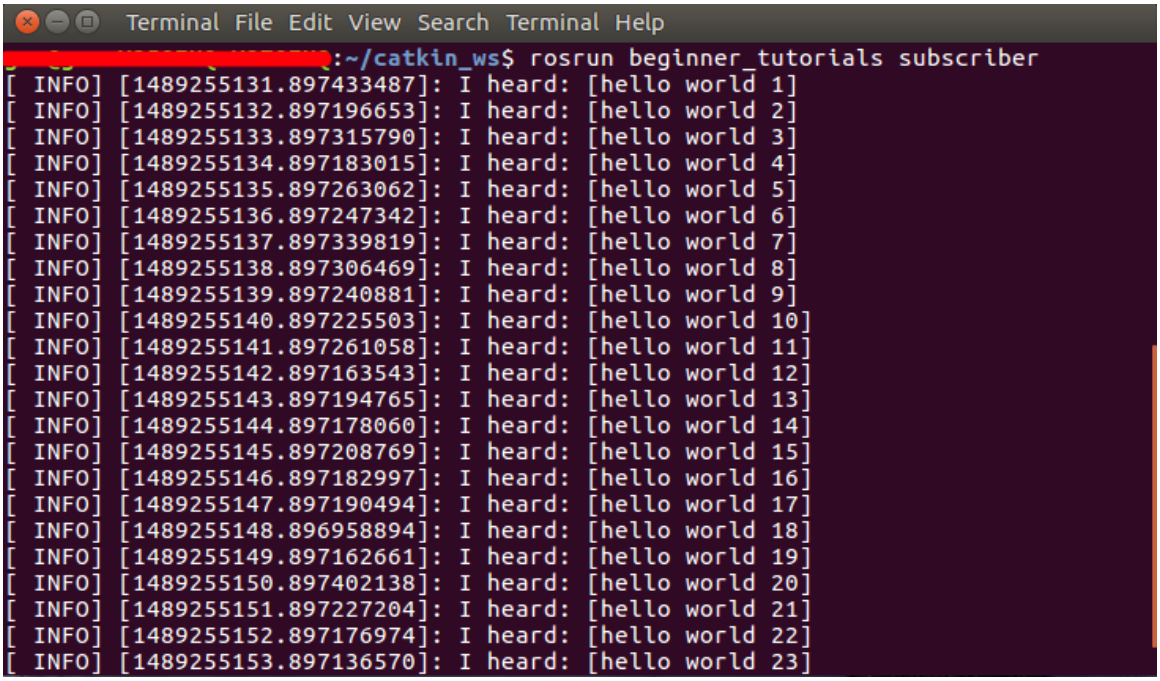
Στο παράθυρο του τερματικού θα μας εμφανιστεί ένα μήνυμα παρόμοιο με της εικόνας 2.9.1




*Εικόνα 2.9.1 Μηνύματα εκδότη*

Εκτελώντας και τον συνδρομητή με την ίδια διαδικασία.

**$ rosrun beginner_tutorials subscriber**




*Εικόνα 4.9.1 Μηνύματα που έχουν ληφθεί*

## 2.10 Υπηρεσία και πελάτης (Service and Client)

Σε προηγούμενη ενότητα δείξαμε πως μπορούμε διαχειριστούμε τις υπηρεσίες του ROS. Σε αυτήν την την ενότητα θα δείξουμε πως μπορούμε να δημιουργήσουμε μια υπηρεσία μέσω ενός προγράμματος c++.

### 2.10.1 Δημιουργία του κόμβου υπηρεσίας

Στον ίδιο φάκελο που αποθηκεύσαμε τον subsrciber και publisher δημιουργούμε ένα αρχείο με το όνομα add_two_ints_service.cpp και αντιγράφουμε τον παρακάτω κώδικα

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "add_two_ints_client");

   if (argc != 3)
 {

  ROS_INFO("usage: add_two_ints_client X Y");

   return 1;

  }

   ros::NodeHandle n;

ros::ServiceClient client = n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");

   beginner_tutorials::AddTwoInts srv;

   srv.request.a = atoll(argv[1]);

   srv.request.b = atoll(argv[2]);

   if (client.call(srv))
  {

   ROS_INFO("Sum: %ld", (long int)srv.response.sum);

 }

  else
 {

   ROS_ERROR("Failed to call service add_two_ints");

    return 1;

  }

  return 0;
```

```
}
```

**Εξήγηση του κώδικα**

- Η επικεφαλίδα περιέχει την υπηρεσία που είχαμε δημιουργήσει σε προηγούμενη ενότητα.

  **#include "beginner_tutorials/AddTwoInts.h"**

- Η συνάρτηση αυτή περιέχει την ζητούμενη υπηρεσία. Είναι μια λογική συνάρτηση η οποία δέχεται σαν είσοδο το αίτημα και την απάντηση και επιστρέφει μια bool μεταβλητή, true αν έχει ολοκληρωθεί ο υπολογισμός της υπηρεσίας, false αν υπήρξε κάποιο πρόβλημα.

  **bool add(beginner_tutorials::AddTwoInts::Request &req,**
  **beginner_tutorials::AddTwoInts::Response &res)**

- Ορίζουμε ένα αντικείμενο της κλάσης Service και έτσι δημιουργούμε την υπηρεσία add_two_ints που με την σειρά της καλεί την συνάρτηση add κάθε φορά που χρειαζόμαστε απάντηση σε κάποιο αίτημα.

  **ros::ServiceServer service = n.advertiseService("add_two_ints", add);**

### 2.10.2 Δημιουργία του κόμβου πελάτη

Δημιουργούμε ένα αρχείο με το όνομα add_two_ints_client.cpp και αντιγράφουμε τον παρακάτω κώδικα

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

 int main(int argc, char **argv)
{

 ros::init(argc, argv, "add_two_ints_client");

     if (argc != 3)

     {

      ROS_INFO("usage: add_two_ints_client X Y");

      return 1;

    }


    ros::NodeHandle n;

ros::ServiceClient client = n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");

    beginner_tutorials::AddTwoInts srv;

    srv.request.a = atoll(argv[1]);

    srv.request.b = atoll(argv[2]);

 if (client.call(srv))
```

```
{
    ROS_INFO("Sum: %ld", (long int)srv.response.sum);
  }
 else
  {
    ROS_ERROR("Failed to call service add_two_ints");
    return 1;
  }
return 0;
}
```

**Εξήγηση του κώδικα**

- Ορίζουμε ένα αντικείμενο από την κλάση ServiceClient το οποίο θα χρησιμοποιηθεί αργότερα για να στείλουμε μια αίτηση προς την υπηρεσία add_two_ints.

  **ros::ServiceClient client =n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");**

- Ορίζουμε ένα αντικείμενο από την υπηρεσία που δημιουργήσαμε την ενότητα 2.4 και αποθηκεύουμε σαν αιτήσεις τις δυο τιμές που παίρνουμε από τον χρήση.

- **beginner_tutorials::AddTwoInts srv;**

  **srv.request.a = atoll(argv[1]);**

  **srv.request.b = atoll(argv[2]);**

- Με την συνάρτηση αυτή καλούμε την υπηρεσία, αν ήταν επιτυχής επιστρέφει την τιμή “true” αλλιώς την τιμή “false”.

  **if (client.call(srv))**

Επαναλαμβάνουμε την ίδια διαδικασία και αντιγράφουμε της παρακάτω εντολές ώστε τα δυο cpp αρχεία να γίνουν ορατά απο το ROS

```
add_executable(add_two_ints_service src/add_two_ints_service.cpp)
target_link_libraries(add_two_ints_service ${catkin_LIBRARIES})
add_dependencies(add_two_ints_service beginner_tutorials_gencpp)

add_executable(add_two_ints_client src/add_two_ints_client.cpp)
target_link_libraries(add_two_ints_client ${catkin_LIBRARIES})
add_dependencies(add_two_ints_client beginner_tutorials_gencpp)
```

### 2.10.3 Εκτέλεση υπηρεσίας και πελάτη

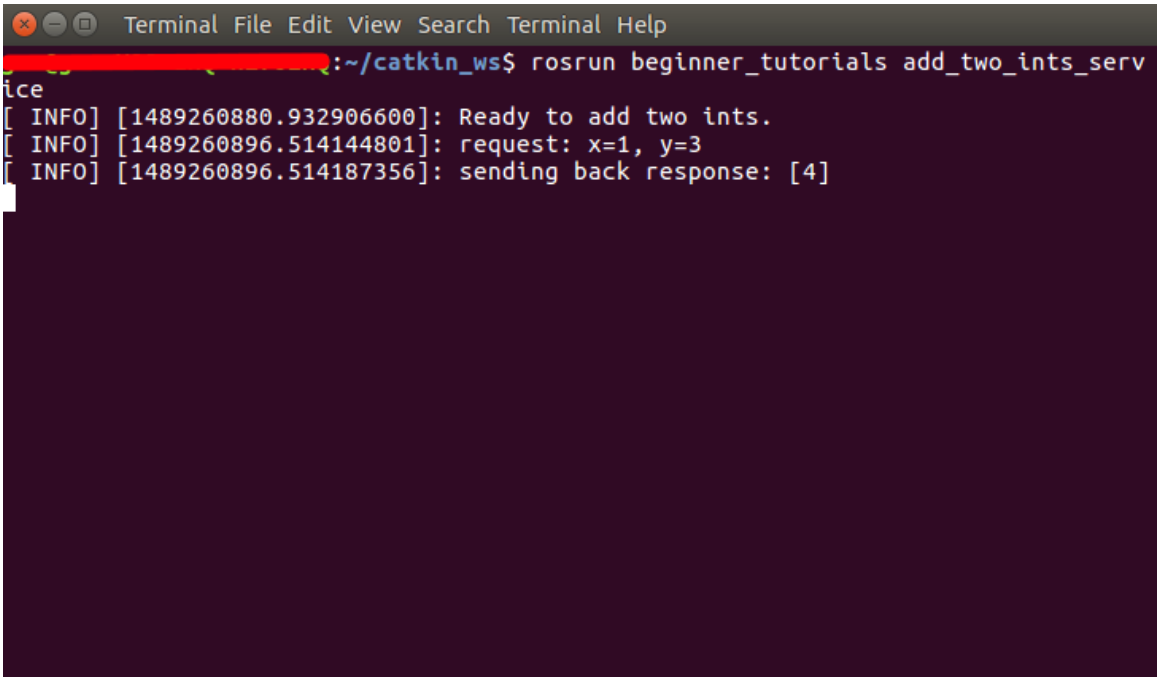
Ανοίγουμε ένα διαφορετικό τερματικό και αφού μεταβούμε στο περιβάλλον εργασίας μας κάνουμε source το αρχείο setup.bas εκτελούμε:

**$ catkin_make**

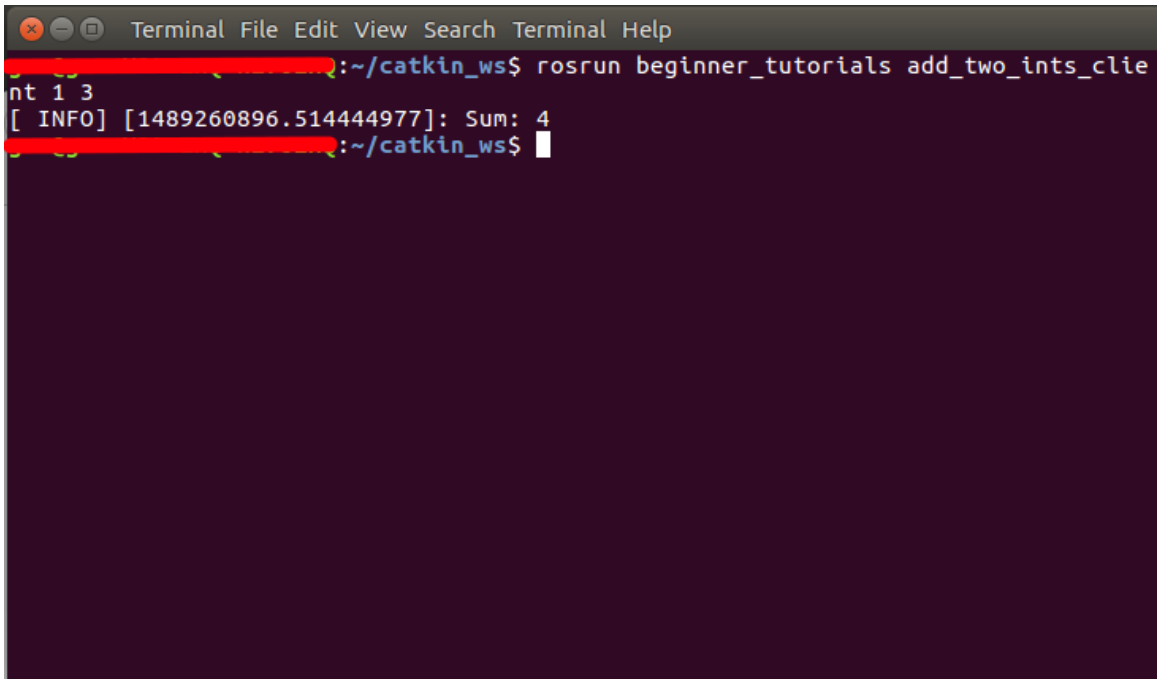
Εφόσον η μεταγλώττιση ήταν επιτυχής ανοίγουμε άλλα δυο τερματικά ακολουθώντας την ίδια διαδικασία και πληκτρολογούμε στο καθένα από αυτά ξεχωριστά:

**$ roscore**
**$ rosrun beginner_tutorials add_two_ints_service**
**$ rosrun beginner_tutorials add_two_ints_client 1 3**

Στο τερματικό της υπηρεσίας θα πρέπει να μας εμφανιστεί το μήνυμα της εικόνας 2.10.1, ενώ στο τερματικό του πελάτη της εικόνας 2.10.2




*Εικόνα 2.10.1 Μήνυμα υπηρεσίας*




*Εικόνα 2.10.2 Μήνυμα πελάτη*

## 2.11 Το τμήμα επικοινωνίας του ROS

Το από την φύση του είναι ένα ανοιχτό λογισμικό. Το ROS έχει δημιουργήσει μια κοινότητα από προγραμματιστές και όχι μόνο που βοηθούν στην συνεχή εξέλιξη του. Κάθε άτομο που χρησιμοποιεί το ROS μπορεί να ανταλλάσσει απόψεις,προγράμματα, θεωρίες και πολλά άλλα με άλλους χρήστες με σκοπό την εξέλιξη και καλύτερη κατανόηση του προγράμματος. Τρεις τρόποι αυτόν είναι:

- **Διανομές.** Οι διανομές του ROS είναι μια σειρά από βιβλιοθήκες έτοιμες προς εγκατάσταση. Οι διανομές αυτές μας βοηθούν στην καλύτερη συνεργασία μεταξύ Linux και ROS όπως επίσης μας παρέχουν πακέτα με σκοπό την καλύτερη κατανόηση του ROS.

- **Αποθήκες κώδικα.** Κάθε προγραμματιστής είτε επαγγελματίας είτε όχι μπορεί να να ανεβάσει ή να πάρει έτοιμα προγράμματα για την αναπτύξει IAV και λειτουργιών του.

- **ROS Wiki.** Είναι ο επίσημος τόπος συζητήσεων του ROS στον οποίο υπάρχουν πληροφορίες, ανανεώσεις ,διορθώσεις και βήμα προς βήμα οδηγίες για το ROS. Επιπλέον καθένας έχει την δυνατότητα να κάνει εγγραφή είτε για να προσφέρει την λύση για κάποια ερώτηση είτε να ρωτήσει κάτι ο ίδιος.

**Σημείωση:** Τα κεφάλαια 1 και 2 αποτελούν έναν Ελληνικό οδηγό κατανόησης του ROS και προέρχονται από http://wiki.ros.org/kinetic/Installation/Ubuntu, http://wiki.ros.org/ROS/Tutorials, σύγγραμα Learning ROS for Robotics Programming, Οι παραπάνω πηγές αναφέρονται και στην βιβλιογραφία.

## Κεφάλαιο 3 Δημιουργία και προσομοίωση ενός IAV

### 3.1 Πρόγραμμα προσομοίωσης

**Gazebo**

Το gazebo είναι ένα πρόγραμμα ανοιχτού λογισμικού που μας επιτρέπει να αναπαριστούμε όχι μόνο IAVs αλλά και το τρισδιάστατο περιβάλλον στο οποίο βρίσκεται. Το gazebo έχει μια μεγάλη βάση δεδομένων η οποία περιέχει έτοιμα μοντέλα τα οποία μπορούμε να χρησιμοποιήσουμε για να δημιουργήσαμε έναν περιβάλλοντα χώρο για το IAVs μας ο οποίος θα μας επιτρέπει να αξιοποιήσουμε όσο το δυνατόν περισσότερο και καλύτερα τις δυνατότητες του. Ένα από τα προτερήματα του gazebo είναι η αναπαράσταση διάφορων τύπων εδάφους (άσφαλτος,χώμα,ανηφόρες) όπως επίσης και η δυνατότητα να χρησιμοποιήσουμε παραμέτρους όπως είναι το φως, το σκοτάδι και η βαρύτητα. Στο ROS εμπεριέχονται βιβλιοθήκες του gazebo έτσι η συνεργασία των δύο προγραμμάτων είναι πάρα πολύ εύκολη. Τέλος έχουμε την δυνατότητα να εισάγουμε διάφορα μοντέλα χρησιμοποιώντας προγράμματα όπως το Blender, sketch up ή ακόμα και να κατεβάσουμε έτοιμα μοντέλα από το google 3D Warehouse και άλλες παρόμοιες τοποθεσίες.

**Rviz**

Το rviz είναι ένα πρόγραμμα προσομοίωσης των δυναμικών δεδομένων του συστήματος. Μας παρουσιάζει το περιβάλλον στο οποίο βρίσκεται οποίο βρίσκεται και αλληλεπιδρά το IAV, βάσει των αισθητήρων του. Επιπλέον έχουμε την δυνατότητα να δούμε και να τροποποιήσουμε τα κινούμενα μηχανικά μέλη ενός IAV όπως για παράδειγμα οι ρόδες του ή κάποιος βραχίονας. Τέλος με την βοήθειά του rviz μπορούμε να έχουμε σε δυναμική απεικόνιση την τρέχουσα εικόνα του περιβάλλοντος προσομοίωσης κατά τη διάρκεια της χαρτογράφησης.

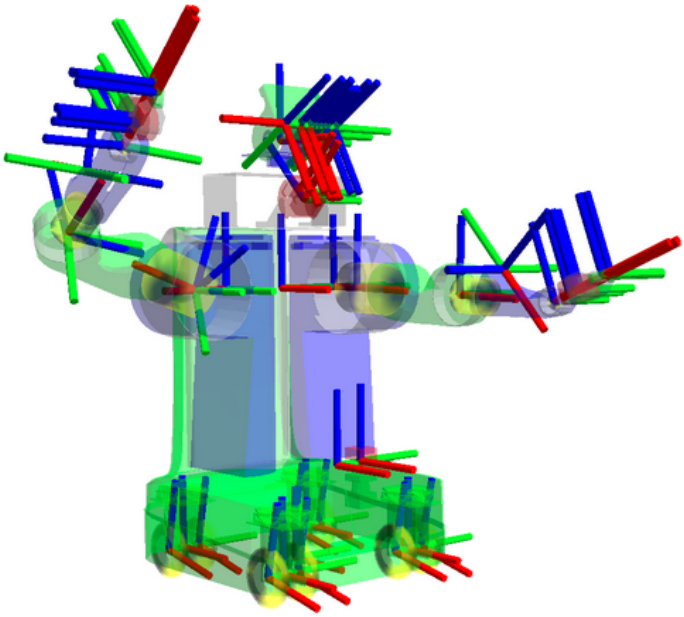
#### 3.1.2 Εγκατάσταση πακέτων

Το ROS για την καλύτερη συνεργασία του με το gazebo και το rviz μας παρέχει έναν μεγάλο αριθμό διαθέσιμων πακέτων. Ανοίγουμε ένα τερματικό και εγκαθιστούμε τα παρακάτω πακέτα:

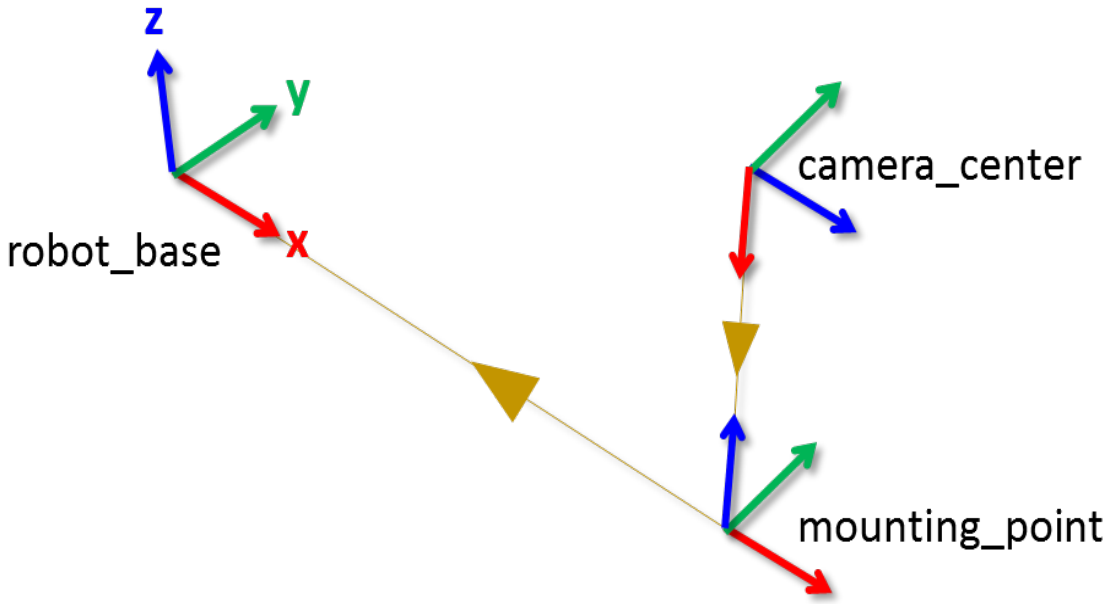
- **ros-kinetic-gazebo-msgs**
- **ros-kinetic-gazebo-plugins**
- **ros-kinetic-gazebo-ros**
- **ros-kinetic-gazebo-ros-control**
- **ros-kinetic-gazebo-ros-pkgs**
- **ros-kinetic-object-recognition-ros-visualization**
- **ros-kinetic-rviz**
- **ros-kinetic-rviz-imu-plugin**
- **ros-kinetic-rviz-plugin-tutorials**
- **ros-kinetic-rviz-python-tutorial**
- **ros-kinetic-rviz-visual-tools**

### 3.2 Μετασχηματισμός συντεταγμένων

Ο μετασχηματισμός συντεταγμένων(transform frame ή tf) είναι ένα εργαλείο του ROS το οποίο μας δίνει την δυνατότητα να αποθηκεύουμε και να επεξεργαζόμαστε πολλαπλά στατικά συστήματα συντεταγμένων ταυτόχρονα σε μία δενδρική δομή. Έχοντας όλες τις συντεταγμένες και τους μετασχηματισμούς των μελών ενός IAV σε σχέση με ένα σταθερό αντικείμενο για μια συγκεκριμένη χρονική στιγμή, μας δίνει την δυνατότητα να μετασχηματίσουμε νέα σημεία ανάμεσα σε σταθερά και κινούμενα μέλη. Για παράδειγμα αν γνωρίζουμε τον μετασχηματισμό ανάμεσα στις συντεταγμένες ενός εμποδίου (που θεωρείται σταθερό σημείο σε έναν χάρτη) και της ρόδας ενός IAV(που θεωρείται κινητό μέλος), επιπλέον γνωρίζουμε τον μετασχηματισμό των συντεταγμένων της ρόδας του IAV με την κάμερα του IAV(που είναι επίσης κινητό μέλος), είμαστε σε θέση να γνωρίζουμε τον μετασχηματισμό ανάμεσα στο εμπόδιο και στην κάμερα του IAV για κάθε χρονική στιγμή. Έτσι μας δύνεται η δυνατότητα να γνωρίζουμε την θέση του IAV και των μελών του στον χάρτη κάθε διαφορετική χρονική στιγμή. Η απεικόνιση του tf φαίνεται στις εικόνες 3.2.1 και 3.2.1



Εικόνα 3.2.1 Απεικόνιση του tf πάνω στο PR2(Personal Robot 2), πηγή (5) Tully Foote, tf: The Transform




Εικόνα 3.2.2 Μετασχηματισμός ανάμεσα σε σημεία, πηγή https://www.mathworks.com/help/robotics/examples/access-the-tf-transformation-tree-in-ros.html

Τέλος για την δημιουργία του δέντρου με τις στατικές συντεταγμένες πρέπει να ορίσουμε δύο κατηγορίες κόμβων

- **Parent node**. Ένας κόμβος που με βάση τις συντεταγμένες του ορίζονται όλοι οι μετασχηματισμοί.
- **Child node**. Όλοι οι υπόλοιποι κόμβοι που οι στατικές συντεταγμένες τους μετασχηματίζονται με τον parent node.

### 3.3 Δημιουργία ενός IAV

Υπάρχουν δυο βασικές κατηγορίες αρχείων με τις οποίες δημιουργούμε την μορφή του IAV, urdf και xarco.Τα αρχεία με κατάληξη urdf ( Unified Robot Description Format) είναι σε xml διαμόρφωση στα οποία ορίζονται τα κινούμενα και μη μέλη(χρησιμοποιώντας αρθρώσεις), τα χαρακτηριστικά του όπως το μέγεθος, χώμα, βάρος ,πλήθος αισθητήρων. Το μειονέκτημα των αρχείων αυτών είναι ότι χρειάζεται μεγάλος αριθμός εντολών για να αναπαραστήσουμε μαθηματικές πράξεις συνεπώς ο όγκος των εντολών που περιέχονται είναι πολύ μεγάλος και καθιστά δύσκολη την συγγραφή και την ανάγνωση τους. Για αυτούς τους σκοπούς δημιουργήθηκε ο τύπος ο τύπος αρχείων με κατάληξη xacro. Μέσω των αρχείων αυτών μπορούμε να αναπαραστήσουμε ένα αρχείο urdf καθώς και μαθηματικές πράξεις χρησιμοποιώντας μαρκο εντολές, συνεπώς το πλήθος των εντολών μειώνεται. Το IAV θα έχει την κατάληξη xarco για ευκολία.Αρχικά θα δημιουργήσουμε ένα πακέτο με την ονομασία mvrobot_description με τις εξής εξαρτήσεις, geometry_msgs, roscpp, rviz, tf, urdf, xacro. Επιπλέον τους φακέλους urdf, meshes, launch και rviz για μετέπειτα χρήση. Στον φάκελο urdf δημιουργούμε ένα αρχείο με όνομα mvrobot.xacro που εμπεριέχει τον κώδικα του IAV.

Ο Κώδικας για την δημιουργία του IAV αποτελείται από τρία βασικά μέρη. Στο πρώτο δηλώνουμε το όνομα, τις σταθερές που θα χρησιμοποιήσουμε,για παράδειγμα τις διαστάσεις από την βάση και τις ρόδες του IAV.

Στο δεύτερο μέρος δηλώνονται ξεχωριστά τα μέλη(βάση,ρόδες αισθητήρες) και οι εξαρτήσεις τους όπως επίσης συνδέσεις τους με το gazebo. Στο τελευταίο μέρος του κωδικά ορίζεται ο τρόπος με τον οποίον θα κινητέ το IAV καθώς και το θέμα στο οποίο θα δέχεται τα δεδομένα για την κίνηση.
Στον κώδικα του IAV δε δηλώσαμε μόνο τα μέρη του και τον αισθητήρα, αλλά τους δώσαμε και κάποιες ιδιότητες όπως οντότητα έτσι ώστε να έχουν βάρος και να αλληλεπιδρούν με την βαρύτητα, δημιουργήσαμε τις αρθρώσεις των ροδών έτσι ώστε να μπορούν να κινούνται, όπως επίσης και tf από την βάση του IAV προς όλα τα υπόλοιπα μέλη του.
Τέλος γίνεται η εισαγωγή του αρχείου weel.xacro στον ίδιο φάκελο όπου τοποθετούμε τα χαρακτηριστικά από τις δύο ρόδες.

Τέλος αναζητούμε από το διαδίκτυο το αρχείο hokuyo.dae και το τοποθετούμε στο φάκελο mashes. Το αρχείο αυτό περιέχει τα χαρακτηρίστηκα του αισθητήρα laser και μας το παρέχει η εταιρία του δωρεάν.

## 3.4 Προσομοίωση και έλεγχος του IAV

### 3.4.1 Προσομοίωση

Έχοντας τελειώσει με την δημιουργία του αρχείου του IAV πρέπει να δημιουργήσουμε ένα εκτελέσιμο αρχείο έτσι ώστε να εισάγουμε και να διαχειριζόμαστε το IAV μέσω του gazebo. Αρχικά δημιουργούμε ένα νέο πακέτο με όνομα mvrobot_gazebo με τις εξής εξαρτήσεις, gazebo_msgs, gazebo, plugins, gazebo_ros, gazebo_ros_control, mvrobot_description. Στο πακέτο αυτό δημιουργούμε ένα φάκελο με το όνομα launch. Μέσα σε αυτόν τον φάκελο θα τοποθετήσουμε ένα εκτελέσιμο αρχείο με το οποίο θα εκτελούμε το gazebo μαζί την εισαγωγή του IAV σε έναν κενό κόσμο. Το αρχείο αυτό ονομάζεται mvrobot_world.launch.

### 3.4.2 Έλεγχος

Για τον έλεγχο του IAV θα χρησιμοποιηθεί ο εγγενής κώδικας του ROS για τον έλεγχο και την κίνηση του turtlebot. Για αυτόν τον λόγο πρέπει να εγκαταστήσουμε το παρακάτω πακέτο, πληκτρολογώντας:

**$s udo apt-get install ros-kinetic-turtlebot-teleop**

Επίσης δημιουργούμε ένα νέο πακέτο με το όνομα mvrobot_navigation με τις εξής εξαρτήσεις, controller_manager, geometry_msgsjoint_state_controller, joy, roscpp, rospy, sensor_msgs, std_msgs. Μέσω αυτού του πακέτου θα χειριζόμαστε το IAV. Στο πακέτο αυτό δημιουργούμε ένα φάκελο με το όνομα launch και μέσα του τοποθετούμε το αρχείο mvrobot_teleop.launch.

## 3.5 Εκτέλεση προσομοίωσης και ελέγχου

### 3.5.1 Εκτέλεση προσομοίωσης

Ανοίγουμε ένα νέο τερματικό, μεταφερόμαστε στο περιβάλλον εργασίας μας και εκτελούμε την εντολή:

$ **catkin_make**

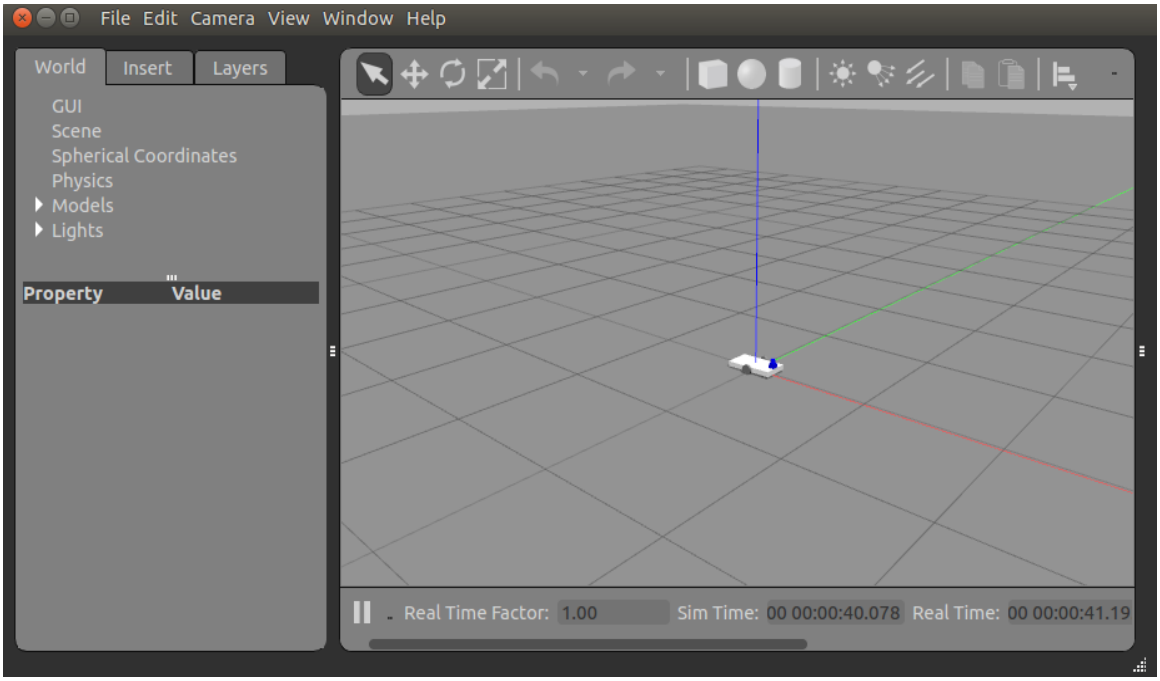
Εφόσον εκτελέστηκε με επιτυχία, στη συνέχεια εκτελούμε την εντολή:

**$ source devel/setup.bash**

Τέλος εκτελούμε το αρχείο mvrobot_world.launch με την χρήση της εντολής roslaunch η οποία δέχεται σαν όρισμα το όνομα του πακέτου στο οποίο βρίσκεται το εκτελέσιμο αρχείο και το όνομα του αρχείου. Συνεπώς πληκτρολογούμε:

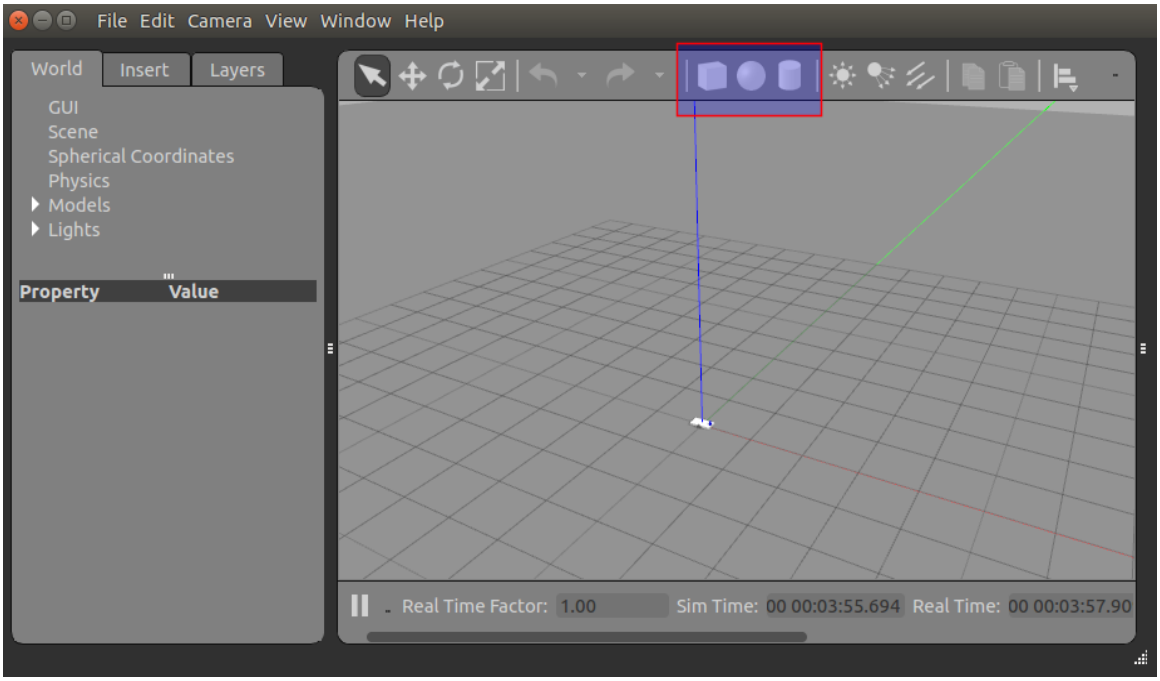
$ **roslaunch mvrobot_gazebo mvrobot_world.launch**

Στην περίπτωση που η διαδικασία ήταν επιτυχής θα εμφανιστεί το παράθυρο της εικόνας 3.5.1



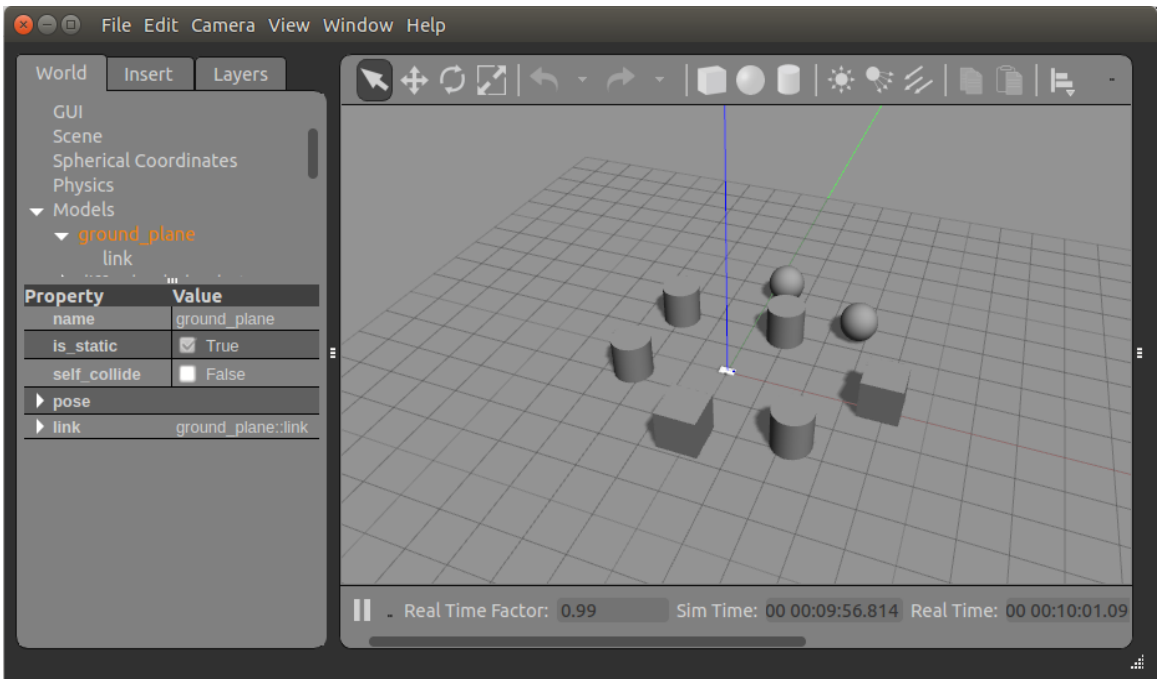

*Εικόνα 3.5.1 Προσομοίωση του IAV*

Όπως παρατηρούμε στην εικόνα 3.5.2 μπορούμε να εισάγουμε διάφορα αντικείμενα στην προσομοίωση που θα λειτουργούν σαν εμπόδια στο IAV




*Εικόνα 3.5.2 Τρόπος εισαγωγής εμποδίων*

Μετά από μια τυχαία εισαγωγή αυτών των τριών εμποδίων η προσομοίωση συμπίπτει περίπου με την εικόνα 3.5.3



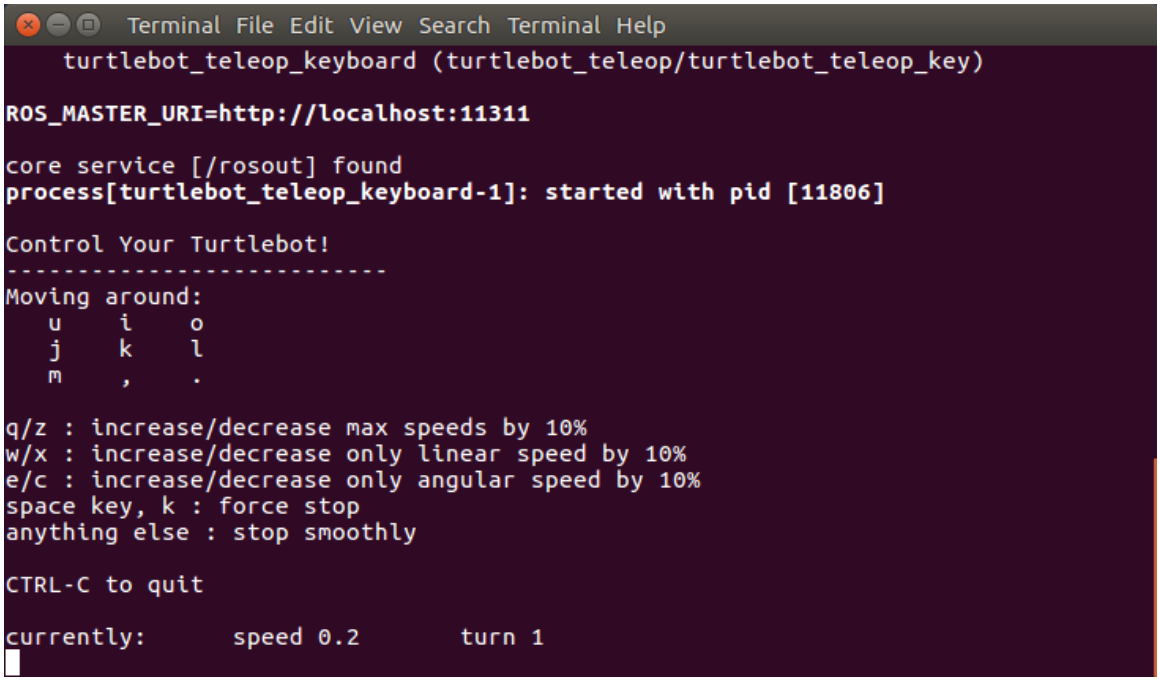

*Εικόνα 3.5.3 Εισαγωγή εμποδίων*

### 3.5.2 Έλεγχος του IAV

Ανοίγοντας ένα νέο τερματικό χωρίς όμως να έχουμε κλείσει το προηγούμενο, μεταφερόμαστε στο περιβάλλον εργασίας και εκτελούμε τις παρακάτω εντολές

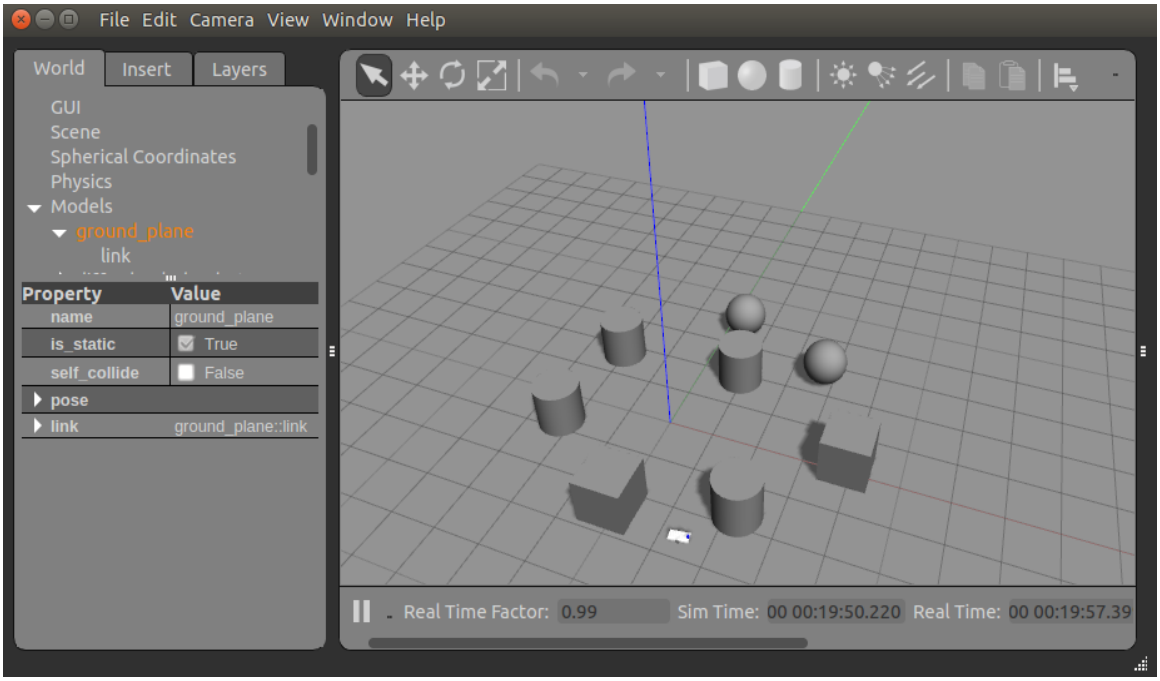
**$ source devel/setup,bash**
**$ roslaunch mvrobot_navigation mvrobot_teleop.launch**

Το τερματικό μας θα πρέπει να μοιάζει με την εικόνα 3.5.4




*Εικόνα 3.5.4 Τηλεχειρισμός του IAV*

Έχοντας επιλεγμένο τον τερματικό του τηλεχειρισμού και πληκτρολογώντας τα αναγραφόμενα πλήκτρα μπορούμε να μετακινήσουμε το IAV σαν την εικόνα 3.5.5.




*Εικόνα 5.5.5 Κίνηση του IAV*

**Σημείωση**:Οι κώδικές και τα αρχεία που αναφέρονται στο Κεφάλαιο 3 εμπεριέχονται στο Παράρτημα Α.

## Κεφάλαιο 4 Navigation Stack

Το Navigation Stack περιέχει έναν μεγάλο αριθμό πακέτων που έχουν ως στόχο την κίνηση του IAV από ένα αρχικό σε ένα τελικό σημείο. Μέσα σε αυτά τα πακέτα υπάρχουν αλγόριθμοι και τεχνικές αποφυγής εμποδίων, εύρεση βέλτιστης διαδρομής, χαρτογράφησης του χώρου και πολλά ακόμα με παρόμοιες λειτουργίες που έχουν ως σκοπό την αυτόνομη κίνηση του IAV. Μερικοί γνωστοί αλγόριθμοι και τεχνικές είναι :

- SLAM(Simulation Localization And Mapping)
- A*
- Dijkstra
- AMCL(adaptive Monte Carlo Localization)

Για να λειτουργήσει το Navigation Stack και να μας παρέχει τα ζητούμενα αποτελέσματα χρειάζεται δυο ειδών εισόδους. Πρώτον πληροφορίες που προέρχονται από αισθητήρες του IAV και σχετίζονται με την θέση και την ταχύτητα του στον χώρο καθώς και την αναγνώριση κοντινών εμποδίων ή αντικειμένων. Δεύτερον απαιτεί από τον χρήση τον ορισμό ενός στόχου-τελικού σημείου στον οποίο θα αποστέλλονται συγκεκριμένοι τύποι μηνυμάτων (geometry_msgs/PoseStamped).Προκείμενου όμως να πραγματοποιηθούν τα παραπάνω το IAV πρέπει να πληρεί κάποιες προϋποθέσεις που αφορούν εξαρτήματα του. Βασική προϋπόθεση είναι η ύπαρξη αισθητήρων για την εύρεση της τοποθεσίας του IAV στο χάρτη, της ταχύτητας (accelerometer-IMU GPS) κοντινών εμποδίων και της χαρτογράφησης χώρου (laser range finder sensor, sonar sensor, camera sensor).

### 4.1 Κατανόηση του Navigation Stack

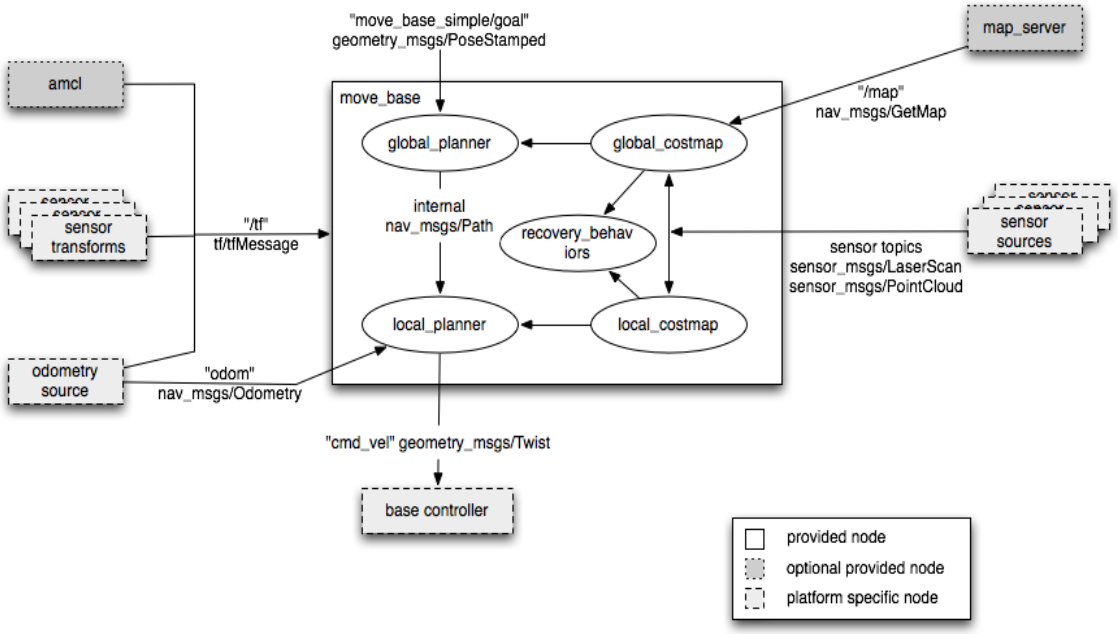
#### 4.1.1 Εγκατάσταση

Το Navigation Stack δεν εμπεριέχεται στην αρχική εγκατάσταση του ROS συνεπώς θα πρέπει να εγκαταστήσουμε εμείς τα ανάλογα πακέτα και βιβλιοθήκες, Ανοίγοντας ένα νέο τερματικό εκτελούμε:

**$ sudo apt-get install ros-kinetic-navigation**
**$ sudo apt-get install ros-kinetic-navigation-layers**
**$ sudo apt-get install ros-kinetic-navigation-stage**
**$ sudo apt-get install ros-kinetic-navigation-tutorials**

#### 4.1.2 Ανάλυση πακέτων

Στο διάγραμμα 4.1.1 απεικονίζονται όλοι οι τύποι πακέτων από τα οποία αποτελείται το Navigation Stack.




Εικόνα 4.1.1 The Navigation Stack, πηγή Learning ROS for Robotics Programming

Όπως έχουμε ήδη αναφέρει, βασική προϋπόθεση για την λειτουργία του Navigation Stack είναι η ύπαρξη δεδομένων που προέρχονται από αισθητήρες και εισάγονται στον βασικό κόμβο για επεξεργασία και λήψη αποφάσεων.

- **Odometry source**. Μας παρέχει την θέση και την ταχύτητα του IAV με βάση τον εκάστοτε χάρτη, Για την έκδοση μηνυμάτων χρησιμοποιούν τον τύπο nav_msgs/Odometry μέσω του θέματος /odom. Κύριες πηγές αυτών των δεδομένων είναι οι πληροφορίες που προέρχονται από τους κινητήρες των τροχών, IMU-accelerometer, δισδιάστατες ή τρισδιάστατες κάμερες.

- **Sensor Source.** Μας δίνει την δυνατότητα να γνωρίζουμε την θέση και την απόσταση εμποδίων από το IAV.Για την μετάδοση μηνυμάτων χρησιμοποιούν τον τύπο sensor_msgs/LaserScan αν προέρχονται από αισθητήρα laser range finder sensor ή sensor_msgs/PointCloud αν προέρχεται από point cloud δεδομένα τα οποία εκδίδονται από το θέμα /scan. Σε συνεργασία με τα δεδομένα για την θέση και την ταχύτητα του IAV βοηθούν να δημιουργήσουμε το global και το local cost map.

- **Sensor Transform**s. Αυτός ο κόμβος εκδίδει όλες τις σχετικές συντεταγμένες των μελών του IAV στο θέμα /tf χρησιμοποιώντας tf/messages.

Εκτός όμως από τους παραπάνω βασικούς κόμβους υπάρχουν και δυο προαιρετικοί

- **Map_server**. Μας παρέχει την δυνατότητα να να αποθηκεύουμε και να φορτώνουμε τον χάρτη που δημιουργήθηκε με την βοήθεια του πακέτου costmap. Εκδίδει μηνύματα τύπου nav_msgs/GetMap μέσω του θέμα /map.

- **Amcl**. Δέχεται δεδομένα odometry, tf, σάρωσης laser, μας βοηθά στον εντοπισμό του IAV στον χάρτη.

Το κεντρικό πακέτο του Navigation Stack ονομάζεται move_base και εμπεριέχει τον κόμβο move_base_node. Ο κόμβος move_base υπάγεται στην κατηγορία simple action server στους οποίους μπορεί να δοθούν εντολές για κάποια ενέργεια. Στον κόμβο αυτόν λαμβάνονται όλες οι πληροφορίες από τους επιμέρους κόμβους και γίνεται η επεξεργασίας του με σκοπό την κίνηση του IAV από ένα αρχικό ως το τελικό σημείο. Ο κόμβος αυτός προκειμένου να κινήσει το IAV χρειάζεται από τον χρήστη να εισάγει ένα σημείο- τελικό στόχο για αυτό. Η λειτουργία αυτή λαμβάνει χώρα στέλνοντας μηνύματα με τύπο geometry_msgs/PoseStamped μέσω του θέματος /move_base_simple/goal.

Για την επεξεργασία των πληροφοριών αυτών το Navigation Stack μας παρέχει τα παρακάτω πακέτα:

- **Global Planner**. Όταν ο χρήστης εισάγει τον τελικό στόχο τα δεδομένα αρχικά στέλνονται στο πακέτο αυτό. Χρησιμοποιώντας αλγορίθμους όπως A* και Dijkstra, το πακέτο αυτό υπολογίζει την βέλτιστη διαδρομή από το τωρινό σημείο του IAV ως τον τελικό στόχο. Το πακέτο αυτό δέχεται ως επιπλέον είσοδο odom και tf μηνύματα καθώς και πληροφορίες από τον αισθητήρα laser.

- **Local Planner.** Μετά την εύρεση της βέλτιστης διαδρομής τα δεδομένα αυτά εκδίδονται στο πακέτο local_panner. Η κύρια λειτουργία του πακέτου αυτού είναι η οδήγηση του IAV από την τωρινή του τοποθεσία στην πλησιέστερη τοποθεσία που έχει οριστεί ως βέλτιστη διαδρομή από το global planner. Προϋπόθεση για την σωστή διεκπεραίωση αυτής τις εντολής είναι η επικοινωνία του πακέτου αυτού τους αισθητήρες του IAV.

- **Rotate Recovery.** Στην περίπτωση που το IAV είναι πολύ κοντά σε ένα εμπόδιο ή στην περίπτωση που το local planner δε μπορεί να εντοπίσει το global planner ενεργοποιείται αυτό το πακέτο και εκτελείται μια περιστροφή 360 μοιρών μέχρι το IAV να ξεπεράσει το εμπόδιο ή το local planner ναι εντοπίσει τη διαδρομή που πρέπει να ακολουθήσει.

- **Costamp 2D.** Το πακέτο αυτό είναι υπεύθυνο για την δημιουργία του χάρτη στον οποίο θα κινείται αυτόνομα το IAV. Το πακέτο αυτό χωρίζεται σε δύο υποκατηγορίες, το γενικό καθώς και τον τοπικό χάρτη βάρους(costmap) που είναι υπεύθυνοι για την δημιουργία του χάρτη σε μεγάλη και σε μικρή κλίμακα αντίστοιχα. Ο γενικός χάρτης βάρους δημιουργεί την βέλτιστη διαδρομή αποφεύγοντας εμπόδια στο γενικό κομμάτι του χάρτη ενώ ο τοπικός χάρτης βάρους στο κομμάτι του χάρτη που βρίσκεται κοντά στο IAV. Λαμβάνοντας πληροφορίες από τους αισθητήρες του IAV σχετικά με την θέση του στον χάρτη και την θέση των εμποδίων που υπάρχουν μέσα σε αυτόν, ο χάρτης βάρους δημιουργεί είτε δισδιάστατους είτε τρισδιάστατους χάρτες οι οποίοι αποτελούνται από πλέγματα. Το κάθε πλέγμα από αυτά παίρνει διαφορετική τιμή ανάλογα με την κατάσταση του. Υπάρχουν τρεις βασικές κατηγορίες για την κατάσταση του κελιού, εμπόδιο, ελεύθερος χώρος, ανεξερεύνητος χώρος.

Τέλος μετά την επεξεργασία που λαμβάνει χώρα στον κόμβο move_base εκδίδει μηνύματα τύπου geometry_msgs/Twist μέσω του θέματος /cmd_vel. Σε αυτό το θέμα εγγράφεται ο κόμβος base_controller. Ο κόμβος αυτός είναι είναι υπεύθυνος για την μετατροπή αυτών τον σε δεδομένα συμβατά με την τον τύπο δεδομένων που χρειάζεται ως είσοδο ο κινητήρας του IAV.

## 4.2 SLAM

### 4.2.1 Αρχή λειτουργίας

Έχοντας κατανοήσει τον ρόλο και την λειτουργία του των πακέτων που εμπεριέχονται στο Navigation Stack το επόμενο βήμα είναι ο ορισμός των κόμβων που είναι υπεύθυνοι για την χαρτογράφηση του χώρου.
Η λειτουργία αυτή ονομάζεται SLAM (Simultaneous Localization and Mapping). Με την χρήση του SLAM έχουμε την δυνατότητα να δημιουργήσουμε έναν δισδιάστατο χάρτη με πλέγματα όπου το κάθε κελί του πλέγματος έχει διαφορετική τιμή ανάλογα με την φυσική υπόσταση του σημείου(ελεύθερος χώρος, εμπόδιο, άγνωστη περιοχή). Ως εισόδους χρησιμοποιεί δεδομένα από τον αισθητήρα laser που κατά προτίμηση τοποθετείται σε οριζόντια θέση την κορυφή του IAV, όπως επίσης και odometry δεδομένα.
Τα τελευταία χρόνια έχουν πραγματοποιηθεί ραγδαίες εξελίξεις στον τομέα της χαρτογράφησης ως συνέπεια έχουν αναπτυχθεί μεθοδολογίες και αλγόριθμοι για τον συγκεκριμένο σκοπό. Δυο από τα κύρια προβλήματα που προέκυψαν είναι τα μεγάλα ποσοστά θορύβου και ο μεγάλος όγκος των δεδομένων που πρέπει να επεξεργαστούν. Για την μείωση αυτών προβλημάτων άρχισαν να αναπτύσσονται τεχνικές βασισμένες σε μεθόδους δειγματοληψίας και νόμους πιθανοτήτων. Ένας πολύ βασικό θεώρημα πιθανοτήτων στον οποίο στηρίζονται οι μέθοδοι αυτοί είναι το θεώρημα της υπό συνθήκη πιθανότητας ή το θεώρημα Bayes.

Πάνω σε αυτό το θεώρημα βασίστηκαν οι δύο πιο βασικές μέθοδοι χαρτογράφησης. Η πρώτη μέθοδος πραγματοποιείται με την χρήση του φίλτρου Kalman.

Το φίλτρο Kalman είναι ένας βέλτιστος γραμμικός ελεγκτής που χρησιμοποιείται σε συνεχή συστήματα. Δύο από τα κύρια προτερήματα του είναι τα πολύ ικανοποιητικά ποσοστά μείωσης θορύβου όπως επίσης και η δυνατότητα πρόβλεψης μελλοντικών τιμών του συστήματος λαμβάνοντας υπόψη την τωρινή αλλά και παλαιότερες τιμές του. Ένα πολύ σοβαρό μειονέκτημα είναι το γεγονός ότι το φίλτρο Kalman απευθύνεται σε γραμμικά συστήματα. Όμως υπάρχουν πολλές περιπτώσεις που οι καταστάσεις σε ένα σύστημα που περιλαμβάνει IAV που περιγράφονται από μη γραμμικά συστήματα. Για αυτές τις περιπτώσεις έχει αναπτυχθεί το επεκτεινόμενο φίλτρο Kalman ή EKF(Extended Kalman Filter).

Η δεύτερη μέθοδος ονομάζεται φίλτρο σωματιδίων ή αλλιώς PF(Particle filter).Το PF έχει αρκετές ομοιότητες με το φίλτρο EKF με την διαφορά ότι ο υπολογισμός της μελλοντικής κατάστασης στηρίζεται αποκλειστικά στην τωρινή. Αρχικά υποθέτουμε πιθανές μελλοντικές τιμές στο σύστημα μας. Στη συνέχεια βάσει τωρινών δεδομένων του συστήματος αντιστοιχίζεται ένας δείκτης βαρύτητας για την κάθε τιμή. Τέλος μετά από μια διαδικασία δειγματοληψίας παραμένουν οι τιμές με τον μεγαλύτερο δείκτη,και η κοντινότερη τιμή στα τωρινά δεδομένα θα είναι και η μελλοντική. Είναι σημαντικό να αναφέρουμε ότι το PF προσεγγίζει με μεγαλύτερη ευκολία τις τιμές σε σχέση με το EKF, επίσης έχει καλύτερη αντιμετώπιση σε θορύβους που δεν ακολουθούν κατανομή Gauss. Ένα ελαττώματα του διαθέτει η μέθοδος αυτή πάνω στο οποίο έχουν πραγματοποιηθεί έρευνες για την μείωση του είναι η εξάλειψη σωματιδίων με μεγάλο δείκτη βαρύτητας. Το φαινόμενο αυτό παρατηρείται κατά την διάρκεια της δειγματοληψίας διότι αλλάζει τιμή ο δείκτης βαρύτητας.

Βάσει των δύο παραπάνω μεθόδων το ROS έχει αναπτύξει πέντε αλγόριθμους για την χαρτογράφηση ενός χώρου.

- **Core slam.** Ο αλγόριθμός αυτός είναι μικρός σε έκταση και έχει σχεδιαστεί ώστε να είναι εύκολος στην κατανόηση, Τα αποτελέσματα που μας προσφέρει συνήθως δεν είναι ικανοποιητικά. Ο υπολογισμός της απόστασης μεταξύ των εμποδίων και του IAV γίνεται με την χρήση του PF. Στις περισσότερες περιπτώσεις αποφεύγεται η χρήση αυτού το αλγορίθμου διότι έχουν παρατηρηθεί προβλήματα στον υπολογισμό της θέσης είτε του IAV είτε τον εμποδίων.

- **Gmapping.** Είναι η πιο διαδεδομένος τρόπος χαρτογράφησης που προσφέρει το ROS. Βασίζεται στην μεθοδολογία PF και χρησιμοποιεί laser range finder για την αναγνώριση αντικειμένων και για την δημιουργία του χάρτη, καθώς και IMU-accelerometer για τον εντοπισμό του IAV. Ένα από τα ελαττώματα του είναι ότι προκειμένου να έχουμε ικανοποιητικά αποτελέσματα χρειαζόμαστε μεγάλο αριθμό σωματιδίων(particles)

- **Hector Slam.** Η ιδιαιτερότητα αυτού του αλγορίθμου είναι ότι χρησιμοποιεί για τον εντοπισμό του IAV δεδομένα από έναν laser αισθητήρα με μεγάλη συχνότητα σάρωσης και απο IMU-accelerometer χωρίς να εκδίδει odom δεδομένα. Για να επιτευχθεί το συγκεκριμένο εγχείρημα χρησιμοποιείται το EKF. Αυτή του η ιδιαιτερότητα τον καθιστά καταλληλότερο όταν πρόκειται για εναέρια ή σε ανώμαλο έδαφος χαρτογράφηση

- **Karto slam.**Ο συγκεκριμένος αλγόριθμος αυτός έχει παρόμοια αρχιτεκτονική με του ROS. Κάθε κόμβος(node) αναπαριστά την θέση και την εικόνα του IAV στο χάρτη καθώς και τις πληροφορίες των αισθητήρων την συγκεκριμένη χρονική στιγμή.Οι κόμβοι συνδέονται μεταξύ τους μέσω των ακμών(arcs) που προκύπτουν από την μετακίνηση του IAV στον χώρο. Για την λειτουργία του χρησιμοποιεί ανάλυση Cholesky για γραμμικά συστήματα. Ο αλγόριθμος αυτός συνιστάται για χώρους με μεγάλη έκταση.

- **Lago slam.** Συντομογραφία από linear approximation for graph optimization, έχει επίσης παρόμοια αρχιτεκτονική με το ROS αλλά σε απλούστερη μορφή. Πρόκειται έναν αλγόριθμό ελαχιστοποίησης μη γραμμικών στοιχείων με χρήση γραμμικών μεθόδων, η επανάληψη αυτή πραγματοποιείται μέχρι να βρεθεί το τελικό ελάχιστο κόστος κίνησης. Ένα από τα προτερήματα του είναι ότι δεν απαιτεί αρχικές τυχαίες εκτιμώμενες τιμές.

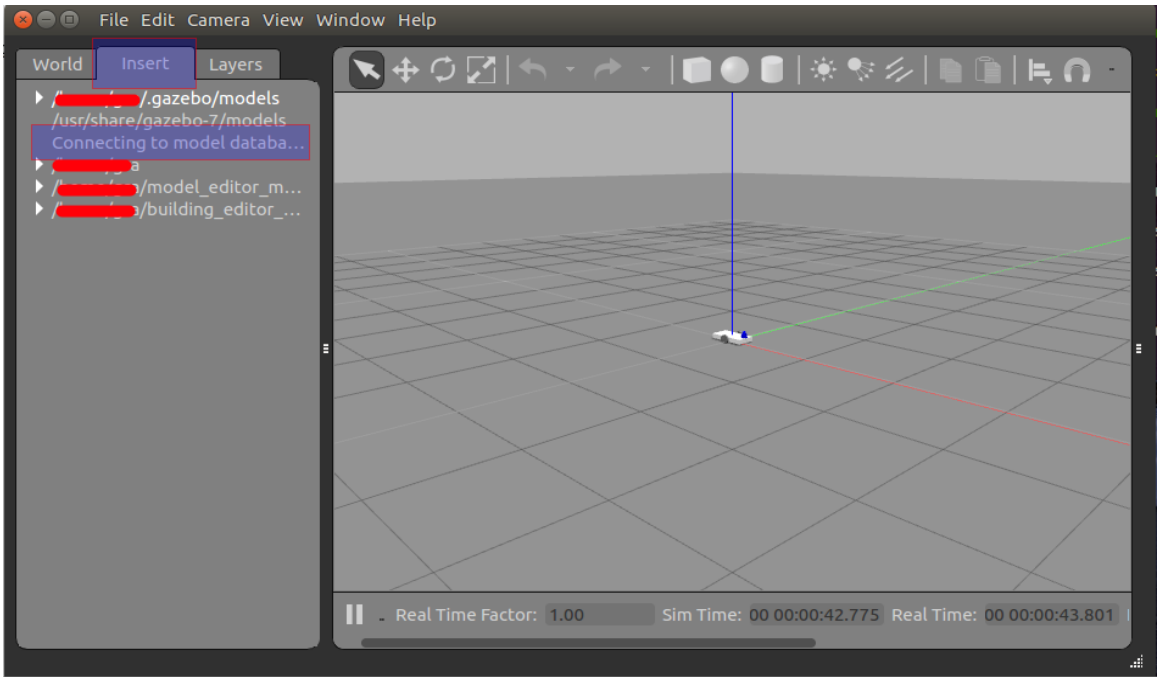
Οι παραπάνω ορισμοί και στοιχεία είναι βασισμένα στην έρευνα που πραγματοποιήθηκε από τους João Machado Santos, David Portugal and P. Rocha και αναφέρονται στην βιβλιογραφία. Βάσει των συμπερασμάτων αυτής της έρευνας οι τρεις επικρατέστερες μέθοδοι χαρτογράφησης είναι οι Karto slam, Hectro slam και Gmapping. Στην εργασία αυτή πραγματοποιείται η χαρτογράφηση με την μέθοδο Gmapping.

### 4.2.2 Ορισμός και ετοιμασία αρχείων για χαρτογράφηση

Στο προηγούμενο κεφάλαιο δημιουργήσαμε το IAV και τοποθετήσαμε πάνω σε αυτό τον αισθητήρα laser, Σε αυτό το κεφάλαιο θα δημιουργήσουμε τα αρχεία που είναι απαραίτητα για να πραγματοποιηθεί η διαδικασία της χαρτογράφησης. Στο ROS έχει ήδη δημιουργηθεί ο κώδικας για αυτό το εγχείρημα μέσω των πακέτων που εγκαταστήσαμε στην αρχή του κεφαλαίου, το μόνο που απομένει είναι να ορίσουμε της παραμέτρους που χρειαζόμαστε. Αρχικά δημιουργούμε ένα νέο πακέτο στο περιβάλλον εργασίας μας με το όνομα mvrobot_navigation χωρίς εξαρτήσεις. Στη συνέχεια δημιουργούμε έναν φάκελο με το όνομα launch μέσα στο πακέτο μας στο οποίο εμπεριέχεται το αρχείο gmapping.launch.
Στο αρχείο gmapping.launch ορίζουμε τον κόμβο gmapping τις εισόδους του, δηλαδή σε ποια θέματα πρέπει να εγγραφεί έτσι ώστε να λάβει τις πληροφορίες που χρειάζεται και τον ρυθμό με τον οποίο θα τα λαμβάνει, όπως και τις εξόδους του ,δηλαδή ποια θέματα θα εκδίδει. Στο τέλος του αρχείου έχουμε δηλώνουμε κάποια αρχεία που εμπεριέχουν παραμέτρους των πακέτων του Navigation Stack τις οποίες θα δηλώσουμε στη συνέχεια.
Δημιουργούμε μέσα στο πακέτο μας έναν φάκελο με το όνομα param και μέσα του ορίζουμε τα εξής αρχεία. base_local_planner_params.yaml, costmap_common_params.yaml, dwa_local_planner_params.yaml, global_costmap_params.yaml, global_costmap_params.yaml, move_base_params.yaml.Στις παραμέτρους που μας παρέχει το ROS έχουν υποστεί κάποιες αλλαγές για την καλύτερη λειτουργία του Navigation Stack.

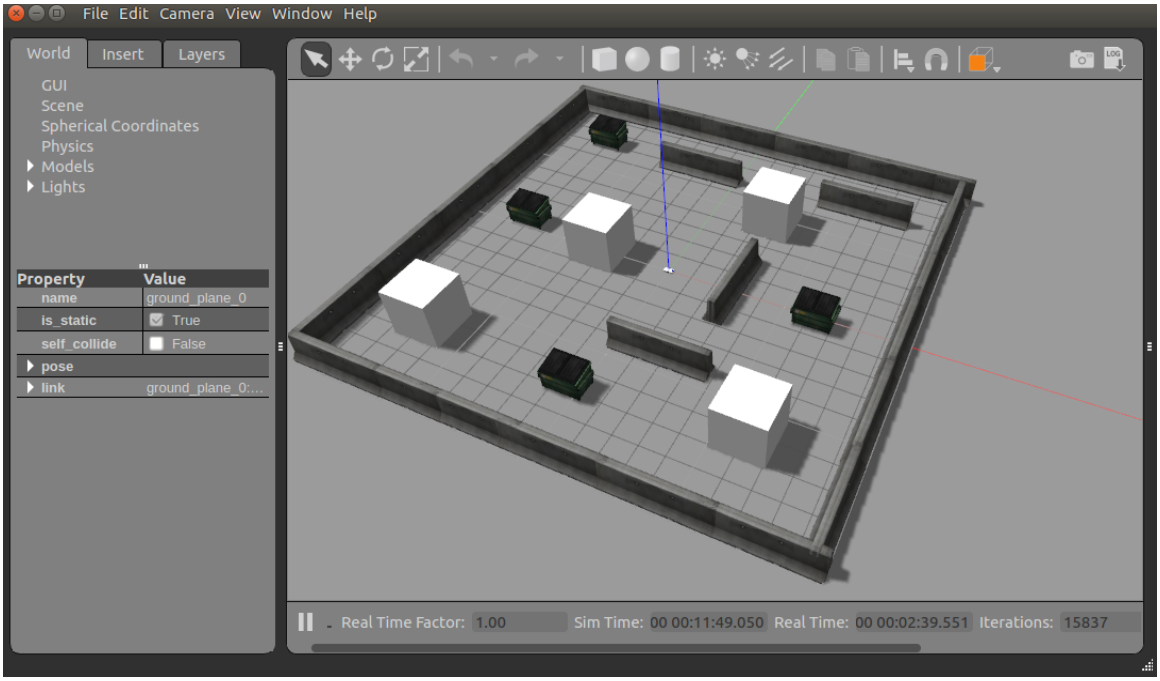
### 4.2.3 Ορισμός του μοντέλου του χάρτη

Το τελικό βήμα πριν ξεκινήσουμε την διαδικασία της χαρτογράφησης είναι να δημιουργήσουμε ένα μοντέλο στο gazebo το οποίο θα χαρτογραφήσουμε.
Ανοίγοντας το mvrobot_world.launch που δημιουργήσαμε στην ενότητα 3.5 μεταφερόμαστε στην καρτέλα insert και επιλέγουμε το connection to model database, μετά από ένα μικρο χρονικά διάστημα θα έχουμε συνδεθεί με την βάση δεδομένων του gazebo και θα έχουμε στην διάθεση μας μερικά έτοιμα μοντέλα.



*Εικόνα 4.2.2 Εισαγωγή έτοιμων μοντέλων στο gazebo*

Επιλέγουμε κάποια μοντέλα με τυχαίο τρόπο και φροντίζουμε ότι ο τελικός χάρτης θα εμπεριέχει πρώτον, κάποιο είδους ορίου(π.χ. τοίχο) έτσι ώστε να έχουμε οριοθετήσει τον χώρο τον οποίο θα κινείται το IAV, δεύτερον αρκετά εμπόδια ώστε να είναι πιο εμφανή τα όρια του αισθητήρα και να παρατηρήσουμε πως δημιουργείται ο χάρτης καθώς το IAV κινείται. Έτσι το τελικό μοντέλο του gazebo έχει την μορφή της εικόνας 4.2.3




*Εικόνα 4.2.3 Μοντέλο χώρου στο gazebo*

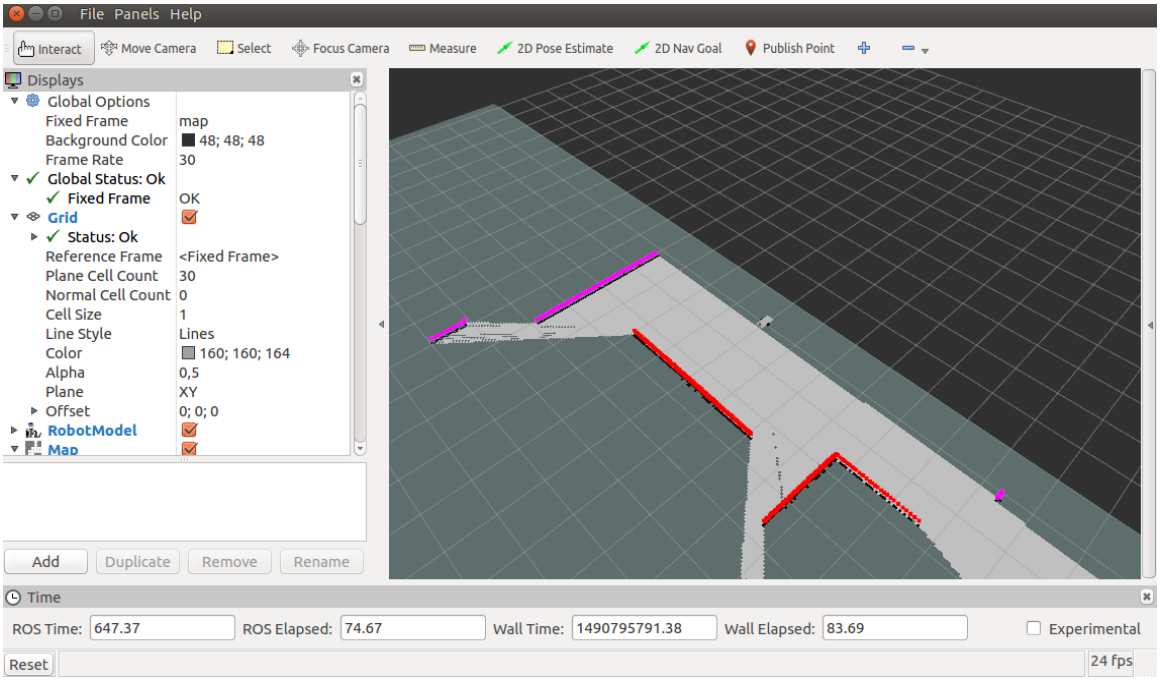
Στη συνέχεια θα δημιουργήσουμε ένα εκτελέσιμο αρχείο με το όνομα mvrobot_rviz_gmapping.launch στο φάκελο launch του πακέτου mvrobot_desrciption.
Μέσω του αρχείου αυτού μπορούμε να εμφανίσουμε την εικόνα του IAV στο rviz.
Το rviz μας δίνει την δυνατότητα να εισάγουμε ένα αρχείο το οποίο θα εμπεριέχει όλες τις ρυθμίζεις που χρειαζόμαστε, για παράδειγμα σε ποια θέματα πρέπει να εγγραφεί το rviz έτσι ώστε να λαμβάνει τις τιμές των αισθητήρων,του χάρτη και την εικόνα του IAV. Στον φάκελο mvrobot_descpiption δημιουργούμε το αρχείο gmapping.rviz το οποίο εμπεριέχει όλες τις απαραίτητες παραμέτρους.

### 4.2.4 Χαρτογράφηση

Σε τρία διαφορετικά τερματικά εκτελούμε τις παρακάτω εντολές στο περιβάλλον εργασίας μας για να ξεκινήσουμε την διαδικασία χαρτογράφησης.

**$ roslaunch mvrobot_gazebo mvrobot_world.launch**
**$ roslaunch mvrobot_description mvrobot_rviz_gmapping.launch**
**$ roslaunch mvrobot gmapping.launch**

Η εικόνα του gazebo παραμένει ιδία, ενώ στο παράθυρο του rviz μας εμφανίζεται η εικόνα 4.2.4

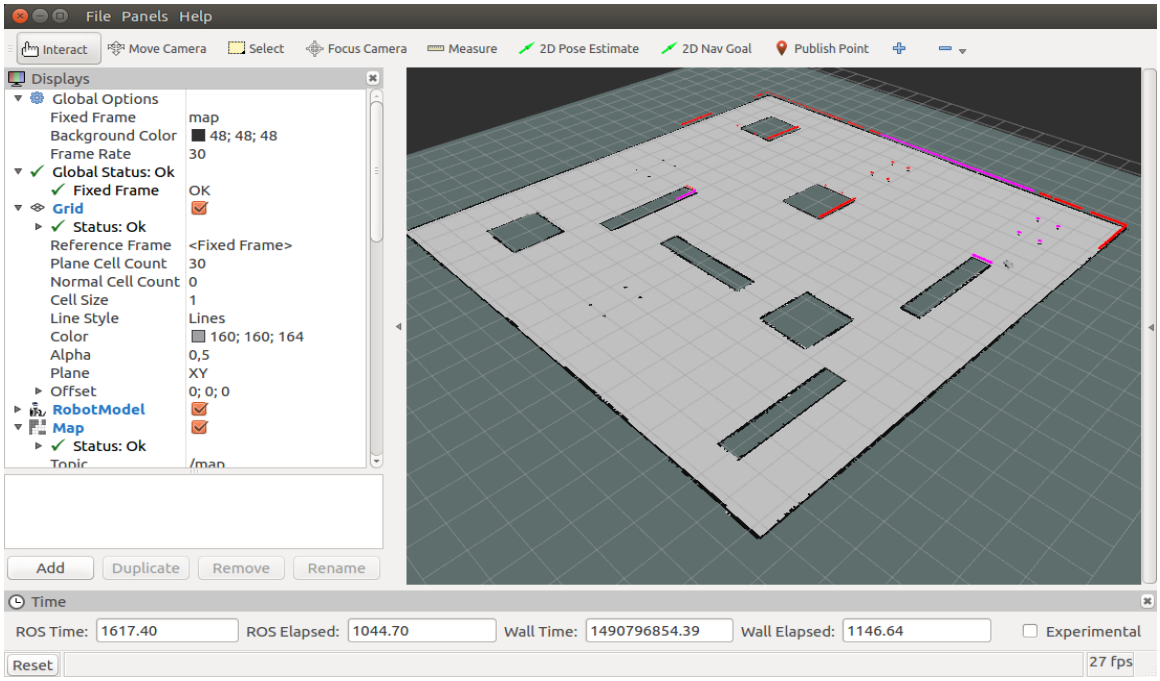



*Εικόνα 4.2.4 Αρχικοποίηση χαρτογράφησης*

Στην παραπάνω εικόνα παρατηρούμε αυτά που αναφέραμε στην θεωρία του slam. Τα pixel της εικόνας χωρίζονται σε τρεις κατηγορίες ανάλογα με την τιμή που τους έχει δοθεί, σε γκρι(ανεξερεύνητος χώρος), άσπρα(χαρτογραφημένος ελεύθερος χώρος), μαύρα(εμπόδια). Με κόκκινο και μοβ χρώμα απεικονίζονται οι ακτίνες του laser αισθητήρα καθώς προσκρούουν στα εμπόδια. Τέλος στην αριστερή στήλη αναφέρονται σε ποια θέματα έχει εγγραφεί το rviz έτσι ώστε να απεικονίσει τα δεδομένα.
Σε ένα νέο τερματικό εκτελούμε:

**$ roslaunch mvrobot_navigation mvrobot_teleop.launch**

Καθώς το IAV κινείται παρατηρούμε ότι όλο και περισσότερα pixel γίνονται άσπρα καθώς σίγα σιγά μειώνεται ο ανεξερεύνητος χώρος μέχρι να καταδείξουμε στην εικόνα 4.2.5

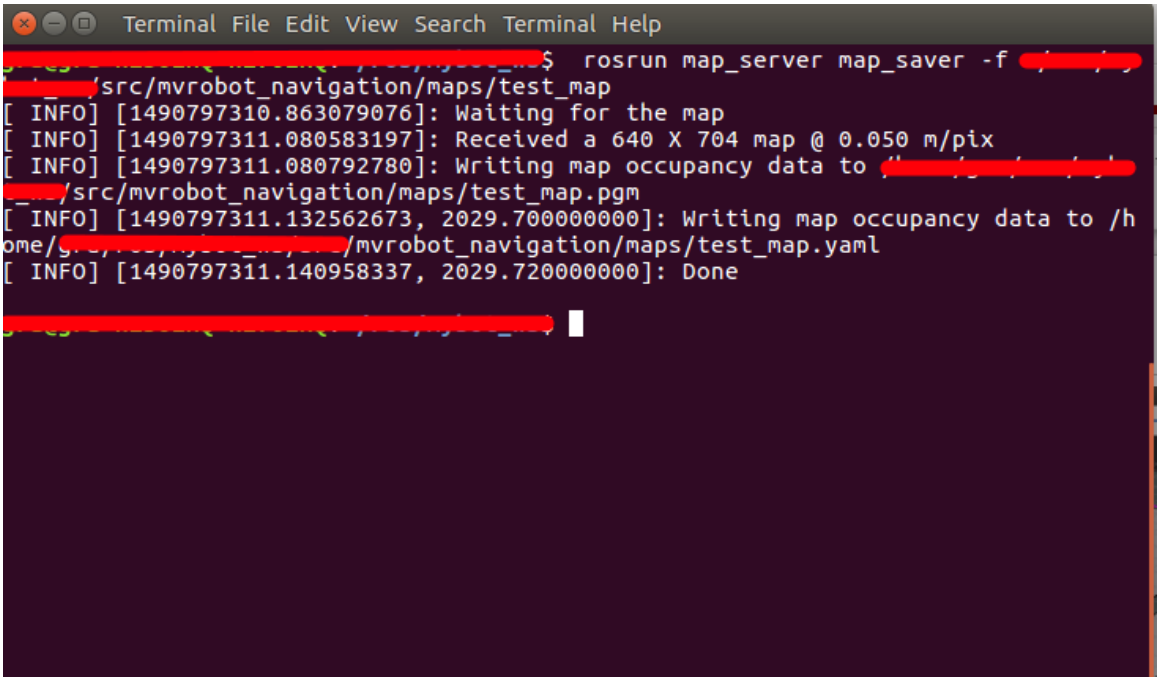



*Εικόνα 4.2.5 Ολοκλήρωση χαρτογράφησης*

Έχοντας ολοκληρώσει την διαδικασία της χαρτογράφησης απομένει να αποθηκεύσουμε την εικόνα για μετέπειτα χρήση στην αυτόνομη πλοήγηση του οχήματος. Στο πακέτο mvrobot_navigation δημιουργούμε έναν φάκελο με το όνομα maps για να αποθηκεύσουμε σε αυτόν την εικόνα. Σε ένα νέο τερματικό αφού μεταβούμε στο περιβάλλον εργασίας μας εκτελούμε:

**$ rosrun map_server map_saver -f ~/catkin_ws/src/mvrobot_navigation/maps/test_map**

Εφόσον στο τερματικό μας εμφανιστεί το μήνυμα της εικόνας 4.5.6 η διαδικασία έχει ολοκληρωθεί.




*Εικόνα 4.2.6 Αποθήκευση χαρτογράφησης*

## 4.3 AMCL

### 4.3.1 Αρχή λειτουργίας

Το τελικό βήμα για την ολοκλήρωση όπως και ο στόχος για το Navigation Stack είναι η αυτόνομη κίνηση του IAV μέσα στον χώρο χωρίς να υπάρχει κίνδυνος πρόσκρουσης με άλλα εμπόδια και η κατεύθυνση του με ακριβείς σχετικές συντεταγμένες μέσα στο χώρο. Για να επιτευχθεί το παραπάνω εγχείρημα έχει αναπτυχθεί μια τεχνική εντοπισμού η οποία ονομάζεται Monte Carlo Localization. Η τεχνική αυτή χρησιμοποιεί την μέθοδο PF (βλέπε ενότητα 4.2.1) και λειτουργεί εξίσου ικανοποιητικά τόσο στον τοπικό (local) όσο και στον γενικό (global) εντοπισμό και προϋποθέτει την είσοδο εικόνας κάποιου χάρτη σαν σημείο αναφοράς. Ο αλγόριθμός αρχικά δέχεται ως είσοδο τον χάρτη στον οποίο κατανέμει ένα πλήθος σωματιδίων σε όλες τις πιθανές θέσεις. Στην συνέχεια λαμβάνοντας δεδομένα από τους αισθητήρες ( odom,tf και σάρωσης) και την εικόνα του χάρτη, κατανέμει ένα δείκτη βαρύτητας στα σωματίδια που βρίσκονται στις πιο πιθανές τοποθεσίες που μπορεί να βρίσκεται το IAV. Έπειτα πραγματοποιεί δειγματοληψία για την απομόνωση των σωματιδίων με τον μεγαλύτερο δείκτη βαρύτητας, η διαδικασία αυτή επαναλαμβάνεται μέχρι να βρεθεί η ζητούμενη τοποθεσία. Δυο κύρια μειονεκτήματα αυτής τις μεθόδου είναι τα εξής.

- Έλλειψη δειγμάτων τόσο στο γενικό όσο και στο τοπικό εντοπισμό. Στον γενικό εντοπισμό(global localization) παρατηρείται δυσκολία στον εντοπισμό λόγω έλλειψής δειγμάτων λόγω της τυχαίας κατανομής τους

- Στην αρχή της διαδικασίας υπάρχει πλήρης αγνοία για την θέση του IAV, ως αποτέλεσμα η διακύμανση των σωματιδίων πραγματοποιείται με τυχαίο τρόπο. Συνεπώς ο αλγόριθμος χρειάζεται παραπάνω κόστος για να ολοκληρωθεί.

Τα παραπάνω προβλήματα λύνονται με την εφαρμογή του AMCL(Adaptive Monte Carlo Localization).
Στον αλγόριθμο αυτόν έχουμε την δυνατότητα είτε να το τοποθετήσουμε το IAV σε μια τυχαία αρχική θέση είτε μπορούμε να μην κάνουμε κάποια αρχική εκτίμηση τοποθέτησης. Καθώς το IAV κινείται δημιουργούνται τα σωματίδια στα οποία κατανέμονται οι δείκτες βαρύτητας. Η δειγματοληψία πραγματοποιείται εφόσον έχουν συγκεντρωθεί τα δεδομένα από τους αισθητήρες και έχουν αντιστοιχηθεί οι δείκτες βαρύτητας. Αν παρατηρηθούν μεγάλες μετρήσεις στους δείκτες βαρύτητας σημαίνει έχουν βρεθεί οι πιθανές θέσεις συνεπώς ο αριθμός των δειγμάτων παραμένει μικρός. Στην περίπτωση στην οποία παρατηρηθούν μικρές τιμές στους δείκτες οφείλεται είτε σε περιπτώσεις γενικού εντοπισμού (global localization) είτε σε περιπτώσεις που από τους αισθητήρες έχουν παρθεί λανθασμένα δεδομένα επειδή το IAV έχασε την πορεία του. Συνεπώς για να ολοκληρωθεί η διαδικασία πρέπει να αυξηθεί ο αριθμός των δειγμάτων.

### 4.3.2 Ορισμός και ετοιμασία αρχείων για τον εντοπισμό

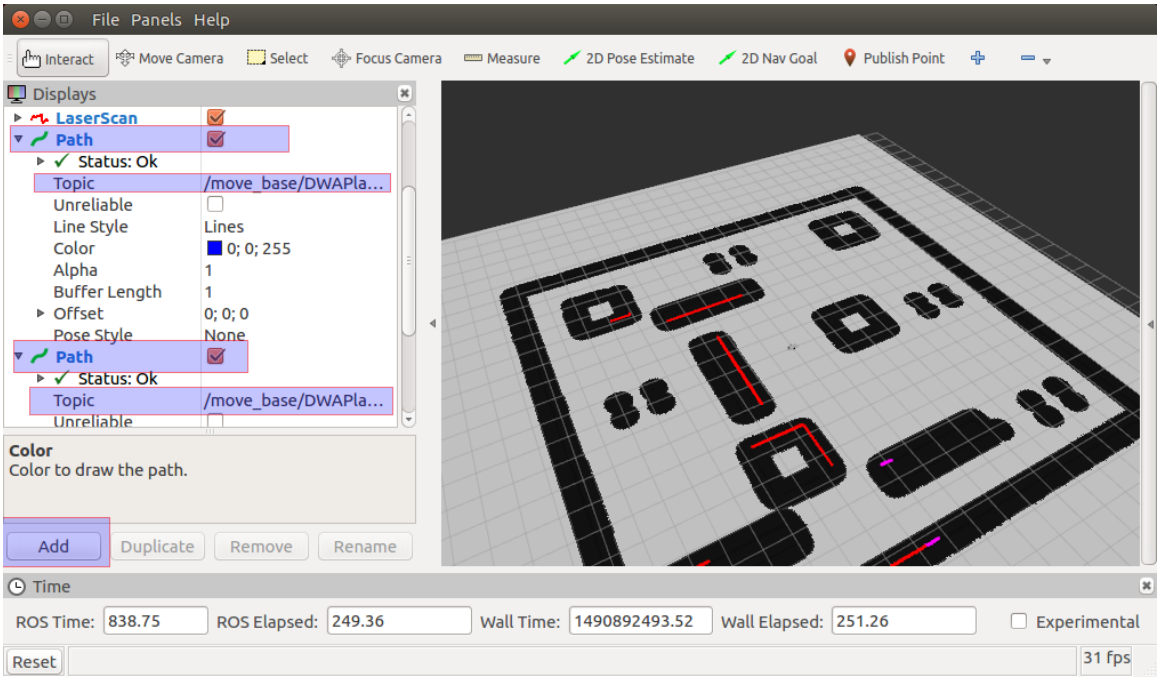
Έχοντας εκτελέσει με επιτυχία την διαδικασία της χαρτογράφησης το τελικό βήμα είναι να ορίσουμε τα αρχεία για την αυτόματη πλοήγηση με συγκεκριμένες συντεταγμένες. Τα αρχεία αυτά θα εμπεριέχουν τις ρυθμίσεις προς το Navigation Stack, τον κόμβο στον οποίο θα εκτελείται το AMCL, από ποιους αισθητήρες θα δέχεται δεδομένα και σε ποια διαδρομή βρίσκεται ο χάρτης τον οποίο θα διαβάζει. Στο φάκελο launch του πακέτου mvrobot_navigation δημιουργούμε το αρχείο amcl.launch.
Στο αρχείου αυτό περιλαμβάνεται το αρχείο amc.launch.xml το οποίο εισάγει όλες της παραμέτρους που έχουμε ορίσει για το Navigation Stack. Βάσει αυτών τον παραμέτρων αρχικοποιούμε τους κόμβους για τον εντοπισμό του IAV και εύρεση της βέλτιστης διαδρομής. Δημιουργούμε το αρχείο αυτό τον φάκελο param.
Για να απεικονίσουμε το IAV στο rviz (όπως και στην προηγούμενη ενότητα) χρειαζόμαστε το παρακάτω αρχείο στο εμπεριέχεται η εικόνα του IAV καθώς και οι αρχικοποίηση των κόμβων joint_state_publisher robot_state_publisher. Στο τέλος του αρχείου εισάγουμε το αρχείο gmapping.rviz απο την προηγούμενη ενότητα. Στο φάκελο launch του πακέτου mvrobot_description τοποθετούμε το αρχείο mvrobot_rviz_amcl.launch

### 4.3.3 Προσομοίωση του AMCL

Σε τρία διαφορετικά τερματικά εκτελούμε τις παρακάτω εντολές στο περιβάλλον εργασίας μας για να ξεκινήσουμε την διαδικασία εντοπισμού.

**$ roslaunch mvrobot_gazebo mvrobot_world.launch**
**$ roslaunch mvrobot_description mvrobot_rviz_amcl.launch**
**$ roslaunch mvrobot amcl.launch**

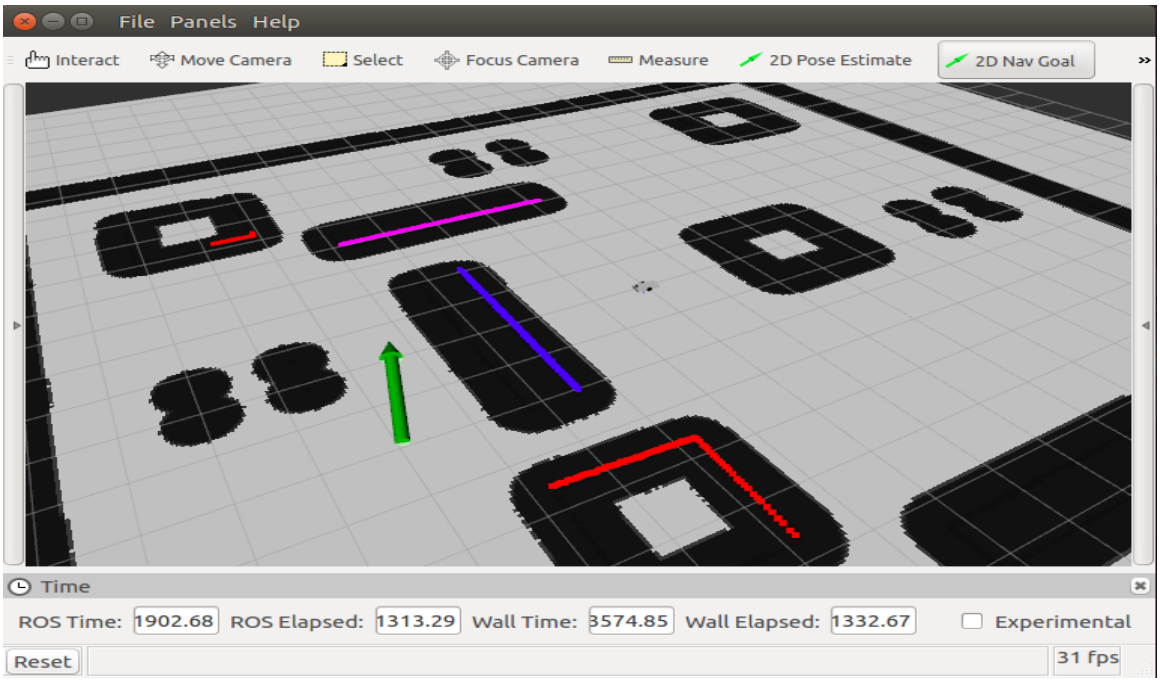
Στο παράθυρο του rviz, εικόνα 4.3.1, προσθέτουμε δυο διαδρομές πατώντας το κουμπί add. Αφού τα έχουμε προσθέσει κάνουμε εγγραφή στο θέμα /move_base/DWAPlannerROS/global_plan για τη πρώτη διαδρομή και /move_base/DWAPlannerROS/local_plan για το δεύτερο, επίσης αλλάζουμε τα χρώματα τους σε μπλε κι πράσινο αντίστοιχα.Στην εικόνα 4.3.1 παρατηρούμε ότι σε κάθε εμπόδιο έχουν προστεθεί εξογκώματα τα οποία δεν αντιστοιχούν σε κάποιο εμπόδιο. Πρόκειται για ένα εικονικό εμπόδιο που έχουμε ορίσει, έμπρακτα είναι η ελάχιστη απόσταση που μπορεί το IAV να πλησιάσει το εμπόδιο. Η απόσταση αυτή ισούται με 40%-60% του μήκους του IAV έτσι ώστε το IAV να μπορεί να εκτελεί οποιαδήποτε είδους κίνηση όταν βρίσκεται κοντά στο εμπόδιο.



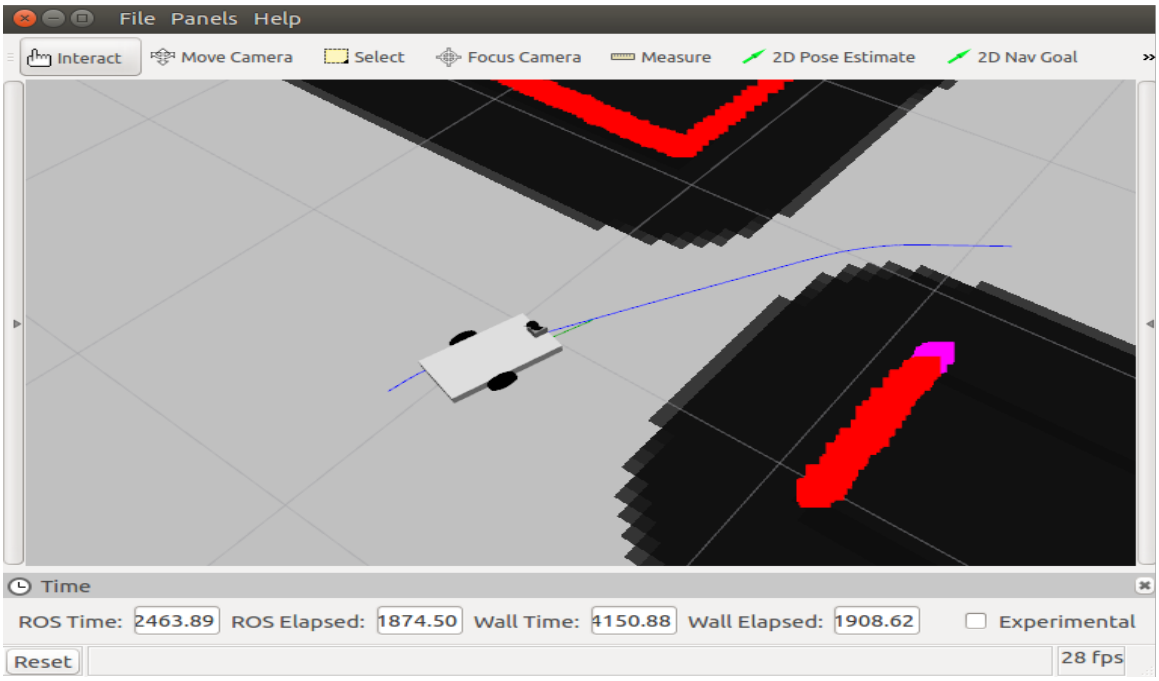
*Εικόνα 4.3.1 Απεικόνιση του IAV στο rviz (amcl)*

### 4.3.3 Αποστολή σημείου-στόχου μέσω του Navigation Stack

Πιέζοντας το πλήκτρο 2d Nav Goal μπορούμε να ορίσουμε μια διαδρομή του IAV ως αυτό το σημείο όπως επίσης και την ακριβή τοποθέτησή του.




*Εικόνα 4.3.2 Ορισμός διαδρομής*




*Εικόνα 4.3.3 Αλγόριθμός για, το global costamap (μπλε γραμμή), το local costmap (πράσινη γραμμή)*

Τέλος έχουμε την δύνατότητα εντοπισμού συντεταγμένων και γωνίας κάθε σημείου στο χάρτη επιλέγοντας το κουμπί 2D Pose Estimate και οποιοδήποτε σημείο στον χάρτη. Τα αποτελέσματα (εικόνα 4.3.4) αναγράφονται στο τερματικό το οποίο τρέξαμε τον κώδικα για το amcl

```
[ INFO] [1490894457.870686134, 2768.580000000]: Setting pose (2768.580000): -2.5
66 -4.195 1.395
[ INFO] [1490894461.286554580, 2771.990000000]: Setting pose (2771.990000): 6.22
3 1.499 2.147
```

*Εικόνα 4.3.4 Εντοπισμός συντεταγμένων*

**Σημείωση**: Οι κώδικές και τα αρχεία που αναφέρονται στο Κεφάλαιο 4 εμπεριέχονται στο Παράρτημα Β.

## Κεφάλαιο 5

Ένας από τους στόχους της εργασίας είναι να συμβάλει στην ανάπτυξη του ROS αυτοματοποιώντας την διαδικασία της χαρτογράφησης ώστε να μην είναι αναγκαία η ανθρώπινη επίβλεψη και καθοδήγηση. Στο κεφάλαιο αυτό αναπτύσσεται ένας αλγόριθμος μέσω του οποίου το IAV αρχικά εκτελεί μια ψευδό τυχαία κίνηση στον χώρο αποφεύγοντας εμπόδια ενώ εκτελεί την διαδικασία της χαρτογράφησης. Ο αλγόριθμος αυτός δοκιμάστηκε με πολύ ενθαρρυντικά αποτελέσματα σε διάφορες συνθήκες.

### 5.1 Ανάλυση κώδικα

Έως τώρα η διαδικασία της χαρτογράφησης πραγματοποιείται με τον τρόπο που παρουσιάστηκε στην παράγραφο 4.2.3. Ο αλγόριθμός που παρουσιάζεται παρακάτω αυτοματοποιεί την διαδικασία της χαρτογράφησης. Το αρχείο αυτό ονομάζεται free_nav_drive.cpp. Στο πρόγραμμα συμπεριλαμβάνεται μια τροποποιημένη έκδοση της κεφαλίδας image.h. Στο πρόγραμμα free_nav_drive υλοποιείται η δημιουργία ενός κόμβου με το όνομα free_nav_drive_node μέσω του οποίου ορίζονται τρεις συναρτήσεις με την μέθοδο επανάκλησης(callback). Η μέθοδος αυτή μας επιτρέπει να εκτελούμε συναντήσεις με την προϋπόθεση κάποιου συμβάντος. Ο κόμβος αυτός εγγράφεται στο θέμα /scan και λαμβάνει τα δεδομένα του αισθητήρα laser και εκδίδει μηνύματα απο το θέμα /cmd_vel με τύπο δεδομένων geometry msgs/Twist. Στην πρώτη ενότητα του προγράμματος δηλώνουμε τις μεταβλητές με της οποίες θα εγγραφούμε και θα εκδώσουμε μηνύματα σε κάποια θεμάτα όπως επίσης και τις συναρτήσεις που εκτελούνται όταν λαμβάνει χώρα κάποιο συμβάν. Οι συναρτήσεις αυτές είναι τρεις, δύο συσχετίζονται με κάποιο χρονικό συμβάν και η τρίτη όταν λαμβάνονται δεδομένα από τον αισθητήρα. Η πρώτη χρονική συνάρτηση movement_timercallback εκτελείται μετά το πέρας του χρονικού ορίου του οποίου έχουμε ορίσει σαν μέγιστο χρονικό όριο που θα κινείται το IAV, ενώ η δεύτερη map_timercallback μετά το πέρας του χρονικού ορίου που έχουμε ορίσει για το διάβασμα, επεξεργασία, αποθήκευση του νέου χάρτη. Η τρίτη συνάρτηση free_nav_drive_callback εκτελείται όταν πραγματοποιείται σάρωση από τον αισθητήρα. Η συνάρτηση αυτή επεξεργάζεται τα δεδομένα από τον αισθητήρα και εκδίδει δεδομένα για την ταχύτητα και την γωνία περιστροφής του IAV. Στην κύρια συνάρτηση του προγράμματος αρχικοποιείται ο κόμβος free_nav_drive_node και δηλώνεται η συχνότητα με την οποία θα διαβάζονται και θα εκδίδονται τα δεδομένα όπως επίσης και η συχνότητα των παραπάνω συναρτήσεων. Μετά το πέρας ενός λεπτού εκτελείται η συνάρτηση map_timercallback. Σε αυτήν την συνάρτηση αρχικά αποθηκεύεται η τωρινή εικόνα του χάρτη ως pgm αρχείο τύπου p5. Στη συνέχεια διαβάζεται κάθε pixel της εικόνας ξεχωριστά και αυξάνεται ένας μετρητής κάθε φόρα που το pixel είναι άσπρο. Επιπλέον αποθηκεύει το μέγεθος της σε pixels καθώς και το ποσοστό του άσπρου. Τέλος το ποσοστό του άσπρου χρώματος αποθηκεύεται σε έναν πίνακα. Όταν το ποσοστό του άσπρου χρώματός παραμείνει ίδιο σε τρεις συνεχόμενες εικόνες του χάρτη σηματοδοτείται το τέλος της κίνησης του IAV διότι έχει ολοκληρωθεί η διαδικασία της χαρτογράφησης. Το ανώτατο όριο της κίνησης του IAV εκτιμάται από το μέγεθος του χώρου αναζήτησης καθώς και το πλήθος των εμποδίων, για τις ανάγκες της εργασίας ο χρόνος αυτός έχει οριστεί στα είκοσι λεπτά. Στην περίπτωση που ο χρόνος αυτός έχει παρέλθει εκτελείται η συνάρτηση movement_timercallback η οποία τερματίζει την κίνηση του IAV. Στο αρχείο image.h ορίζεται η κλάση image όπου μέσω των συναρτήσεων που έχουν οριστεί σε αυτήν πραγματοποιείται το διάβασμα και η επεξεργασία της εκάστοτε εικόνας του χάρτη.

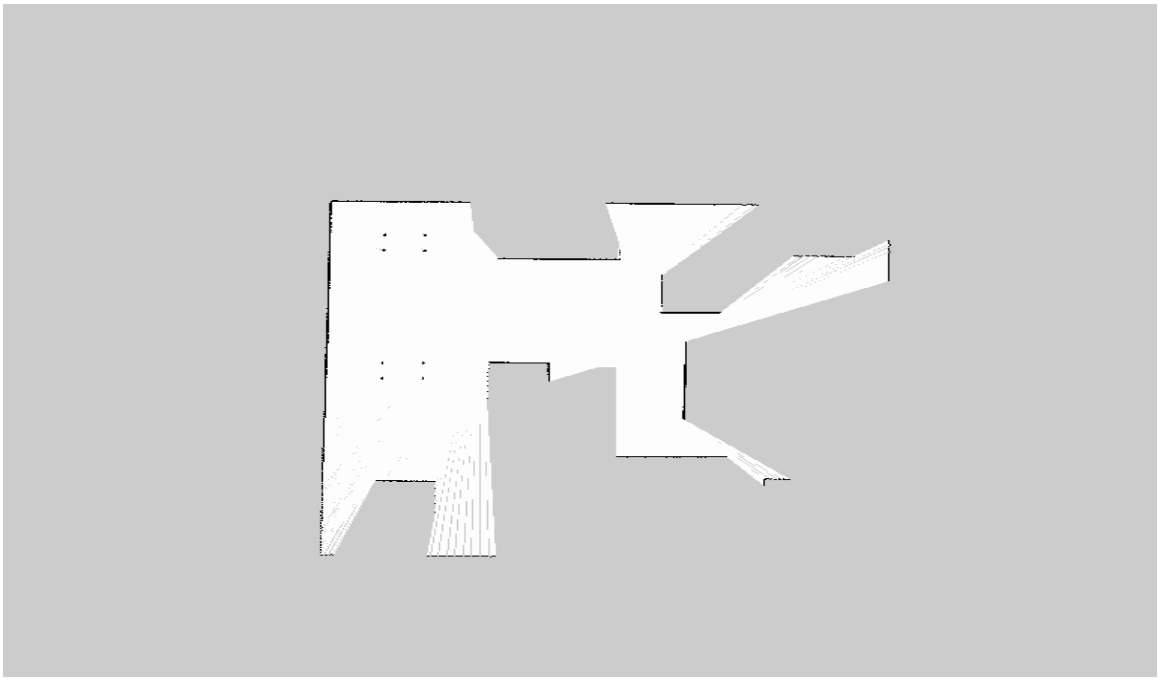
**Σημείωση**:Στον κώδικα του αρχείου free_nav_drive.cpp χρησιμοποιούνται τα μαθηματικά μοντέλα του stdr_obstacle_avoidance.cpp γραμμές 118-131 από:
http://wiki.ros.org/stdr_simulator/Tutorials/Create%20a%20map%20with%20gmapping.
Η πηγή αναφέρεται στην βιβλιογραφία

## 5.2 Δοκιμή του αλγορίθμου σε διάφορα μοντέλα χώρου του gazebo

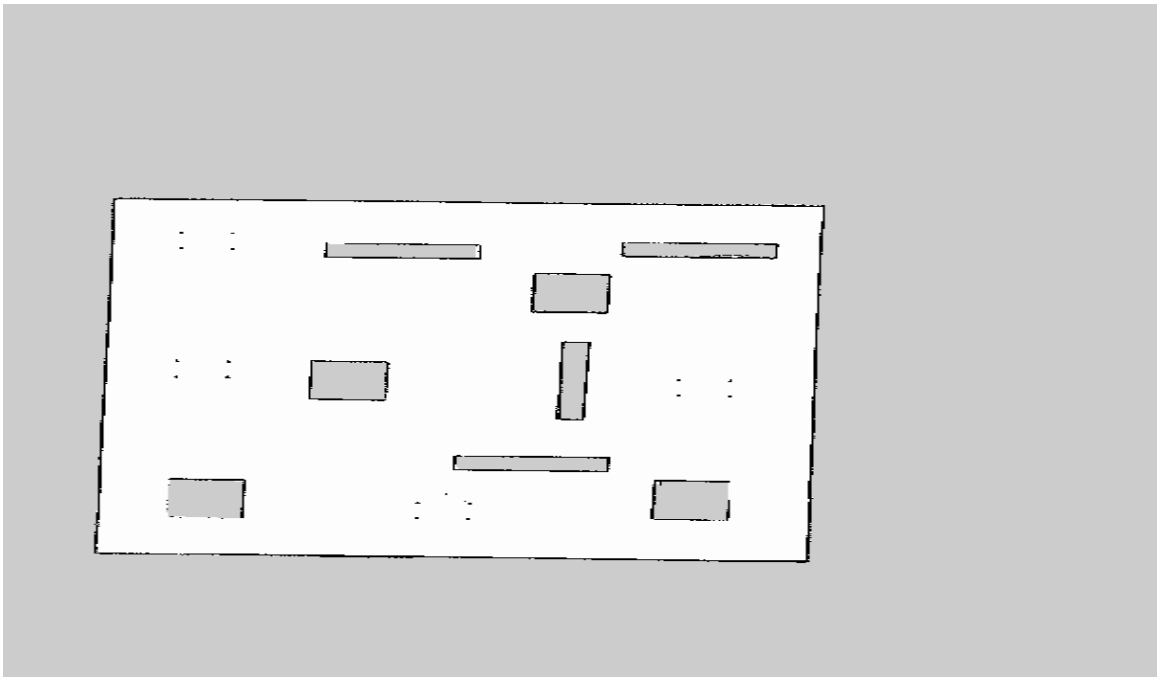
Σε αυτήν την ενότητα θα δοκιμαστεί ο αλγόριθμός σε δύο διαφορετικά μοντέλα χώρου, στο μοντέλο που δημιουργήσαμε στο προηγούμενο κεφάλαιο και σε περιβάλλον αποθήκης.

### 5.2.1 Δοκιμή στο δοκιμαστικό μοντέλο(test_map)

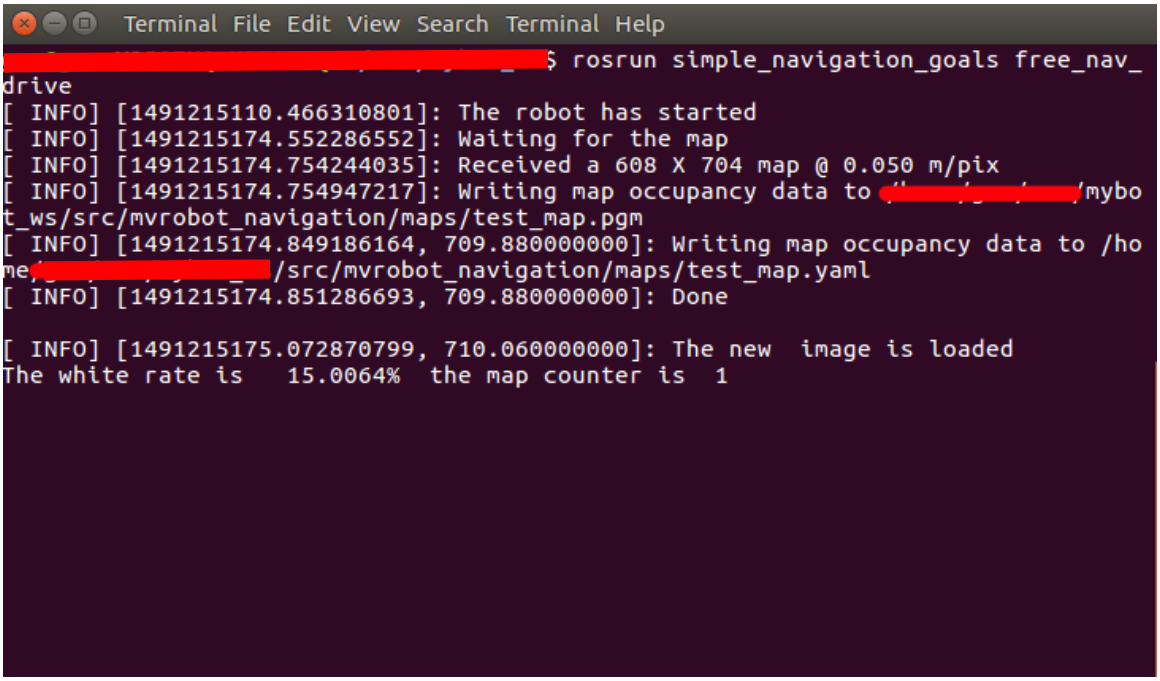
Το δοκιμαστικό μοντέλο είναι το μοντέλο που δημιουργήσαμε στο προηγούμενο κεφάλαιο. Παρακάτω απεικονίζονται η αρχική και η τελική εικόνα του χάρτη, όπως και η πρώτη και η τελευταία ένδειξη του τερματικού.



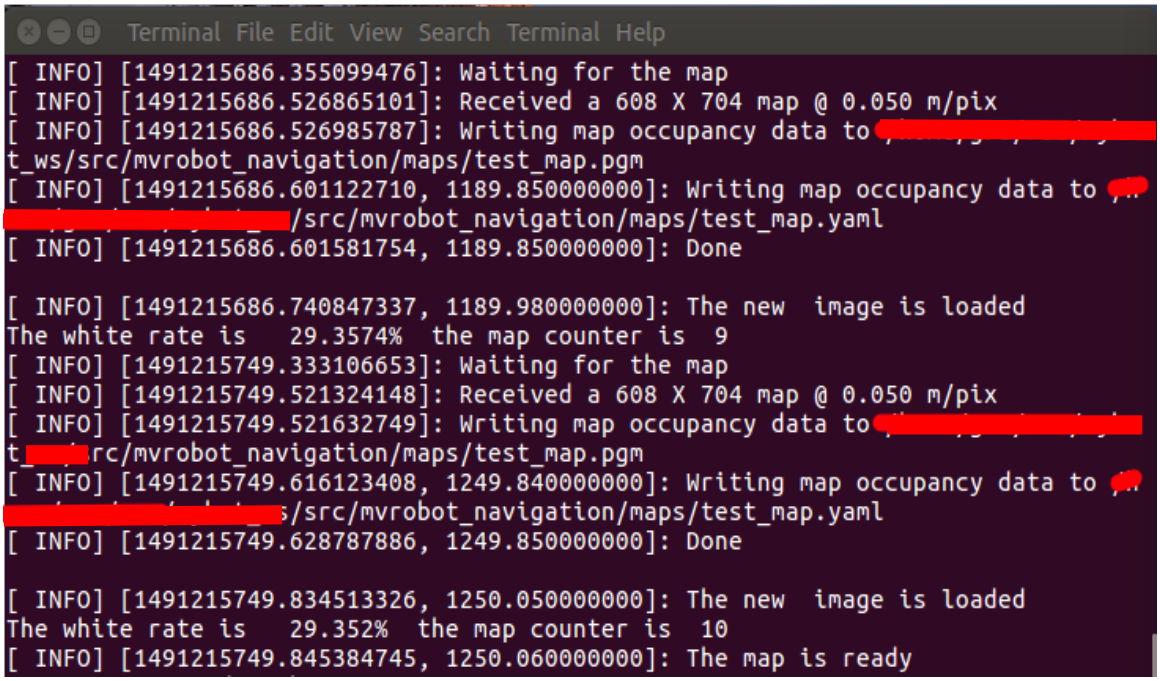
*Εικόνα 5.2.1 Πρώτη αποθηκευμένη εικόνα του χάρτη(test_map)*



*Εικόνα 5.2.2 Ολοκληρωμένη εικόνα του χάρτη (test_map)*




*Εικόνα 5.2.3 Αποθήκευση της πρώτης εικόνας (test_map)*

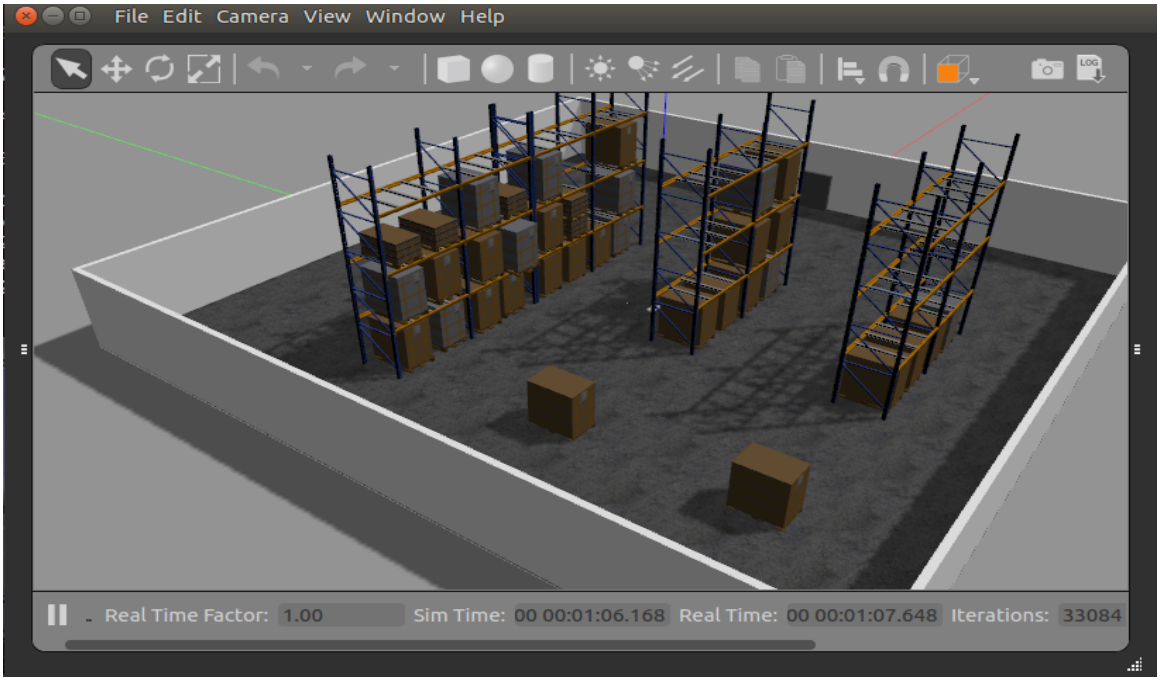



*Εικόνα 5.2.4 Ολοκλήρωση της διαδικασίας (test_map)*

Παρατηρούμε στις εικόνες 5.2.3 και 5.2.4 ότι στο παράθυρο του τερματικού εμφανίζεται το μέγεθός, ο σειριακός αριθμός καθώς και το ποσοστό επί % του άσπρου της εικόνας, επιπλέον εμφανίζεται και σε ποιον φάκελο την έχουμε αποθηκεύσει. Εκτιμώμενος χρόνος ολοκλήρωσης της διαδικασίας 10-12 λεπτά.

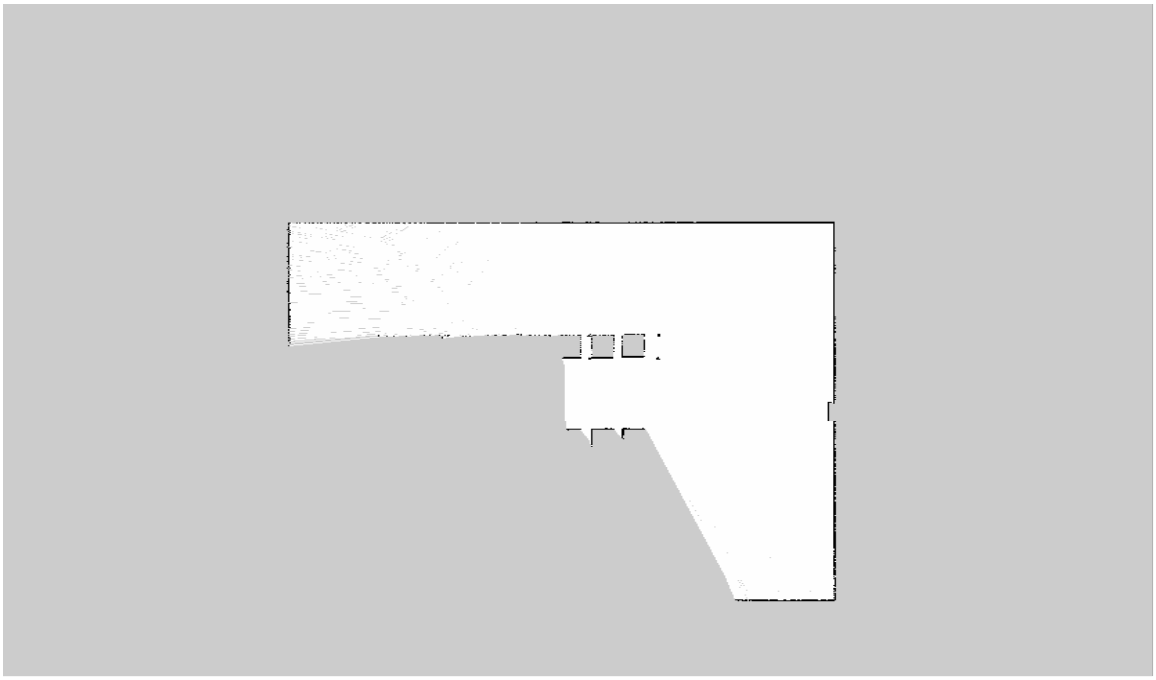
### 5.2.2 Δοκιμή σε μοντέλο αποθήκης

Για τους σκοπούς της εργασίας δημιουργήθηκε ένα περιβάλλον προσομοίωσης αποθήκης. Στο μοντέλο αυτό χρησιμοποιούνται αντικείμενα(ράφια, κούτες, παλέτες) τα οποία δεν περιλαμβάνονται στην βάση δεδομένων του gazebo, αλλά έχουν δημιουργηθεί από άλλους χρήστες. Το όνομα του αρχείου είναι warehouse.world και είναι τύπου xml. Στην εικόνα 5.2.5 απεικονίζεται το μοντέλο της αποθήκης

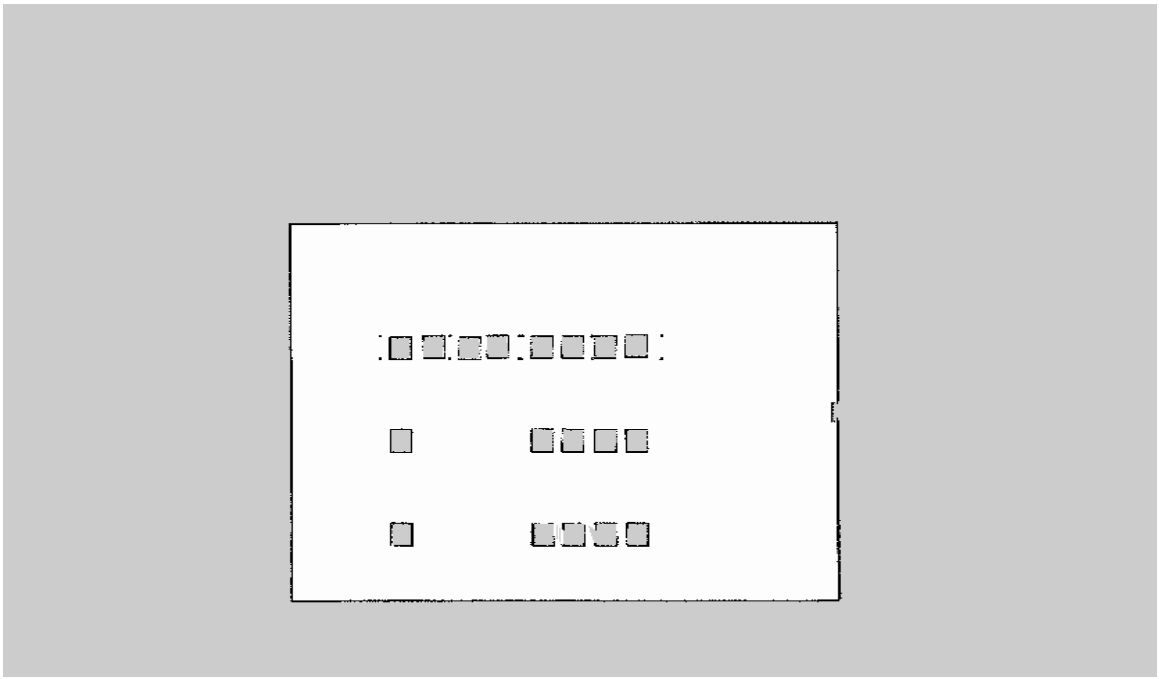



*Εικόνα 5.2.5 Μοντέλο αποθήκης*

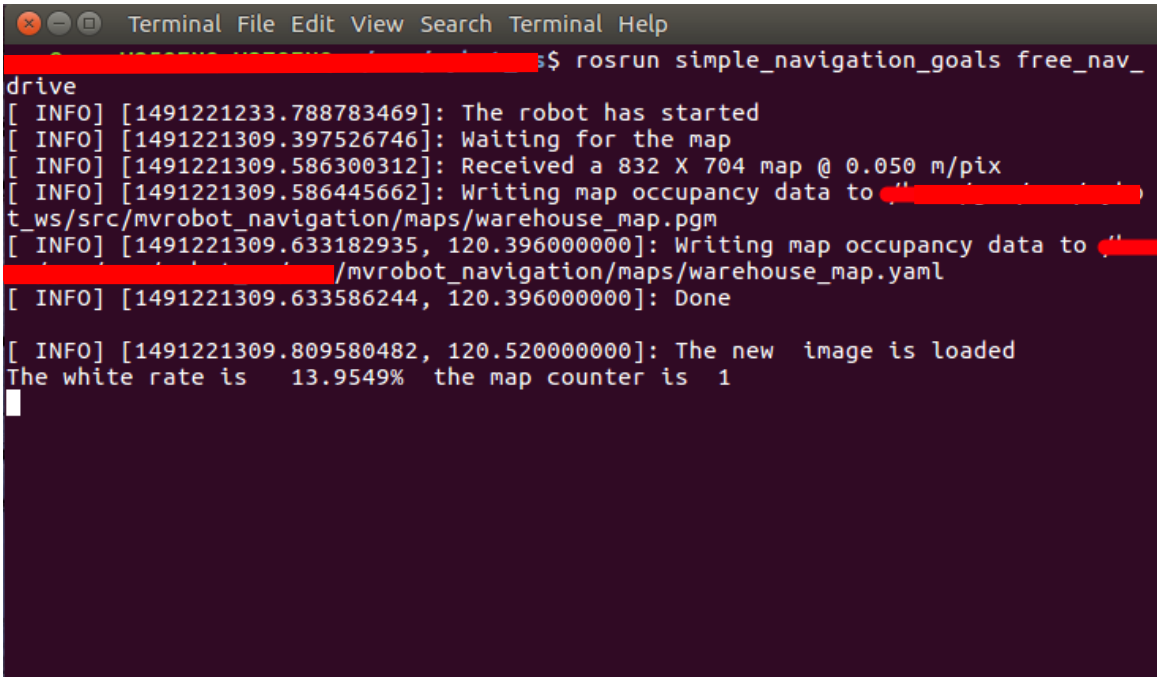
Παρακάτω απεικονίζονται η αρχική και η τελική εικόνα του χάρτη, όπως και η πρώτη και η τελευταία ένδειξη του τερματικού.



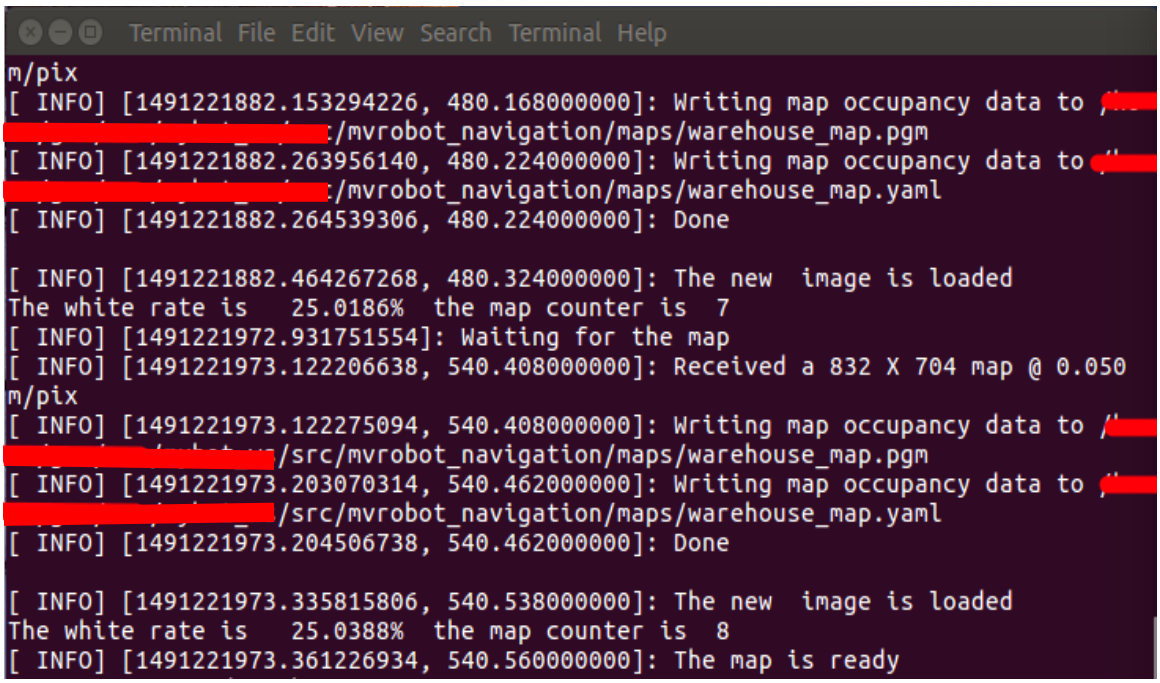
*Εικόνα 5.2.6 Πρώτη αποθηκευμένη εικόνα του χάρτη(warehouse_map)*



*Εικόνα 5.2.7 Ολοκληρωμένη εικόνα του χάρτη (warehouse_map)*




*Εικόνα 5.2.8 Αποθήκευση της πρώτης εικόνας (test_map)*

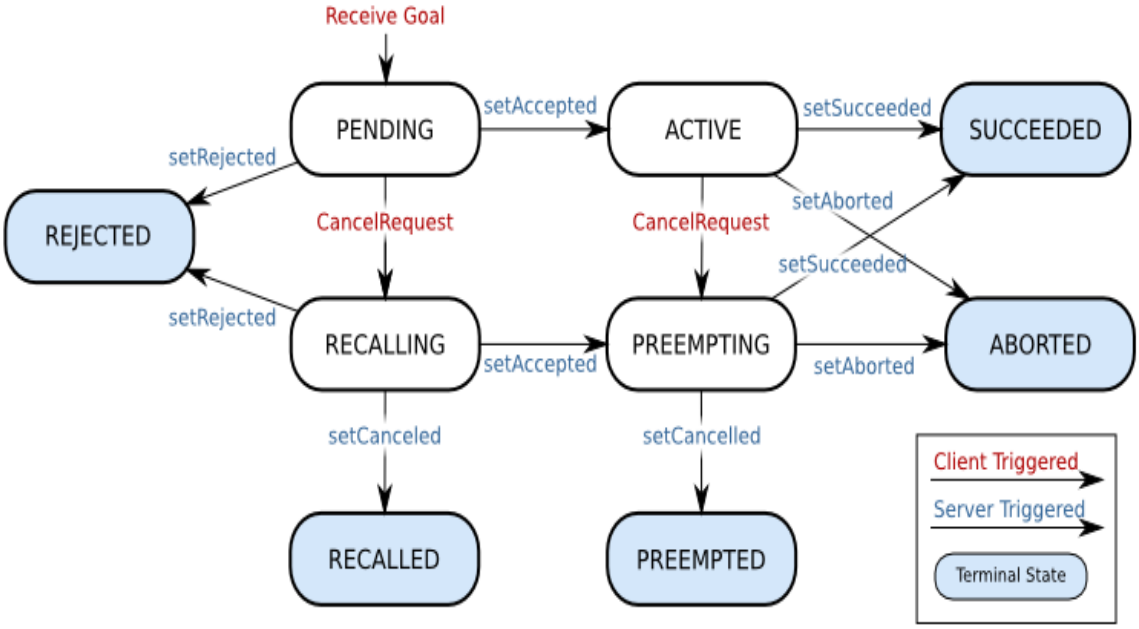



*Εικόνα 5.2.9 Ολοκλήρωση της διαδικασίας*

Εκτιμώμενος χρόνος ολοκλήρωσης της διαδικασίας 8-10 λεπτά

### 5.3 Μεταφορά παλετών από την αποθήκη

Για την πλοήγηση του IAV μέσω του Navigation Stack το ROS μας δίνει την δυνατότητα να στείλουμε τελικούς στόχους εκτός από την χρήση του rviz(ενότητα 4.3.2) αλλά και μέσω ενός προγράμματος σε c++. Στο πρόγραμμα πρέπει να αρχικοποιήσουμε έναν actionlib πελάτη. Ο κεντρικός κόμβος του Navigation Stack (move_base) ανήκει στην κατηγορία των simple action server κόμβων και για να μπορέσουμε είναι επιτύχουμε επικοινωνία μαζί του πρέπει να δημιουργήσουμε έναν simple action client κόμβο και να επικοινωνήσουμε μεσώ του actionlib stack. Το actionlib stack προσφέρει την δυνατότητα επικοινωνίας μεταξύ δυο κόμβων. Ένα από τα βασικά προτερήματα του είναι οτι μας δίνει την δυνατότητα να γνωρίζουμε σε ποιο στάδιο βρίσκεται η υπηρεσία που έχει σταλθεί από τον simple action client προς τον simple action server καθώς και η δυνατότητα να ακυρώσουμε την υλοποίηση της υπηρεσίας. Στην εικόνα 5.3.1 φαίνεται η διαδικασία που λαμβάνει χώρα στο actionlib stuck από την ώρα που θα σταλθεί η υπηρεσία μέχρι την πραγματοποίηση/ακύρωση της.



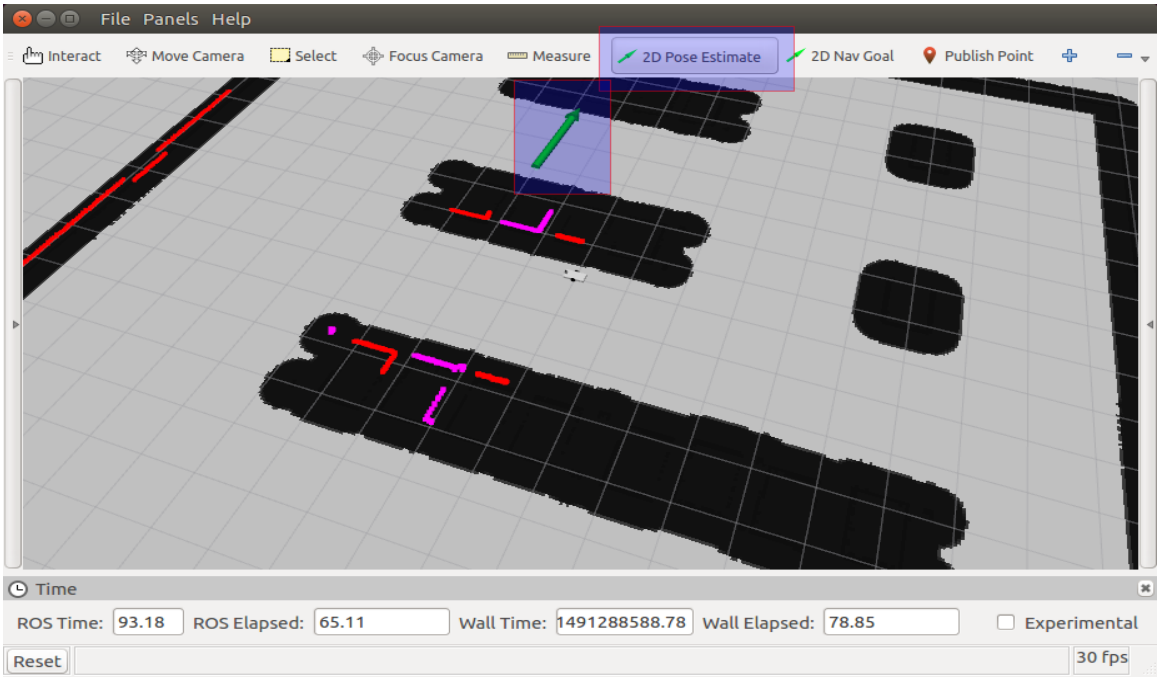

Εικόνα 5.3.1 Αρχιτεκτονική actionlib stack
http://wiki.ros.org/actionlib/DetailedDescription

### 5.3.1 Δημιουργία ενός Simple Action Client

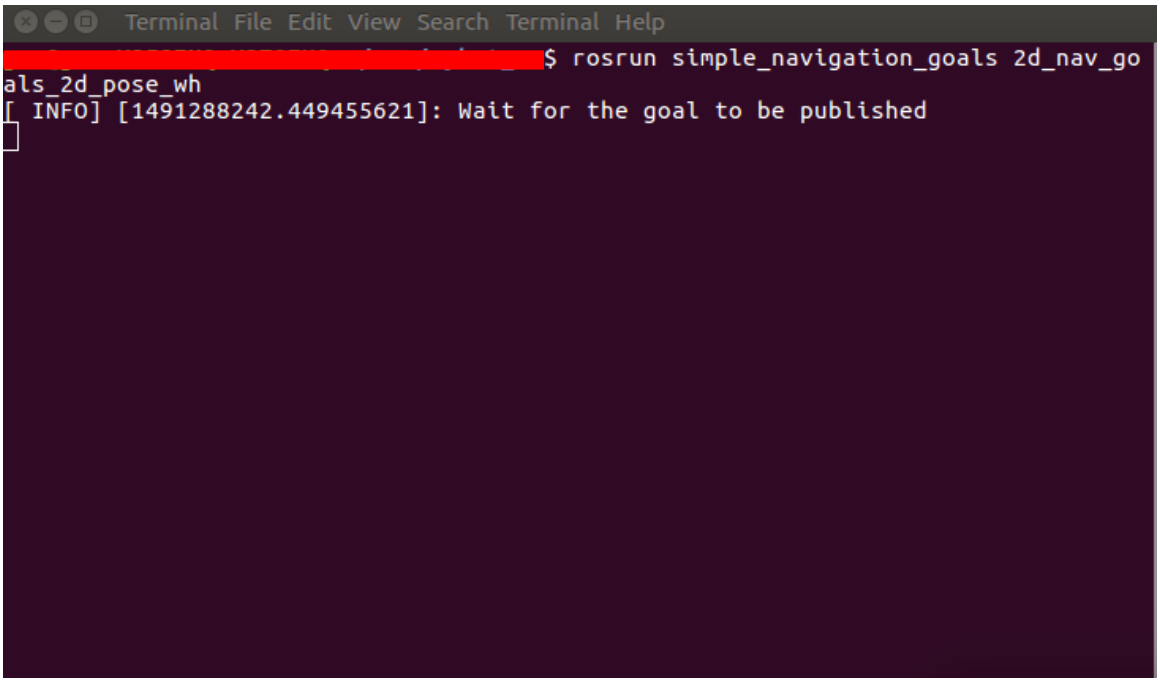
Όπως έχουμε αναφέρει στην ενότητα 4.3.3, όταν χρησιμοποιούμε το πλήκτρο 2D Pose Estimate του rviz εκδίδεται ένα μήνυμα στο θέμα /initialpose με την θέση και την γωνία που έχουμε επιλέξει εμείς στον χάρτη. Χρησιμοποιώντας αυτή την λειτουργία δημιουργήσαμε μια συνάρτηση τύπου callback που ενεργοποιείται κάθε φορά που εκδίδεται ένα μήνυμα σε αυτό το θέμα. Το πρόγραμμα ονομάζεται 2d_nav_goals_2d_pose_wh.cpp και αποτελεί την υλοποίηση ενός simple action client που στέλνει μηνύματα στον κόμβο move_base έτσι ώστε το IAV να εκτελεί συγκεκριμένες κινήσεις.
Στην πρώτη ενότητα δηλώνουμε τρεις μεταβλητές-πελάτες του namespace MoveBaseGoal και MoveBaseClient μέσω των οποίων θα στείλουμε τα δεδομένα στον κόμβο move_base και θα ορίσουμε τον κόμβο navigation_goals ως πελάτη. Επιπλέον ορίζουμε την συνάρτηση με την οποία θα εγγραφούμε στο θέμα /initialpose έτσι ώστε να σηματοδοτηθεί η έναρξη της callback συνάρτησης pose_callback. Η συνάρτηση αυτή κάθε φορα που εκτελείται λαμβάνει τα δεδομένα απο τον κόμβο /initialpose και τα μεταφέρει στην μεταβλητή goal. Στην συνέχεια η μεταβλητή αυτή αποστέλνεται στον κόμβο move_base. Αφού ολοκληρωθεί η αποστολή, δηλαδή το IAV φτάσει στον τελικό επιθυμητό στόχο μέσα απο το πρόγραμμα δίνεται η εντολή το IAV να κατευθυνθεί προς την έξοδο της αποθήκης. Τέλος με την προϋπόθεση έχει φτάσει στην έξοδο, δίνεται η εντολή να κατευθυνθεί στο σημείου που έχει οριστεί ως χώρος στάθμευσης και να παραμείνει εκεί μέχρι να το δοθεί νέα αποστολή. Το παραπάνω πρόγραμμα προσομοιώνει την διαδικασία της ανάθεσης ενός σημείου-στόχου της αποθήκης για την παραλαβή ενός πακέτου από τον χρήστη προς το IAV, την παράδοση του στον χρήστη και τέλος την κατεύθυνση του IAV προς τον χώρο στάθμευσης και αναμονή για νέα αποστολή.

### 5.3.2 Προσομοίωση

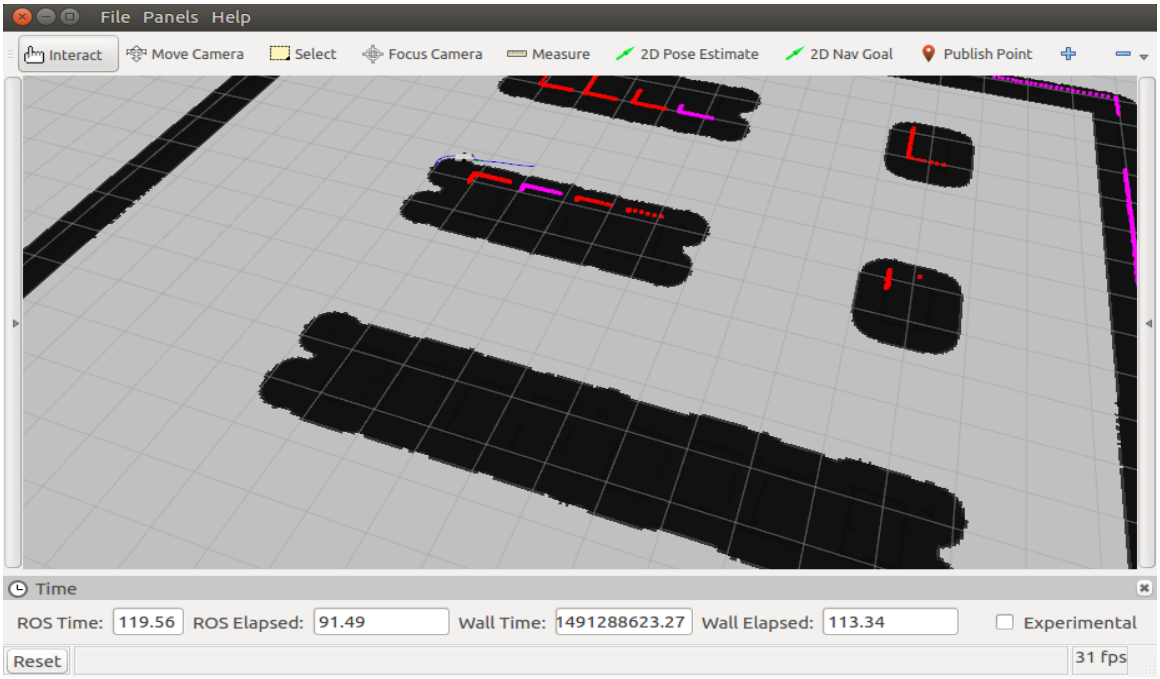
Ακολουθώντας ξανά την διαδικασία τς ενότητας 4.3.2 ανοίγουμε το gazebo ,το rviz καθώς και τον κώδικα για το amcl και επιπλέον εκτελούμε τον κωδικά της ενότητας 5.3.1.
Αρχικά το IAV βρίσκεται σε μια τυχαία θέση, ενώ ο κόμβος που έχουμε δημιουργεί αναμένει την εισαγωγή δεδομένων εικόνα 5.3.3. Όταν δοθεί η εντολή από το πλήκτρο 2D Pose Estimate εικόνα 5.3.2, το θέμα /initialpose εκδίδει δεδομένα και ενεργοποιείται η συνάρτηση pose_callback που έχουμε δημιουργήσει έτσι το IAV ξεκινάει προς τον στόχο που το έχουμε ορίσει, εικόνα 5.3.4.




*Εικόνα 5.3.2 Ορισμός στόχου μέσω του πλήκτρου 2D Pose Estimate*

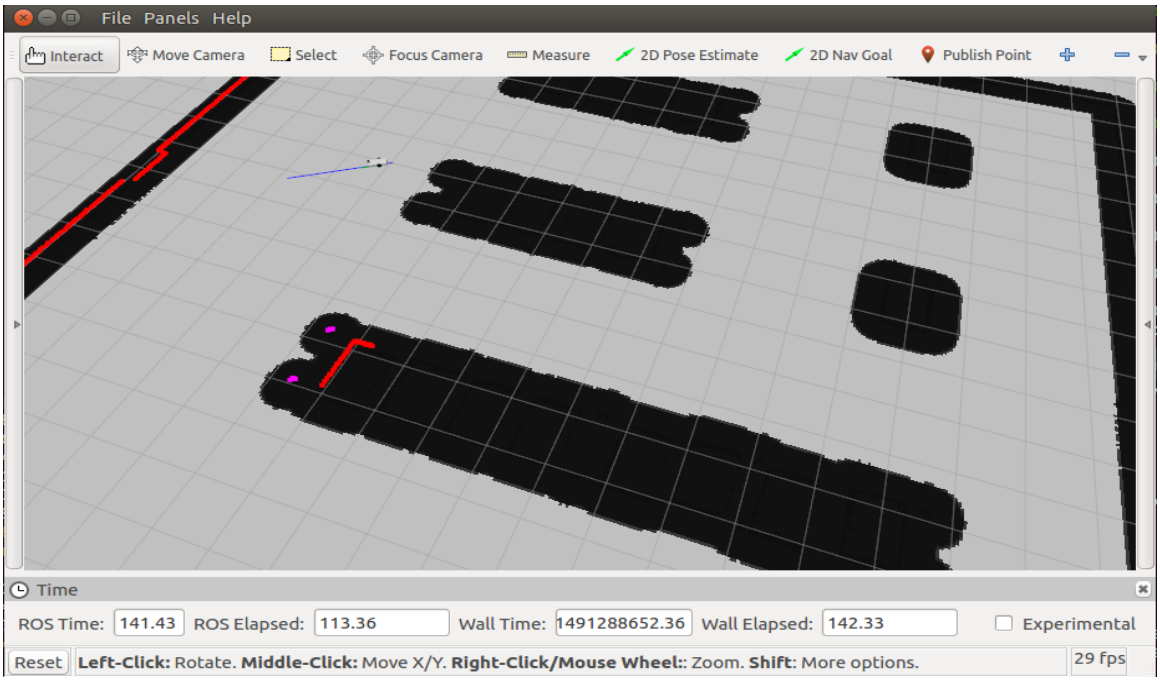



*Εικόνα 5.3.3 Αναμονή για την έκδοση δεδομένων*



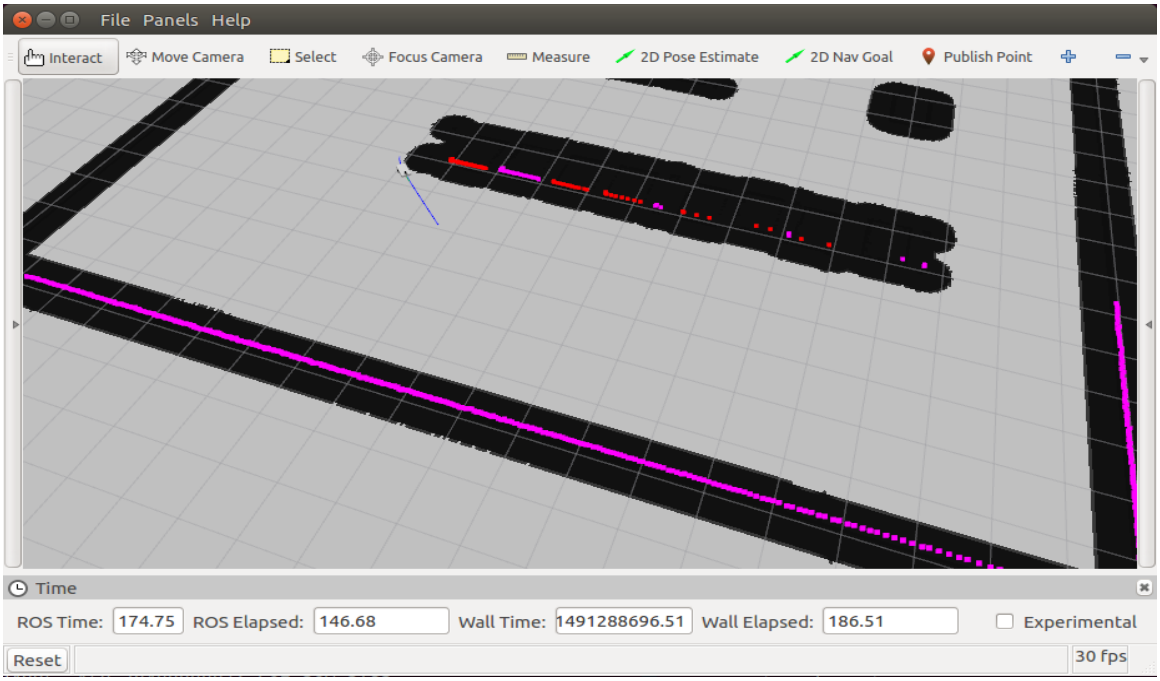

*Εικόνα 5.3.4 Κατεύθυνση προς τον τελικό στόχο*

Όταν το IAV φτάσει στον στόχο αναμένει πέντε δευτερόλεπτα, έπειτα κατευθύνεται προς στην έξοδο, εικόνα 5.3.5

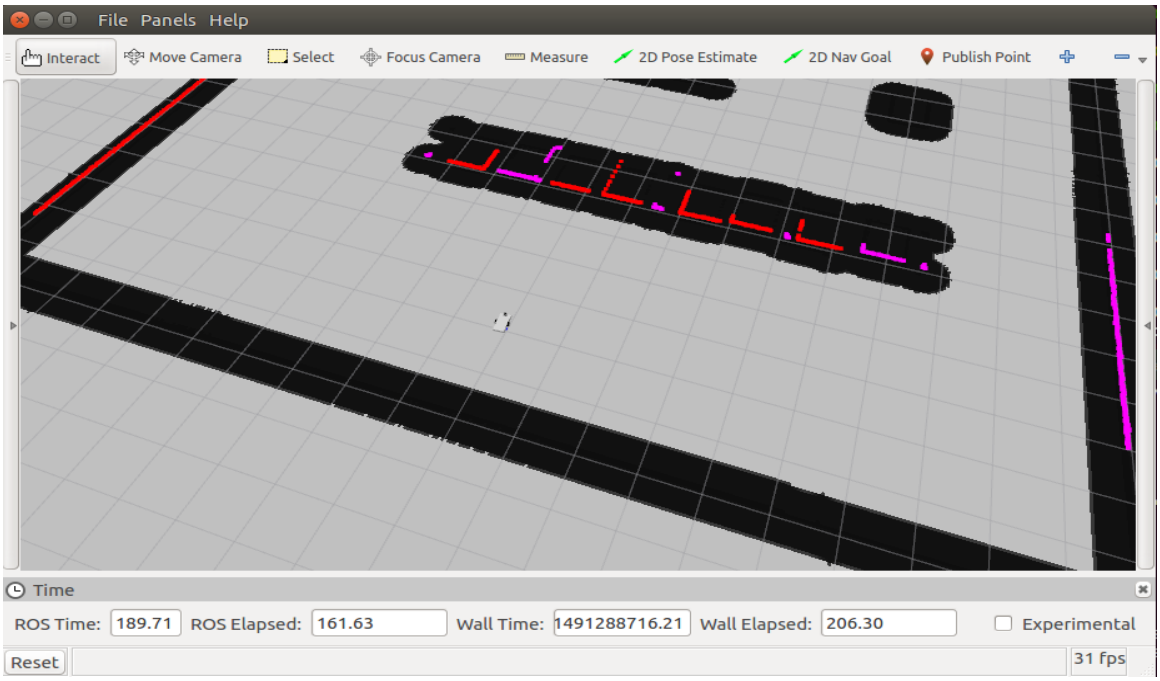



*Εικόνα 5.3.5 Κατεύθυνση προς την έξοδο*

Μετά το πέρας των πέντε δευτερολέπτων το IAV κατευθύνεται προς την θέση στάθμευσης και αναμένει την έκδοση μιας νέας αποστολής εικόνες 5.3.6 και 5.3.7 αντίστοιχα.

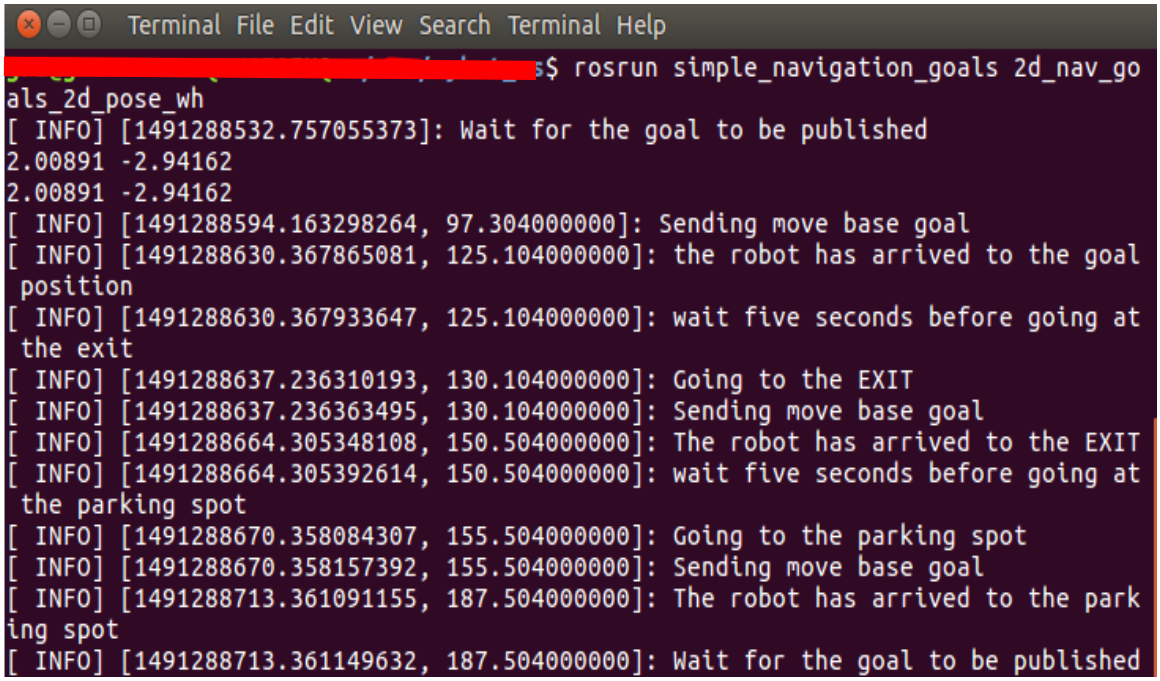



*Εικόνα 5.3.6 Κατεύθυνση προς την θέση στάθμευσης*




*Εικόνα 5.3.7 Αναμονή για την έκδοση νέας αποστολής*

Στον τερματικό που εκτελείται ο simple action client αναγράφεται η κατάσταση του IAV κάθε φορά που υπάρχει αλλαγή της, εικόνα 5.3.8.



*Εικόνα 5.3.8 Αναφορά κατάστασης*

## 5.4 Συμπεράσματα και μελλοντικές επεκτάσεις

### 5.4.1 Συμπεράσματα

Τα αποτελέσματα από τις δοκιμές των δύο αλγορίθμων ήταν πολύ ικανοποιητικά και μπορούν να συμβάλουν στον προγραμματισμό των IAVs για την καλύτερη και πιο αξιόπιστη λειτουργία τους στην διεκπεραίωση αποστολών. Μετά την χρήση και έρευνα στο ROS για τους σκοπούς της εργασίας έχουν παρατηρηθεί τα εξής:

- Είναι ικανό να οδηγήσει ένα IAV με ακρίβεια παρά τις τυχόν ανωμαλίες του εδάφους υπό δύσκολες συνθήκες.
- Το ROS απασχολεί όλο και περισσότερο την επιστημονική κοινότητα διότι αποτελεί ένα πανίσχυρο εργαλείο για τον προγραμματισμό ρομποτικών συστημάτων, όλο και περισσότερα ερευνητικά ινστιτούτα χρησιμοποιούν το χρησιμοποιούν σε ρομποτικά συστήματα.
- Δίνει την δυνατότητα αυτοματοποίησης διεργασιών με χρήση προγραμμάτων σε C++ και Python
- Μας παρέχει μεγάλο εύρος αλγορίθμων και βιβλιοθηκών που χρησιμοποιούνται ως βάση για την ανάπτυξη νέων προγραμμάτων
- Έχει μεγάλη καμπύλη εκμάθησης
- Δεν έχει φιλικό περιβάλλον διεπαφής για τους χρήστες που ξεκινάνε να εργάζονται

### 5.4.2 Μελλοντικές επεκτάσεις

Τα IAVs αποκτούν όλο και μεγαλύτερο ρόλο στην βιομηχανία καθώς η τεχνολογία παρουσιάζει ραγδαία εξέλιση. Από την απλή ανάθεση διαδρομών πλέον έχουμε την δυνατότητα να προγραμματίσουμε τα οχήματα αυτά να εκτελούν πολύπλοκες ενέργειες με μεγάλη ακρίβεια για την εξυπηρέτησή των αναγκών μας. Ο τομέας της τεχνητής υπόσχεται μια σημαντική προσθήκη στα IAVs ώστε με την επεξεργασία από τα δεδομένα των αισθητήρων (εικόνα) να αναγνωρίζουν αντικείμενα όπως και το περιβάλλον στο οποίο βρίσκονται και να λαμβάνουν αποφάσεις σε πραγματικό χρόνο. Επιπλέον σε ερευνητικό η ανάπτυξη είτε νέων είτε ήδη υπάρχων αλγορίθμων για την χαρτογράφηση μιας περιοχής με παραπάνω από ένα IAV με σκοπό την μείωση του χρονικού εργασίας. Τέλος η εφαρμογή της παραπάνω τεχνογνωσίας σε πραγματικά IAV

**Σημείωση**: Οι κώδικές και τα αρχεία που αναφέρονται στο Κεφάλαιο 5 εμπεριέχονται στο Παράρτημα Γ.

## Παράρτημα Α

## Δημιουργία ενός IAV

### Αρχείο mvrobot.xacro

```
<?xml version="1.0"?>
<robot name="mvrobot" xmlns:xacro="http://www.ros.org/wiki/xacro">


  <xacro:include filename="$(find mvrobot_description)/urdf/wheel.urdf.xacro" />


<!-- Defining the colors used in this robot -->

  <material name="Black">
    <color rgba="0.0 0.0 0.0 1.0"/>
  </material>

  <material name="Red">
    <color rgba="0.8 0.0 0.0 1.0"/>
  </material>

  <material name="White">
    <color rgba="1.0 1.0 1.0 1.0"/>
  </material>


  <material name="Blue">
    <color rgba="0.0 0.0 0.8 1.0"/>
  </material>



  <!-- PROPERTY LIST -->
  <!--All units in m-kg-s-radians unit system -->

  <property name="M_PI" value="3.1415926535897931" />

  <!-- Main body radius and height -->
  <!-- Main Body box base   -->

  <property name="base_height" value="0.05" />
  <property name="base_length" value="0.40" />
  <property name="base_wirth" value="0.20" />
  <property name="base_mass" value="5" /> <!-- in kg-->



  <!-- caster wheel radius and height -->
  <!-- caster wheel mass -->

  <property name="caster_f_height" value="0.02" />
  <property name="caster_f_radius" value="0.025" />
  <property name="caster_f_mass" value="0.5" /> <!-- in kg-->



  <!-- caster wheel radius and height -->
  <!-- caster wheel mass -->
```

```
  <property name="caster_b_height" value="0.02" />
  <property name="caster_b_radius" value="0.025" />
  <property name="caster_b_mass" value="0.5" /> <!-- in kg-->



  <!-- Wheels -->

  <property name="wheel_mass" value="2.5" /> --> <!-- in kg-->

  <property name="base_x_origin_to_wheel_origin" value="0.25" />
  <property name="base_y_origin_to_wheel_origin" value="0.3" />
  <property name="base_z_origin_to_wheel_origin" value="0.0" />


  <!-- Hokuyo Laser scanner -->

  <property name="hokuyo_size" value="0.05" />



  <!-- Macro for calculating inertia of the box -->

   <macro name="box_inertia" params="m x y z">
    <inertia  ixx="${m*(y*y+z*z)/12}" ixy = "0" ixz = "0"
      iyy="${m*(x*x+z*z)/12}" iyz = "0"
      izz="${m*(x*x+z*z)/12}"/>
  </macro>


  <!-- BASE-FOOTPRINT -->
  <!-- base_footprint is a fictitious link(frame) that is on the ground right below base_link origin -->

 <link name="base_footprint">
    <inertial>
      <mass value="0.0001" />
      <origin xyz="0 0 0" />
      <inertia ixx="0.0001" ixy="0.0" ixz="0.0"
          iyy="0.0001" iyz="0.0"
          izz="0.0001" />
    </inertial>
    <visual>
       <origin xyz="0 0 0" rpy="0 0 0" />
       <geometry>
          <box size="0.001 0.001 0.001" />
       </geometry>
    </visual>
  </link>

  <gazebo reference="base_footprint">
    <turnGravityOff>false</turnGravityOff>
  </gazebo>

  <joint name="base_footprint_joint" type="fixed">
    <origin xyz="0 0 ${wheel_radius - base_z_origin_to_wheel_origin}" rpy="0 0 0" />
    <parent link="base_footprint"/>
    <child link="base_link" />
  </joint>
```

```
    <!-- BASE-LINK -->
    <!--Actual body/chassis of the robot-->
    <link name="base_link">
      <inertial>
        <mass value="${base_mass}" />
        <origin xyz="0 0 0" />

        <!--The 3x3 rotational inertia matrix. -->
        <cylinder_inertia  m="${base_mass}" r="${base_length}" h="${base_height}" />

    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
         <box size="${base_length} ${base_wirth} ${base_height}"/>
            </geometry>
      <material name="White" />
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0 " />
      <geometry>

<box size="${base_length} ${base_wirth} ${base_height}"/>

      </geometry>
    </collision>
  </link>
  <gazebo reference="base_link">
    <material>Gazebo/White</material>
    <turnGravityOff>false</turnGravityOff>
  </gazebo>

  <!--Caster front  -->

      <link name="caster_front_link">

      <visual>
        <origin xyz="0.00 0.02 0" rpy="${M_PI/2} 0 0" />
        <geometry>
             <sphere radius="${caster_f_radius}" />
        </geometry>
        <material name="Black" />
      </visual>
             <collision>
          <geometry>

             <sphere radius="${caster_f_radius+0.005}" />

          </geometry>
        <origin xyz="0.00 0.02 0" rpy="${M_PI/2} 0 0" />
        </collision>
        <inertial>
          <mass value="${caster_f_mass}" />
          <origin xyz="0 0 0" />
          <inertia ixx="0.001" ixy="0.0" ixz="0.0"
                   iyy="0.001" iyz="0.0"
                   izz="0.001" />
        </inertial>
      </link>
```

```
 <joint name="caster_front_joint" type="fixed">
     <parent link="base_link"/>
     <child link="caster_front_link"/>
     <origin xyz="0.135 0.0 0" rpy="${-M_PI/2} 0 0"/>
   </joint>


  <gazebo reference="caster_front_link">
   <turnGravityOff>false</turnGravityOff>
  </gazebo>

<!--Caster back  -->

   <link name="caster_back_link">

   <visual>
    <origin xyz="0.00 0.02 0 " rpy="${M_PI/2} 0 0" />

    <geometry>


         <sphere radius="${caster_b_radius}" />


    </geometry>
    <material name="Black" />
   </visual>



    <collision>
     <geometry>
        <sphere radius="${caster_b_radius}" />
     </geometry>
    <origin xyz="0 0.02 0 " rpy="${M_PI/2} 0 0" />
    </collision>
    <inertial>
     <mass value="${caster_b_mass}" />
     <origin xyz="0 0 0" />
     <inertia ixx="0.001" ixy="0.0" ixz="0.0"
              iyy="0.001" iyz="0.0"
              izz="0.001" />
    </inertial>
   </link>

   <joint name="caster_back_joint" type="fixed">
    <parent link="base_link"/>
    <child link="caster_back_link"/>
    <origin xyz="-0.135 0.0 -0.005" rpy="${-M_PI/2} 0 0"/>
   </joint>


  <gazebo reference="caster_back_link">
   <turnGravityOff>false</turnGravityOff>
  </gazebo>


<!-- Wheel Definitions -->

  <wheel fb="front" lr="right" parent="base_link" translateX="0" translateY="0.375" flipY="1"/>
```

```
<wheel fb="front" lr="left" parent="base_link" translateX="0" translateY="-0.375" flipY="1"/>

<!-- SENSORS -->

<!-- hokuyo -->

<link name="hokuyo_link">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
         <mesh filename="package://mvrobot_description/meshes/hokuyo.dae"/>
      <box size="${hokuyo_size} ${hokuyo_size} ${hokuyo_size}"/>
         </geometry>
    <material name="Blue" />
  </visual>
</link>

<joint name="hokuyo_joint" type="fixed">
  <origin xyz="${base_length/2 - hokuyo_size/2} 0 ${base_height+hokuyo_size/4}" rpy="0 0 0" />
  <parent link="base_link"/>
  <child link="hokuyo_link" />
</joint>



  <inertial>
    <mass value="1e-5" />
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
  </inertial>


<gazebo reference="hokuyo_link">
  <material>Gazebo/Blue</material>
  <turnGravityOff>false</turnGravityOff>
  <sensor type="ray" name="head_hokuyo_sensor">
    <pose>${hokuyo_size/2} 0 0 0 0 0</pose>
    <visualize>true</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-1.570796</min_angle>
          <max_angle>1.570796</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.10</min>
        <max>30.0</max>
        <resolution>0.01</resolution>
      </range>
          <noise>
        <type>gaussian</type>
        <!-- Noise parameters based on published spec for Hokuyo laser
             achieving "+-30mm" accuracy at range < 10m.  A mean of 0.0m and
             stddev of 0.01m will put 99.7% of samples within 0.03m of the true
             reading. -->
        <mean>0.0</mean>
```

```
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
      <topicName>/scan</topicName>
      <frameName>hokuyo_link</frameName>
    </plugin>
  </sensor>
</gazebo>


<!-- Differential drive controller  -->

<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">

    <rosDebugLevel>Debug</rosDebugLevel>
    <publishWheelTF>false</publishWheelTF>
    <robotNamespace>/</robotNamespace>
    <publishTf>1</publishTf>
    <publishWheelJointState>false</publishWheelJointState>
    <alwaysOn>true</alwaysOn>
    <updateRate>100.0</updateRate>
    <leftJoint>front_left_wheel_joint</leftJoint>
    <rightJoint>front_right_wheel_joint</rightJoint>
    <wheelSeparation>${2*base_length}</wheelSeparation>
    <wheelDiameter>${2*wheel_radius}</wheelDiameter>
    <broadcastTF>1</broadcastTF>
    <wheelTorque>30</wheelTorque>
    <wheelAcceleration>1.8</wheelAcceleration>
    <commandTopic>cmd_vel</commandTopic>
    <odometryFrame>odom</odometryFrame>
    <odometryTopic>odom</odometryTopic>
    <robotBaseFrame>base_footprint</robotBaseFrame>


  </plugin>
</gazebo>

</robot>
```

**Αρχείο weel.xacro**

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

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

  <!-- Wheels -->

  <property name="wheel_radius" value="0.04" />

  <property name="wheel_height" value="0.02" />

  <property name="wheel_mass" value="2.5" /> <!-- in kg-->
```

```
  <property name="base_x_origin_to_wheel_origin" value="0.25" />

  <property name="base_y_origin_to_wheel_origin" value="0.3" />

  <property name="base_z_origin_to_wheel_origin" value="0.0" />

  <macro name="cylinder_inertia" params="m r h">

    <inertia  ixx="${m*(3*r*r+h*h)/12}" ixy = "0" ixz = "0"
              iyy="${m*(3*r*r+h*h)/12}" iyz = "0"
              izz="${m*r*r/2}" />

  </macro>

  <xacro:macro name="wheel" params="fb lr parent translateX translateY flipY"> <!--fb : front, back;
lr:left, right -->

    <link name="${fb}_${lr}_wheel">

      <visual>

        <origin xyz="0 0 0" rpy="${flipY*M_PI/2} 0  0 " />

        <geometry>

          <cylinder length="${wheel_height}" radius="${wheel_radius}" />

        </geometry>

        <material name="DarkGray" />

      </visual>

      <collision>

        <origin xyz="0 0 0" rpy="${flipY*M_PI/2} 0 0 " />

        <geometry>

          <cylinder length="${wheel_height}" radius="${wheel_radius}" />

        </geometry>

      </collision>

      <inertial>

        <mass value="${wheel_mass}" />

        <origin xyz="0 0 0" />

        <cylinder_inertia  m="${wheel_mass}" r="${wheel_radius}" h="${wheel_height}" />

      </inertial>

    </link>

    <gazebo reference="${fb}_${lr}_wheel">

      <mu1 value="1.0"/>
```

```
      <mu2 value="1.0"/>
      <kp  value="10000000.0" />
      <kd  value="1.0" />
      <fdir1 value="1 0 0"/>
      <material>Gazebo/Grey</material>
      <turnGravityOff>false</turnGravityOff>
    </gazebo>
    <joint name="${fb}_${lr}_wheel_joint" type="continuous">
      <parent link="${parent}"/>
      <child link="${fb}_${lr}_wheel"/>
                  <origin xyz="${translateX * base_x_origin_to_wheel_origin} ${translateY *
base_y_origin_to_wheel_origin} ${base_z_origin_to_wheel_origin}" rpy="0 0 0" />
      <axis xyz="0 1 0" rpy="0  0" />
      <limit effort="100" velocity="100"/>
      <joint_properties damping="0.0" friction="0.0"/>
    </joint>
    <!-- Transmission is important to link the joints and the controller -->
    <transmission name="${fb}_${lr}_wheel_joint_trans">
      <type>transmission_interface/SimpleTransmission</type>
      <joint name="${fb}_${lr}_wheel_joint" />
      <actuator name="${fb}_${lr}_wheel_joint_motor">
        <hardwareInterface>EffortJointInterface</hardwareInterface>
        <mechanicalReduction>1</mechanicalReduction>
      </actuator>
    </transmission>
  </xacro:macro>
</robot>
```

**Αρχείο mvrobot_world.launch**

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
```

```
  <!-- these are the arguments you can pass this launch file, for example paused:=true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>
  <!-- We resume the logic in empty_world.launch -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>
  <!-- urdf xml robot description loaded on the Parameter Server-->
        <param name="robot_description" command="$(find xacro)/xacro.py '$(find
mvrobot_description)/urdf/mvrobot.xacro'" />
  <!-- Run a python script to the send a service call to gazebo_ros to spawn a URDF robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
output="screen"
args="-urdf -model mvrobot -param robot_description"/>
</launch>
```

**Αρχείο mvrobot_teleop.launch**

```
<?xml version="1.0"?>
<launch>
  <!-- turtlebot_teleop_key already has its own built in velocity smoother -->
  <node pkg="turtlebot_teleop" type="turtlebot_teleop_key" name="turtlebot_teleop_keyboard"
output="screen">
    <param name="scale_linear" value="0.5" type="double"/>
    <param name="scale_angular" value="1.5" type="double"/>
    <remap from="turtlebot_teleop_keyboard/cmd_vel" to="cmd_vel"/>
```

```
  </node>
</launch>
```

## Παράρτημα Β

## Διαμόρφωση παραμέτρων του Navigation Stack

### Αρχείο gmapping.launch

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
  <arg name="scan_topic" default="scan" />

  <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping" output="screen">
    <param name="base_frame" value="base_footprint"/>
    <param name="odom_frame" value="odom"/>
    <param name="map_update_interval" value="5.0"/>
    <param name="maxUrange" value="20.0"/>
    <param name="maxRange" value="20.0"/>
    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="lskip" value="0"/>
    <param name="minimumScore" value="100"/>
    <param name="srr" value="0.01"/>
    <param name="srt" value="0.02"/>
    <param name="str" value="0.01"/>
    <param name="stt" value="0.02"/>
    <param name="linearUpdate" value="0.5"/>
    <param name="angularUpdate" value="0.436"/>
    <param name="temporalUpdate" value="-1.0"/>
    <param name="resampleThreshold" value="0.5"/>
    <param name="particles" value="80"/>

    <param name="xmin" value="-1.0"/>
    <param name="ymin" value="-1.0"/>
    <param name="xmax" value="1.0"/>
    <param name="ymax" value="1.0"/>

    <param name="delta" value="0.05"/>
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/>
    <remap from="scan" to="$(arg scan_topic)"/>
  </node>



      <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">

          <rosparam file="$(find mvrobot_navigation)/param/costmap_common_params.yaml" command="load" ns="global_costmap" />

          <rosparam file="$(find mvrobot_navigation)/param/costmap_common_params.yaml" command="load" ns="local_costmap" />
```

```
        <rosparam file="$(find mvrobot_navigation)/param/local_costmap_params.yaml"
command="load" />

        <rosparam file="$(find mvrobot_navigation)/param/global_costmap_params.yaml"
command="load" />

        <rosparam file="$(find mvrobot_navigation)/param/base_local_planner_params.yaml"
command="load" />

        <rosparam file="$(find mvrobot_navigation)/param/dwa_local_planner_params.yaml"
command="load" />

        <rosparam file="$(find mvrobot_navigation)/param/move_base_params.yaml"
command="load" />


  </node>

</launch>
```

**Αρχείο base_local_planner_params.yaml**

```
TrajectoryPlannerROS:

# Robot Configuration Parameters
  max_vel_x: 0.5
  min_vel_x: 0.01

  max_vel_theta:  1.0
  min_vel_theta: -1.0
  min_in_place_vel_theta: 0.1

  acc_lim_x: 0.5
  acc_lim_y: 0.5
  acc_lim_theta: 1.0

# Goal Tolerance Parameters
  yaw_goal_tolerance: 0.3
  xy_goal_tolerance: 0.2

# Forward Simulation Parameters
  sim_time: 3.0
  vx_samples: 6
  vtheta_samples: 20

# Trajectory Scoring Parameters
  meter_scoring: true
  pdist_scale: 0.6
  gdist_scale: 0.8
        occdist_scale: 0.01
  heading_lookahead: 0.325
  dwa: true

# Oscillation Prevention Parameters
  oscillation_reset_dist: 0.05

# Differential-drive robot configuration
  holonomic_robot: false
  max_vel_y: 0.0
```

min_vel_y: 0.0
acc_lim_y: 0.0
vy_samples: 1

**Αρχείο costmap_common_params.yaml**

max_obstacle_height: 0.60
obstacle_range: 2.5
raytrace_range: 3.0

robot_radius: 0.4509

inflation_radius: 0.50 # max. distance from an obstacle at which costs are incurred for planning paths.
cost_scaling_factor: 5 # exponential rate at which the obstacle cost drops off (default: 10)

map_type: costmap
transform_tolerance: 5 # seconds

origin_z: 0.0
z_resolution: 0.2
z_voxels: 2
publish_voxel_map: false

observation_sources: laser_scan_sensor

laser_scan_sensor: {sensor_frame: hokuyo, data_type: LaserScan, topic: /scan, marking: true, clearing: true}

**Αρχείο dwa_local_planner_params.yaml τοποθετούμε**

DWAPlannerROS:

max_vel_x: 0.5 # 0.55
min_vel_x: 0.0

max_vel_y: 0.0 # diff drive robot
min_vel_y: 0.0 # diff drive robot

max_trans_vel: 0.5
min_trans_vel: 0.1 # this is the min trans velocity when there is negligible rotational velocity
trans_stopped_vel: 0.1

max_rot_vel: 5.0
min_rot_vel: 0.4 # this is the min angular velocity when there is negligible translational velocity
rot_stopped_vel: 0.4

```
  acc_lim_x: 1.0 # maximum is theoretically 2.0
  acc_lim_theta: 2.0
  acc_lim_y: 0.0      # diff drive robot

 # Goal Tolerance Parameters

  yaw_goal_tolerance: 0.3  # 0.05
  xy_goal_tolerance: 0.15  # 0.10


 # Forward Simulation Parameters

  sim_time: 1.0       # 1.7
  vx_samples: 6       # 3
  vy_samples: 1
  vtheta_samples: 20

 # Trajectory Scoring Parameters

  path_distance_bias: 64.0      # 32.0   - weighting for how much it should stick to the global path plan
  goal_distance_bias: 24.0      # 24.0   - wighting for how much it should attempt to reach its goal
  occdist_scale: 0.50           # 0.01   - weighting for how much the controller should avoid obstacles
  forward_point_distance: 0.325 # 0.325  - how far along to place an additional scoring point
  stop_time_buffer: 0.2         # 0.2    - amount of time a robot must stop in before colliding for a valid
traj.
  scaling_speed: 0.25           # 0.25   - absolute velocity at which to start scaling the robot's footprint
  max_scaling_factor: 0.2       # 0.2    - how much to scale the robot's footprint when at speed.

 # Oscillation Prevention Parameters

  oscillation_reset_dist: 0.05  # 0.05   - how far to travel before resetting oscillation flags

 # Debugging

 publish_traj_pc : true
 publish_cost_grid_pc: true
 global_frame_id: odom
```

**Αρχείο global_costmap_params.yaml**

```
 global_costmap:
  global_frame: /map
  robot_base_frame: /base_link
  update_frequency: 1.0
  publish_frequency: 0.5
  static_map: true
  transform_tolerance: 0.5
  width: 10.0
  height: 10.0
```

Αρχείο local_costmap_params.yaml

```
 local_costmap:
   global_frame: odom
   robot_base_frame: /base_footprint
   update_frequency: 5.0
   publish_frequency: 2.0
   static_map: false
```

```
  rolling_window: true
  width: 4.0
  height: 4.0
  resolution: 0.05
  transform_tolerance: 0.5
```

**Αρχείο move_base_params.yaml**

```
#  http://www.ros.org/wiki/move_base

shutdown_costmaps: false

controller_frequency: 5.0
controller_patience: 3.0

planner_frequency: 1.0
planner_patience: 5.0

oscillation_timeout: 10.0
oscillation_distance: 0.2

# local planner - default is trajectory rollout
base_local_planner: "dwa_local_planner/DWAPlannerROS"
```

**Αρχείο mvrobot_rviz_gmapping.launch**

```
<?xml version="1.0"?>
<launch>
	<arg name="model" />

	<!-- Parsing xacro and setting mvrobot_description parameter -->

	<param name="mvrobot_description_param" command="$(find
xacro)/xacro.py $(find mvrobot_description)/urdf/mvrobot.xacro" />

	<!-- Setting gui parameter to true for display joint slider -->

	<param name="use_gui" value="true"/>

	<!-- Starting Joint state publisher node which will publish the joint values -->

	<node name="joint_state_publisher" pkg="joint_state_publisher"
	type="joint_state_publisher" />

	<!-- Starting robot state publish which will publish tf -->

	<node name="robot_state_publisher" pkg="robot_state_publisher"
	type="state_publisher" />

	<!-- Launch visualization in rviz -->

<node name="rviz" pkg="rviz" type="rviz" args="-d $(find mvrobot_description)/gmapping.rviz"
required="true" />
	</launch>
```

**Αρχείο gmapping.rviz**

```
Panels:
  - Class: rviz/Displays
    Help Height: 78
    Name: Displays
    Property Tree Widget:
      Expanded:
        - /Global Options1
        - /Status1
        - /Grid1
        - /Map1
        - /LaserScan1
      Splitter Ratio: 0.5
    Tree Height: 565
  - Class: rviz/Selection
    Name: Selection
  - Class: rviz/Tool Properties
    Expanded:
      - /2D Pose Estimate1
      - /2D Nav Goal1
      - /Publish Point1
    Name: Tool Properties
    Splitter Ratio: 0.588679016
  - Class: rviz/Views
    Expanded:
      - /Current View1
    Name: Views
    Splitter Ratio: 0.5
  - Class: rviz/Time
    Experimental: false
    Name: Time
    SyncMode: 0
    SyncSource: LaserScan
Visualization Manager:
  Class: ""
  Displays:
    - Alpha: 0.5
      Cell Size: 1
      Class: rviz/Grid
      Color: 160; 160; 164
      Enabled: true
      Line Style:
        Line Width: 0.0299999993
        Value: Lines
      Name: Grid
      Normal Cell Count: 0
      Offset:
        X: 0
        Y: 0
        Z: 0
      Plane: XY
      Plane Cell Count: 30
      Reference Frame: <Fixed Frame>
      Value: true
    - Alpha: 1
      Class: rviz/RobotModel
      Collision Enabled: false
      Enabled: true
      Links:
```

```
      All Links Enabled: true
      Expand Joint Details: false
      Expand Link Details: false
      Expand Tree: false
      Link Tree Style: Links in Alphabetic Order
      base_footprint:
        Alpha: 1
        Show Axes: false
        Show Trail: false
        Value: true
      base_link:
        Alpha: 1
        Show Axes: false
        Show Trail: false
        Value: true
      caster_back_link:
        Alpha: 1
        Show Axes: false
        Show Trail: false
        Value: true
      caster_front_link:
        Alpha: 1
        Show Axes: false
        Show Trail: false
        Value: true
      front_left_wheel:
        Alpha: 1
        Show Axes: false
        Show Trail: false
        Value: true
      front_right_wheel:
        Alpha: 1
        Show Axes: false
        Show Trail: false
        Value: true
      hokuyo_link:
        Alpha: 1
        Show Axes: false
        Show Trail: false
        Value: true
    Name: RobotModel
    Robot Description: robot_description
    TF Prefix: ""
    Update Interval: 0
  Value: true
  Visual Enabled: true
- Alpha: 0.699999988
  Class: rviz/Map
  Color Scheme: map
  Draw Behind: false
  Enabled: true
  Name: Map
  Topic: /map
  Unreliable: false
  Value: true
- Alpha: 1
  Autocompute Intensity Bounds: true
  Autocompute Value Bounds:
    Max Value: 10
    Min Value: -10
```

```
    Value: true
  Axis: Z
  Channel Name: intensity
  Class: rviz/LaserScan
  Color: 255; 255; 255
  Color Transformer: Intensity
  Decay Time: 0
  Enabled: true
  Invert Rainbow: false
  Max Color: 255; 255; 255
  Max Intensity: 999999
  Min Color: 0; 0; 0
  Min Intensity: 0
  Name: LaserScan
  Position Transformer: XYZ
  Queue Size: 10
  Selectable: true
  Size (Pixels): 3
  Size (m): 0.0799999982
  Style: Flat Squares
  Topic: /scan
  Unreliable: false
  Use Fixed Frame: true
  Use rainbow: true
  Value: true
Enabled: true
Global Options:
  Background Color: 48; 48; 48
  Fixed Frame: odom
  Frame Rate: 30
Name: root
Tools:
  - Class: rviz/Interact
    Hide Inactive Objects: true
  - Class: rviz/MoveCamera
  - Class: rviz/Select
  - Class: rviz/FocusCamera
  - Class: rviz/Measure
  - Class: rviz/SetInitialPose
    Topic: /initialpose
  - Class: rviz/SetGoal
    Topic: /move_base_simple/goal
  - Class: rviz/PublishPoint
    Single click: true
    Topic: /clicked_point
Value: true
Views:
  Current:
    Class: rviz/Orbit
    Distance: 28.0668468
    Enable Stereo Rendering:
      Stereo Eye Separation: 0.0599999987
      Stereo Focal Distance: 1
      Swap Stereo Eyes: false
      Value: false
    Focal Point:
      X: 0
      Y: 0
      Z: 0
    Focal Shape Fixed Size: false
```

```
      Focal Shape Size: 0.0500000007
      Name: Current View
      Near Clip Distance: 0.00999999978
      Pitch: 0.785398185
      Target Frame: <Fixed Frame>
      Value: Orbit (rviz)
      Yaw: 0.785398185
    Saved: ~
  Window Geometry:
  Displays:
    collapsed: false
  Height: 846
    Hide Left Dock: false
    Hide Right Dock: false
  QmainWindowState:
000000ff00000000fd0000000400000000000016a000002c4fc0200000008fb0000001200530065006c00650
06300740069006f006e00000001e10000009b0000006400ffffffb0000001e0054006f006f006c0020005000720
06f007000650072007400690065007302000001ed000001df00000185000000a3fb0000012005600690065 0
07700730020005400 6f006f02000001df000002110000018500000122fb000000200054006f006f006c002000
500072006f007000650072007400690065007300320300000288000001 1d000002210000017afb000000100
044006900730070006c006100790073010000002800000 2c4000000dd00ffffffb0000002000730065006c006
500630074006900 6f006e0020006200750066006600650072020000013800000 0aa0000023a00000294fb00
000014005700690064006500530074006500720065006f02000000e6000000d2000003ee0000030bfb00000
00c004b0069006e006500630074020000018600000106000003 0c0000026100000001000001 0f000002c4fc0
200000003fb0000001e0054006f006f006c00200050007200 6f007000650072007400690065007301000000 4
1000000780000000000000000fb0000000a00560069006500770073010000002800000 2c4000000b000ffffffb
0000001200530065006c006500630074006900 6f006e010000025a000000b2000000000000000000000020
0000490000000a9fc0100000001fb0000000a00560069006500770073030000004e00000080000002e10000
01970000000300000 4b00000003efc0100000002fb0000000800540069006d006501000000000000 04b0000
0030000ffffffb0000000800540069006d006501000000000000045000000000000000000000022b000002c40
0000004000000040000000800000008fc0000000100000002000000010000000a0054006f006f006c0073010
0000000ffffffff0000000000000000
  Selection:
    collapsed: false
  Time:
    collapsed: false
  Tool Properties:
    collapsed: false
  Views:
    collapsed: false
  Width: 1200
  X: 486
  Y: 96
```

**Αρχείο amcl.launch**

```
<?xml version="1.0"?>
<launch>
  <master auto="start"/>

  <!-- Map server -->

  <arg name="map_file" default="$(find mvrobot_navigation)/maps/test_map.yaml"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />

  <!-- Place map frame at odometry frame -->

  <node pkg="tf" type="static_transform_publisher" name="map_odom_broadcaster"
```

```
    args="0 0 0 0 0 0 /map /odom 100"/>

  <!-- Localization -->

 <include file= "$(find mvrobot_navigation)/launch/includes/amcl.launch.xml"/>


  <!-- Move base -->

       <node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">
            <rosparam file="$(find mvrobot_navigation)/param/costmap_common_params.yaml"
command="load" ns="global_costmap" />
            <rosparam file="$(find mvrobot_navigation)/param/costmap_common_params.yaml"
command="load" ns="local_costmap" />
              <rosparam file="$(find mvrobot_navigation)/param/local_costmap_params.yaml"
command="load" />
             <rosparam file="$(find mvrobot_navigation)/param/global_costmap_params.yaml"
command="load" />
           <rosparam file="$(find mvrobot_navigation)/param/base_local_planner_params.yaml"
command="load" />
            <rosparam file="$(find mvrobot_navigation)/param/dwa_local_planner_params.yaml"
command="load" />


<remap from="cmd_vel" to="cmd_vel"/>
<remap from="odom" to="odom"/>
<remap from="scan" to="/scan"/>
<param name="move_base/DWAPlannerROS/yaw_goal_tolerance" value="1.0"/>
<param name="move_base/DWAPlannerROS/xy_goal_tolerance" value="1.0"/>

</node>
```

**Αρχείο amcl.launch.xml**

```
<launch>
  <arg name="use_map_topic"  default="false"/>
  <arg name="scan_topic"     default="scan"/>
  <arg name="initial_pose_x" default="0.0"/>
  <arg name="initial_pose_y" default="0.0"/>
 <arg name="initial_pose_a" default="0.0"/>

  <node pkg="amcl" type="amcl" name="amcl">
    <param name="use_map_topic"            value="$(arg use_map_topic)"/>

    <!-- Publish scans from best pose at a max of 10 Hz -->

    <param name="odom_model_type"          value="diff"/>
    <param name="odom_alpha5"              value="0.1"/>
    <param name="gui_publish_rate"         value="10.0"/>
    <param name="laser_max_beams"            value="60"/>
    <param name="laser_max_range"          value="12.0"/>
    <param name="min_particles"            value="500"/>
    <param name="max_particles"            value="2000"/>
    <param name="kld_err"                  value="0.05"/>
    <param name="kld_z"                    value="0.99"/>
    <param name="odom_alpha1"              value="0.2"/>
    <param name="odom_alpha2"              value="0.2"/>
```

```
    <!-- translation std dev, m -->

    <param name="odom_alpha3"              value="0.2"/>
    <param name="odom_alpha4"              value="0.2"/>
    <param name="laser_z_hit"             value="0.5"/>
    <param name="laser_z_short"            value="0.05"/>
    <param name="laser_z_max"              value="0.05"/>
    <param name="laser_z_rand"             value="0.5"/>
    <param name="laser_sigma_hit"          value="0.2"/>
    <param name="laser_lambda_short"       value="0.1"/>
    <param name="laser_model_type"         value="likelihood_field"/>

    <!-- <param name="laser_model_type" value="beam"/> -->

    <param name="laser_likelihood_max_dist" value="2.0"/>
    <param name="update_min_d"             value="0.5"/>
    <param name="update_min_a"             value="1"/>
    <param name="odom_frame_id"            value="odom"/>
    <param name="base_frame_id"            value="base_link"/>
    <param name="resample_interval"        value="1"/>

    <!-- Increase tolerance because the computer can get quite busy -->

    <param name="transform_tolerance"      value="1.0"/>
    <param name="recovery_alpha_slow"      value="0.0"/>
    <param name="recovery_alpha_fast"      value="0.0"/>
    <param name="initial_pose_x"           value="$(arg initial_pose_x)"/>
    <param name="initial_pose_y"           value="$(arg initial_pose_y)"/>
    <param name="initial_pose_a"           value="$(arg initial_pose_a)"/>
    <remap from="scan"                  to="$(arg scan_topic)"/>
  </node>
</launch>
```

**Αρχείο mvrobot_rviz_amcl.launch**

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

        <launch>
        <arg name="model" />

        <!-- Parsing xacro and setting mastering_ros_robot_description_pkg parameter -->

        <param name="mastering_ros_robot_description_pkg" command="$(find
        xacro)/xacro.py $(find

mvrobot_description)/urdf/mvrobot.xacro" />

        <!-- Setting gui parameter to true for display joint slider -->

        <param name="use_gui" value="true"/>

        <!-- Starting Joint state publisher node which will publish the joint values -->

        <node name="joint_state_publisher" pkg="joint_state_publisher"
        type="joint_state_publisher" />

        <!-- Starting robot state publish which will publish tf -->
```

```
    <node name="robot_state_publisher" pkg="robot_state_publisher"
    type="state_publisher" />

    <!-- Launch visualization in rviz -->

    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
    mvrobot_navigation/gmapping.rviz" required="true" />

</launch>
```

## Παράρτημα Γ

## Μοντέλο αποθήκης του Gazebo

### Αρχείο free_nav_drive.cpp

```
/*
 * free_nav_drive.cpp
 *
 *  Created on: Dec 3, 2016
 *      Author: Moisiadis Vasileios
 */

#include <sstream>
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Twist.h"
#include "sensor_msgs/LaserScan.h"
#include "sensor_msgs/Range.h"
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cmath>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <time.h>
#include "image.h"

using namespace std;



//declare publisher
ros::Publisher cmd_vel_pub;

//declare subscriber
ros::Subscriber laser_scan_sub;

sensor_msgs::LaserScan laser_scan_msgs;

//function for free navigation drive
void free_nav_drive_function(void);

void free_nav_drive_callback(const sensor_msgs::LaserScan& laser_scan_msgs);


//this timer counts the time that the robot moves
void movement_timercallback(const ros::TimerEvent&);

//this timer counts the time that the program saves and process the pgm imaga
void map_timercallback(const ros::TimerEvent&);

//the functions that reads the image and the header of the image
int read_image_header(char[], int&, int&, int&, bool&);
int read_image(char[], Image&);

bool stop_flag=false;
```

```
//at the beginning of the program timercallback function is called two times with no reason ,so i   use
a flag to prevent this
bool error_flag =false;

//this flag is used once when the time runs out and the program stops
bool one_time_flag=true;

double white_rate[22]; //stores the % of the white color for each store

int rows, columns,gray_scale; // rows, cols, grayscale

            int val;

            bool type;


char file_name[100]="/home/gra/ros/mybot_ws/src/mvrobot_navigation/maps/test_map.pgm";


int main(int argc ,char **argv)
{
   //initialize the node
  ros::init(argc,argv,"free_nav_drive_node");
  ros::NodeHandle n;

    ROS_INFO("The robot has started");

   //initialize the general_time object for 60*30 sec~= 20 minutes
     ros::Timer movement_timer=n.createTimer(ros::Duration(120*60),movement_timercallback);
     ros::Timer map_timer=n.createTimer(ros::Duration(60),map_timercallback);

          //let the topic know that i want to subscribe /publish :
          //subscribe
          //call the callback function;
          laser_scan_sub=n.subscribe("/scan",10,free_nav_drive_callback);

          //publish
          cmd_vel_pub=n.advertise<geometry_msgs::Twist>("/cmd_vel",1000);

          ros::spin();

          return 0;

}



void free_nav_drive_callback(const sensor_msgs::LaserScan& laser_scan_msgs)
{

    geometry_msgs::Twist cmd_msg_for_publish;

          float linear=0,angular=0;

          for(unsigned int i =0 ;i<laser_scan_msgs.ranges.size(); i++)
          {
              float real_distance=laser_scan_msgs.ranges[i];

          linear -=cos(laser_scan_msgs.angle_min + i*laser_scan_msgs.angle_increment)/
          (1.0+real_distance*real_distance);
```

```
            angular -= sin(laser_scan_msgs.angle_min+ i*laser_scan_msgs.angle_increment)/
            (1.0+real_distance*real_distance);
            }



            linear /= laser_scan_msgs.ranges.size();

            angular /= laser_scan_msgs.ranges.size();

            if(linear>0.3) //maybe 0.5
            {
              linear=0.3;
            }
            else if(linear<-0.3)
            {
              linear=-0.3;
            }


 if(stop_flag)
 {
            cmd_msg_for_publish.linear.x=0;
            cmd_msg_for_publish.angular.z=0;
            cmd_vel_pub.publish(cmd_msg_for_publish);  // publish x=0 z=0 and sleep for 0.5 sec
            then end the application

     ros::Duration(0, 500000000).sleep();

     std::terminate();

      }
      else
      {
      cmd_msg_for_publish.linear.x=0.3+linear;
      cmd_msg_for_publish.angular.z=angular;
      cmd_vel_pub.publish(cmd_msg_for_publish);

      error_flag=true;
       }


}


void movement_timercallback(const ros::TimerEvent&)
{
            if(error_flag && one_time_flag)
            {
            stop_flag=true;

            ROS_INFO("Robot has stopped");

            ROS_INFO("The time is up");

            one_time_flag=false;
            }
}
```

```
void map_timercallback(const ros::TimerEvent&)
{
          if(error_flag)
          {
             static int map_count=0; //counts the times that map changes

          //save the current map
            std::system("rosrun map_server map_saver -f

~/ros/mybot_ws/src/mvrobot_navigation/maps/test_map");

          map_count++;

    //open the current map
    // read image header-confirm that the image is pgm p5
         read_image_header(file_name, rows, columns, gray_scale, type);

       // allocate memory for the image array
          Image image(rows, columns, gray_scale);
        // read image
          read_image(file_name, image);

          // take the white rate and save it in a matrix  "white_rate"

          white_rate[map_count]=image.white_pixels();
           cout<<"The white rate is   "<<white_rate[map_count]<<"%"<<"  the map counter is
          "<<map_count<<endl;

           //if we have 5 maps check for white % ,if it is the same ,stop the programm
            if(map_count>=5)
          {
          if((white_rate[map_count]-white_rate[map_count-1])<0.05 && (white_rate[map_count]-
white_rate[map_count-2])<=0.05 && (white_rate[map_count]-white_rate[map_count-3])<=0.05)

           {
             stop_flag=true;
            ROS_INFO("The map is ready ");
           }

          }

         }
}

int read_image(char fname[], Image& image)
{
   int i, j;
   int rows, columns, gray_scale;
   unsigned char *charImage;
   char header [100], *ptr;
   ifstream ifp;

   ifp.open(fname,  ios::binary);

   if (!ifp)
   {
      cout << "Can't read image: " << fname << endl;
      exit(1);
   }
```

```
 // read header

    ifp.getline(header,100,'\n');
    if ( (header[0]!=80) || (header[1]!=53) )
    {
        cout << "Image " << fname << " is not PGM" << endl;
        exit(1);
    }

    ifp.getline(header,100,'\n');
    while(header[0]=='#')
        ifp.getline(header,100,'\n');


    columns=strtol(header,&ptr,0);
     rows=atoi(ptr);

    ifp.getline(header,100,'\n');

    gray_scale=strtol(header,&ptr,0);

    charImage = (unsigned char *) new unsigned char [columns*rows];

    ifp.read( reinterpret_cast<char *>(charImage), (columns*rows)*sizeof(unsigned char));

    if (ifp.fail())
    {
        cout << "Image " << fname << " has wrong size" << endl;
        exit(1);
    }

    ifp.close();
     ROS_INFO("The new  image is loaded");


 //
 // Convert the unsigned characters to integers
 //

    int val;

    for(i=0; i<rows; i++)
        for(j=0; j<columns; j++)
        {
            val = (int)charImage[i*columns+j];

            image.set_pixel_value(i,j,val);

        }

    delete [] charImage;


    return (1);

}

int read_image_header(char fname[], int& rows, int& columns, int& gray_scale, bool& type)
{
    int i, j;
```

```
    unsigned char *charImage;
    char header [100], *ptr;
    ifstream ifp;

    ifp.open(fname, ios::in | ios::binary);

    if (!ifp)
    {
        cout << "Can't read image: " << fname << endl;
        exit(1);
    }

  // read header

    type = false; // PGM

    ifp.getline(header,100,'\n');
    if ( (header[0] == 80) && (header[1]== 53) )
    {
      type = false;
    }
    else if ( (header[0] == 80) && (header[1] == 54) )
    {
      type = true;
    }
    else
    {
        cout << "Image " << fname << " is not PGM or PPM" << endl;
        exit(1);
    }

    ifp.getline(header,100,'\n');
    while(header[0]=='#')
        ifp.getline(header,100,'\n');


    columns=strtol(header,&ptr,0);
   rows=atoi(ptr);

    ifp.getline(header,100,'\n');

    gray_scale=strtol(header,&ptr,0);

    ifp.close();

    return(1);
}
```

**Αρχείο image.h**

```
/*
 * image.h
 *
 *  Created on: Jan 29, 2017
 *      Author: gra
 */



#ifndef IMAGE_H
```

```
#include <iostream>
using namespace std;

#define IMAGE_H

class Image
{
     public:
          Image()
             /* Creates an Image 0x0 */
           {
              N = 0;
              M = 0;
              Q = 0;

             pixelVal = NULL;
           }
       Image(int numRows, int numCols, int grayLevels)
       /* Creates an Image of numRows x numCols and creates the arrays for it*/
       {

          N = numRows;
          M = numCols;
          Q = grayLevels;

          pixelVal = new int *[N];
          for(int i = 0; i < N; i++)
          {
              pixelVal[i] = new int [M];
              for(int j = 0; j < M; j++)
                  pixelVal[i][j] = 0;
          }
       }
       ~Image()
       /*destroy image*/
       {
           N = 0;
           M = 0;
           Q = 0;

          for(int i = 0; i < N; i++)
              delete pixelVal [N];

          delete pixelVal;
       }

       Image(const Image& oldImage)
       /*copies oldImage into new Image object*/
       {
           N = oldImage.N;
           M = oldImage.M;
           Q = oldImage.Q;

          pixelVal = new int* [N];
          for(int i = 0; i < N; i++)
          {
              pixelVal[i] = new int [M];
              for(int j = 0; j < M; j++)
              pixelVal[i][j] = oldImage.pixelVal[i][j];
          }
```

```
        }


        void set_image_info(int numRows, int numCols, int maxVal)
        /*sets the number of rows, columns and graylevels*/
        {
            N = numRows;
            M = numCols;
            Q = maxVal;
        }
        void get_image_info(int &numRows, int &numCols, int &maxVal)

        /*returns the number of rows, columns and gray levels*/
        {
           numRows = N;
           numCols = M;
           maxVal = Q;
        }

        int get_pixel_value(int row, int col)
        /*returns the gray value of a specific pixel*/
        {
           return pixelVal[row][col];
        }

        void set_pixel_value(int row, int col, int value)
        /*sets the gray value of a specific pixel*/
        {
           pixelVal[row][col] = value;
        }

        double white_pixels()
        {
            int white_pixel=0;
            for(int i=0;i<N;i++)
            {
            for(int j=0;j<M;j++)
            {
              if(pixelVal[i][j]>240)
               white_pixel++;



           }

           }
                double cells=N*M;
                return ((white_pixel/cells)*100);

           }


     private:
            int N; // number of rows
            int M; // number of columns
            int Q; // number of gray levels
            int **pixelVal;
};

#endif
```

**Αρχείο warehouse.world**

```
 <?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
    <!-- A global light source -->
    <include>
      <uri>model://sun</uri>
    </include>
    <!-- A ground plane -->
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://asphalt_plane</uri>
    </include>
    <physics name='default_physics' default='0' type='ode'>
      <gravity>0 0 -9.8066</gravity>
      <ode>
        <solver>
          <type>quick</type>
          <iters>10</iters>
          <sor>1.3</sor>
          <use_dynamic_moi_rescaling>0</use_dynamic_moi_rescaling>
        </solver>
        <constraints>
          <cfm>0</cfm>
          <erp>0.2</erp>
          <contact_max_correcting_vel>100</contact_max_correcting_vel>
          <contact_surface_layer>0.001</contact_surface_layer>
        </constraints>
      </ode>
      <max_step_size>0.002</max_step_size>
      <real_time_factor>1</real_time_factor>
      <real_time_update_rate>500</real_time_update_rate>
      <magnetic_field>6e-06 2.3e-05 -4.2e-05</magnetic_field>
    </physics>

    <!--Wall-->

    <include>
      <name>wall</name>
      <uri>model://wall1</uri>
      <pose> 0 0 0   0 0 0</pose>
    </include>

  <!--Door-->

    <include>
      <name>door</name>
      <uri>model://door2</uri>
      <pose> 9.699201 0 0.741822  0 0 0</pose>
    </include>

    <!-- SHELVES  1-->
    <include>
      <name>shelves 1</name>
      <uri>model://shelves_high2</uri>
      <pose>1 3.4 0   0 0 0</pose>
    </include>
```

```
<include>
  <name>shelves 2</name>
  <uri>model://shelves_high2</uri>
  <pose>-4.13394 3.4 0 0 -0 0</pose>
</include>

<!-- SHELVES 2 -->
<include>
  <name>shelves 5</name>
  <uri>model://shelves_high2_no_collision</uri>
  <pose>1 -1.5 0   0 0 0</pose>
</include>


<!-- SHELVES 3 -->
<include>
  <name>shelves 9</name>
  <uri>model://shelves_high2_no_collision</uri>
  <pose>1 -6.4 0   0 0 0</pose>
</include>

<!--Ground level-->

<!-- PALLET 1 -->
<include>
  <name>pallet 1 box</name>
  <uri>model://big_box3</uri>
  <pose>1.48626 3.390705 0.760253 0 0.000143 -1.57317</pose>
</include>
<include>
  <name>pallet 1 support</name>
  <uri>model://europallet</uri>
  <pose>1.48088 3.400944 0.0635 0 0 -1.56736</pose>
</include>

<!-- PALLET 4 -->
<include>
  <name>pallet 4 box</name>
  <uri>model://big_box3</uri>
  <pose>-2.37439 3.440964 0.776474 0 0 -1.59804</pose>
</include>
<include>
  <name>pallet 4 support</name>
  <uri>model://europallet</uri>
  <pose>-2.40127 3.415746 0.0635 0 0 -1.5558</pose>
</include>

<!-- PALLET 7 -->
<include>
  <name>pallet 7 box</name>
  <uri>model://big_box4</uri>
  <pose>2.61297 3.464591 0.774725 0 0 -1.59992</pose>
</include>
<include>
  <name>pallet 7 support</name>
  <uri>model://europallet</uri>
  <pose>2.59788 3.454115 0.0635 0 0 -1.56921</pose>
</include>

<!-- PALLET 10 -->
```

```
<include>
  <name>pallet 10 box</name>
  <uri>model://big_box3</uri>
  <pose>-3.45548 3.349857 0.774725 0 0 -1.56894</pose>
</include>
<include>
  <name>pallet 10 support</name>
  <uri>model://europallet</uri>
  <pose>-3.41303 3.343739 0.0635 0 -0 1.56536</pose>
</include>

<!-- PALLET 11 -->
<include>
  <name>pallet 11 box</name>
  <uri>model://big_box4</uri>
  <pose>-4.67439 3.440964 0.776474 0 0 -1.59804</pose>
</include>
<include>
  <name>pallet 11 support</name>
  <uri>model://europallet</uri>
  <pose>-4.70127 3.415746 0.0635 0 0 -1.5558</pose>
</include>

<!-- PALLET 12 -->
<include>
  <name>pallet 12 box</name>
  <uri>model://big_box3</uri>
  <pose>-5.95548 3.349857 0.774725 0 0 -1.56894</pose>
</include>
<include>
  <name>pallet 12 support</name>
  <uri>model://europallet</uri>
  <pose>-5.91303 3.343739 0.0635 0 -0 1.56536</pose>
</include>



<!--1st FLOOR-->

<!-- PALLET 2 -->
<include>
  <name>pallet 2 box</name>
  <uri>model://big_box3</uri>
  <pose>-3.25548 3.349857 2.3397 0 0 -1.56894</pose>
</include>
<include>
  <name>pallet 2 support</name>
  <uri>model://europallet</uri>
  <pose>-3.21303 3.343739 1.6373 0 -0 1.56536</pose>
</include>


<!-- PALLET 5 - SMALL BOXES -->
<include>
  <name>pallet 5 multi boxes</name>
  <uri>model://pallet_full</uri>
  <pose>0.334986 3.337637 1.63184 0 0 -1.57575</pose>
  <static>true</static>
</include>
```

```
<!-- PALLET 17 - SMALL BOXES -->
<include>
  <name>pallet 17 multi boxes</name>
  <uri>model://pallet_full</uri>
  <pose>0.334986 3.337637 2.31 0 0 -1.57575</pose>
  <static>true</static>
</include>

<!-- PALLET 8 -->
<include>
  <name>pallet 8 box</name>
  <uri>model://big_box4</uri>
  <pose>2.56918 3.321158 2.35696 0 0 -1.56652</pose>
</include>
<include>
  <name>pallet 8 support</name>
  <uri>model://europallet</uri>
  <pose>2.64128 3.308661 1.64676 0 0 -1.55378</pose>
</include>

<!-- PALLET 9 -->
<include>
  <name>pallet 9 box</name>
  <uri>model://big_box4</uri>
  <pose>-2.30257 2.987644 2.35696 0 0 -1.57937</pose>
</include>
<include>
  <name>pallet 9 support</name>
  <uri>model://europallet</uri>
  <pose>-2.28744 2.915349 1.64676 0 -0 1.5541</pose>
</include>

<!-- PALLET 13 -->
<include>
  <name>pallet 13 box</name>
  <uri>model://big_box3</uri>
  <pose>-4.67439 3.440964 2.35696 0 0 -1.59804</pose>
</include>
<include>
  <name>pallet 13 support</name>
  <uri>model://europallet</uri>
  <pose>-4.70127 3.415746 1.64676 0 0 -1.5558</pose>
</include>

<!-- PALLET 14 -->
<include>
  <name>pallet 14 box</name>
  <uri>model://big_box4</uri>
  <pose>-5.95548 3.349857 2.35696 0 0 -1.56894</pose>
</include>
<include>
  <name>pallet 14 support</name>
  <uri>model://europallet</uri>
  <pose>-5.91303 3.343739 1.64676 0 -0 1.56536</pose>
</include>

<!-- PALLET 19 -->
<include>
  <name>pallet 19 box</name>
  <uri>model://big_box3</uri>
```

```
  <pose>-0.8 3.337637 2.3397 0 0 -1.56894</pose>
</include>
<include>
  <name>pallet 19 support</name>
  <uri>model://europallet</uri>
  <pose>-0.8 3.337637 1.6373 0 -0 1.56536</pose>
</include>


<!--2nd FLOOR -->

<!-- PALLET 6 - SMALL BOXES -->
<include>
  <name>pallet 6 multi boxes</name>
  <uri>model://pallet_full</uri>
  <pose>-2.18966 3.351916 3.27046 0 0 -1.58092</pose>
  <static>true</static>
</include>

<!-- PALLET 15 - SMALL BOXES -->
<include>
  <name>pallet 15 multi boxes</name>
  <uri>model://pallet_full</uri>
  <pose>-4.68966 3.351916 3.27046 0 0 -1.58092</pose>
  <static>true</static>
</include>

<!-- PALLET 16 - SMALL BOXES -->
<include>
  <name>pallet 16 multi boxes</name>
  <uri>model://pallet_full</uri>
  <pose>-5.88966 3.351916 3.27046 0 0 -1.58092</pose>
  <static>true</static>
</include>


<!-- PALLET 20 -->
<include>
  <name>pallet 20 box</name>
  <uri>model://big_box4</uri>
  <pose>-3.25548 3.349857 3.97286 0 0 -1.56894</pose>
</include>
<include>
  <name>pallet 20 support</name>
  <uri>model://europallet</uri>
  <pose>-3.21303 3.343739 3.27046 0 -0 1.56536</pose>
</include>

<!-- PALLET 21 -->
<include>
  <name>pallet 21 box</name>
  <uri>model://big_box3</uri>
  <pose>2.56918 3.321158 3.97286 0 0 -1.56652</pose>
</include>
<include>
  <name>pallet 21 support</name>
  <uri>model://europallet</uri>
  <pose>2.64128 3.308661 3.27046 0 0 -1.55378</pose>
</include>
```

```
  <!-- PALLET 22 -->
  <include>
    <name>pallet 22 box</name>
    <uri>model://big_box4</uri>
    <pose>-0.8 3.337637 3.97286 0 0 -1.56894</pose>
  </include>
  <include>
    <name>pallet 22 support</name>
    <uri>model://europallet</uri>
    <pose>-0.8 3.337637 3.27046 0 -0 1.56536</pose>
  </include>


  <!--SHELVES - Ground level-->

  <!-- PALLET 1 -->
  <include>
    <name>pallet B-0-1 box</name>
    <uri>model://big_box3</uri>
    <pose>1.48626 -1.5 0.760253 0 0.000143 1.57317</pose>
  </include>
  <include>
    <name>pallet B-0-1 support</name>
    <uri>model://europallet</uri>
    <pose>1.48088 -1.5 0.0635 0 0 1.56736</pose>
  </include>

  <!-- PALLET 7 -->
  <include>
    <name>pallet B-0-2 box</name>
    <uri>model://big_box4</uri>
    <pose>2.61297 -1.5 0.774725 0 0 1.59992</pose>
  </include>
  <include>
    <name>pallet B-0-2 support</name>
    <uri>model://europallet</uri>
    <pose>2.61297 -1.5 0.0635 0 0 1.56921</pose>
  </include>

 <!-- PALLET 23 -->
  <include>
    <name>pallet B-2-3 box</name>
    <uri>model://big_box3</uri>
    <pose>1.48626 -6.4 0.760253 0 0.000143 1.57317</pose>
  </include>
  <include>
    <name>pallet B-2-3 support</name>
    <uri>model://europallet</uri>
    <pose>1.48088 -6.4 0.0635 0 0 1.56736</pose>
  </include>

  <!-- PALLET 24 -->
  <include>
    <name>pallet B-2-4 box</name>
    <uri>model://big_box4</uri>
    <pose>2.61297 -6.4 0.774725 0 0 1.59992</pose>
  </include>
  <include>
    <name>pallet B-2-4 support</name>
    <uri>model://europallet</uri>
```

```
      <pose>2.61297 -6.4 0.0635 0 0 1.56921</pose>
    </include>

    <!-- PALLET 25 -->
    <include>
      <name>pallet 25 box</name>
      <uri>model://big_box3</uri>
      <pose>-0.8 -1.5 0.774725 0 0 1.59992</pose>
    </include>
    <include>
      <name>pallet 25 support</name>
      <uri>model://europallet</uri>
      <pose>-0.8 -1.5 0.0635 0 0 1.56921</pose>
    </include>

  <!-- PALLET 26 -->
    <include>
      <name>pallet 26 box</name>
      <uri>model://big_box3</uri>
      <pose>-0.8 3.4 0.774725 0 0 1.59992</pose>
    </include>
    <include>
      <name>pallet 26 support</name>
      <uri>model://europallet</uri>
      <pose>-0.8 3.4 0.0635 0 0 1.56921</pose>
    </include>

    <!-- PALLET 27 -->
    <include>
      <name>pallet 27 box</name>
      <uri>model://big_box3</uri>
      <pose>-0.8 -6.4 0.774725 0 0 1.59992</pose>
    </include>
    <include>
      <name>pallet 27 support</name>
      <uri>model://europallet</uri>
      <pose>-0.8 -6.4 0.0635 0 0 1.56921</pose>
    </include>

   <!-- PALLET 28 -->
    <include>
      <name>pallet 28 box</name>
      <uri>model://big_box3</uri>
      <pose>0.3 -6.4 0.774725 0 0 1.59992</pose>
    </include>
    <include>
      <name>pallet 28 support</name>
      <uri>model://europallet</uri>
      <pose>0.3 -6.4 0.0635 0 0 1.56921</pose>
    </include>

  <!-- PALLET 29 -->
    <include>
      <name>pallet 29 box</name>
      <uri>model://big_box3</uri>
      <pose>0.3 -1.5 0.774725 0 0 1.59992</pose>
    </include>
    <include>
      <name>pallet 29 support</name>
      <uri>model://europallet</uri>
```

```
      <pose>0.3 -1.5 0.0635 0 0 1.56921</pose>
    </include>

  <!-- PALLET 30 -->
    <include>
      <name>pallet 30 box</name>
      <uri>model://big_box3</uri>
      <pose>0.3 3.4 0.774725 0 0 1.59992</pose>
    </include>
    <include>
      <name>pallet 30 support</name>
      <uri>model://europallet</uri>
      <pose>0.3 3.4 0.0635 0 0 1.56921</pose>
    </include>

   <!-- PALLET 31 -->
    <include>
      <name>pallet 31 box</name>
      <uri>model://big_box3</uri>
      <pose>-5.95548 -1.5 0.774725 0 0 -1.56894</pose>
    </include>
    <include>
      <name>pallet 31 support</name>
      <uri>model://europallet</uri>
      <pose>-5.91303 -1.5 0.0635 0 -0 1.56536</pose>
    </include>

   <!-- PALLET 32 -->
    <include>
      <name>pallet 32 box</name>
      <uri>model://big_box3</uri>
      <pose>-5.95548 -6.4 0.774725 0 0 -1.56894</pose>
    </include>
    <include>
      <name>pallet 32 support</name>
      <uri>model://europallet</uri>
      <pose>-5.91303 -6.4 0.0635 0 -0 1.56536</pose>
    </include>

  <!-- SHELVES B - 1st level-->

    <!-- PALLET 1 -->
    <include>
      <name>pallet B-1-1 box</name>
      <uri>model://big_box3</uri>
      <pose>1.6 -1.5 2.30 0 0.000143 1.57317</pose>
      <static>true</static>
    </include>
    <include>
      <name>pallet B-1-1-1 support</name>
      <uri>model://europallet</uri>
      <pose>1.6 -1.5 1.60 0 0 1.56736</pose>
      <static>true</static>
    </include>

    <!-- PALLET 7 -->
    <include>
      <name>pallet B-1-2 box</name>
      <uri>model://big_box3</uri>
      <pose>2.9 -1.5 2.30 0 0 1.59992</pose>
```

```
      <static>true</static>
    </include>
    <include>
      <name>pallet B-1-2 support</name>
      <uri>model://europallet</uri>
      <pose>2.9 -1.5 1.60 0 0 1.56921</pose>
      <static>true</static>
    </include>

    <!--SHELVES B - 2st level-->

    <!-- PALLET 1 -->
    <include>
      <name>pallet B-2-1 box</name>
      <uri>model://big_box4</uri>
      <pose>1.6 -1.5 3.97 0 0.000143 1.57317</pose>
      <static>true</static>
    </include>
    <include>
      <name>pallet B-1-1-1 support</name>
      <uri>model://europallet</uri>
      <pose>1.6 -1.5 3.27 0 0 1.56736</pose>
      <static>true</static>
    </include>

    <!-- PALLET 7 -->
    <include>
      <name>pallet B-2-2 box</name>
      <uri>model://big_box3</uri>
      <pose>2.9 -1.5 3.97 0 0 1.59992</pose>
      <static>true</static>
    </include>
    <include>
      <name>pallet B-1-2 support</name>
      <uri>model://europallet</uri>
      <pose>2.9 -1.5 3.27 0 0 1.56921</pose>
      <static>true</static>
    </include>
    </world>
  </sdf>
```

**Αρχείο 2d_nav_goals_2d_pose_wh.cpp**

```
/*
 * 2d_nav_goals_2d_pose_wh.cpp
 *
 *  Created on: Feb 4, 2017
 *      Author: Moisiadis Vasileios
 */

#include <ros/ros.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <geometry_msgs/PoseWithCovarianceStamped.h>
#include "std_msgs/String.h"
#include <actionlib/client/simple_action_client.h>
#include <tf/transform_broadcaster.h>
#include <sstream>
#include <iostream>
```

```
#include <time.h>

//declare subscriber
ros::Subscriber pose_sub;

geometry_msgs::PoseWithCovarianceStamped pose_goal;



void pose_callback(const geometry_msgs::PoseWithCovarianceStamped& pose_goal);



//Declaring move base goals
//define_move_base_clients();
move_base_msgs::MoveBaseGoal goal; //the input goal

     move_base_msgs::MoveBaseGoal goalD; //parking spot
     move_base_msgs::MoveBaseGoal goalE; // exit spot


//Declaring a new SimpleActionClient with action of move_base_msgs::MoveBaseAction;
typedef
          actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction>
          MoveBaseClient;
      /*
       define the data of the D,E points

          E=The exit of the warehouse
          D=parking spot

          */

int main(int argc, char** argv)

          {

            ros::init(argc, argv, "navigation_goals");

               ros::NodeHandle n;

               ROS_INFO("Wait for the goal to be published");

               //subscribes to the /initialpose to get the goal pose for the robot
                   pose_sub=n.subscribe("/initialpose",100,pose_callback);
                    ros::spin();

            return 0;
          }

          void pose_callback(const geometry_msgs::PoseWithCovarianceStamped& pose_goal)
          {

            try
          {
            goal.target_pose.pose.position.x=pose_goal.pose.pose.position.x;
            goal.target_pose.pose.position.y=pose_goal.pose.pose.position.y;
            goal.target_pose.pose.position.z=pose_goal.pose.pose.position.z;
```

```
            goal.target_pose.pose.orientation.w=pose_goal.pose.pose.orientation.w;
            goal.target_pose.pose.orientation.z=pose_goal.pose.pose.orientation.z;


          goalD.target_pose.pose.position.x = 0.0154;

          goalD.target_pose.pose.position.y = 7.9827;

          goalD.target_pose.pose.position.z=0;


goalD.target_pose.pose.orientation.x=0;

goalD.target_pose.pose.orientation.y=0;

goalD.target_pose.pose.orientation.z=-0.7143;

goalD.target_pose.pose.orientation.w =0.69977;

           goalE.target_pose.pose.position.x =8;
           goalE.target_pose.pose.position.y = 0.009;
          goalE.target_pose.pose.position.z=0;

          goalE.target_pose.pose.orientation.x=0;

goalE.target_pose.pose.orientation.y=0;

goalE.target_pose.pose.orientation.z=-0.99

goalE.target_pose.pose.orientation.w =0.1;

          }
          catch (int e)
          {

          goal.target_pose.pose.position.x = 1.0;

          goal.target_pose.pose.position.y = 1.0;


goal.target_pose.pose.orientation.w =1.0;

          ROS_INFO("Error while trying to send the goal");
          }


std::cout<<goal.target_pose.pose.position.x<<" "<<goal.target_pose.pose.position.y<<std::endl;

std::cout<<goal.target_pose.pose.position.x<<" "<<goal.target_pose.pose.position.y<<std::endl;

//Initiating move_base client
MoveBaseClient ac("move_base", true);

         //Waiting for server to start
          while(!ac.waitForServer(ros::Duration(5.0)))
           {

             ROS_INFO("Waiting for the move_base action server");

          }
```

```
//Setting target frame id and time in the goal action
 goal.target_pose.header.frame_id = "map";
          goal.target_pose.header.stamp = ros::Time::now();
 ROS_INFO("Sending move base goal");

//Sending goal
ac.sendGoal(goal);
 ac.waitForResult();

if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
   ROS_INFO("the robot has arrived to the goal position");
else
  {
    ROS_INFO("The base failed for some reason");

     exit(1);
  }

 //before going to the exit wait for five seconds
ROS_INFO("wait five seconds before going at the exit");

            ros::Duration(5.0).sleep();
ROS_INFO("Going to the EXIT");

  goalE.target_pose.header.frame_id = "map";
 goalE.target_pose.header.stamp = ros::Time::now();

 //sending the robot at the exit of the warehouse
  ROS_INFO("Sending move base goal");
 ac.sendGoal(goalE);
 ac.waitForResult();

  if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
          ROS_INFO("The robot has arrived to the EXIT");
  else
            ROS_INFO("The base failed for some reason");


 //before g0oing to the parking spot wait for five seconds

ROS_INFO("wait five seconds before going at the parking spot");
ros::Duration(5.0).sleep();


ROS_INFO("Going to the parking spot");
goalD.target_pose.header.frame_id = "map";
          goalD.target_pose.header.stamp = ros::Time::now();

 //sending the robot at the parking spot of the warehouse
  ROS_INFO("Sending move base goal");
             ac.sendGoal(goalD);
              ac.waitForResult();

if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
            ROS_INFO("The robot has arrived to the parking spot");
  else
          ROS_INFO("The base failed for some reason");
          ROS_INFO("Wait for the goal to be published");
}
```

## Βιβλιογραφία

### Συγγράμματα και Εργασίας


(1) Günter Ullrich - Automated Guided Vehicle Systems_ A Primer with Practical Applications Voerde, September 2014

(2) Aaron Martinez, Enrique Fernández, Learning ROS for Robotics Programming, September 2013 Copyright © 2013 Packt Publishing

(3) Lentin Joseph, Mastering ROS for Robotics Programming, December 2015 Copyright © 2015 Packt Publishing

(4) Morgan Quigley, Brian Gerkey, and William D. Smart, Programming Robots With ROS, January 2010 Copyright © 2010

(5) H. Martínez-Barberá * , D. Herrero-Pérez Department of Information and Communications Engineering, University of Murcia, Autonomous navigation of an automated guided vehicle industrial environments, 30100 Murcia, Spain, Received 5 November 2007

(6) Stefan Kohlbrecher and Oskar von Stryk, Johannes Meyer and Uwe Klingauf, A Flexible and Scalable SLAM System with Full 3D Motion Estimation, Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium Nov. 2011

(7) Tully Foote, tf: The Transform, LibraryOpen Source Robotics Foundation Mountain View, CA 94043

(8) Giorgio Grisetti, Cyrill Stachniss, Wolfram Burgard, Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters, IEEE Transactions on Robotics ( Volume: 23, Issue: 1, Feb. 2007 )

(9) Sebastian Thrun, Wolfram Bugrard, Dieter Fox, Probabilistic Robotics Published by MIT Press, 2005 ISBN 10: 0262201623

(10) João Pedro Machado dos Santos, SmokeNav – Simultaneous Localization and Mapping in Reduced Visibility Scenarios September 2013 University of Coimbra

(11) João Machado Santos, David Portugal and Rui P. Rocha,An Evaluation of 2D SLAM Techniques Available in Robot Operating System, Safety, Security, and Rescue Robotics (SSRR), 2013 IEEE International Symposium Oct. 2013

(12) Dieter Fox, KLD-Sampling: Adaptive Particle Filters, Department of Computer Science & Engineering University of Washington 2001

(13) Frank Dellaert, Dieter Fox, Wolfram Burgard, Sebastian Thrun, Monte Carlo Localization for Mobile Robots, Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference May 1999

(14) Dieter Fox, Wolfram Burgard,Frank Dellaert, Sebastian Thrun, Monte Carlo Localization: Efficient Position Estimation for Mobile Robots, January 1999, Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, 1999, Orlando, Florida, USA

(15) Sebastian Thrun, Particle Filters in Robotics, August 2002, Proceeding UAI 02 Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence

**Ιστότοποι**


(14) http://wiki.ros.org/kinetic/Installation/Ubuntu

(15) http://wiki.ros.org/ROS/Tutorials

(16) http://wiki.ros.org/tf

(17) http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF

(18) http://wiki.ros.org/geometry/CoordinateFrameConventions

(19) http://gazebosim.org/

(20) http://wiki.ros.org/navigation

(21) https://www.openslam.org/gmapping.html

(22) https://www.openslam.org/linearslam.html

(23) http://wiki.ros.org/amcl

(24) https://openslam.org/kld-sampling.html

(25) http://wiki.ros.org/stdr_simulator

(26) http://wiki.ros.org/actionlib

(27) http://wiki.ros.org/actionlib/DetailedDescription