ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ, ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ ΣΤΗ ΡΟΜΠΟΤΙΚΗ

# ΔΗΜΙΟΥΡΓΙΑ ΕΚΠΑΙΔΕΥΤΙΚΩΝ ΠΡΟΣΟΜΟΙΩΣΕΩΝ ΚΑΙ ΛΟΓΙΣΜΙΚΩΝ ΜΕ ΧΡΗΣΗ ΤΟΥ ROS 2

ΦΟΙΤΗΤΗΣ: ΔΗΜΗΤΡΗΣ ΚΑΤΟΣ
ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Δρ ΒΟΛΟΓΙΑΝΝΙΔΗΣ ΣΤΑΥΡΟΣ


ΣΕΡΡΕΣ,
ΜΑΡΤΙΟΣ, 2024

**Υπεύθυνη δήλωση**

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπροσθέτως, βεβαιώνω ότι αυτή η διπλωματική εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Μεταπτυχιακού με τίτλο «Ρομποτική» (MSc in Robotics) του τμήματος Μηχανικών Πληροφορική, Υπολογιστών και Τηλεπικοινωνιών της Σχολής Μηχανικών του Διεθνές Πανεπιστημίου Ελλάδος, κόμβος Σερρών.

Ο Δηλών

Κάτος Δημήτριος

## Ευχαριστίες

Η εκπόνηση της εργασίας αποτελεί το τελικό στάδιο για την απόκτηση του μεταπτυχιακού διπλώματος στη Ρομποτική. Η δουλειά και ο κόπος που καταβλήθηκε για την περάτωση της εργασίας έγινε με μεγάλη αγάπη και αφοσίωση. Η εργασία αποσκοπεί στο να βοηθήσει τους μελλοντικούς φοιτητές του Μεταπτυχιακού τμήματος στη Ρομποτική. Θα ήθελα να ευχαριστήσω όλους τους καθηγητές του τμήματος για την βοήθεια τους κατά τη διάρκεια των σπουδών, οι οποίοι με ενέπνευσαν να συνεχίσω και να επιδιώκω το κάτι παραπάνω. Ξεχωριστή αναφορά στον κ. Βολογιαννίδη Σταύρο, επιβλέπων καθηγητή της πτυχιακής εργασίας, ο οποίος με εμπιστεύτηκε για την ανάθεση ενός εξαιρετικά ενδιαφέροντος project αλλά και την βοήθεια του καθόλη την επεξεργασία και βελτίωση της εργασίας.

Τέλος, ένα μεγάλο ευχαριστώ στην οικογένεια και στους φίλους μου, οι οποίοι με ανέχτηκαν υπομονετικά και με υποστήριξαν καθόλη τη διάρκεια των σπουδών μου αλλά και στην εκπόνηση της εργασίας.

## Περίληψη

Η Διπλωματική εργασία παρουσιάζει προσομοιώσεις και λογισμικά τα οποία είναι προορισμένα για την διδασκαλία τους στην Τριτοβάθμια εκπαίδευση. Ο στόχος της εργασίας είναι να βοηθήσει φοιτητές να κατανοήσουν κάποιες από τις λειτουργίες του λογισμικού ROS 2 μέσα από την ενασχόληση τους με προσομοιώσεις ρομπότ οι οποίες μπορούν να έχουν και πραγματικές προεκτάσεις. Στο πρώτο κεφάλαιο, παρουσιάζεται το λογισμικού του ROS 2, πραγματοποιείται η εγκατάσταση του, αναλύονται εκτενώς τα χαρακτηριστικά του και αναφέρονται κάποιες διαφορές συγκριτικά με την προηγούμενη έκδοση του. Στο δεύτερο κεφάλαιο, δημιουργείται μια προσομοίωση ενός ρομπότ με δύο ρόδες και παρουσιάζονται το περιβάλλον οπτικοποίησης Rviz2 και προσομοίωσης Gazebo Classic. Στο τρίτο κεφάλαιο, δημιουργείται ένας βραχίονας δύο βαθμών ελευθερίας και επιτυγχάνεται ο έλεγχος του με χρήση του πακέτου ros2_control. Το τέταρτο κεφάλαιο, παρουσιάζει την προσθήκη αισθητήρων σε ένα ρομπότ διαφορικής κίνησης και αναπτύσσεται ένας απλός αλγόριθμος αποφυγής εμποδίων σε γλώσσα Python. Τα δύο τελευταία κεφάλαια της εργασίας έχουν σαν κύριο στόχο την παρουσίαση κάποιων απλών χαρακτηριστικών δύο πακέτων του ROS2, τα οποία είναι το Nav2 και MoveIt2. Παράλληλα σαν συμπλήρωμα της διπλωματικής εργασίας, δημιουργήθηκαν παρουσιάσεις κατάλληλες για διδασκαλία και διαμορφώθηκαν κατάλληλα αποθετήρια ώστε οι κώδικες που αναπτύχθηκαν να είναι προσβάσιμοι στο ευρύ κοινό.

## Abstract

The thesis presents simulations and software which are intended for teaching them in higher education. The aim of the thesis is to help students to understand some of the functions of ROS 2 software through dealing with robot simulations which can have real-world implications. In the first chapter, the ROS 2 software is introduced, its installation is performed, its features are extensively analyzed and some differences compared to the previous version are mentioned. In the second chapter, a simulation of a robot with two wheels is created and the visualization environment Rviz2 and simulation environment Gazebo Classic are presented. In the third chapter, a two-degree-of-freedom arm is created and its control is achieved using the ros2_control package. Chapter four, presents the addition of sensors to a differential drive robot and a simple obstacle avoidance algorithm is developed using Python. The last two chapters of the thesis have as their main goal the presentation of some simple features of two ROS2 packages, which are Nav2 and MoveIt2. As a complement to the thesis, presentations suitable for teaching were created and appropriate repositories were formed so that the codes developed are accessible to the general public.

## Πίνακας Περιεχομένων

## Ευρετήριο Εικόνων

## Ευρετήριο Πινάκων

# Κεφάλαιο 1

# Εισαγωγή στο ROS 2

## 1.1 Τι είναι το ROS 2

Το ROS 2 (Robot Operating System) είναι ένα ανοικτού κώδικα, μετά-λειτουργικό σύστημα το οποίο προσφέρει μια συλλογή από βιβλιοθήκες και εργαλεία τα οποία συντελούν στην δημιουργία ρομποτικών εφαρμογών. Καλείται λειτουργικό σύστημα διότι παρουσιάζει πολλές ομοιότητες ενός λειτουργικού συστήματος, αλλά απαιτεί ένα άλλο λειτουργικό σύστημα, όπως είναι τα Linux, για να χρησιμοποιηθεί. Ένας από τους βασικούς σκοπούς του είναι να προσφέρει εύκολη επικοινωνία μεταξύ του χρήστη, του λειτουργικού συστήματος του υπολογιστή, ενός ρομπότ και των αισθητήρων του.

Το ROS 2 βασίζεται πάνω στην επιτυχημένη έκδοση του ROS. Η πρώτη έκδοση του ROS κυκλοφόρησε το 2007 από την εταιρεία Willow Garage. Ο αρχικός στόχος της εταιρείας ήταν να προσφέρει εργαλεία λογισμικού στους χρήστες που ήθελαν να αναλάβουν καινοτόμες έρευνες και αναπτυξιακά πρότζεκτ για τον χειρισμό του ρομπότ PR2. Ωστόσο, με την πάροδο του χρόνου έγινε αντιληπτό ότι αυτά τα αναπτυξιακά λογισμικά και εργαλεία μπορούν να ικανοποιήσουν και άλλου τύπου ρομπότ.

Από την κυκλοφορία της πρώτης έκδοσης του ROS έως και σήμερα έχουν αλλάξει αρκετά πράγματα στο χώρο του προγραμματισμού και της πληροφορικής με αποτέλεσμα να θεωρείται αναγκαίο να χτιστεί η επόμενη γενιά της πλατφόρμας του ROS. Η νέα έκδοση του ROS έχει σαν στόχο να συνεχίσει να προσφέρει αναπτυξιακά εργαλεία βελτιώνοντας τα υπάρχοντα και ακολουθώντας τις σύγχρονες τεχνολογίες. Επιπροσθέτως στόχος του ROS 2 είναι η παραγωγή προϊόντων που θα είναι ROS-based, όπως εργοστασιακά, αγροτικά ρομπότ ή οι συνηθισμένες ρομποτικές σκούπες.

Συνολικά το ROS 2 είναι ένα λογισμικό το οποίο προσφέρει την ικανότητα χειρισμού ενός ρομπότ χωρίς να είναι απαραίτητο να υπάρχει η γνώση όλων των λεπτομερειών που κρύβονται πίσω από αυτό. Το ROS 2 βοηθάει τους χρήστες να δημιουργήσουν όλα τα απαραίτητα components για τη λειτουργία ενός ρομπότ και έπειτα να τα συνδέσουν για να αναπτύξουν αυτόνομες ρομποτικές εφαρμογές. Τέλος, προσφέρονται πρωτοποριακά λογισμικά που μπορούν να βοηθήσουν στην επιθυμητή λειτουργία των ρομπότ και στον έλεγχο αυτών από το χρήστη μέσω συστημάτων πραγματικού χρόνου.

## 1.2 Γιατί ROS 2

Η χρήση του ROS 2 για την ανάπτυξη μιας ρομποτικής εφαρμογής είναι ένας αρκετά γρήγορος τρόπος για τη δημιουργία μιας πραγματικής εφαρμογής ή προσομοίωσης. Το ROS 2 προσφέρει ρομποτικά εργαλεία,

βιβλιοθήκες και δυνατότητες που χρειάζονται για την ανάπτυξη ρομποτικών εφαρμογών, παρέχοντας στους χρήστες και στις εταιρείες τη δυνατότητα να εστιάσουν στα προβλήματα που θεωρούν ότι είναι σημαντικά για την έρευνα τους. Το ROS 2 είναι ανοιχτού κώδικα και προσφέρει ευελιξία ώστε να αποφασιστεί πότε και που θα χρησιμοποιηθεί και παράλληλα προσφέρεται η ελευθερία προσαρμογής στις ανάγκες της εκάστοτε έρευνας.

Το ROS έχει ευρεία εφαρμογή στην εκπαιδευτική ρομποτική στην τριτοβάθμια εκπαίδευση. Θεωρείται η βάση για πάρα πολλές ρομποτικές εφαρμογές οι οποίες αποτελούν έργο φοιτητών αλλά και από μεγάλες εταιρίες. Η αναβαθμισμένη εκδοχή του λογισμικού έχει σαν στόχο την δημιουργία ρομπότ που θα είναι διαθέσιμα στην αγορά, ωστόσο προσφέρονται επιλογές σχεδίασης με στόχο τα ρομπότ να μπορούν να χρησιμοποιηθούν ακόμη και από κυβερνήσεις ή τεράστιους οργανισμούς. Χαρακτηριστικό παράδειγμα αυτού αποτελεί η NASA η οποία έχει σαν στόχο την ανάπτυξη ενός ανθρωποειδούς ρομπότ βασισμένο στο ROS 2 για τον διαστημικό σταθμό της.

Ένα πολύ σπουδαίο χαρακτηριστικό του ROS 2 είναι στο γεγονός ότι η χρήση του υποστηρίζεται από πολλά λειτουργικά συστήματα όπως είναι τα Windows, τα Linux και τα MacOs. Συγχρόνως υποστηρίζεται και σε ενσωματωμένα συστήματα όπως μικροεπεξεργαστές (πχ. Raspberry pi) αλλά και μικροελεγκτές (μέσω του πακέτου micro-ROS).

Συνολικά προσφέρονται εργαλεία και πακέτα που είναι χρήσιμα για αυτόνομα ρομποτικά οχήματα, τα οποία βοηθούν στη χαρτογράφηση ή τον εντοπισμό τους (Nav2 package), για βραχίονες πολλών βαθμών ελευθερίας (MoveIt2) καθώς και τον έλεγχο των ρομπότ (ros2_control). Με αυτό τον τρόπο το ROS μπορεί να χρησιμοποιηθεί σε οποιοδήποτε τύπου ρομπότ και να διαχειριστεί οποιαδήποτε αποστολή του ανατεθεί. Συνεπώς, τα ρομποτικά συστήματα που βασίζονται σε αυτό το λογισμικό μπορούν να είναι εσωτερικού ή εξωτερικού χώρου και να προορίζονται για χερσαία, εναέρια ή υποθαλάσσια αποστολή.

Επίσης, προσφέρονται και εργαλεία που διαθέτουν γραφικό περιβάλλον. Τέτοια εργαλεία είναι το rqt, το Rviz, το Gazebo και το Webots. Συνοπτικά το Rviz, είναι ένα εργαλείο που βοηθάει στην οπτικοποίηση των δεδομένων, δηλαδή προσφέρει ένα φιλικό περιβάλλον για να εξεταστούν τα δεδομένα που εισέρχονται από αισθητήρες όπως οι κάμερες, τα lidar και άλλοι. Τα δύο τελευταία περιβάλλοντα είναι πολύ σημαντικά καθώς εκεί πραγματοποιούνται όλες οι προσομοιώσεις των ρομπότ και ελέγχεται η συμπεριφορά τους στον πραγματικό κόσμο.

Για όλους τους παραπάνω λόγους το ROS είναι ιδιαίτερα δημοφιλής στην ρομποτική κοινότητα και η χρήση του θα συνεχίσει να αυξάνεται μελλοντικά. Επιπροσθέτως το ROS διαθέτει μια τεράστια κοινότητα που αποτελείται από αρκετούς επιστήμονες οι οποίοι συνεργάζονται μεταξύ τους λύνοντας προβλήματα.

## 1.3 Εγκατάσταση του ROS 2

Το ROS 2 απαιτεί ένα προ-εγκατεστημένο λειτουργικό σύστημα για να εκτελεστεί, το οποίο δεν πρέπει να είναι απαραίτητα ένα λογισμικό Linux όπως γινόταν στην προηγούμενη έκδοση του. Συνολικά δίνεται η δυνατότητα εγκατάστασης στα ακόλουθα λειτουργικά συστήματα:

- Ubuntu Linux – Jammy Jellyfish (22.04)
- Windows 10
- macOS

Το ROS 2 μπορεί να εκτελεστεί και μέσω του WSL 2 (Windows Subsystem for Linux) των Windows 11. Το WSL 2 είναι ένα πανίσχυρο εργαλείο, καθώς είναι μια μηχανή μέσω της οποίας μπορεί να εκτελεστεί το λογισμικό των Linux εντός του περιβάλλοντος των Windows. Με αυτό το τρόπο μπορεί να γίνει η εκμετάλλευση των θετικών χαρακτηριστικών που διαθέτουν τα Windows 11.

Ένα πολύ μεγάλο κομμάτι του ROS 2 αποτελούν τα ROS Distributions ή αλλιώς ‘distro’. Ένα ROS Distribution είναι μια έκδοση του ROS που περιέχει διάφορα πακέτα και συχνά παρομοιάζεται με τα αντίστοιχα Linux Distributions. Ο σκοπός αυτών των ROS ‘distro’ είναι να επιτρέψει στους διαχειριστές του ROS να δουλέψουν και να βελτιώσουν μια συγκεκριμένα έκδοση του μέχρι να είναι έτοιμοι να προχωρήσουν στην περαιτέρω αναβάθμιση του.

Όλες εκδόσεις του ROS 2 έχουν ημερομηνία λήξης (end-of-life) για αυτό το λόγο επιλέγεται η εγκατάσταση της έκδοσης που θα υποστηρίζεται μακροχρόνια. Πιο συγκεκριμένα, επιλέγεται η έκδοση Humble Hawksbill και παρακάτω παρουσιάζεται η εγκατάσταση της στο περιβάλλον των Linux. Το πρώτο βήμα είναι να γίνει η βεβαίωση ότι ο προσωπικός υπολογιστής διαθέτει ένα locale το οποίο υποστηρίζει UTF-8. Σε ένα τερματικό εκτελούνται οι παρακάτω εντολές.

```
$ locale  # check for UTF-8

$ sudo apt update && sudo apt install locales

$ sudo locale-gen en_US en_US.UTF-8

$ sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8

$ export LANG=en_US.UTF-8
```

Πίνακας 1. Έλεγχος του locale.

Στη συνέχεια γίνεται η προσθήκη του ROS 2 apt repository στο σύστημα ώστε ο υπολογιστής να γνωρίζει που έχει εγκατασταθεί το ROS 2. Αρχικά πρέπει να επιβεβαιωθεί ότι το Ubuntu Universal repository είναι ενεργό. Εκτελούνται οι παρακάτω εντολές.

```
$ sudo apt install software-properties-common
$ sudo add-apt-repository universe
```

Πίνακας 2. Έλεγχος του Ubuntu Universal repository.

Στη συνέχεια γίνεται η προσθήκη ενός κλειδιού GPG, οπότε εκτελούνται σε ένα τερματικό οι παρακάτω εντολές.

```
$ sudo apt update && sudo apt install curl –y
$ sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
/usr/share/keyrings/ros-archive-keyring.gpg
```

Πίνακας 3. Εισαγωγή κλειδιού GPG.

Στη συνέχεια προστίθεται το repository στη λίστα με τα sources με χρήση της παρακάτω εντολής.

```
$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-
keyring.gpg] http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo
$UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

Πίνακας 4. Εισαγωγή του repository.

Το επόμενο βήμα είναι η εγκατάσταση των πακέτων του ROS 2. Προτού εγκατασταθούν αυτά τα πακέτα συστήνεται η ενημέρωση και η αναβάθμιση των ήδη εγκατεστημένων πακέτων των Linux. Είναι καλό να ακολουθείτε αυτή η διαδικασία πριν την εγκατάσταση νέων πακέτων. Οπότε σε ένα τερματικό εκτελούνται τα ακόλουθα:

```
$ sudo apt update
$ sudo apt upgrade
```

Πίνακας 5. Ενημέρωση του συστήματος.

Έπειτα από την εκτέλεση των παραπάνω εντολών, το επόμενο βήμα είναι η εγκατάσταση του ROS 2. Το ROS προσφέρει τρεις επιλογές εγκατάστασης οι οποίες είναι οι ακόλουθες:

1) **Η πλήρης εγκατάσταση του ROS 2**. Εμπεριέχεται το ROS, το περιβάλλον οπτικοποίησης (Rviz 2), demos και διάφορα tutorials. Η εγκατάσταση του επιτυγχάνεται με την παρακάτω εντολή.

```
$ sudo apt install ros-humble-desktop
```

Πίνακας 6. Πλήρης εγκατάσταση του ROS2.

2) **Η εγκατάσταση του ROS-Base** (Bare Bones). Η συγκεκριμένη έκδοση εμπεριέχει βιβλιοθήκες επικοινωνίας, πακέτα μηνυμάτων και εντολές που εκτελούνται στο τερματικό. Άξιο αναφοράς είναι ότι δεν εγκαθίσταται κάποιο πακέτο το οποίο να εμπεριέχει γραφικό περιβάλλον(GUI). Η συγκεκριμένη εγκατάσταση βρίσκει ευρεία εφαρμογή σε πλακέτες τύπου raspberry pi.

```
$ sudo apt install ros-humble-ros-base
```

Πίνακας 7. Εγκατάσταση του ROS-Base.

3) **Εγκατάσταση Development tools**. Εμπεριέχει μεταγλωττιστές (compilers) και διάφορα εργαλεία για την κατασκευή ROS πακέτων.

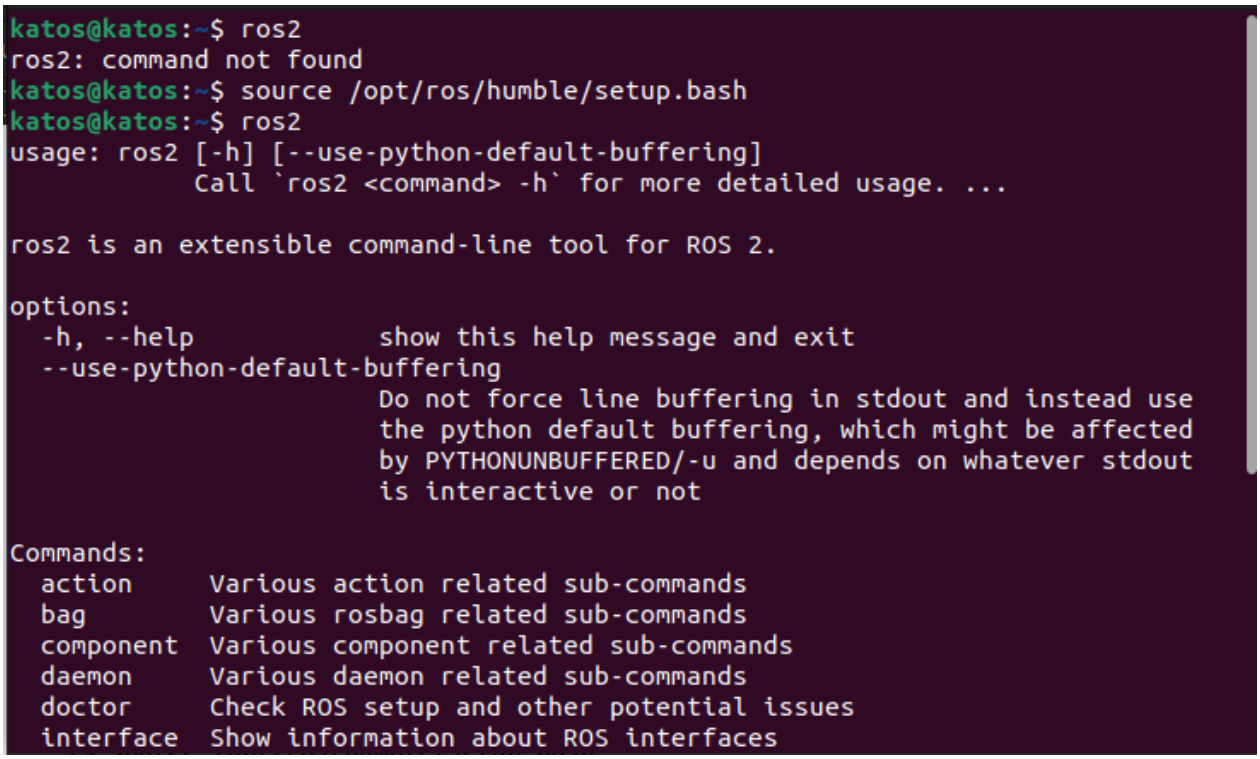
```
$ sudo apt install ros-dev-tools
```

Πίνακας 8. Εγκατάσταση των Development tools.

Για την ανάπτυξη των εφαρμογών που θα αναπτυχθούν, είναι αναγκαία η πλήρης εγκατάσταση του ROS 2. Έπειτα από την επιτυχημένη εγκατάσταση του, τότε το λογισμικό είναι έτοιμο για χρήση. Απαραίτητη προϋπόθεση είναι να γίνεται source το περιβάλλον του ROS μέσω της παρακάτω εντολής. Σε περίπτωση που δεν πραγματοποιείται το source του περιβάλλοντος, θα λαμβάνεται σφάλμα σε όλες τις διαδικασίες που αφορούν το λογισμικό.

```
$ source /opt/ros/humble/setup.bash
```

Πίνακας 9. Στήσιμο του περιβάλλοντος.




Εικόνα 1. Χρήση της εντολής ros2.

Στην παραπάνω εικόνα παρατηρείται η εκτέλεση της εντολής ros2, από την οποία προκύπτει ένα σφάλμα. Αφού γίνει το source του περιβάλλοντος του ROS, ξανά εκτελείται η εντολή και προκύπτει ένα manual του ROS. Το source του περιβάλλοντος είναι μια διαδικασία απαραίτητη για την λειτουργία του λογισμικού. Επειδή, η εκτέλεση της εντολής είναι εύκολο να ξεχαστεί, εκτελείται για μοναδική φορά το ακόλουθο.

| $ echo 'source /opt/ros/humble/setup.bash' >> ~/.bashrc |
|---|

Πίνακας 10. Τοποθέτηση εντολής στο bashrc αρχείο.

Μέσω της παραπάνω εντολής, επιτυγχάνεται η προσθήκη μιας επιπρόσθετης εντολής στο αρχείο bashrc των Linux. Με αυτό τον τρόπο σε κάθε νέο τερματικό που εκκινεί θα εκτελείται η εντολή του πίνακα 9 με αποτέλεσμα το περιβάλλον του ROS να στήνεται αυτόματα σε κάθε νέο terminal και συνεπώς μπορούν να πραγματοποιηθούν διεργασίες σε αυτό.

Στο σημείο αυτό μπορεί να γίνει η εκτέλεση κάποιων demos του ROS 2, για να εξασφαλιστεί η επιτυχημένη εγκατάσταση του. Θα γίνει η χρήση δύο τερματικών, στα οποία θα εκτελούνται δύο διαφορετικά ROS nodes. Το ένα θα δημοσιεύει κάποια δεδομένα και το άλλο θα πρέπει να λαμβάνει αυτά τα δεδομένα.

```
katos@KATOS:~$ ros2 run demo_nodes_cpp talker
[INFO] [1690287533.980611628] [talker]: Publishing: 'Hello World: 1'
[INFO] [1690287534.980487888] [talker]: Publishing: 'Hello World: 2'
[INFO] [1690287535.980628589] [talker]: Publishing: 'Hello World: 3'
[INFO] [1690287536.980592752] [talker]: Publishing: 'Hello World: 4'
[INFO] [1690287537.980483203] [talker]: Publishing: 'Hello World: 5'
[INFO] [1690287538.980543977] [talker]: Publishing: 'Hello World: 6'
[INFO] [1690287539.980576501] [talker]: Publishing: 'Hello World: 7'
[INFO] [1690287540.980703952] [talker]: Publishing: 'Hello World: 8'
[INFO] [1690287541.980503720] [talker]: Publishing: 'Hello World: 9'
[INFO] [1690287542.980582404] [talker]: Publishing: 'Hello World: 10'
[INFO] [1690287543.980622243] [talker]: Publishing: 'Hello World: 11'
[INFO] [1690287544.980508229] [talker]: Publishing: 'Hello World: 12'
[INFO] [1690287545.980576858] [talker]: Publishing: 'Hello World: 13'
[INFO] [1690287546.980352933] [talker]: Publishing: 'Hello World: 14'
[INFO] [1690287547.980438770] [talker]: Publishing: 'Hello World: 15'
[INFO] [1690287548.980415661] [talker]: Publishing: 'Hello World: 16'
[INFO] [1690287549.980432300] [talker]: Publishing: 'Hello World: 17'
[INFO] [1690287550.980379462] [talker]: Publishing: 'Hello World: 18'
[INFO] [1690287551.980370432] [talker]: Publishing: 'Hello World: 19'
[INFO] [1690287552.980347223] [talker]: Publishing: 'Hello World: 20'
```

Εικόνα 2. Talker node.

```
katos@KATOS:~$ ros2 run demo_nodes_py listener
[INFO] [1690287715.362154892] [listener]: I heard: [Hello World: 6]
[INFO] [1690287716.349892084] [listener]: I heard: [Hello World: 7]
[INFO] [1690287717.349741782] [listener]: I heard: [Hello World: 8]
[INFO] [1690287718.348696681] [listener]: I heard: [Hello World: 9]
[INFO] [1690287719.348385889] [listener]: I heard: [Hello World: 10]
[INFO] [1690287720.348513925] [listener]: I heard: [Hello World: 11]
[INFO] [1690287721.348669439] [listener]: I heard: [Hello World: 12]
[INFO] [1690287722.349237661] [listener]: I heard: [Hello World: 13]
[INFO] [1690287723.349594503] [listener]: I heard: [Hello World: 14]
[INFO] [1690287724.349860838] [listener]: I heard: [Hello World: 15]
[INFO] [1690287725.349459616] [listener]: I heard: [Hello World: 16]
[INFO] [1690287726.349565457] [listener]: I heard: [Hello World: 17]
[INFO] [1690287727.349919243] [listener]: I heard: [Hello World: 18]
[INFO] [1690287728.348126930] [listener]: I heard: [Hello World: 19]
[INFO] [1690287729.348589935] [listener]: I heard: [Hello World: 20]
[INFO] [1690287730.349167032] [listener]: I heard: [Hello World: 21]
[INFO] [1690287731.348657435] [listener]: I heard: [Hello World: 22]
[INFO] [1690287732.349611707] [listener]: I heard: [Hello World: 23]
[INFO] [1690287733.348201908] [listener]: I heard: [Hello World: 24]
[INFO] [1690287734.348400237] [listener]: I heard: [Hello World: 25]
[INFO] [1690287735.348088911] [listener]: I heard: [Hello World: 26]
[INFO] [1690287736.348477402] [listener]: I heard: [Hello World: 27]
[INFO] [1690287737.349443919] [listener]: I heard: [Hello World: 28]
[INFO] [1690287738.348344074] [listener]: I heard: [Hello World: 29]
```

Εικόνα 3. Listener Node.

## 1.4 Δημιουργία ενός ROS workspace

Η κατασκευή ενός workspace στο ROS είναι πολύ σημαντική, καθώς είναι το σύστημα μεταγλώττισης όλων των πακέτων του ROS. Στο ROS 2 χρησιμοποιείται το colcon για το χτίσιμο των πακέτων σε ένα ROS workspace. Το colcon είναι παρόμοιο με το catkin make το οποίο χρησιμοποιείται στην πρώτη έκδοση του λογισμικού. Για να γίνει η χρήση του colcon πρέπει πρώτα να εγκατασταθεί.

```
$ sudo apt install python3-colcon-common-extensions
```

Πίνακας 11. Εγκατάσταση του colcon.

Ένα ROS workspace είναι ένας κατάλογος με αρκετά περίεργη δομή. Θα δημιουργηθεί ένας υποκατάλογος ο οποίος είναι ο src. Στο συγκεκριμένο υποκατάλογο θα δημιουργούνται όλα τα νέα πακέτα του ROS 2 και πιο συγκεκριμένα ο κώδικας που θα αναπτυχθεί μέσω κάποιων πακέτων.

Πληροφοριακά η φακελοδομή του ROS workspace είναι η ακόλουθη:

- src/ Εδώ τοποθετείται ο κώδικας προς μεταγλώττιση, δηλαδή όλα τα πακέτα.
- build/ Τοποθετούνται κάποια ενδιάμεσα αρχεία που είναι χρήσιμα για τη μεταγλώττιση.
- install/ Είναι ένας κατάλογος στον οποίο εγκαθίσταται κάθε πακέτο.
- log/ Ο κατάλογος περιέχει ποικιλία από πληροφορίες που αφορούν κάθε επίκληση του colcon.

Το επόμενο βήμα είναι η δημιουργία του workspace, του υποκαταλόγου src και έπειτα θα ακολουθήσει το χτίσιμο του workspace.

```
$ mkdir -p ~/ros2_ws/src
$ cd ~/ros2_ws
```

Πίνακας 12. Δημιουργία του workspace.

Ακολουθεί η κλήση της εντολής colcon για το χτίσιμο του workspace. Έπειτα γίνεται η κλήση μιας της εντολής η οποία εκτελεί κάποια test για τα πακέτα που έχουν εγκατασταθεί.

```
$ colcon build
$ colcon test
```

Πίνακας 13. Χτίσιμο του workspace.

Αφού έχει ολοκληρωθεί η διαδικασία της δημιουργίας του ROS workspace, πρέπει να στηθεί το περιβάλλον του workspace. Η συγκεκριμένη εντολή είναι ιδιαίτερα σημαντική καθώς είναι απαραίτητο να εκτελείται σε κάθε τερματικό που θα χρησιμοποιηθεί το workspace. Η εντολή θα πρέπει να εκτελείται ακόμη και αν δεν έχει προηγηθεί το χτίσιμο του workspace. Το source του περιβάλλοντος επιτυγχάνεται με την ακόλουθη εντολή.

```
$ source install/setup.bash
```

Πίνακας 14. Setup του workspace.

Η εντολή αυτή θα πρέπει να εκτελείται κάθε φορά που είναι επιθυμητό να χρησιμοποιηθεί το workspace. Για το λόγο αυτό εκτελείται η παρακάτω εντολή, η οποία επιτυγχάνει το στήσιμο του περιβάλλοντος σε κάθε νέο τερματικό που προτίθεται να γίνει η χρήση του workspace.

```
$ echo 'source ~/ros2_ws/install/setup.bash' >> ~/.bashrc
```

Πίνακας 15. Η εντολή για το source στο bashrc αρχείο.

Μια ιδιαίτερα χρήσιμη εντολή του colcon είναι η colcon_cd. Μέσω αυτής της εντολής μπορεί να επιτευχθεί η μετάβαση σε οποιοδήποτε πακέτο του workspace, ασχέτως της τωρινής καρτέλας που βρίσκεται ένα τερματικό. Ο τρόπος κλήσης της εντολής φαίνεται στο παρακάτω πίνακα.

```
$ colcon_cd <package_name>
```

Πίνακας 16. Χρήση της εντολής colcon_cd.

Με χρήση της παραπάνω εντολής γίνεται η μετάβαση εντός του workspace, του φακέλου src και του πακέτου που δηλώθηκε. Για να μπορέσει να γίνει η χρήση του colcon_cd είναι απαραίτητη η εκτέλεση κάποιων επιπρόσθετων εντολών οι οποίες φαίνονται στο παρακάτω πίνακα και επεξεργάζονται το bashrc file των Linux.

```
$ echo "source /usr/share/colcon_cd/function/colcon_cd.sh" >> ~/.bashrc

$ echo "export _colcon_cd_root=/opt/ros/humble/" >> ~/.bashrc

$ echo "source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash" >> ~/.bashrc
```

Πίνακας 17. Επεξεργασία του bashrc αρχείου.

## 1.5 Εξοικείωση με χρήση του Turtlesim

Το turtlesim είναι ένα πακέτο προσομοίωσης τους ROS 2 το οποίο είναι ιδανικό για την εκμάθηση του λογισμικού. Είναι μια δισδιάστατη προσομοίωση μιας χελώνας μέσω της οποιάς επεξηγούνται οι πιο βασικές δυνατότητες του λογισμικού. Παράλληλα, επιτυγχάνει να δώσει μια ιδέα για το πως πραγματοποιείται η επικοινωνία μεταξύ των ROS module.

Το ros2 είναι μια command line εντολή και αποτελεί ένα πολύτιμο εργαλείο του ROS 2, με το οποίο δίνεται η δυνατότητα να γίνει η διαχείριση του συστήματος, να πραγματοποιηθεί ενδοσκόπηση και να επιτευχθεί αλληλεπίδραση με όλο το δίκτυο του ROS. Υποστηρίζει πολλές εντολές στο τερματικό που στοχεύουν σε διαφορετικές πτυχές του συστήματος. Κάποιες από αυτές τις πτυχές μπορεί να είναι η εκκίνηση ενός ή πολλών κόμβων (node) ταυτόχρονα μέσω launch αρχείων, η αλλαγή κάποιων παραμέτρων ή η λήψη πληροφοριών από κάποιο module καθώς και πολλές ακόμη διεργασίες.

Το ros2 είναι ένα εργαλείο το οποίο εγκαθίσταται κατά τη διαδικασία εγκατάστασης του λογισμικού. Άλλο ένα πολύ σημαντικό εργαλείο του ROS αποτελεί το rqt, το οποίο είναι ένα εργαλείο που εμπεριέχει γραφικό περιβάλλον. Οποιαδήποτε διεργασία πραγματοποιείται με χρήση του rqt μπορεί να γίνει και με χρήση εντολών στο τερματικό, ωστόσο το rqt προσφέρει ένα πολύ φιλικό και εύκολο τρόπο για να γίνονται διεργασίες στο ROS 2.

Η εγκατάσταση του turtlesim και του rqt επιτυγχάνονται με τη χρήση των παρακάτω εντολών :

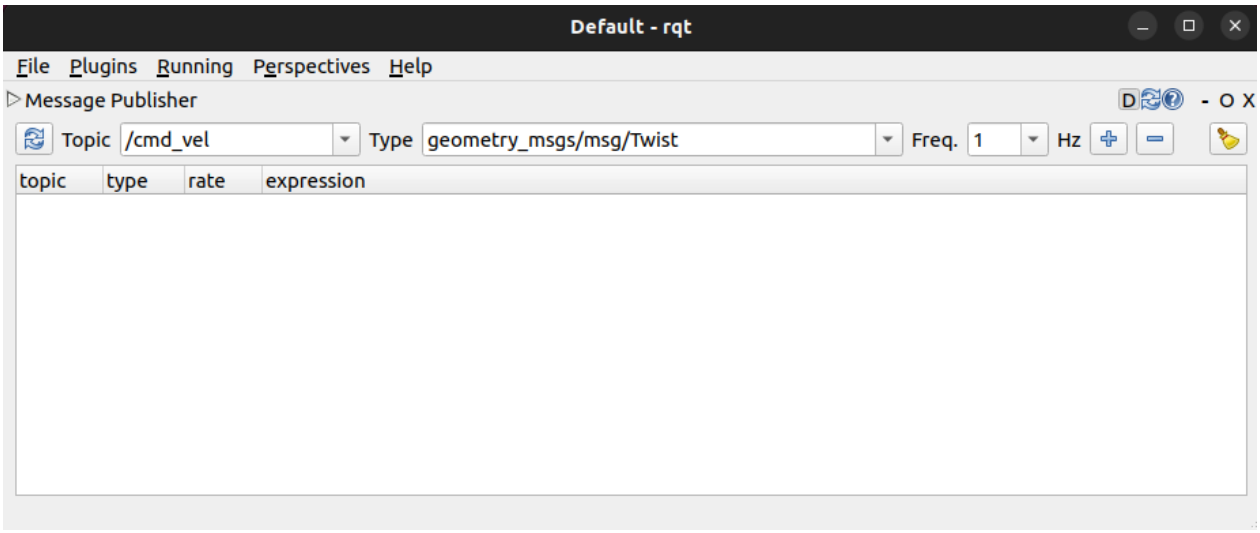
```
$ sudo apt update

$ sudo apt install ros-humble-turtlesim

$ sudo apt install ~nros-humble-rqt*
```

Πίνακας 18. Εγκατάσταση του turtlesim και του rqt.

Η εκκίνηση του rqt επιτυγχάνεται με χρήση της παρακάτω εντολής:

```
$ rqt
```

Πίνακας 19. Εκκίνηση του rqt.




Εικόνα 4. Το γραφικό περιβάλλον του rqt.

Το γραφικό περιβάλλον του rqt προσφέρει την δυνατότητα να επιλεγούν τα διαθέσιμα plugins του συστήματος. Έπειτα από την επιλογή τους μπορεί να γίνει η διαχείρηση του συστήματος και να πραγματοποιηθούν διεργασίες που αφορούν τα ρομπότ.

### 1.5.1 ROS nodes

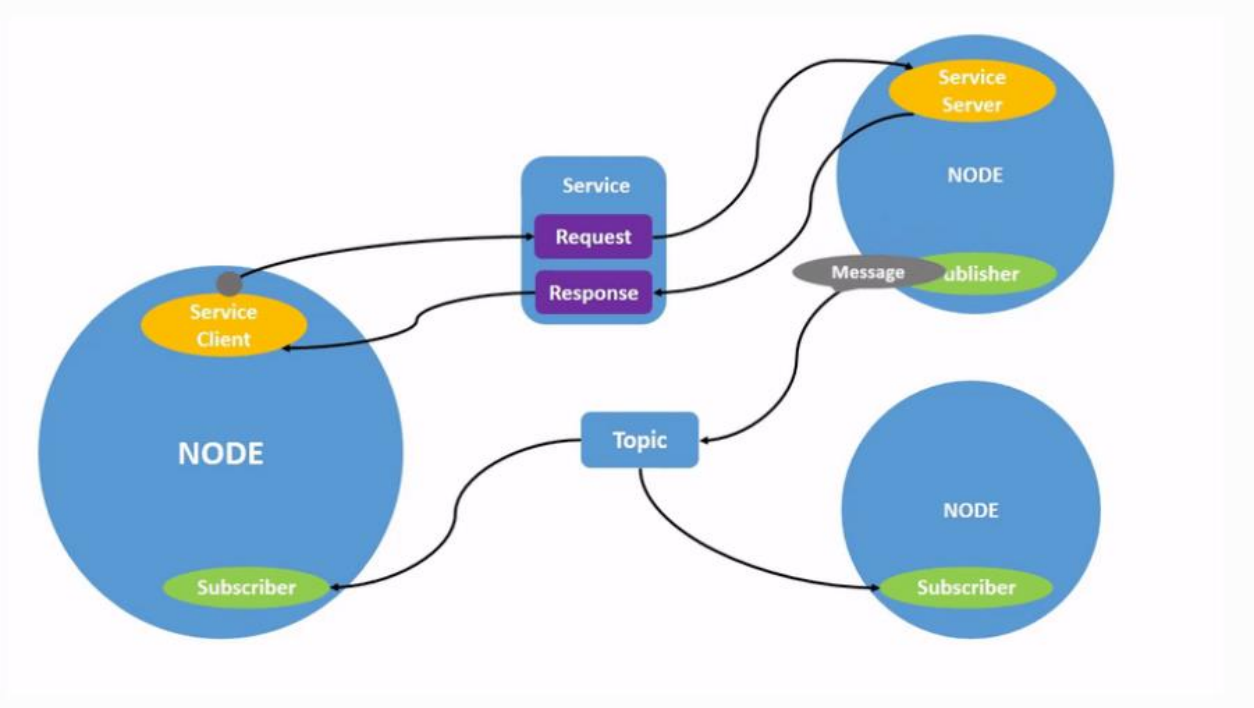
Ένας από τους βασικούς σκοπούς του ROS 2 είναι να διευκολύνει την επικοινωνία μεταξύ κάποιων ROS modules, τα οποία ονομάζονται nodes (κόμβοι). Αυτοί οι κόμβοι αναπαριστούν κάποια εκτελέσιμα και είναι υπεύθυνα για ένα και μόνο σκοπό κάθε φορά. Για παράδειγμα, σε ένα ρομπότ μπορεί να υπάρχει ένα node το οποίο θα ελέγχει τους σερβοκινητήρες των τροχών του ή μπορεί να υπάρχει ένα άλλον node το οποίο λαμβάνει μετρήσεις απόστασης από εμπόδια χρησιμοποιώντας κάποιο αισθητήρα μέτρησης αποστάσεων.

Ένα node μπορεί να δημιουργηθεί ως μέρος ενός κώδικα και μπορεί να είναι γραμμένο σε Python ή C++ χρησιμοποιώντας τις βιβλιοθήκες rclpy ή rclcpp, ανάλογα με την γλώσσα που έχει προτιμηθεί. Επίσης τα nodes μπορούν να δημιουργηθούν και μέσω κάποιων configuration files που είναι ανεπτυγμένα σε yaml μορφή. Ωστόσο αρκετές φορές θα εκκινούν nodes που έχουν δημιουργήσει άλλοι προγραμματιστές και εξυπηρετούν καλύτερα κάποιο σκοπό. Συνήθως τα nodes θα εκκινούν μέσω της δημιουργίας και εκτέλεσης κάποιων launch files, με χρήση των οποίων θα επιτυγχάνεται η εκκίνηση πολλαπλών κόμβων για ένα ρομποτικό σύστημα.

Ένα node του ROS 2 μπορεί να αναπαριστά μόνο μια λειτουργία του συστήματος και ενίοτε δημοσιεύει πληροφορίες (publish) και άλλοτε κάνει λήψη διάφορων πληροφοριών (subscribe). Δηλαδή τα nodes, στέλνουν ή λαμβάνουν πληροφορία από κάποια άλλα ROS modules τα οποία είναι τα topics, τα actions ή τα services.

Η επικοινωνία των nodes με τα άλλα modules επιτυγχάνεται μέσω του ROS Graph. Το ROS Graph είναι ένα δίκτυο από στοιχεία του ROS 2 το οποίο επεξεργάζεται δεδομένα που παράγονται την ίδια στιγμή. Περικλείει όλα τα εκτελέσιμα και τις διασυνδέσεις μεταξύ των modules του ROS, δηλαδή είναι υπεύθυνο για την επικοινωνία των ROS modules.

Σε ένα πλήρης ρομποτικό σύστημα, θα υπάρχουν αρκετά nodes τα οποία θα πρέπει να λειτουργούν με αρμονία. Στο ROS 2 ένα εκτελέσιμο, μπορεί να εμπεριέχει περισσότερα από ένα nodes τα οποία και θα ενεργοποιηθούν. Πληροφοριακά, στην προηγούμενη έκδοση του ROS, η επικοινωνία μεταξύ των ROS modules επιτυγχάνονταν μέσω του ROS Master. Η έννοια του ROS Master δεν υφίσταται στην νεότερη έκδοση του ROS και επικοινωνία επιτυγχάνεται μέσω άλλων πρωτοκόλλων επικοινωνίας.

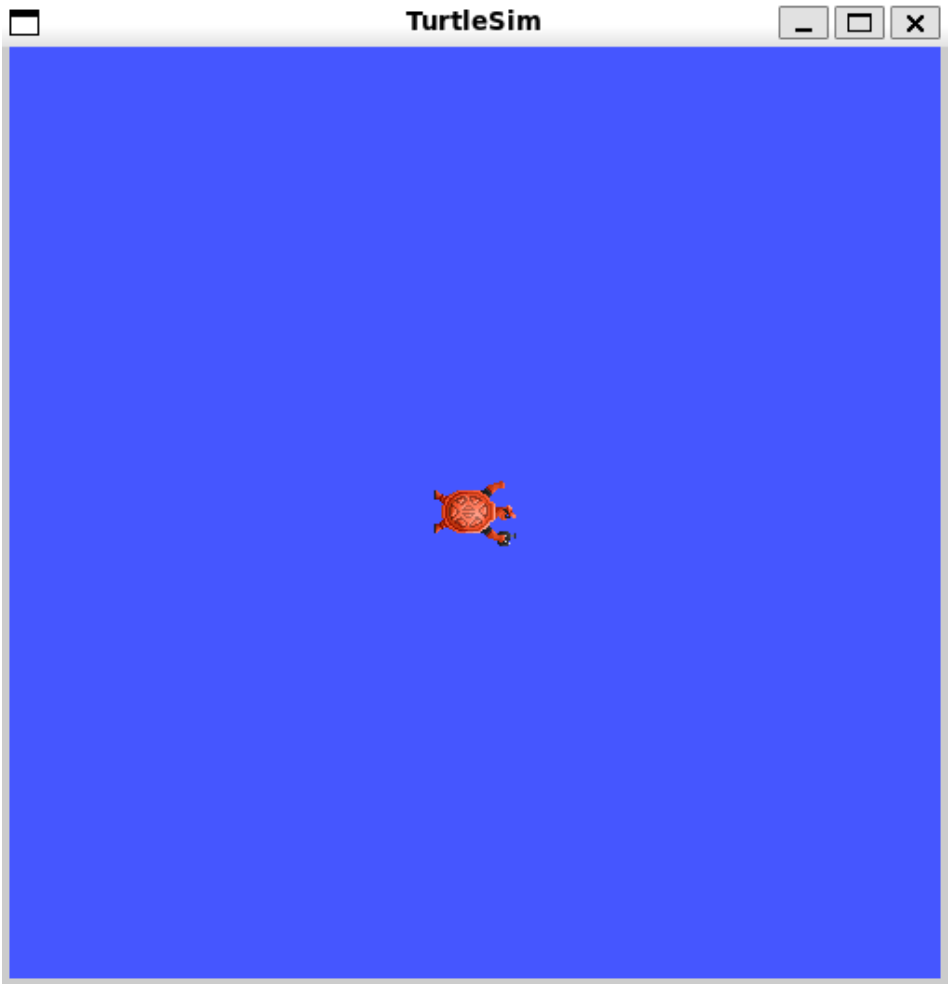



Εικόνα 5. Παράδειγμα ενός ROS Graph.

Συνεπώς τα nodes, είναι το σημαντικότερο module του ROS για την επικοινωνία του δικτύου και ανάλογα με το Ros module που συνεργάζονται χρησιμοποιούν μια συγκεκριμένη μορφή επικοινωνίας. Αφού έχει γίνει κατανοητή η χρησιμότητα των ROS nodes, ο παρακάτω πίνακας παρουσιάζει το τρόπο έναρξης ενός node σε γενική μορφή αλλά και συγκεκριμένα στο turtlesim.

| |
|---|
| $ #ros2 run <package_name> <executable_name> |
| $ ros2 run turtlesim turtlesim_node |

Πίνακας 20. Κλήση του turtlesim node.




Εικόνα 6. Προσομοίωση του turtlesim.

Αφού εκτελεστεί η εντολή του παραπάνω πίνακα εκκινεί ένα node και παράλληλα ανοίγει ένα νέο γραφικό περιβάλλον. Όταν εκκινεί ένα node στο δίκτυο, υπάρχουν κάποιες ιδιαίτερα χρήσιμες εντολές που βοηθούν στη κατανόηση του συστήματος. Συγκεκριμένα μπορεί να δοθεί κατάλληλη εντολή που δίνει στην έξοδο όλους τους ενεργούς κόμβους που υπάρχουν τη δεδομένη στιγμή στο ROS δίκτυο. Παράλληλα προσφέρεται εντολή που παρουσιάζει χρήσιμες πληροφορίες του συστήματος που μπορεί να χρειαστούν, όπως από που λαμβάνει ή δέχεται πληροφορία το node.

| |
|---|
| $ ros2 node list<br><br>$ # ros2 node info <node_name><br><br>$ ros2 node info /turtlesim |

Πίνακας 21. Βοηθητικές εντολές για τα ROS nodes.

Τα αποτελέσματα των παραπάνω εντολών είναι τα ακόλουθα:

```
katos@KATOS:~/ros2_ws$ ros2 node list
/turtlesim
katos@KATOS:~/ros2_ws$ ros2 node info /turtlesim
/turtlesim
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /turtle1/cmd_vel: geometry_msgs/msg/Twist
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
    /turtle1/color_sensor: turtlesim/msg/Color
    /turtle1/pose: turtlesim/msg/Pose
  Service Servers:
    /clear: std_srvs/srv/Empty
    /kill: turtlesim/srv/Kill
    /reset: std_srvs/srv/Empty
    /spawn: turtlesim/srv/Spawn
    /turtle1/set_pen: turtlesim/srv/SetPen
    /turtle1/teleport_absolute: turtlesim/srv/TeleportAbsolute
    /turtle1/teleport_relative: turtlesim/srv/TeleportRelative
    /turtlesim/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /turtlesim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /turtlesim/get_parameters: rcl_interfaces/srv/GetParameters
    /turtlesim/list_parameters: rcl_interfaces/srv/ListParameters
    /turtlesim/set_parameters: rcl_interfaces/srv/SetParameters
    /turtlesim/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:
    /turtle1/rotate_absolute: turtlesim/action/RotateAbsolute
  Action Clients:
```

Εικόνα 7. Έξοδος εντολής ros2 node info.

Η πρώτη εντολή του πίνακα 21, θα επιστρέψει στην έξοδο μόνο ένα ενεργό node, το turtlesim. Το αποτέλεσμα που προκύπτει είναι απόλυτα λογικό αφού έχει ενεργοποιηθεί μόλις ένα node. Ωστόσο το ROS δίνει την δυνατότητα να υπάρχουν ενεργά όσα nodes είναι επιθυμητό. Συνεπώς οι παρακάτω εντολές θα ενεργοποιήσουν δύο νέα nodes.

```
$ ros2 run turtlesim turtlesim_node --ros-args --remap __node:=my_turtle # Second terminal

$ ros2 run turtlesim turtle_teleop_key # Third terminal

$ ros2 node list # Fourth terminal
```

Πίνακας 22. Ενεργοποίηση δύο νέων ROS nodes.

Η πρώτη εντολή του παραπάνω πίνακα, ενεργοποιεί ένα καινούργιο node αντίστοιχο με το turtlesim με το όνομα my_turtle. Η επόμενη ενεργοποιεί ένα επιπρόσθετο node που δίνει τη δυνατότητα αλληλεπίδρασης με το πληκτρολόγιο, δηλαδή η χελώνα μπορεί να μετακινηθεί με τα βελάκια του πληκτρολογίου.

Ιδιαίτερη προσοχή πρέπει να δοθεί στο γεγονός ότι κάθε node πρέπει να είναι ενεργό σε διαφορετικό τερματικό, το οποίο θα παραμένει ανοιχτό. Συνεπώς τα ενεργά nodes που υπάρχουν είναι τρία και για να ληφθούν πληροφορίες για τα nodes που είναι ενεργά αυτή τη στιγμή ή να πραγματοποιηθούν άλλες διεργασίες, η εκτέλεση εντολών θα γίνει σε ένα νέο τερματικό.

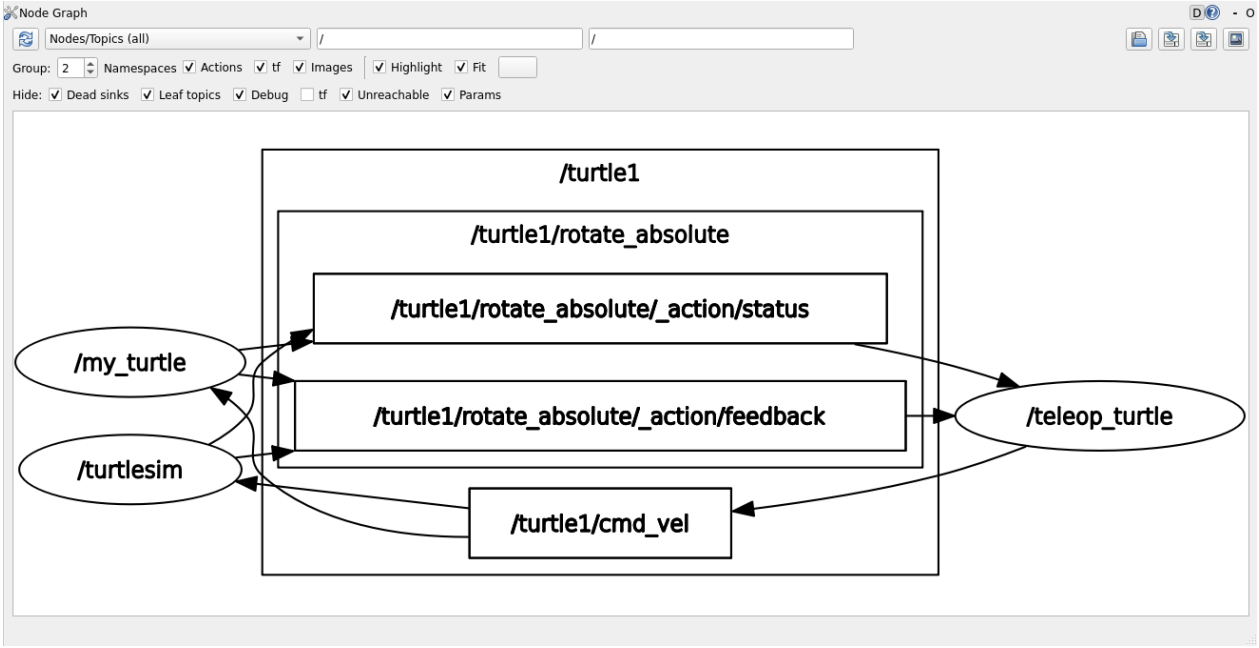
### 1.5.2 ROS topics

Το ROS 2 διαχωρίζει το σύνθετο σύστημα του σε πολλά modules. Τα topics αποτελούν ένα καθοριστικό ρόλο καθώς επιτυγχάνουν την εναλλαγή μηνυμάτων μεταξύ των nodes. Ένα node μπορεί να δημοσιεύει πληροφορίες (publish) σε πολλά topics και ταυτόχρονα μπορεί να δέχεται πληροφορίες (subscribe) από αρκετά topics. Τα topics είναι ο βασικός τρόπος μεταφοράς πληροφοριών μεταξύ των nodes και συνεπώς μεταξύ διάφορων λειτουργιών του συστήματος ενός ρομπότ.

Για να εξεταστεί η χρησιμότητα των ROS topics γίνεται χρήση του turtlesim συνεπώς θα πρέπει να είναι ενεργοποιημένα τα nodes που χρησιμοποιήθηκαν παραπάνω (Πίνακας 20, 22). Άλλο ένα πολύ ισχυρό εργαλείο του ROS είναι το rqt graph, με το οποίο δίνεται η δυνατότητα να ελεγχθεί ο τρόπος που γίνεται η επικοινωνία των nodes και topics που είναι ενεργά στο ROS δίκτυο με ένα γραφικό τρόπο.

```
$ rqt_graph
```

Πίνακας 23. Κλήση του rqt graph.




Εικόνα 8. Το rqt_graph.

Πληροφοριακά υπάρχει η δυνατότητα το rqt graph να εκκινήσει και μέσω του rqt. Για να επιτευχθεί αυτό πρέπει να εκτελέσει κατάλληλη εντολή για την εκκίνηση του περιβάλλοντος του rqt (Πίνακας 19). Έπειτα να επιλεγεί η κατάλληλη καρτέλα που είναι η ακόλουθη : **Plugins** >> **Introspection** >> **Node Graph**.

Το παραπάνω γράφημα δείχνει πως επιτυγχάνεται η επικοινωνία στο παρόν ROS δίκτυο. Πιο συγκεκριμένα οι ελλείψεις δείχνουν τα ενεργά nodes. Τα παραλληλόγραμμα είναι κάποια modules του ROS όπως τα topics και actions.

Το εργαλείο ros2 προσφέρει πολυπληθής εντολές για τα topics και μέσω αυτών μπορεί να παρθούν χρήσιμες πληροφορίες για τα ενεργά topics στο δίκτυο του ROS. Η πρώτη εντολή του παρακάτω πίνακα επιστρέφει τα ενεργά topics στο δίκτυο, ενώ η δεύτερη επιστρέφει παρόμοιο αποτέλεσμα με επιπρόσθετη πληροφορία τον τύπο μηνύματος που δέχεται κάθε topic.

| |
|---|
| $ ros2 topic list<br><br>$ ros2 topic list –t |

Πίνακας 24. Χρήσιμες εντολές για τα topics.

Το αποτέλεσμα εξόδου της δεύτερης εντολής του παραπάνω πίνακα είναι τα ακόλουθα:

| |
|---|
| /parameter_events [rcl_interfaces/msg/ParameterEvent] |
| /rosout [rcl_interfaces/msg/Log] |
| /turtle1/cmd_vel [geometry_msgs/msg/Twist] |
| /turtle1/color_sensor [turtlesim/msg/Color] |
| /turtle1/pose [turtlesim/msg/Pose] |

Πίνακας 25. Έξοδος εντολής ros2 topic list -t.

Η επικοινωνία στο δίκτυο του ROS, επιτυγχάνεται μέσω των ROS modules, ωστόσο άξιο αναφοράς είναι το γεγονός ότι η πληροφορία που μεταφέρεται στο δίκτυο πρέπει να έχει ένα συγκεκριμένο τύπου μηνύματος. Για παράδειγμα, η δήλωση επιθυμητής ταχύτητας δέχεται μηνύματα τύπου geometry_msgs /msg/Twist με κατάλληλα ορίσματα ενώ η πόζα του ρομπότ είναι μήνυμα τύπου turtlesim/msg/Pose.

Πολλές φορές είναι χρήσιμο να υπάρχει μια εικόνα για τα δεδομένα που γίνονται publish σε ένα topic. Μέσω αυτών των πληροφοριών μπορούν να εξαχθούν χρήσιμα συμπεράσματα για τη συμπεριφορά του ρομπότ καθώς και να πραγματοποιηθούν διορθώσεις μη επιθυμητών συμπεριφορών. Για να μπορέσουν να τυπωθούν τα δεδομένα από ένα topic θα πρέπει να είναι γνωστό το πλήρες όνομα του topic.

Με τις παρακάτω εντολές τυπώνεται η γραμμική και περιστροφική ταχύτητα που έχει το ρομπότ κάθε στιγμή. Επίσης διατίθεται και μια επιπρόσθετη εντολή, το αποτέλεσμα της οποίας τυπώνει διάφορες πληροφορίες για το συγκεκριμένο topic.

| |
|---|
| $ # ros2 topic echo <topic_name> |
| $ ros2 topic echo /*turtle1*/cmd_vel |
| $ ros2 topic info /turtle1/cmd_vel |

Πίνακας 26. Επιπρόσθετες εντολές για τα topics.

Εκτός από το γεγονός ότι μπορούν να τυπωθούν τα δεδομένα που υπάρχουν σε ένα topic, πολλές φορές είναι επιθυμητό να σταλεί πληροφορία σε ένα topic, δηλαδή να γίνει publish πληροφορία. Για παράδειγμα, μπορεί να δηλωθεί η επιθυμητή ταχύτητα του ρομπότ-χελώνα. Για να επιτευχθεί αυτό είναι απαραίτητη η γνώση του topic αλλά και του είδους του μηνύματος που δέχεται αυτό.

Αφού υπάρχει η γνώση του topic και του τύπου μηνύματος, τότε πρέπει να υπάρχει η γνώση για τα ορίσματα που θα δεχτεί το topic. Συνεπώς το αποτέλεσμα της παρακάτω εντολής, θα επιστρέψει τα ορίσματα που περιμένει να δεχθεί ο συγκεκριμένος τύπος μηνύματος. Πιο συγκεκριμένα, για τη δήλωση επιθυμητής ταχύτητας, το topic αναμένει να του δοθούν δύο ορίσματα, δηλαδή ένα διάνυσμα για την γραμμική ταχύτητα και ένα για την γωνιακή.

```
$ ros2 interface show geometry_msgs/msg/Twist
```

Πίνακας 27. Εύρεση των ορισμάτων.

Αφού υπάρχει η γνώση του topic, του είδους μηνύματος αλλά και των ορισμάτων που δέχεται αυτό, ακολουθεί η αποστολή δεδομένων. Με την παρακάτω εντολή μπορεί επιτευχθεί η δημοσίευση δεδομένων σε ένα topic με εντολές command line:

```
$ # ros2 topic pub <topic_name> <msg_type> '<args>'

$ ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"

$ros2 topic pub --rate 1 /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"
```

Πίνακας 28. Δημοσίευση πληροφοριών σε ένα topic.

Η πρώτη εντολή του παραπάνω πίνακα δείχνει την γενική μορφή που πρέπει να έχει η εντολή που δημοσιεύει δεδομένα σε ένα ROS topic. Στη δεύτερη εντολή στέλνετε το μήνυμα μια φορά ενώ στην τρίτη μια φορά το δευτερόλεπτο.

Στο σημείο αυτό θα παρατηρηθεί η κίνηση του ρομπότ χελώνας στο γραφικό περιβάλλον. Παράλληλα θα παρατηρηθεί και η μεταβολή στη πληροφορία που έχει το topic.

### 1.5.3 ROS Services

Τα services είναι μια ακόμη μέθοδος επικοινωνίας μεταξύ των nodes στο ROS δίκτυο. Η λειτουργία των services είναι βασισμένη στο μοντέλο call and response (κλήση και απάντηση). Το μοντέλο αυτό διαφέρει από το publisher-subscriber των topics.

Τα topics επιτρέπουν στα nodes να κάνουν subscribe ροές δεδομένων και να δέχονται updates συνέχεια, εν αντιθέση τα ROS services έχουν μια άλλη μορφή επικοινωνίας. Συγκεκριμένα τα services προσφέρουν

δεδομένα μόνο όταν τα καλέσει ένας client-node και όχι με ένα συνεχή τρόπο όπως στα topics. Για να γίνει αισθητή η διαφορά των services με τα topics θα πραγματοποιηθεί η χρήση της προσομοίωσης του turtlesim για την κατανόηση των διαφοροποιήσεων.

Για να μπορέσει να γίνει η μελέτη των ROS services, πρέπει να ενεργοποιηθούν κάποια ROS nodes. Σε δύο νέα τερματικά εκτελούνται οι εντολές του Πίνακα 22. Οι εντολές αυτές εκκινούν το turtlesim node καθώς και το turtle_teleop_key/ node που επιτρέπει την πλοήγηση του ρομπότ-χελώνα με χρήση του πληκτρολογίου.

Όπως για τα nodes και topics, το εργαλείο ros2 διαθέτει πληθώρα εντολών για τα services. Με κατάλληλη εντολή τυπώνονται όλα τα ενεργά services που υπάρχουν στο δίκτυο και ο τύπος μηνύματος τους.

```
$ ros2 service list –t

$ # ros2 service type <service_name>

$ ros2 service type /clear
```

Πίνακας 29. Χρήσιμες εντολές για τα services.

Επισης δίνεται η δυνατότητα να βρεθούν τα services που είναι ενεργά βάση τον τύπο κάποιου μηνύματος.

```
$ # ros2 service find <type_name>

$ ros2 service find std_srvs/srv/Empty
```

Πίνακας 30. Εύρεση ενός service βάση του τύπου.

Τα services είναι ένας τρόπος επικοινωνίας μεταξύ των nodes. Η διαφορά του με τα topics είναι ότι η ροή δεδομένων μέσω services επιτυγχάνεται μόνο όταν το επιθυμεί ο χρήστης. Για να γίνει η κλήση ενός service, πρέπει να υπάρχει η γνώση του service το οποίο θα χρησιμοποιηθεί καθώς επίσης και ο τύπος μηνύματος που δέχεται αυτό αλλά και τα ορίσματα που πρέπει να του δοθούν.

Οι παρακάτω εντολές δείχνουν πως πραγματοποιείται η εύρεση των ορισμάτων που δέχεται ένα service:

```
$ # ros2 interface show <type_name>

$  ros2 inteface show std_srvs/srv/Empty

---
```

```
$ ros2 interfaces show turtesim/srv/Spawn

float32 x
float32 y
float32 theta
string name # Optional.  A unique name will be created and returned if this is empty
---
string name
```

Πίνακας 31. Εύρεση ορισμάτων των services.

Παρατηρείται ότι η πρώτη εντολή έχει σαν αποτέλεσμα ένα κενό μήνυμα. Αυτό συμβαίνει διότι το συγκεκριμένο service δεν δέχεται κάποιο όρισμα. Η επόμενη εντολή του παραπάνω πίνακα, της οποίας η ιδιότητα είναι αναγεννήσει μια νέα χελώνα στο ήδη υπάρχων node, δέχεται σαν όρισμα ένα διάνυσμα τριών διαστάσεων το οποίο είναι η τοποθέτηση στην οποία θα αναγεννηθεί η νέα χελώνα.

Αφού υπάρχει η γνώση των παραπάνω, τότε μπορεί να γίνει η κλήση των services. Η κλήση των services επιτυγχάνεται με εκτέλεση των παρακάτω εντολών. Πιο συγκεκριμένα, στην πρώτη εντολή παρουσιάζεται η γενική μορφή της κλήση των services. Η δεύτερη εντολή αναγεννάει μια νέα χελώνα στο γραφικό περιβάλλον. Τέλος, η τρίτη εντολή καθαρίζει το γραφικό περιβάλλον από τα μονοπάτια που δημιούργησαν οι χελώνες καθώς κινούνταν στο περιβάλλον τους.

```
$ # ros2 sevice call <service_name> <service_type> <argument>

$ ros2 service call /spawn turtlesim/srv/Spawn "{x: 2, y: 2, theta: 0.2, name: 'turtle2'}"

$ ros2 service call clear std_srvs/srv/Empty # Clears the path of the turtles
```

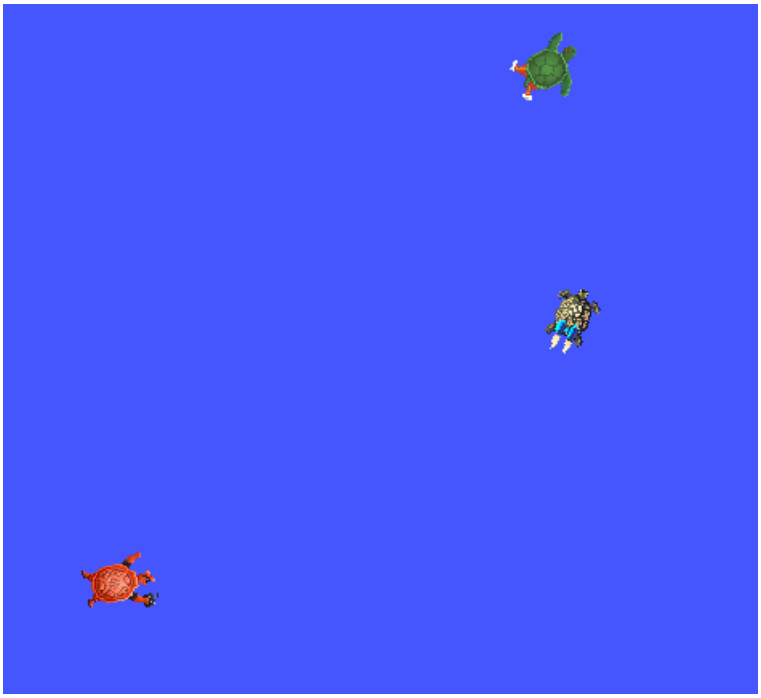
Πίνακας 32. Κλήση κάποιων ROS services.

Αφού αναγεννηθεί μια νέα χελώνα, για να μπορέσει να γίνει η οδήγηση της με χρήση του πληκτρολογίου μπορεί να δοθεί η παρακάτω εντολή.

```
$ ros2 run turtlesim turtle_teleop_key --ros-args --remap turtle1/cmd_vel:=turtle2/cmd_vel
```

Πίνακας 33. Αλληλεπίδραση πληκτρολογίου για τη δεύτερη χελώνα.

Εικόνα 9. Το νέο turtlesim node με τρεις χελώνες.



Εικόνα 10. Το node χωρίς την πορεία κάθε χελώνας.

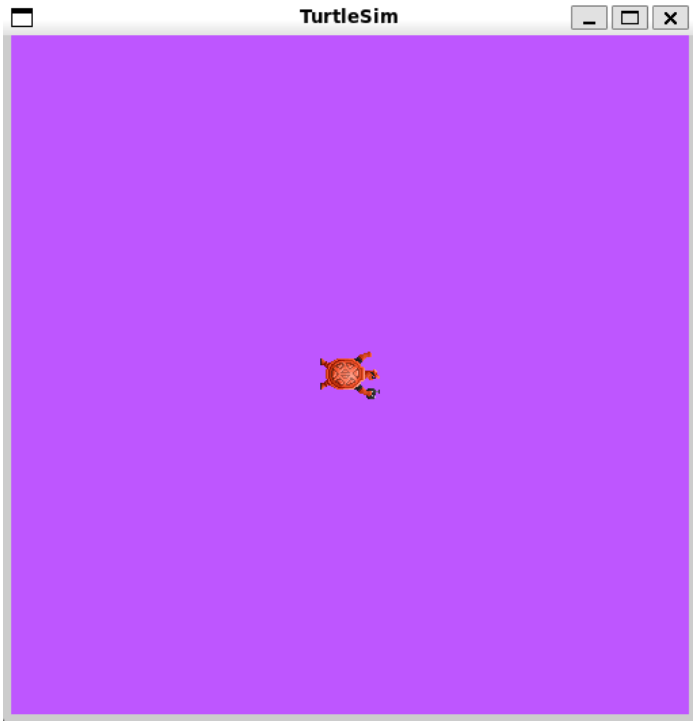
### 1.5.4. ROS parameters

Τα ROS parameters είναι η παραμετροποίηση των τιμών ενός node και συχνά παρομοιάζονται ως οι ρυθμίσεις αυτών. Ένα node μπορεί να αποθηκεύει τις παραμέτρους του ως αριθμούς με υποδιαστολή, ακεραίους, Boolean, συμβολοσειρές ή σε μορφή λίστας. Στο ROS 2, κάθε node διατηρεί τους δικούς της παραμέτρους.

Αξιοσημείωτες εντολές για τα ROS parameters είναι η εύρεση όλων των διαθέσιμων παραμέτρων που είναι ενεργοί στο δίκτυο, η εύρεση της τρέχουσας τιμής ενός parameter αλλά και η αλλαγή της.

| |
|---|
| $ ros2 param list |
| $ ros2 param get /turtlesim background_g |
| Integer value is: 86 |
| $ ros2 param set /turtlesim background_g 133 |
| Set parameter successful |

Πίνακας 34. Χρήσιμες εντολές για τα parameters.



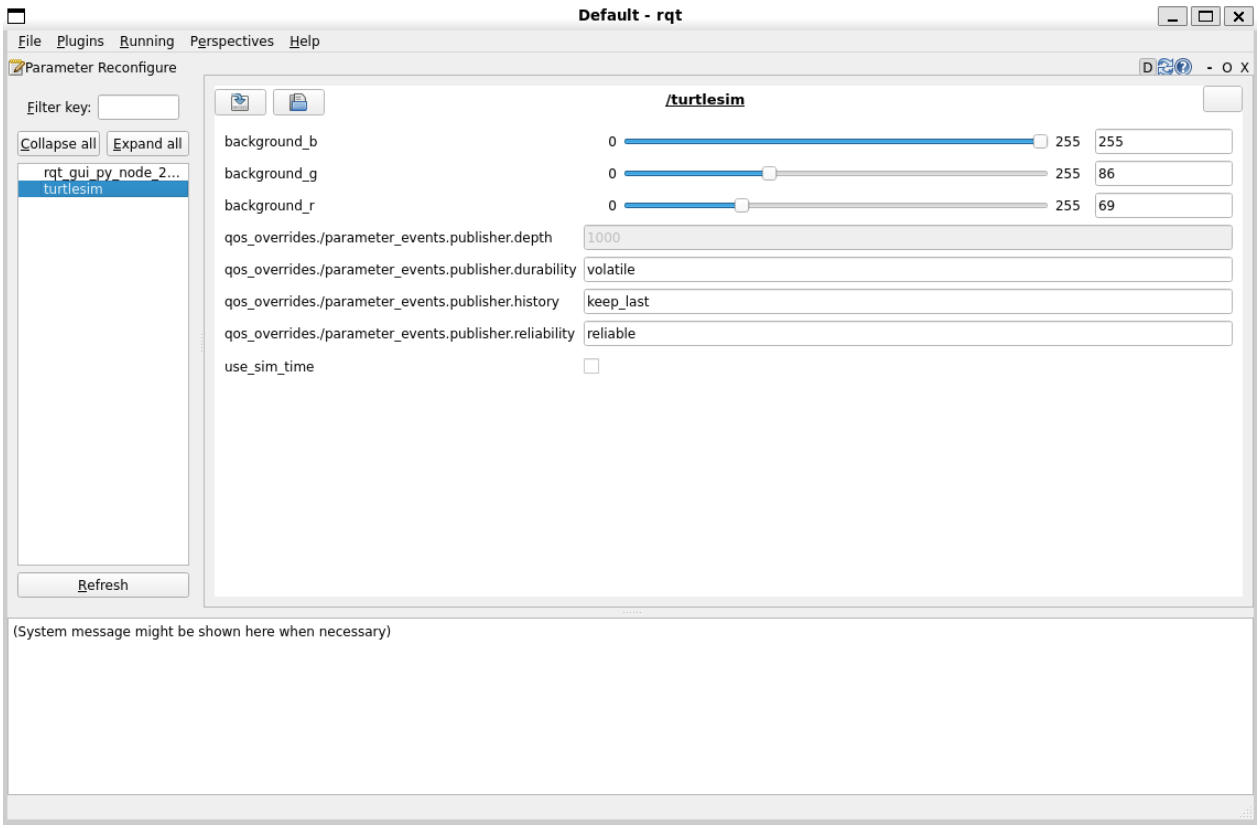

Εικόνα 11. Αλλαγή ενός parameter.

Ένα parameter του ROS μπορεί να είναι ένα αρχείο τύπου yaml. Τα αρχεία yaml είναι υπεύθυνα για τον ορισμό των παραμέτρων ενός node. Μπορεί να γίνει η φόρτωση ενός yaml file που έχει δημιουργηθεί νωρίτερα ή να αποθηκευτούν οι αλλαγές των παραμέτρων σε ένα νέο αρχείο.

```
$ ros2 run turtlesim turtlesim_node --ros-args --params-file my_turtlesim.yaml

$ ros2 param load /turtlesim turtlesim.yaml
```

Πίνακας 35. Αποθήκευση και εισαγωγή των parameters.

Οι αλλαγές στις παραμέτρους του συστήματος μπορούν να επιτευχθούν και με χρήση του εργαλείου rqt. Αφού εκκινήσει το rqt επιλέγεται η καρτέλα: **Plugins** >> **Configuration** >> **Dynamic Reconfigure**. Στην μπάρα που βρισκεται στο αριστερό μέρος του γραφικού περιβάλλοντος, επιλέγεται η χελώνα και θα εμφανιστεί το ακόλουθο γραφικό περιβάλλον στο οποίο μπορούν να πραγματοποιηθούν όλες οι παραπάνω διεργασίες.




Εικόνα 12. Το Dynamic Reconfigure.

### 1.5.5 ROS actions

Τα ROS actions είναι μια ακόμη μορφή επικοινωνίας μεταξύ των nodes στο ROS 2, βρίσκονται για πρώτη φορά στον πυρήνα του λογισμικού και προορίζονται για μακροσκελής εργασίες. Αποτελούνται από τρία μέρη: το στόχο, την αναφορά (feedback) και το αποτέλεσμα.

Τα actions είναι βασισμένα στα topics και τα services. Η λειτουργικότητα τους είναι παρόμοια με αυτή των services με τη διαφορά ότι τα actions μπορούν να ακυρωθούν. Επίσης παρέχουν μια διαρκής αναφορά της διεργασίας που έχουν αναλάβει, εν αντίθεση με τα services που απλώς επιστρέφουν μια απάντηση.

Το μοντέλο επικοινωνίας που χρησιμοποιούν τα actions είναι το call and response αλλά είναι αρκετά όμοιο με το μοντέλο publisher-subscriber που χρησιμοποιούν για την επικοινωνία τους τα topics. Αναλυτικότερα, υπάρχουν δύο nodes, εκ των οποίων το ένα είναι ο “actions client” και το άλλο ο “actions server”. Το πρώτο δημοσιεύει ένα στόχο στο δεύτερο το οποίο αναγνωρίζει το στόχο και επιστρέφει μια αναφορά και το αποτέλεσμα.

Η κατανόηση των ROS actions θα γίνει μέσω του turtlesim για αυτό και πρέπει να γίνει η ενεργοποίηση κάποιων nodes τα οποία παρουσιάζονται στον παρακάτω πίνακα. Ωστόσο για να κατανοηθούν τα actions πρέπει να γίνει η πλήρης κατανόηση της εξόδου της δεύτερη εντολής.

| |
|---|
| $ ros2 run turtlesim turtlesim_node<br><br>$ ros2 run rurtlesim turtlesim_teleop_key # **Second terminal**<br><br>Use arrow keys to move the turtle.<br>Use G\|B\|V\|C\|D\|E\|R\|T keys to rotate to absolute orientations. 'F' to cancel a rotation. |

Πίνακας 36. Έξοδος του turtlesim_teleop_key.

Η έξοδος της δεύτερης εντολής συνοδεύεται από κάποιες οδηγίες χρήσεως. Το ROS ενημερώνει ότι υπάρχει η δυνατότητα ανάδραση της χελώνας με το πληκτρολόγιο. Δηλαδή η χελώνα μπορεί να μετακινηθεί με κατάλληλα πλήκτρα.

Συγκεκριμένα όλα τα γειτονικά πλήκτρα του ‘F’, έχουν την δυνατότητα να περιστρέψουν την χελώνα προς κάποια χαρακτηριστικά σημεία του χώρου. Για παράδειγμα, το πλήκτρο ‘E’ περιστρέφει τη χελώνα με τέτοιο τρόπο ώστε να κοιτάζει προς την πάνω αριστερή γωνία. Σε περίπτωση που επιλεγεί το ‘E’, τότε ενεργοποιείται ένα action με στόχο την περιστροφή της χελώνας στην επιθυμητή κατεύθυνση. Παράλληλα

τυπώνεται και μια διαρκής ενημέρωση για την λειτουργία του action. Η σπουδαιότητα που έχουν τα actions είναι ότι μπορούν να ακυρωθούν και αυτό επιτυγχάνεται πληκτρολογώντας το 'F'.

Όπως και για τα υπόλοιπα ROS modules, και τα action συνοδεύονται από πληθώρα εντολών που βοηθούν στην κατανόηση του συστήματος. Με τις παρακάτω εντολές τυπώνονται χρήσιμες πληροφορίες για όλα τα ενεργά actions. Πιο συγκεκριμένα, τυπώνεται μια λίστα των ενεργών actions αλλά και ο τύπος μηνύματος αυτών. Επιπρόσθετα, η χρήση ενός action απαιτεί τη γνώση του τύπου μηνύματος, αλλά και τα ορίσματα που δέχεται αυτό.

```
$ ros2 action list

$ ros2 action list -t

$ ros2 action info /turtle1/rotate_absolute

$ ros2 interfaces show turtlesim/action/RotateAbsolute
```

Πίνακας 37. Χρήσιμες εντολές για τα actions.

Αφού υπάρχει η γνώση του action, του τύπου μηνύματος αλλά και των ορισμάτων του, τότε μπορεί να σταλεί ο επιθυμητός στόχος προς το action. Με την παρακάτω εντολή, στέλνετε ένα επιθυμητός στόχος σε ένα action, το οποίο θα προσπαθήσει να περιστρέψει την χελώνα προς τη συγκεκριμένη κατεύθυνση που του δηλώνεται.

```
$ ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.57}"
```

Πίνακας 38. Αποστολή δεδομένων στο action.

Τα actions είναι παρόμοια με τα services τα οποία επιτρέπουν την εκτέλεση μακροχρόνιων στόχων. Ωστόσο προσφέρουν συχνή ενημέρωση και η λειτουργία τους μπορεί να ακυρωθεί οποιαδήποτε στιγμή. Τα actions βρίσκουν ευρεία εφαρμογή σε ένα ρομποτικό σύστημα το οποίο έχει σαν στόχο να πλοηγηθεί (navigation).

### 1.6 Διαφορές του ROS με το ROS 2

Η πρώτη έκδοση του ROS κυκλοφόρησε το 2007, αναπτύχθηκε αρχικά από την Willow Garage και αμέσως αποτέλεσε μια από τις δημοφιλέστερες πλατφόρμες στη ρομποτική κοινότητα. Ωστόσο από την κυκλοφορία του και έπειτα έχουν αλλάξει αρκετά στο χώρο του προγραμματισμού και της πληροφορικής. Παράλληλα έχουν παρατηρηθεί αρκετά χαρακτηριστικά του λογισμικού που λείπουν ή που μπορούν να βελτιωθούν. Πιο συγκεκριμένα, στο ROS παρατηρούνται αδυναμίες όπως είναι οι real-time εφαρμογές και η ασφάλεια.

Η πρώτη έκδοση του ROS είναι αρκετά διαδεδομένη στην εκπαιδευτική κοινότητα ενώ το αντίθετο συμβαίνει στον τομέα των επιχειρήσεων. Σε αντίθεση το ROS 2 αναπτύχθηκε ώστε να έχει ευρεία εφαρμογή σε πιο εμπορικές εφαρμογές και παράλληλα να διορθώσει τις αδυναμίες της προηγούμενης έκδοσης.

Συνολικά οι διαφορές μεταξύ των δύο λογισμικών εστιάζουν σε 3 συγκεκριμένες περιοχές οι οποίες είναι οι ακόλουθες:

- Δημιουργία των nodes.
- Επικοινωνία.
- Πακέτα, workspace και περιβάλλον.

Όσον αφορά τη δημιουργία των nodes εντοπίζεται μια αρκετά μεγάλη διαφορά. Αρχικά το ROS 2 υποστηρίζει τις τελευταίες εκδόσεις την Python και Cpp εν αντιθέση με το ROS που χρησιμοποιεί παλαιότερες εκδόσεις. Η σημαντικότερη διαφορά των δύο λογισμικών είναι οι βιβλιοθήκες στις οποίες στηρίζεται η δημιουργία των nodes. Αναλυτικότερα, στο ROS χρησιμοποιείται η βιβλιοθήκη roscpp για την γλώσσα Cpp ενώ για την Python η rospy. Το πρόβλημα που εντοπίζεται στις δύο βιβλιοθήκες είναι ότι έχουν αναπτυχθεί ξεχωριστά με αποτέλεσμα αρκετές διεργασίες της μιας βιβλιοθήκης να μην υποστηρίζονται από την άλλη και το αντίθετο.

Στη ανανεωμένη έκδοση του ROS, υπάρχει μια βιβλιοθήκη η οποία ονομάζεται rcl και έχει αναπτυχθεί βασισμένη στην γλώσσα C. Ωστότο η βιβλιοθήκη αυτή δεν χρησιμοποιείται αμέσως από τους χρήστες. Πιο συγκεκριμένα, έχουν χτιστεί δύο νέες βιβλιοθήκες που στηρίζονται στην rcl, μια για την Python (rclpy) και μια για την Cpp (rclcpp). Αυτή η αλλαγή του ROS 2 προσφέρει τρία σπουδαία χαρακτηριστικά. Αρχικά οι δύο βιβλιοθήκες έχουν παρόμοια χαρακτηριστικά, με αποτέλεσμα να μπορούν να υλοποιηθούν οι επιθυμητές διεργασίες σε οποιαδήποτε γλώσσα. Επιπρόσθετα κάθε νέο χαρακτηριστικό θα είναι διαθέσιμο πιο γρήγορα και τέλος μπορεί να διευκολυνθεί η χρήση βιβλιοθηκών όπως είναι η rcljava και rclnodejs.

Μια επιπρόσθετη διαφορά μεταξύ των δύο εκδόσεων του ROS αφορά τα launch αρχεία. Αυτά τα αρχεία είναι υπεύθυνα να εκκινούν πολλά nodes ταυτόχρονα καθορίζοντας ακριβώς κάποιες παραμέτρους και ρυθμίσεις του συστήματος. Η διαφορά εντοπίζεται στη γλώσσα προγραμματισμού που χρησιμοποιείται για την ανάπτυξη τέτοιων αρχείων. Η πρώτη έκδοση χρησιμοποιεί αποκλειστικά τη γλώσσα XML ενώ το ROS 2 χρησιμοποιεί την Python και την XML. Ωστόσο, προτιμάται η πρώτη από τις δύο γλώσσες για δύο λόγους, αφενός γιατί προσφέρει περισσότερες επιλογές στο χρήστη και αφετέρου είναι πιο καλά τεκμηριωμένη.

Το ROS 2 εισάγει και δύο νέα χαρακτηριστικά τα οποία είναι τα ROS components και τα lifecycled nodes. Το πρώτο χαρακτηριστικό μπορεί να βρεθεί και στην πρώτη έκδοση του ROS με την ονομασία Nodelets. Πιο συγκεκριμένα, το χαρακτηριστικό αυτό πλέον βρίσκεται στον πυρήνα του ROS και βοηθάει στην εκκίνησει πολλαπλών nodes μέσω ενός εκτελέσιμου. Δηλαδή, μπορούν να εκκινήσουν δύο nodes μόνο με χρήση μιας εντολής. Τα components μπορούν να εκκινήσουν μέσω launch files ή εντολές τερματικού και μπορούν να διευκολύνουν την επικοινωνία μεταξύ του ROS δικτύου.

Τα lifecycled nodes έχουν διαφορετικές καταστάσεις οι οποίες είναι: unconfigured, inactive, active και finalized. Είναι αρκετά χρήσιμα όταν προσπαθεί να στηθεί το σύστημα πριν αυτό αρχίσει να εκτελείται. Όταν ξεκινάει ένα τέτοιο node βρίσκεται στην κατάσταση unconfigured. Μέσα από τις διαθέσιμες διεπαφές μπορεί να ζητηθεί η μετάβαση σε μια νέα κατάσταση. Όταν αλλάξει η κατάσταση τότε μια άλλη προκαθορισμένη κατάσταση θα ενεργοποιηθεί εντός του node.

Όσον αφορά την επικοινωνία που υπάρχει στο ROS δίκτυο παρατηρείται μια μεγάλη διαφορά. Στο ROS η επικοινωνία επιτυγχάνεται με χρήση του ROS Master ο οποίος είναι υπεύθυνος για την επικοινωνία όλων των feature του δικτύου. Στο ROS 2, δεν υπάρχει ένα κεντρικό σύστημα επικοινωνίας, αλλά κάθε node μπορεί να ανακαλύψει τα υπόλοιπα nodes. Επιπρόσθετα, όταν δημιουργείται μια εφαρμογή που απαιτεί την συνεργασία δύο μηχανών, δεν χρειάζεται να καθοριστεί κάποια συσκευή ως master. Κάθε μηχάνημα θα είναι ανεξάρτητο και θα μπορεί να ξεκινάει από μόνο του.

Τα ROS action είναι μια νέα προσθήκη στο λογισμικό, μέσω των οποίων επιτυγχάνεται μια καινούργια μορφή επικοινωνίας στο δίκτυο. Πληροφοριακά τα actions είχαν προστεθεί από την προηγούμενη έκδοση του ROS ως μέρος των topics. Το συγκεκριμένο χαρακτηριστικό, εισήχθει στο ROS με στόχο να αντιμετωπιστούν κάποια προβλήματα που παρατηρήθηκαν στα services. Τα services είναι ασύγχρονα και δεν προσφέρουν κάποιο feedback ούτε κάποιο μηχανισμό ακύρωσης. Τα actions αντιμετωπίζουν με επιτυχία αυτά τα προβλήματα και πλέον αποτελούν ένα ξεχωριστο feature του ROS.

Το Quality of Service (QoS) εισάγεται στο ROS 2. Είναι ένα νέο χαρακτηριστικό του λογισμικού το οποίο δίνει τη δυνατότητα χειρισμού επικοινωνίας των nodes. Πιο συγκεκριμένα το QoS, μπορεί να χειριστεί τα μηνύματα προς τα nodes, με διάφορους τρόπους. Για παράδειγμα, είναι επιθυμητό να στέλνονται όλα τα μηνύματα από ένα topic προς ένα node ή μπορούν κάποια να απορριφθούν. Επίσης, σε περίπτωση που σταλεί ένα καινούργιο μήνυμα τι θα συμβεί αν η επεξεργασία των προηγούμενων δεδομένων δεν έχει ολοκληρωθεί.

Όσον αφορά το περιβάλλον, workspace και τα πακέτα παρατηρούνται και εκεί αρκετά σημαντικές διαφορές. Αρχικά το ROS 2 είναι συμβατό και σε λογισμικά όπως είναι τα Windows, MacOs αλλά και τα

Linux που ήταν το μοναδικό που υποστηρίζεται από την προηγούμενη έκδοση. Με αυτή την αλλαγή μπορεί να επεκταθεί η χρήση του λογισμικού σε περισσότερους χρήστες, ενώ παράλληλα όλα τα λογισμικά μπορούν να συνεργαστούν μεταξύ τους και να δημιουργήσουν ένα ρομποτικό σύστημα.

Μια επιπρόσθετη διαφορά των δύο λογισμικών είναι στον τρόπο κατασκευής των πακέτων εντός των workspaces. Πιο συγκεκριμένα, στο ROS χρησιμοποιείται το catkin για το χτίσιμο και την εγκατάσταση των πακέτων, ενώ στην νέα έκδοση χρησιμοποιείται το ament για την δημιουργία πακέτων και το εργαλείο colcon για το χτίσιμο των αυτών. Για την μεταγλώττιση του κώδικα που έχει αναπτυχθεί θα χρησιμοποιείται το εργαλείο colcon build.

Στην πρώτη έκδοση του ROS δημιουργούνται πακέτα μόνο με χρήση της Cpp ωστόσο στο ROS 2 μπορούν να δημιουργηθούν πακέτα είτε με γλώσσα Python είτε με Cpp που υποστηρίζονται επισήμως. Ωστόσο μπορούν να δημιουργηθούν και πακέτα με χρήση της Java αλλά και με τη γλώσσα προγραμματισμού Rust. Ανάλογα με τον τρόπο κατασκευής του πακέτου αλλάζει και η φακελοδομή που δημιουργείται εντός του πακέτου και συνεπώς ο χειρισμός του πακέτου πριν από την διαδικασία μεταγλώττισης.

Κλείνοντας την ενότητα, άξιο αναφοράς αποτελεί το ερώτημα για το αν πρέπει να γίνει η μετάβαση στο νέο λογισμικό. Συνολικά το ROS 2 προσφέρει αναβαθμισμένες υπηρεσίες και βελτιώσεις συγκριτικά με την έκδοση του ROS. Η μετάβαση στην αναβαθμισμένη έκδοση δεν θα πρέπει να θεωρείται δεδομένη καθώς το ROS 1 είναι αρκετά ισχυρό, παρέχεται καλύτερη τεκμηρίωση κάποιων χαρακτηριστικών του και περιέχει κάποια επιπρόσθετα plugins. Ένα σημαντικό πρόβλημα του είναι ότι το μοναδικό distribution που είναι ενεργό αυτή τη στιγμή είναι το Ros Noetic το οποίο και θα υποστηρίζεται μέχρι τον Ιανουάριο του 2025.

Συνεπώς η μετάβαση στις νέες τεχνολογίες που προσφέρει το ROS 2 είναι κάτι που αναμενόμενα θα επακολουθήσει. Ωστόσο, μπορεί να χρησιμοποιηθούν κάποιες γέφυρες επικονωνίας μεταξύ των δύο εκδόσεων μέσω του πακέτου ros1_bridge. Με τη σειρά του το ROS 2, προσφέρει ποικιλία νέων εργαλείων που διευκολύνουν τις διεργασίες που μπορούν να πραγματοποιηθούν με αυτό. Ωστόσο η κοινότητα του ROS 2 δεν είναι ιδιαίτερα πολυπληθής με συνέπεια κάποια πακέτα να μην είναι πλήρως τεκμηριωμένα και να μην υπάρχει αρκετό διαθέσιμο υλικό σε διαδικτυακές πηγές.

### 1.7 Το ROS 2 στο WSL2

Το Windows Subsystem for Linux ή εν συντομία WSL, είναι μια δυνατότητα των Windows που επιτρέπει στους προγραμματιστές να χρησιμοποιούν ένα περιβάλλον Linux χωρίς την ανάγκη ξεχωριστής εικονικής

μηχανής (πχ. Virtual Box) ή την χρήση διπλής εγκατάστασης λογισμικού στον ηλεκτρονικό υπολογιστή. Το ROS 2 μπορεί να εκτελεστεί με την ίδια αποτελεσματικότητα με χρήση του WSL.

Η εγκατάσταση του ROS 2 καθώς και όλες οι υπόλοιπες διεργασίες που παρουσιάζονται στις προηγούμενες ενότητες του κεφαλαίου υποστηρίζονται πλήρως από το WSL. Ωστόσο, το WSL είναι ένα περιβάλλον που συστήνεται περισσότερο σε προγραμματιστές που έχουν μια παλαιότερη εμπειρία στη χρήση τερματικού των Linux. Συνολικά η εκτέλεση του ROS 2 απαιτεί ένα υπολογιστή με αρκετά υψηλή υπολογιστική ισχύ και ιδιαίτερα όταν θα εκκινήσουν κάποιες προσομοιώσεις.

## Κεφάλαιο 2

## Προσομοίωση ενός δίτροχου ρομπότ

### 2.1 Δημιουργία ενός ROS package

Για τη προσομοίωση ενός ρομπότ αλλά και για την ανάπτυξη λογισμικών στο ROS 2 πολύ καθοριστικό ρόλο αποτελούν τα ROS 2 packages. Ένα πακέτο είναι μια οργανωτική μονάδα για τον κώδικα στο ROS 2. Για να μπορέσει να γίνει η εγκατάσταση ή η κοινοποίηση του κώδικα που δημιουργείται στο ROS 2, τότε αυτός θα πρέπει να είναι οργανωμένος σε ένα πακέτο. Μέσω των πακέτων μπορεί να δημοσιευτεί η δουλειά ενός προγραμματιστή και μπορεί να χρησιμοποιηθεί εύκολα και από άλλους χρήστες.

Το ROS 2 χρησιμοποιεί το εργαλείο ament για τη δημιουργία του πακέτου και το εργαλείο colcon για το χτίσιμο του πακέτου. Ένα πακέτο μπορεί να δημιουργηθεί χρησιμοποιώντας είτε CMake είτε Python τα οποία υποστηρίζονται επισήμως από το ROS 2 αλλά και με άλλους τρόπους. Πληροφοριακά στην προηγούμενη έκδοση του ROS τα πακέτα μπορούσαν να δημιουργηθούν μόνο μέσω του CMake χρησιμοποιώντας το εργαλείο catkin.

Το πακέτο που θα δημιουργηθεί παρακάτω θα είναι με χρήση της Python. Έπειτα από τη δημιουργία ενός ROS 2 Python πακέτου, αυτό διαθέτει κάποια απαιτούμενα περιεχόμενα. Αυτά είναι τα ακόλουθα:

- package.xml : Περιέχει κάποιες πληροφορίες για το πακέτο.
- resource/<package_name>: Είναι ένας Marker file.
- setup.cfg: Απαιτείται όταν το πακέτο έχει εκτελέσιμα, ώστε η εντολή ros2 run να μπορεί να τα βρει.
- setup.py : Περιέχει οδηγίες εγκατάστασης του πακέτου.
- <package_name> : Είναι ένας φάκελος με το ίδιο όνομα που έχει το πακέτο και χρησιμοποιείται από τα εργαλεία του ROS 2 για την εύρεση του πακέτου.

Κάθε νέο πακέτο του ROS 2 δημιουργείται μέσα σε ένα workspace και πιο συγκεκριμένα στο υποφάκελο src. Σε κάθε workspace μπορούν να δημιουργηθούν απεριόριστα πακέτα άσχετα με τον τρόπο κατασκευής τους. Συνεπώς για την δημιουργία ενός ROS 2 package πρέπει να γίνει μετάβαση στο φάκελο src του workspace και έπειτα να δημιουργηθεί το πακέτο προσομοίωσης του δίτροχου ρομπότ με το όνομα dd_robot.

```
$ cd ~/ros2_ws/src

$ #ros2 pkg create --build-type ament-python <package_name>

$ ros2 pkg create --build-type ament_python dd_robot
```

Πίνακας 39. Δημιουργία ενός πακέτου.

Αφού έχει δημιουργηθεί το ROS 2 Python package, ακολουθεί το χτίσιμο αυτού στην ρίζα του workspace. Κάθε νέο πακέτο που δημιουργείται πρέπει να χτίζεται με χρήση του εργαλείου colcon. Για το χτίσιμο ενός πακέτου πρέπει να εκτελεστούν οι ακόλουθες εντολές.

```
$ cd ~/ros2_ws

$ colcon build --packages-select dd_robot
```

Πίνακας 40. Χτίσιμο του πακέτου με χρήση του colcon.

Κάποιες φορές το χτίσιμο κάποιων πακέτων μπορεί να μην ολοκληρωθεί λόγω σφάλματος. Για να διορθωθεί αυτό πρέπει να εκτελεστούν οι ακόλουθες εντολές.

```
$ sudo apt install pip

$ pip install setuptools==58.2.0

$ colcon build --packages-select dd_robot
```

Πίνακας 41. Εγκατάσταση των setuptools.

Οι παραπάνω εντολές εγκαθιστούν μια παλαιότερη έκδοση μιας βιβλιοθήκης της Python, συγκεκριμένα της setuptools. Έπειτα γίνεται το χτίσιμο του πακέτου dd_robot. Σε περίπτωση που δεν προκύψει κάποιο σφάλμα κατά τη διαδικασία χτισίματος του πακέτου τότε οι εντολές του Πίνακα 41 δεν είναι απαραίτητες.

Έπειτα πρέπει να γίνει το στήσιμο του περιβάλλοντος στη ρίζα του workspace. Ωστόσο, στο πρώτο κεφάλαιο έχει προστεθεί αυτή η εντολή στο bashrc file των Linux, με αποτέλεσμα αυτή η διαδικασία να πραγματοποιείται κάθε φορά σε ένα νέο τερματικό.

Σε αυτό το σημείο το νέο πακέτο είναι έτοιμο για χρήση. Κάθε εκτελέσιμο που θα δημιουργηθεί από εδώ και πέρα μπορεί να χρησιμοποιηθεί.

## 2.2 RVIZ

Το Rviz είναι μια συντομογραφία του ROS Visualization. Είναι ένα πολύ ισχυρό εργαλείο το οποίο προσφέρει τρισδιάστατες απεικονίσεις. Δίνεται η δυνατότητα να προβληθούν ρομποτικά μοντέλα και τα δεδομένα που προέρχονται από τους αισθητήρες του ρομπότ στο τρισδιάστατο περιβάλλον. Με χρήση του Rviz μπορεί να εξεταστεί τι βλέπει, τι αισθάνεται, τι κάνει το ρομπότ και μετέπειτα να διορθωθούν τυχόν σφάλματα που προκύπτουν.

Στο Rviz μπορούν να απεικονιστούν δεδομένα από τρισδιάστατους αισθητήρες όπως στερεοσκοπικές κάμερες καθώς επίσης και δισδιάστατους αισθητήρες όπως απλές κάμερες και αποστασιόμετρα. Είτε το ρομπότ είναι πραγματικό είτε βρίσκεται σε μια προσομοίωση τότε το Rviz θα απεικονίζει την τωρινή κατάσταση που βρίσκεται αυτό. Για παράδειγμα αν το ρομπότ είναι ένα βραχίονας, ο οποίος μεταβαίνει από μια θέση σε μια άλλη του χώρου εργασίας του, τότε το Rviz θα προβάλει την κίνηση που ακολούθησε αυτός. Παράλληλα θα ανανεώνει τα δεδομένα που λαμβάνει από τους αισθητήρες τους.

Το Rviz συνήθως εγκαθίσταται μαζί με την πλήρη εγκατάσταση του ROS 2. Σε περίπτωση που αυτό δεν έχει εγκατασταθεί, τότε αυτό μπορεί να επιτευχθεί με χρήση της ακόλουθης εντολής.

```
$ sudo apt install ros-humble-rviz2
```

Πίνακας 42. Εγκατάσταση του Rviz 2.

Σε αυτό το σημείο είναι αναγκαίο να τονιστεί μια πολύ σημαντική διαφορά μεταξύ της πλήρης εγκατάστασης των Linux στο προσωπικό υπολογιστή και της εγκατάστασης στο WSL. Πιο συγκεκριμένα, κατά της διαδικασία εκκίνησης του Rviz στο WSL παρατηρείται ένα σφάλμα με αποτέλεσμα το κεντρικό παράθυρο του Rviz είναι μαυρισμένο (black screen). Για να διορθωθεί αυτό πρέπει να εκτελεστεί η ακόλουθη εντολή η οποία πρέπει να εκτελείται σε κάθε τερματικό που προτίθεται να εκκινήσει το Rviz.

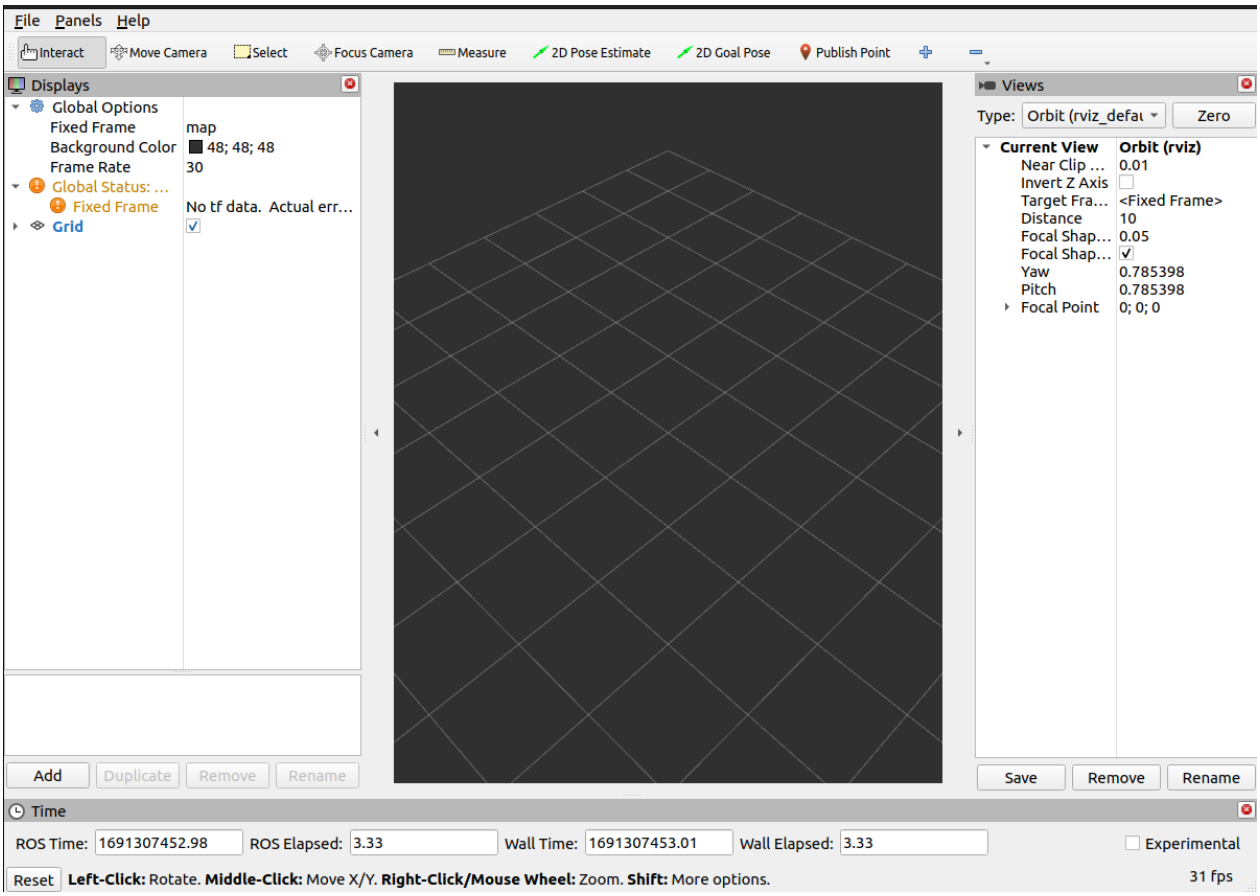
```
$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0
```

Πίνακας 43. Σημαντική εντολή για την εκτέλεση του Rviz.

Το Rviz εκκινεί με την παρακάτω εντολή. Ωστόσο τις περισσότερες φορές που θα χρησιμοποιείται, θα εκκινείτε μέσω της δημιουργίας και εκτέλεσης κάποιων launch αρχείων.

```
$ rviz2
```

Πίνακας 44. Εκκίνηση του Rviz.

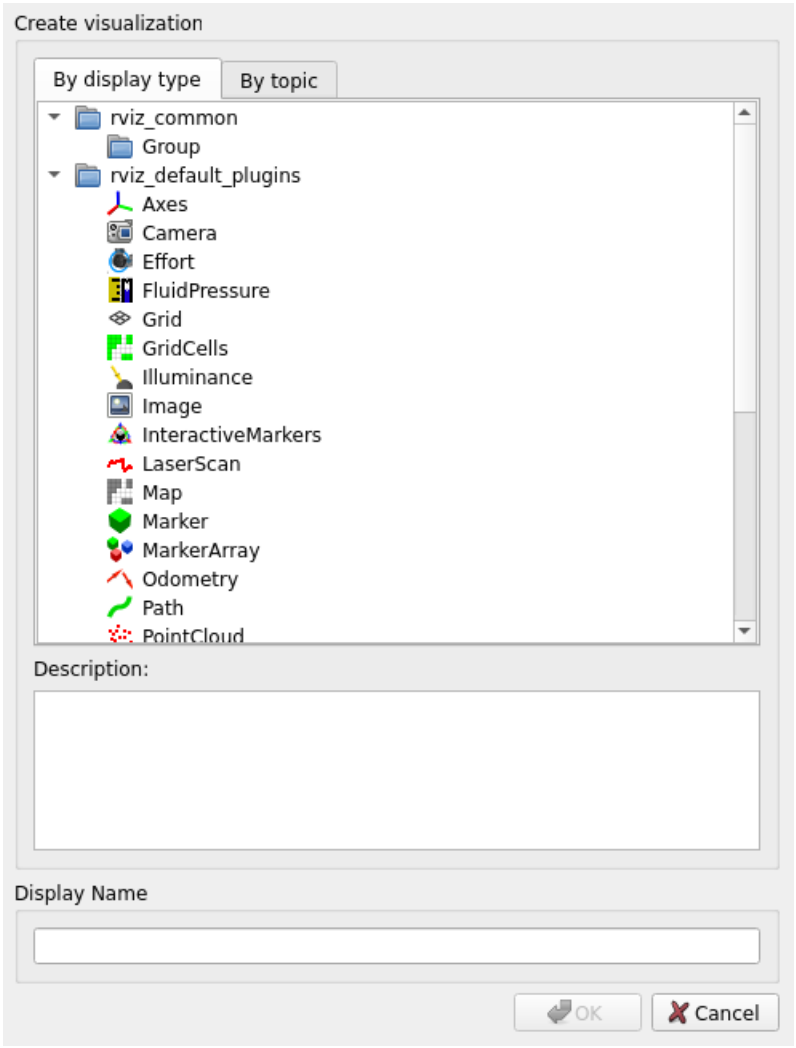



Εικόνα 13. Το γραφικό περιβάλλον του Rviz.

Κατά την εκκίνηση του Rviz παρατηρούνται τέσσερις βασικές περιοχές. Η βασικότερη από αυτές είναι το κεντρικό παράθυρο στο οποίο αποτυπώνονται τα τρισδιάστατα μοντέλα. Στο παράθυρο αυτό συνήθως υπάρχει μόνο ένα πλέγμα ή αυτό παραμένει κενό μέχρι να προστεθεί κάποιο μοντέλο. Στα αριστερά παρατηρείται το Displays panel, δεξιά βρίσκεται το Views panel και τέλος στη βάση του γραφικού περιβάλλοντος υπάρχει το Time panel. Πάνω από το κεντρικό παράθυρο υπάρχει ένα toolbar και ένα μενού για το γραφικό περιβάλλον του Rviz.

Το Display panel αποτελεί το δεύτερο σημαντικότερο παράθυρο του γραφικού περιβάλλοντος. Μέσω αυτού μπορούν να προστεθούν, να αντιγραφτούν ή να διαγραφούν τα στοιχεία που οπτικοποιούνται στο τρισδιάστατο χώρο καθώς και να πραγματοποιηθούν αλλαγές στις ρυθμίσεις του συστήματος. Επιλέγοντας το **Add** στο **Display panel**, εμφανίζεται ένα νέο μενού όπως παρατηρείται

στην παρακάτω εικόνα. Αυτό το μενού αποτυπώνει όλα τα διαθέσιμα στοιχεία που είναι πιθανόν να αποτυπωθούν στο τρισδιάστατο περιβάλλον.




Εικόνα 14. Το μενού του Add.

Μέσω αυτού του γραφικού περιβάλλοντος μπορούν να προστεθούν στοιχεία οπτικοποίησης όπως είναι οι κάμερες, τα λέιζερ, οι χάρτες, τα ρομποτικά μοντέλα και τα συστήματα συντεταγμένων που έχουν οριστεί στο ρομπότ καθώς και άλλα στοιχεία. Μια σύντομη περιγραφή του κάθε στοιχείου μπορεί να αποτυπωθεί επιλέγοντας κάποιο από αυτά και η περιγραφή εμφανίζεται στο παράθυρο Description. Για να γίνει η προσθήκη κάποιου στοιχείου, τότε γίνεται η επιλογή του και ακολούθως κάνοντας κλικ στο OK. Επίσης διατίθεται και η επιλογή να προστεθούν στοιχεία στο τρισδιάστατο χώρο του Rviz βάση των ενεργών topics που υπάρχουν διαθέσιμα στο δίκτυο του ROS τη δεδομένη χρονική στιγμή. Αυτό μπορεί να επιτευχθεί επιλέγοντας την καρτέλα **By topic** στο πάνω μέρος του παραθύρου και αντιστοίχως το επιθυμητό topic που είναι ενεργό.

Επιπρόσθετα το Display panel έχει πρόσβαση σε καθολικές ρυθμίσεις του συστήματος. Τέτοιες είναι το χρώμα που υπάρχει στο Background ή ο ρυθμός μετάδοσης δεδομένων (Frame Rate). Παράλληλα μπορούν να πραγματοποιηθούν αλλαγές στα χαρακτηριστικά που έχουν προστεθεί μέσω του Displays Panel όπως κάμερες ή λέιζερ.

Η περιήγηση στο τρισδιάστατο χώρο του Rviz επιτυγχάνεται με χρήση του ποντικιού του υπολογιστή. Πιο συγκεκριμένα διατίθενται οι παρακάτω επιλογές:

- **Αριστερό πλήκτρο ποντικιού**. Περιστροφή γύρω από το σημείο εστίασης.
- **Δεξιό πλήκτρο ποντικιού**. Πραγματοποιείται zoom in /out στο σημείο εστίασης.
- **Μεσαίο πλήκτρο**. Μεταφέρεται το σημείο εστίασης όπου είναι επιθυμητό.
- **Κύλιση της ροδέλας**. Γίνεται zoom in /out στο σημείο εστίασης.

Το toolbar που υπάρχει στη κορυφή του Rviz προσφέρει τις δυνατότητες που είναι οι παρακάτω:

- **Interact**. Αλληλεπίδραση με σημεία εστίασης όταν αυτά υπάρχουν.
- **Move camera**. Είναι η θέση της κάμερας η οποία αντιδρά στις μεταβολές από το ποντίκι.
- **Select**. Επιτρέπει ένα αντικείμενο να επιλεχθεί από το ποντίκι. Το επιλεγμένο αντικείμενο θα έχει ένα wireframe box γύρω του.
- **Focus Camera**. Με διπλό κλικ σε ένα σημείο του τρισδιάστατου χώρου το σημείο αυτό γίνεται το σημείο ενδιαφέροντος της κάμερας.
- **Measure.** Γίνεται η μέτρηση δύο σημείων του χώρου.
- **2D Nav Goal and 2D Pose Estimate.** Πραγματοποιείται δήλωση επιθυμητής τοποθέτησης και θέσης για το ρομπότ.

Επιπροσθέτως υπάρχει και το κεντρικό μενού του Rviz, στο οποίο υπάρχουν οι καρτέλες File, Panels και Help. Οι βασικές τους ιδιότητες είναι οι ακόλουθες:

- **File**: Επιλογές για άνοιγμα αρχείου, αποθήκευση αρχείου, αποθήκευση φωτογραφίας και έξοδος.
- **Panels**: Επιλογές για προσθήκη και αφαίρεση Panels.
- **Help**: Επιλογές για την εμφάνιση βοήθειας και άνοιγμα του browser του rviz.

## 2.3 Launch *αρχεία*

Για την δημιουργία μιας προσομοίωσης ενός ρομπότ ή τη δημιουργία ενός λογισμικού με χρήση του ROS 2, απαιτείται η δημιουργία πολλών φακέλων εντός του πακέτου. Ο λόγος για τον οποίο πραγματοποιείται αυτό είναι για λόγους οργάνωσης του λογισμικού που θα δημιουργηθεί. Ένας από τους φακέλους που θα χρειαστεί να δημιουργηθούν είναι ο φάκελος launch στον οποίο θα αποθηκεύονται όλα τα αρχεία αυτού του τύπου. Αυτά τα αρχεία είναι γραμμένα σε Python και μπορούν να ξεκινήσουν ή να σταματήσουν πολλά nodes. Τα nodes δέχονται αρκετά ορίσματα μέσω των launch αρχείων με αποτέλεσμα να αλλάζει η συμπεριφορά αυτών των εκτελέσιμων αρχείων.

Τα αρχεία launch στο ROS 2 είναι υπεύθυνα στο να βοηθούν το χρήστη να περιγράφει τις παραμέτρους του συστήματος του και έπειτα να τους εκτελεί με τον τρόπο που περιγράφηκαν. Στο launch αρχείο περιέχονται τα προγράμματα που θα εκτελεστούν, σε ποιο μέσο θα εκτελεστούν και τις παραμέτρους με τις οποίες θα εκκινήσουν. Επίσης, τα αρχεία αυτά είναι υπεύθυνα για την παρακολούθηση της εξέλιξης του συστήματος, καθώς επίσης και για την ενημέρωση αλλά και αντίδραση σε τυχόν αλλαγές του συστήματος.

Σε αυτό το σημείο αλλά και στη συνέχεια θα χρειαστεί ένας code editor για τις αλλαγές που πρόκειται να πραγματοποιηθούν στα υπάρχοντα αλλά και νέα αρχεία του πακέτου. Για αυτό το λόγο χρησιμοποιείται το Visual Studio Code που διατίθεται ελεύθερα από τη Microsoft. Για την εγκατάσταση του μπορεί να εκτελεστεί η ακόλουθη εντολή:

```
$ sudo snap install --classic code
```

Πίνακας 45. Εγκατάσταση του VS Code.

Αξιοσημείωτο όσον αφορά το Visual Studio Code είναι το γεγονός ότι διατίθεται ελεύθερα ένα extension του ROS το οποίο συστήνεται προς εγκατάσταση. Μπορεί να βρεθεί στην καρτέλα extension και έπειτα στην αναζήτηση πληκτρολογώντας ROS. Η εγκατάσταση του συγκεκριμένου extension προσφέρει βιβλιοθήκες, εργαλεία και δυνατότητες οι οποίες είναι ιδιαίτερα χρήσιμες για την ανάπτυξη λογισμικού στο VS Code.

Το Visual Studio Code μπορεί να εκκινήσει μέσω τερματικού εντός του πακέτου που είναι επιθυμητό να πραγματοποιηθούν διεργασίες. Το πλεονέκτημα είναι η μεγάλη ευκολία πρόσβασης σε όλα τα αρχεία του πακέτο. Αυτό επιτυγχάνεται με τα παρακάτω:

```
$ cd ~/ros2_ws/src/dd_robot

$ code .
```

Πίνακας 46. Εκκίνηση του VS Code μέσω τερματικού.

Αφού εκκινήσει το VS Code, θα δημιουργηθεί ο φάκελος launch εντός του πακέτου.

```
$ cd ~/ros2_ws/src/dd_robot

$ mkdir launch
```

Πίνακας 47. Δημιουργία φακέλου launch.

Για να μπορέσει να γίνει η χρήση των αρχείων που θα δημιουργηθούν εντός του φακέλου launch πρέπει να πραγματοποιηθεί μια συγκεκριμένη διαδικασία. Είναι αναγκαίο να γίνει η επεξεργασία του αρχείου setup.py που βρίσκεται εντός του πακέτου που έχει δημιουργηθεί. Πιο συγκεκριμένα, θα γίνει η προσθήκη δύο νέων βιβλιοθηκών και ενός μονοπατιού στη λίστα data_files.

Πιο συγκεκριμένα, θα προστεθούν οι ακόλουθες βιβλιοθήκες στην αρχή του αρχείου setup.py

```
$ import os

$ from glob import glob
```

Πίνακας 48. Προσθήκη βιβλιοθηκών στο setup.py.

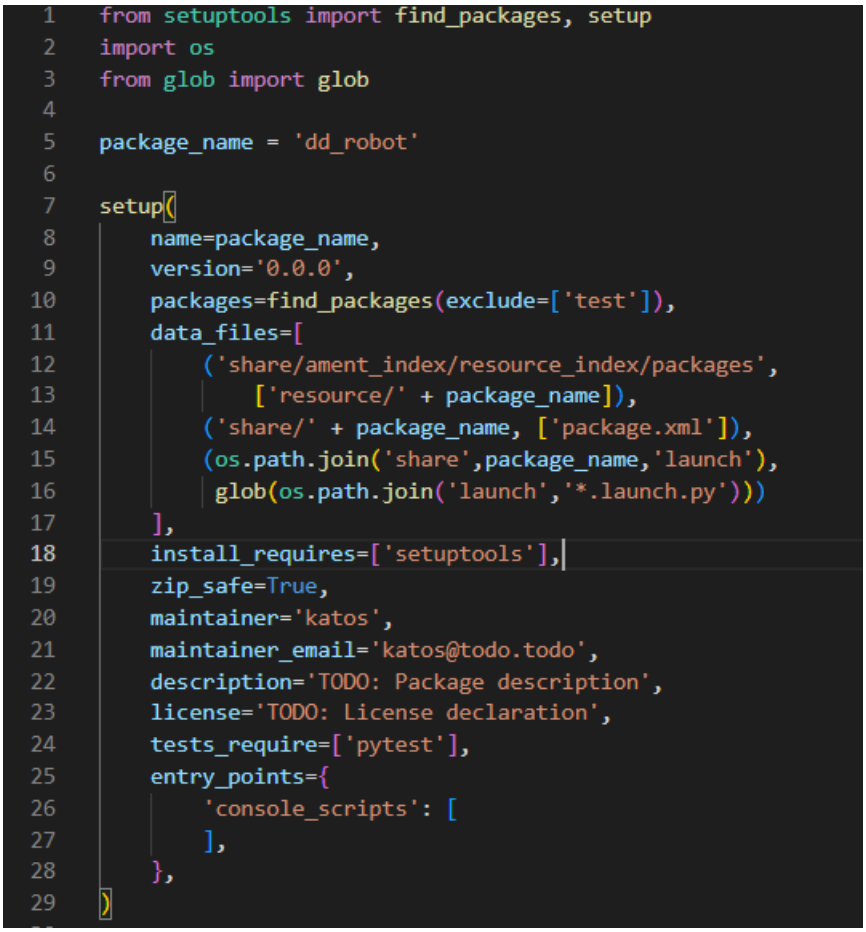
Στη συνέχεια πρέπει να προστεθούν τα ακόλουθα στη λίστα data_files:

```
$ (os.path.join('share',package_name,'launch'),

        glob(os.path.join('launch','*.launch.py'))),
```

Πίνακας 49. Προσθήκη εντολών στη λίστα data_files.

Η παραπάνω διαδικασία είναι ιδιαίτερα σημαντική καθώς είναι απαραίτητη ώστε ο κώδικας που θα δημιουργείται να μπορεί εγκατασταθεί με χρήση του εργαλείου colcon. Σε περίπτωση μη επεξεργασίας του setup.py αρχείου τότε ο κώδικα που θα αναπτυχθεί δεν θα μπορέσει να εκτελεστεί.

Έπειτα από την προσθήκη των παραπάνω, το αρχείο setup.py θα πρέπει να είναι το ακόλουθο.



```python
from setuptools import find_packages, setup
import os
from glob import glob

package_name = 'dd_robot'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share',package_name,'launch'),
         glob(os.path.join('launch','*.launch.py')))
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='katos',
    maintainer_email='katos@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
        ],
    },
)
```

Εικόνα 15. Το νέο setup.py αρχείο.

Η προσθήκη των παραπάνω είναι απαραίτητη διότι το setup.py αρχείο είναι υπεύθυνο για την εγκατάσταση όλων των launch αρχείων με κατάληξη launch.py που θα δημιουργηθούν. Αφού ολοκληρωθούν οι αλλαγές αυτές τότε το πακέτο πρέπει να χτιστεί.

```
$ cd ~/ros2_ws

$ colcon build
```

Πίνακας 50. Χτίσιμο του πακέτου.

Πληροφοριακά η διαδικασία αυτή δεν χρειαζόταν στη πρώτη έκδοση του ROS. Επιπροσθέτως, αντίστοιχες αλλαγές θα είναι απαραίτητο να πραγματοποιηθούν αν το πακέτο δημιουργούνταν βάση του CMake. Οπότε σε αυτό το σημείο τα καινούργια αρχεία που θα δημιουργούνται μέσα στο φάκελο launch μπορούν να εκτελεστούν.

## 2.4 Δημιουργία δίτροχου ρομπότ με URDF

Το URDF είναι ένα XML format το οποίο είναι καθορισμένο να αναπαριστά διάφορα ρομποτικά μοντέλα τα οποία μπορούν να δημιουργηθούν από την αρχή. Τα URDF αρχεία πολλές φορές μπορούν να γίνουν μακροσκελή και δύσχρηστα σε πολυσύνθετα ρομποτικά συστήματα. Για αυτό και υπάρχει η XML Macros (Xacro) η οποία είναι μια XML macro language που δημιουργήθηκε για να διευκολύνει τη δημιουργία ρομποτικών συστημάτων.

Στην ενότητα θα αναπτυχθούν αρκετά ρομποτικά μοντέλα με URDF τα οποία θα αναπαριστούν ένα δίτροχο ρομπότ. Τα αρχεία αυτά θα δημιουργούνται σταδιακά μέχρι να ολοκληρωθούν όλες οι διαδικασίες σχεδίασης του ρομπότ. Τα ρομπότ θα τοποθετούνται στο Rviz ώστε να επιβεβαιωθεί η ορθή δημιουργία τους. Επιπρόσθετα το ρομποτικό μοντέλο θα τοποθετηθεί στη προσομοίωση του Gazebo.

Αρχικά θα πρέπει εντός του πακέτου να δημιουργηθεί ένας νέος φάκελος στον οποίο θα αποθηκεύονται όλα τα αρχεία τύπου URDF που θα δημιουργηθούν.

```
$ cd ~/ros2_ws/src/dd_robot

$ mkdir urdf
```

Πίνακας 51. Δημιουργία φακέλου urdf.

Όπως και νωρίτερα όταν δημιουργήθηκε ο launch φάκελος εντός του πακέτου, ακολούθησε η επεξεργασία του setup.py αρχείου που βρίσκεται στο εντός του πακέτου. Κατά παρόμοιο τρόπο θα γίνει ξανά η επεξεργασία του συγκεκριμένου αρχείου. Θα πρέπει στη λίστα data_files να προστεθούν τα ακόλουθα.

```
$ (os.path.join('share',package_name,'urdf'),

        glob(os.path.join('urdf','*.urdf'))),
```

Πίνακας 52. Προσθήκη του urdf στο setup.py.

Σε αυτό το σημείο το αρχείο setup.py μπορεί να αποθηκευτεί. Θα ακολουθήσει το χτίσιμο του πακέτου στη ρίζα του workspace (Πίνακας 52).

Η XML (eXternal Markup Language) είναι μια γλώσσα σήμανσης, που περιέχει ένα σύνολο από κανόνες για την ηλεκτρονική κωδικοποίηση κειμένων. Σχεδιάστηκε δίνοντας έμφαση στην απλότητα, τη γενικότητα και τη χρησιμότητα στο Διαδίκτυο. Αν και η XML εστιάζει στα κείμενα, χρησιμοποιείται ευρέως για την αναπαράσταση αυθαίρετων δομών δεδομένων, που προκύπτουν για παράδειγμα στις υπηρεσίες ιστού.

Κάποια από τα χαρακτηριστικά της γλώσσας XML είναι τα ακόλουθα:

- **Tag – Ετικέτα**. Είναι ένα στοιχείο σήμανσης που ξεκινάει με το χαρακτήρα '<' και καταλήγει στο χαρακτήρα '>'. Υπάρχουν τρία είδη ετικέτας.
    - **Start-tag**, για παράδειγμα <section>.
    - **End-tag**, για παράδειγμα </section>.
    - **Empty element-tag**, για παράδειγμα <line-break/>.
- **Element- Στοιχείο**. Είναι ένα λογικό απόσπασμα ενός κειμένου, που είτε ξεκινάει με μια ετικέτα start-tag και καταλήγει σε μια end-tag, είτε αποτελείται μόνο από μια empty element-tag. Οι χαρακτήρες που υπάρχουν μεταξύ της ετικέτας αρχής και τέλους, αποτελούν το περιεχόμενο του στοιχείου. Αυτό το περιεχόμενο είναι πιθανό να περιέχει και κάποια άλλα στοιχεία που ονομάζονται στοιχεία-παιδιά.
- **Attributes- Χαρακτηριστικά**. Είναι ένα στοιχείο σήμανσης που αποτελείται από ένα ζευγάρι όνομα/τιμή, το οποίο υπάρχει μέσα σε μια ετικέτα αρχής ή σε μια ετικέτα χωρίς περιεχόμενο.

Οι λεπτομέρειες της XML θα είναι εμφανές μέσα από τα πλήρη αρχεία που θα παρουσιαστούν παρακάτω.

### 2.4.1 Δημιουργία του chassis

Το ρομποτικό μοντέλο που θα δημιουργηθεί θα είναι το σασί του δίτροχου ρομπότ και θα αποθηκευτεί με το όνομα ddrobot.urdf εντός του πακέτου dd_robot και συγκεκριμένα στο φάκελο urdf.

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

<robot name="dd_robot">
  <link name="base_link">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <box size="0.5 0.5 0.25"/>
      </geometry>
    </visual>
  </link>

</robot>
```

Ο παραπάνω XML κώδικας καθορίζει ένα ρομπότ με το όνομα dd_robot το οποίο διαθέτει ένα link, του οποίου το οπτικό μέρος είναι ένα κουτί με μήκος και πλάτος 0.5 μέτρα καθώς και ύψος 0.25 μέτρα. Επίσης το ρομπότ είναι τοποθετημένο στην αρχή των αξόνων του τρισδιάστατου χώρου (origin tag) και δεν έχει εφαρμοστεί κάποια περιστροφή σε αυτό (rpy).

Αν χρησιμοποιείται το VS Code μπορεί να προβληθεί το μοντέλο που έχει δημιουργηθεί μέχρι στιγμής. Πιο συγκεκριμένα, πληκτρολογώντας Ctrl+Shift+P και έπειτα γίνει η αναζήτηση του Preview URDF και η επιλογή του, τότε θα εμφανιστεί το ρομποτικό μοντέλο σε ένα νέο παράθυρο. Το βήμα αυτό δεν είναι απαραίτητο, αλλά είναι βοηθητικό καθώς οι αλλαγές στο μοντέλο εμφανίζονται ακαριαία αποθηκεύοντας το μοντέλο.

### 2.4.2 Το ρομποτικό μοντέλο στο Rviz

Για να τοποθετηθεί το ρομποτικό μοντέλο που δημιουργήθηκε στο Rviz πρέπει να δημιουργηθεί ένα launch αρχείο. Τα launch αρχεία είναι πολύ σημαντικά καθώς δίνουν την δυνατότητα να εκκινήσουν πολλά nodes ταυτόχρονα. Στο ROS 2 τα αρχεία αυτά είναι ανεπτυγμένα, τις περισσότερες φορές, σε γλώσσα Python εν αντίθεση με το ROS 1 που χρησιμοποιούνταν αποκλειστικά η γλώσσα XML. Ωστόσο και στην ανανεωμένη έκδοση του ROS προσφέρεται η XML για τη ανάπτυξη launch αρχείων. Το πλεονέκτημα της Python έγκειται στο γεγονός ότι προσφέρει περισσότερες επιλογές στο χρήστη.

Πριν γίνει η ανάπτυξη του launch αρχείου, καθίσταται η ανάγκη εγκατάστασης κάποιων πακέτων του ROS.

```
$ sudo apt install ros-humble-joint-state-publisher ros-humble-joint-state-publisher-gui

$ sudo apt install ros-humble-urdf-tutorial
```

Πίνακας 53. Εγκατάσταση νέων πακέτων.

Το παρακάτω launch αρχείο θα αποθηκευτεί εντός του φακέλου launch που βρίσκεται στο πακέτο. Το όνομα με το οποίο θα αποθηκευτεί είναι dd_robot_rviz.launch.py. Τα launch αρχεία θα έχουν πάντα την κατάληξη launch.py. Επίσης συνηθίζεται δίπλα από το όνομα του αρχείου να προστίθεται και η κατάληξη rviz. Με αυτό το τρόπο είναι εύκολα αντιληπτό ποια nodes θα εκκινήσουν αλλά και τον σκοπό του συγκεκριμένου αρχείου. Το Python αρχείο είναι το ακόλουθο:

```
from ament_index_python.packages import get_package_share_path
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.conditions import IfCondition, UnlessCondition
from launch.substitutions import Command, LaunchConfiguration
```

```
from launch_ros.actions import Node
from launch_ros.parameter_descriptions import ParameterValue

def generate_launch_description():

    dd_robot_path = get_package_share_path('dd_robot')
    default_model_path = dd_robot_path / 'urdf/ddrobot.urdf'

    gui_arg = DeclareLaunchArgument(name='gui', default_value='true', choices=['true', 'false'],
                                    description='Flag to enable joint_state_publisher_gui')
    model_arg = DeclareLaunchArgument(name='model', default_value=str(default_model_path),
                                      description='Absolute path to robot urdf file')

    robot_description = ParameterValue(Command(['xacro ', LaunchConfiguration('model')]),
                                       value_type=str)

    robot_state_publisher_node = Node(
        package='robot_state_publisher',executable='robot_state_publisher',
        parameters=[{'robot_description': robot_description}])

    joint_state_publisher_node = Node(package='joint_state_publisher', executable='joint_state_publisher',
        condition=UnlessCondition(LaunchConfiguration('gui')))

joint_state_publisher_gui_node = Node(package='joint_state_publisher_gui',executable=
            'joint_state_publisher_gui', condition=IfCondition(LaunchConfiguration('gui')) )

    rviz_node = Node( package='rviz2',executable='rviz2',
        name='rviz2', output='screen',)

    return LaunchDescription([
        gui_arg, model_arg, joint_state_publisher_node,
        joint_state_publisher_gui_node, robot_state_publisher_node,, rviz_node ])
```

Συνοπτικά το παραπάνω launch αρχείο επιτυγχάνει τα ακόλουθα :

- Φορτώνει το μοντέλο και το αποθηκεύει σαν παράμετρο.
- Δημοσιεύει δεδομένα που αφορούν το ρομπότ στο topic, /robot description.
- Εκκινεί nodes για τη δημοσίευση πληροφοριών στο joint state publisher node προσφέροντας τη δυνατότητα χρήσης ορισμάτων που είναι το gui και το ρομποτικό μοντέλο.

- Εκκινεί το Rviz με ένα αρχείο παραμέτρων.

Αναλυτικότερα, στον παραπάνω κώδικα γίνεται εισαγωγή κάποιων απαραίτητων βιβλιοθηκών. Έπειτα δημιουργείται μια συνάρτηση, της οποίας το όνομα θα πρέπει να παραμένει ως έχει σε κάθε launch αρχείο που θα δημιουργείται. Στην αρχή της συνάρτησης ορίζονται κάποια μονοπάτια, ώστε να μπορεί να βρεθεί εύκολα το ρομποτικό μοντέλο που θα εισαχθεί στο Rviz. Έπειτα δημιουργούνται κάποια nodes τα οποία στη συνέχεια θα εκτελεστούν. Στο τέλος της συνάρτησης, καλείται η ιδιότητα return στην οποία και δηλώνονται τα nodes και ορίσματα που θα εκκινήσουν.

Για να πραγματοποιηθεί η εκτέλεση του launch αρχείου πρέπει πρώτα να χτιστεί το πακέτο. Οπότε για την εκκίνηση του launch file εκτελούνται τα παρακάτω.

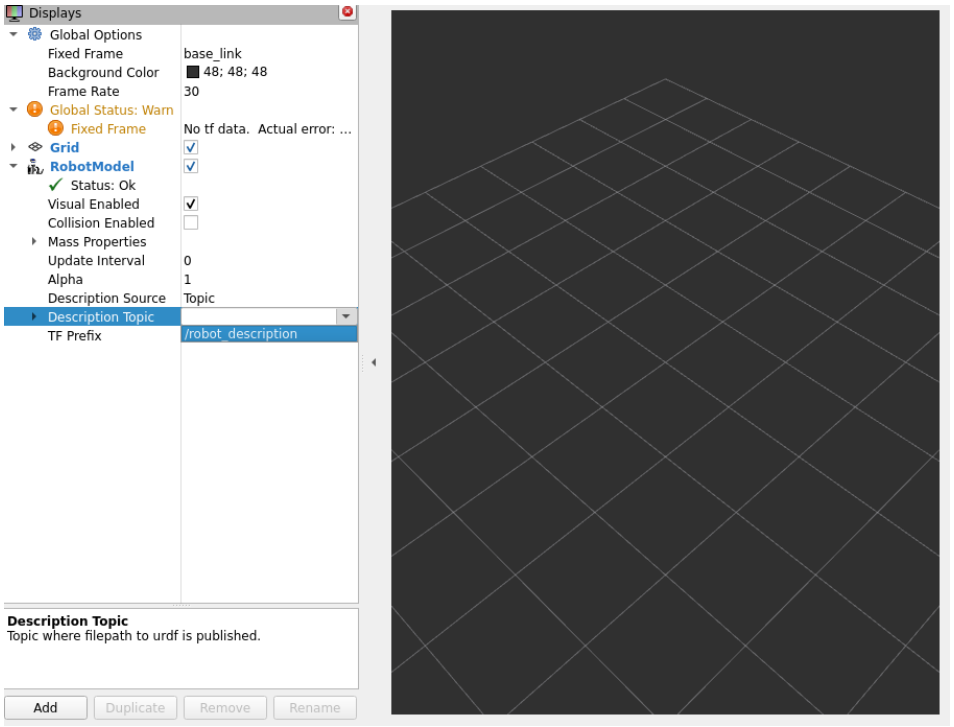
```
$ cd ~/ros2_ws

$ colcon build

$ cd src/dd_robot  # if you are using WSL, run the next command.

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch dd_robot dd_robot_rviz.launch.py model:=urdf/ddrobot.urdf gui:='false'
```

Πίνακας 54. Εκτέλεση του launch αρχείου.

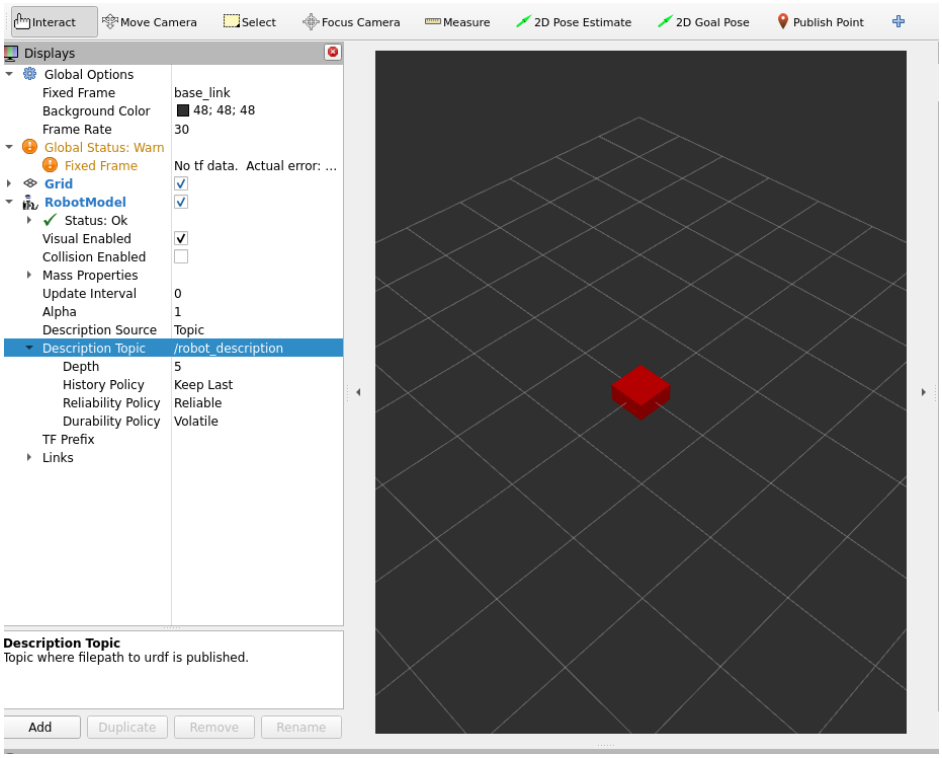
Από τις παραπάνω εντολές παρατηρείται ότι η εντολή εκτέλεσης του launch αρχείου μπορεί να δεχτεί δύο ορίσματα. Αυτό είναι αρκετά βοηθητικό καθώς το συγκεκριμένο launch αρχείο θα χρησιμοποιείται συνέχεια και μέσω εντολών στο τερματικό διαφοροποιούνται τα ορίσματα με τα οποία θα εκκινεί το Rviz.

Με την εκτέλεση τη τελευταίας εντολής του παραπάνω πίνακα, θα εκκινήσει το γραφικό περιβάλλον του Rviz. Για να προστεθεί το ρομποτικό μοντέλο, επιλέγεται το **Add** που βρίσκεται στο Display panel. Έπειτα επιλέγεται το **RobotModel** και μετά το **Ok**. Στη συνέχεια στο **Display Panel** εντοπίζεται το **Fixed Frame** και κάνοντας διπλό κλικ πληκτρολογείτε **base_link**.

Στη συνέχεια κάνοντας διπλό κλικ στο **RobotModel**, αναδιπλώνεται ένα μενού ρυθμίσεων για το ρομποτικό μοντέλο. Ακολούθως, επιλέγοντας την καρτέλα **Description Topic** και επιλέγοντας το **/robot_description** γίνεται η προσθήκη του ρομποτικού μοντέλου στο τρισδιάστατο χώρο του Rviz.




Εικόνα 16. Επιλογή του topic.




Εικόνα 17. Το μοντέλο στο Rviz.

### 2.4.3 Προσθήκη τροχών

Αφού δημιουργηθούν και προστεθούν στο μοντέλο δύο νέα link, τα οποία θα αναπαριστούν τις ρόδες του ρομπότ, θα πρέπει να δημιουργηθούν κάποια joints που να περιγράφουν τη σχέση μεταξύ των link. Τα στοιχεία του μοντέλου πρέπει να ενωθούν για να δημιουργήσουν το τελικό μοντέλο. Οι συνδέσεις μεταξύ δύο link είτε θα είναι fixed είτε movable. Συνολικά στο URDF, υπάρχουν οι παρακάτω επιλογές σύνδεσης μεταξύ των links:

- **Fixed**: Με χρήση αυτής της σύνδεσης επιτυγχάνεται η ένωση δύο links, χωρίς να υπάρχει κάποια επιτρεπτή κίνηση μεταξύ αυτών.
- **Revolute**: Αυτός τύπος άρθρωσης επιτρέπει την περιστροφή γύρω από ένα άξονα και υπάρχουν άνω και κάτω όρια.
- **Continuous**: Επιτρέπεται η περιστροφική κίνηση γύρω από έναν άξονα, χωρίς άνω και κάτω όρια.
- **Prismatic**: Η άρθρωση αυτή επιτρέπει την κίνηση κατά μήκος ενός άξονα και υπάρχουν άνω και κάτω όρια.
- **Floating**: Επιτρέπει την κίνηση και στους έξι βαθμούς ελευθερίας.
- **Planar**: Είναι μια άρθρωση στην οποία επιτρέπονται όλες οι κινήσεις πάνω σε ένα επίπεδο.

Για την δημιουργία του ρομπότ διαφορικής κίνησης θα χρησιμοποιηθούν μόνο τα joints τύπου continuous. Το ανανεωμένο μοντέλο βασίζεται στο προηγούμενο προσθέτοντας δύο ακόμη γεωμετρικά σχήματα και τις αρθρώσεις μεταξύ του σασί και των δύο ροδών. Συνεπώς δημιουργείται ένα νέο ρομποτικό μοντέλο το οποίο θα αποθηκευτεί ως ddrobot2.urdf στο φάκελο urdf του πακέτου και ο κώδικας σε URDF είναι ο ακολούθως:

```
<?xml version='1.0'?>
<robot name="dd_robot">

 <!-- Base Link -->
 <link name="base_link">
  …

 <!-- Right Wheel -->
 <link name="right_wheel">
  <visual>
   <origin xyz="0 0 0" rpy="1.570795 0 0" />
```

```
    <geometry>
      <cylinder length="0.1" radius="0.2" />
    </geometry>
   </visual>
  </link>

 <!-- Right Wheel Joint -->
  <joint name="joint_right_wheel" type="continuous">
   <parent link="base_link"/>
   <child link="right_wheel"/>
   <origin xyz="0 -0.30 0" rpy="0 0 0" />
   <axis xyz="0 1 0" />
  </joint>

  <!-- Left Wheel -->
 <link name="left_wheel">
   <visual>
    <origin xyz="0 0 0" rpy="1.570795 0 0" />
    <geometry>
      <cylinder length="0.1" radius="0.2" />
    </geometry>
   </visual>
  </link>

 <!-- Left Wheel Joint -->
  <joint name="joint_left_wheel" type="continuous">
   <parent link="base_link"/>
   <child link="left_wheel"/>
   <origin xyz="0 0.30 0" rpy="0 0 0" />
   <axis xyz="0 1 0" />
  </joint>
 </robot>
```

Στο παραπάνω URDF παρατηρούνται τα ακόλουθα:

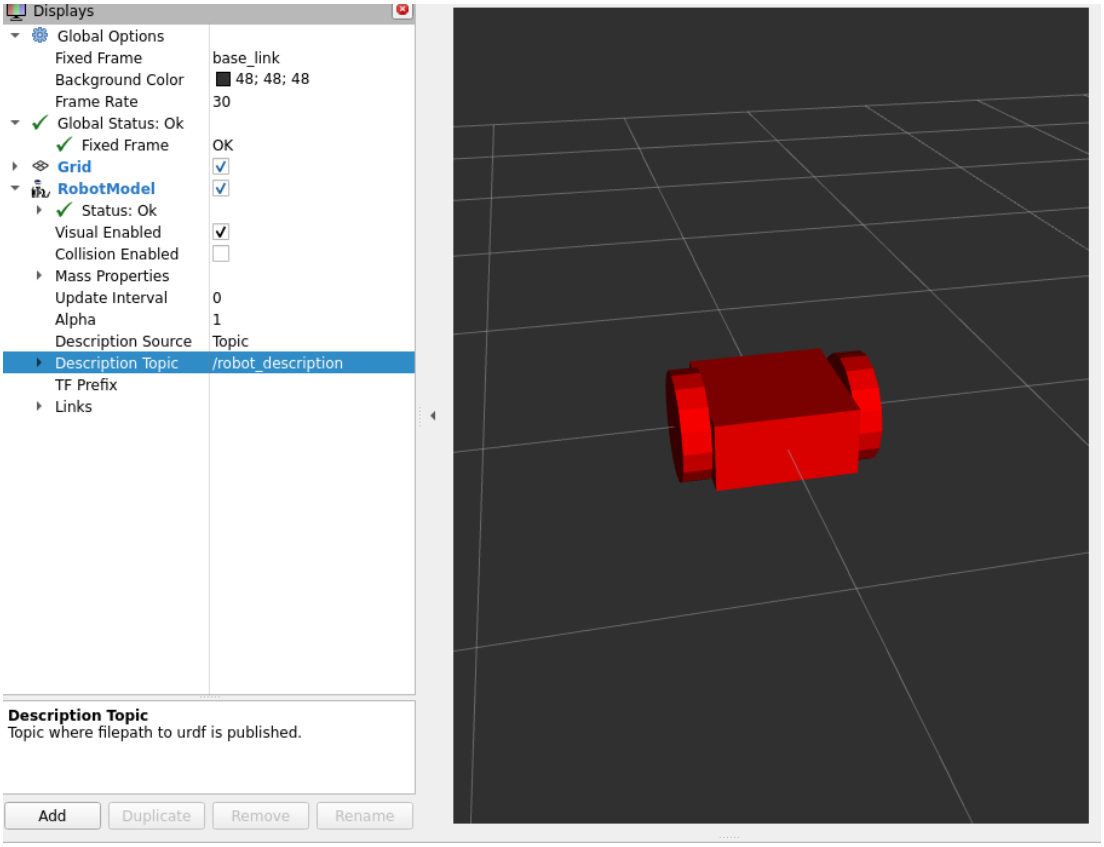
- Η κάθε ρόδα έχει διάμετρο 0.2 μέτρα και πλάτος 0.1 μέτρο. Τοποθετείτε στην αρχή των αξόνων (0,0,0) και έχει μια περιστροφή 90 μοιρών ως προς τον άξονα των x.
- Στην άρθρωση μεταξύ του σασί και της ρόδας παρατηρείται ένα parent link και ένα child link. Η ένωση μεταξύ των δύο link γίνεται κατά 0.3 μέτρα ως προς τον άξονα χ για την αριστερή ρόδα και -0.3 μέτρα για την δεξιά ρόδα.

Για να επιβεβαιωθεί η ορθή δημιουργία του ρομποτικού μοντέλου μπορεί να γίνει ο έλεγχος στο Preview του VS Code αλλά ιδανικότερα γίνεται η εισαγωγή του στο Rviz.

```
$ cd ~/ros2_ws

$ colcon build

$ cd src/dd_robot      # For WSL run the next command.

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch dd_robot dd_robot_rviz.launch.py model:=urdf/ddrobot2.urdf gui:='false'
```

Πίνακας 55. Εκκίνηση του launch αρχείου για το ddrobot2.urdf.

Με την τελευταία εντολή εκτελείται το launch αρχείο που δημιουργήθηκε στην προηγούμενη παράγραφο, αλλά αυτή τη φορά δέχεται σαν παράμετρο το ρομπότ που δημιουργήθηκε παραπάνω. Οπότε αφού γίνουν οι κατάλληλες αλλαγές στο Fixed Frame και προστεθεί το RobotModel προκύπτει το παρακάτω:




Εικόνα 18. Το ddrobot2.urdf στο Rviz.

### 2.4.4 Προσθήκη ενός caster

Το επόμενο βήμα στη δημιουργία του δίτροχου ρομπότ αποτελεί η προσθήκη ενός caster το οποίο θα προσφέρει την ισορροπία που χρειάζεται το όχημα. Στο ρομποτικό όχημα που έχει δημιουργηθεί μέχρι στιγμής, θα προστεθεί μια σφαίρα στη βάση αυτού και θα έχει fixed joint με το σασί.

Δημιουργείται ένα νέο αρχείο το οποίο ονομάζεται ddrobot3.urdf και αποθηκεύεται στο φάκελο urdf του πακέτου. Το ddrobot3.urdf είναι το ακόλουθο:

```
<?xml version='1.0'?>
<robot name="dd_robot">

 <!-- Base Link -->
 <link name="base_link">
  …
 </link>

 <!-- Caster -->
 <link name="caster">
  <visual>
   <origin xyz="0.0 0 0" rpy="0 0 0" />
   <geometry>
    <sphere radius="0.05" />
   </geometry>
  </visual>
 </link>

 <!-- Caster joint -->
 <joint name="joint_caster" type="fixed">
  <parent link="base_link"/>
  <child link="caster"/>
  <origin xyz="0.2 0 -0.125" rpy="0 0 0" />
 </joint>

 <!-- Right Wheel -->
 <link name="right_wheel">
  ..
 </link>
 <joint name="joint_right_wheel" type="continuous">
  ….
 </joint>
```

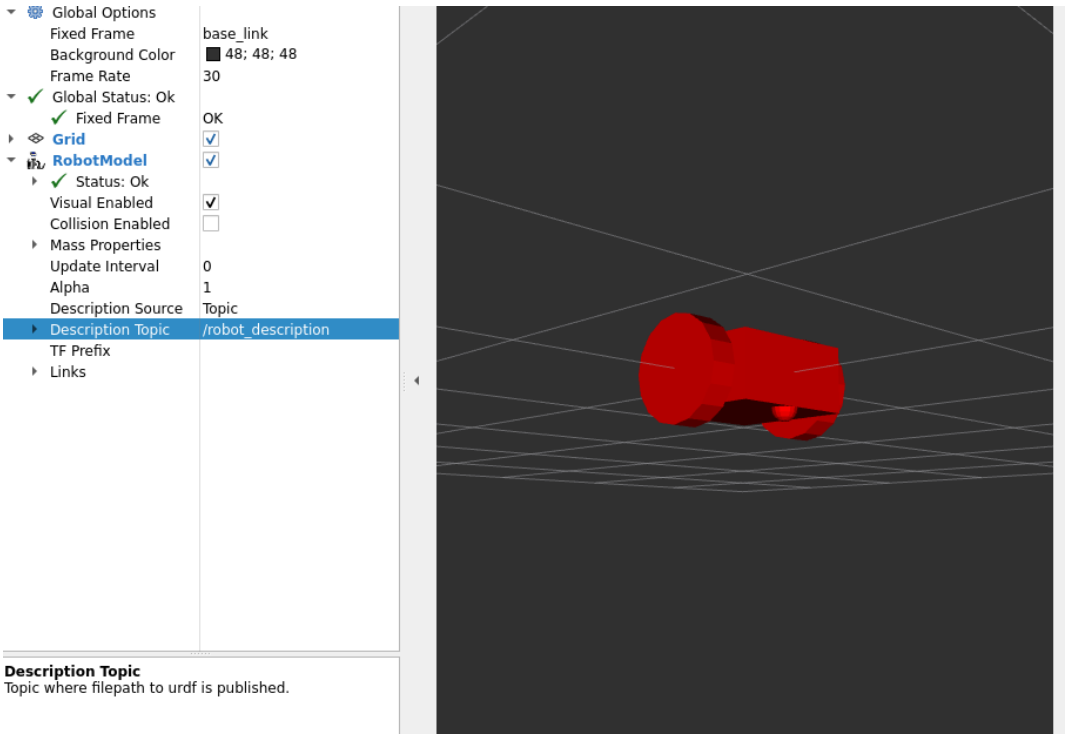
```
  <!-- Left Wheel -->
  <link name="left_wheel">
        …
  </link>
  <joint name="joint_left_wheel" type="continuous">
    …
  </joint>
</robot>
```

Στο νέο urdf έχει προστεθεί το caster που είναι μια σφαίρα με διάμετρο 0.05 μέτρα. Το joint μεταξύ του σασί και του caster είναι fixed και σε κατάλληλη θέση ώστε να ισορροπεί το δίτροχο ρομπότ.

Η επιβεβαίωση ότι το caster έχει τοποθετηθεί στο ρομποτικό όχημα γίνεται μέσω του Rviz. Εκκινεί το launch file ωστόσο θα πρέπει πρώτα προηγηθεί το χτίσιμο του πακέτου. Επίσης, αν γίνεται η χρήση του ROS μέσω του WSL πρέπει να εκτελέσει μια επιπρόσθετη εντολή για να τοποθετηθεί το ρομποτικό μοντέλο στο Rviz. Οι εντολές παρουσιάζονται στους πίνακες 54 και 55.

| $ ros2 launch dd_robot dd_robot_rviz.launch.py model:=urdf/ddrobot3.urdf gui:='false' |
|---|

Πίνακας 56. Εκκίνηση του launch αρχείου για το ddrobot3.urdf.




Εικόνα 19. Το ddrobot3.urdf στο Rviz.

### 2.4.5 Προσθήκη χρωμάτων

Το χρώμα κάθε link είναι βοηθητικό καθώς μέσω αυτού μπορούν να διαχωρίζονται το ένα από το άλλο. Το χρώμα κάθε link μπορεί να ρυθμιστεί κατάλληλα καθορίζοντας τις διάφορες τιμές rgba εντός του <material> tag που θα προστεθεί σε κάθε link.

Δημιουργείται ένα νέο αρχείο το ddrobot4.urdf το οποίο είναι παρόμοιο με τα προηγούμενα μοντέλα με διαφορά ότι υπάρχει η προσθήκη χρωμάτων σε κάθε link, όπως φαίνεται παρακάτω:

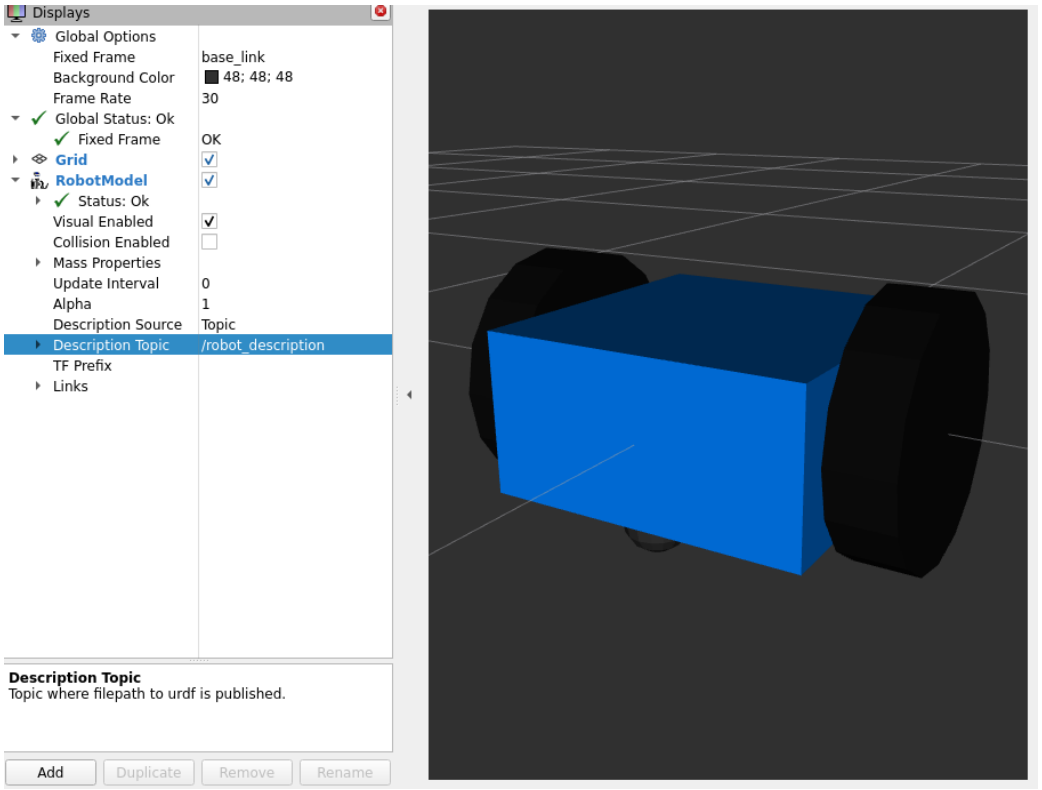
```
<?xml version='1.0'?>
<robot name="dd_robot">
  <!-- Base Link -->
  <link name="base_link">
    <visual>
          …
      <material name="blue">
        <color rgba="0 0.5 1 1"/>
      </material>
    </visual>
  </link>
  <!-- Caster -->
  <link name="caster">
          …
      <material name="darkgray">
        <color rgba=".2 .2 .2 1"/>
      </material>
    </visual>
  </link>
  <!-- Caster joint -->
  <joint name="joint_caster" type="fixed">
    …
  </joint>
  <!-- Right Wheel -->
  <link name="right_wheel">
    <visual>
      …
      <material name="black">
        <color rgba="0.05 0.05 0.05 1"/>
      </material>
    </visual>
  </link>
  <joint name="joint_right_wheel" type="continuous">
```

```
  …
 </joint>
 <!-- Left Wheel -->
 <link name="left_wheel">
  <visual>
   …
   <material name="black"/>
  </visual>
 </link>
 <joint name="joint_left_wheel" type="continuous">
  …
 </joint>
</robot>
```

Στη συνέχεια τοποθετείται το μοντέλο στο Rviz αφού προηγηθεί το χτίσιμο του πακέτου. Οι διαδικασίες αυτές παρουσιάζονται στους πίνακες 54 και 55.

```
$ ros2 launch dd_robot dd_robot_rviz.launch.py model:=urdf/ddrobot4.urdf gui:='false'
```

Πίνακας 57. Εκκίνηση του launch αρχείου για το ddrobot4.urdf.




Εικόνα 20. Το ddrobot4.urdf στο Rviz.

### 2.4.6 Προσθήκη Collisions

Στην ενότητα αυτή θα προστεθεί ένα collision tag σε κάθε link του αρχείου URDF. Το collision tag καθορίζει τα όρια κάθε γεωμετρικού σχήματος που έχει δημιουργηθεί. Τα συγκεκριμένα tags είναι απαραίτητα παρά το γεγονός ότι έχουν καθοριστεί νωρίτερα οι διαστάσεις κάθε σχήματος. Αυτό το βήμα είναι απαραίτητο όταν θα τοποθετηθεί το ρομπότ σε μια προσομοίωση.

Σε πολλές προσομοιώσεις δεν θα υπάρχουν μόνο απλά γεωμετρικά σχήματα, αλλά κάποια κομμάτια ενός ρομπότ θα έχουν σύνθετη γεωμετρία. Σε εκείνα τα μοντέλα όταν θα προστίθενται collisions τα οποία θα είναι όσο πιο απλά γίνεται. Ο λόγος για αυτό είναι ότι καταναλώνονται πολλοί υπολογιστικοί πόροι κατά την προσομοίωση του ρομπότ, κάνοντας δύσκολες τις διεργασίες που θα έχουν νόημα.

Δημιουργείται ένα νέο αρχείο με όνομα ddrobot5.urdf, το οποίο είναι το ακόλουθο:

```
<?xml version='1.0'?>
<robot name="dd_robot">

 <!-- Base Link -->
 <link name="base_link">
  ...
  <!-- Base collision, mass and inertia -->
  <collision>
   <origin xyz="0 0 0" rpy="0 0 0" />
   <geometry>
      <box size="0.5 0.5 0.25"/>
   </geometry>
  </collision>
 </link>

 <!-- Caster -->
 <link name="caster">
  …
  <!-- Caster collision, mass and inertia -->
  <collision>
   <origin xyz="0 0 0" rpy="0 0 0" />
   <geometry>
    <sphere radius="0.05" />
   </geometry>
  </collision>
 </link>
 <!-- Right Wheel -->
```

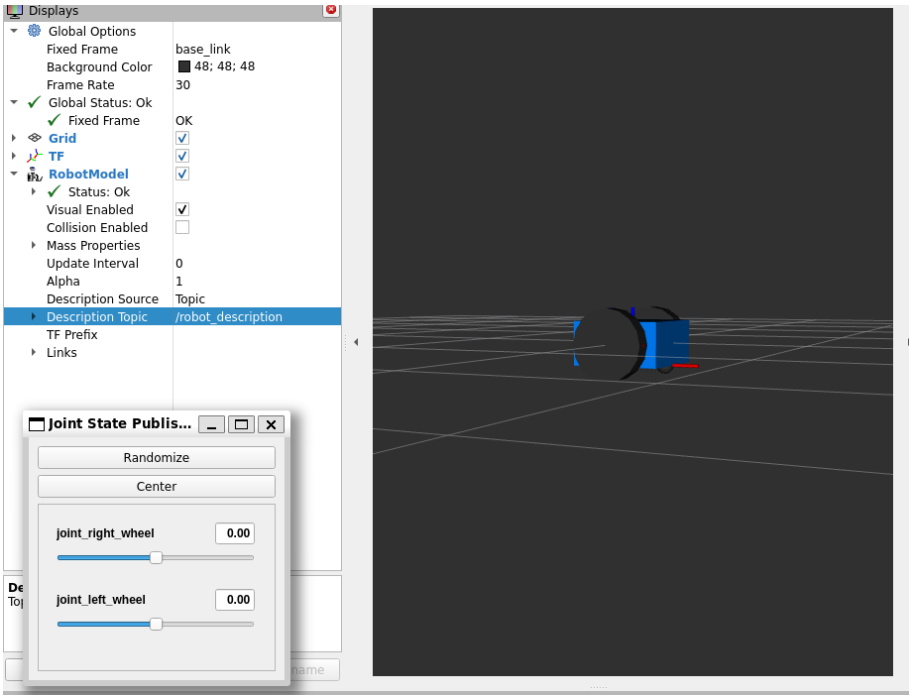
```
 <link name="right_wheel">
  …
  <!-- Right Wheel collision, mass and inertia -->
  <collision>
   <origin xyz="0 0 0" rpy="1.570795 0 0" />
   <geometry>
      <cylinder length="0.1" radius="0.2" />
   </geometry>
  </collision>
 </link>

 <!-- Left Wheel -->
 <link name="left_wheel">
  …
  <!-- Left Wheel collision, mass and inertia -->
  <collision>
   <origin xyz="0 0 0" rpy="1.570795 0 0" />
   <geometry>
      <cylinder length="0.1" radius="0.2" />
   </geometry>
  </collision>
 </link>
<!-- Right Wheel joint -->
…
<!-- Left Wheel joint -->
…
</robot>
```

Στον παραπάνω κώδικα παρατηρείται ότι έχουν προστεθεί τα collisions για κάθε γεωμετρικό σχήμα που υπάρχει μέχρι στιγμής. Τα collision tags είναι ακριβώς ίδια με τα visual tags των σχημάτων. Όσον αφορά το μοντέλο που θα τοποθετηθεί στο Rviz, δεν θα υπάρχει κάποια διαφορά με το προηγούμενο μοντέλο διότι τα collisions θα χρειαστούν μόνο κατά την προσομοίωση του ρομπότ.

Για τη τοποθέτηση του ρομπότ στο Rviz πρέπει να γίνουν τα βήματα του Πίνακα 54, 55 και έπειτα να εκτελεστεί η ακόλουθη εντολή.

```
$ ros2 launch dd_robot dd_robot_rviz.launch.py model:=urdf/ddrobot5.urdf
```

Πίνακας 58. Εκκίνηση του launch αρχείου για το ddrobot5.urdf.




Εικόνα 21. Το ddrobot5.urdf στο Rviz.

Κατά την εκκίνηση του Rviz παρατηρείται και ένα νέο παράθυρο, το joint state publisher. Αυτό το παράθυρο επιτρέπει την κίνηση των αρθρώσεων εντός των ορίων που έχουν οριστεί. Μέσω του παραθύρου αυτού εξετάζεται αν η κίνηση των αρθρώσεων είναι η επιθυμητή. Το Randomize button επιλέγει μια τυχαία θέση για κάθε άρθρωση, ενώ το Center button επιστρέφει τις αρθρώσεις στην αρχική θέση.

Στο σημείο αυτό μπορούν να εξεταστούν τα ενεργά nodes και topics που βρίσκονται στο ROS δίκτυο.

```
katos@KATOS:~$ ros2 node list
/joint_state_publisher
/robot_state_publisher
/rviz2
/transform_listener_impl_55e6e515eaf0
katos@KATOS:~$ ros2 topic list
/clicked_point
/goal_pose
/initialpose
/joint_states
/parameter_events
/robot_description
/rosout
/tf
/tf_static
```

Εικόνα 22. Ενεργά nodes και topics.

Από τα παραπάνω, ιδιαίτερο ενδιαφέρον έχει το tf και το robot_state_publisher. Το πακέτο tf είναι ένα από τα βασικότερα πακέτα του ROS και σχετίζεται με τα συστήματα συντεταγμένων που φέρει ένα ρομπότ. Κάθε ρομποτικό σύστημα είτε αυτό είναι σε προσομοίωση είτε πραγματικό φέρει πάνω του πολλά συστήματα συντεταγμένων τα οποία βοηθούν στην λειτουργία του συστήματος. Το πακέτο tf βοηθάει στην τοποθέτηση των συστημάτων συντεταγμένων στα components που είναι απαραίτητο. Στο dd_robot τα συστήματα συντεταγμένων είναι τρία, ένα στο κέντρο κάθε ρόδας και ένα στο κέντρο του σασί.

Το robot_state_publisher node λαμβάνει πληροφορίες από το /joint_states topic, δηλαδή τη θέση που βρίσκονται τα joints του ρομπότ και έπειτα δημοσιεύει πληροφορίες στο /tf topic. Με αυτό το τρόπο επιτυγχάνεται η μετατόπιση ή η στροφή του κάθε συστήματος συντεταγμένων του ρομπότ σε σύγκριση με την αρχική τοποθέτηση του. Τα συστήματα συντεταγμένων μπορούν να οπτικοποιηθούν μέσω του Rviz, με χρήση της καρτέλας Display panel και έπειτα της επιλογής του **TF**.

### 2.4.7 Προσθήκη φυσικών ιδιοτήτων

Για να μπορεί να γίνει η προσομοίωση του ρομποτικού μοντέλου, απαραίτητη είναι η προσθήκη φυσικών ιδιοτήτων στο μοντέλο URDF. Οι φυσικές ιδιότητες που θα προστεθούν είναι η μάζα και η ροπή αδράνειας για κάθε link.

Τα δύο αυτά υποστοιχεία που θα χρησιμοποιηθούν είναι τα εξής:

- <**mass**>: Το βάρος με μονάδα μέτρησης τα κιλά.
- <**inertia**>: Είναι ένας πίνακας 3x3 ο οποίος είναι συμμετρικός. Συνεπώς αρκεί να υπολογιστούν τα έξι από τα εννιά στοιχεία του πίνακα. Για κάθε γεωμετρικό σχήμα του URDF οι εξισώσεις υπολογισμού διαφέρουν. Η Wikipedia προσφέρει μια λίστα από πολλά γεωμετρικά σχήματα και τους τύπους υπολογισμού τους (https://en.wikipedia.org/wiki/ List_of_moments_of_inertia.).

Το νέο URDF αρχείο αποθηκεύεται με το όνομα ddrobot6.urdf και οι αλλαγές που πραγματοποιούνται στο προηγούμενο μοντέλο είναι οι ακόλουθες:

```
<?xml version='1.0'?>
<robot name="dd_robot">

  <!-- Base Link -->
 <link name="base_link">
        …
   <inertial>
    <mass value="5"/>
```

```
    <inertia ixx="0.13" ixy="0.0" ixz="0.0" iyy="0.21" iyz="0.0" izz="0.13"/>
  </inertial>
</link>

<!-- Caster -->
<link name="caster">
  <inertial>
   <mass value="0.5"/>
   <inertia ixx="0.0001" ixy="0.0" ixz="0.0" iyy="0.0001" iyz="0.0" izz="0.0001"/>
  </inertial>
        …
</link>
<!-- Right Wheel -->
<link name="right_wheel">
        …
  <inertial>
   <mass value="0.5"/>
   <inertia ixx="0.01" ixy="0.0" ixz="0.0" iyy="0.005" iyz="0.0" izz="0.005"/>
  </inertial>
</link>

<!-- Left Wheel -->
<link name="left_wheel">
        …
  <inertial>
   <mass value="0.5"/>
   <inertia ixx="0.01" ixy="0.0" ixz="0.0" iyy="0.005" iyz="0.0" izz="0.005"/>
  </inertial>
</link>
</robot>
```

Οι αλλαγές που γίνονται στο παραπάνω αρχείο δεν επηρεάζουν το μοντέλο στο Rviz, απλώς είναι αναγκαίες για την προσομοίωση αυτού. Ωστόσο το ρομπότ μπορεί να τοποθετηθεί στο Rviz, αφού γίνουν οι απαραίτητες προεργασίες που έχουν αναφερθεί παραπάνω και παρουσιάζονται στον Πίνακα 54, 55.

### 2.5 Το πρόγραμμα προσομοίωσης Gazebo

Το Gazebo είναι ελεύθερο και ανοιχτού κώδικα περιβάλλον προσομοίωσης ρομπότ το οποίο αναπτύχθηκε από τη Willow Garage. Το Gazebo είναι ένα πολυχρηστικό εργαλείο το οποίο χρησιμοποιείται για πάρα πολλούς σκοπούς. Συγκεκριμένα μπορούν να πραγματοποιηθούν δοκιμές πραγματικού χρόνου με χρήση διάφορων ρομπότ τα οποία άλλοτε βρίσκονται σε εσωτερικό ή εξωτερικό περιβάλλον. Επίσης είναι πολύ

χρήσιμο καθώς μπορούν να εξεταστούν διάφοροι αλγόριθμοι καθώς επίσης και διάφοροι αισθητήρες όπως κάμερες και αποστασιόμετρα.

Η εγκατάσταση του gazebo πραγματοποιείται με την παρακάτω εντολή:

```
$ sudo apt install ros-humble-gazebo-ros
```

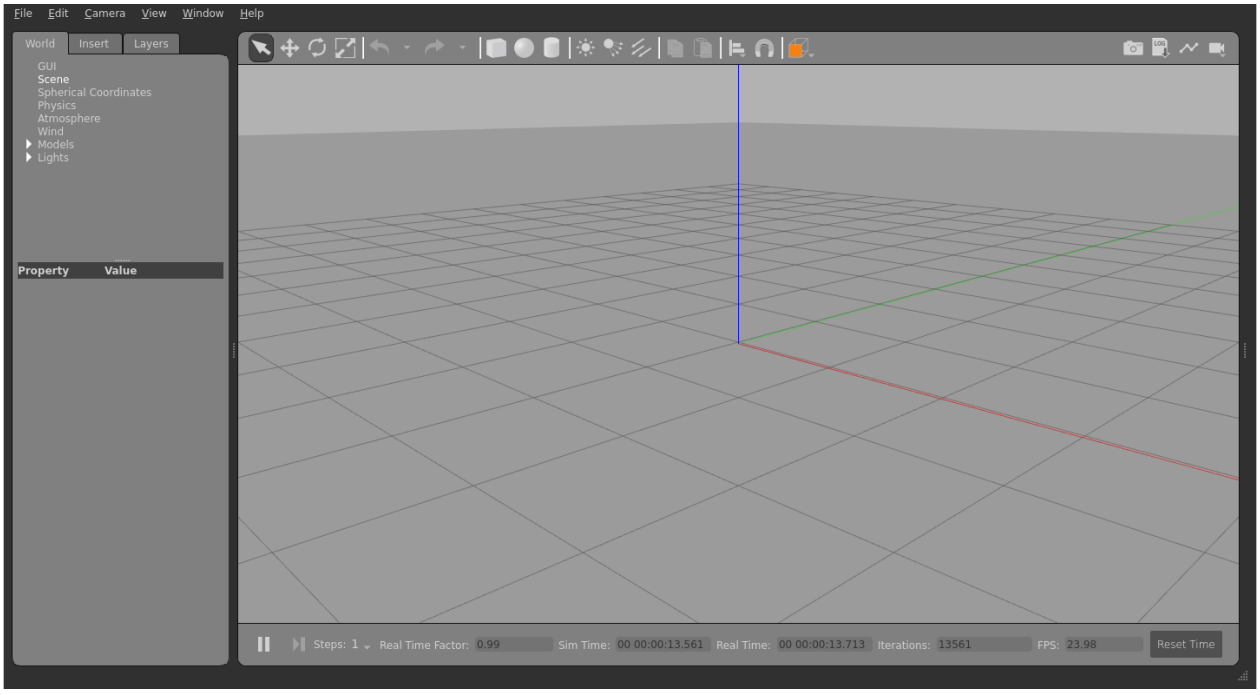
Πίνακας 59. Εγκατάσταση του Gazebo.

Το Gazebo εκκινεί με την παρακάτω εντολή ωστόσο αν εκτελείται μέσω του WSL, πριν γίνει η εκκίνηση του, πρέπει να εκτελεστεί μια εντολή η οποία είναι η αντίστοιχη με αυτή που χρησιμοποιήθηκε και για το Rviz, η εκτέλεση της οποίας είναι απαραίτητη καθώς παρατηρείται αντίστοιχο πρόβλημα.

```
$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0 # for WSL run this.

$ gazebo
```

Πίνακας 60. Εκκίνηση του Gazebo.

Οι παραπάνω εντολές θα πρέπει να εκκινήσουν το παρακάτω γραφικό περιβάλλον το οποίο είναι το γραφικό περιβάλλον του Gazebo.




Εικόνα 23. Το γραφικό περιβάλλον του Gazebo.

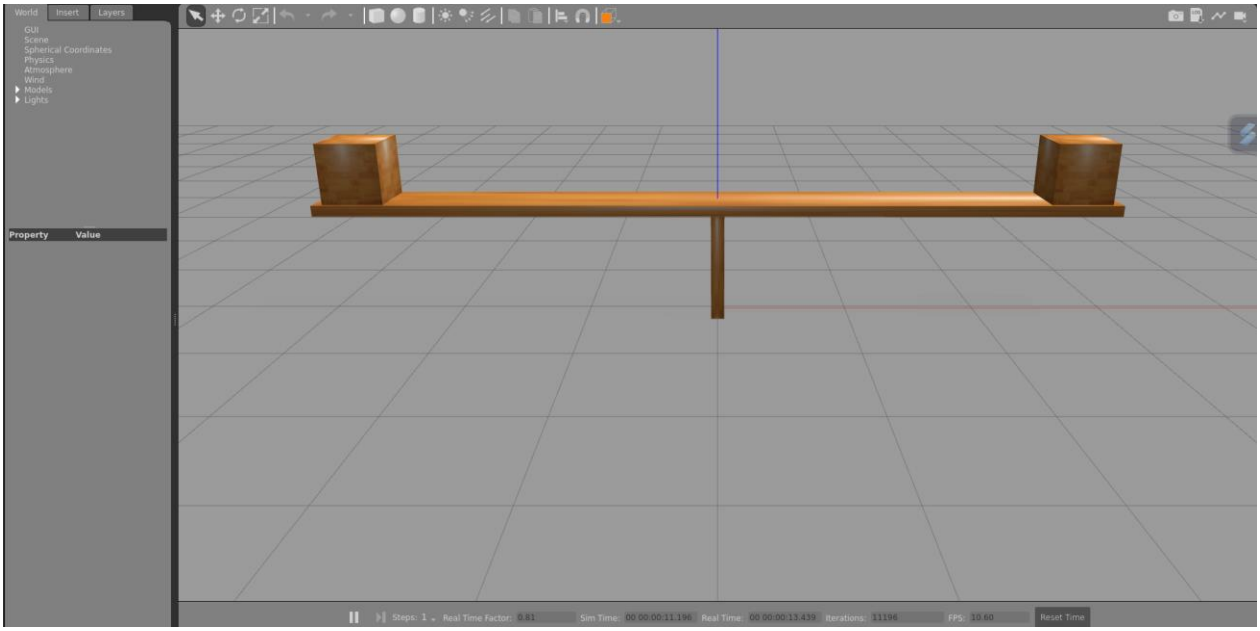
Η παραπάνω εντολή εκκινεί δύο εκτελέσιμα. Το πρώτο είναι Gazebo server και το δεύτερο το Gazebo client. Το Gazebo server, gzserver, θα εκτελέσει όλες τις διαδικασίες προσομοίωσης στις οποίες εμπεριέχεται μια μηχανή φυσικής. Επίσης παρέχεται ενημέρωση για όλα τα δεδομένα που παράγονται. Το gzserver αποτελεί το πυρήνα του Gazebo και μπορεί να εκτελείται ανεξάρτητα από το γραφικό περιβάλλον του Gazebo. Για παράδειγμα το gzserver μπορεί να εκκινήσει σε ένα υπολογιστή με πολλούς υπολογιστικούς πόρους και το γραφικό περιβάλλον του Gazebo σε έναν άλλον υπολογιστή.

Με τη σειρά του το Gazebo Client, gzclient, είναι υπεύθυνο για την οπτικοποίηση της προσομοίωσης. Με λίγα λόγια είναι υπεύθυνο για το γραφικό περιβάλλον του Gazebo. Το Gazebo έχει μια δυσκολία στο να τερματίσει, για αυτό το λόγο συστήνεται η έξοδος μέσω τερματικού πληκτρολογώντας Ctrl +C.

Όπως παρατηρείται στην παραπάνω εικόνα, έχει φορτωθεί ένας άδειος κόσμος στο Gazebo. Ωστόσο υπάρχουν και άλλοι κόσμοι οι οποίοι εγκαθίστανται μαζί με το Gazebo. Ενδεικτικά παρουσιάζονται κάποιοι έτοιμοι κόσμοι του Gazebo.

| |
|---|
| $ gazebo /usr/share/gazebo-11/worlds/shapes_layers.world<br><br>$ gazebo /usr/share/gazebo-11/worlds/seesaw.world |

Πίνακας 61. Εγκατεστημένοι κόσμοι του Gazebo.




Εικόνα 24. Ο κόσμος seesaw.world.

Οι παραπάνω εντολές είναι αρκετά πιθανό να καθυστερήσουν την εκκίνηση του Gazebo την πρώτη φορά που θα εκτελεστούν. Ο λόγος που γίνεται αυτό είναι διότι οι κόσμοι αυτοί δεν έχουν ληφθεί στον προσωπικό υπολογιστή, με αποτέλεσμα η διαδικασία λήψης να γίνεται τη στιγμή που εκτελείται η εντολή για το άνοιγμα του κόσμου. Για τον τερματισμό του Gazebo εκτελείται Ctrl+C στο τερματικό.

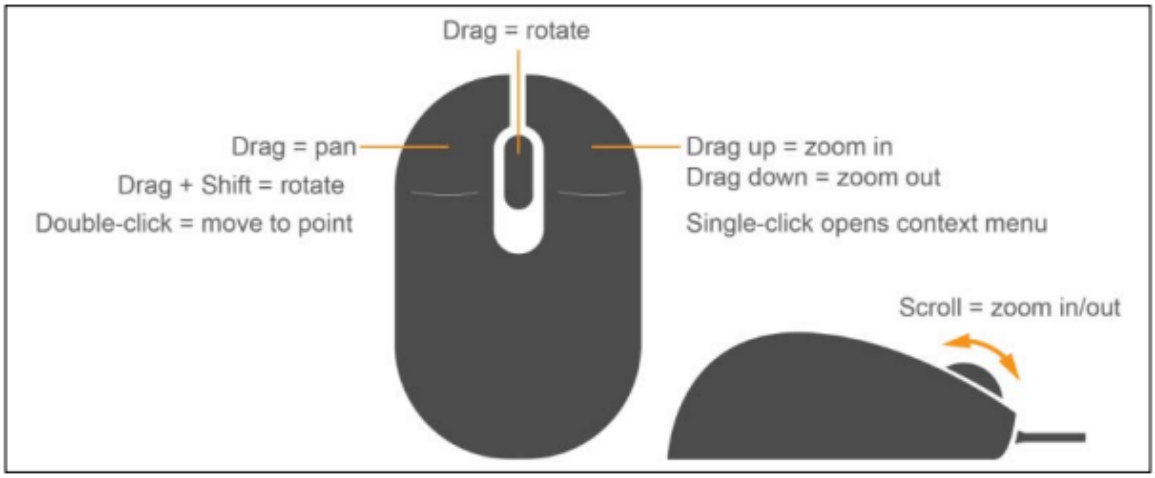
Εκτός από τους κόσμους που αναφέρονται στον προηγούμενο πίνακα υπάρχουν και άλλοι διαθέσιμοι κόσμοι. Για να μπορέσει να γίνει η μελέτη των διαθέσιμων κόσμων του Gazebo μπορούν να εκτελεστούν τα παρακάτω.

```
$ # Take the next command and paste it to a terminal. Then double press the tab button.

$ gazebo /usr/share/gazebo-11/worlds/
```

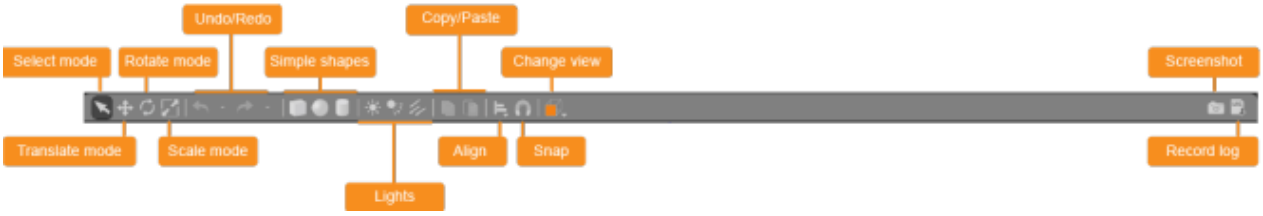
Πίνακας 62. Εύρεση όλων των διαθέσιμων κόσμων.

Το γραφικό περιβάλλον του Gazebo είναι αρκετά παρόμοιο με αυτό του Rviz. Στο κεντρικό παράθυρο προβάλλεται ο τρισδιάστατος κόσμος του Gazebo. Το πλέγμα που εμφανίζεται στο κεντρικό παράθυρο είναι το ground plane, δηλαδή το επίπεδο στο οποίο όλα τα μοντέλα έλκονται από τη βαρύτητα. Επίσης στο παράθυρο αυτό παρατηρούνται τρεις άξονες οι οποίοι αναπαριστούν το καρτεσιανό σύστημα συντεταγμένων.

Στο συγκεκριμένο παράθυρο μπορεί να γίνει πλοήγηση με χρήση του ποντικιού του προσωπικού υπολογιστή και οι δυνατότητες που προσφέρονται παρουσιάζονται στην ακόλουθη εικόνα.




Εικόνα 25. Πλοήγηση μέσω ποντικιού στο Gazebo.




Εικόνα 26. Το Environmental toolbar.

Εκτός του κεντρικού παραθύρου, παρατηρούνται και άλλα τρία σημαντικά panels. Στα αριστερά παρατηρείται ένα panel με καρτέλες **World, Insert** και **Layers**. Επιπροσθέτως υπάρχει το simulation panel στο κάτω μέρος του κεντρικού παραθύρου και στο πάνω μέρος υπάρχει το **Environment toolbar**. Τέλος, υπάρχει και το βασικό μενού του παραθύρου του Gazebo.

Στο **Environment toolbar** προσφέρονται όλες οι δυνατότητες της Εικόνας 26. Αρχικά δίνεται η δυνατότητα να επιλεγεί ένα αντικείμενο και μετέπειτα να πραγματοποιηθεί η αντιγραφή ή η διαγραφή του στο χώρο (Copy/Paste). Επιπρόσθετα ένα αντικείμενο μπορεί να μεταφερθεί σε οποιοδήποτε σημείο του τρισδιάστατου χώρου (Translate mode) αλλά και να περιστραφεί ως προς όποιο άξονα είναι επιθυμητό (Rotate mode). Μπορεί να γίνει η μεγέθυνση ενός μοντέλου και να γίνει Undo ή Redo με τις κατάλληλες επιλογές.

Επίσης μπορούν να προστεθούν διάφορα είδη φωτισμού (Lights), καθώς επίσης απλά γεωμετρικά σχήματα (Simple Shapes) ή να δημιουργηθούν joints μεταξύ αυτών. Τέλος, δίνεται η δυνατότητα του Screenshot και της καταγραφής δεδομένων μέσω βίντεο.

Το panel που βρίσκεται στα αριστερά είναι το πιο σημαντικό panel του Gazebo. Αναλυτικότερα διαθέτει την καρτέλα **World** και **Insert**. Στην πρώτη εμφανίζονται όλα τα στοιχεία που είναι ενεργά στο Gazebo εκείνη τη στιγμή. Πιο συγκεκριμένα μπορεί να πραγματοποιηθεί η επεξεργασία διάφορων παραμέτρων των στοιχείων που είναι ενεργά. Συνεπώς μπορούν να υπάρξουν μεταβολές στην βαρύτητα του περιβάλλοντος, το χρώμα του φωτισμού καθώς και άλλες ρυθμίσεις του συστήματος. Επιπρόσθετα στην καρτέλα αυτή εμφανίζονται τα μοντέλα που έχουν προστεθεί στο Gazebo, όπως το ground plane, τα ρομποτικά μοντέλα ή αντικείμενα.

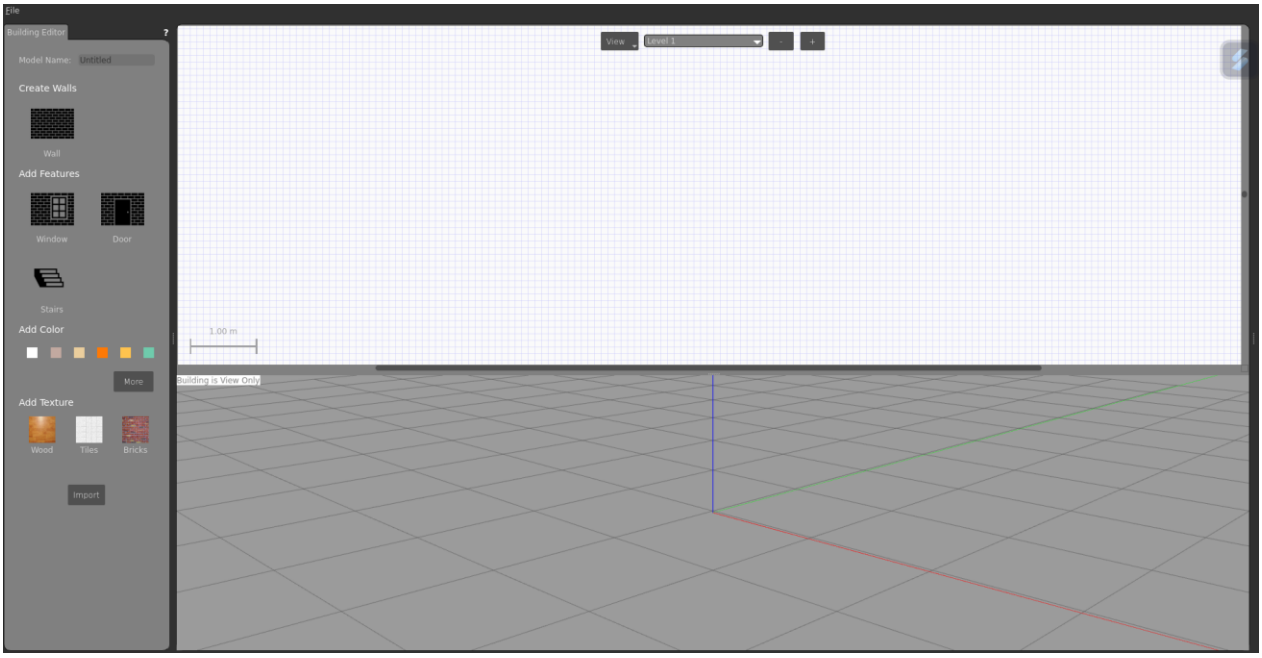
Στην καρτέλα **Insert**, μπορούν να προστεθούν μοντέλα που είναι εγκατεστημένα στον υπολογιστή ή μπορούν να εγκατασταθούν εκείνη τη στιγμή από το διαδικτυακό αποθετήριο του Gazebo. Σε περίπτωση που δεν εμφανίζονται μοντέλα στα αποθετήρια τότε γίνεται διπλό κλικ στους δύο συνδέσμους που υπάρχουν στην καρτέλα.

Στο κάτω μέρος του γραφικού περιβάλλοντος υπάρχει το **Simulation panel**. Αυτό είναι ένα εύχρηστο εργαλείο το οποίο εκτελεί σενάρια προσομοίωσης. Μέσω αυτού μπορεί να σταματήσει ο χρόνος τη προσομοίωσης ή να παρατηρηθεί ο χρόνος με τον οποίο εκτελείται η προσομοίωση και να συγκριθεί με τον πραγματικό χρόνο. Είναι αρκετά χρήσιμο όταν γίνεται καταγραφή μέσω βίντεο και όταν επαναλαμβάνονται προσομοιώσεις.

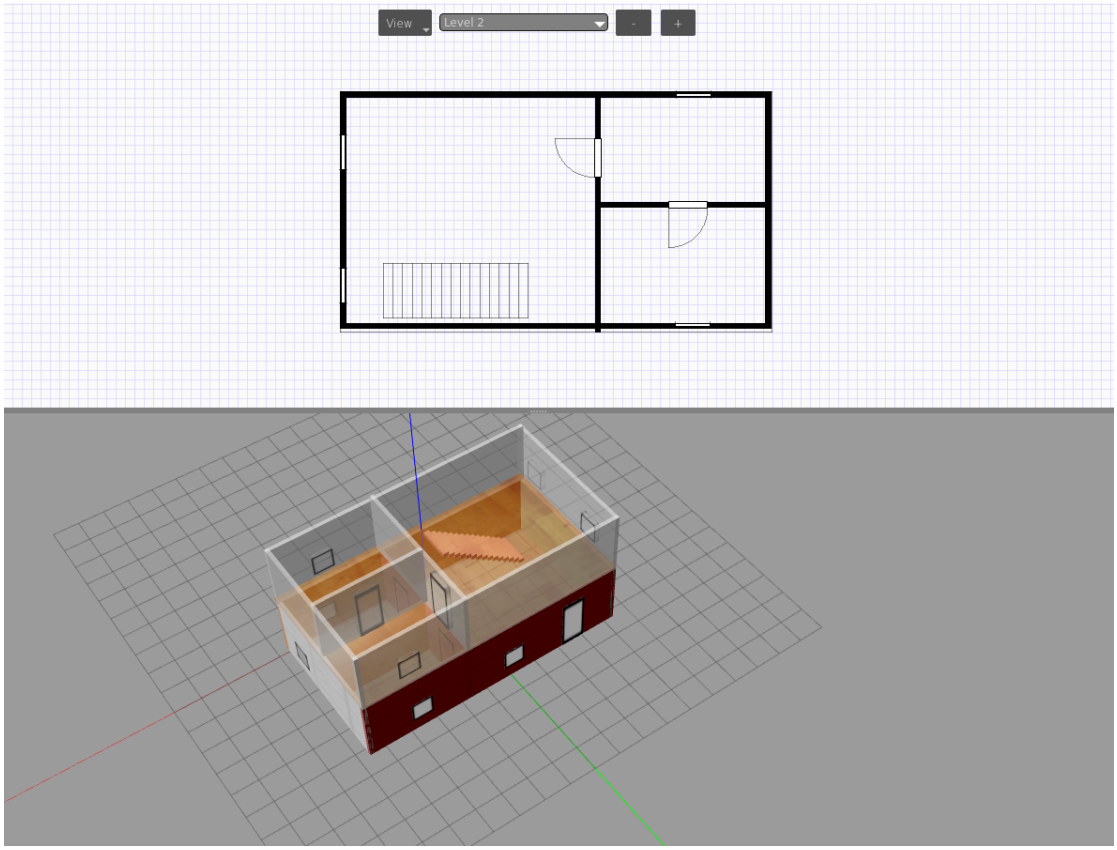
Όπως σε όλα τα γραφικά περιβάλλοντα έτσι και στο Gazebo, προσφέρονται οι ρυθμίσεις του κεντρικού παραθύρου. Οι επιλογές που προσφέρονται είναι οι ακόλουθες:

- **Files**: Save World, Save World As, Save Configuration, Clone World, and Quit.
- **Edit**: Reset Model Poses, Reset World, Building Editor, and Model Editor.
- **Camera:** Περιέχει διάφορες επιλογές για τον τρόπο προβολής.
- **View:** Δίνονται επιλογές για τα Joints, Collisions και άλλα.
- **Window:** Διάφορες επιλογές όπως η πλήρης οθόνης.

Ιδιαίτερα χρήσιμη δυνατότητα είναι το Building Editor στην καρτέλα Edit. Επιλέγοντας το προκύπτει το ακόλουθο παράθυρο στο γραφικό περιβάλλον του Gazebo.




Εικόνα 27. Building editor στο Gazebo.




Εικόνα 28. Κατασκευή σπιτιού.

Με το Building Editor μπορεί να γίνει η σχεδίαση σπιτιών ή κτιρίων, να προστεθούν τοίχοι, πόρτες, παράθυρά και σκάλες, στο μέγεθος και στο ύψος που είναι επιθυμητό. Μια ακόμη σπουδαία ιδιότητα του building editor, βρίσκεται στην καρτέλα **Building Editor<< Import**. Επιλέγοντάς το δίνεται η δυνατότητα να προστεθεί μια έτοιμη μοκέτα ενός κτιρίου και να σχεδιαστεί εκ νέου ένα κτίριο βάση της συγκεκριμένης μοκέτας. Έπειτα αποθηκεύεται το μοντέλο και μπορεί να χρησιμοποιηθεί εντός του Gazebo οποιαδήποτε στιγμή, αφού πλέον θα μπορεί να βρεθεί στην καρτέλα Insert.

## 2.6 Το ρομπότ στο Gazebo

Το Gazebo και το ROS παρότι αναπτύχθηκαν από την ίδια εταιρία αποτελούν δύο διαφορετικές πλατφόρμες οι οποίες αναπτύσσονται ξεχωριστά. Αυτό συνεπάγεται ότι το Gazebo δεν είναι μέρος του ROS, ωστόσο οι δύο αυτές πλατφόρμες μπορούν να συνεργαστούν μεταξύ τους.

Το Gazebo αναμένει ένα ρομποτικό μοντέλο το οποίο είναι σε μορφή SDF, το οποίο είναι format παρόμοιο με το URDF, ωστόσο χρησιμοποιεί κάποια άλλα XML Tags. Η μετατροπή του αρχείου από URDF σε SDF

γίνεται αυτόματα. Για να γίνει η προσθήκη του ρομπότ εντός του Gazebo είναι απαραίτητο να γίνουν κάποιες επιπλέον διαδικασίες.

Στο Gazebo για να προσδιοριστούν κάποια επιπλέον στοιχεία πρέπει να προστεθούν επιπλέον tags, τα οποία είναι τα <gazebo> tags και ονομάζονται Gazebo plugins. Για να προστεθούν αυτά πρέπει να δημιουργηθεί ένα καινούργιο αρχείο URDF το οποίο θα αποθηκευτεί στον αντίστοιχο φάκελο του πακέτου και θα έχει το όνομα ddrobot7.urdf. Το ddrobot7.urdf είναι παρόμοιο με το προηγούμενο με τη διαφορά ότι πρέπει να προστεθούν τα ακόλουθα tags στο αρχείο.

```
<gazebo reference="base_link">
  <material>Gazebo/Blue</material>
</gazebo>
<gazebo reference="right_wheel">
  <material>Gazebo/Black</material>
</gazebo>
  <gazebo reference="left_wheel">
<material>Gazebo/Black</material>
</gazebo>
<gazebo reference="caster">
  <material>Gazebo/Red</material>
</gazebo>
```

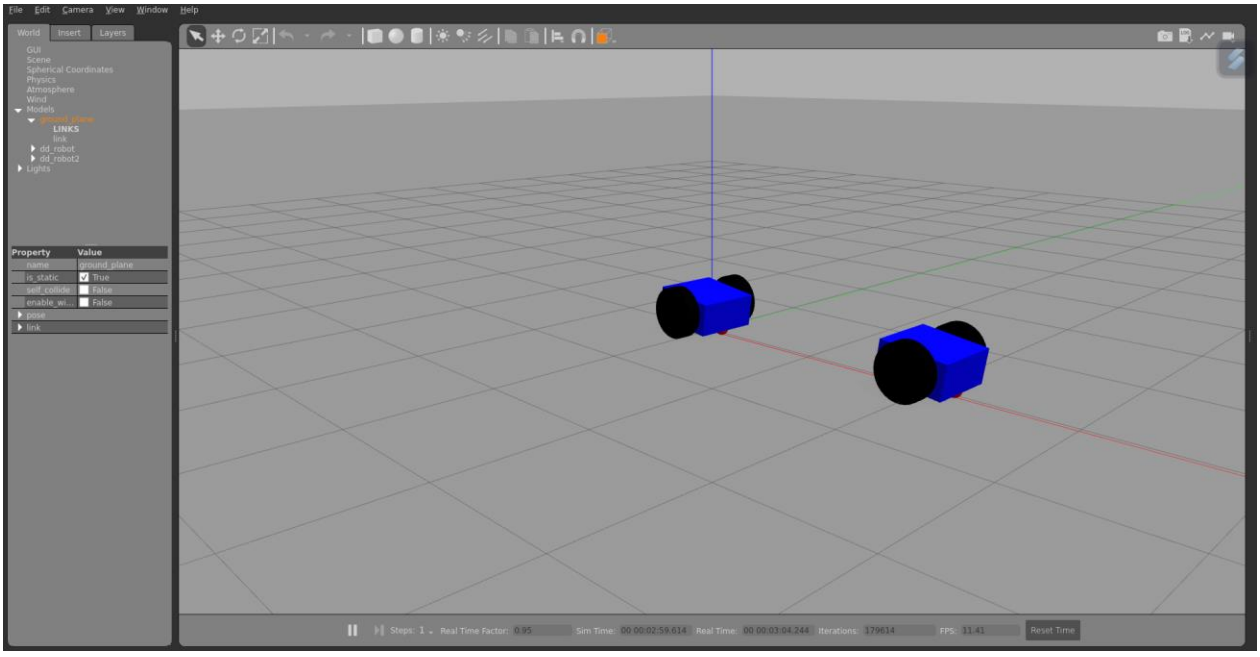
Η προσθήκη των παραπάνω tags είναι ιδιαίτερα σημαντική, διότι με αυτά καθορίζεται το χρώμα των links στο Gazebo. Σε αυτό το σημείο και αφού έχουν προστεθεί τα παραπάνω tags, το ρομποτικό μοντέλο μπορεί να εισαχθεί στο Gazebo. Αρχικά θα πρέπει να χτιστεί το πακέτο και έπειτα να εκτελεστούν οι παρακάτω εντολές για την τοποθέτηση του ρομπότ στο Gazebo.

```
$ # If you are using WSl, you must run the next command.

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch gazebo_ros gazebo.launch.py

$ # Open another terminal and run the next commands.

$ cd ~/ros2_ws

$ cd src/dd_robot

$ ros2 run robot_state_publisher robot_state_publisher urdf/ddrobot7.urdf
```

| |
|---|
| $ # Let the terminals running their executable and open a third one.<br><br>$ ros2 run gazebo_ros spawn_entity.py -topic robot_description -entity dd_robot<br><br>$ # if you want you can spawn **another** robot, in a different position.<br><br>$ ros2 run gazebo_ros spawn_entity.py -topic robot_description -entity dd_robot2 -x 2.0 -y 2.0 |

Πίνακας 63. Spawn ddrobot7 in gazebo μέσω τερματικού.

Οι παραπάνω εντολές επιτυγχάνουν να τοποθετήσουν το ρομπότ στο Gazebo με τη βοήθεια εντολών Command Line. Ιδιαίτερη προσοχή στις παραπάνω εντολές καθώς θα πρέπει να είναι ενεργά τρία διαφορετικά terminal και επίσης πρέπει να εκτελεστεί μια επιπρόσθετη εντολή για τη χρήση του Gazebo σε WSL. Πιο συγκεκριμένα, οι παραπάνω εντολές εκκινούν ένα νέο κόσμο στο Gazebo ο οποίος είναι κενός. Στη συνέχεια ενεργοποιείται το /robot_state_publisher Node που βρίσκεται στο αντίστοιχο πακέτο. Αυτό έχει σαν αποτέλεσμα την δημοσίευση πληροφοριών που αφορούν το ρομπότ στο ROS δίκτυο. Έπειτα καλείται το εκτελέσιμο spawn_entity του πακέτου gazebo_ros με χρήση δύο παραμέτρων. Τελικά επιτυγχάνεται ο στόχος της εισαγωγής του ρομπότ στο Gazebo όπως φαίνεται και στην ακόλουθη εικόνα.



Εικόνα 29. Το δίτροχο ρομπότ στο Gazebo.

Παρά το γεγονός ότι το ρομπότ τοποθετείται επιτυχώς στην προσομοίωση, η διαδικασία τοποθέτησης είναι αρκετά χρονοβόρα και εμπεριέχει αρκετές λεπτομέρειες. Για αυτό το λόγο δημιουργείται ένα καινούργιο launch αρχείο, το οποίο καλεί όλα τα παραπάνω εκτελέσιμα με χρήση μιας εντολής. Το αρχείο αυτό είναι το ακόλουθο και αποθηκεύεται στον αντίστοιχο φάκελο με το όνομα dd_robot_gazebo.launch.py.

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node
import xacro

def generate_launch_description():
    # Specify the name of the package and path to xacro file within the package
    pkg_name = 'dd_robot'
    file_subpath = 'urdf/ddrobot7.urdf'

    # Use xacro to process the file
    xacro_file = os.path.join(get_package_share_directory(pkg_name),file_subpath)
    robot_description_raw = xacro.process_file(xacro_file).toxml()

    # Configure the node
    node_robot_state_publisher = Node( package='robot_state_publisher',
        executable='robot_state_publisher',  output='screen',
        parameters=[{'robot_description': robot_description_raw,
        'use_sim_time': True}] # add other parameters here if required )

    # Launch Gazebo
    gazebo = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([os.path.join(
            get_package_share_directory('gazebo_ros'), 'launch'), '/gazebo.launch.py']), )

    spawn_entity = Node(package='gazebo_ros', executable='spawn_entity.py',
                        arguments=['-topic', 'robot_description', '-entity', 'ddrobot'],
                        output='screen')

    # Run the node
    return LaunchDescription([ gazebo,
            node_robot_state_publisher, spawn_entity ])
```

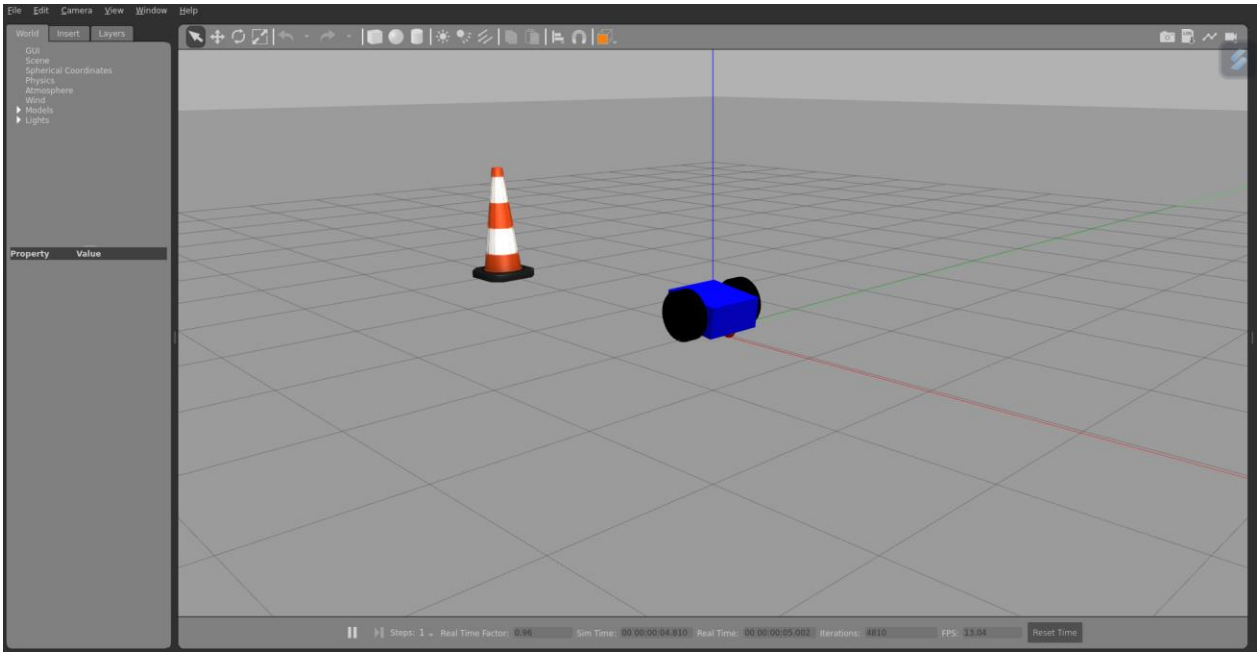
Ο παραπάνω κώδικας επιτυγχάνει να τοποθετήσει το ρομπότ στο περιβάλλον του Gazebo. Στην αρχή του κώδικα εισάγονται κάποιες βιβλιοθήκες που είναι χρήσιμες και έπειτα ορίζεται μια νέα συνάρτηση. Στην αρχή της συνάρτησης καθορίζονται κάποια αντικείμενα τα οποία είναι μονοπάτια για τα πακέτα που θα χρησιμοποιηθούν. Τα μονοπάτια αφορούν το πακέτο και το ρομποτικό μοντέλο που θα τοποθετηθεί στην προσομοίωση.

Στη συνέχεια δημιουργούνται τα node που θα εκκινήσουν στο σύστημα. Πιο συγκεκριμένα, δημιουργείται το robot state publisher και το spawn entity καθώς επίσης και το Gazebo. Τα nodes αυτά θα προστεθούν στο LaunchDecription για να εκκινήσουν το οποίο παρατηρείται στη μέθοδο return της συνάρτησης.

Σε αυτό το σημείο πρέπει να χτιστεί το πακέτο και έπειτα μπορεί να γίνει η εκκινήσει του launch αρχείου για την τοποθέτηση του ρομπότ στο Gazebo.

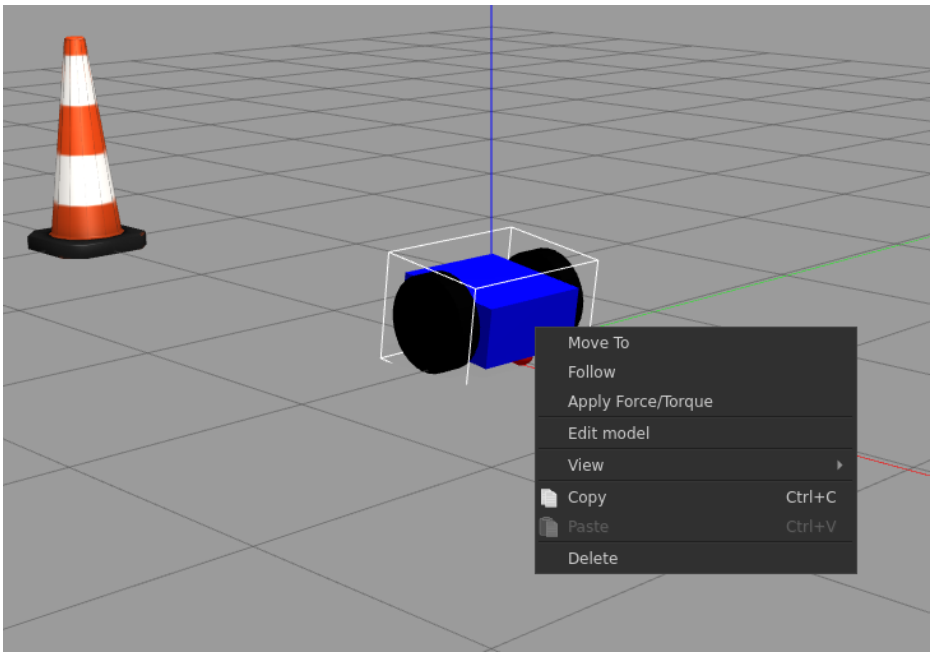
```
$ cd ~/ros2_ws

$ colcon build # For WSL you must run the next command.

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch dd_robot dd_robot_gazebo.launch.py
```

Πίνακας 64. Εκκίνηση του dd_robot_gazebo.launch.py




Εικόνα 30. Το ddrobot7.urdf στο περιβάλλον προσομοίωσης.

Σε αυτό το σημείο και αφού έχει προστεθεί το ρομποτικό μοντέλο στο Gazebo, ενδιαφέρον προσελκύει η άσκηση δυνάμεων στο ρομποτικό όχημα. Αυτό επιτυγχάνεται αρχικά επιλέγοντας το ρομποτικό όχημα και έπειτα κάνοντας δεξί κλικ και επιλέγοντας την επιλογή **Apply Force/Torque**.




Εικόνα 31. Μενού επιλογών για το ρομπότ.

## Κεφάλαιο 3

## Προσομοίωση ενός βραχίονα

Στο συγκεκριμένο κεφάλαιο θα δημιουργηθεί μια προσομοίωση ενός βραχίονα δύο βαθμών ελευθερίας από την αρχή. Αναλυτικότερα θα δημιουργηθούν κάποια αρχεία URDF με χρήση μακροεντολών xacro και θα αναπτυχθούν launch αρχεία για την τοποθέτηση του ρομπότ στο Rviz και στο Gazebo. Στο τέλος του κεφαλαίου θα γίνει χρήση του πακέτου ros2 control για τον έλεγχο του βραχίονα.

### 3.1 Δημιουργία του πακέτου rrbot

Τα πακέτα στο ROS είναι πολύ σημαντικά, καθώς μέσω αυτών μπορεί να εγκατασταθούν και να δημιουργηθούν λογισμικά. Παράλληλα, μπορεί να δημοσιευθεί ο κώδικας που αναπτύχθηκε σε άλλους προγραμματιστές. Όπως και στο προηγούμενο κεφάλαιο το πακέτο που θα δημιουργηθεί θα είναι με χρήση της Python.

Ο βραχίονας θα είναι δύο βαθμών ελευθερίας και τον όνομα του πακέτου θα είναι rrbot. Το όνομα του ρομπότ προκύπτει από το γεγονός ότι θα έχει δύο αρθρώσεις περιστροφής, συνεπώς θα είναι ένας βραχίονας τύπου ‘‘Revolute-Revolute Manipulator’’. Το πακέτο θα δημιουργηθεί εντός ενός ROS workspace και πιο συγκεκριμένα εντός του υποφακέλου src.

```
$ cd ~/ros2_ws/src

$ ros2 pkg create --build-type ament_python rrbot
```

Πίνακας 65. Δημιουργία πακέτου rrbot.

Αφού δημιουργηθεί το πακέτο, τότε γίνεται η μετάβαση στη ρίζα του workspace με στόχο να χτιστεί αυτό.

```
$ cd ~/ros2_ws

$ colcon build --packages-select rrbot
```

Πίνακας 66. Χτίσιμο του πακέτου.

Όπως παρατηρήθηκε στο προηγούμενο κεφάλαιο, έτσι και στη προσομοίωση του βραχίονα θα αναπτυχθούν πολλά αρχεία διαφορετικού τύπου. Για αυτό το λόγο είναι απαραίτητο να υπάρχουν φάκελοι στους οποίους θα αποθηκεύονται τα αντίστοιχα αρχεία. Ο λόγος για τον οποίο γίνεται αυτό είναι για λόγους οργάνωσης του κώδικα που θα αναπτυχθεί.

Αναλυτικότερα θα δημιουργηθούν οι φάκελοι urdf και launch οι οποίοι παρατηρούνται και στη προσομοίωση του δίτροχου ρομπότ. Παράλληλα, θα δημιουργηθούν και δύο νέοι φάκελοι οι οποίοι θα ονομαστούν meshes και config και η χρήση τους θα αναλυθεί παρακάτω. Ακολουθεί η δημιουργία των φακέλων εντός του πακέτου.

```
$ cd ~/ros2_ws/src/rrbot

$ mkdir urdf launch world meshes config
```

Πίνακας 67. Δημιουργία απαραίτητων φακέλων στο πακέτο rrbot.

Για να μπορέσουν να χρησιμοποιηθούν τα αρχεία που θα δημιουργηθούν εντός του εκάστοτε φακέλου, θα πρέπει να γίνει η επεξεργασία του αρχείου setup.py που βρίσκεται εντός του πακέτου. Αρχικά θα πρέπει να προστεθούν δύο νέες βιβλιοθήκες στο συγκεκριμένο αρχείο και έπειτα στην λίστα data_files να προστεθούν κάποια νέα μονοπάτια.

Στο αρχείο setup.py του πακέτου θα προστεθούν οι παρακάτω βιβλιοθήκες στη αρχή του κώδικα.

```
$ from glob import glob
$ import os
```

Πίνακας 68. Προσθήκη βιβλιοθηκών στο αρχείο setup.py.

Στη συνέχεια πρέπει να προστεθούν τα ακόλουθα στην λίστα data files

```
(os.path.join('share',package_name,'launch'),
        glob(os.path.join('launch','*.launch.py'))),
 (os.path.join('share',package_name,'urdf'),
        glob(os.path.join('urdf','*.xacro'))),
  (os.path.join('share',package_name,'urdf'),
        glob(os.path.join('urdf','*.gazebo'))),
  (os.path.join('share',package_name,'worlds'),
        glob(os.path.join('worlds','*.world'))),
 (os.path.join('share',package_name,'meshes'),
        glob(os.path.join('meshes','*.dae'))),
 (os.path.join('share',package_name,'meshes'),
        glob(os.path.join('meshes','*.png'))),
(os.path.join('share',package_name,'config'),
        glob(os.path.join('config','*.yaml'))
```

Πίνακας 69. Προσθήκη μονοπατιών στη λίστα data files.

```python
from setuptools import find_packages, setup
from glob import glob
import os

package_name = 'rrbot'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),

        (os.path.join('share',package_name,'launch'),
         glob(os.path.join('launch','*.launch.py'))),

         (os.path.join('share',package_name,'urdf'),
         glob(os.path.join('urdf','*.xacro'))),

         (os.path.join('share',package_name,'urdf'),
         glob(os.path.join('urdf','*.gazebo'))),

        (os.path.join('share',package_name,'worlds'),
         glob(os.path.join('worlds','*.world'))),

        (os.path.join('share',package_name,'meshes'),
         glob(os.path.join('meshes','*.dae'))),

        (os.path.join('share',package_name,'meshes'),
         glob(os.path.join('meshes','*.png'))),

        (os.path.join('share',package_name,'meshes'),
         glob(os.path.join('meshes','*.stl'))),

        (os.path.join('share',package_name,'config'),
         glob(os.path.join('config','*.yaml'))),
    ],
```

Εικόνα 32. Το νέο setup.py του πακέτου rrbot.

Έπειτα από τις προσθήκες των παραπάνω, η λίστα του αρχείου setup.py θα πρέπει να είναι ανανεωμένη και να περιέχει τα παραπάνω στοιχεία. Έπειτα από την τροποποίηση του setup.py πρέπει να χτιστεί το πακέτο και συνεπώς πρέπει να εκτελεστεί η εντολή του πίνακα 66.

## 3.2 Χαρακτηριστικά του Xacro

Το Xacro είναι η XML macro language για το ROS. Προσφέρει ένα πακέτο από macro υπηρεσίες για την αντικατάσταση κάποιων επαναλαμβανόμενων δηλώσεων με ένα πιο σύντομο και περιεκτικό τρόπο ο οποίος

αναπτύσσεται σε XML format. Το Xacro μπορεί να χρησιμοποιηθεί σε οποιοδήποτε αρχείο XML, αλλά είναι πολύ χρήσιμο σε μεγάλα και σύνθετα URDF αρχεία.

Συνολικά, η xacro μακροεντολές επιτρέπουν τη δημιουργία συντομότερων και πιο ευκολοδιάβαστων XML αρχείων για τα URDF ρομπότ. Η Xacro παρέχει κάποια πλεονεκτήματα σε πολλές διαφορετικές περιοχές της. Τα πιο σημαντικά πλεονεκτήματα είναι τα ακόλουθα:

- **Properties and property blocks**: Σε περίπτωση που μια πληροφορία επαναλαμβάνεται στο URDF αρχείο, τότε το <property> tag χρησιμοποιείται για τον καθορισμό μιας τιμής ως σταθερά. Θα μπορούσαν να χαρακτηριστούν σαν μεγέθη τα οποία η τιμή τους μπορεί να μεταβληθεί αργότερα. Τα properties tags συνήθως καθορίζονται στην αρχή του XML format, ωστόσο ο ορισμός κάποιου σε οποιοδήποτε σημείο του κώδικα δεν θα επηρεάσει το συνολικό αρχείο. Επιπροσθέτως, η μεταβλητή αυτή μπορεί να χρησιμοποιηθεί είτε έχει οριστεί νωρίτερα είτε θα οριστεί αργότερα. Ο ορισμός ενός property γίνεται ως εξής:

    - <xacro:property name=”height” value=”0.5” />

  Το παραπάνω property ορίστηκε με το όνομα height και έχει την τιμή 0.5. Για να γίνει η κλήση του εντός του XML format αυτό πραγματοποιείται μέσω της έκφρασης ${height}.

- **Simple math**: Μπορούν να πραγματοποιηθούν απλές μαθηματικές πράξεις χρησιμοποιώντας τις τέσσερις βασικές πράξεις +, -, * και /. Αυτά θα πρέπει να είναι εσώκλειστα στο ακόλουθο ${}. Μπορούν να χρησιμοποιηθούν είτε ακέραιοι αριθμοί είτε αριθμοί με υποδιαστολή. Επίσης, για μαθηματικές πράξεις μπορούν να χρησιμοποιηθούν και properties που έχουν οριστεί ή θα οριστούν παρακάτω.

- **Macros:** Αυτό είναι το βασικότερο χαρακτηριστικό που προσφέρει η Xacro. Ένα macro, δημιουργείται με χρήση του <xacro:macro> tag και αποτελείται από starting και ending tag. Οτιδήποτε βρίσκεται εντός του start-tag και end-tag είναι μέρος του. Πιο συγκεκριμένα, το macro είναι χρήσιμο όταν υπάρχουν επαναλαμβανόμενες δηλώσεις εντός του XML format. Δηλαδή θα μπορούσε να οριστεί ένα macro το οποίο θα εκτελεί μια συγκεκριμένη διαδικασία η οποία χρησιμοποιείται αρκετές φορές εντός του αρχείο. Συνοπτικά τα macros θα μπορούσαν να παρομοιαστούν με τις συναρτήσεις που μπορούν να οριστούν στη Python ή σε άλλες αντικειμενοστρεφή γλώσσες προγραμματισμού.

- **Συνδυασμός πολλών Xacro Αρχείων**: Σε ένα Xacro αρχείο μπορούν να γίνουν include άλλα Xacro αρχεία τα οποία άλλοτε μπορεί να έχουν οριστεί στο ίδιο πακέτο ή διαφορετικό. Ωστόσο τα αρχεία που μπορούν να γίνουν include πρέπει απαραίτητα να βρίσκονται στο ίδιο workspace. Το include πραγματοποιείται με τον εξής τρόπο.
    - <xacro:include filename="$(find package_name)/filename" />
- Επιπρόσθετα χαρακτηριστικά για τις Xacro μακροεντολές μπορούν να βρεθούν στον σύνδεσμο: http://wiki.ros.org/xacro.

Τα αρχεία τύπου xacro θα αποθηκεύονται στο φάκελο urdf του πακέτου με την κατάληξη .xacro. Μια επιπρόσθετη διαφορά του xacro με το URDF είναι ότι κάθε αρχείο της μορφής xacro θα πρέπει να περιέχει την ακόλουθη εντολή στην αρχή του κώδικα.

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

Η προσθήκη του xmlns στην αρχή του αρχείου είναι ζωτικής σημασίας καθώς είναι απαραίτητο για να γίνει η συντακτική ανάλυση ορθά.

### 3.3 Δημιουργία βραχίονα με χρήση Xacro

Στην πρώτη προσέγγιση κατασκευής του βραχίονα rrbot, θα αναπτυχθεί ένα URDF αρχείο στο οποίο θα καθορίζονται τρία links τα οποία θα έχουν <visual>, <collision> και <inertia> tags. Επιπροσθέτως, θα υπάρχουν δύο joints μεταξύ των links τα οποία με τη σειρά τους θα περιέχουν τα <parent>, <child>, <origin> και <axis> tags.

Το format του rrbot URDF θα είναι αρκετά παρόμοιο με το URDF του δίτροχου ρομπότ που αναπτύχθηκε στο προηγούμενο κεφάλαιο. Οι αλλαγές που προκύπτουν λόγω του Xacro format είναι οι ακόλουθες:

- Προσθήκη του XML namespace στη δεύτερη γραμμή του κώδικα, το οποίο προστίθεται για την συντακτική ανάλυση του κώδικα.
- Χρήση των Xacro <property> tags για τον καθορισμό κάποιων σταθερών.
- Προσθήκη των property names αντί των τιμών κατά τον ορισμό των links.
- Απλές μαθηματικές πράξεις για τον υπολογισμό του link <origin> στην τιμή z.
- Οι αρθρώσεις (joints) του βραχίονα έχουν επιπρόσθετα tags τα οποία είναι τα <dynamics> και <limit>.

Το αρχείο URDF για το βραχίονα δύο βαθμών ελευθερίας είναι το ακόλουθο και θα αποθηκευτεί με το όνομα rrbot.xacro στο φάκελο urdf του πακέτου.

```
<?xml version="1.0"?>
<!-- Revolute-Revolute Manipulator -->
<robot name="rrbot" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <!-- Constants for robot dimensions -->
  <xacro:property name="width" value="0.1" />   <!-- Beams are square in length and width -->
  <xacro:property name="height1" value="2" />   <!-- Link 1 -->
  <xacro:property name="height2" value="1" />   <!-- Link 2 -->
  <xacro:property name="height3" value="1" />   <!-- Link 3 -->
  <xacro:property name="axle_offset" value="0.05" /> <!-- Space between joint and end of beam -->
  <xacro:property name="damp" value="0.7" />    <!-- damping coefficient -->

  <!-- Base Link -->
  <link name="base_link">
    <visual>
      <origin xyz="0 0 ${height1/2}" rpy="0 0 0"/>
        <geometry>
            <box size="${width} ${width} ${height1}"/>
        </geometry>
    </visual>
    <collision>
      <origin xyz="0 0 ${height1/2}" rpy="0 0 0"/>
        <geometry>
            <box size="${width} ${width} ${height1}"/>
        </geometry>
    </collision>
    <inertial>
      <origin xyz="0 0 ${height1/2}" rpy="0 0 0"/>
      <mass value="1"/>
      <inertia
            ixx="1.0" ixy="0.0" ixz="0.0"
            iyy="1.0" iyz="0.0"
            izz="1.0"/>
    </inertial>
  </link>


  <!-- Joint between Base Link and Middle Link -->
  <joint name="joint_base_mid" type="revolute">
    <parent link="base_link"/>
```

```
  <child link="mid_link"/>
  <origin xyz="0 ${width} ${height1 - axle_offset}" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <dynamics damping="${damp}"/>
  <limit effort="100.0" velocity="0.5" lower="-3.14" upper="3.14" />
</joint>

<!-- Middle Link -->
<link name="mid_link">
  <visual>
    <origin xyz="0 0 ${height2/2 - axle_offset}" rpy="0 0 0"/>
    <geometry>
          <box size="${width} ${width} ${height2}"/>
    </geometry>
  </visual>
  <collision>
    <origin xyz="0 0 ${height2/2 - axle_offset}" rpy="0 0 0"/>
    <geometry>
          <box size="${width} ${width} ${height2}"/>
    </geometry>
  </collision>
  <inertial>
    <origin xyz="0 0 ${height2/2 - axle_offset}" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia
          ixx="1.0" ixy="0.0" ixz="0.0"
          iyy="1.0" iyz="0.0"
          izz="1.0"/>
  </inertial>
</link>

<!-- Joint between Middle Link and Top Link -->
<joint name="joint_mid_top" type="revolute">
  <parent link="mid_link"/>
  <child link="top_link"/>
  <origin xyz="0 ${width} ${height2 - axle_offset*2}" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
  <dynamics damping="${damp}"/>
  <limit effort="100.0" velocity="0.5" lower="-3.14" upper="3.14" />
</joint>

<!-- Top Link -->
<link name="top_link">
```

```
    <visual>
      <origin xyz="0 0 ${height3/2 - axle_offset}" rpy="0 0 0"/>
        <geometry>
          <box size="${width} ${width} ${height3}"/>
        </geometry>
    </visual>
    <collision>
      <origin xyz="0 0 ${height3/2 - axle_offset}" rpy="0 0 0"/>
        <geometry>
            <box size="${width} ${width} ${height3}"/>
        </geometry>
    </collision>
    <inertial>
      <origin xyz="0 0 ${height3/2 - axle_offset}" rpy="0 0 0"/>
      <mass value="1"/>
      <inertia
              ixx="1.0" ixy="0.0" ixz="0.0"
              iyy="1.0" iyz="0.0"
              izz="1.0"/>
    </inertial>
  </link>
</robot>
```

Το παραπάνω XML format καθορίζει ένα βραχίονα με όνομα rrbot, ο οποίος έχει τρία links, τα οποία έχουν μήκος και πλάτος 0.1 μέτρα. Το base_link έχει ύψος 2 μέτρα, ενώ τα δύο εναπομείναντα links έχουν ύψος ένα μέτρο. Στο κώδικα παρατηρείται ότι η βάση του βραχίονα που είναι το base link, είναι τοποθετημένο ένα μέτρο προς τα πάνω. Αυτό καθορίζεται στο <origin> tag του αντίστοιχου link και ο λόγος για τον οποίο πραγματοποιείται αυτό είναι για να τοποθετηθεί το ρομπότ πάνω από το επίπεδο (ground plane), όταν εισαχθεί στο Rviz.

Σε περίπτωση που χρησιμοποιείται το VS Code για την ανάπτυξη του XML format, τότε μπορεί μέσω του Preview URDF να προβληθεί το ρομποτικό μοντέλο που έχει δημιουργηθεί. Με Ctrl+Shift+P και επιλογή του Preview URDF ανοίγει ένα νέο παράθυρο στο οποίο προβάλλεται ο βραχίονας. Ωστόσο το Preview, δεν εγγυάται την ορθή δημιουργία του ρομπότ, καθώς δεν είναι εφικτό να εξεταστεί αν οι αρθρώσεις περιστρέφονται κατά τον επιθυμητό τρόπο και στα επιθυμητά όρια.

Όπως και για το δίτροχο ρομπότ έτσι και για τον βραχίονα, θα δημιουργηθεί ένα launch αρχείο το οποίο εκκινεί το Rviz και κάποια απαραίτητα nodes τα οποία επιτυγχάνουν την τοποθέτηση του ρομπότ στο Rviz.

Το launch αρχείο είναι παρόμοιο με αυτό που χρησιμοποιήθηκε και στο δίτροχο ρομπότ με δύο μικρές αλλαγές. Για αρχή πρέπει να γίνει η αλλαγή του πακέτου και του ρομποτικού μοντέλου εντός του κώδικα.

Το Python αρχείο είναι το ακόλουθο και αποθηκεύεται στον αντίστοιχο φάκελο με το όνομα rrbot_rviz.launch.py.

```
from ament_index_python.packages import get_package_share_path
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.conditions import IfCondition, UnlessCondition
from launch.substitutions import Command, LaunchConfiguration
from launch_ros.actions import Node
from launch_ros.parameter_descriptions import ParameterValue

def generate_launch_description():
  # Define the paths
   rrbot_path = get_package_share_path('rrbot')
   default_model_path = rrbot_path / 'urdf/rrbot.xacro'

   # Define the argument
   gui_arg = DeclareLaunchArgument(name='gui', default_value='true', choices=['true', 'false'],
                                   description='Flag to enable joint_state_publisher_gui')

   model_arg = DeclareLaunchArgument(name='model', default_value=str(default_model_path),
                                     description='Absolute path to robot urdf file')

   robot_description = ParameterValue(Command(['xacro ', LaunchConfiguration('model')]),
                                      value_type=str)

   # Define all the Nodes which will start
   robot_state_publisher_node = Node( package='robot_state_publisher',
      executable='robot_state_publisher',
      parameters=[{'robot_description': robot_description}] )

   joint_state_publisher_node = Node( package='joint_state_publisher',
      executable='joint_state_publisher', condition=UnlessCondition(LaunchConfiguration('gui')))

   joint_state_publisher_gui_node = Node( package='joint_state_publisher_gui',
      executable='joint_state_publisher_gui', condition=IfCondition(LaunchConfiguration('gui')))

   rviz_node = Node( package='rviz2', executable='rviz2',
      name='rviz2', output='screen',)
```

```
    return LaunchDescription([
        gui_arg, model_arg,  joint_state_publisher_node,
        joint_state_publisher_gui_node, robot_state_publisher_node, rviz_node])
```

Έπειτα από τη δημιουργία του launch αρχείου, ακολουθεί το χτίσιμο του πακέτου στη ρίζα του workspace. Στη συνέχεια γίνεται η κλήση κατάλληλης εντολής για την τοποθέτηση του ρομπότ στο Rviz. Σε περίπτωση που χρησιμοποιείται το WSL, τότε απαραίτητη είναι η εκτέλεση μιας επιπρόσθετης εντολής πριν την κλήση του launch αρχείου.

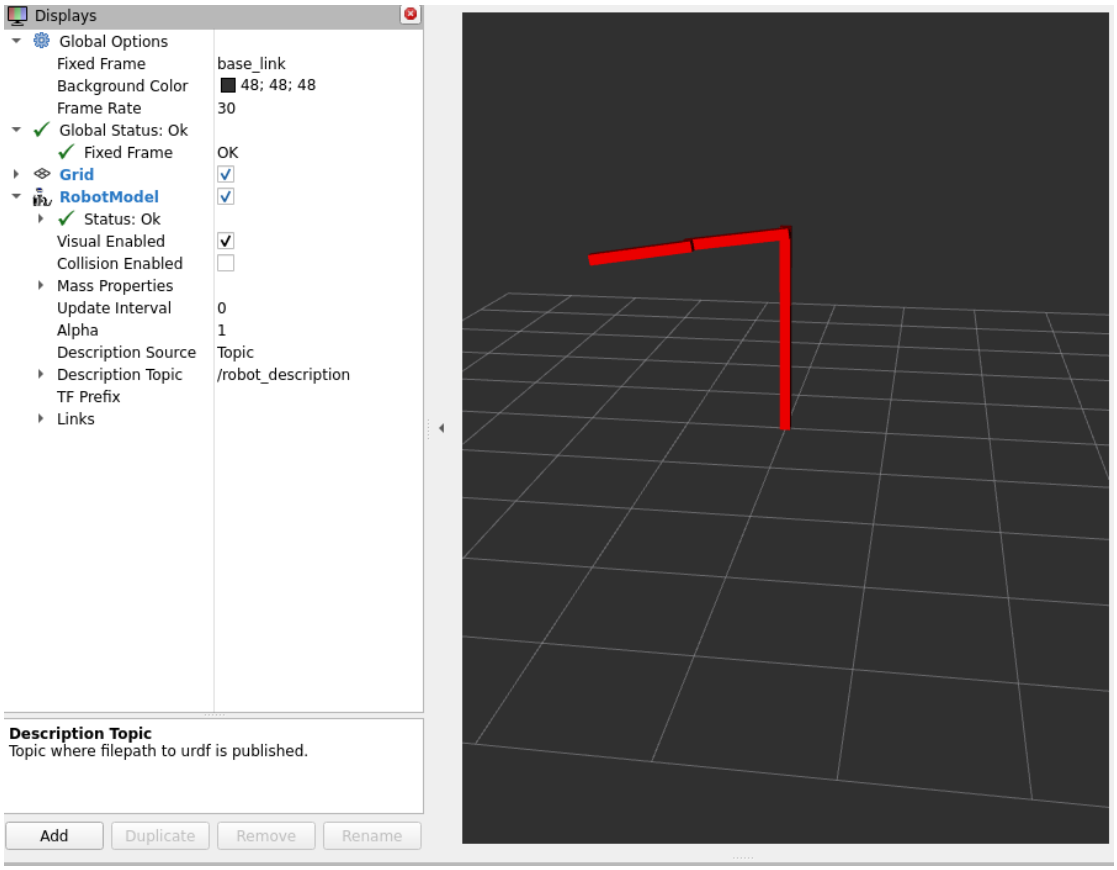
```
$ cd ~/ros2_ws

$ colcon build

$ cd src/rrbot # For WSL, run the next command

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch rrbot rrbot_rviz.launch.py model:=urdf/rrbot.xacro gui:='true'
```

Πίνακας 70. Εκκίνηση του launch αρχείου για τοποθέτηση του rrbot.xacro στο Rviz.

Η τελευταία εντολή εκκινεί το γραφικό περιβάλλον του Rviz, χωρίς να ωστόσο να έχει τοποθετηθεί το ρομποτικό μοντέλο στο τρισδιάστατο χώρο. Για να επιτευχθεί η τοποθέτηση του βραχίονα στο Rviz πρέπει να πραγματοποιηθούν τα ακόλουθα βήματα:

- Αλλαγή της επιλογής στο Fixed frame. Πληκτρολογείτε base_link.
- Μετάβαση στο Display panel και επιλογή του Add. Στο νέο παράθυρο επιλέγεται το Robot Model.
- Αλλαγή του topic στην καρτέλα Robot Model, επιλέγοντας το /robot_description.

Έπειτα από τις παραπάνω αλλαγές το ρομποτικό μοντέλο θα έχει εισαχθεί στο τρισδιάστατο χώρο. Επιπροσθέτως παρατηρείται το παράθυρο του joint state publisher gui παράθυρο, μέσω του οποίου μπορεί να ελεγχθεί η ορθή λειτουργία των αρθρώσεων του ρομποτικού μοντέλου μέσα στα όρια που έχουν οριστεί στο xacro αρχείο.




Εικόνα 33. Το rrbot.xacro στο Rviz.

Παρατηρείται η έλλειψη διαφορετικών χρωμάτων σε κάθε ένα από τα links. Η προσθήκη χρωμάτων θα πραγματοποιηθεί στην επόμενη ενότητα με χρήση των δυνατοτήτων του xacro.

## 3.4 Βελτίωση του rrbot με τη βοήθεια του Xacro

Το Xacro είναι ιδιαίτερα χρήσιμο στην δημιουργία ρομποτικών μοντέλων που φτιάχνονται από την αρχή, προσφέροντας πολλά εργαλεία που είναι χρήσιμα. Στη δημιουργία του rrbot.xacro παρατηρείται η χρήση του <xacro:property> μέσω του οποίου επιτυγχάνεται ο ορισμός κάποιων σταθερών. Επίσης μέσω του Xacro επιτυγχάνεται και ο υπολογισμός κάποιων απλών αριθμητικών πράξεων. Στην ενότητα αυτή θα χρησιμοποιηθούν δύο νέα εργαλεία το <xacro:include> και <xacro:macros>.

Η προσθήκη χρωμάτων στο ρομποτικό μοντέλο θα πραγματοποιηθεί με τη δημιουργία ενός νέου αρχείου και έπειτα μέσω της μεθόδου <xacro:include>, με την οποία το αρχείο αυτό θα εισαχθεί στο rrbot.xacro. Το νέο αυτό αρχείο είναι το materials.xacro και θα αποθηκευτεί στον φάκελο urdf του πακέτου.

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

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

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

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

 <material name="grey">
  <color rgba="0.2 0.2 0.2 1.0"/>
 </material>

 <material name="orange">
  <color rgba="${255/255} ${108/255} ${10/255} 1.0"/>
 </material>

 <material name="brown">
  <color rgba="${222/255} ${207/255} ${195/255} 1.0"/>
 </material>

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

</robot>
```

Στο παραπάνω XML format παρατηρείται ο ορισμός κάποιων βασικών χρωμάτων όπως το κόκκινο και το μπλε. Επίσης, με χρήση του xacro και συγκεκριμένα της ιδιότητας υπολογισμού απλών αριθμητικών παραστάσεων ορίζονται και κάποια άλλα χρώματα όπως είναι το πορτοκαλί και το καφέ, το χρώμα των οποίων υπολογίζεται βάση κάθε συνιστώσας του rgba.

Το αρχείο materials.xacro θα γίνει μέρος του URDF προσθέτοντας το ακόλουθο στο rrbot.xacro:

```
<xacro:include filename="$(find rrbot)/urdf/materials.xacro" />
```

Το χρώμα του κάθε link καθορίζεται με τον ακόλουθο τρόπο εντός του <visual> tag:

<material name="select any color defined in materials.xaccro"/>

Παρατηρείται ότι τα τρία links του rrbot.xacro έχουν παρόμοιο το tag <inertial>, μέσω του οποίου καθορίζεται η μάζα και η αδράνειά του κάθε link. Για αυτό το λόγο χρησιμοποιείται η δυνατότητα <xacro:macro> του Xacro. Οπότε προκύπτει το παρακάτω.

```
<!-- Default Inertial -->
 <xacro:macro name="default_inertial" params="z_value mass">
  <inertial>
    <origin xyz="0 0 ${z_value}" rpy="0 0 0"/>
    <mass value="${mass}" />
    <inertia ixx="1.0" ixy="0.0" ixz="0.0"
          iyy="1.0" iyz="0.0"
          izz="1.0" />
    </inertial>
 </xacro:macro>
```

Το παραπάνω υπολογίζει την ροπή αδράνειας κάθε link και δέχεται δύο παραμέτρους. Η μια αφορά τη μάζα κάθε link και η δεύτερη τη θέση ως προς τον άξονα z. Συνεπώς αφού έχει δημιουργηθεί το παραπάνω format, απομένει η προσθήκη του στο rrbot.xacro και έπειτα οι προσθήκες χρωμάτων σε κάθε ένα από τα links.

Οπότε δημιουργείται ένα νέο αρχείο, το rrbot2.xacro, το οποίο αποθηκεύεται στον φάκελο urdf. Το rrbot2.xacro είναι παρόμοιο με το προηγούμενο μοντέλο για αυτό παρουσιάζονται μόνο οι βελτιώσεις που γίνονται στο προηγούμενο μοντέλο.

```
<?xml version="1.0"?>
<!-- Revolute-Revolute Manipulator -->
<robot name="rrbot" xmlns:xacro="http://www.ros.org/wiki/xacro">

 <!-- Constants for robot dimensions -->
 …
 <!-- Import Rviz colors -->
 <xacro:include filename="$(find rrbot)/urdf/materials.xacro" />

 <!-- Default Inertial -->
 <xacro:macro name="default_inertial" params="z_value mass">
  <inertial>
    <origin xyz="0 0 ${z_value}" rpy="0 0 0"/>
```

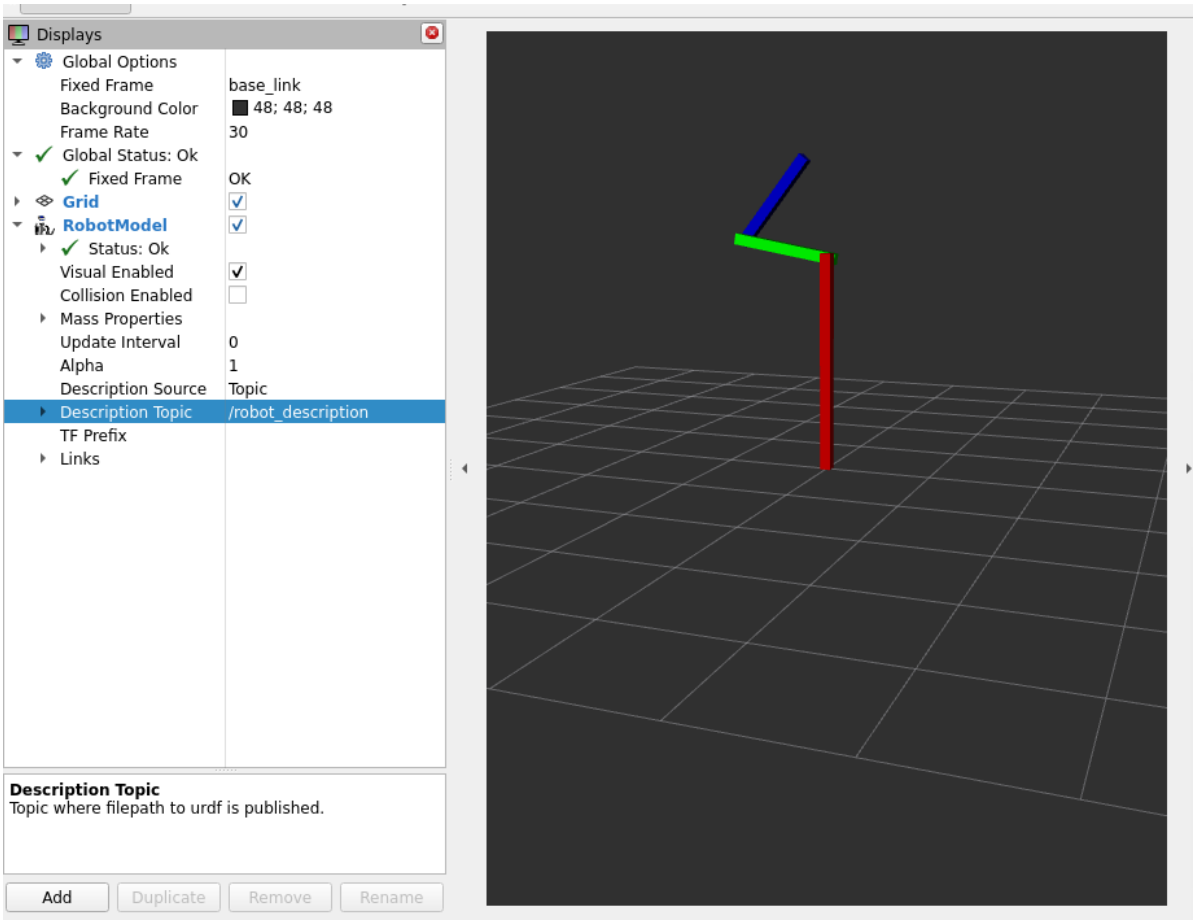
```
    <mass value="${mass}" />
    <inertia ixx="1.0" ixy="0.0" ixz="0.0"
             iyy="1.0" iyz="0.0"
             izz="1.0" />
    </inertial>
 </xacro:macro>

 <!-- Base Link -->
 <link name="base_link">
  <visual>
          …
    <material name="red"/>
  </visual>
          …
  <xacro:default_inertial z_value="${height1/2}" mass="1"/>
 </link>

 <!-- Joint between Base Link and Middle Link -->
 …
 <!-- Middle Link -->
 <link name="mid_link">
  <visual>
          …
    <material name="green"/>
  </visual>
          …
  <xacro:default_inertial z_value="${height2/2 - axle_offset}" mass="1"/>
 </link>

 <!-- Joint between Middle Link and Top Link -->
…

 <!-- Top Link -->
 <link name="top_link">
  <visual>
          …
    <material name="blue"/>
  </visual>
          …
 <xacro:default_inertial z_value="${height3/2 - axle_offset}" mass="1"/>
 </link>
</robot>
```

Η επιβεβαίωση ότι το μοντέλο έχει κατασκευαστεί ορθά θα πραγματοποιηθεί τοποθετώντας το στο Rviz.

```
$ cd ~/ros2_ws

$ colcon build  --packages-select rrbot

$ cd src/rrbot # For WSL you need to run the next command

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch rrbot rrbot_rviz.launch.py model:=urdf/rrbot2.xacro gui:='true'
```

Πίνακας 71. Εκκίνηση του launch για την εισαγωγή του rrbot2 στο Rviz.

Η τελευταία εντολή εκκινεί το γραφικό περιβάλλον του Rviz, αλλά ο βραχίονας δεν έχει τοποθετηθεί εντός του τρισδιάστατου χώρου. Η διαδικασία με την οποία επιτυγχάνεται το παραπάνω είναι η ίδια με νωρίτερα.




Εικόνα 34. Το rrbot2.xacro στο Rviz.

### 3.5 Το ρομποτικό μοντέλο στο Gazebo

Το Gazebo είναι ένα πάρα πολύ σπουδαίο εργαλείο το οποίο σε συνδυασμό με το ROS 2, προσφέρει τη δυνατότητα προσομοίωσης ρομποτικών συστημάτων. Όπως και με το δίτροχο ρομπότ έτσι και ο βραχίονας θα εισαχθεί στο Gazebo και θα ελεγχθεί η συμπεριφορά του στο περιβάλλον προσομοίωσης.

Η επικοινωνία του ROS με το Gazebo επιτυγχάνεται με τη χρησιμοποίηση κάποιων <tags> τα οποία λέγονται Gazebo plugins και πρέπει να εισαχθούν στο αρχείο URDF που αναπτύσσεται. Αυτά τα tags καθορίζουν κάποιες παραμέτρους, όπως είναι το χρώμα, του μοντέλου που θα εισαχθεί στο Gazebo. Στο προηγούμενο κεφάλαιο αυτά τα tags προστέθηκαν στο τέλος του URDF. Στο ρομποτικό βραχίονα, επειδή θα χρησιμοποιηθούν πολλά tags του Gazebo, θα δημιουργηθεί ένα νέο αρχείο και έπειτα αυτό θα γίνει include μέσω της ιδιότητας <xacro:include>.

Δημιουργείται ένα νέο αρχείο το rrbot.gazebo και αποθηκεύεται στο φάκελο urdf. Το αρχείο είναι το ακόλουθο:

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

  <!-- Base Link -->
  <gazebo reference="base_link">
    <material>Gazebo/Red</material>
  </gazebo>

  <!-- Middle Link -->
  <gazebo reference="mid_link">
    <mu1>0.2</mu1>
    <mu2>0.2</mu2>
    <material>Gazebo/Green</material>
  </gazebo>

  <!-- Top Link -->
  <gazebo reference="top_link">
    <mu1>0.2</mu1>
    <mu2>0.2</mu2>
    <material>Gazebo/Blue</material>
  </gazebo>

</robot>
```

Το παραπάνω XML format καθορίζει το χρώμα που θα έχει κάθε link του ρομποτικού μοντέλου. Επίσης καθορίζονται και κάποιοι συντελεστές τριβής για κάθε link.

Αφού ολοκληρωθεί το παραπάνω τότε πρέπει να γίνει include στο ρομποτικό μοντέλο που αναπτύσσεται. Δημιουργείται ένα καινούργιο αρχείο το rrbot3.xacro στο οποίο θα γίνει η χρήση της μεθόδου <xacro:include>. Το αρχείο αυτό είναι παρόμοιο με το προηγούμενο μοντέλο αλλά θα εισαχθεί σε αυτό το αρχείο rrbot.gazebo, συνεπώς στο rrbot3.xacro προστίθεται το ακόλουθο:

```
<!-- Import Gazebo elements, including Gazebo colors -->
<xacro:include filename="$(find rrbot)/urdf/rrbot.gazebo" />
```

Για την τοποθέτηση του ρομπότ εντός του Gazebo θα πρέπει να δημιουργηθεί ένα launch αρχείο, το οποίο θα κάνει spawn το μοντέλο στο Gazebo. Προτού, γίνει η ανάπτυξη του συγκεκριμένου αρχείου, πρέπει να τονιστεί η σημαντικότητα ενός ακόμη βήματος. Όταν το ρομπότ εισαχθεί στο Gazebo, λόγο της δύναμης της βαρύτητας που θα ασκηθεί πάνω του, το ρομπότ θα πέσει. Για το λόγο αυτό θα πρέπει να δημιουργηθεί ένα νέο joint τύπου fixed μεταξύ της βάσης του βραχίονα και του επιπέδου.

Οπότε στο rrbot3.xacro προστίθεται και το ακόλουθο:

```
<!-- Used for fixing rrbot frame to Gazebo world frame -->
<link name="world"/>

<joint name="fixed" type="fixed">
  <parent link="world"/>
  <child link="base_link"/>
</joint>
```

Έπειτα από αυτό, ακολουθεί η ανάπτυξη ενός launch αρχείου για την εισαγωγή του ρομποτικού μοντέλου στο Gazebo. Στο φάκελο launch, δημιουργείται το αρχείο rrbot_gazebo.launch.py το οποίο είναι το παρακάτω:

```
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node
import xacro
```

```
def generate_launch_description():
    # Specify the name of the package and path to xacro file within the package
    pkg_name = 'rrbot'
    file_subpath = 'urdf/rrbot3.xacro'

    # Use xacro to process the file
    xacro_file = os.path.join(get_package_share_directory(pkg_name),file_subpath)
    robot_description_raw = xacro.process_file(xacro_file).toxml()

    # Configure the node
    node_robot_state_publisher = Node(package='robot_state_publisher',
        executable='robot_state_publisher', output='screen',
        parameters=[{'robot_description': robot_description_raw,
        'use_sim_time': True}] # add other parameters here if required )

    # Launch Gazebo
    gazebo = IncludeLaunchDescription(
        PythonLaunchDescriptionSource([os.path.join(
            get_package_share_directory('gazebo_ros'), 'launch'), '/gazebo.launch.py']), )

    spawn_entity = Node(package='gazebo_ros', executable='spawn_entity.py',
                arguments=['-topic', 'robot_description',
                           '-entity', 'rrbot'],  output='screen')

    # Run the node
    return LaunchDescription([
            gazebo, node_robot_state_publisher,spawn_entity  ])
```

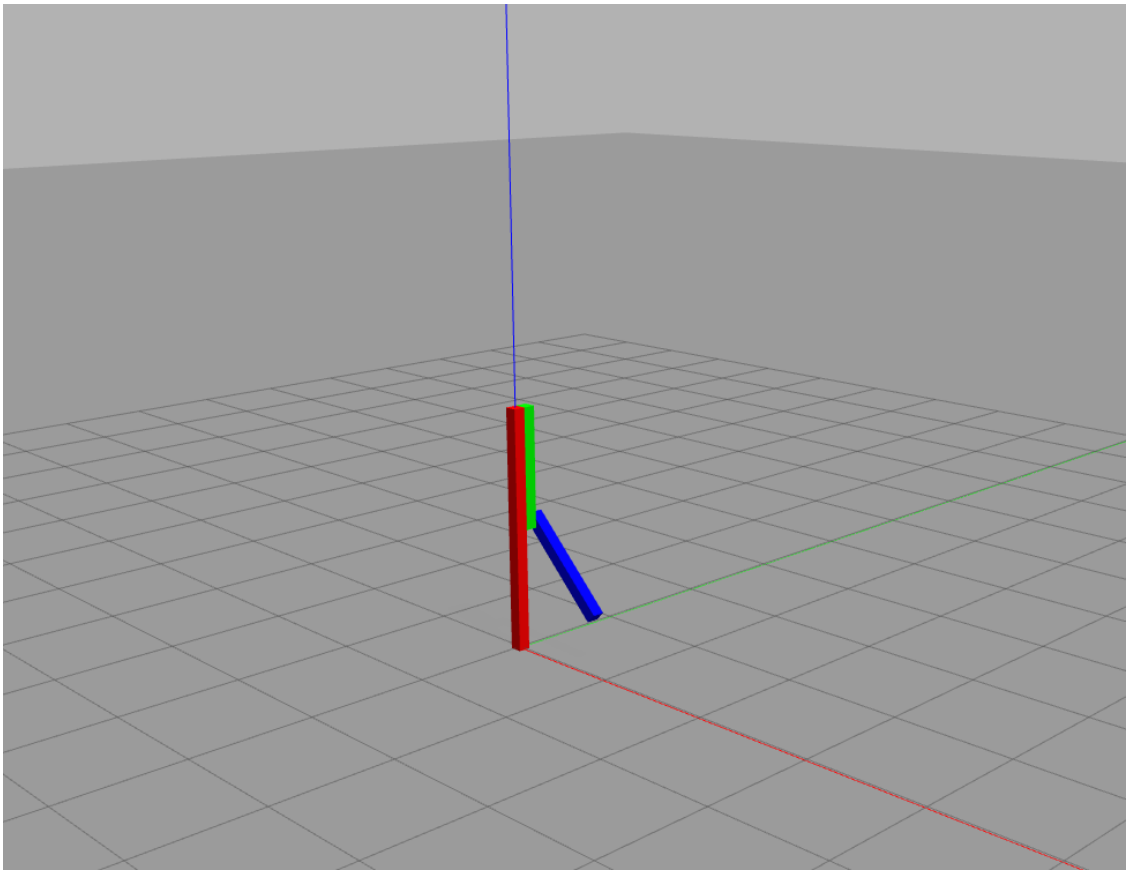
Το παραπάνω αρχείο εκκινεί το Gazebo σε ένα κενό κόσμο και επιτυγχάνει να τοποθετήσει το ρομπότ μέσα στο περιβάλλον προσομοίωσης. Όπως συνηθίζεται, στην αρχή του κώδικα εισάγονται οι απαιτούμενες βιβλιοθήκες. Στη συνέχεια, ορίζεται η συνάρτηση generate_launch_description, στην αρχή της οποίας ορίζονται κάποια μονοπάτια για τα πακέτα που θα χρησιμοποιηθούν και έπειτα ορίζονται κάποια nodes. Έπειτα χρησιμοποιείται η μέθοδος, Python Launch Description Source, η οποία βρίσκει ένα εκτελέσιμο ενός άλλου πακέτου και το εισάγει στον κώδικα. Έπειτα τα nodes προστίθενται στη μέθοδο return της συνάρτησης ώστε να εκκινήσουν όταν καλεστεί το launch file.

Έπειτα από τα παραπάνω, για να γίνει η εκκίνηση του launch αρχείου, πρέπει να γίνει το χτίσιμο του πακέτου, να εκτελεστεί μια επιπρόσθετη εντολή σε περίπτωση που το ROS εκτελείται μέσω WSL και τέλος να γίνει η κλήση του launch αρχείου.

```
$ cd ~/ros2_ws

$ colcon build --package-select rrbot

$ cd src/rrbot # For WSL you need to run the next command.

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch rrbot rrbot_gazebo.launch.py
```

Πίνακας 72. Εκκίνηση του rrbot_gazebo.launch.py.

Κατά την εκκίνηση του Gazebo, ο βραχίονας βρίσκεται στην κατάσταση που έχει οριστεί και φαίνεται στο Rviz. Ωστόσο, αρχίζει και ασκείται η βαρύτητα σε αυτόν και αυτό έχει σαν αποτέλεσμα οι αρθρώσεις του βραχίονα να αλλάξουν κατάσταση και να πέσουν. Αυτό θα διορθωθεί παρακάτω καθώς θα γίνει η προσθήκη ελεγκτών στο ρομποτικό σύστημα.



Εικόνα 35. Το rrbot3.xacro στο Gazebo.

### 3.6 Προσθήκη ενός gripper

Για την πλήρη κατασκευή του βραχίονα απαραίτητη είναι η προσθήκη ενός εργαλείου στο άκρο του. Τα εργαλεία που συνηθίζεται να υπάρχουν στην άρκη του βραχίονα είναι κυρίως αρπάγες. Στο ρομποτικό μοντέλο rrbot θα προστεθεί ένα απλό gripper στην άκρη του βραχίονα. Το μοντέλο της αρπάγης που θα προστεθεί δεν θα έχει απλή γεωμετρία όπως τα γεωμετρικά σχήματα που έχουν χρησιμοποιηθεί μέχρι τώρα. Μοντέλα με σύνθετη γεωμετρία ονομάζονται mesh files και εισάγονται με την μέθοδο xacro include στα μοντέλα URDF.

Αρχικά θα πρέπει να βρεθεί αυτό το τρισδιάστατο μοντέλο αρπάγης. Γίνεται η μετάβαση στον σύνδεσμο: https://github.com/DimitrisKatos/rrbot/tree/master/meshes. Έπειτα ακολουθεί η λήψη των αρχείων και η αποθήκευση τους στο φάκελο meshes του πακέτου.

Για να μπορέσει να γίνει η προσθήκη της αρπάγης απαιτούνται και κάποια επιπρόσθετα βήματα. Για αρχή στο αρχείο rrbot.gazebo, πρέπει να προστεθούν τα ακόλουθα:

```
<!-- Gripper Elements -->
  <gazebo reference="left_gripper">
    <mu1>0.2</mu1>
    <mu2>0.2</mu2>
   </gazebo>

   <gazebo reference="right_gripper">
    <mu1>0.2</mu1>
    <mu2>0.2</mu2>
   </gazebo>

   <gazebo reference="left_tip" />
   <gazebo reference="right_tip" />
```

Τα παραπάνω είναι απαραίτητα, καθώς μέσω αυτών καθορίζεται ο συντελεστής τριβής της αρπάγης. Στη συνέχεια γίνεται η ανάπτυξη του αρχείου της αρπάγης χρησιμοποιώντας τα mesh files που εγκαταστάθηκαν στο πακέτο. Το αρχείο αυτό αποθηκεύεται στο φάκελο urdf με το όνομα gripper.xacro και μπορεί να βρεθεί στον σύνδεσμο: https://github.com/DimitrisKatos/rrbot/blob/master/urdf/gripper.xacro.

Στο μοντέλο gripper.xacro καθορίζονται δύο νέα link, μέσω των οποίων καθορίζονται η αριστερή και η δεξιά αρπάγη του εργαλείου. Έπειτα καθορίζονται και δύο revolute joints, με τα οποία επιτυγχάνεται η ένωση της αρπάγης με το άκρο του ρομποτικού βραχίονα. Άξιο αναφοράς είναι τα collision tags του

μοντέλου δεν είναι ίδια με αυτά του visual tag. Πιο συγκεκριμένα, η γεωμετρία ενός mesh file είναι σύνθετη με αποτέλεσμα αν τοποθετηθεί το ίδιο collision να δημιουργηθούν καθυστερήσεις στη προσομοίωση.

Ο λόγος που αλλάζει το collision είναι διότι θα υπήρχε μεγάλη κατανάλωση υπολογιστικών πόρων κατά την προσομοίωση με αποτέλεσμα να μην πραγματοποιούνται οι διεργασίες που είναι επιθυμητό λόγο καθυστερήσεων. Για τον παραπάνω λόγο η αρπάγη έχει όσο πιο απλό collision γίνεται και για αυτό το λόγο ορίζεται μια γεωμετρία τύπου box σε κάθε μια από τις αρπάγες του.

Στη συνέχεια δημιουργείται ένα ανανεωμένο URDF που καθορίζει το ρομποτικό βραχίονα με το εργαλείο του στην άκρη αυτού. Το μοντέλο θα ονομαστεί rrbot4.xacro και αποθηκεύεται στον αντίστοιχο φάκελο. Το αρχείο είναι παρόμοιο με το rrbot3.xacro αλλά προστίθεται το ακόλουθο.

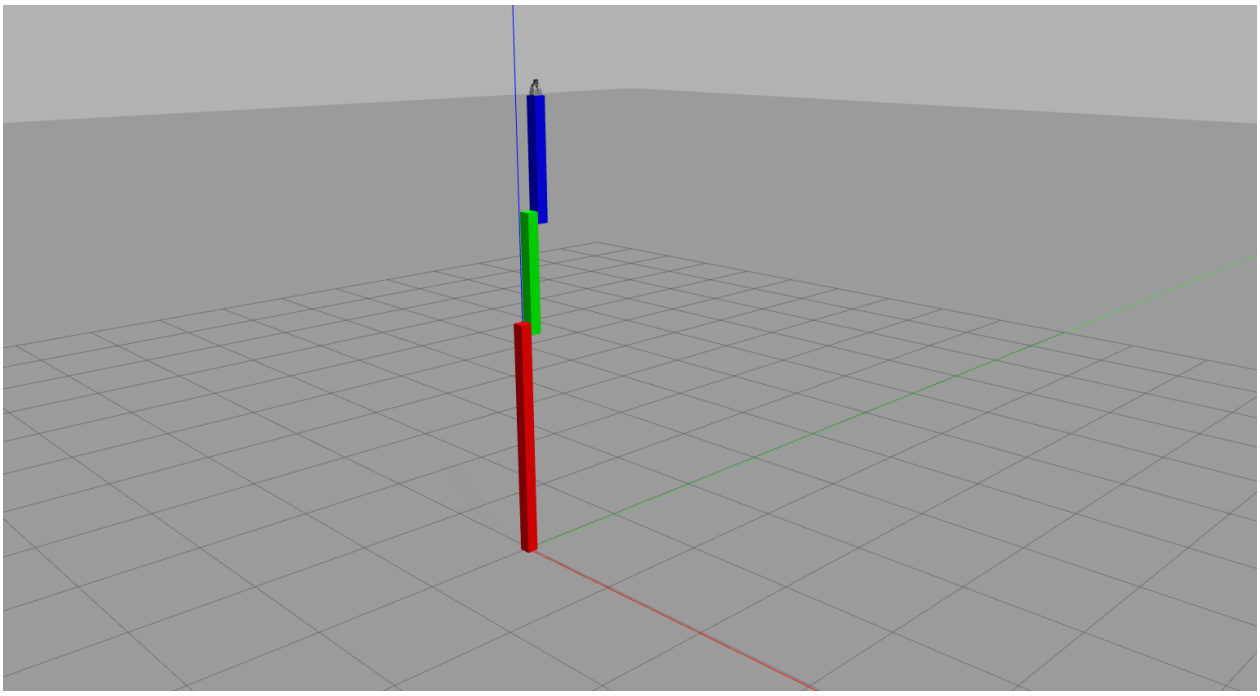
```
<!-- Import gripper URDF -->
<xacro:include filename="$(find rrbot)/urdf/gripper.xacro" />
```

Έπειτα από τα παραπάνω βήματα το ρομποτικό μοντέλο μπορεί να εισαχθεί στο gazebo. Πριν γίνει η εκκίνηση του launch file, είναι απαραίτητο να γίνει η αλλαγή του ρομποτικού μοντέλου που θα τοποθετηθεί στο Gazebo μέσω του αρχείου rrbot_gazebo.launch.py.

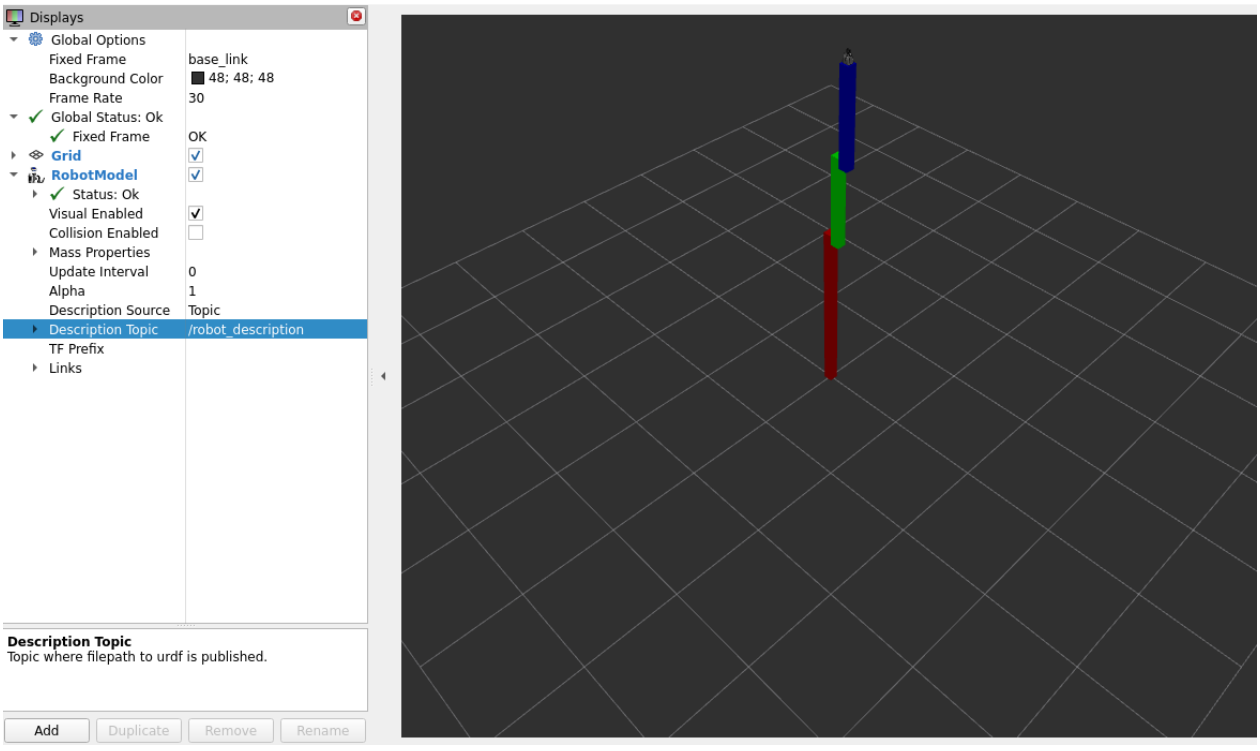
Έπειτα θα ακολουθήσει το χτίσιμο του πακέτου, η εκτέλεση επιπρόσθετης εντολής σε περίπτωση που τα Linux εκτελούνται μέσω WSL και τέλος να εκκινήσει το launch file. Επιπροσθέτως μπορεί να εκκινήσει και το Rviz σε ένα άλλο τερματικό με στόχο να παρατηρηθεί η αρπάγη και στο συγκεκριμένο περιβάλλον. Στο περιβάλλον του Rviz μπορεί να γίνει ο έλεγχος της ορθής λειτουργίας των αρθρώσεων της αρπάγης.

```
$ cd ~/ros2_ws

$ colcon build --package-select rrbot

$ cd src/rrbot # For WSL you need to run the next command.

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch rrbot rrbot_gazebo.launch.py

$ # Another terminal. If you are using WSL run one more command.

$ ros2 launch rrbot rrbot_rviz.launch.py model:=urdf/rrbot4.xacro gui:='true'
```

Πίνακας 73. Η αρπάγη στο Gazebo και Rviz.



Εικόνα 36. Το rrbot με το gripper στο Gazebo.




Εικόνα 37. Το rrbot με το gripper στο Rviz.

### 3.7 Το πακέτο ros2_control

Το ros2_control είναι ένα framework πραγματικού χρόνου για τον έλεγχο ρομποτικών συστημάτων. Αποτελείται από πλήθος πακέτων και η βάση του είναι το πακέτο ros_control που χρησιμοποιούνταν στο ROS 1. Ο στόχος του πακέτου είναι να απλοποιήσει κάποια καινούργια hardware και να αντιμετωπίσει κάποια μειονεκτήματα της προηγούμενης έκδοσης.

Για την εγκατάσταση του πακέτου αλλά και των διαθέσιμων αρχείων που υπάρχουν στο αποθετήριο του ros2_control, πρέπει να εκτελεστούν οι παρακάτω εντολές:

```
$ cd ~/ros2_ws/

$ wget https://raw.githubusercontent.com/ros-controls/control.ros.org/master/ros_controls.$ROS_DISTRO.repos

$ vcs import src < ros_controls.$ROS_DISTRO.repos
```

Πίνακας 74. Λήψη του ros2_control.

Για την εγκατάσταση του πακέτου πρέπει να εκτελεστούν τα παρακάτω:

```
$ rosdep update --rosdistro=$ROS_DISTRO

$ sudo apt-get update

$ rosdep install --from-paths src --ignore-src -r -y
```

Πίνακας 75. Εγκατάσταση του ros2_control.

Για το χτίσιμο των πακέτων απαιτείται η εκτέλεση των παρακάτω:

```
$ . /opt/ros/${ROS_DISTRO}/setup.sh

$ colcon build
```

Πίνακας 76. Χτίσιμο του ros2_control.

Μέσω των παραπάνω εντολών εγκαθίστανται απαραίτητα πακέτα, βιβλιοθήκες και εργαλεία για τον έλεγχο ρομπότ. Επιπρόσθετα το πακέτο συνοδεύεται από εντολές τερματικού οι οποίες βοηθούν στη λειτουργία του συστήματος και οι οποίες εγκαθίστανται με τις παραπάνω εντολές.

Για την δημιουργία ενός ελεγκτή για το rrbot, είναι απαραίτητη η εγκατάσταση κάποιων επιπρόσθετων πακέτων, μέσω των οποίων εγκαθίστανται κάποια demos του πακέτου. Αναλυτικότερα, εγκαθιστάτε

ποικιλία παραδειγμάτων για τον έλεγχο ενός βραχίονα δύο αρθρώσεων περιστροφής, ενός οχήματος διαφορικής οδήγησης και ενός βραχίονα έξι αρθρώσεων περιστροφής. Επίσης, εγκαθίστανται και παραδείγματα που εκτελούνται σε περιβάλλοντα προσομοίωσης όπως είναι το Gazebo, το Gazebo Classic και το Webots.

Για την εγκατάσταση των πακέτων εκτελούνται οι ακόλουθες εντολές:

```
$ cd ~/ros2_ws/src

$ git clone https://github.com/ros-controls/ros2_control_demos

$ cd ~/ros2_ws/

$ sudo apt install ros-humble-controller-manage

$ rosdep update --rosdistro=$ROS_DISTRO

$ sudo apt-get update

$ rosdep install --from-paths src --ignore-src -r –y

$ . /opt/ros/${ROS_DISTRO}/setup.sh

$ colcon build

$ sudo apt install ros-humble-gazebo-ros2-control-demos
```

Πίνακας 77. Εγκατάσταση demos του ros2_control package.

Το ros2_control μπορεί να υποστηρίξει διάφορων τύπου ρομπότ όπως βραχίονες και αυτόνομα ρομπότ. Όσον αφορά τα αυτόνομα ρομπότ, με χρήση του πακέτου μπορεί να γίνει ο έλεγχος ενός οχήματος διαφορικής οδήγησης, ενός δίκυκλου οχήματος, ενός τρίκυκλου με τον ένα τροχό να είναι οδηγούμενος και τους άλλους δύο να είναι σταθεροί και συνδεδεμένοι μεταξύ τους. Επίσης δίνεται η δυνατότητα ελέγχου και οχημάτων τύπου Ackerman, δηλαδή οχημάτων που οι κινηματικές του εξισώσεις είναι παρόμοιες με αυτές ενός αυτοκινήτου.

Όσον αφορά τους βραχίονες, υπάρχει η δυνατότητα ελέγχου ασχέτως των βαθμών ελευθερίας που θα έχει αυτός. Πιο συγκεκριμένα, μπορεί να ενεργοποιηθούν ελεγκτές μέσω των οποίων να δίνεται επιθυμητή θέση (position), επιθυμητή ταχύτητα (velocity), επιθυμητή επιτάχυνση (acceleration) ή επιθυμητή προσπάθεια (effort). Στο rrbot θα χρησιμοποιεί ένας controller τύπου joint_trajectory_controller και θα του γίνεται δήλωση επιθυμητής θέσης.

Παράλληλα με την επιλογή του τύπου ενός ελεγκτή, πολύ σημαντικό ρόλο έχει και η επιλογή του hardware που θα χρησιμοποιηθεί στην εκάστοτε προσομοίωση. Δίνεται η δυνατότητα ανάπτυξης hardware, τα οποία είναι αρχεία ανεπτυγμένα σε C++. Ωστόσο αφού ο βραχίονας θα τοποθετηθεί σε μοντέλο προσομοίωσης θα χρησιμοποιηθεί το έτοιμο hardware του gazebo. Στα διάφορα demos που εγκαθίστανται αναπτύσσονται hardwares τα οποία μπορούν να μελετηθούν.

Η εκκίνηση του ελεγκτή κάθε άρθρωσης θα πραγματοποιείται μέσω της εκκίνησης κάποιων launch αρχείων. Για την ορθή λειτουργία των controllers θα πρέπει να ενεργοποιηθεί και ένας controller τύπου broadcaster. Πιο συγκεκριμένα, θα εκκινήσει το joint_state_broadcaster το οποίο είναι υπεύθυνο για την επικοινωνία μεταξύ κάποιων topics που θα ενεργοποιηθούν κατά την εκκίνηση του συστήματος.

Συμπληρωματικά, το πακέτο ros2_control υποστηρίζει και άλλους ελεγκτές τύπου broadcaster. Οι ελεγκτές αυτού του τύπου είναι υπεύθυνοι να διαβάζουν πληροφορίες του περιβάλλοντος του ρομπότ. Πιο συγκεκριμένα, υποστηρίζονται άλλα τρία ήδη broadcasters, τα οποία είναι τα imu_sensor_broadcaster, τα range_sensor_broadcaster και το force_torque_sensor_broadcaster.

Εκτός από του παραπάνω διαθέσιμου ελεγκτές, με χρήση του πακέτου μπορεί να δημιουργηθεί ένα ελεγκτής από την αρχή που να ανταποκρίνεται καλύτερο στο εκάστοτε πρόβλημα που είναι επιθυμητό να λυθεί. Το ros2_control είναι από τα σπουδαιότερα και μεγαλύτερα πακέτου του ROS 2 και εξελίσσεται διαρκώς.

### 3.8 Ο έλεγχος του rrbot

Για τον έλεγχο του βραχίονα δύο αρθρώσεων περιστροφής θα δημιουργηθεί ένας PID controller για κάθε άρθρωση του βραχίονα αλλά και για την αρπάγη. Έπειτα ο ελεγκτής θα δοκιμαστεί στο περιβάλλον προσομοίωσης του Gazebo, θα σχολιαστεί η σύνδεση του με το Rviz. Τέλος θα γίνει προσπάθεια βελτίωσης του ελεγκτή σε περίπτωση που αυτός δεν προσφέρει επιθυμητά αποτελέσματα. Παράλληλα θα γίνει η χρήση του εργαλείου rqt.

Για την υλοποίηση του ελέγχου του ρομπότ θα χρειαστεί να δημιουργηθούν πληθώρα νέων αρχείων. Για αρχή είναι αναγκαία η δημιουργία ενός νέου τύπου αρχείου, το οποίο θα είναι ανεπτυγμένο σε μορφή yaml. Έπειτα θα πρέπει να προστεθεί ένα νέο Gazebo plugin το οποίο θα δέχεται για παράμετρο το yaml αρχείο των controllers. Στη συνέχεια είναι απαραίτητη η δημιουργία ενός νέου αρχείου στο οποίο ορίζονται απαραίτητα στοιχεία που χρειάζονται για την εκκίνηση του ros2_control. Κλείνοντας, θα αναπτυχθεί ένα νέο launch αρχείο για την εκκίνηση του Gazebo και των Controllers.

### 3.8.1 Δημιουργία του PID Controller

Στην ενότητα αυτή αναπτύσσεται ένα αρχείο τύπου yaml στο οποίο θα οριστεί ο τύπος κάθε controller καθώς επίσης και ο PID Controller που θα έχει η κάθε άρθρωση του βραχίονα αλλά και η αρπάγη. Στο φάκελο config θα δημιουργηθεί ένα νέο αρχείο με όνομα rrbot_controllers.yaml το οποίο έχει την ακόλουθη μορφή:

```
controller_manager:
  ros__parameters:
    update_rate: 100  # Hz
    joint_base_mid_position_controller:
      type: joint_trajectory_controller/JointTrajectoryController
    joint_gripper_controller:
      type: joint_trajectory_controller/JointTrajectoryController
    joint_mid_top_position_controller:
      type: joint_trajectory_controller/JointTrajectoryController
    joint_state_broadcaster:
      type: joint_state_broadcaster/JointStateBroadcaster

joint_base_mid_position_controller:
  ros__parameters:
    joints:
      - joint_base_mid
    command_interfaces:
      - position
    state_interfaces:
      - position
      - velocity
    pid:
      joint_base_mid:
        p: 10
        i: 0.7
        d: 1

joint_mid_top_position_controller:
  ros__parameters:
    joints:
      - joint_mid_top
    command_interfaces:
      - position
    state_interfaces:
      - position
```

```
    - velocity
  pid:
    joint_base_mid:
      p: 10
      i: 0.7
      d: 1
joint_gripper_controller:
  ros__parameters:
    joints:
      - right_gripper_joint
      - left_gripper_joint
    command_interfaces:
      - position
    state_interfaces:
      - position
      - velocity
    pid:
      right_gripper_joint:
        p: 1.0
        i: 0.00
        d: 0.0
      left_gripper_joint:
        p: 1.0
        i: 0.00
        d: 0.0
```

Στο παραπάνω καθορίζονται οι ελεγκτές που θα εκκινήσουν στο σύστημα και συνολικά για κάθε ελεγκτή δηλώνονται τέσσερις παράμετροι. Πιο συγκεκριμένα, ορίζονται δύο ξεχωριστοί ελεγκτές για κάθε άρθρωση, ένας ο οποίος είναι υπεύθυνος για την αρπάγη και ένας ελεγκτής τύπου broadcaster. Ο τύπος ελεγκτή είναι ο joint_trajectory_controller για τις αρθρώσεις και την αρπάγη, ο οποίος είναι ένας τύπος ελεγκτή που προσφέρεται από το ros2_control package.

Αφού οι τρεις ελεγκτές οριστούν εντός του controller_manager, ξεκινάει ο ορισμός παραμέτρων σε κάθε ένα από αυτά. Η πρώτη παράμετρος αφορά την άρθρωση που θα έχει εφαρμογή ο ελεγκτής. Η δεύτερη αφορά το command interface και δηλώνεται το position, συνεπώς μπορούν να δοθούν μόνο επιθυμητές τοποθετήσεις μέσω command line εντολών. Στη συνέχεια δηλώνονται τα state interface που παρέχουν το feedback για το position και velocity. Τέλος, δηλώνονται τιμές για τον εκάστοτε PID ελεγκτή.

Για τον ελεγκτή τύπου broadcaster δηλώνεται να είναι τύπου /joint_state_broadcaster. Ο broadcaster δεν είναι ελεγκτής αλλά πρέπει να εκκινήσει με το υπόλοιπο σύστημα καθώς αυτός διαβάζει την κατάσταση

που βρίσκονται οι αρθρώσεις και δημοσιεύει τα δεδομένα σε κατάλληλα topic του συστήματος. Συνεπώς οι broadcasters δεν έχουν command line εντολές.

Συνοπτικά, όταν θα γίνει η εκκίνηση του παραπάνω αρχείου, θα δημιουργηθούν τρία action ένα για κάθε ελεγκτή. Με εντολές τερματικού, μπορεί να δοθεί εντολή επιθυμητής θέσης για κάθε άρθρωση και να παρατηρηθεί η ενημέρωση που παρέχει το action. Η ενημέρωση αυτή θα αφορά το position και velocity που δηλώθηκαν στο state interface. Τελικά μέσω της ενημέρωσης που παρέχεται θα μπορεί να εξεταστεί αν οι παράμετροι που έχουν δοθεί στον PID controller ικανοποιούν τις απαιτήσεις του συστήματος.

Με κατάλληλες διαφοροποιήσεις μπορούν να δοθούν άλλα ορίσματα στο παραπάνω configuration file και να αλλάξει η δομή αλλά και η αλληλεπίδραση που θα έχει το πακέτο με όλο το σύστημα. Για παράδειγμα μπορεί να διαμορφωθεί κατάλληλα ώστε να μπορούν να δοθούν με command line εντολές επιθυμητές ταχύτητες ή επιταχύνσεις αλλά και να τυπώνονται περισσότερα feedbacks.

### 3.8.2 Προσθήκες στο urdf

Μετά τη δημιουργία των ελεγκτών του βραχίονα και της αρπάγης, ακολουθούν κάποιες απαραίτητες προσθήκες στο urdf αρχείο που έχει δημιουργηθεί έως τώρα. Πιο συγκεκριμένα, θα δημιουργηθεί ένα νέο αρχείο το οποίο θα είναι αφιερωμένο στο ros2_control. Εκεί θα οριστεί το hardware που θα χρησιμοποιηθεί, δηλαδή του Gazebo, και επιπροσθέτως θα οριστούν κάποιες ακόμη παράμετροι του συστήματος, όπως είναι τα command και state interface. Παράλληλα θα οριστούν και τα transmission-επενεργητές για κάθε άρθρωση.

Το αρχείο θα έχει το όνομα rrbot_control.xacro, θα αποθηκευτεί στο φάκελο urdf και είναι το ακόλουθο:

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

 <ros2_control name="GazeboSystem" type="system">
  <hardware>
   <plugin>gazebo_ros2_control/GazeboSystem</plugin>
  </hardware>

  <joint name="joint_base_mid">
   <command_interface name="position">
    <param name="min">-3.14</param>
    <param name="max">3.14</param>
   </command_interface>
   <state_interface name="position">
```

```
    <param name="initial_value">1.0</param>
  </state_interface>
  <state_interface name="velocity"/>
</joint>

<joint name="joint_mid_top">
  <command_interface name="position">
    <param name="min">-3.14</param>
    <param name="max">3.14</param>
  </command_interface>
  <state_interface name="position">
    <param name="initial_value">1.0</param>
  </state_interface>
  <state_interface name="velocity"/>
</joint>

<joint name="right_gripper_joint">
  <command_interface name="position">
    <param name="min">-0.548</param>
    <param name="max">0.0</param>
  </command_interface>
  <state_interface name="position">
    <param name="initial_value">1.0</param>
  </state_interface>
  <state_interface name="velocity"/>
</joint>

<joint name="left_gripper_joint">
  <command_interface name="position">
    <param name="min">-0.548</param>
    <param name="max">0.0</param>
  </command_interface>
  <state_interface name="position">
    <param name="initial_value">1.0</param>
  </state_interface>
  <state_interface name="velocity"/>
</joint>
  <transmission name="transmission1">
    <plugin>transmission_interface/SimpleTransmission</plugin>
    <actuator name="actuator1" role="actuator1"/>
    <joint name="joint_base_mid" role="joint_base_mid">
      <mechanical_reduction>2.0</mechanical_reduction>
      <offset>0.0</offset>
```

```
      </joint>
    </transmission>
    <transmission name="transmission2">
      <plugin>transmission_interface/SimpleTransmission</plugin>
      <actuator name="actuator2" role="actuator2"/>
      <joint name="joint_mid_top" role="joint_mid_top">
        <mechanical_reduction>4.0</mechanical_reduction>
        <offset>0.0</offset>
      </joint>
    </transmission>
    <transmission name="transmission3">
      <plugin>transmission_interface/SimpleTransmission</plugin>
      <actuator name="actuator1" role="actuator1"/>
      <joint name="left_gripper_joint" role="left_gripper_joint">
        <mechanical_reduction>2.0</mechanical_reduction>
        <offset>0.0</offset>
      </joint>
    </transmission>
    <transmission name="transmission4">
      <plugin>transmission_interface/SimpleTransmission</plugin>
      <actuator name="actuator2" role="actuator2"/>
      <joint name="right_gripper_joint" role="right_gripper_joint">
        <mechanical_reduction>4.0</mechanical_reduction>
        <offset>0.0</offset>
      </joint>
    </transmission>
  </ros2_control>
</robot>
```

Η επικοινωνία του Gazebo με το ROS 2 επιτυγχάνεται με την προσθήκη κάποιων νέων tags στο αρχείο urdf, τα οποία είναι τα Gazebo plugins. Όλα τα Gazebo plugins, έχουν αναπτυχθεί στο αρχείο rrbot.gazebo. Για καλύτερη οργάνωση του κώδικα, το plugin που είναι υπεύθυνο για την ενεργοποίηση του controller στο Gazebo, θα τοποθετηθεί σε εκείνο το αρχείο. Συνεπώς, το νέο rrbot2.gazebo είναι ίδιο με το προηγούμενο αρχείο με την παρακάτω προσθήκη:

```
  <!-- ros_control plugin -->
  <gazebo>
    <plugin filename="libgazebo_ros2_control.so" name="gazebo_ros2_control">
      <parameters>$(find rrbot)/config/rrbot_controllers.yaml</parameters>
    </plugin>
  </gazebo>
```

Στη συνέχεια δημιουργείται ένα καινούργιο ρομποτικό μοντέλο το rrbot5.xacro το οποίο είναι παρόμοιο με το προηγούμενο αλλά με την προσθήκη του αρχείου ορισμού των ελεγκτών. Η προσθήκη του αρχείου επιτυγχάνεται με τη τοποθέτηση της ακόλουθης εντολής.

```
<!-- Import ros control-->
<xacro:include filename="$(find rrbot)/urdf/rrbot_control.xacro"/>
```

### 3.8.3 Εκκίνηση των Controllers μέσω ενός launch file

Στην ενότητα αυτή θα αναπτυχθεί ένα νέο launch αρχείο, το οποίο θα εκκινεί τους ελεγκτές κάθε άρθρωσης. Πιο συγκεκριμένα, θα επιτευχθεί η τοποθέτηση του ρομπότ στο Gazebo και παράλληλα θα ενεργοποιηθούν οι Controllers του βραχίονα. Επιπρόσθετα, θα εκτελεστούν κατάλληλες εντολές τερματικού που θα επιτυγχάνουν την αλλαγή θέσης κάθε άρθρωσης. Παράλληλα θα χρησιμοποιηθεί το εργαλείο rqt με στόχο να δοθεί εντολή αλλαγής κάποιας άρθρωσης του βραχίονα.

Το παρακάτω launch αρχείο θα έχει το όνομα rrbot_control_gazebo.launch.py και θα αποθηκευτεί στον αντίστοιχο φάκελο. Το Python αρχείο είναι το ακόλουθο:

```
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch_ros.actions import Node
import os
import xacro

def generate_launch_description():
    pkg_name = 'rrbot'
    file_subpath = 'urdf/rrbot5.xacro'

    xacro_file = os.path.join(get_package_share_directory(pkg_name),file_subpath)
    robot_description_raw =xacro.process_file(xacro_file).toxml()
    gazebo_pkg = get_package_share_directory('gazebo_ros')

    robot_state_publisher_node = Node(
        package='robot_state_publisher', executable='robot_state_publisher',
        output="screen", parameters=[{'robot_description':robot_description_raw,
        'use_sim_time':True}])

    gazebo = IncludeLaunchDescription(
```

```
    PythonLaunchDescriptionSource([os.path.join( gazebo_pkg, 'launch'), '/gazebo.launch.py' ]) )

  spawn_entity = Node(
    package="gazebo_ros",  executable="spawn_entity.py",
    arguments=["-topic", "robot_description", "-entity", "my_robot_description"], output="screen",)

  joint_state_broadcaster_spawner = Node(
    package="controller_manager", executable="spawner",
    arguments=["joint_state_broadcaster", "--controller-manager", "/controller_manager"], )

  base_mid_controller_spawner = Node(
    package="controller_manager",   executable="spawner",
    arguments=["joint_base_mid_position_controller", "--controller-manager", "/controller_manager"], )

  mid_top_controller_spawner = Node(
        package="controller_manager", executable="spawner",
        arguments=["joint_mid_top_position_controller", "--controller-manager",
                   "/controller_manager"],)

  gripper_controller_spawner = Node(
        package="controller_manager", executable="spawner",
        arguments=["joint_gripper_controller", "--controller-manager", "/controller_manager"], )

  return LaunchDescription([
         robot_state_publisher_node,  gazebo,  spawn_entity,
         joint_state_broadcaster_spawner,  base_mid_controller_spawner,
        mid_top_controller_spawner,  gripper_controller_spawner])
```

Το παραπάνω Python αρχείο επιτυγχάνει τα ακόλουθα:

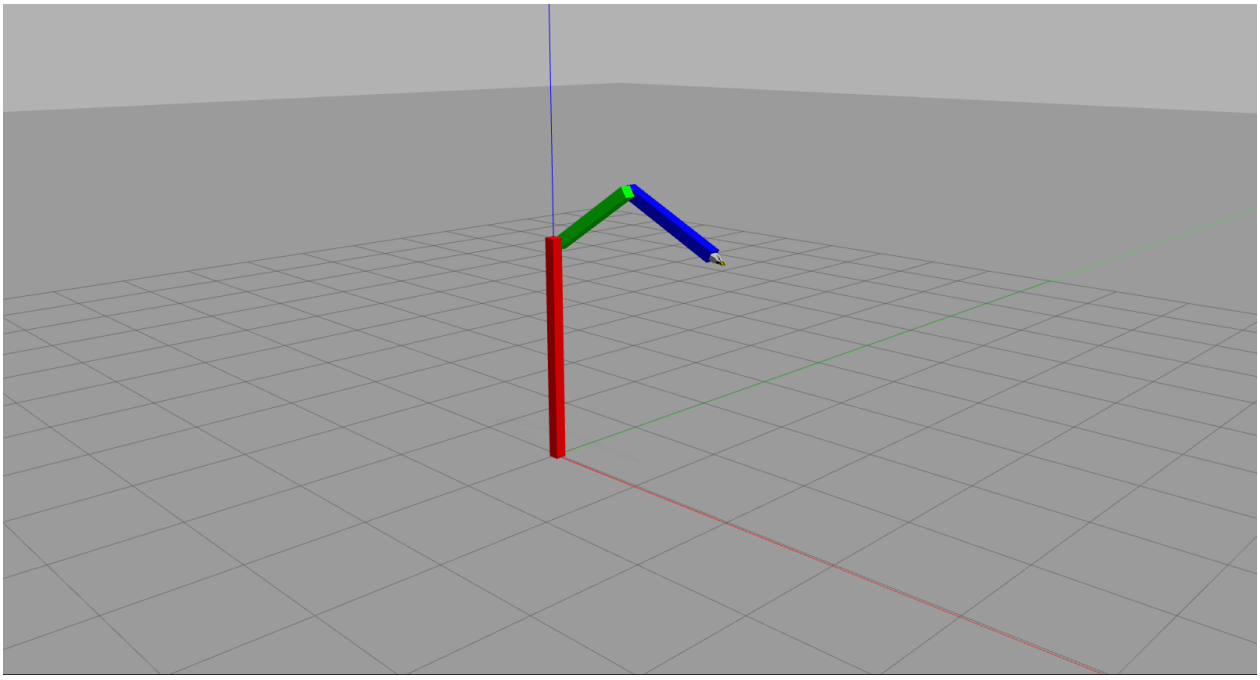
- Δημιουργία κάποιων αντικειμένων, τα οποία είναι χρήσιμα για τον εντοπισμό πακέτων και ρομποτικών μοντέλων.
- Δημιουργία του Node, robot state publisher, που είναι χρήσιμο καθώς δημοσιεύει πληροφορίες για την κατάσταση του ρομπότ.
- Χρήση της δυνατότητας PythonLaunchDescriptionSource για την εύρεση του εκτελέσιμου στο πακέτο gazebo_ros για την εκκίνηση της προσομοίωσης.
- Δημιουργία ενός Node για την τοποθέτηση του ρομποτικού μοντέλου στο Gazebo, μέσω του Spawn entity.

- Δημιουργία Nodes, που εκκινούν τους ελεγκτές του βραχίονα (πχ. base_mid_controller_spawner) αλλά και του state broadcaster.
- Χρήση της μεθόδου return της συνάρτησης μέσω της οποίας ορίζονται τα nodes ή actions που θα εκκινήσουν.

Έπειτα από τη δημιουργία του παραπάνω αρχείου, ακολουθεί η εκτέλεση του. Η διαδικασία που πρέπει να γίνει είναι συγκεκριμένη, καθώς πρέπει να χτιστεί το πακέτο και έπειτα να εκτελεστεί το launch αρχείο. Τα βήματα παρουσιάζονται στον παρακάτω πίνακα.

| |
|---|
| $ cd ~/ros2_ws |
| $ colcon build |
| $ **# For WSL, run the next command.** |
| $ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0 |
| $ ros2 launch rrbot rrbot_control_gazebo.launch.py |

Πίνακας 78. Ενεργοποίηση των Controllers του βραχίονα.



Εικόνα 38. Ο βραχίονας με τους ελεγκτές στο Gazebo.

Πλέον στο ρομποτικό μοντέλο είναι ενεργοποιημένοι οι ελεγκτές του βραχίονα. Συνεπώς, παρά το γεγονός ότι ασκούνται δυνάμεις βαρύτητας σε αυτό, οι αρθρώσεις του παραμένουν στη θέση που είναι χωρίς να μετακινούνται.

Στο σημείο αυτό, ενδιαφέρον έχει να εξεταστούν τα ενεργά nodes, topics και actions που υπάρχουν στο Ros δίκτυο. Για να γίνει αυτό θα πρέπει να εκτελεστούν κατάλληλες εντολές σε ένα νέο τερματικό. Ιδιαίτερη προσοχή στο γεγονός ότι θα πρέπει να υπάρχει ένα τερματικό, στο οποίο θα είναι ενεργοί οι ελεγκτές και το Gazebo. Εκτελούνται οι παρακάτω εντολές, για τη λήψη αποτελεσμάτων.

| |
|---|
| $ ros2 node list<br><br>$ ros2 topic list<br><br>$ ros2 action list |

Πίνακας 79. Ενεργά nodes, topics και actions.

```
katos@KATOS:~$ ros2 node list
/controller_manager
/gazebo
/gazebo_ros2_control
/joint_base_mid_position_controller
/joint_gripper_controller
/joint_mid_top_position_controller
/joint_state_broadcaster
/robot_state_publisher
katos@KATOS:~$ ros2 topic list
/clock
/dynamic_joint_states
/joint_base_mid_position_controller/controller_state
/joint_base_mid_position_controller/joint_trajectory
/joint_base_mid_position_controller/state
/joint_base_mid_position_controller/transition_event
/joint_gripper_controller/controller_state
/joint_gripper_controller/joint_trajectory
/joint_gripper_controller/state
/joint_gripper_controller/transition_event
/joint_mid_top_position_controller/controller_state
/joint_mid_top_position_controller/joint_trajectory
/joint_mid_top_position_controller/state
/joint_mid_top_position_controller/transition_event
/joint_state_broadcaster/transition_event
/joint_states
/parameter_events
/performance_metrics
/robot_description
/rosout
/tf
/tf_static
katos@KATOS:~$ ros2 action list
/joint_base_mid_position_controller/follow_joint_trajectory
/joint_gripper_controller/follow_joint_trajectory
/joint_mid_top_position_controller/follow_joint_trajectory
```

Εικόνα 39. Τα ενεργά nodes, topics και actions.

Παρατηρείται η εκκίνηση όλων των nodes που ορίστηκαν στο launch αρχείο. Επιπρόσθετα nodes είναι το controller manager που δημιουργήθηκε μέσω του yaml αρχείου αλλά και το gazebo_ros2_control το οποίο εκκινεί μέσω του αρχείου, rrbot_control.xacro. Παράλληλα με την εκκίνηση των nodes, παρατηρείται και η εκκίνηση των topics και actions. Με αυτό το τρόπο έχουν εκκινήσει όλα τα απαραίτητα attributes για την επικοινωνία του ρομποτικού συστήματος.

Αυτό που είναι αρκετά ενδιαφέρον και θα χρησιμοποιηθεί, είναι τα actions που είναι ενεργά στο δίκτυο του ROS. Με χρήση αυτών θα γίνονται οι δηλώσεις για επιθυμητή θέση των αρθρώσεων του βραχίονα. Συνεπώς για γίνει η αποστολή μηνυμάτων σε αυτά τα actions, πρέπει να υπάρχει η γνώση για τον τύπου μηνύματος που δέχεται κάθε actions. Έπειτα από το τύπου του μηνύματος, θα πρέπει να υπάρχει η γνώση για τα ορίσματα που δέχεται αυτό.

| |
|---|
| $ ros2 action  info /joint_base_mid_position_controller/follow_joint_trajectory<br><br>$ ros2 interface show control_msgs/action/FollowJointTrajectory |

Πίνακας 80. Εύρεση πληροφοριών για τα actions.

```
katos@KATOS:~$ ros2 action list -t
/joint_base_mid_position_controller/follow_joint_trajectory [control_msgs/action/FollowJointTrajectory]
/joint_gripper_controller/follow_joint_trajectory [control_msgs/action/FollowJointTrajectory]
/joint_mid_top_position_controller/follow_joint_trajectory [control_msgs/action/FollowJointTrajectory]
katos@KATOS:~$ ros2 action  info /joint_base_mid_position_controller/follow_joint_trajectory
Action: /joint_base_mid_position_controller/follow_joint_trajectory
Action clients: 0
Action servers: 1
    /joint_base_mid_position_controller
katos@KATOS:~$ ros2 interface show control_msgs/action/FollowJointTrajectory
# The trajectory for all revolute, continuous or prismatic joints
trajectory_msgs/JointTrajectory trajectory
        std_msgs/Header header
                builtin_interfaces/Time stamp
                        int32 sec
                        uint32 nanosec
                string frame_id
        string[] joint_names
        JointTrajectoryPoint[] points
                float64[] positions
                float64[] velocities
                float64[] accelerations
                float64[] effort
                builtin_interfaces/Duration time_from_start
                        int32 sec
                        uint32 nanosec
# The trajectory for all planar or floating joints (i.e. individual joints with more than one DOF)
trajectory_msgs/MultiDOFJointTrajectory multi_dof_trajectory
        std_msgs/Header header
                builtin_interfaces/Time stamp
```

Εικόνα 40. Πληροφορίες για τα actions.

Με τα παραπάνω παρατηρείται ότι και για τα τρία actions, υπάρχει ένας συγκεκριμένος τύπος μηνύματος, ο οποίος είναι το control_msgs/action/FollowJointTrajectory. Η δεύτερη εντολή χρησιμοποιείται για να δώσει πληροφορίες για το action και πιο συγκεκριμένα τα nodes που δέχεται ή στέλνει πληροφορία. Τέλος, το action δέχεται αρκετά ορίσματα, εκ των οποίων ελάχιστα θα είναι αυτά που θα χρησιμοποιηθούν.

Σε αυτό το σημείο και γνωρίζοντας τον τύπου μηνύματος που δέχεται το κάθε action, μπορεί να γίνει η δήλωση για επιθυμητή θέση της άρθρωσης του βραχίονα. Για να επιτευχθεί αυτό, θα πρέπει σε ένα τερματικό να είναι ενεργό το launch αρχείο που έχει αναπτυχθεί παραπάνω. Έπειτα εκτελείται η ακόλουθη εντολή σε ένα νέο τερματικό.

```
$ ros2 action send_goal /joint_base_mid_position_controller/follow_joint_trajectory
control_msgs/action/FollowJointTrajectory -f "{trajectory: { joint_names: [joint_base_mid],

  points: [ { positions: [-2.9], time_from_start: { sec: 2 } }, ] }}"
```

Πίνακας 81. Δήλωση επιθυμητής θέσης για το joint_base_mid.

```
joint_names:
- joint_base_mid
desired:
  positions:
  - 2.8525001408511033
  velocities:
  - 0.9499971829779355
  accelerations:
  - 0.0
  effort: []
  time_from_start:
    sec: 0
    nanosec: 0
actual:
  positions:
  - 2.8430047739296 78
  velocities:
  - 0.004604908353255 94
  accelerations: []
  effort: []
  time_from_start:
    sec: 0
    nanosec: 0
error:
  positions:
  - 0.009495366921425052
  velocities:
  - 0.0
  accelerations: []
```

Εικόνα 41. Αποτελέσματα της εντολής επιθυμητής θέσης.

Με χρήση της παραπάνω εντολής, στέλνετε ένα μήνυμα στο επιθυμητό action. Πιο συγκεκριμένα, στέλνετε μήνυμα στο actions, /joint_base_mid_position_controller να λάβει τη θέση -2.9. Παράλληλα, δηλώνεται και ο συνολικός χρόνος που είναι επιθυμητό να διαρκέσει η διεργασία. Έπειτα από την εκτέλεση της η εντολή, παρέχεται διαρκής ενημέρωση της θέσης και της ταχύτητας της άρθρωσης, μέχρι αυτή να λάβει την τελική της θέση. Τα αποτελέσματα, της εντολής φαίνονται στην παραπάνω εικόνα.

Παρατηρείται ότι η τελική θέση της άρθρωσης του βραχίονα δεν είναι αυτή που ζητήθηκε, ωστόσο η αυτή είναι πολύ κοντά στην επιθυμητή. Επίσης παρατηρείται και το λάθος που έχει προκύψει, το οποίο είναι αρκετά μικρό. Συνεπώς, προκύπτει το συμπέρασμα ότι ο PID Controller που έχει δημιουργηθεί είναι αρκετά αποδοτικός, ωστόσο υπάρχουν περιθώρια βελτίωση του.

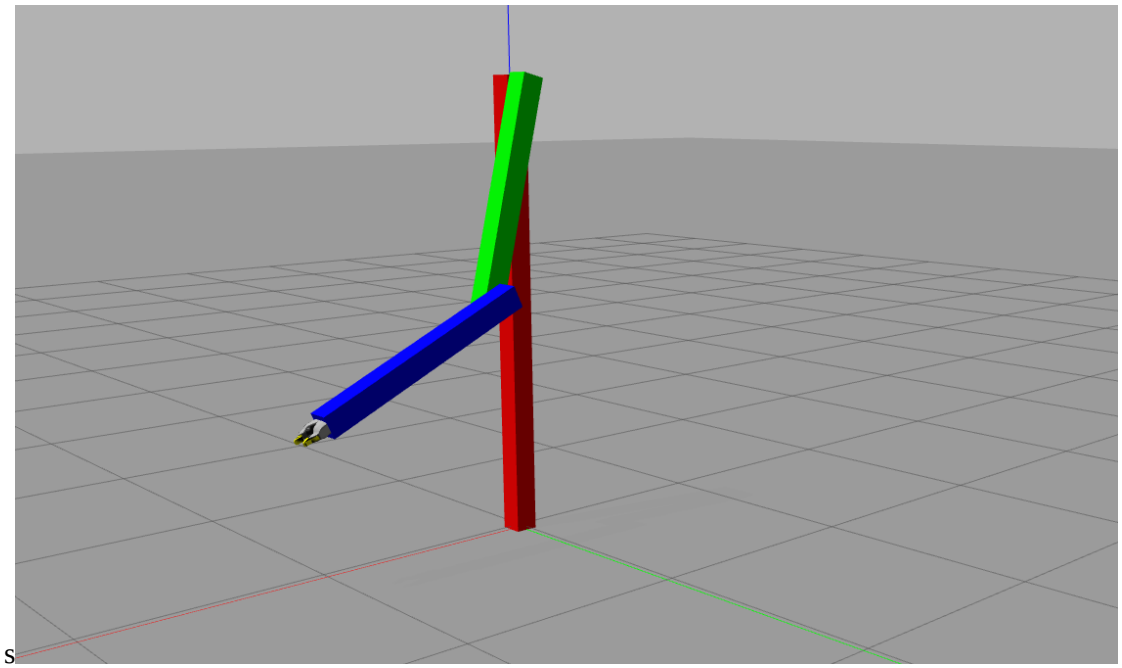
Παράλληλα με τη δήλωση της επιθυμητής θέσης για τη μια άρθρωση του βραχίονα, μπορούν να δοθούν κατάλληλες εντολές για την αλλαγή θέσης της δεύτερη άρθρωσης αλλά και του gripper.

```
$  ros2 action send_goal /joint_mid_top_position_controller/follow_joint_trajectory
control_msgs/action/FollowJointTrajectory -f "{trajectory: { joint_names: [joint_mid_top],

   points: [ { positions: [-0.9], time_from_start: { sec: 2 } }, ] }}"

$ ros2 action send_goal /joint_gripper_controller/follow_joint_trajectory
control_msgs/action/FollowJointTrajectory -f "{ trajectory: { joint_names: [right_gripper_joint,
left_gripper_joint],

   points: [ { positions: [-0.3, -0.4], time_from_start: { sec: 2 } }, ]  }}"
```

Πίνακας 82. Επιθυμητή θέση δεύτερη άρθρωσης και αρπάγης.

Όπως και για την πρώτη άρθρωση, έτσι και για τη δεύτερη η εντολή είναι παρόμοια. Η εντολή που δίνεται για την αρπάγη διαφέρει ελάχιστα σε σχέση με τις άλλες δύο. Παράλληλα, οι εντολές αυτές ενημερώνουν διαρκώς για την τωρινή θέση και ταχύτητα τους. Στο τέλος, μπορεί να γίνει ένας έλεγχος για τον PID ελεγκτή και αν αυτός είναι αποδοτικό.

Παράλληλα οι αλλαγές στις αρθρώσεις του βραχίονα αλλά και της αρπάγης επιτυγχάνονται και τα αποτελέσματα τους είναι ορατά στο περιβάλλον προσομοίωσης Gazebo. Αυτά φαίνονται στην ακόλουθη εικόνα.



Εικόνα 42. Ο βραχίονας με τις επιθυμητές θέσεις που δόθηκαν.

Κλείνοντας, άξιο αναφοράς είναι το γεγονός ότι το πακέτο ros2_control προσφέρει και πληθώρα command line εντολών. Αυτές οι εντολές ποικίλουν και άλλοτε προσφέρουν χρήσιμες πληροφορίες για τους ενεργούς ελεγκτές και άλλες φορές δίνεται η δυνατότητα αλληλεπίδρασης με το σύστημα. Χρήσιμες εντολές είναι οι παρακάτω:

| |
|---|
| $ ros2 control list_controllers<br><br>$ ros2 control list_hardware_interfaces |

Πίνακας 83. Χρήσιμες εντολές του πακέτου ros2_control.

Επίσης προσφέρονται εντολές για την αλλαγή του τύπου ενός ελεγκτή. Για να πραγματοποιηθεί αυτό, θα πρέπει νωρίτερα να έχει οριστεί αυτός μέσα στον κώδικα που έχει αναπτυχθεί. Έπειτα με κατάλληλες εντολές να γίνει η αλλαγή του τύπου. Πολλές πληροφορίες για την αλλαγή του τύπου ενός controller αλλά και άλλες ενδιαφέροντες επιλογές του πακέτου μπορούν να βρεθούν στον ιστότοπο του πακέτου και μέσω των demos που προσφέρονται ελεύθερα.

### 3.8.4 Η αλληλεπίδραση με το Rviz

Το Rviz είναι το τρισδιάστατο μοντέλο οπτικοποίησης δεδομένων του ROS 2. Μέσω αυτού είναι εφικτό να ελεγχθεί τι αισθάνεται και τι βλέπει το ρομπότ. Εν ολίγοις, δίνει τη δυνατότητα να παρατηρούνται τα

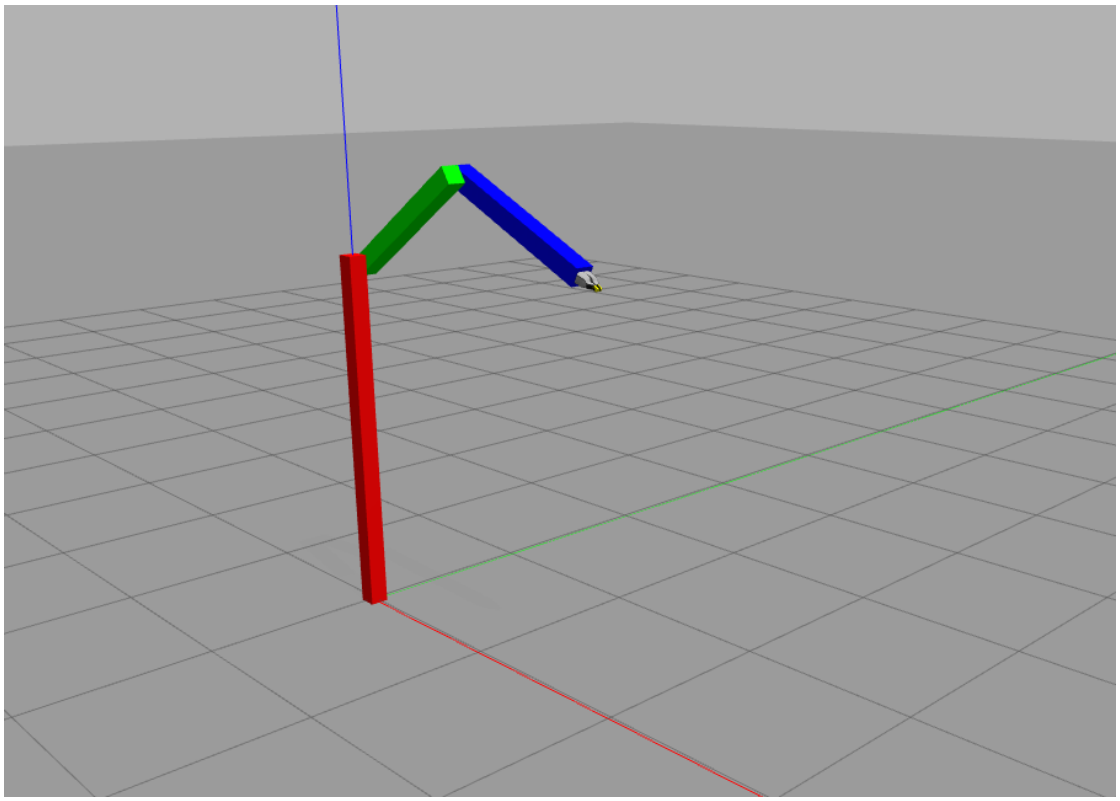
δεδομένα που αντλούνται από τους αισθητήρες του ρομπότ. Παρά το γεγονός ότι το rrbot δεν διαθέτει αισθητήρες, είναι χρήσιμο να γίνει η αξιοποίηση του.

Για την τοποθέτηση του ρομπότ στο Rviz και την άμεση επικοινωνία με το Gazebo πρέπει να είναι ενεργό το Gazebo και οι ελεγκτές σε ένα τερματικό. Έπειτα μπορεί να πραγματοποιηθεί η εκκίνηση του Rviz με την παρακάτω εντολή:

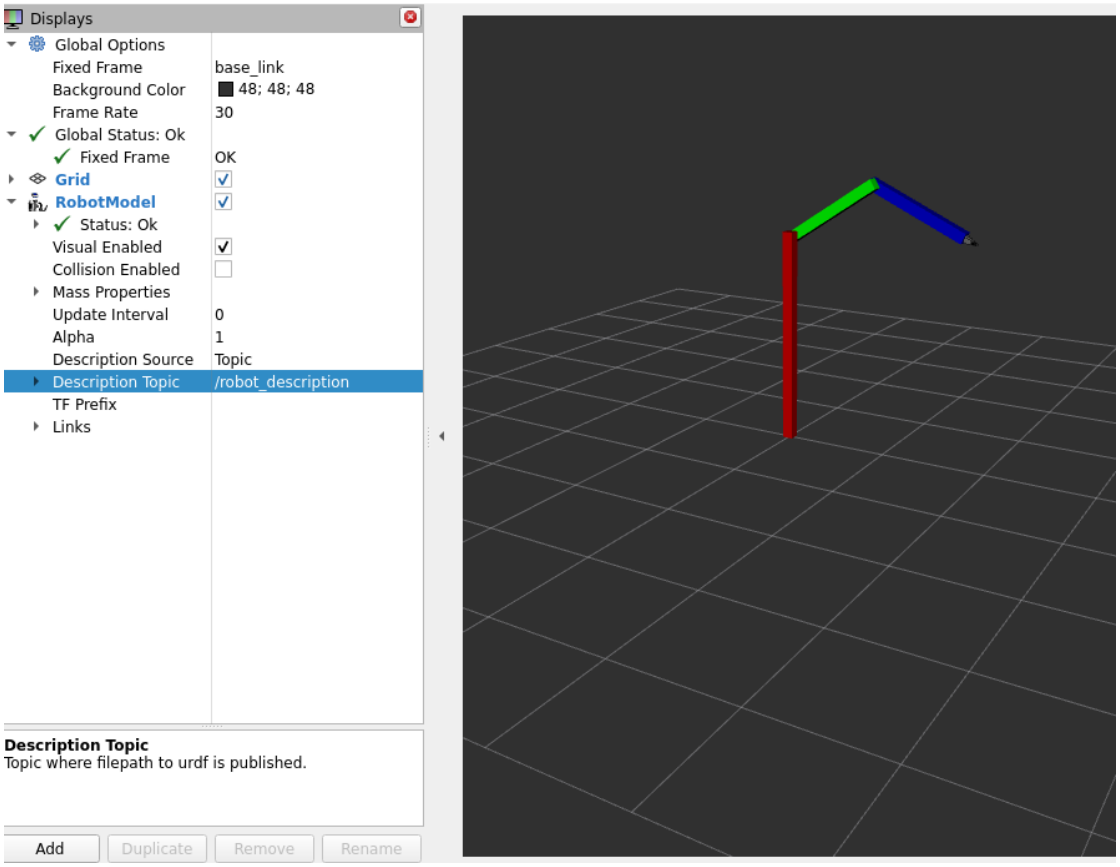
```
$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0 # for WSL
$ rviz2
```

Πίνακας 84. Εκκίνηση του Rviz.

Αφού εκκίνηση το Rviz, τότε θα προστεθεί το ρομποτικό μοντέλο στον τρισδιάστατο χώρο του γραφικού περιβάλλοντος. Αυτό επιτυγχάνεται μέσω του **Displays Panel**, επιλέγοντας το **Add** και έπειτα το **RobotModel**. Στη συνέχεια, πρέπει να επιλεχθεί το κατάλληλο topic και να αλλάξει η επιλογή στο **Fixed Frame** σε base_link. Έπειτα από αυτές τις αλλαγές, παρατηρείται ότι η θέση του βραχίονα στο Rviz είναι ακριβώς η ίδια με αυτή του Gazebo.



Εικόνα 43. Το ρομποτικό μοντέλο στο Gazebo.



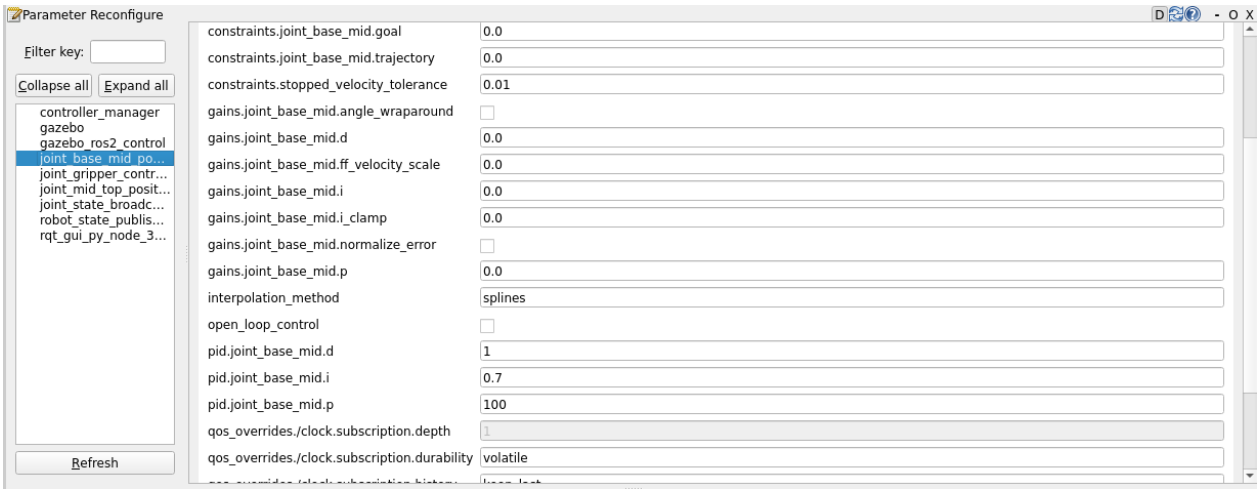

Εικόνα 44. Το ρομποτικό μοντέλο στο Rviz.

Αφού το ρομπότ εισαχθεί στο περιβάλλον οπτικοποίησης, τότε οι θέσεις των αρθρώσεων είναι παρόμοιες με αυτές της προσομοίωσης. Σε περίπτωση αλλαγής της θέση των αρθρώσεων στην προσομοίωση τότε η μετακίνηση θα παρατηρηθεί και στο Rviz.

### 3.8.5 Το εργαλείο rqt

Το rqt είναι ένα πολυχρηστικό εργαλείο με τη χρήση του οποίου μπορούν να πραγματοποιηθούν διάφορες διεργασίες και αλλαγές στο σύστημα. Το rqt παρέχει ένα γραφικό περιβάλλον μέσα από το οποίο μπορούν να γίνουν διεργασίες που θα πραγματοποιούνταν μέσω εντολών command line. Συνεπώς είναι αρκετά χρήσιμο και μπορεί να βοηθήσει στη βελτίωση του ρομποτικού μοντέλου και τις λειτουργίες του.

Αφού εκκινήσει το rqt, επιλέγεται η καρτέλα **Plugins**>> **Configuration** >>**Dynamic Reconfigure**. Οπότε και θα υπάρχει το παρακάτω γραφικό περιβάλλον.




Εικόνα 45**.** το γραφικό περιβάλλον του Dynamic Reconfigure.

Το Dynamic Reconfigure είναι ένα πολύ σπουδαίο εργαλείο του rqt. Μέσω αυτού μπορούν να πραγματοποιηθούν μεταβολές στους ελεγκτές του βραχίονα. Οι τιμές μέσω των οποίων μπορεί να γίνει η αλλαγή είναι το pid.joint_base_mid.d και αντιστοίχως οι καταλήξεις σε i και d. Τα action είναι ιδιαίτερα χρήσιμα καθώς προσφέρουν feedback κατά τη διαδικασία εκπλήρωσης της αποστολής τους αλλά και στο τέλος. Από την εικόνα 41, παρατηρείται ότι το σφάλμα που υπάρχει όσον αφορά την επιθυμητή και την πραγματική θέση είναι πολύ μικρή. Αυτό έχει σαν αποτέλεσμα ο ελεγκτής να είναι αποδοτικός.

Σε άλλες περιπτώσεις οι παράμετροι του ελεγκτή δεν θα έχουν ρυθμιστεί σωστά. Μέσω του Dynamic Reconfigure μπορεί να γίνει η μεταβολή των παραμέτρων αυτού, με στόχο τη μείωση του λάθους. Σε περίπτωση αλλαγής των παραμέτρων, τότε μπορούν να εξαχθούν οι νέοι παράμετροι σε ένα νέο yaml file. Έπειτα μπορεί να γίνει η κλήση των συγκεκριμένων παραμέτρων.

## Κεφάλαιο 4

## Έλεγχος του δίτροχου ρομπότ

Το κεφάλαιο αυτό είναι αφιερωμένο στη δημιουργία ενός ρομπότ διαφορικής κίνησης με στόχο την αποφυγή εμποδίων σε ένα περιβάλλον προσομοίωσης. Θα κατασκευαστεί ένα ρομπότ διαφορικής κίνησης χρησιμοποιώντας κάποιες μεθόδους xacro και στο οποίο θα προστεθούν δύο αισθητήρες. Πιο συγκεκριμένα, θα τοποθετηθεί μια κάμερα και ένα lidar με στόχο, τη μέτρηση αποστάσεων κάποιων εμποδίων από το ρομπότ. Έπειτα θα προστεθεί ένα plugin διαφορικής κίνησης για το ρομπότ και θα αναπτυχθεί ένα Python Script, με στόχο την μέτρηση αποστάσεων και την αποφυγή εμποδίων. Στο τέλος του κεφαλαίου θα δοκιμαστεί ο αλγόριθμος στο περιβάλλον προσομοίωσης.

### 4.1 Το δίτροχο ρομπότ

Για ρομπότ διαφορική κίνησης θα δημιουργηθεί ένα νέο πακέτο και θα αναπτυχθεί ένα urdf αρχείο από την αρχή. Το urdf θα είναι ένα δίτροχο ρομπότ, παρόμοιο με αυτό του δεύτερου κεφαλαίου, ωστόσο θα χρησιμοποιηθούν μέθοδοι Xacro, με στόχο να υλοποιηθούν κάποιες διεργασίες ευκολότερα. Έπειτα θα προστεθεί το ρομποτικό μοντέλο στο Rviz και στο Gazebo.

Αρχικά θα πρέπει να δημιουργηθεί ένα νέο πακέτο:

```
$ cd ~/ros2_ws/src

$ ros2 pkg create --build-type ament_python my_robot
```

Πίνακας 85. Δημιουργία πακέτου my_robot.

Αφού δημιουργηθεί το πακέτο, είναι αναγκαίο να δημιουργηθούν φάκελοι οι οποίοι είναι χρήσιμοι για την καλύτερη οργάνωση του κώδικα που θα δημιουργηθούν στο πακέτο.

```
$ cd ~/ros2_ws/src

$ mkdir launch urdf meshes worlds
```

Πίνακας 86. Δημιουργία φακέλων εντός του πακέτου.

Έπειτα από τη δημιουργία των φακέλων, θα πρέπει να ενημερωθεί το αρχείο setup.py, ώστε να μπορεί να γίνει install ο κώδικας που θα δημιουργηθεί. Στο setup.py θα πρέπει να προστεθούν δύο βιβλιοθήκες.

```
$ from glob import glob

$ import os
```

Πίνακας 88. Εισαγωγή βιβλιοθηκών στο setup.py.

Έπειτα θα πρέπει να ενημερωθεί η λίστα data_files και να προστεθούν τα ακόλουθα:

```
    (os.path.join('share',package_name,'launch'),
     glob(os.path.join('launch','*.launch.py'))),
    (os.path.join('share',package_name,'urdf'),
     glob(os.path.join('urdf','*.xacro'))),
     (os.path.join('share',package_name,'urdf'),
     glob(os.path.join('urdf','*.gazebo'))),
    (os.path.join('share',package_name,'worlds'),
     glob(os.path.join('worlds','*.world'))),
    (os.path.join('share',package_name,'meshes'),
     glob(os.path.join('meshes','*.dae'))),
```

Πίνακας 87. Προσθήκες στη λίστα data_files του setup.py.

Αφού ολοκληρωθούν οι παραπάνω διαδικασίες τότε το πακέτο πρέπει να χτιστεί και στη συνέχεια είναι έτοιμο για χρήση και συνεπώς μπορεί να αρχίσει η δημιουργία κώδικα.

Το δίτροχο ρομπότ είναι παρόμοιο με του κεφαλαίου 2, με διαφορά στις διαστάσεις που έχει αυτό. Το ρομποτικό μοντέλο θα ονομαστεί my_robot.xacro, θα αποθηκευτεί στο φάκελο urdf του πακέτου και μπορεί βρεθεί στον σύνδεσμο https://github.com/DimitrisKatos/my_robot/blob/master/urdf/myrobot.xacro.

Για την τοποθέτηση του ρομπότ στο περιβάλλον οπτικοποίησης δεδομένων (Rviz) και στο περιβάλλον προσομοίωσης (Gazebo), χρησιμοποιούνται τα αρχεία launch που έχουν δημιουργηθεί στα προηγούμενα δύο κεφάλαια. Οι τροποποιήσεις που πρέπει να πραγματοποιηθούν στους κώδικες εκείνους είναι η αλλαγή στο όνομα του πακέτου και του ρομποτικού μοντέλου.

Επειδή τα δεδομένα που παράγονται από την προσομοίωση πρέπει να οπτικοποιούνται στο περιβάλλον του Rviz, πρέπει τα δύο περιβάλλοντα να επικοινωνούν μεταξύ τους. Για αυτό πρέπει, να εκκινείτε το Gazebo και έπειτα το Rviz είτε να δημιουργηθεί ένα launch file που εκκινεί και τα δύο περιβάλλοντα ταυτόχρονα.

### 4.2 Προσθήκη κάμερας

Κάθε αυτόνομο ρομποτικό σύστημα, για να μπορέσει να πλοηγηθεί στο χώρο εργασίας του πρέπει να φέρει αισθητήρες μέσω των οποίων θα αντιλαμβάνεται το περιβάλλον του. Οι κάμερες είναι ένας από τους βασικότερους αισθητήρες που μπορεί να φέρει ένα ρομπότ, καθώς μέσω αυτού μπορούν να εξαχθούν αρκετά συμπεράσματα για το χώρο εργασίας του.

Για την τοποθέτηση της κάμερας στο ρομπότ θα χρειαστεί η προσθήκη ενός νέου link στο urdf και έπειτα η δημιουργία ενός νέου joint μεταξύ της κάμερας και του σασί. Στο ρομπότ διαφορικής οδήγησης έχουν οριστεί δύο xacro properties μέσω των οποίων ορίζονται δύο σταθερές που χρησιμοποιούνται για την δημιουργία του link της κάμερας. Έπειτα ακολουθεί η δημιουργία ενός joint μεταξύ του chassis και της κάμερας.

Για την επικοινωνία του ROS με το Gazebo πρέπει να προστεθούν κάποια νέα tags που ονομάζονται Gazebo plugins. Στα προηγούμενα κεφάλαια έχουν χρησιμοποιηθεί κάποια Gazebo plugins για την προσθήκη χρωμάτων και τριβής στο ρομποτικό σύστημα. Για την ενεργοποίηση της κάμερας, θα χρησιμοποιηθεί ένα επιπρόσθετο Gazebo plugin το οποίο θα εκκινεί τη κάμερα στο περιβάλλον προσομοίωσης.

Με χρήση των Gazebo plugins μπορούν να προστεθούν αρκετοί αισθητήρες στο περιβάλλον προσομοίωσής. Τα Gazebo plugins μπορούν να βρεθούν στον παρακάτω σύνδεσμο και με κατάλληλες τροποποιήσεις να χρησιμοποιηθούν: https://classic.gazebosim.org/tutorials?tut=ros_gzplugins.

Θα δημιουργηθεί ένα νέο αρχείο, στο οποίο θα ορίζονται όλα τα tags που αφορούν το Gazebo και είναι απαραίτητα για την προσομοίωση του ρομπότ διαφορικής οδήγησης. Το αρχείο αυτό θα ονομαστεί robot.gazebo και θα αποθηκευτεί στο φάκελο URDF. Στο αρχείο αυτό θα προστεθούν τα χρώματα και οι τριβές για κάθε link. Επιπρόσθετα θα προστεθεί και το Gazebo plugin για την ενεργοποίηση της κάμερας.

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

  <!-- Base Link -->
  <gazebo reference="chassis">
    <material>Gazebo/Green</material>
  </gazebo>

  <!-- CASTER-->
  <gazebo reference="caster">
    <mu1>0.2</mu1>
    <mu2>0.2</mu2>
```

```
    <material>Gazebo/Grey</material>
  </gazebo>

  <!--Right wheel-->
  <gazebo reference="right_wheel">
    <mu1>0.2</mu1>
    <mu2>0.2</mu2>
    <material> Gazebo/Grey</material>
  </gazebo>

  <!--Left wheel-->
  <gazebo reference="left_wheel">
    <mu1>0.2</mu1>
    <mu2>0.2</mu2>
    <material>Gazebo/Grey</material>
  </gazebo>

  <!-- Camera -->
  <gazebo reference ="camera">
    <mu1>0.1</mu1>
    <mu2>0.1</mu2>
    <material>Gazebo/White</material>
  </gazebo>

<!-- camera -->
<gazebo reference="camera">
  <sensor type="camera" name="camera1">
    <update_rate>30.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
```

```
      <cameraName>my_robot/camera1</cameraName>
      <imageTopicName>image_raw</imageTopicName>
      <cameraInfoTopicName>camera_info</cameraInfoTopicName>
      <frameName>camera_link</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
 </gazebo>
</robot>
```

Στο παραπάνω κείμενο παρατηρείται το Gazebo plugin για την ενεργοποίηση της κάμερας. Σε αυτό καθορίζονται αρκετοί παράμετροι της κάμερας όπως είναι το μέγεθος και το εύρος αυτής. Παράλληλα χρησιμοποιούνται και κάποια tags που είναι απαραίτητα για την λειτουργία της κάμερας. Το σημαντικότερο tag του plugin είναι και το πρώτο καθώς εκεί δηλώνεται το link που θα αντιπροσωπεύει η κάμερα. Σε περίπτωση προσθήκης δεύτερης κάμερας τότε πρέπει να γίνει κατάλληλη αλλαγή στο όνομα του topic που θα έχει η δεύτερη κάμερα.

Έπειτα από τη δημιουργία του παραπάνω αρχείου, αυτό θα πρέπει να εισαχθεί στο βασικό URDF με χρήση της μεθόδου xacro:include. Συνεπώς στο my_robot.xacro, προστίθεται το ακόλουθο:

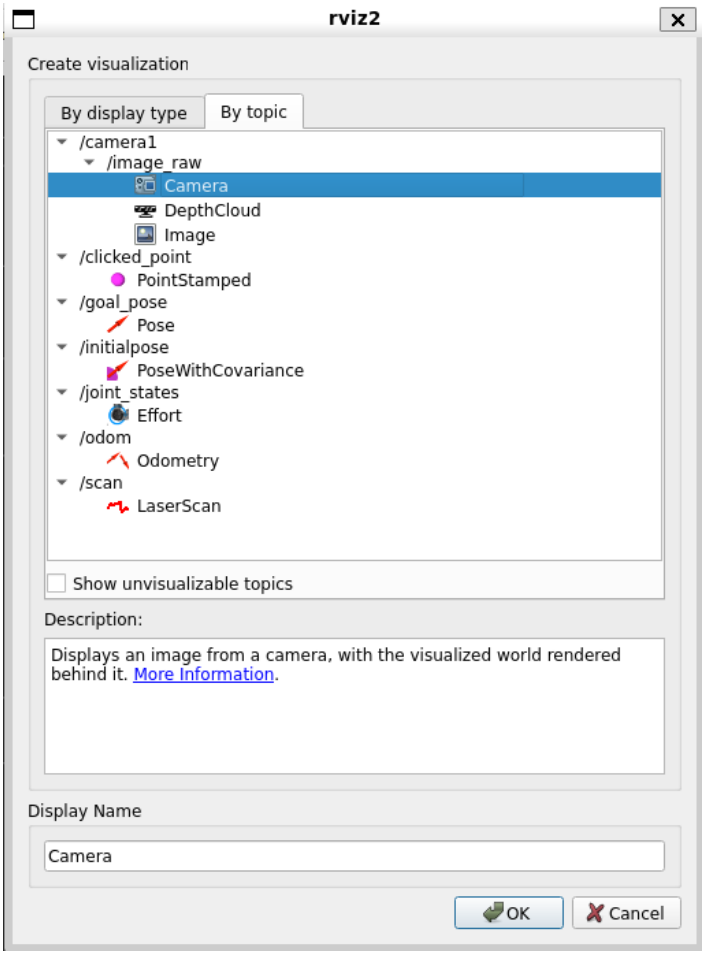
```
<xacro:include filename="$(find my_robot)/urdf/robot.gazebo" />
```

Έπειτα εκκινεί το launch file για την τοποθέτηση του ρομπότ στο Gazebo. Το launch file για την εκκίνηση του Gazebo είναι το my_robot_gazebo.launch.py.

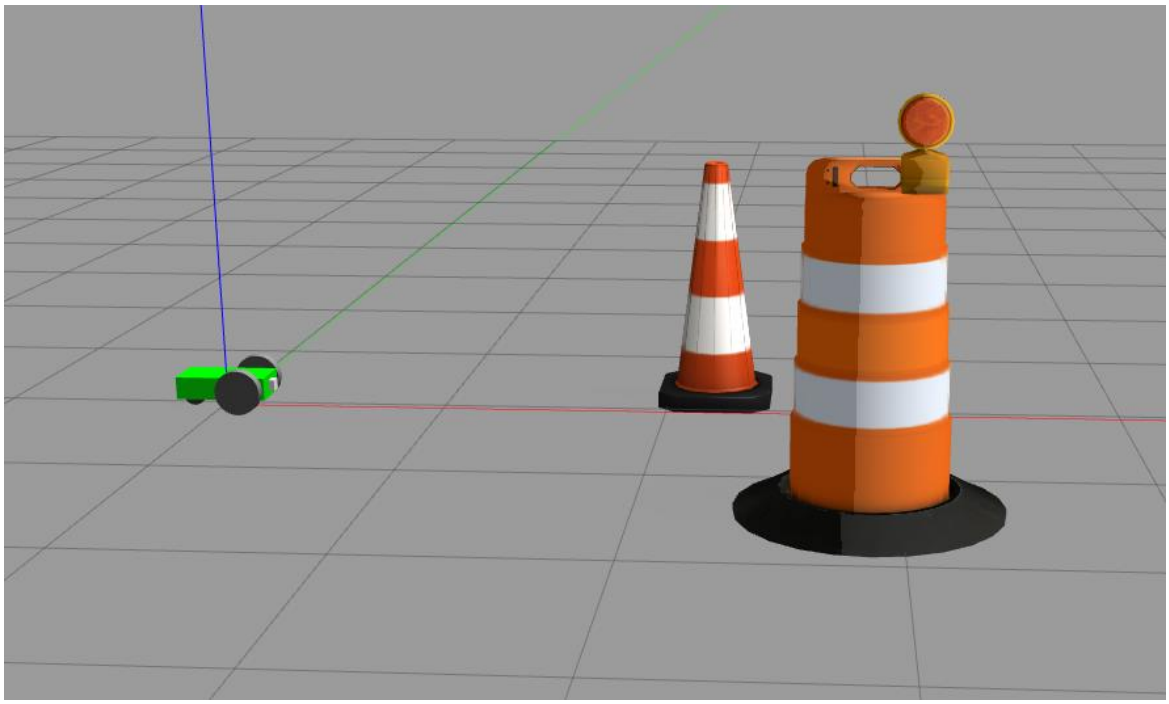
```
$ cd ~/ros2_ws

$ colcon build --package-select my_robot

$ cd src/my_robot # For WSL run the next command

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch my_robot my_robot_gazebo.launch.py
```

Πίνακας 88. Χρήση της κάμερας στο Gazebo.

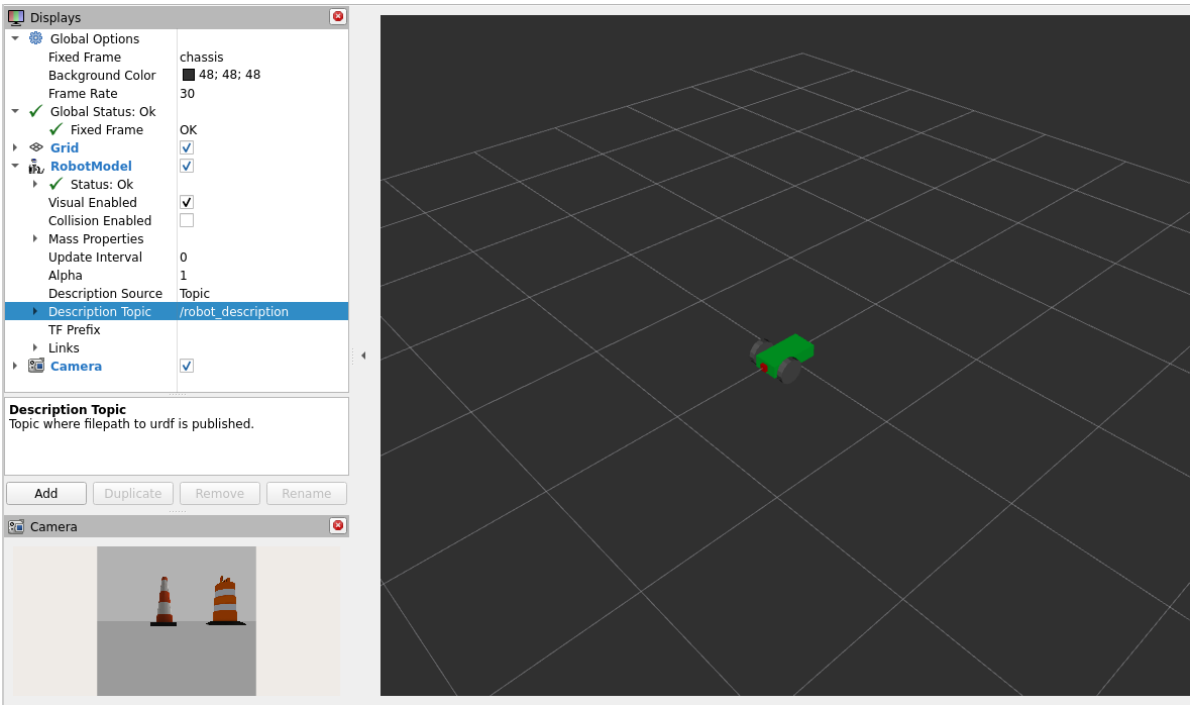
Με το παραπάνω αρχείο θα εκκινήσει το Gazebo και θα δημοσιευτούν τα πληροφορίες για το ρομπότ στο /robot_description topic. Επειδή είναι επιθυμητός ο συγχρονισμός του Gazebo με το Rviz, το περιβάλλον οπτικοποίησης θα εκκινήσει σε ένα νέο τερματικό. Σε περίπτωση που το ROS εκτελείται στο περιβάλλον του WSL, θα χρειαστεί επιπρόσθετη εντολή που παρουσιάζεται και στον παραπάνω πίνακα. Στη συνέχεια, προστίθεται το **RobotModel** από το **Displays Panel**, ακολουθεί η αλλαγή στο topic του και τέλος γίνεται η αλλαγή στο **Fixed_frame**. Στην παρακάτω εικόνα, παρουσιάζεται ο τρόπος με τον οποίο θα προστεθεί η κάμερα μέσω του Displays Panel του Rviz.




Εικόνα 46. Προσθήκη της κάμερας στο Rviz.



Εικόνα 47. Το my_robot στο Gazebo με ένα κώνο.




Εικόνα 48. Τα αποτελέσματα από την κάμερα του ρομπότ.

Αφού προστεθεί η κάμερα στο Rviz, ακολουθεί η προσθήκη ενός αντικειμένου στο περιβάλλον προσομοίωσης. Έπειτα στο Displays Panel του Rviz, παρατηρείται ότι έχει εμφανιστεί η κάμερα του ρομπότ και τι βλέπει αυτό. Συνεπώς ένας από τους σημαντικότερους αισθητήρες που μπορεί να φέρει ένα ρομπότ, έχει τοποθετηθεί κατά τον επιθυμητό τρόπο στην προσομοίωση.

### 4.3 Προσθήκη ενός lidar

Η κάμερα είναι ένας πολύ σημαντικός αισθητήρας για την εξαγωγή συμπερασμάτων για το περιβάλλον ενός ρομπότ. Ωστόσο πολλές φορές με την κάμερα, δεν μπορούν να εξαχθούν δεδομένα που αφορούν την απόσταση που απέχει το ρομπότ από ένα αντικείμενο ή εμπόδιο. Για να μπορέσει να γίνει η μέτρηση αποστάσεων μέσω κάμερας, θα πρέπει αυτή να είναι μια στερεοσκοπική κάμερα. Τις περισσότερες φορές σε ένα αυτόνομο ρομποτικό σύστημα, χρησιμοποιούνται αποστασιόμετρα για τις μετρήσεις αποστάσεων ενός εμποδίου.

Τα δύο συνηθέστερα αποστασιόμετρα που χρησιμοποιούνται σε αυτόνομα ρομποτικά συστήματα είναι τα lidar και τα αποστασιόμετρα υπερήχων. Τα τελευταία είναι αρκετά οικονομικά και χρησιμοποιούνται σε ρομποτικές εφαρμογές χαμηλού κόστους. Επίσης τα αποτελέσματα του αισθητήρα αρκετές φορές δεν είναι ακριβή καθώς μπορούν να επηρεαστούν από το περιβάλλον τους. Αντίθετα τα lidar έχουν πολύ μεγαλύτερη ακρίβεια στα αποτελέσματα τους και έχουν το πλεονέκτημα ότι μπορούν να μετρήσουν αποστάσεις σε όλο το επίπεδο σε σύγκριση με τα άλλα αποστασιόμετρα που μετρούν αποστάσεις μόνο προς κάποια κατεύθυνση.

Στο ρομποτικό μοντέλο θα γίνει η προσθήκη ενός lidar με στόχο την μέτρηση αποστάσεων από το ρομπότ. Για να επιτευχθεί αυτό θα προστεθεί ένα νέο tag στο URDF, το οποίο θα είναι ένα mesh file και θα αναπαριστά ένα πραγματικό lidar. Έπειτα θα προστεθεί ένα νέο gazebo plugin για την ενεργοποίηση του lidar στο Gazebo.

Πιο συγκεκριμένα, θα πρέπει να προστεθεί το link για το lidar στο urdf αρχείο. Συνεπώς στο my_robot.xacro θα προστεθεί το ακόλουθο:

```
<joint name="hokuyo_joint" type="fixed">
  <axis xyz="0 1 0" />
  <origin xyz=".15 0 .1" rpy="0 0 0"/>
  <parent link="base_link"/>
  <child link="hokuyo"/>
</joint>
```

```
<!-- Hokuyo Laser -->
  <link name="hokuyo">
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
    <box size="0.1 0.1 0.1"/>
      </geometry>
    </collision>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
          <mesh filename="file://$(find my_robot)/meshes/hokuyo.dae"/>
      </geometry>
    </visual>
    <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>
  </link>
```

Το παραπάνω αναπαριστά ένα νέο link, το οποίο στο visual tag έχει ένα mesh αρχείο. Το collision tag δεν έχει την ίδια γεωμετρία με αυτή του mesh file, αλλά ενός κύβου. Αυτό γίνεται για την αποφυγή κατανάλωσης πόρων του υπολογιστή κατά την προσομοίωση του ρομπότ. Έπειτα πραγματοποιείται το fixed joint του lidar με το base_link. Το mesh file μπορεί να βρεθεί στον ακόλουθο σύνδεσμο https://github.com/DimitrisKatos/my_robot/blob/master/meshes/hokuyo.dae και θα αποθηκευτεί στο φάκελο meshes με το όνομα hokuyo.dae.

Έπειτα από την προσθήκη του παραπάνω link, πρέπει να γίνει η προσθήκη ενός νέο gazebo plugin για την ενεργοποίηση του lidar στο περιβάλλον προσομοίωσης. Αυτό επιτυγχάνεται προσθέτοντας το παρακάτω στο αρχείο robot.gazebo.

```
<!-- hokuyo -->
    <gazebo reference="hokuyo">
      <sensor name="laser" type="ray">
        <pose> 0 0 0 0 0 0 </pose>
        <visualize>true</visualize>
        <update_rate>20</update_rate>
        <ray>
          <scan>
```

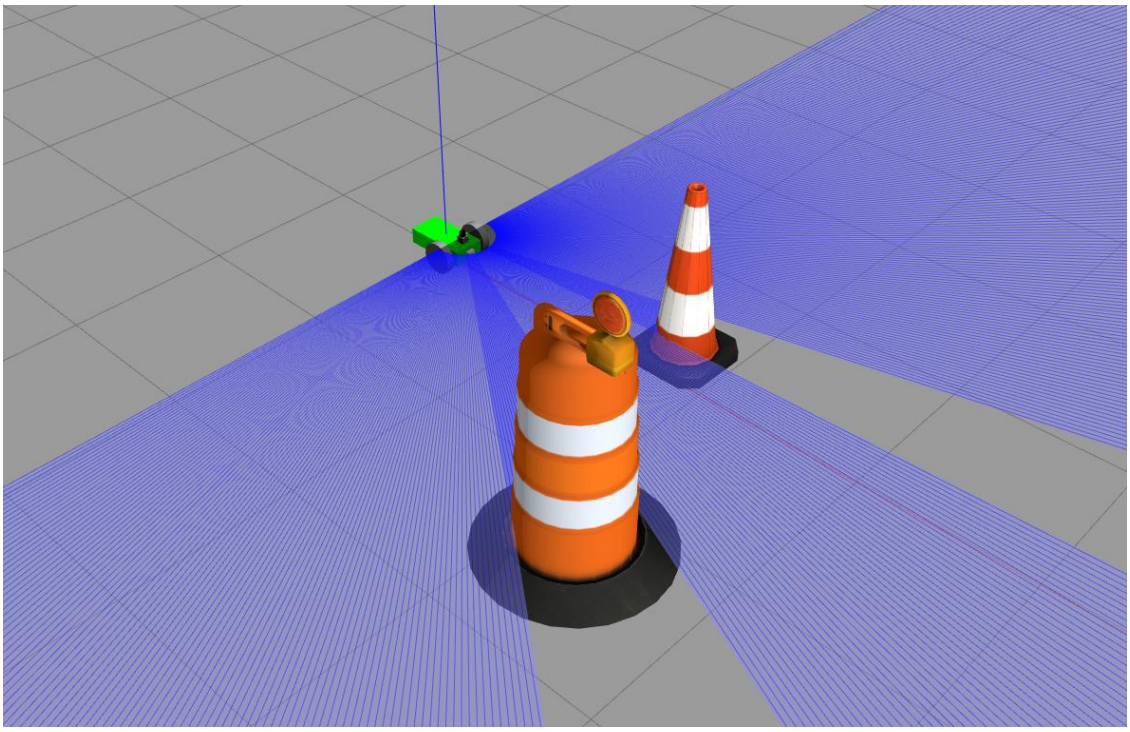
```
            <horizontal>
              <samples>360</samples>
              <min_angle>-1.57</min_angle>
              <max_angle>1.57</max_angle>
            </horizontal>
          </scan>
          <range>
            <min>0.2</min>
            <max>12</max>
          </range>
        </ray>
        <plugin name="laser_controller"filename="libgazebo_ros_ray_sensor.so">
          <ros>
            <argument>~/out:=scan</argument>
          </ros>
          <output_type>sensor_msgs/LaserScan</output_type>
          <frame_name>hokuyo</frame_name>
        </plugin>
      </sensor>
    </gazebo>
```

Με το παραπάνω καθορίζονται αρκετά από τα χαρακτηριστικά που θα έχει το lidar. Αρχικά ορίζεται η πόζα που θα έχει αυτό και το frame rate του. Έπειτα καθορίζεται το μέτρο της γωνίας που θα λαμβάνονται αποτελέσματα, δηλαδή από -1.57 έως 1.57. Επίσης ορίζεται η ελάχιστη και μέγιστη απόσταση που θα γίνεται η μέτρηση αποστάσεων. Τέλος, ορίζεται το plugin με όνομα laser_controller, το οποίο θα εκκινήσει ένα νέο node στο σύστημα και το οποίο θα επικοινωνεί με κάποια topics που θα εκκινήσουν.

Αφού ολοκληρωθούν τα παραπάνω, απομένει να επιβεβαιωθεί η ορθή λειτουργία του lidar. Για να γίνει αυτό, θα πρέπει το ρομποτικό μοντέλο να εισαχθεί στο περιβάλλον προσομοίωσης Gazebo και στο Rviz. Οπότε πρέπει να εκτελεστούν τα παρακάτω.

```
$ cd ~/ros2_ws

$ colcon build --package-select my_robot # For WSL, run the next command

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch my_robot my_robot_gazebo.launch.py
```

Πίνακας 89. Χρήση του lidar στο Gazebo και Rviz.



Εικόνα 49. Το lidar στο Gazebo.

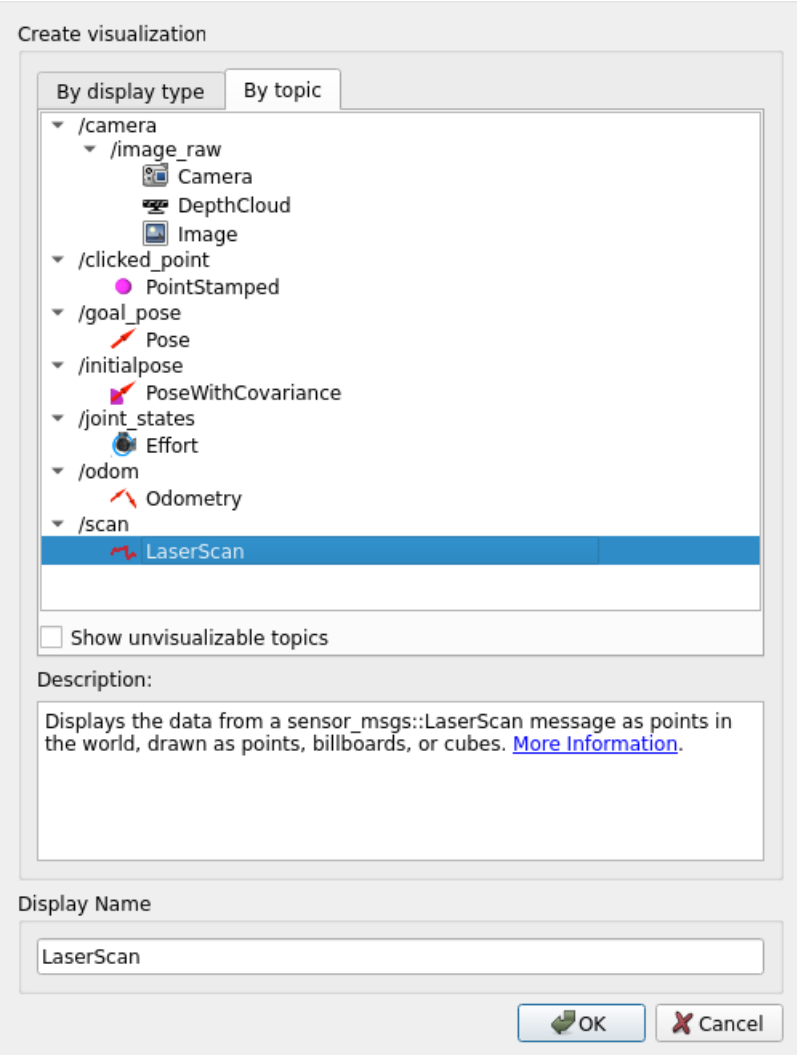
Στην παραπάνω εικόνα παρατηρούνται οι μπλε οριζόντιες γραμμές, κάτι που είναι ενδεικτικό ότι γίνεται μέτρηση απόστασης προς το καθορισμένο σημείο που ορίστηκε εντός του Gazebo plugin. Στη συνέχεια γίνεται η προσθήκη δύο κώνων, στους οποίους φαίνεται ότι η μπλε γραμμή διακόπτεται. Αυτό σημαίνει ότι το lidar έχει εντοπίσει το εμπόδιο και μετράει την αντίστοιχη απόσταση.

Για να γίνει η επιβεβαίωση ότι λειτουργεί το lidar, θα πρέπει να δοθούν σε ένα νέο τερματικό οι εντολές που δείχνουν ποια topics και nodes είναι ενεργά στο δίκτυο του ROS. Στη λίστα με τα topics θα πρέπει να εμφανιστεί το topic / scan. Στη λίστα με τα nodes πρέπει να εμφανιστεί το /laser_controller node. Τέλος, με την παρακάτω εντολή, μπορεί να γίνει η προβολή των μετρήσεων του lidar ανά μοίρα.

```
$ ros2 topic echo /scan
```

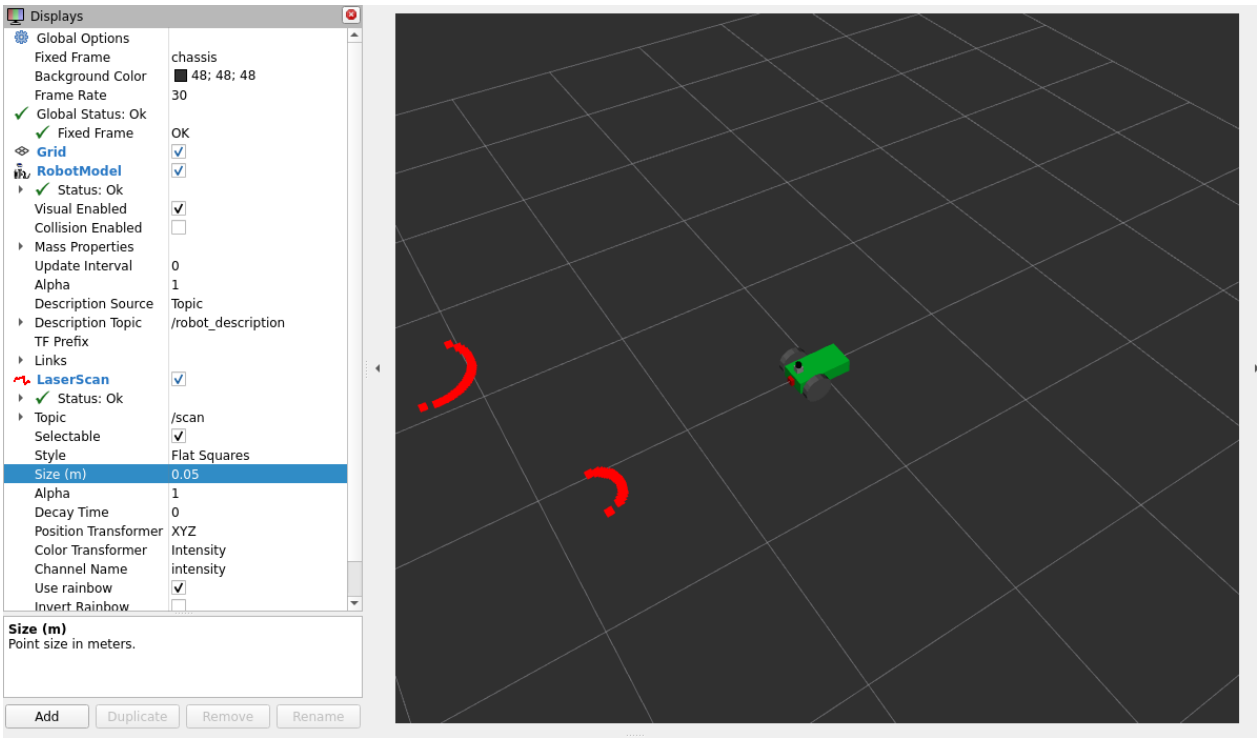
Πίνακας 90. Τα αποτελέσματα από το lidar.

Εκτός από το Gazebo, μπορεί να εκκινήσει το Rviz με παρόμοιο τρόπο όπως και για την κάμερα. Έπειτα πραγματοποιείται η προσθήκη του feature της LaserScan όπως παρατηρείται στην ακόλουθη εικόνα.




Εικόνα 50. Προσθήκη του laser στο Rviz.

Αφού προστεθεί το laser, πλέον παρατηρείται ότι τα δεδομένα από τον αισθητήρα οπτικοποιούνται στο τρισδιάστατο περιβάλλον του Rviz. Στο Display Panel υπάρχει ένα μενού για την τροποποίηση κάποιων ρυθμίσεων που αφορούν την οπτικοποίηση των δεδομένων του laser scan. Αλλάζοντας την τιμή της ρύθμισης size (M) σε 0.05 μέτρα, τότε μεγεθύνονται τα δεδομένα που αφορούν τα εμπόδια που βρίσκονται μπροστά από το ρομπότ. Μια ακόμη ενδιαφέρουσα επιλογή σε αυτές τις ρυθμίσεις είναι το Alpha, στο οποίο καθορίζεται η φωτεινότητα των αποτελεσμάτων από το lidar.




Εικόνα 51. Τα αποτελέσματα του lidar στο Rviz.

## 4.4 Η διαφορική οδήγηση του οχήματος

Το ρομπότ αποτελείται από δύο σταθερούς τροχούς, δηλαδή αυτοί μπορούν να εκτελέσουν περιστροφή μόνο ως προς έναν άξονα. Αυτού του είδους η κίνηση ονομάζεται διαφορική κίνηση. Για να μπορέσει να γίνει η χρήση της διαφορικής οδήγησης στο ρομπότ, θα πρέπει να προστεθεί ένα νέο Gazebo plugin στο URDF. Αυτό το νέο plugin θα ενεργοποιεί εκτός από τη διαφορική κίνηση αλλά και τη μέτρηση για την οδομετρία του ρομπότ.

Το νέο gazebo plugin θα προστεθεί στο robot.gazebo και είναι το ακόλουθο:

```
<!--Differential Drive Controller for 2-wheel-robot-->

 <gazebo>
   <plugin name="diff_drive" filename="libgazebo_ros_diff_drive.so">
     <!-- Wheel Information -->
     <left_joint>left_wheel_hinge</left_joint>
     <right_joint>right_wheel_hinge</right_joint>
     <wheel_separation>0.4</wheel_separation>
     <wheel_diameter>0.1</wheel_diameter>
```

```
        <!-- Limits -->
        <max_wheel_torque>200</max_wheel_torque>
        <max_wheel_acceleration>10.0</max_wheel_acceleration>
        <!-- Output -->
        <odometry_frame>odom</odometry_frame>
        <robot_base_frame>chassis</robot_base_frame>
        <publish_odom>true</publish_odom>
        <publish_odom_tf>true</publish_odom_tf>
        <publish_wheel_tf>true</publish_wheel_tf>
      </plugin>
    </gazebo>
```

Στο παραπάνω ορίζεται η διαφορική κίνηση για το ρομποτικό μοντέλο. Καθορίζονται τα joints για τη δεξιά και αριστερή ρόδα, η απόσταση των δύο τροχών και η διάμετρος του τροχού. Έπειτα καθορίζονται τα όρια για την μέγιστη ροπή και επιτάχυνση. Τέλος, ενεργοποιείται η οδομετρία για το ρομπότ.

Έπειτα από τη προσθήκη του παραπάνω, το ρομποτικό μοντέλο θα πρέπει να τοποθετηθεί στο Gazebo για να ελεγχθεί η λειτουργία της διαφορικής κίνησης του ρομπότ. Για να επιτευχθεί αυτό, θα πρέπει πρώτα να χτιστεί το πακέτο και να εκκινήσει το launch αρχείο.

Με τη διαφορική κίνηση που προστέθηκε παραπάνω, πλέον είναι εφικτό να δοθεί επιθυμητή ταχύτητα στο ρομπότ, είτε μέσω command line είτε μέσω του εργαλείου rqt. Για αρχή θα σταλεί επιθυμητή ταχύτητα μέσω command line. Για να επιτευχθεί αυτό, θα πρέπει να υπάρχει η γνώση του topic που θα αποσταλεί το μήνυμα, τον τύπο μηνύματος αλλά και των ορισμάτων που πρέπει να δοθούν.

```
$ ros2 topic list –t # find all the nodes and the type of message of every topic.

$ ros2 topic info /cmd_vel

$ ros2 interface show geometry_msgs/msg/Twist
```

Πίνακας 91. Ο τύπος μηνύματος για την επιθυμητή ταχύτητα και τα ορίσματα του.

Με χρήση των παραπάνω εντολών, εξάγεται το συμπέρασμα ότι θα πρέπει να σταλεί μήνυμα στο /cmd_vel topic. Το μήνυμα θα είναι τύπου geometry_msgs/msg/Twist και θα δέχεται σαν ορίσματα δύο διανύσματα. Τελικά για τη δήλωση επιθυμητής ταχύτητα για το ρομπότ πρέπει να δοθεί η ακόλουθη εντολή.

```
$  ros2 topic pub /cmd_vel geometry_msgs/msg/Twist '{linear:{x: 0.4}, angular:{z: 0.2}}'
```

Πίνακας 92. Δήλωση επιθυμητής ταχύτητας.

Με την παραπάνω εντολή δηλώνεται η επιθυμητή ταχύτητα ως προς τον άξονα x να είναι ίση με 0.4 και η γωνιακή ταχύτητα ως προς z θα ισούται με 0.2. Συνεπώς, το ρομπότ διαφορικής κίνησης θα κινείται πάνω σε ένα κύκλο και η κίνηση του μπορεί να παρατηρηθεί στο Gazebo. Για να σταματήσει η κίνηση του ρομπότ, θα πρέπει να δοθεί η παραπάνω εντολή με μηδενικές ταχύτητες.

Η επιθυμητή ταχύτητα εκτός της δήλωσης μέσω τερματικού μπορεί να δοθεί και από το γραφικό περιβάλλον του rqt. Το rqt είναι ένα πολύ ισχυρό εργαλείο μέσω του οποίου μπορούν να καθοριστούν πολλές ρυθμίσεις του συστήματος μέσω ενός γραφικού περιβάλλοντος.

Για να εκκινήσει το rqt, εκτελείται σε ένα τερματικό η εντολή rqt. Αφού εκκινήσει το γραφικό περιβάλλον του εργαλείου, επιλέγεται η καρτέλα **Plugins<< Topics << Message Publisher**. Αφού γίνει η μετάβαση στην συγκεκριμένα καρτέλα, τότε μπορεί να γίνει η αποστολή μηνύματος σε οποιοδήποτε topic που είναι ενεργό στο ROS δίκτυο. Στην καρτέλα topic επιλέγεται το /cmd_vel και έπειτα γίνεται η προσθήκη του μέσω της επιλογής +.

Στο σημείο αυτό μπορεί να καθοριστεί η επιθυμητή γραμμική ταχύτητα ως προς τον άξονα χ και η γωνιακή ως προς τον άξονα z. Αφού αποφασιστεί η ταχύτητα που είναι επιθυμητή, τότε επιλέγοντας το τετραγωνάκι αριστερά του /cmd_vel, τα δεδομένα στέλνονται στο αντίστοιχο topic και ξεκινάει η κίνηση του ρομπότ στο Gazebo.


Εικόνα 52. Αποστολή δεδομένων μέσω του rqt.

### 4.5 Αλγόριθμος αποφυγής εμποδίων

Αφού πλέον έχουν προστεθεί όλα τα component που είναι απαραίτητα για την αυτόνομη οδήγηση του ρομπότ διαφορικής κίνησης, μπορεί να αναπτυχθεί ένας απλός αλγόριθμος σε Python με στόχο την αποφυγή εμποδίων σε ένα περιβάλλον προσομοίωσης. Συνοπτικά, ο αλγόριθμος θα διαβάζει δεδομένα από το lidar και ανάλογα την απόσταση που θα βρίσκεται το εμπόδιο, θα συνεχίζει την πορεία του ή θα περιστρέφεται για να το αποφύγει.

Ο αλγόριθμος που θα αναπτυχθεί παρακάτω, είναι απλός και έχει σαν στόχο να δώσει έμφαση στη δημιουργία ενός publisher και subscriber μέσω της δημιουργίας ενός κώδικα. Δηλαδή ο αλγόριθμος θα κάνει subscribe από ένα topic που είναι ενεργό στο δίκτυο του ROS. Έπειτα ανάλογα τα αποτελέσματα που θα γίνονται subscribe, τότε θα γίνεται publish ένα μήνυμα συγκεκριμένου τύπου σε ένα άλλο ROS topic. Τέλος, θα παρουσιαστεί ο τρόπος για να γίνει το παραπάνω αρχείο εκτελέσιμο του πακέτου.

Ο αλγόριθμος είναι ανεπτυγμένος σε γλώσσα Python, ωστόσο το ROS προσφέρει την δυνατότητα ανάπτυξης αλγορίθμων τύπου Publisher-Subscriber και σε γλώσσα C++. Ο παρακάτω κώδικας θα αποθηκευτεί στον φάκελο my_robot του πακέτου με το όνομα diff_drive.py και είναι ο ακόλουθος:

```
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan

class diff_drive(Node):

    def __init__(self):
        super().__init__('obstacle_avoider')
        # Create the publisher, define the type of message, the topic
        # and the frame rate.
        self.__publisher = self.create_publisher(Twist,'cmd_vel',1)

        # Create the subscriber, define the type of message,
        # the topic and the frame rate
        self.subscription = self.create_subscription(
            LaserScan,
            '/scan',
            self.listener_callback,
            10)

    # Create a characteristic of the class.
```

```
    def listener_callback(self,msg):
        # Create a variable and Define the position of the scanner,
        # which will subscribe the distance.
        self.msg = min(msg.ranges[165:179]+msg.ranges[180:195])
        # Print the measurement distance.
        print(f"The measurement distance is {msg}.")

        # Create a variable, and give him the type of Twist()
        command_msg = Twist()

        # Define the speed of the robot to 0.2 m/s in x axis
        command_msg.linear.x = 0.2

        if self.msg < 0.5:
            # Define the speed of the robot, if the obstacle is in front.
            command_msg.angular.z = 2.0
            command_msg.linear.x = 0.0

        # Publish the speed.
        self.__publisher.publish(command_msg)

def main(args=None):
    rclpy.init(args=args)
    # Create an instance
    avoider = diff_drive()
    # run the instance
    rclpy.spin(avoider)

    # End the programm with Cntr+C
    avoider.destroy_node()
    rclpy.shutdown()

# Call the main function
if __name__=='__main__':
    main()
```

Με χρήση του παραπάνω κώδικα επιτυγχάνεται η αποστολή μηνυμάτων για την αλλαγή ταχύτητας του ρομπότ, βάση της απόστασης ενός εμποδίου. Στην αρχή του κώδικα εισάγονται κάποιες βιβλιοθήκες που βοηθούν στη συγγραφή του κώδικα. Πιο συγκεκριμένα, εισάγονται βιβλιοθήκες για το είδος μηνύματος κάθε topic και έπειτα δημιουργείται η κλάση diff_drive. Η κλάση έχει δύο συναρτήσεις την __init()__ η

οποία εκτελείται κάθε φορά που δημιουργείται ένα νέο στιγμιότυπο και τη συνάρτηση listener_callback() η οποία αποτελεί ένα χαρακτηριστικό της κλάσης.

Στην __init__() ορίζεται ο publisher και ο subscriber, τα οποία με τη σειρά τους δέχονται κάποια ορίσματα, τα οποία είναι ο τύπος μηνύματος, το topic και ο ρυθμός μετάδοσης μηνυμάτων. Στο χαρακτηριστικό listener_callback() καθορίζεται το εύρος που θα λαμβάνονται τα δεδομένα από το scanner και έπειτα καθορίζεται η ταχύτητα του ρομπότ αν υπάρχει ή όχι εμπόδιο μπροστά από αυτό. Τέλος ορίζεται η συνάρτηση main() εκτός της κλάσης. Σε αυτή δημιουργείται ένα στιγμιότυπο της κλάσης και εκτελείται σαν ατέρμονος βρόγχος μέχρι το πρόγραμμα να τερματίσει. Ο κώδικας ολοκληρώνεται με την εκτέλεση της συνάρτησης main().

Σε αυτό το σημείο, ο παραπάνω κώδικας θα πρέπει να γίνει εκτελέσιμο του πακέτου. Για να επιτευχθεί αυτό θα πρέπει να γίνει η επεξεργασία ενός αρχείου του πακέτου my_robot. Πιο συγκεκριμένα, στο αρχείο setup.py πρέπει να γίνει η επεξεργασία του λεξικού entry_points. Το λεξικό θα αντικατασταθεί και θα προστεθεί το ακόλουθο :

```
entry_points={
        'console_scripts': [
            'diff_drive = my_robot.diff_drive:main'
        ], },
```

Πίνακας 93. Αλλαγή του λεξικού entry_points.

Αφού πλέον έχει γίνει το παραπάνω βήμα τότε μπορεί να εκτελεστεί το εκτελέσιμο και να παρατηρηθούν τα αποτελέσματα του αλγορίθμου. Απαραίτητο βήμα για την εκτέλεση του παραπάνω κώδικα είναι το χτίσιμο του πακέτου. Έπειτα να τοποθετηθεί το ρομπότ στο Gazebo και τέλος να προστεθεί ένα εμπόδιο μπροστά από το ρομπότ ή αυτό να τοποθετηθεί σε ένα κτήριο. Σε ένα δεύτερο τερματικό εκτελούνται οι παρακάτω εντολές.

```
$ cd ~/ros2_ws

$ ros2 run my_robot diff_drive
```

Πίνακας 94. Εκκίνηση του εκτελέσιμου diff_drive.

Σε αυτό το σημείο το εκτελέσιμο θα εκκινήσει και θα πρέπει να παρατηρηθεί η κίνηση του ρομπότ στο περιβάλλον προσομοίωσης. Όταν το ρομπότ προσεγγίσει το εμπόδιο, τότε θα πρέπει να περιστραφεί για το αποφύγει. Αν το ρομπότ προσπαθεί να αποφύγει το εμπόδιο τότε η βασική λειτουργία του αλγορίθμου, δηλαδή το publish και subscribe προς και από τα topics , έχει επιτευχθεί.

## Κεφάλαιο 5

## Εντοπισμός και Χαρτογράφηση αυτόνομου ρομπότ

Το κεφάλαιο αυτό είναι αφιερωμένο στην χρήση ενός πολύ σημαντικού πακέτου του ROS 2 το οποίο είναι το Nav 2. Το πακέτο αυτό είναι ιδιαίτερα χρήσιμο για τον εντοπισμό, τη χαρτογράφηση και τη πλοήγηση ενός ρομπότ καθώς και σε άλλες διεργασίες που αφορούν την αυτονομία του. Στην αρχή του κεφαλαίου θα εγκατασταθεί το πακέτο και έπειτα θα εκτελεστεί ένα demo του. Έπειτα θα δημιουργηθεί ένα ρομπότ διαφορική κίνησης στο οποίο θα προστεθούν κατάλληλοι αισθητήρες που βοηθούν στον εντοπισμό και την χαρτογράφηση. Τέλος, θα παρουσιαστούν τα βήματα για τον εντοπισμό του ρομπότ, τη χαρτογράφηση, την επαναχρησιμοποίηση του χάρτη και την πλοήγηση στο χώρο εργασίας ενός ρομπότ.

### 5.1 Εγκατάσταση του πακέτου

Το πακέτο Nav2 είναι ο διάδοχος του ROS Navigation Stack το οποίο χρησιμοποιείται σε παρόμοιες τεχνολογίες που απορρέουν από αυτόνομα ρομποτικά συστήματα. Το project του Nav2 επιδιώκει να βρει ασφαλείς τρόπους μετακίνησης ενός ρομπότ στο χώρο εργασίας του και την ολοκλήρωσή αποστολών ενός ρομποτικού συστήματος σε διάφορα περιβάλλοντα και για τις διάφορες κινηματικές εξισώσεις των ρομπότ.

Το πακέτο εκτός από το γεγονός ότι μπορεί να βοηθήσει ένα ρομπότ να μετακινηθεί σε διάφορα σημεία, επιτρέπει σε αυτό να καταλάβει διάφορες ενδιάμεσες πόζες, αλλά και να επιτύχει άλλες πιο σύνθετες διεργασίες όπως για παράδειγμα να ακολουθήσει ένα κινούμενο εμπόδιο. Πληροφοριακά το Nav2 είναι ένα πραγματικού χρόνου υψηλής ποιότητας framework για την πλοήγηση ρομπότ και χρησιμοποιείται από αρκετές εταιρίες.

Μέσω του πακέτου προσφέρονται ικανότητες αντίληψης του περιβάλλοντος, έλεγχος και εντοπισμός για το ρομποτικό σύστημα, οπτικοποίηση των δεδομένων του και άλλα χαρακτηριστικά με τα οποία μπορεί να κατασκευαστεί ένα αρκετά αξιόπιστο αυτόνομο ρομπότ. Το πακέτο χρησιμοποιεί τα behavior trees (δέντρα αποφάσεων) για την δημιουργία προσαρμοζόμενων και έξυπνων συμπεριφορών πλοήγησης για το ρομπότ. Επίσης προσφέρονται εργαλεία που μπορούν να χρησιμοποιηθούν για τη πλοήγηση και εντοπισμό ενός ρομπότ. Κάποια από αυτά τα εργαλεία είναι τα παρακάτω:

- Map Server: ανέβασμα και αποθήκευση ενός χάρτη.
- AMCL: εντοπισμός του ρομπότ σε ένα χάρτη.
- Nav2 Planner: σχεδίαση διαδρομής από ένα σημείο σε ένα άλλο αποφεύγοντας τα εμπόδια.

- Nav2 Control: έλεγχος του ρομπότ καθώς ακολουθεί την πορεία που σχεδίασε.
- Nav2 Costmap 2D: μετατροπή των δεδομένων των αισθητήρων σε ένα χάρτη του κόσμου.

Τα παραπάνω εργαλεία είναι από τα σημαντικότερα, αλλά προσφέρονται και αρκετά ακόμη. Αρκετές πληροφορίες για το πακέτο, τις επεκτάσεις του αλλά και για τα υπόλοιπα εργαλεία που προσφέρει αυτό, μπορούν να βρεθούν στην ιστοσελίδα του πακέτου.

Η εγκατάσταση του πακέτου πραγματοποιείται με την ακόλουθη εντολή.

```
$ sudo apt install ros-humble-navigation2 ros-humble-nav2-bringup ros-humble-turtlebot3-gazebo
```

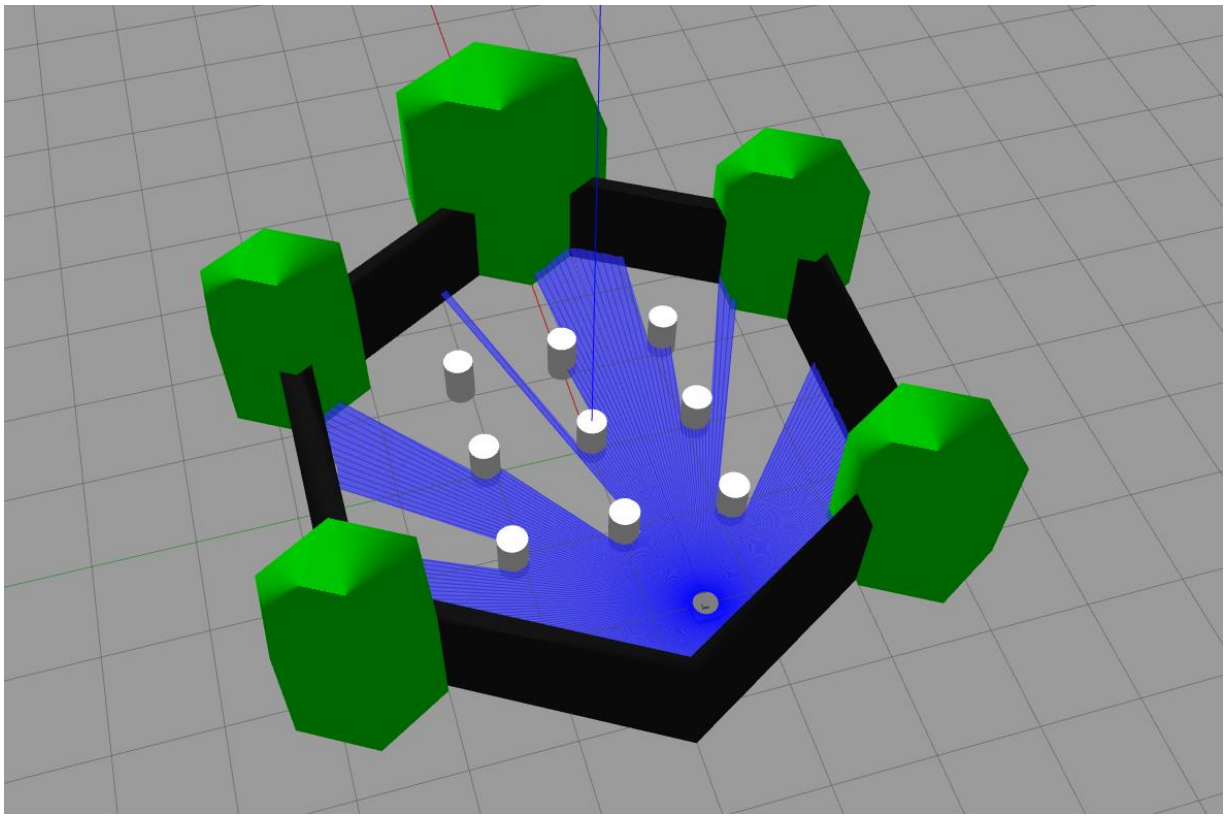
Πίνακας 95. Εγκατάσταση του Nav2.

Μέσω της παραπάνω εντολής, εγκαθίστανται κάποια βασικά εργαλεία του πακέτου Nav2, παράλληλα εγκαθίστανται και κάποια πακέτα του Gazebo για την προσομοίωση του turtlebot3. Για την εκτέλεση του demo πρέπει να εκτελεστούν τα ακόλουθα:

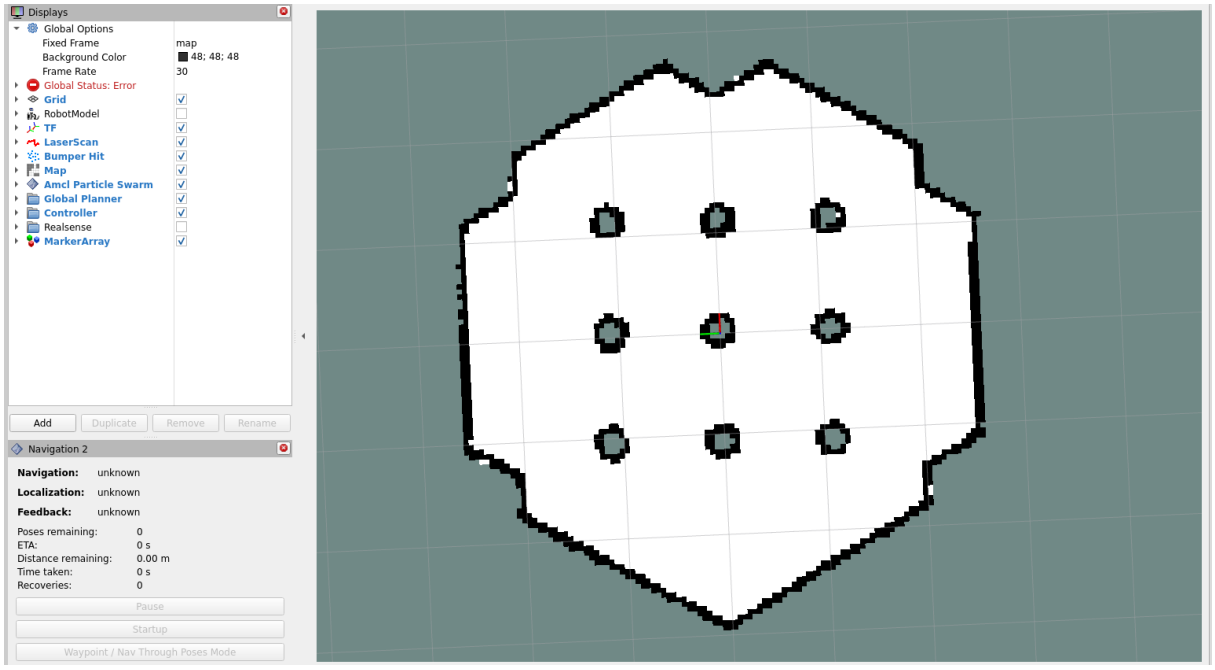
```
$ source /opt/ros/humble/setup.bash

$ export TURTLEBOT3_MODEL=waffle

$ export
GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:opt/ros/humble/share/turtlebot3_gazebo/mo
dels

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0 # FOR WSL

$ ros2 launch nav2_bringup tb3_simulation_launch.py headless:=False
```

Πίνακας 96. Εκτέλεση του demo για τον navigation του turtlebot3.

Με την εκτέλεση των παραπάνω εντολών, καθορίζονται κάποιες μεταβλητές για το περιβάλλον και έπειτα εκτελείται ένα launch αρχείο του πακέτου nav2_bringup. Το συγκεκριμένο launch αρχείο, θα εκκινήσει το Nav2 μαζί με τα εργαλεία, map server, AMCL localizer και άλλα εργαλεία στον turtlebot3_world κόσμο. Παράλληλα θα εκκινήσουν το περιβάλλον του Gazebo και Rviz, το robot_state_publisher node και τα συστήματα συντεταγμένων του ρομπότ. Οπότε αν δεν προκύψει κάποιο σφάλμα θα πρέπει να εμφανιστούν τα παρακάτω:



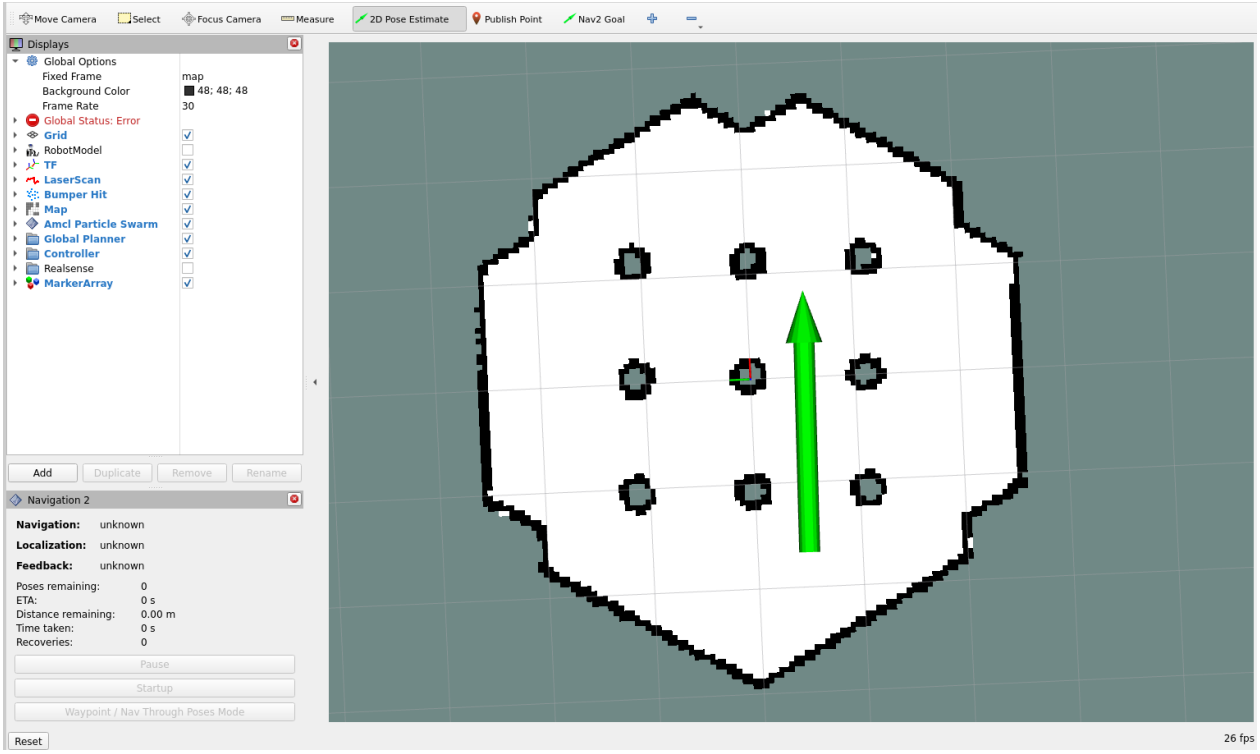
Εικόνα 53. Το turtlebot3 στο Gazebo.



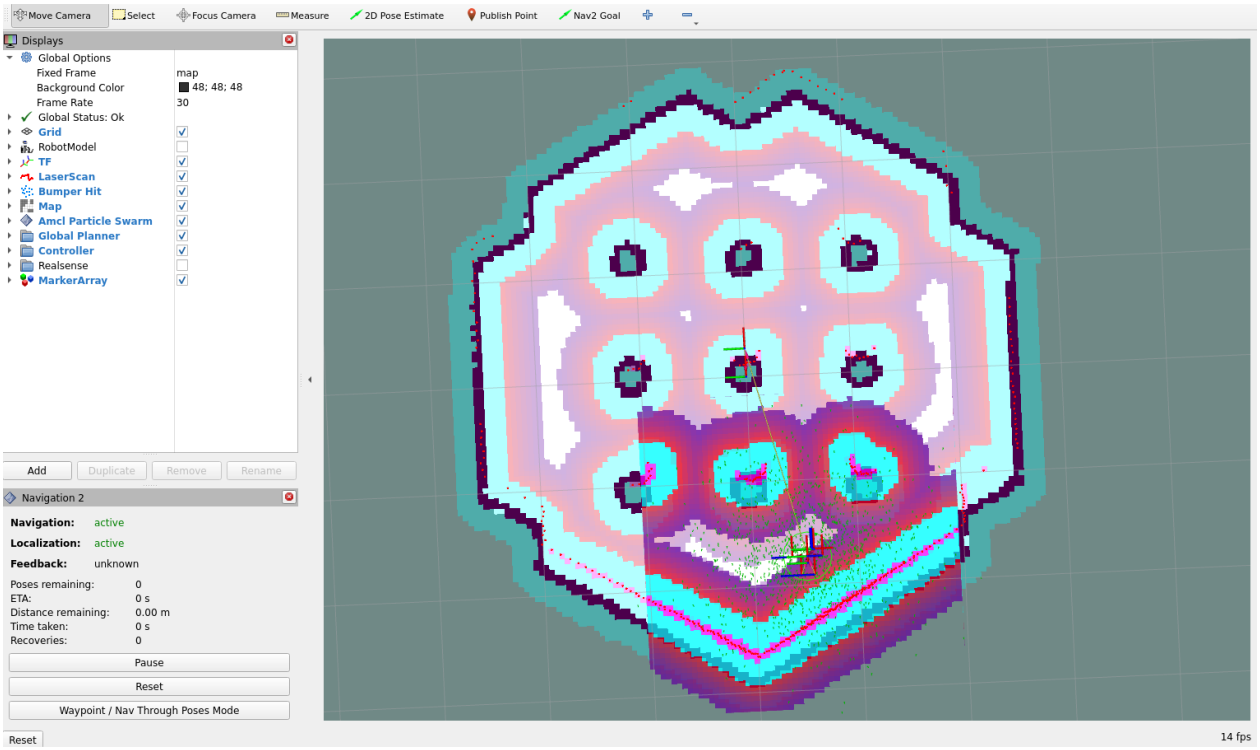

Εικόνα 54. Το περιβάλλον του Rviz για το turtlebot3.

Σε περίπτωση που δεν εμφανιστεί ο χάρτης της εικόνας 54, τότε γίνεται η επιλογή του Startup που βρίσκεται στο Display Panel. Με το παραπάνω launch αρχείο, εκκινούν πολλά nodes, topics και actions του συστήματος. Επίσης στο Rviz έχει φορτωθεί και ένα χάρτης του περιβάλλοντος του ρομπότ. Τα attributes που είναι ενεργά στο δίκτυο του ROS μπορούν να εμφανιστούν με κατάλληλες εντολές σε ένα τερματικό. Επίσης μπορεί να χρησιμοποιηθεί και το rqt μέσω του οποίου μπορούν να υλοποιηθούν εργασίες ευκολότερα μέσω γραφικού περιβάλλοντος.

Αφού εκκινήσουν τα παραπάνω, το ρομπότ δεν γνωρίζει ποια θέση καταλαμβάνει στο χάρτη. Αυτός είναι και ο λόγος που δεν εμφανίζεται στο περιβάλλον του Rviz. Το Nav2 αναμένει να του δοθεί η κατά προσέγγιση αρχική τοποθέτηση του ρομπότ ώστε να μπορέσει να αρχίσει να διαβάζει το περιβάλλον του. Για να γίνει αυτό, πρέπει να παρατηρηθεί η θέση του ρομπότ στο Gazebo, έπειτα στο Rviz να επιλεγεί το "2D Pose Estimate" που βρίσκεται στο toolbar του γραφικού περιβάλλοντος. Τέλος στον τρισδιάστατο χώρο του Rviz να γίνει η επιλογή της κατά προσέγγισης τοποθέτησης που βρίσκεται το ρομπότ. Αυτό παρατηρείται στην ακόλουθη εικόνα.



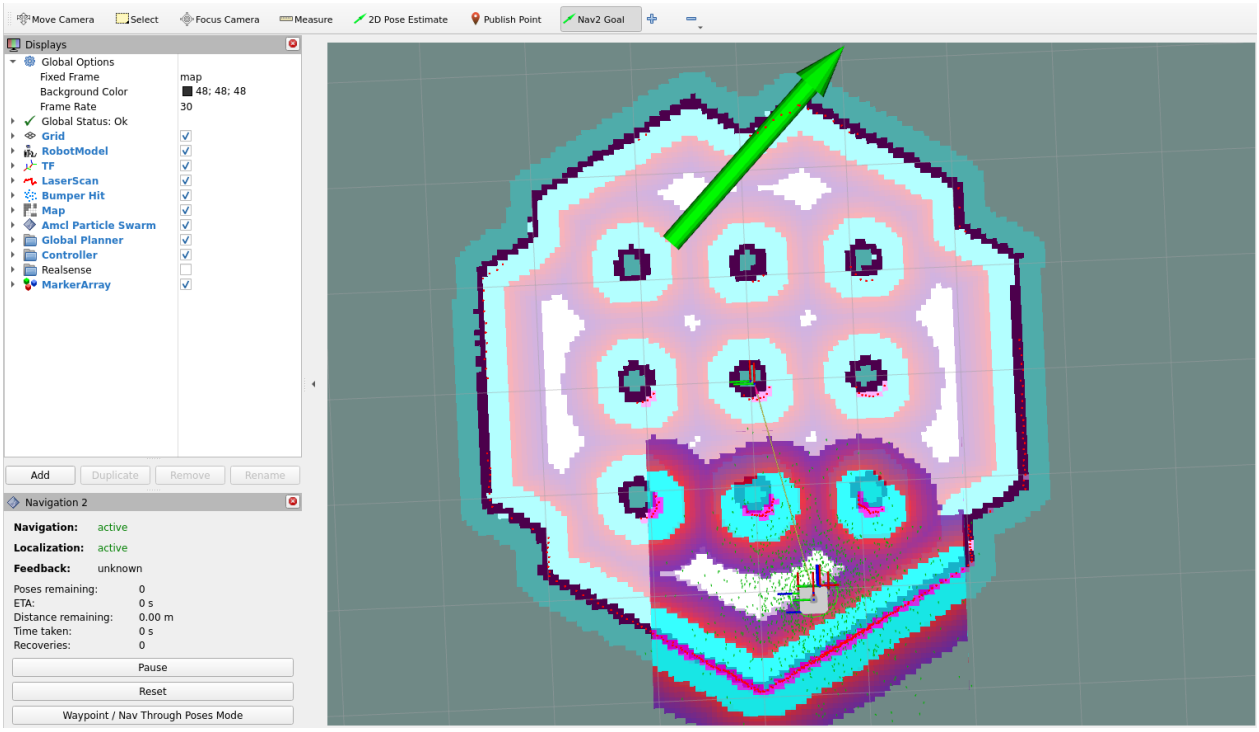

Εικόνα 55. Καθορισμός της αρχικής θέσης του ρομπότ.



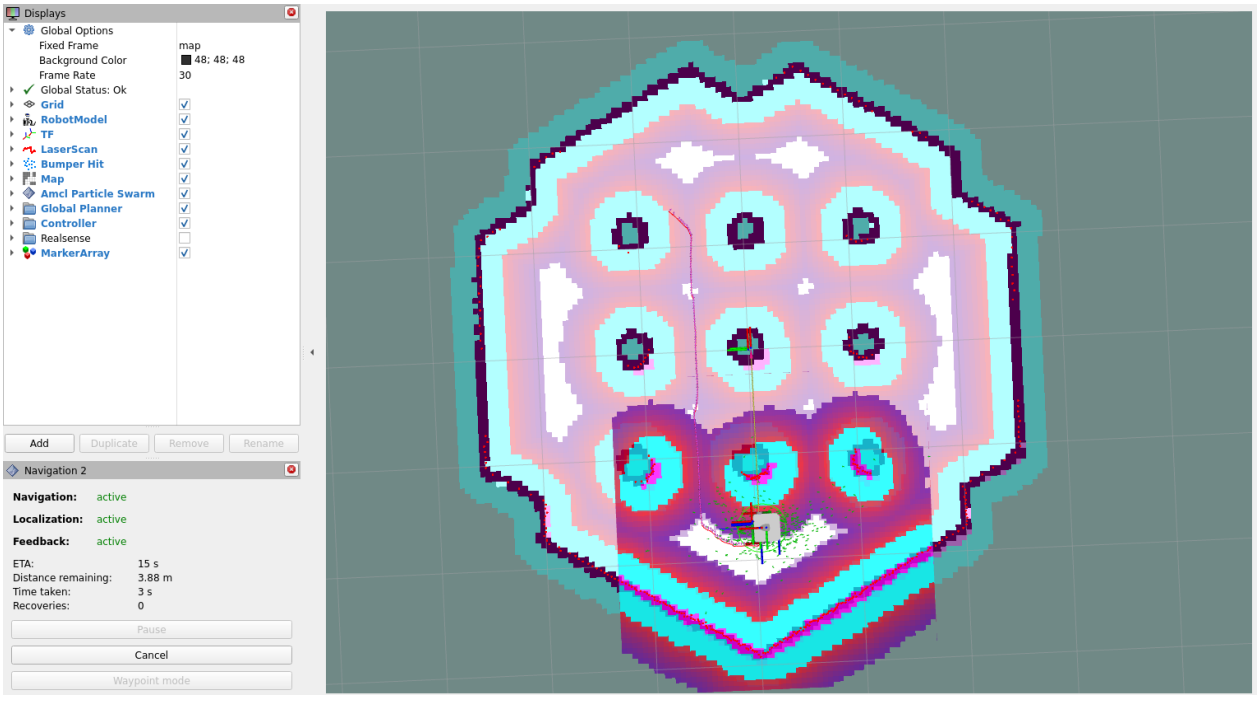

Εικόνα 56. Το ρομπότ με particle cloud.

Αφού πλέον έχει εντοπιστεί η θέση και η πόζα του ρομπότ, πλέον στον τρισδιάστατο χώρο του Rviz, παρατηρούνται οι αποστάσεις που έχει το ρομπότ από τα εμπόδια μέσω ενός costmap. Επιπρόσθετα μπορεί να προστεθεί και το ρομποτικό μοντέλο στο Rviz. Αυτό μπορεί να επιτευχθεί από το Display Panel, επιλέγοντας το Robot Model.

Ένα πολύ σημαντικό Feature που διαθέτει το Rviz, βρίσκεται στο toolbar του γραφικού περιβάλλοντος. Πιο συγκεκριμένα, μπορεί να επιλεγεί το "Navigation2 Goal" και να δοθεί ένας επιθυμητός στόχος για το ρομπότ. Δηλαδή, μια τελική τοποθέτηση η οποία είναι επιθυμητό να προσεγγίσει το ρομπότ. Επιλέγοντας το " Navigation2 Goal" και δηλώνοντας επιθυμητή θέση και πόζα για το ρομπότ εντός του τρισδιάστατου χώρου του Rviz, θα ενεργοποιηθεί το Behavior Trees navigation. Αυτό σημαίνει ότι το ρομπότ θα σχεδιάσει μια διαδρομή ώστε μέσω αυτής το ρομπότ να προσεγγίσει την επιθυμητή θέση και πόζα και παράλληλα να αποφύγει τα εμπόδια που θα προκύψουν στην πορεία του.




Εικόνα 57. Καθορισμός της νέας τοποθέτησης του ρομπότ.




Εικόνα 58. Η χάραξη της διαδρομής που ακολουθεί το ρομπότ.

Αφού δηλωθεί η νέα επιθυμητή τοποθέτηση για το ρομπότ, τότε αυτό αρχίζει να κινείται πάνω στην πορεία που έχει χαράξει. Παράλληλα ανανεώνεται ο χάρτης, καθώς πλέον άλλα εμπόδια εντοπίζονται μπροστά από το ρομπότ. Επίσης παρατηρείται ότι το ρομπότ κινείται στην προσομοίωση με τον τρόπο που φαίνεται και στο Rviz.

Το παραπάνω demo, είναι ένα παράδειγμα στο οποίο παρουσιάζονται κάποια από τα βασικά χαρακτηριστικά του πακέτου Nav2. Πολλές λεπτομέρειες για τις λειτουργίες του συστήματος που χρησιμοποιούνται στο παραπάνω παράδειγμα μπορούν να βρεθούν στην ιστοσελίδα του πακέτου. Στη συνέχεια του κεφαλαίου θα παρουσιαστεί η δημιουργία ενός ρομπότ διαφορικής κίνησης από την αρχή, χρησιμοποιώντας λειτουργίες του Nav2 για τον εντοπισμό του αλλά και την χαρτογράφηση.

## 5.2 Δημιουργία του πακέτου και το δίτροχο ρομπότ

Στην ενότητα αυτή θα δημιουργηθεί ένα νέο πακέτο και παράλληλα θα σχολιαστεί η δημιουργία του δίτροχου ρομπότ που θα χρησιμοποιηθεί στο κεφάλαιο αυτό. Παρά το γεγονός ότι ο κώδικας που θα αναπτυχθεί θα βασίζεται αρκετά στα προηγούμενα κεφάλαια, προτιμάται η δημιουργία ενός νέου πακέτου. Το πακέτου θα έχει όνομα robot_loc_map, όπου το loc είναι συντομογραφία του localization και αντίστοιχα το map αποτελεί την συντομογραφία του mapping.

```
$ cd ~/ros2-ws/src

$ ros2 pkg create --build-type ament_python robot_loc_map
```

Πίνακας 97. Δημιουργία του πακέτου robot_loc_map.

Όπως σε κάθε πακέτο, πρέπει να δημιουργηθούν folders εντός του πακέτου για καλύτερη οργάνωση του κώδικα που θα αναπτυχθεί. Θα δημιουργηθούν παρόμοιοι φάκελοι όπως και στα προηγούμενα κεφάλαια.

```
$ cd ~/ros2-ws/src

$ mkdir urdf launch meshes worlds config
```

Πίνακας 98. Δημιουργία folders στο πακέτο robot_loc_map.

Έπειτα για να μπορέσει να γίνει η εγκατάσταση του κώδικα που θα δημιουργηθεί, θα πρέπει να γίνει η επεξεργασία του αρχείου setup.py του πακέτου. Θα πρέπει να εισαχθούν δύο νέες βιβλιοθήκες και έπειτα να καθοριστούν κάποια μονοπάτια. Συνεπώς η λίστα data_files του αρχείο setup.py θα πρέπει να είναι ενημερωμένη και να είναι η ακόλουθη.

```
from setuptools import find_packages, setup
from glob import glob
import os

package_name = 'robot_loc_nav'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
        (os.path.join('share',package_name,'launch'),
         glob(os.path.join('launch','*.launch.py'))),
        (os.path.join('share',package_name,'urdf'),
         glob(os.path.join('urdf','*.xacro'))),
         (os.path.join('share',package_name,'urdf'),
         glob(os.path.join('urdf','*.gazebo'))),
        (os.path.join('share',package_name,'worlds'),
         glob(os.path.join('worlds','*.world'))),
        (os.path.join('share',package_name,'meshes'),
         glob(os.path.join('meshes','*.dae'))),
        (os.path.join('share',package_name,'meshes'),
         glob(os.path.join('meshes','*.png'))),
        (os.path.join('share',package_name,'meshes'),
         glob(os.path.join('meshes','*.stl'))),
        (os.path.join('share',package_name,'config'),
         glob(os.path.join('config','*.yaml'))),
    ],
```

Εικόνα 59. Το νέο setup.py του πακέτου robot_loc_nav.

Αφού ολοκληρωθεί το παραπάνω βήμα, ακολουθεί η δημιουργία του URDF και του launch αρχείου που θα χρησιμοποιηθούν. Για το URDF μοντέλο προτιμάται η ανάπτυξη του αρχείου με μεθόδους Xacro οι λειτουργίες του οποίο βοηθούν στην ευκολότερη ανάπτυξη του κώδικα. Πιο συγκεκριμένα, θα χρησιμοποιηθούν οι μέθοδοι, include, macro και properties. Το μοντέλο URDF μπορεί να βρεθεί στον σύνδεσμο: https://github.com/DimitrisKatos/robot_loc_nav/blob/master/urdf/robot_loc_nav.xacro.

Παράλληλα με το URDF δημιουργούνται άλλα τρία αρχεία, τα οποία χρησιμοποιώντας τη μέθοδο include του Xacro, εισάγονται στο βασικό URDF. Το πρώτο από τα τρία αρχεία είναι το materials.xacro στο οποίο ορίζονται τα χρώματα του ρομποτικού μοντέλου τα οποία και θα χρησιμοποιηθούν στο περιβάλλον του Rviz. Στα επόμενα δύο αρχεία αναπτύσσονται δύο Gazebo plugins. Το πρώτο από τα δύο αρχεία θα ονομαστεί my_robot.gazebo και στο οποίο ορίζονται οι τριβές, το χρώμα κάθε link στο Gazebo και ορίζεται η διαφορική κίνηση του ρομπότ. Στο επόμενο αρχείο, γίνεται η ενεργοποίηση του laser στο gazebo και το

όνομα του αρχείου θα είναι laser.gazebo. Τα αρχεία αυτά είναι παρόμοια με αυτά που αναπτύχθηκαν στο προηγούμενο κεφάλαιο αλλά μπορούν να βρεθούν μέσω του προηγούμενου συνδέσμου.

Στα Gazebo plugins που ορίζονται για το νέο ρομπότ, είναι απαραίτητο να δοθεί ιδιαίτερη προσοχή στα ορίσματα που δέχονται αυτά. Πιο συγκεκριμένα, θα χρειαστεί να τροποποιηθούν παράμετροι όπως η διάμετρος της ακτίνας των ροδών και η απόσταση μεταξύ των δύο τροχών. Επιπρόσθετα, στο plugin που ορίζεται το laser, θα τροποποιηθεί το εύρος μετρήσεων του laser με τέτοιο τρόπο ώστε η μέτρηση αποστάσεων να πραγματοποιείται σε όλο το επίπεδο και όχι μόνο προς μια κατεύθυνση. Ο λόγος που γίνεται αυτό είναι διότι στη χαρτογράφηση του ρομπότ είναι απαραίτητη η μέτρηση αποστάσεων σε όλο το επίπεδο. Έπειτα από τις τροποποιήσεις που θα γίνουν στα plugins, αυτά πρέπει να γίνουν import στο βασικό URDF με χρήση της μεθόδου xacro include.

Στο σημείο αυτό θα πραγματοποιηθεί μια ακόμη προσθήκη στο URDF η οποία είναι το footprint του ρομπότ. Το footprint, δηλαδή το ίχνος αυτού, είναι ένα δισδιάστατο σχήμα το οποίο προβάλλεται στο επίπεδο και είναι ιδιαίτερα σημαντικό για την χαρτογράφηση και πλοήγηση του ρομπότ. Το footprint είναι σημαντικό καθώς οι αλγόριθμοι χαρτογράφησης του πακέτου Nav2, το χρησιμοποιούν για τη δημιουργία νέων χαρτών ή την ανανέωση αυτών, την πλοήγηση και για την αποφυγή εμποδίων.

Στο URDF για το δίτροχο ρομπότ πρέπει να προστεθεί το παρακάτω:

```
<!-- Robot Footprint -->
  <link name="base_footprint">
    <xacro:default_inertia_boxes mass="0" width="0" depth="0" height="0"/>
  </link>

  <joint name="base_joint" type="fixed">
    <parent link="base_link"/>
    <child link="base_footprint"/>
    <origin xyz="0.0 0.0 ${-(radius+0.05)}" rpy="0 0 0"/>
  </joint>
```

Στο παραπάνω καθορίζεται ένα νέο link με όνομα base footprint με μηδενική ροπή αδράνειας. Έπειτα καθορίζεται ένα νέο joint μεταξύ του base link και του base footprint, το οποίο έχει οριστεί να βρίσκεται στο δάπεδο, δηλαδή το footprint του ρομπότ θα είναι ορατό όταν γίνεται η προβολή του επιπέδου.

Έπειτα από τη δημιουργία του URDF, είναι απαραίτητη η τοποθέτηση του ρομποτικού μοντέλου στο Rviz και Gazebo ώστε να εξεταστεί η συμπεριφορά του ρομπότ. Για να επιτευχθεί αυτό, είναι απαραίτητη η δημιουργία ενός launch αρχείου εντός του πακέτου. Το αρχείο που θα εκκινήσει το Rviz και το Gazebo και

θα πετύχει την τοποθέτηση του ρομπότ και στα δύο περιβάλλοντα μπορεί να βρεθεί στον σύνδεσμο: https://github.com/DimitrisKatos/robot_loc_nav/blob/master/launch/display.launch.py.

### 5.3 Εντοπισμός του ρομπότ

Ο εντοπισμός του ρομπότ στο περιβάλλον του είναι ένα αρκετά σύνθετο πρόβλημα, ωστόσο το Nav2 προσφέρει αρκετά εργαλεία που βοηθούν στην επίλυση του προβλήματος. Για τον εντοπισμό του ρομπότ, καθοριστικό ρόλο αποτελεί η οδομετρία αυτού, καθώς μέσω αυτής προσφέρονται ακριβής εκτιμήσεις για την τοποθέτηση αλλά και την ταχύτητα του ενός ρομπότ.

Η οδομετρία του ρομπότ μπορεί να αντληθεί από μια μεγάλη ποικιλία πηγών, δηλαδή των αισθητήρων που φέρει αυτό. Οι αισθητήρες αυτοί είναι κατά κύριο λόγο τα IMU's, τα Radar και οι επενεργητές (encoders) που υπάρχουν στους τροχούς του ρομπότ. Ένα ρομποτικό σύστημα τις περισσότερες φορές, θα φέρει ένα IMU και encoders, οι οποίοι χρησιμοποιούνται ταυτόχρονα για τον εντοπισμό του ρομπότ. Οι δύο αισθητήρες έχουν κάποια αρνητικά χαρακτηριστικά, ωστόσο η ταυτόχρονη δράση τους βοηθάει στην επίλυση του προβλήματος του εντοπισμού του ρομπότ. Πιο συγκεκριμένα, τα IMU's έχουν σφάλματα με το πέρας του χρόνου, ενώ για τους encoders τα σφάλματα εντοπίζονται όταν αυξάνεται η απόσταση που έχει διανύσει το ρομπότ

Με βάση τα παραπάνω, είναι απαραίτητη η προσθήκη ενός IMU στο ρομποτικό σύστημα. Αυτό θα γίνει με την προσθήκη ενός νέου link στο URDF και έπειτα προσθέτοντας ένα νέο Gazebo plugin για την ενεργοποίηση του αισθητήρα στο περιβάλλον προσομοίωσης. Συνεπώς στο URDF προστίθεται το ακόλουθο:

```
<!-- imu link-->
<link name="imu_link">
  <visual>
    <geometry>
      <box size="0.1 0.1 0.1"/>
    </geometry>
  </visual>
  <collision>
    <geometry>
      <box size="0.1 0.1 0.1"/>
    </geometry>
  </collision>
  <xacro:default_inertia_boxes mass="0.1" width="0.1" depth="0.1" height="0.1"/>
</link>
<joint name="imu_joint" type="fixed">
```

```
    <parent link="base_link"/>
    <child link="imu_link"/>
    <origin xyz="0 0 0.01"/>
  </joint>
```

Με το παραπάνω καθορίζεται ένα νέο link, του οποίου η γεωμετρία είναι ένα κύβος. Έπειτα καθορίζεται το collision του, η μάζα και η ροπή αδράνειας του. Έπειτα ορίζεται ένα νέο fixed joint μεταξύ του IMU και του base_link. Για την ενεργοποίηση του IMU στο Gazebo θα δημιουργηθεί ένα νέο αρχείο, το οποίο θα ονομαστεί imu.gazebo, στο οποίο θα ορίζεται το gazebo plugin για τον αισθητήρα. Έπειτα, αυτό το αρχείο θα γίνει include στο βασικό URDF. Το imu.gazebo είναι το ακόλουθο:

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

<robot>
<gazebo reference="imu_link">
 <sensor name="imu_sensor" type="imu">
  <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
    <initial_orientation_as_reference>false</initial_orientation_as_reference>
  </plugin>
  <always_on>true</always_on>
  <update_rate>100</update_rate>
  <visualize>true</visualize>
  <imu>
   <angular_velocity>
    <x>
     <noise type="gaussian">
      <mean>0.0</mean>
      <stddev>2e-4</stddev>
      <bias_mean>0.0000075</bias_mean>
      <bias_stddev>0.0000008</bias_stddev>
     </noise>
    </x>
    <y>
     <noise type="gaussian">
      <mean>0.0</mean>
      <stddev>2e-4</stddev>
      <bias_mean>0.0000075</bias_mean>
      <bias_stddev>0.0000008</bias_stddev>
     </noise>
    </y>
    <z>
```

```
        <noise type="gaussian">
          <mean>0.0</mean>
          <stddev>2e-4</stddev>
          <bias_mean>0.0000075</bias_mean>
          <bias_stddev>0.0000008</bias_stddev>
        </noise>
       </z>
     </angular_velocity>
     <linear_acceleration>
       <x>
        <noise type="gaussian">
          <mean>0.0</mean>
          <stddev>1.7e-2</stddev>
          <bias_mean>0.1</bias_mean>
          <bias_stddev>0.001</bias_stddev>
        </noise>
       </x>
       <y>
        <noise type="gaussian">
          <mean>0.0</mean>
          <stddev>1.7e-2</stddev>
          <bias_mean>0.1</bias_mean>
          <bias_stddev>0.001</bias_stddev>
        </noise>
       </y>
       <z>
        <noise type="gaussian">
          <mean>0.0</mean>
          <stddev>1.7e-2</stddev>
          <bias_mean>0.1</bias_mean>
          <bias_stddev>0.001</bias_stddev>
        </noise>
       </z>
     </linear_acceleration>
   </imu>
  </sensor>
</gazebo>
 </robot>
```

Αφού δημιουργηθεί το imu.gazebo, ακολουθεί η τοποθέτηση του ρομποτικού μοντέλου στο Gazebo και Rviz με χρήση του launch αρχείου που έχει δημιουργηθεί. Έπειτα από την εκκίνηση του launch file, με

κατάλληλη εντολή σε ένα νέο τερματικό, τυπώνεται η λίστα με όλα τα ενεργά topics στο δίκτυο του ROS. Θα παρατηρηθεί το /imu topic και έπειτα μπορούν τα τυπωθούν τα αποτελέσματα του αισθητήρα.

```
$ ros2 topic list

$ ros2 topic echo /imu
```

Πίνακας 99. Αποτελέσματα του imu.

Αφού έχει γίνει η προσθήκη του IMU και της οδομετρίας του ρομπότ, τώρα μπορεί να πραγματοποιηθεί ο εντοπισμός του ρομπότ στο περιβάλλον του. Στο ROS 2 προσφέρεται το robot localization πακέτο, το οποίο χρησιμοποιείται για να προσφέρει μια ακριβής ενημέρωση για την οδομετρία του ρομπότ , λαμβάνοντας υπόψη του τους αισθητήρες του.

Όταν πολλές πηγές δεδομένων προσφέρονται στο πακέτο robot localization, τότε αυτό με κατάλληλες διεργασίες μπορεί να συνδυάσει τα δεδομένα που του παρέχονται μέσω κάποιων estimation nodes που θα εκκινήσουν. Τα nodes αυτά χρησιμοποιούν είτε το Extended Kalman Filter είτε το Unscented Kalman Filter για τον εντοπισμό του ρομπότ.

Το πακέτο robot localization δεν εγκαθίσταται μαζί με το πακέτο Nav2 και για αυτό είναι απαραίτητο να δοθεί εντολή για την εγκατάσταση του.

```
$ sudo apt install ros-humble-robot-localization
```

Πίνακας 100. Εγκατάσταση του robot_localization.

Αφού επιτευχθεί η εγκατάσταση του πακέτου, τότε θα πρέπει να καθοριστούν οι παράμετροι του node για την εκτίμηση του ρομπότ. Για τον καθορισμό των παραμέτρων θα δημιουργηθεί ένα νέο αρχείο τύπου yaml, με το όνομα ekf_filter.yaml και θα αποθηκευτεί στο φάκελο config του πακέτου. Το αρχείο είναι το ακόλουθο:

```
### ekf config file ###
ekf_filter_node:
   ros__parameters:
      frequency: 30.0
      two_d_mode: false
      publish_acceleration: true
      publish_tf: true
      map_frame: map             # Defaults to "map" if unspecified
      odom_frame: odom           # Defaults to "odom" if unspecified
```

```
    base_link_frame: base_link  # Defaults to "base_link" if unspecified
    world_frame: odom           # Defaults to the value of odom_frame if unspecified
    odom0: /odom
    odom0_config: [true,  true,  true,
                   false, false, false,
                   false, false, false,
                   false, false, true,
                   false, false, false]
    imu0: /imu
    imu0_config: [false, false, false,
                  true,  true,  true,
                  false, false, false,
                  false, false, false,
                  false, false, false]
```

Στο παραπάνω configuration file καθορίζονται οι τιμές κάποιων παραμέτρων του node που θα εκκινήσει. Οι παράμετροι αυτοί είναι το frequency, two_d_mode, publish_acceleration, publish_tf, map_fram, base_link_frame, odom_frame και world_frame. Για τη λήψη δεδομένων από τους αισθητήρες, πρέπει να καθοριστούν και δύο επιπλέον παράμετροι, που είναι το odom0 και imu0. Η τιμή κάθε παραμέτρου είναι το αντίστοιχο topic που κάνει publish τα δεδομένα του ο αισθητήρας.

Στις δύο τελευταίες παραμέτρους του αρχείου, παρατηρείται ο ορισμός ενός πίνακα 5x3. Στους πίνακες καθορίζονται ποιες τιμές των αισθητήρων θα χρησιμοποιηθούν. Για παράδειγμα στον odom0_config, παρατηρείται ότι όλες οι τιμές στην πρώτη γραμμή είναι true. Με αυτό δηλώνεται ότι από τον αισθητήρα οδομετρίας χρησιμοποιούνται οι παράμετροι x, y και z που λαμβάνονται από αυτόν. Η δεύτερη γραμμή του imu0_config έχει την τιμή true, δηλαδή από το IMU χρησιμοποιούνται οι παράμετροι roll, pitch και yaw.

Αφού καθοριστούν οι παράμετροι του estimation node, τότε θα πρέπει αυτό να οριστεί στο launch file του πακέτου για να εκκινήσει με όλα τα υπόλοιπα nodes. Επιπλέον θα χρησιμοποιηθεί το use sim time argument, που θα βοηθήσει στο συγχρονισμό των nodes που θα εκκινήσουν. Συνεπώς στο launch αρχείο θα πρέπει να προστεθούν τα ακόλουθα:

```
robot_localization_node = Node( package='robot_localization',  executable='ekf_node',
    name='ekf_filter_node', output='screen',
    parameters=[os.path.join(robot_path, 'config/ekf_filter.yaml'),
                {'use_sim_time': LaunchConfiguration('use_sim_time')}])

sim_time= DeclareLaunchArgument(name='use_sim_time', default_value='True',
                                description='Flag to enable use_sim_time')
```

Έπειτα από την δημιουργία των παραπάνω, αυτά θα πρέπει να προστεθούν στο LaunchDescription μαζί με όλα τα nodes και arguments που θα εκκινήσουν. Στη συνέχεια θα πρέπει να χτιστεί το πακέτο και τέλος να εκκινήσει το launch αρχείο με τα επιπρόσθετα nodes. Σε ένα νέο τερματικό θα εκτελεστούν τα ακόλουθα και η τελευταία εντολή τυπώνει την τοποθέτηση του ρομπότ στο περιβάλλον του.

```
$ cd ~/ros2_ws/

$ colcon build --packages-select robot_loc_nav

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch robot_loc_nav display.launch.py

$ ros2 run tf2_ros tf2_echo odom base_link # New terminal.
```

Πίνακας 101. Εκκίνηση του εκτελέσιμου tf2_echo.

Έπειτα από την εκτέλεση της παραπάνω εντολής, τυπώνονται τα δεδομένα που λαμβάνονται από τον αισθητήρα οδομετρίας και το IMU. Συνεπώς με αυτό το τρόπο μπορεί να εντοπιστεί το ρομπότ στο περιβάλλον του. Σε αυτό το σημείο, ενδιαφέρον εντοπίζεται στη μεταβολή των τιμών όταν το ρομπότ αποκτήσει κάποια ταχύτητα.

```
katos@KATOS:~$ ros2 run tf2_ros tf2_echo odom base_link
[INFO] [1697615380.198966159] [tf2_echo]: Waiting for transform odom ->  base_link: Invalid frame
 ID "odom" passed to canTransform argument target_frame - frame does not exist
At time 6.608000000
- Translation: [0.000, -0.000, 0.150]
- Rotation: in Quaternion [0.000, -0.000, -0.000, 1.000]
- Rotation: in RPY (radian) [0.000, -0.000, -0.000]
- Rotation: in RPY (degree) [0.000, -0.007, -0.026]
- Matrix:
  1.000  0.000 -0.000  0.000
 -0.000  1.000 -0.000 -0.000
  0.000  0.000  1.000  0.150
  0.000  0.000  0.000  1.000
At time 7.458000000
- Translation: [0.000, -0.000, 0.150]
- Rotation: in Quaternion [0.000, -0.000, -0.000, 1.000]
- Rotation: in RPY (radian) [0.000, -0.000, -0.000]
- Rotation: in RPY (degree) [0.000, -0.007, -0.027]
- Matrix:
  1.000  0.000 -0.000  0.000
 -0.000  1.000 -0.000 -0.000
  0.000  0.000  1.000  0.150
  0.000  0.000  0.000  1.000
```

Εικόνα 60. Τα αποτελέσματα του localization για το ρομπότ.

## 5.4 Προσθήκη κάμερας βάθους

Όπως και στον εντοπισμό του ρομπότ, έτσι και στη χαρτογράφηση του περιβάλλοντος του ρομπότ, καθοριστικό ρόλο αποτελούν οι αισθητήρες που φέρει αυτό. Μέσω των αισθητήρων λαμβάνονται πληροφορίες που συντελούν στη δημιουργία ή στην αναθεώρηση του χάρτη του περιβάλλοντος του ρομπότ. Παράλληλα οι αισθητήρες αυτοί βοηθούν στον εντοπισμό του ρομπότ και με τη χρήση τους το ρομπότ μπορεί να αποφύγει εμπόδια και να εκτελέσει διάφορες αποστολές. Η χαρτογράφηση και η αναθεώρηση του χάρτη είναι απαραίτητες διεργασίες για την αποτελεσματική και ασφαλή πλοήγηση του ρομπότ σε δυναμικά και μη δυναμικά περιβάλλοντα.

Οι πιο συχνοί αισθητήρες που μπορεί να φέρει ένα ρομπότ είναι τα IMU, τα GPS, τα lidar, οι κάμερες και οι encoders των τροχών. Οι αισθητήρες αυτοί συνήθως δημοσιεύουν τα δεδομένα τους σε κάποια topics. Η πληροφορία που δημοσιεύεται στα topics έχει ένα συγκεκριμένο τύπο μηνύματος που διαφέρει ανάλογα με τον αισθητήρα. Το ROS προσφέρει το sensor_msgs πακέτο, μέσω του οποίου καθορίζονται συγκεκριμένου τύπου μηνύματα που διασφαλίζουν την ευκολότερη διεπαφή του ROS με του αισθητήρες του. Συνεπώς μπορεί να χρησιμοποιηθεί οποιοσδήποτε αισθητήρας είναι επιθυμητό αρκεί να ακολουθείτε το βασικό format του πακέτου.

Σε ένα πραγματικό ρομπότ, οι αισθητήρες που θα προστεθούν στο ρομπότ θα έχουν κάποιους έτοιμους ROS drivers που ακολουθούν τις βασικές διεπαφές του πακέτου sensor_msgs. Δηλαδή υπάρχουν κάποια έτοιμα nodes τα οποία επικοινωνούν με τους αισθητήρες, μετατρέπουν τα δεδομένα σε μηνύματα και τα δημοσιεύουν σε κάποια topics που θα χρησιμοποιούνται από το ρομπότ. Τα nodes αυτά χρησιμοποιούν καταλλήλως το πακέτο sensor_msgs για να δημοσιεύσουν την πληροφορία με τον κατάλληλο τύπο μηνύματος ανάλογα με τον αισθητήρα που χρησιμοποιείται. Τα nodes αυτά συνήθως έχουν δημιουργηθεί από άλλους χρήστες τους ROS, κάνοντας ευκολότερη την εκ νέου χρήση τους. Συνεπώς το sensor_msgs πακέτο διευκολύνει την χρήση διάφορων αισθητήρων από διάφορους κατασκευαστές.

Όσον αφορά τις προσομοιώσεις στο περιβάλλον προσομοίωσης του Gazebo υπάρχουν τα Gazebo Plugins. Αυτά τα sensor plugins πρέπει να προστεθούν στο URDF μοντέλο και δημιουργούν nodes για τον επιθυμητό αισθητήρα και χρησιμοποιούν διεπαφές του πακέτου sensor_msgs.

Για τη δημιουργία ενός χάρτη του ρομπότ, ιδιαίτερα χρήσιμη είναι η προσθήκη μιας κάμερας βάθους στο ρομποτικό μοντέλο. Για την προσθήκη της κάμερας είναι απαραίτητη η δημιουργία ενός νέου link, η προσθήκη ενός νέου joint και η ανάπτυξη ενός gazebo sensor plugin για την ενεργοποίηση της κάμερας βάθους. Στο ρομποτικό μοντέλο πρέπει να προστεθεί το παρακάτω:

```
<!-- Camera Link-->
  <link name="camera_link">
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.015 0.130 0.022"/>
      </geometry>
    </visual>

    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.015 0.130 0.022"/>
      </geometry>
    </collision>

    <inertial>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <mass value="0.035"/>
      <inertia ixx="0.001"  ixy="0"  ixz="0" iyy="0.001" iyz="0" izz="0.001" />
    </inertial>
  </link>

  <joint name="camera_joint" type="fixed">
    <parent link="base_link"/>
    <child link="camera_link"/>
    <origin xyz="0.215 0 0.05" rpy="0 0 0"/>
  </joint>

  <link name="camera_depth_frame"/>
  <joint name="camera_depth_joint" type="fixed">
    <origin xyz="0 0 0" rpy="${-pi/2} 0 ${-pi/2}"/>
    <parent link="camera_link"/>
    <child link="camera_depth_frame"/>
  </joint>
```

Με τον παραπάνω κώδικα, δημιουργείται ένα νέο link για την κάμερα και έπειτα ορίζεται το joint της κάμερας με το base_link. Στη συνέχεια, δημιουργείται ένα νέο αρχείο με το όνομα depth_camera.gazebo και αποθηκεύεται στο φάκελο urdf του πακέτου. Μέσω αυτού ενεργοποιείται η κάμερα βάθους στο Gazebo. Το plugin είναι το ακόλουθο:

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

<robot>
   <gazebo reference="camera_link">
  <sensor name="depth_camera" type="depth">
   <visualize>true</visualize>
   <update_rate>30.0</update_rate>
   <camera name="camera">
    <horizontal_fov>1.047198</horizontal_fov>
    <image>
     <width>640</width>
     <height>480</height>
     <format>R8G8B8</format>
    </image>
    <clip>
     <near>0.05</near>
     <far>3</far>
    </clip>
   </camera>
   <plugin name="depth_camera_controller" filename="libgazebo_ros_camera.so">
    <baseline>0.2</baseline>
    <alwaysOn>true</alwaysOn>
    <updateRate>0.0</updateRate>
    <frame_name>camera_depth_frame</frame_name>
    <pointCloudCutoff>0.5</pointCloudCutoff>
    <pointCloudCutoffMax>3.0</pointCloudCutoffMax>
    <distortionK1>0</distortionK1>
    <distortionK2>0</distortionK2>
    <distortionK3>0</distortionK3>
    <distortionT1>0</distortionT1>
    <distortionT2>0</distortionT2>
    <CxPrime>0</CxPrime>
    <Cx>0</Cx>
    <Cy>0</Cy>
    <focalLength>0</focalLength>
    <hackBaseline>0</hackBaseline>
   </plugin>
  </sensor>
</gazebo>

</robot>
```

Έπειτα από τη δημιουργία του παραπάνω αρχείου, αυτό θα πρέπει να γίνει include στο URDF του μοντέλου. Θα χρησιμοποιηθεί η μέθοδος xacro:include με τρόπο που φαίνεται παρακάτω:

```
<!-- Import Gazebo plugin for Depth Camera-->
<xacro:include filename="$(find robot_loc_nav)/urdf/depth_camera.gazebo"/>
```

Έπειτα και από τα παραπάνω θα πρέπει να ελεγχθεί η χρήση της κάμερας βάθους στο Gazebo και για αυτό εκκινεί το launch αρχείο με το οποίο τοποθετείται το ρομποτικό μοντέλο στο Gazebo και Rviz. Ωστόσο σε αυτό το σημείο είναι προτιμότερο να εκκινήσει η προσομοίωση σε ένα κόσμο με αντικείμενα γύρω του. Συνεπώς θα πρέπει να δημιουργηθεί ένας νέος κόσμος στο Gazebo, ο οποίος θα περιέχει το επίπεδο, φωτισμό και ένα κώνο ή οποιοδήποτε άλλο αντικείμενο είναι επιθυμητό. Δηλαδή θα δημιουργηθεί ένα νέο αρχείο με όνομα my_world.world στο φάκελο worlds του πακέτου.

Έπειτα θα πρέπει να πραγματοποιηθούν κάποιες αλλαγές στο launch αρχείο που θα εκκινήσει. Θα προστεθεί ένα νέο module μιας βιβλιοθήκης, θα ακολουθήσει ο ορισμός ενός path για να εντοπίζεται το αρχείο που βρίσκεται ο κόσμος. Τέλος θα προστεθεί ένα ExecuteProcess στο LaunchDescription.

Το module της βιβλιοθήκης που πρέπει να προστεθεί είναι το ακόλουθο:

```
from launch.actions import DeclareLaunchArgument,ExecuteProcess
```

Το μονοπάτι που θα προστεθεί είναι το παρακάτω εντός της συνάρτησης generate_launch_description:

```
world_path=os.path.join(robot_path, 'worlds/my_world.world'),
```

Στο LaunchDescription θα προστεθεί το ακόλουθο, ώστε να εκτελεστεί ταυτόχρονα με τα άλλα nodes ή arguments:

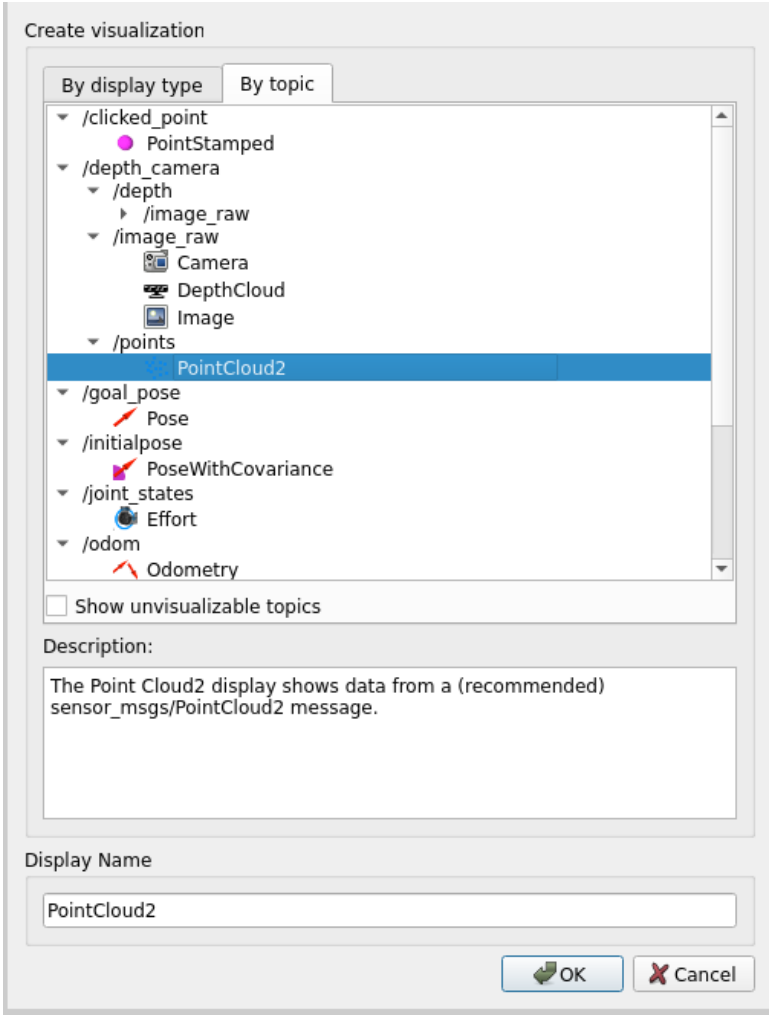
```
ExecuteProcess(cmd=['gazebo', '--verbose', '-s', 'libgazebo_ros_init.so', '-s', 'libgazebo_ros_factory.so',
world_path], output='screen'),
```

Αφού ολοκληρωθούν τα παραπάνω βήματα, πρέπει να εκκινήσει το launch αρχείο το οποίο θα τοποθετεί το δίτροχο ρομπότ σε ένα κόσμο του gazebo ο οποίος δεν θα είναι κενός . Θα πρέπει να εκτελεστούν οι παρακάτω εντολές:

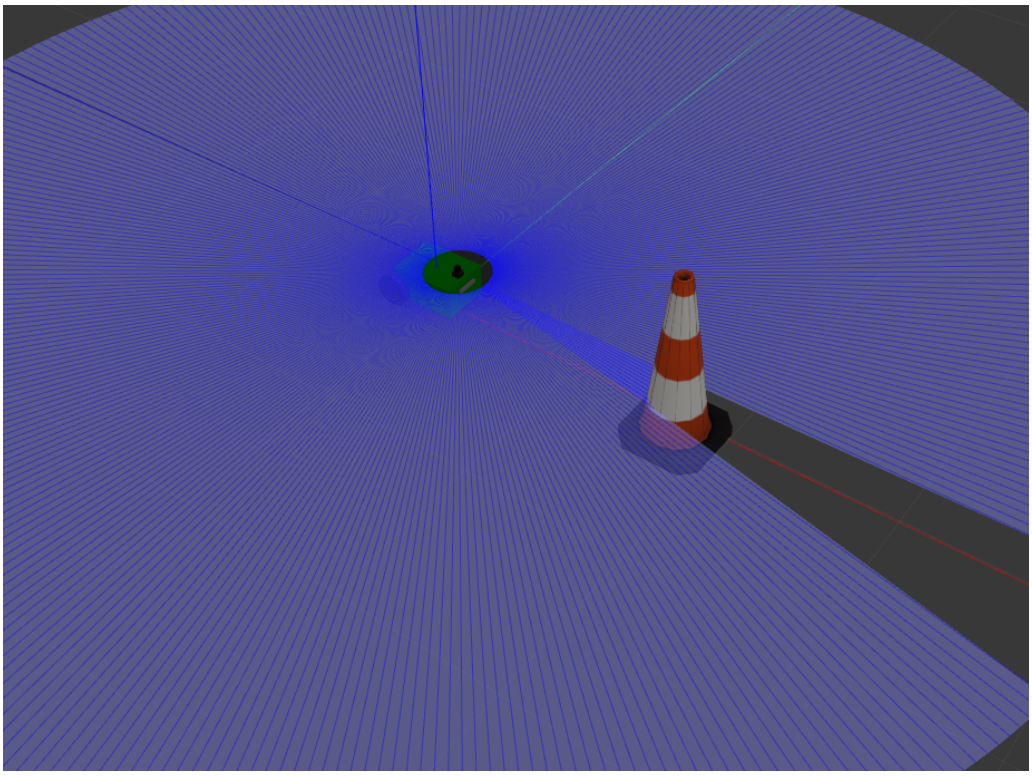
```
$ cd ~/ros2_ws

$ colcon build --packages-select robot_loc_nav

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch robot_loc_nav display.launch.py
```

Πίνακας 102. Τοποθέτηση του robot_loc_nav σε ένα νέο κόσμο.

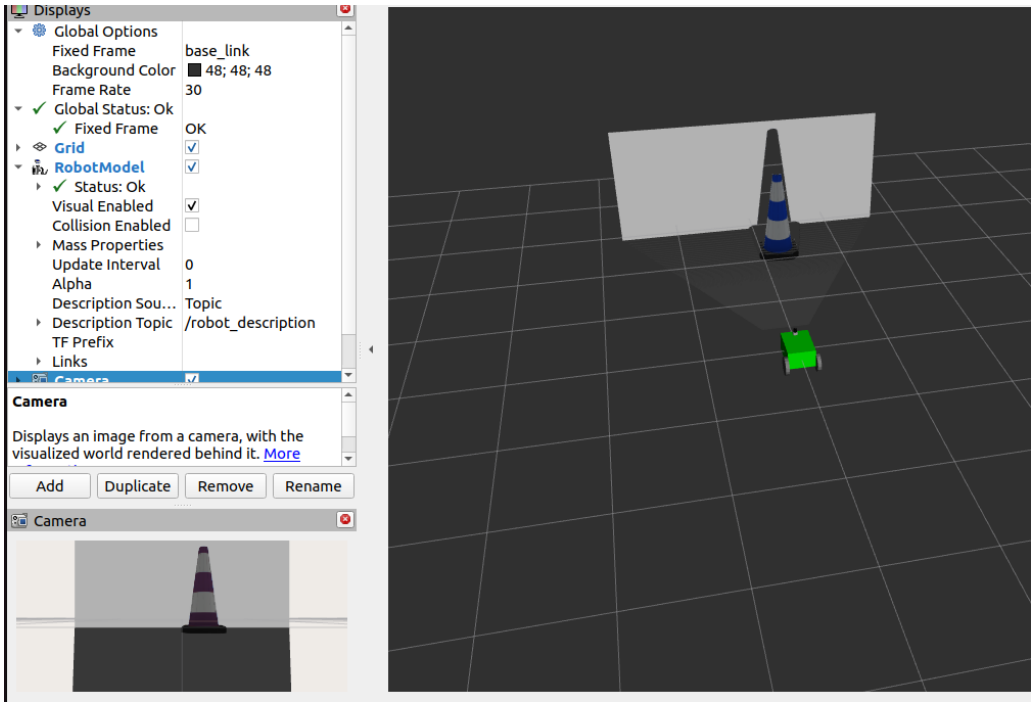
Με την εκκίνηση του launch file, θα εκκινήσουν το Rviz και Gazebo. Στο Rviz θα γίνει η προσθήκη του ρομποτικού μοντέλου και έπειτα το feature PoinCloud2 όπως φαίνεται στη παρακάτω εικόνα. Με αυτό το τρόπο θα γίνει η είσοδος της κάμερας βάθους στο Rviz και θα είναι ορατά τα αποτελέσματα της.




Εικόνα 61. Προσθήκη του  Point Cloud2 στο Rviz.



Εικόνα 62. Το robot_loc_nav στο Gazebo.




Εικόνα 63. Η depth camera στο Rviz.

### 5.5 Χαρτογράφηση του ρομπότ

Η δημιουργία του χάρτη ενός ρομπότ είναι καταλυτικής σημασίας καθώς μέσω του χάρτη μπορεί να υπάρξει καλύτερη γνώση για την τοποθέτηση του ρομπότ αλλά και για την πλοήγηση αυτού. Πριν την δημιουργία του χάρτη είναι απαραίτητη η δημιουργία ενός configuration file, στο οποίο θα ορίζονται οι παράμετροι με τους οποίους θα δημιουργείται ο χάρτης. Το αρχείο θα είναι σε μορφή yaml και θα αποθηκευτεί στον φάκελο config με το όνομα nav2_params.yaml.

```
slam_toolbox:
  ros__parameters:
    # Plugin params
    solver_plugin: solver_plugins::CeresSolver
    ceres_linear_solver: SPARSE_NORMAL_CHOLESKY
    ceres_preconditioner: SCHUR_JACOBI
    ceres_trust_strategy: LEVENBERG_MARQUARDT
    ceres_dogleg_type: TRADITIONAL_DOGLEG
    ceres_loss_function: None

    # ROS Parameters
    odom_frame: odom
    map_frame: map
    base_frame: base_footprint
    scan_topic: /scan
    use_map_saver: true
    mode: mapping #localization

    #map_file_name: test_steve
    # map_start_pose: [0.0, 0.0, 0.0]
    #map_start_at_dock: true

    debug_logging: false
    throttle_scans: 1
    transform_publish_period: 0.02 #if 0 never publishes odometry
    map_update_interval: 5.0
    resolution: 0.05
    max_laser_range: 20.0 #for rastering images
    minimum_time_interval: 0.5
    transform_timeout: 0.2
    tf_buffer_duration: 30.
    stack_size_to_use: 40000000 #// program needs a larger stack size to serialize large maps
    enable_interactive_mode: true
```

```
# General Parameters
use_scan_matching: true
use_scan_barycenter: true
minimum_travel_distance: 0.5
minimum_travel_heading: 0.5
scan_buffer_size: 10
scan_buffer_maximum_scan_distance: 10.0
link_match_minimum_response_fine: 0.1
link_scan_maximum_distance: 1.5
loop_search_maximum_distance: 3.0
do_loop_closing: true
loop_match_minimum_chain_size: 10
loop_match_maximum_variance_coarse: 3.0
loop_match_minimum_response_coarse: 0.35
loop_match_minimum_response_fine: 0.45

# Correlation Parameters - Correlation Parameters
correlation_search_space_dimension: 0.5
correlation_search_space_resolution: 0.01
correlation_search_space_smear_deviation: 0.1

# Correlation Parameters - Loop Closure Parameters
loop_search_space_dimension: 8.0
loop_search_space_resolution: 0.05
loop_search_space_smear_deviation: 0.03

# Scan Matcher Parameters
distance_variance_penalty: 0.5
angle_variance_penalty: 1.0
fine_search_angle_offset: 0.00349
coarse_search_angle_offset: 0.349
coarse_angle_resolution: 0.0349
minimum_angle_penalty: 0.9
minimum_distance_penalty: 0.5
use_response_expansion: true
```

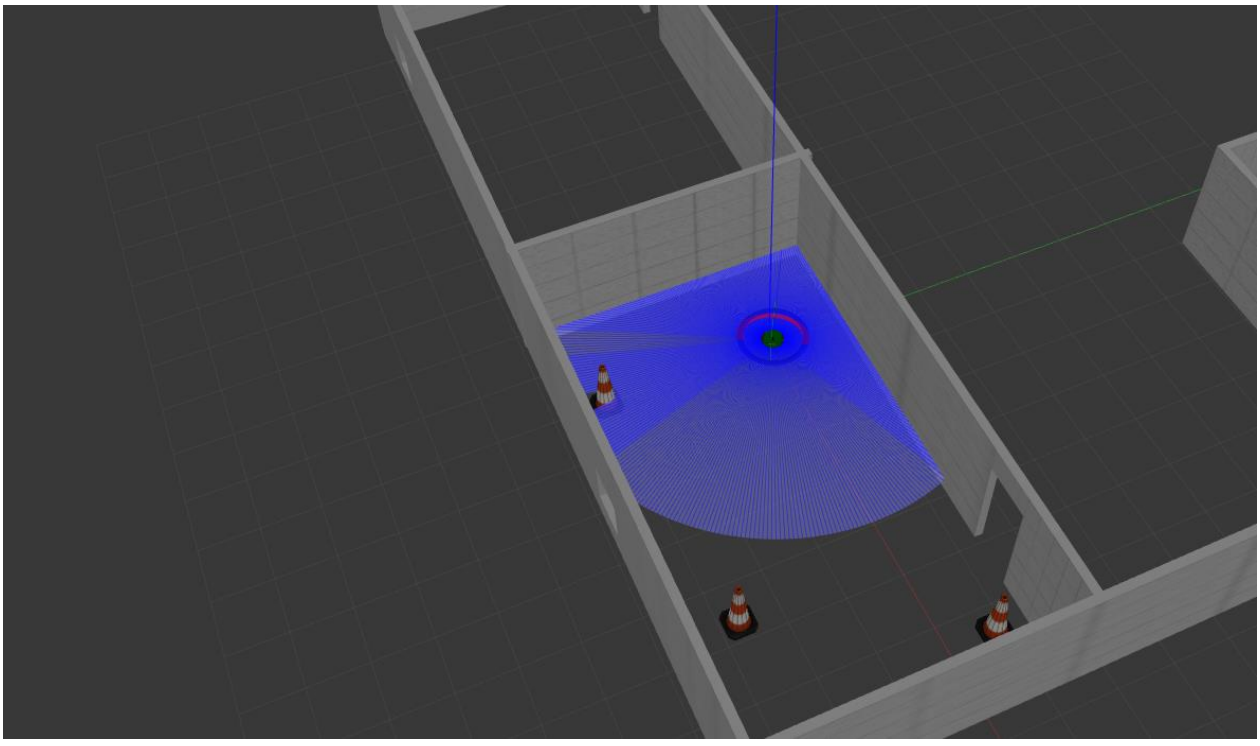
Οι σημαντικότεροι παράμετροι του παραπάνω config file, είναι αυτοί που ορίζονται στο ROS parameters. Εκεί πρέπει να οριστούν κατάλληλα η οδομετρία του ρομπότ, το footprint του, το topic του lidar και το topic του χάρτη. Στη συνέχεια θα πρέπει να εκκινήσει το launch file που έχει δημιουργηθεί έως τώρα και έπειτα να εκκινήσει το slam_toolbox με χρήση των παραμέτρων που ορίστηκαν παραπάνω ώστε να ελεγχθεί η συμπεριφορά του ρομπότ στο περιβάλλον προσομοίωσης.

Στο σημείο αυτό είναι επιθυμητή η τοποθέτηση του ρομπότ σε ένα περιβάλλον που υπάρχουν αρκετά εμπόδια και κυρίως τοίχοι. Μέσω του building editor του Gazebo, δημιουργείται ένα σπίτι στο οποίο θα τοποθετηθεί το ρομπότ. Έπειτα το κτίριο που θα δημιουργηθεί αποθηκεύεται στο φάκελο world του πακέτου. Στη συνέχεια γίνεται το χτίσιμο του πακέτου και πραγματοποιείται η τοποθέτηση του ρομπότ στο Gazebo. Τέλος, σε ένα νέο τερματικό εκτελούνται οι παρακάτω εντολές που εκκινούν ένα launch αρχείο του πακέτου slam toolbox.

```
$ cd ~/ros2_ws

$ ros2 launch slam_toolbox online_async_launch.py
params_file:=src/robot_loc_nav/config/nav2_params.yaml use_sim_tim:=true
```

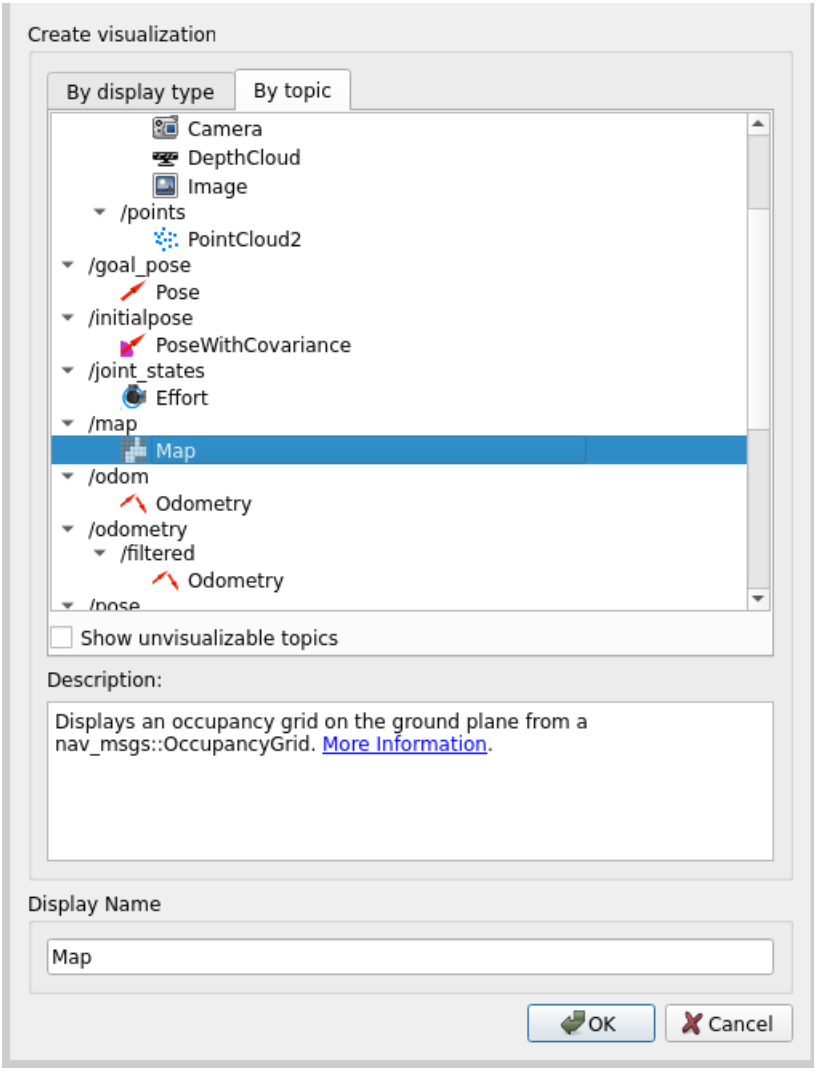
Πίνακας 103. Εκκίνηση του node slam toolbox.

Με την παραπάνω εντολή εκκινεί ένα νέο launch αρχείο το οποίο ανήκει στο πακέτο slam toolbox και αυτό δέχεται δύο παραμέτρους. Αρχικά δέχεται το configuration file που δημιουργήθηκε παραπάνω και έπειτα γίνεται χρήση του use_sim_time για τον συγχρονισμό του με το Gazebo και Rviz.

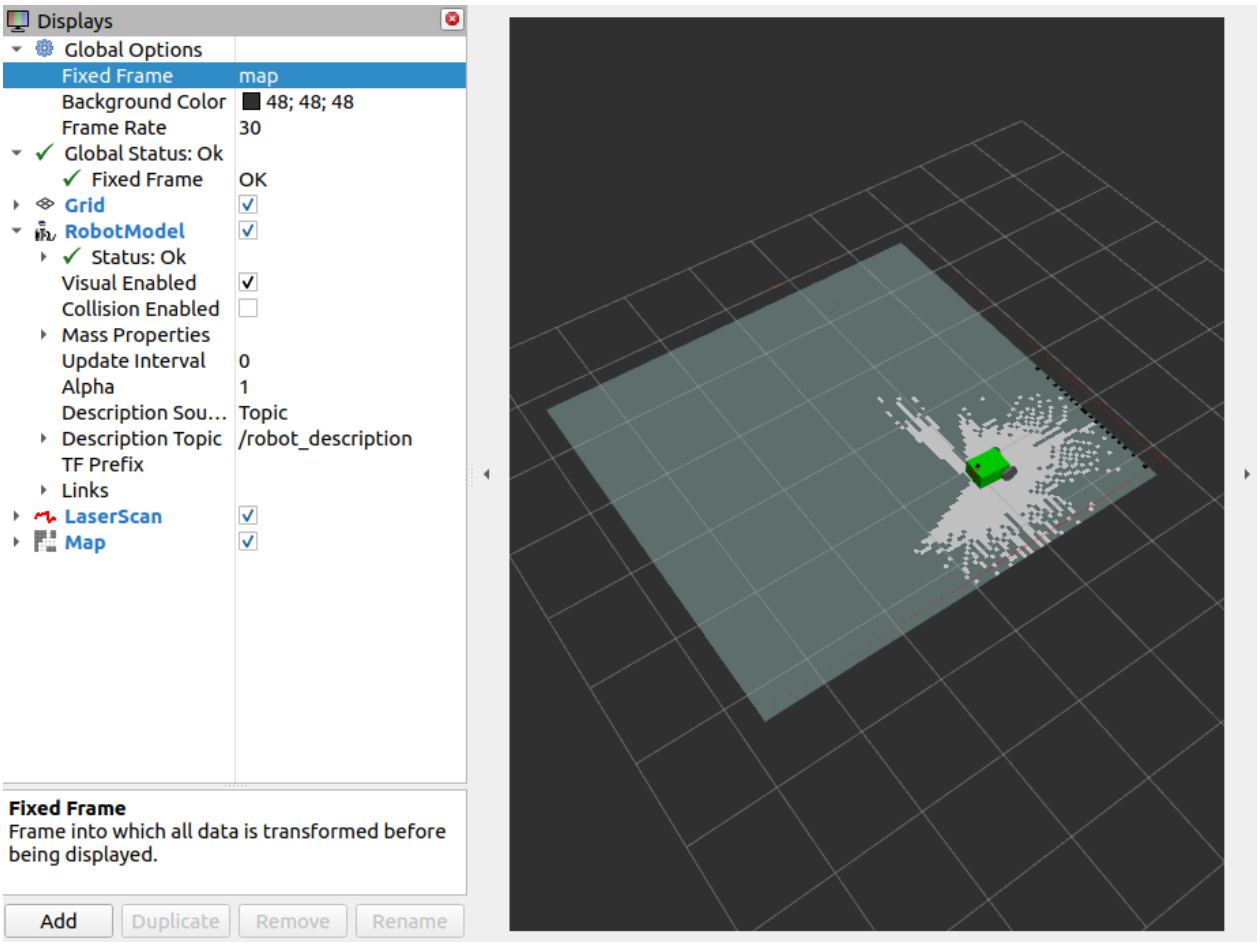


Εικόνα 64. Το ρομπότ σε ένα σπίτι.

Έπειτα από την εκκίνηση των launch files, γίνεται η μετάβαση στο Rviz. Εκεί πρέπει να προστεθούν το ρομποτικό μοντέλο στο τρισδιάστατο χώρο καθώς και το lidar του ρομπότ. Στη συνέχεια θα προστεθεί ένα νέο feature στο περιβάλλον του Rviz και αυτό είναι το map, μέσω του οποίου θα προβάλλεται ο χάρτης του περιβάλλοντος του ρομπότ που έχει δημιουργηθεί. Για την προσθήκη του επιλέγεται το Add που βρίσκεται στο Display Panel, έπειτα επιλέγεται το By Topic στη κορυφή του παραθύρου και τέλος γίνεται η επιλογή του feature **Map**.



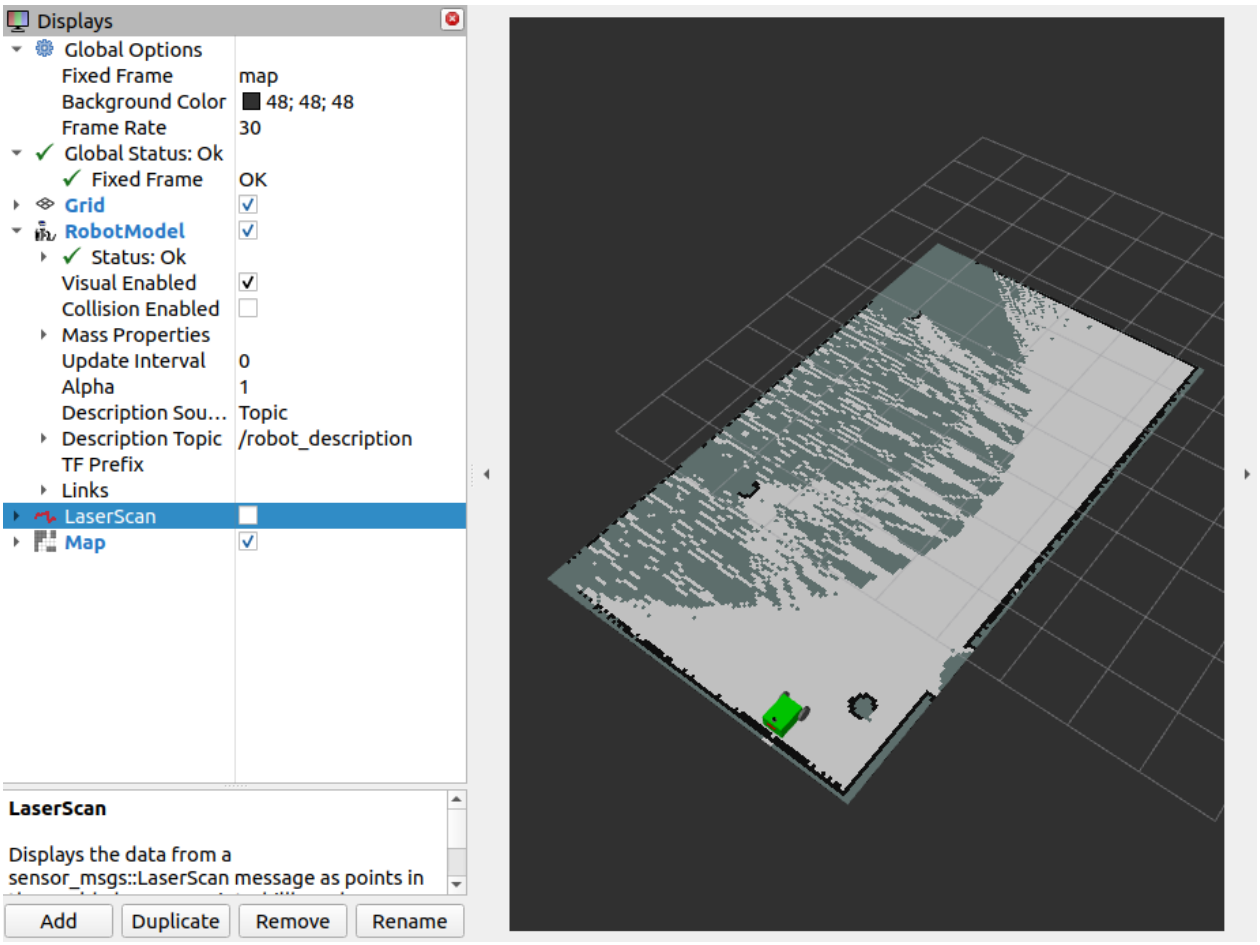

Εικόνα 65. Προσθήκη του Feature Map.




Εικόνα 66. Η χαρτογράφηση του ρομπότ.

Αφού προστεθεί το παραπάνω feature, θα πρέπει να παρατηρηθεί η δημιουργία ενός χάρτη στο τρισδιάστατο περιβάλλον του Rviz, με τρόπο που φαίνεται στη παραπάνω εικόνα. Στη συνέχεια, δίνεται μια επιθυμητή ταχύτητα στο ρομπότ ή εκκινεί ο αλγόριθμος αποφυγής εμποδίων του προηγούμενου κεφαλαίου. Μέσω της δήλωσης επιθυμητής ταχύτητας δίνεται η δυνατότητα στο ρομπότ να εξερευνήσει το υπόλοιπο κτίριο στο οποίο βρίσκεται και να συνεχίσει τη δημιουργία και την ανανέωση του χάρτη του.

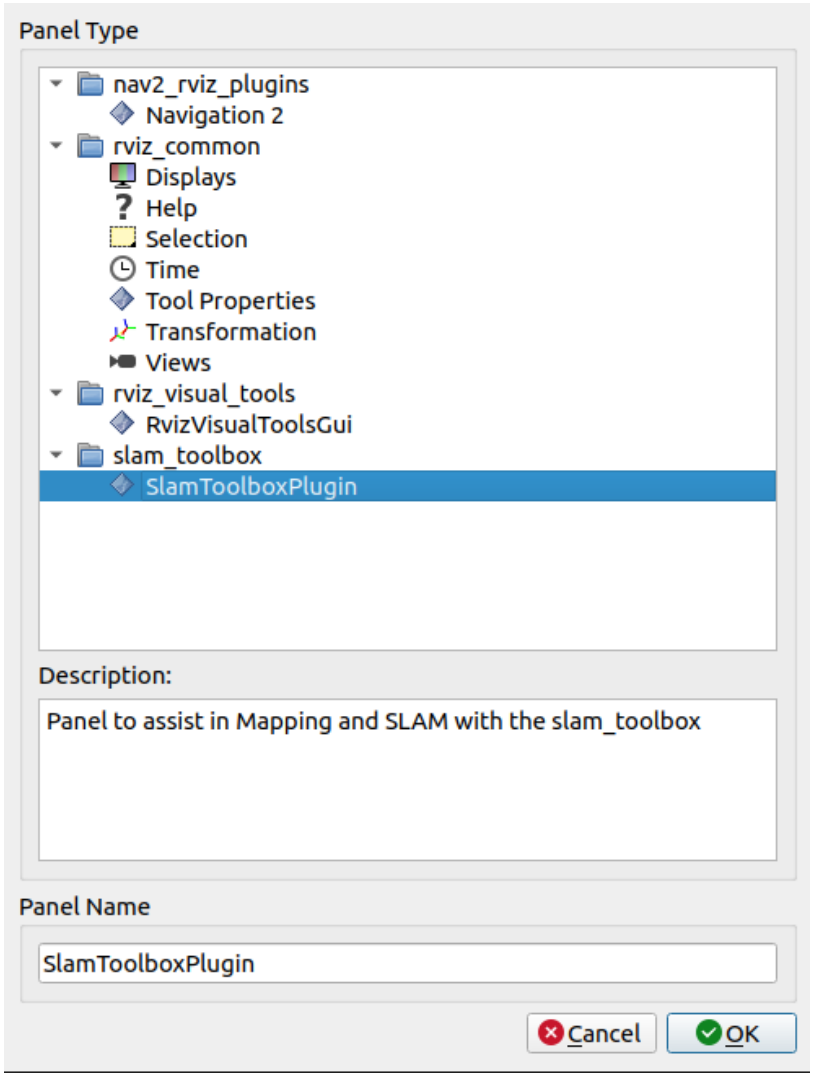
Έπειτα από την δημοσίευση ταχύτητας στο κατάλληλο topic, πλέον παρατηρείται ότι ο χάρτης ανανεώνεται, και το ρομπότ αντιλαμβάνεται περισσότερα από τα εμπόδια που βρίσκονται στο περιβάλλον του. Άξιο αναφοράς, είναι το γεγονός ότι στο Fixed Frame, που βρίσκεται στο display panel, έχει γίνει η επιλογή του map εν αντιθέση με το base_link. Η επιλογή αυτή γίνεται, ώστε να παρατηρείται η κίνηση του ρομπότ και ο χάρτης να παραμένει σταθερός και να ανανεώνεται.




Εικόνα 67. Η χαρτογράφηση έπειτα από την κίνηση του ρομπότ.

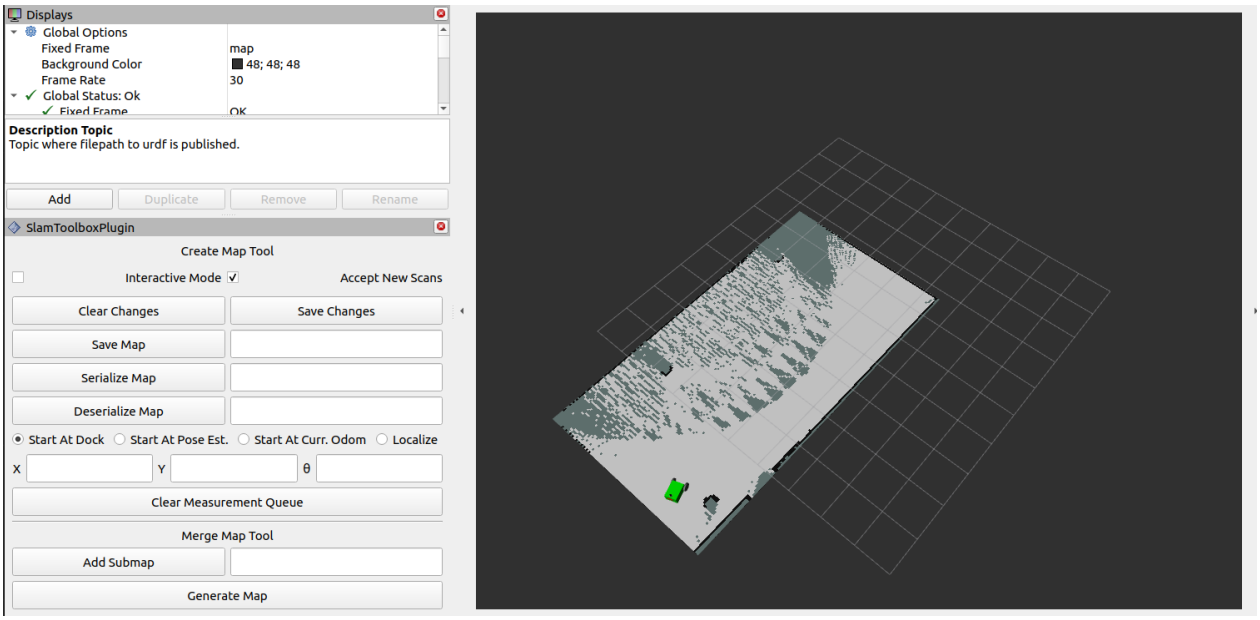
Αφού έχει δημιουργηθεί ένα κομμάτι του χάρτη για το ρομπότ, τότε είναι ιδιαίτερα σημαντικό να αποθηκευτεί αυτός με στόχο την μελλοντική επαναχρησιμοποίηση του. Η δημιουργία ενός χάρτη για το ρομπότ είναι αρκετά απαιτητική, διότι χρησιμοποιούνται πολλοί υπολογιστικοί πόροι, με αποτέλεσμα αυτός να είναι απαραίτητο να αποθηκευτεί. Με την εκ νέου χρήση του χάρτη το ρομπότ, θα γνωρίζει ένα μέρος του χώρου εργασίας του και θα μπορεί να τον βελτιώσει με εκ νέου εξερεύνηση στο περιβάλλον του.

Το slam_toolbox πακέτο προσφέρει ένα πολύ σημαντικό εργαλείο για την αποθήκευση του χάρτη που έχει δημιουργήσει το ρομπότ. Πιο συγκεκριμένα, μέσω του περιβάλλοντος του Rviz προσφέρει ένα panel, μέσω του οποίου μπορεί να γίνει η αποθήκευση του χάρτη εντός του Rviz. Για την προσθήκη του συγκεκριμένου panel, επιλέγεται η επιλογή Panels που βρίσκεται στην επιλογές παραθύρου του Rviz και επιλέγεται το Add Panel. Έπειτα θα εκκινήσει ένα παράθυρο με όλα τα διαθέσιμα panels, όπως φαίνεται στην παρακάτω εικόνα.




Εικόνα 68. Προσθήκη του Panel, SlamToolboxPlugin.

Από τα διαθέσιμα panels, επιλέγεται το SlamToolboxPlugin και έπειτα επιλέγεται το OK. Με αυτό το τρόπο στο Display Panel του Rviz θα προστεθεί ένα νέο Feature. Μέσω αυτού μπορεί να αποθηκεύονται οι χάρτες του ρομπότ σε διάφορα format και χρησιμοποιώντας κάποιες ακόμη ρυθμίσεις.




Εικόνα 69. Το SlamToolbox panel.

Στην παραπάνω εικόνα, παρατηρείται η προσθήκη του SlamToolbox μέσω του οποίου μπορεί να γίνει η αποθήκευση του χάρτη. Δίπλα από τις επιλογές Save Map και Serialize Map, πληκτρολογείτε το όνομα που είναι επιθυμητό να ονομαστεί ο χάρτης. Το όνομα που θα δοθεί μπορεί να είναι το ίδιο και στα δύο πλαίσια. Σε περίπτωση που δοθεί κοινό όνομα στους δύο χάρτες, αυτή η κοινή ονοματοδοσία δεν αποτελεί πρόβλημα, καθώς θα αποθηκευτούν οι χάρτες με διαφορετικά format.

Αφού δοθούν τα ονόματα, γίνεται η επιλογή των πλαισίων Save Map και Serialize Map. Με αυτό το τρόπο αποθηκεύονται οι χάρτες με δύο διαφορετικά format. Τα αρχεία αυτά αποθηκεύονται εντός του workspace που βρίσκεται το πακέτο. Ωστόσο, είναι καλό να δημιουργηθεί ένας νέος φάκελος στο πακέτο, με το όνομα maps, και οι χάρτες να μεταφερθούν εκεί. Η επιλογή Save Map, αποθηκεύει το χάρτη σε format το οποίο δεν μπορεί να χρησιμοποιηθεί ξανά για την φόρτωση του στο Rviz. Εν αντίθεση η επιλογή Serialize map, αποθηκεύει το χάρτη κατάλληλα ώστε να μπορεί να γίνει επαναχρησιμοποίηση του μελλοντικά.

## 5.6 Επαναχρησιμοποίηση του χάρτη

Στην ενότητα αυτή θα παρουσιαστούν οι διαδικασίες που απαιτούνται για την φόρτωση του χάρτη στο Rviz. Επιπλέον, θα παρουσιαστεί ο τρόπος που μπορεί να γίνει ο εντοπισμός του ρομπότ στο χάρτη, αλλά και πως μπορεί το δίτροχο ρομπότ να σχεδιάσει μια διαδρομή μέσα σε αυτόν.

Η επαναχρησιμοποίηση του χάρτη που έχει δημιουργήσει το ρομπότ είναι καθοριστικής σημασίας για πολλούς λόγους. Ένας από αυτούς είναι ότι η διαδικασία δημιουργίας του χάρτη είναι ιδιαίτερα απαιτητική

και δαπανά αρκετούς υπολογιστικούς πόρους. Αυτό έχει σαν αποτέλεσμα τη δυσκολία άλλων διεργασιών για το ρομπότ. Άλλος ένας λόγος που ο χάρτης είναι αρκετά χρήσιμος είναι για τον εντοπισμό του ρομπότ στο χώρο εργασίας του αλλά και στη πλοήγηση του σε αυτόν.

Το πρώτο βήμα για την επαναχρησιμοποίηση του χάρτη είναι η μεταφορά των χαρτών που δημιούργησε το ρομπότ στο φάκελο maps του πακέτου. Έπειτα θα πρέπει να γίνει η εκ νέου επεξεργασία του setup.py αρχείου του πακέτου για να μπορέσουν να γίνουν install οι χάρτες. Θα προστεθούν δύο νέα μονοπάτια στη λίστα data files του αρχείου και είναι τα παρακάτω:

```
$ (os.path.join('share',package_name,'maps'),
        glob(os.path.join('maps','*.yaml'))),
$ (os.path.join('share',package_name,'maps'),
        glob(os.path.join('maps','*.pgm'))),
```

Πίνακας 104. Εκ νέου επεξεργασία του setup.py.

Ιδιαίτερα σημαντική είναι η προσθήκη της κατάληξης .pgm, καθώς το αρχείο με την συγκεκριμένη κατάληξη καλείται μέσω του my_map.yaml και συνεπώς πρέπει να υπάρχει στο χτίσιμο του πακέτου.

Αφού γίνουν οι παραπάνω αλλαγές στο αρχείο setup.py, πρέπει να δημιουργηθούν κάποια νέα launch file που θα βοηθήσουν στην φόρτωση του χάρτη στο Rviz. Πιο συγκεκριμένα, θα γίνει η χρήση δύο launch files του πακέτου nav2 bringup. Τα δύο αυτά αρχεία είναι το localization_launch.py και το navigation_launch.py. Το πρώτο αρχείο εκκινεί κάποια απαραίτητα nodes μέσω των οποίων επαναχρησιμοποιείται ο χάρτης. Με την εκκίνηση του δεύτερου launch file, το ρομπότ πλέον θα είναι αυτόνομο. Με χρήση της επιλογής Nav2 Goal του Rviz το ρομπότ θα μπορεί να δημιουργήσει μια διαδρομή προς μια επιθυμητή τοποθέτηση στο χώρο εργασίας τους. Γίνεται η αντιγραφή των αρχείων στο πακέτο μέσω των παρακάτω εντολών.

```
$ cd ~/ros2_ws/src/robot_loc_nav
$ cp /opt/ros/humble/share/nav2_bringup/laumch/navigation_launch.py launch/
$ cp /opt/ros/humble/share/nav2_bringup/laumch/localization_launch.py launch/
$ cp /opt/ros/humble/share/nav2_bringup/params/nav2_params.yaml config/
```

Πίνακας 105. Αντιγραφή αρχείων από το Nav2_bringup package.

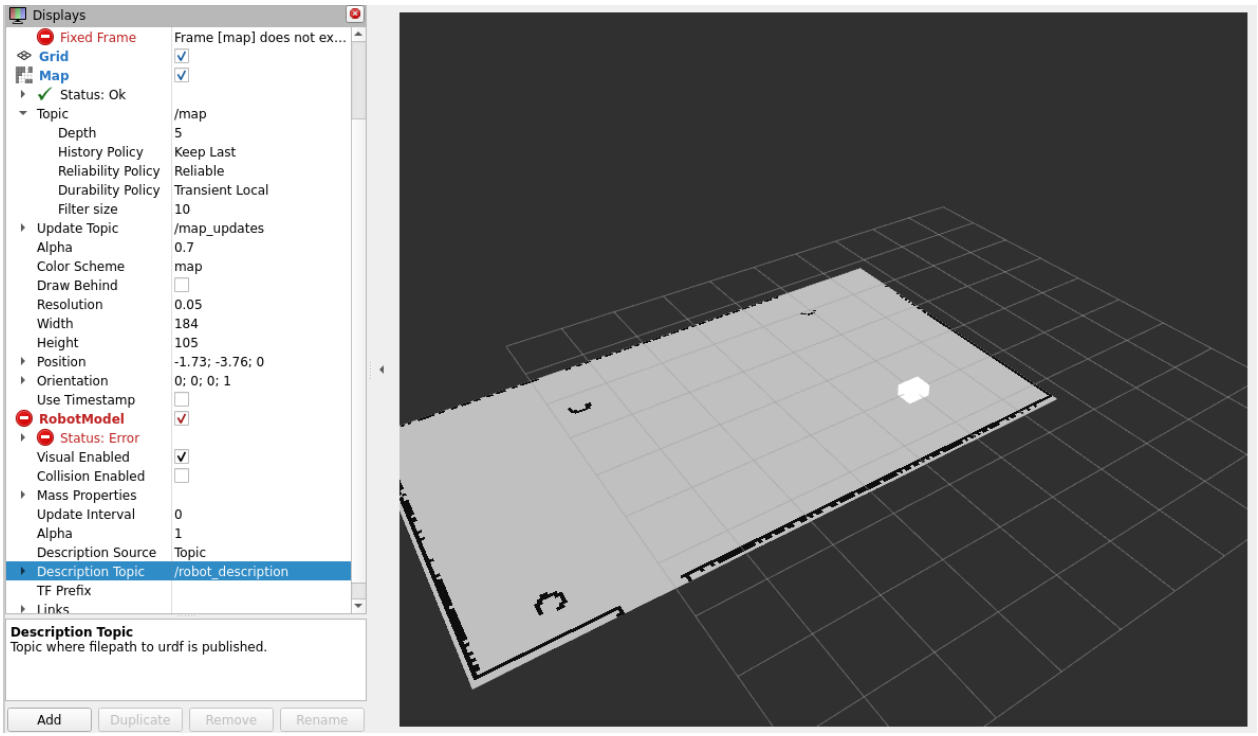
Με τις παραπάνω εντολές γίνεται η μετάβαση στο πακέτο. Έπειτα αντιγράφονται τα δύο επιθυμητά launch αρχεία εντός του πακέτου και επιπρόσθετα ένα yaml αρχείο με κάποιες παραμέτρους που θα εκκινήσει το ένα από τα δύο launch files. Στο σημεία αυτό πρέπει να γίνει μια πολύ σημαντική παρατήρηση. Τα δύο launch αρχεία που αντιγράφηκαν δεν έχουν την κατάλληλη επέκταση στο όνομα του αρχείου τους. Σε περίπτωση που χτιστεί το πακέτο τότε αυτά τα δύο αρχεία δεν θα εγκατασταθούν στο πακέτο και συνεπώς δεν θα μπορούν να εκτελεστούν. Συνεπώς θα πρέπει να αλλάξουν οι καταλήξεις των αρχείων σε .launch.py.

Εκτός από τις αλλαγές στα ονόματα των αρχείων τότε θα πρέπει να γίνουν κάποιες επιπρόσθετες αλλαγές. Πιο συγκεκριμένα, θα πρέπει εντός των launch αρχείων να αλλάξει το όνομα του πακέτου που ορίζεται και να τοποθετηθεί το όνομα του πακέτου που έχει δημιουργηθεί στο κεφάλαιο, δηλαδή το robot_loc_nav. Συνεπώς στον κώδικα ορίζεται το αντικείμενο bringup_dir και στο οποίο πρέπει να δοθεί το όνομα του πακέτου.

Μια επιπρόσθετη αλλαγή που πρέπει να πραγματοποιηθεί στα launch αρχεία είναι κατά τον ορισμό της μεταβλητής declare_params_file_cmd. Θα πρέπει να αντικατασταθεί το /param σε /config καθώς όλα τα αρχεία παραμετρικοποίησης αποθηκεύονται στο φάκελο config. Στο σημείο αυτό το πακέτο θα χτιστεί και θα καλεστούν τα launch αρχεία για την φόρτωση του χάρτη στο Rviz. εκτελούνται τα ακόλουθα:

```
$ cd ~/ros2_w

$ colcon build --packages-select robot_loc_nav

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch robot_loc_nav display.launch.py

$ # Open new terminal

$ cd src/robot_loc_nav

$ ros2 launch robot_loc_nav localization.launch.py  map:= /maps/my_map.yaml
```

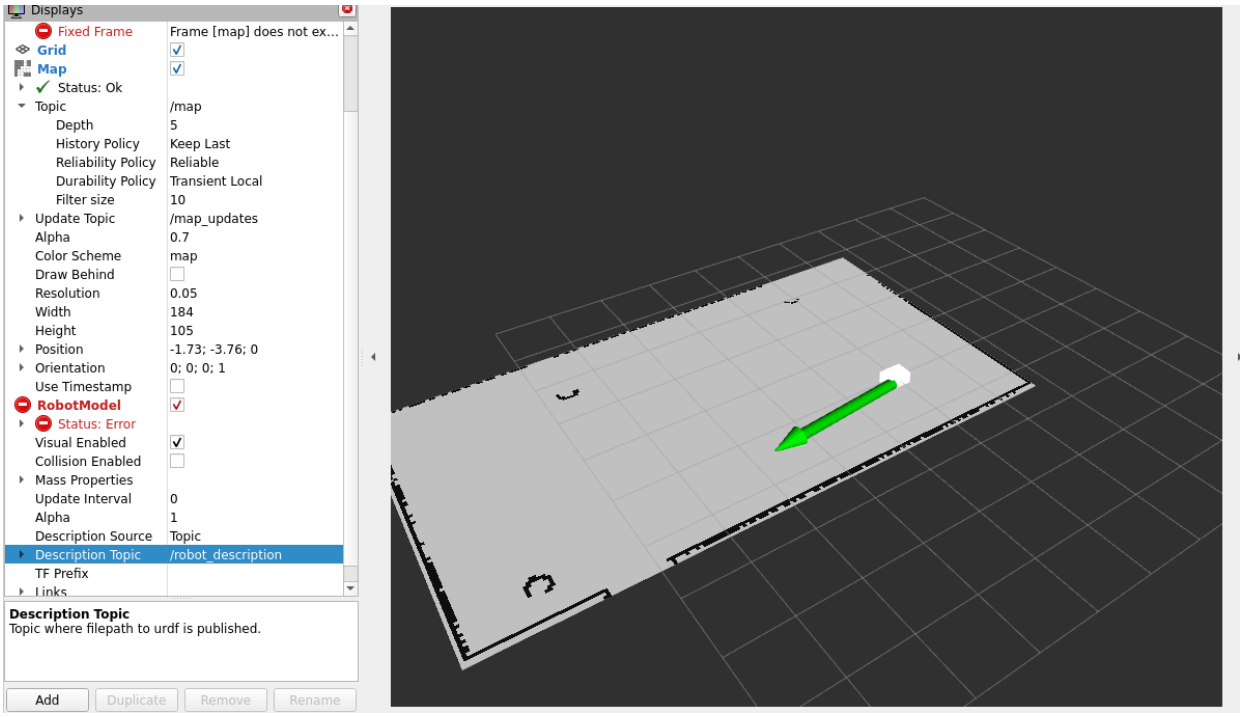
Πίνακας 106. Επαναχρησιμοποίηση του χάρτη στο Rviz.



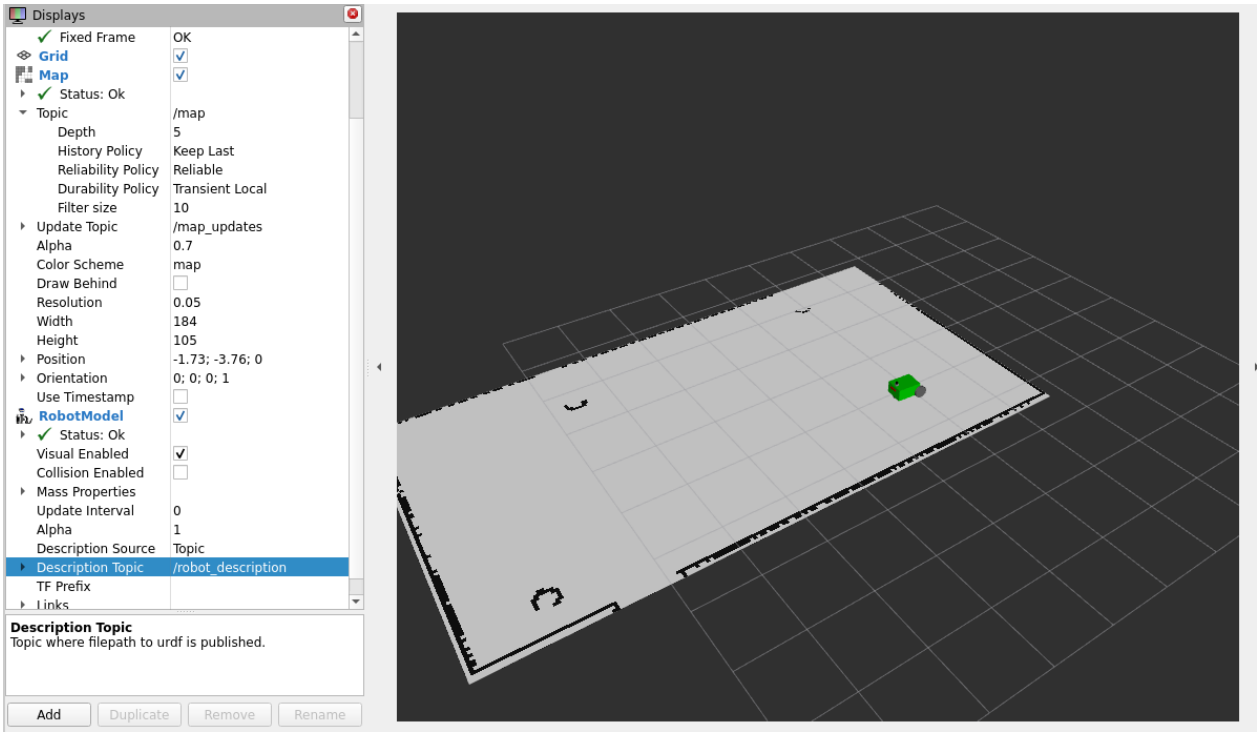

Εικόνα 70. Φόρτωση του χάρτη στο Rviz.

Στην παραπάνω εικόνα παρατηρείται ότι έχει επανέλθει ο χάρτης του ρομπότ στο τρισδιάστατο χώρο του Rviz. Αρχικά το Rviz ήταν κενό και θα πρέπει να προστεθούν το ρομποτικό μοντέλο και το ο χάρτης αυτού μέσα από το **Display Panel**. Στο σημείο αυτό πρέπει να γίνουν δύο σημαντικές παρατηρήσεις. Αρχικά, για να εμφανιστεί ο χάρτης, όπως φαίνεται παραπάνω, θα πρέπει να μεταβληθούν κάποιες ρυθμίσεις από το Displays Panel. Πιο συγκεκριμένα, θα πρέπει να επιλεγεί το **Map**, έπειτα το **Topic** και τέλος στο **Durability Policy** πρέπει να γίνει η επιλογή του **Transient Local**.

Η δεύτερη παρατήρηση, είναι ότι κατά την επαναχρησιμοποίηση του χάρτη, το ρομπότ δεν γνωρίζει την αρχική του τοποθέτηση στο χώρο. Για να μπορέσει το ρομπότ να μάθει την αρχική του τοποθέτηση, γίνεται η χρήση του εργαλείου **2D Pose Estimate** που βρίσκεται στην καρτέλα επιλογών του Rviz. Με αυτό το τρόπο θα καθοριστεί η αρχική τοποθέτηση του ρομπότ στον χώρο εργασίας του. Αφού επιλεγεί αυτό το εργαλείο γίνεται η μετάβαση στον τρισδιάστατο χώρο και έπειτα καθορίζεται η θέση και η στροφή του ρομπότ, όπως φαίνεται στην παρακάτω εικόνα. Με αυτό το τρόπο το ρομπότ αντιλαμβάνεται τη θέση του στο χάρτη που έχει φορτωθεί στο Rviz.




Εικόνα 71. Καθορισμός της τοποθέτησης του ρομπότ.




Εικόνα 72. Το ρομπότ με γνωστή την αρχική του τοποθέτηση.

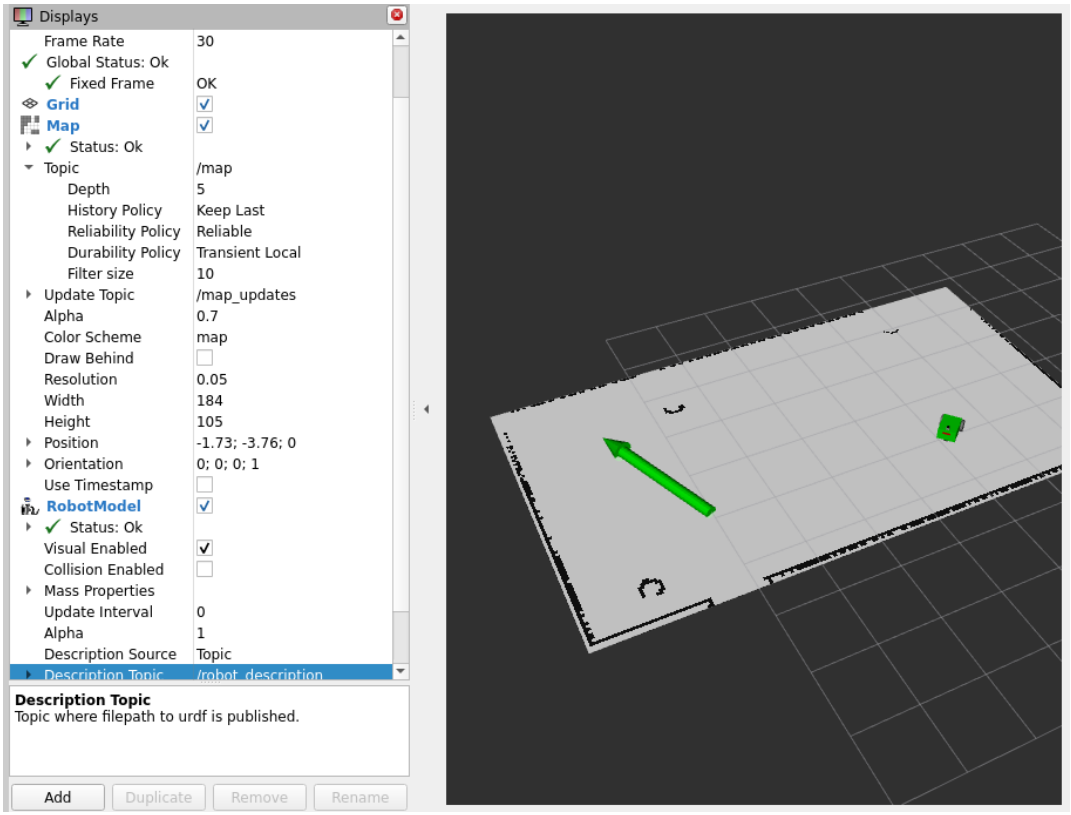
Μέχρι το σημείο αυτό, έχει επιτευχθεί η φόρτωση του χάρτη και η επαναχρησιμοποίηση του καθώς και ο εντοπισμός του ρομπότ με χρήση εργαλείων του Rviz. Το επόμενο βήμα είναι η αυτονομία του ρομπότ. Θα εκκινήσει ένα νέο launch αρχείο μέσω του οποίο, με εργαλείο του Rviz θα δηλώνεται μια επιθυμητή τοποθέτηση στο ρομπότ και αυτό θα δημιουργεί μια διαδρομή για να μεταβεί σε εκείνο το σημείο. Συνεπώς θα πρέπει να εκτελεστεί το navigation.launch.py αρχείο, το οποίο νωρίτερα αντιγράφηκε εντός του πακέτου

```
$ ros2 launch robot_loc_nav navigation.launch.py use_sim_time:=true
```

Πίνακας 107. Εκτέλεση του navigation.launch.py.

Η πλοήγηση του ρομπότ σε ένα γνωστό χώρο εργασίας, είναι πολύ ευκολότερη συγκριτικά με την πλοήγηση σε ένα ανεξερεύνητο χώρο για πολλούς λόγους. Αρχικά, το ρομπότ έχει πρωτύτερη γνώση για τα εμπόδια του χώρου εργασίας του καθώς επίσης η πλοήγηση είναι μια βαριά διαδικασία που ωστόσο μπορεί να πραγματοποιηθεί ικανοποιητικά όταν δεν υπάρχει ταυτόχρονη χαρτογράφηση.

Για τον καθορισμό μιας επιθυμητής τοποθέτησης, γίνεται η επιλογή του **2D Goal Pose** από το Rviz και στη συνέχεια δίνεται η επιθυμητή τοποθέτηση μέσα από το τρισδιάστατο χώρο του Rviz.




Εικόνα 73. Καθορισμός νέας θέσης για το ρομπότ.

Έπειτα από τον καθορισμό μία επιθυμητής τοποθέτησης για το ρομπότ, παρατηρείται η κίνηση αυτού προς αυτή τη θέση που δηλώθηκε στο χώρο εργασίας του και παράλληλα γίνεται η αποφυγή εμποδίων. Σε περίπτωση που προκύψει κάποιο εμπόδιο κατά την πορεία του, το ρομπότ θα το αποφύγει και θα συνεχίσει εκ νέου την πορεία του προς τον τελικό στόχο. Στο σημείο αυτό το ρομπότ διαφορικής κίνησης είχε αποκτήσει αυτονομία, καθώς μπορεί να χαράξει μόνο του διαδρομές εντός του χώρου εργασίας του.

## Κεφάλαιο 6

# Το πακέτο MoveIt 2

Στο κεφάλαιο αυτό θα παρουσιαστεί ένα πολύ σημαντικό πακέτο του ROS 2 που είναι το MoveIt. Το συγκεκριμένο πακέτο είναι ιδιαίτερα χρήσιμο και δημοφιλές, καθώς με τη χρήση του μπορούν να πραγματοποιηθούν απαραίτητες διεργασίες για τον έλεγχο και χειρισμό βραχιόνων με χρήση του ROS. Στη αρχή του κεφαλαίου θα παρουσιαστεί η εγκατάσταση και το χτίσιμο του πακέτου. Έπειτα θα ακολουθήσει η εκτέλεση κάποιων demos του MoveIt και τέλος θα παρουσιαστεί η διαδικασία ενσωμάτωσης του MoveIt σε ένα βραχίονα με χρήση του MoveIt Setup Assistant.

## 6.1 Εγκατάσταση του MoveIt 2

Το MoveIt 2 είναι μια πλατφόρμα για τον χειρισμό ρομποτικών βραχιόνων στο ROS 2, δηλαδή περιέχει πακέτα και εργαλεία που βοηθήσουν ένα ρομποτικό βραχίονα να αλληλοεπιδράσει με αντικείμενα στο χώρο εργασίας του αλλά και να λάβει διάφορες τοποθετήσεις σε αυτόν. Στο πακέτο εμπεριέχονται όλες οι αναβαθμίσεις που έχουν προκύψει στο motion planning, manipulation, στις κινηματικές εξισώσεις, στους ελεγκτές και στη πλοήγηση συγκριτικά με την πρώτη έκδοση του πακέτου για το ROS. Το πακέτο δημοσιεύτηκε για το ROS 2 το 2019 και μπορεί να υποστηρίξει τα περισσότερα distribution του ROS 2.

Το MoveIt 2 είναι ένα πανίσχυρο και χρηστικό εργαλείο που βοηθάει στην αντιμετώπιση προβλημάτων που αφορούν τους ρομποτικούς βραχίονες παντός τύπου. Διευκολύνει πολλά από το προβλήματα που παρατηρούνται στους βραχίονες όπως η ευθύ και αντίστροφη κινηματική των ρομπότ. Αυτό όμως έχει και ένα κόστος, το οποίο είναι η απαίτηση για υψηλή υπολογιστική ισχύ.

Παρακάτω παρουσιάζεται η εγκατάσταση του πακέτου, η οποία αποτελείται από τη δημιουργία ενός νέου workspace, την εγκατάσταση των πακέτων από το αποθετήριο του MoveIt και το χτίσιμο των πακέτων. Πληροφοριακά η διαδικασία εγκατάστασης είναι αρκετά πιο χρονοβόρα από αυτή που απαιτήθηκε κατά την εγκατάσταση στου ROS 2.

Για αρχή πρέπει να εγκατασταθεί το rosdep, μέσω του οποίου εγκαθίστανται κάποιες απαραίτητες λειτουργίες τους συστήματος:

```
$ sudo apt install python3-rosdep
```

Πίνακας 108. Εγκατάσταση του rosdep.

Στη συνέχεια είναι επιθυμή η εγκατάσταση κάποιων πακέτων. Τα πακέτα αυτά μπορεί να έχουν ήδη εγκατασταθεί, ωστόσο συνίσταται η εκτέλεση των παρακάτω εντολών ώστε τα πακέτα να είναι ενημερωμένα:

```
$ sudo rosdep init

$ rosdep update

$ sudo apt update

$ sudo apt dist-upgrade
```

Πίνακας 109. Εγκατάσταση κάποιων επιπρόσθετων πακέτων.

Στη συνέχεια εγκαθίστανται το εργαλείο colcon, με το οποίο χτίζονται τα πακέτα του ROS. Για το χτίσιμο των πακέτων του MoveIt θα χρειαστεί το mixin, το οποίο είναι μια επιπρόσθετη ιδιότητα που βοηθάει στο χτίσιμο των πακέτων καθώς είναι απαραίτητο για την εγκατάσταση αυτών.

```
$ sudo apt install python3-colcon-common-extensions

$ sudo apt install python3-colcon-mixin

$ colcon mixin add default https://raw.githubusercontent.com/colcon/colcon-mixin-
repository/master/index.yaml

$ colcon mixin update default

$ sudo apt install python3-vcstool
```

Πίνακας 110. Εγκατάσταση του colcon mixin και του vctool.

Στη συνέχεια θα δημιουργηθεί ένα νέο ROS 2 workspace και έπειτα θα γίνουν λήψη τα πακέτα του MoveIt. Στη συνέχεια θα εγκατασταθεί ο πηγαίος κώδικας του πακέτου.

```
$ mkdir -p ~/ws_moveit/src

$ cd ~/ws_moveit/src

$ git clone https://github.com/ros-planning/moveit2_tutorials

$ vcs import < moveit2_tutorials/moveit2_tutorials.repos
```

Πίνακας 111. Εγκατάσταση των πακέτων του MoveIt.

Αφού εγκατασταθούν όλα τα πακέτα, ακολουθεί το χτίσιμο του πακέτου στη ρίζα του workspace. Συνεπώς εκτελείται το παρακάτω:

```
$ cd ~/ws_moveit

$ colcon build --mixin release
```

Πίνακας 112. Χτίσιμο του πακέτου.

Το χτίσιμο του πακέτου είναι ιδιαίτερα χρονοβόρο και εξαρτάται από την ταχύτητα του επεξεργαστή αλλά και από τη διαθέσιμη μνήμη RAM του υπολογιστή.

Έπειτα από το επιτυχές χτίσιμο του πακέτου πρέπει να γίνει source το περιβάλλον του workspace. Συνεπώς θα εκτελεστεί η ακόλουθη εντολή η οποία θα προσθέσει στο bashrc αρχείο κατάλληλη εντολή, ώστε να μην χρειάζεται η διαδικασία να επαναλαμβάνεται μετά το χτίσιμο του πακέτου.

```
$ echo 'source ~/ws_moveit/install/setup.bash' >> ~/.bashrc
```

Πίνακας 113. Η εντολή για το source του ws_moveit στο bashrc file.

Αφού ολοκληρωθούν οι παραπάνω διαδικασίες, τότε το πακέτο είναι έτοιμο για χρήση. Στην επόμενη ενότητα θα εκτελεστεί ένα demo του MoveIt και θα παρουσιαστούν κάποιες απλές αλλά ιδιαίτερα χρήσιμες λειτουργίες του.

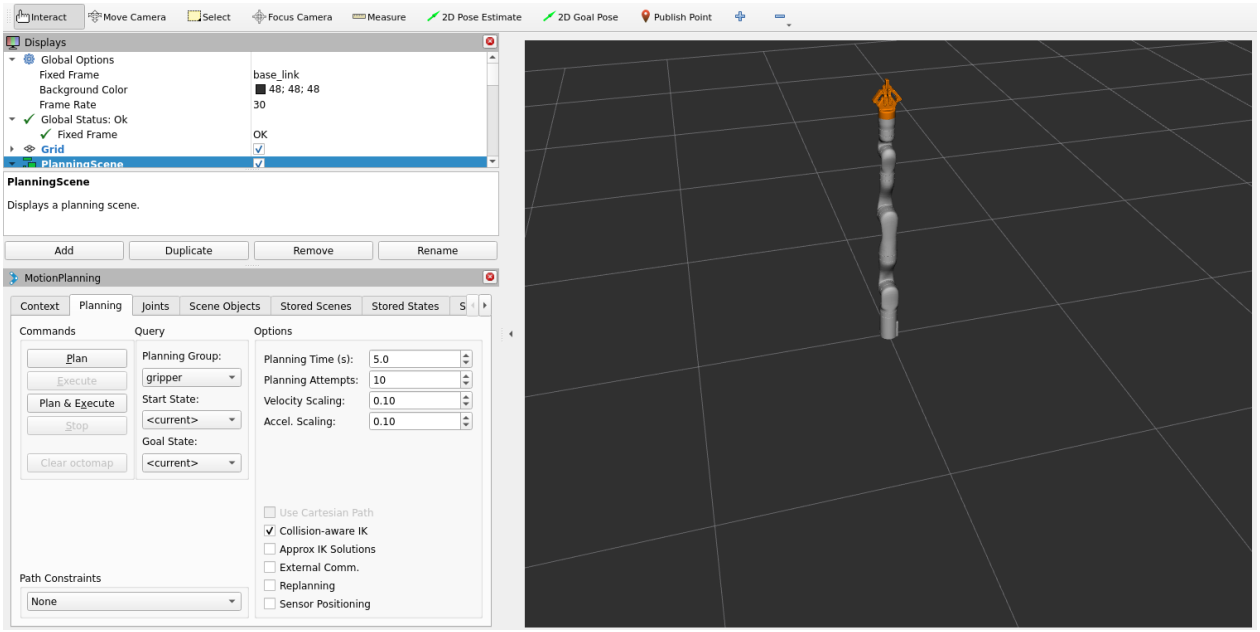
## 6.2 Το MoveIt στο περιβάλλον του Rviz

Το Rviz είναι ένα πανίσχυρο εργαλείο το οποίο προσφέρει τρισδιάστατες απεικονίσεις και μπορεί να βοηθήσει στη διόρθωση της συμπεριφοράς ενός ρομπότ. Μέσω της οπτικοποίησης μπορούν να εξαχθούν συμπεράσματα για τις λειτουργίες ενός ρομποτικού συστήματος και να εξεταστεί αν αυτό επιτυγχάνει τις λειτουργίες που είναι επιθυμητές. Το MoveIt συνεργάζεται με το Rviz προσφέροντας ένα plugin μέσω του οποίου μπορούν να πραγματοποιηθούν πολλές διεργασίες που αφορούν ένα βραχίονα.

Για τη χρήση του MoveIt με το Rviz πρέπει να εκκινήσει ένα launch αρχείο. Συνεπώς εκτελείται το ακόλουθο:

```
$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0 # For WSL

$ ros2 launch moveit2_tutorials demo.launch.py
```

Πίνακας 114. Έναρξη του ενός demo του MoveIt.

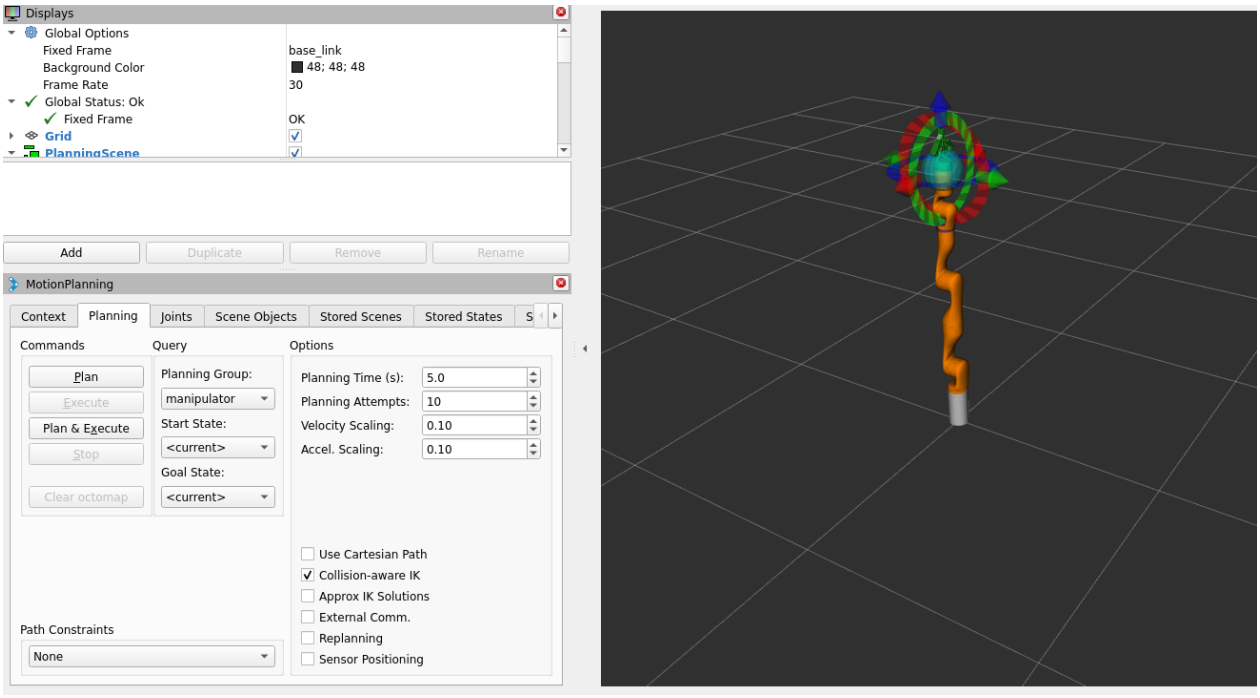



Εικόνα 74. Το plugin του MoveIt στο Rviz.

Κατά την πρώτη φορά που εκτελείται το παραπάνω launch file, υπάρχει περίπτωση να εκκινήσει το Rviz χωρίς τον βραχίονα στο τρισδιάστατο χώρο αλλά και χωρίς τον MotionPlanning plugin που είναι εμφανές στο κάτω αριστερά μέρος του Rviz. Προτού γίνει η προσθήκη του συγκεκριμένου plugin θα πρέπει αρχικά να προστεθεί το **PlanningScene**. Για τη προσθήκη του επιλέγεται το **Add** στο Displays panel και έπειτα επιλέγεται το Planning Scene. Αναλυτικότερα, θα πρέπει να γίνουν αλλαγές στα **Robot Description** και **Planning Scene Topic** και να περαστούν τα ακόλουθα topics κατά αντιστοιχία, robot_description και monitored_planning_scene.

Αφού πραγματοποιηθούν οι συγκεκριμένες αλλαγές στα topic τότε το ρομποτικό μοντέλο θα έχει προστεθεί στο τρισδιάστατο χώρο του Rviz. Επιπλέον, θα πρέπει να προστεθούν και δύο νέα feature μέσω του displays panel, τα οποία είναι το **Trajectory** και **MotionPlanning**. Στο feature Trajectory θα γίνουν αλλαγές στα topic και συγκεκριμένα στο πλαίσιο **Planned Path** θα περαστεί το /displayed_planned_path topic. Στο **MotionPlanning**, θα πρέπει να πραγματοποιηθούν αλλαγές στα topics και να επιλεγούν οι αντίστοιχες επιλογές όπως και στο feature **PlanningScene**.

Στο σημείο αυτό, έχει εμφανιστεί και το MotionPlanning plugin στο Displays panel του Rviz. Μια τελευταία αλλαγή που πρέπει να πραγματοποιηθεί είναι η επιλογή του Planning Group που βρίσκεται στο plugin η οποία πρέπει να αλλαχθεί σε **manipulation**.

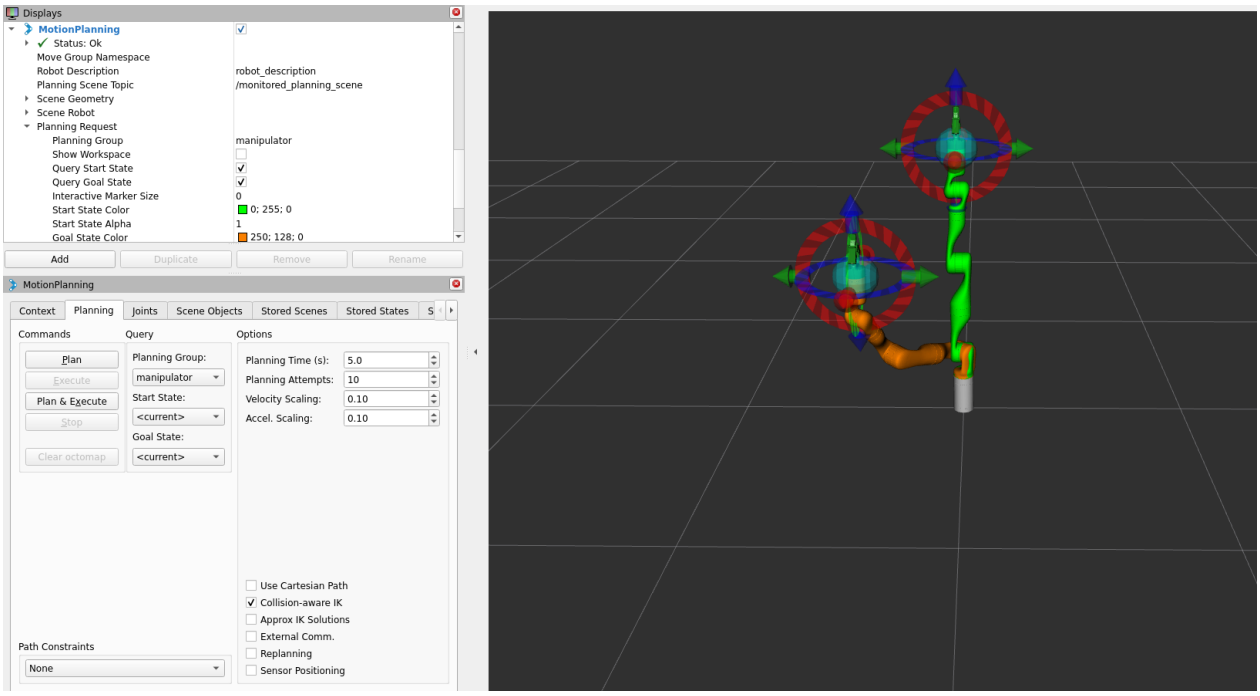



Εικόνα 75. Ο βραχίονας έπειτα από τις αλλαγές.

Αφού έχουν πραγματοποιηθούν τα παραπάνω, πλέον μπορεί να γίνει η αλληλεπίδραση με τον βραχίονα. Πιο συγκεκριμένα, μια από τις πιο χαρακτηριστικές ιδιότητες που παρέχεται είναι η επίλυση της αντίστροφης κινηματικής ενός βραχίονα. Δηλαδή, δεδομένης μια τοποθέτησης του βραχίονα και μιας νέας επιθυμητής τοποθέτησης στο χώρο εργασίας του, τότε το MoveIt μπορεί να αναπαραστήσει τον τρόπο με τον οποίο θα γίνει η μετακίνηση από τη μια θέση στην άλλη, δηλαδή να λυθεί το πρόβλημα της αντίστροφης κινηματικής.

Όπως παρατηρείται στο περιβάλλον του Rviz ο βραχίονας θα έχει λάβει πορτοκαλί χρώμα και στο άκρο του παρατηρείται μια γαλάζια σφαίρα. Επιλέγοντας την σφαίρα με το ποντίκι του υπολογιστή, μπορεί να γίνει η μετάβαση του βραχίονα σε οποιοδήποτε σημείο του χώρου εργασίας του. Πληροφοριακά, ο βραχίονας, Kinova Gen 3, που χρησιμοποιείται είναι έξι βαθμών ελευθερίας και συνεπώς ο χώρος εργασίας του είναι μια σφαίρα.

Συνεπώς όταν ο βραχίονας έχει λάβει το πορτοκαλί χρώμα, αυτό μεταφράζεται στην τοποθέτηση στόχος που έχει δηλωθεί για το ρομπότ. Για να καθοριστεί η αρχική θέση του βραχίονα γίνεται η μετάβαση στο Displays Panel και στο feature MotionPlanning. Στη συνέχεια γίνεται μετάβαση στο Planning Request και επιλέγεται το **Query Start State** όπως φαίνεται στην παρακάτω εικόνα.

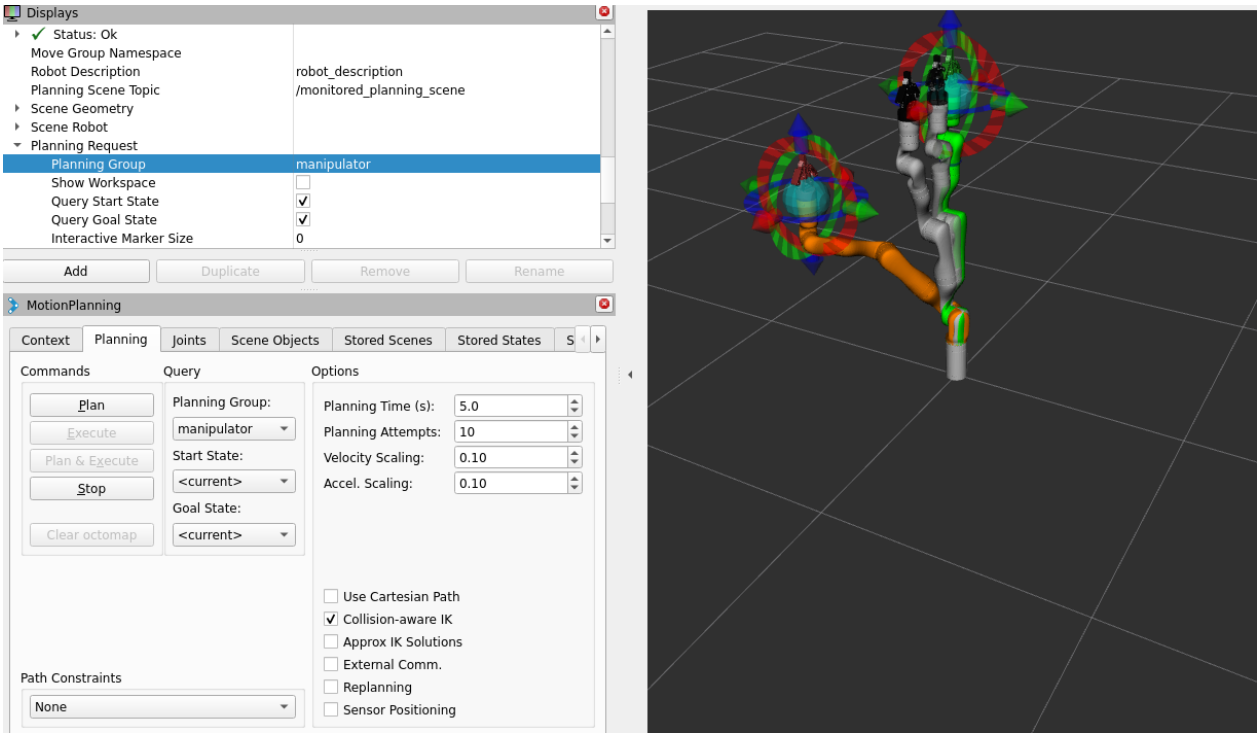



Εικόνα 76. Ο βραχίονας με το Query Start State.

Αφού ενεργοποιηθεί το Query Start State τότε η αρχική θέση του ρομπότ θα αποκτήσει το πράσινο χρώμα. Συνολικά, στο Rviz θα υπάρχουν τρεις απεικονίσεις του βραχίονα. Το γκρι είναι ο πραγματικός βραχίονας και ποια είναι η τοποθέτηση του στο χώρο εργασίας. Ο βραχίονας με πράσινο χρώμα δηλώνει την αρχική θέση του ρομπότ και ο βραχίονας με πορτοκαλί χρώμα δηλώνει τη θέση που είναι επιθυμητό να λάβει το ρομπότ. Στην παραπάνω εικόνα, η αρχική θέση και η πραγματική θέση ταυτίζονται και για αυτό αποτυπώνεται μόνο η αρχική θέση με πράσινο χρώμα.

Αφού δηλωθεί η αρχική και η τελική τοποθέτηση του βραχίονα τότε μπορεί να σχεδιαστεί η διαδρομή που θα ακολουθήσει το ρομπότ, δηλαδή να λυθεί το αντίστροφο κινηματικό πρόβλημα του ρομπότ. Για να υλοποιηθεί αυτό επιλέγεται το **Plan**, το οποίο βρίσκεται στο MotionPlanning Plugin. Με αυτό τον τρόπο σχεδιάζεται η διαδρομή που θα ακολουθήσει ο βραχίονας. Αφού γίνει το Plan, επιλέγεται το **Execute** και με αυτό τον τρόπο πραγματοποιείται η μετακίνηση του ρομπότ από την αρχική τοποθέτηση σε αυτή που του ζητήθηκε.

Άξιο αναφοράς είναι ότι μπορεί να γίνει ταυτόχρονα το Plan και το Execute, επιλέγοντας την αντίστοιχη επιλογή στο MotionPlanning Plugin. Τέλος, στο τρισδιάστατο χώρο του Rviz, αρχίζει και προβάλλεται η πορεία που θα ακολουθήσει το ρομπότ.




Εικόνα 77. Το Plan και Execute της πορείας του ρομπότ.

Το **MotionPlanning** Plugin και η επίλυση της αντίστροφης κινηματικής ενός ρομπότ είναι μόνο ένα από τις πολλές διεργασίες που προσφέρει το πακέτο MoveIt. Συμπληρωματικά, το λογισμικό του MoveIt είναι ανεπτυγμένο σε γλώσσα C++, και στο επίσημο Documentation προσφέρονται ποικιλία εφαρμογών με χρήση του Rviz και αλληλεπίδρασης με τη συγκεκριμένη γλώσσα προγραμματισμού. Πληροφοριακά, παρουσιάζεται η δημιουργία ενός βραχίονα με τη γλώσσα C++, η τοποθέτηση του στο περιβάλλον του Rviz, η σχεδίαση διαδρομής γύρω από εμπόδια και η χρήση της αρπάγης ώστε αυτή να μεταφέρει αντικείμενα και να τα τοποθετεί σε κάποιο άλλο σημείο.

## 6.3 MoveIt Setup Assistant

Το MoveIt Setup Assistant είναι ένα πολύ ισχυρό εργαλείο του πακέτου το οποίο παρέχει γραφικό περιβάλλον και χρησιμοποιείται ώστε να διαμορφώσει κατάλληλες παραμέτρους για τα ρομπότ που είναι επιθυμητό να αλληλεπιδράσουν με το MoveIt. Ο βασικός σκοπός του είναι να παράγει ένα Semantic Robot Description Format (SRDF) αρχείο για ένα ρομπότ. Το αρχείο αυτό θα περιγράφει κάποιες επιπρόσθετες πληροφορίες για το ρομπότ, όπως είναι τα planning groups, οι end effectors και διάφοροι παράμετροι για τις κινηματικές εξισώσεις του ρομπότ.

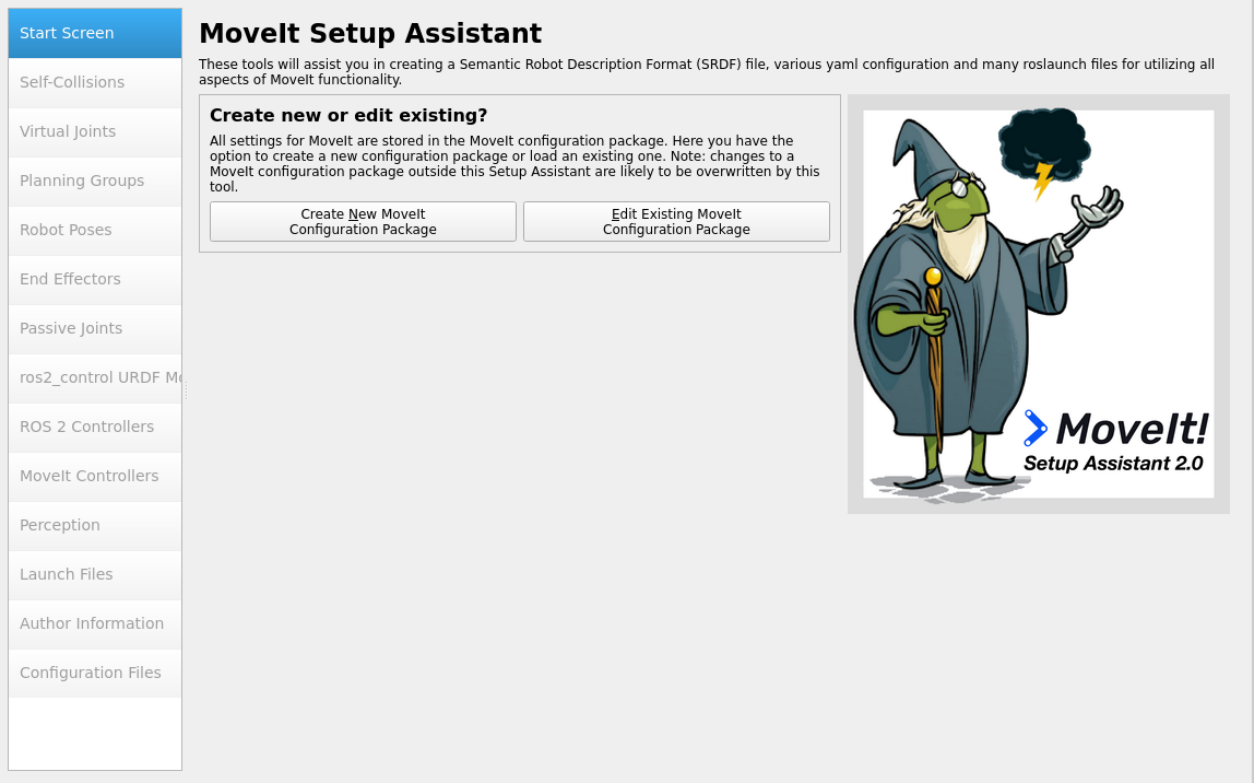
Επιπροσθέτως, μέσω του περιβάλλοντος αυτού παράγονται κάποια απαραίτητα configuration files τα οποία είναι χρήσιμα για το MoveIt pipeline. Για την χρήση του MoveIt Setup Assistant, είναι απαραίτητη η χρήση ενός URDF μοντέλου που καθορίζει το ρομπότ. Αφού υπάρχει ένα αρχείο URDF, τότε αυτό μπορεί να γίνει import στο γραφικό περιβάλλον και να καθοριστούν κάποιοι παράμετροι του συστήματος.

Αρχικά θα πρέπει να εκκινήσει το γραφικό περιβάλλον του εργαλείου, το οποίο πραγματοποιείται με τα ακόλουθα.

```
$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0 # For WSL

$ ros2 launch moveit_setup_assistant setup_assistant.launch.py
```

Πίνακας 115. Εκκίνηση του Setup Assistant.

Έπειτα από την εκτέλεση της παραπάνω εντολής θα πρέπει να εκκινήσει το παρακάτω γραφικό περιβάλλον. Για τη χρήση του εργαλείου, θα χρησιμοποιηθεί ένα URDF ενός βραχίονα που εγκαθίσταται μαζί με το πακέτο.




Εικόνα 78. Το γραφικό περιβάλλον του Setup Assistant.

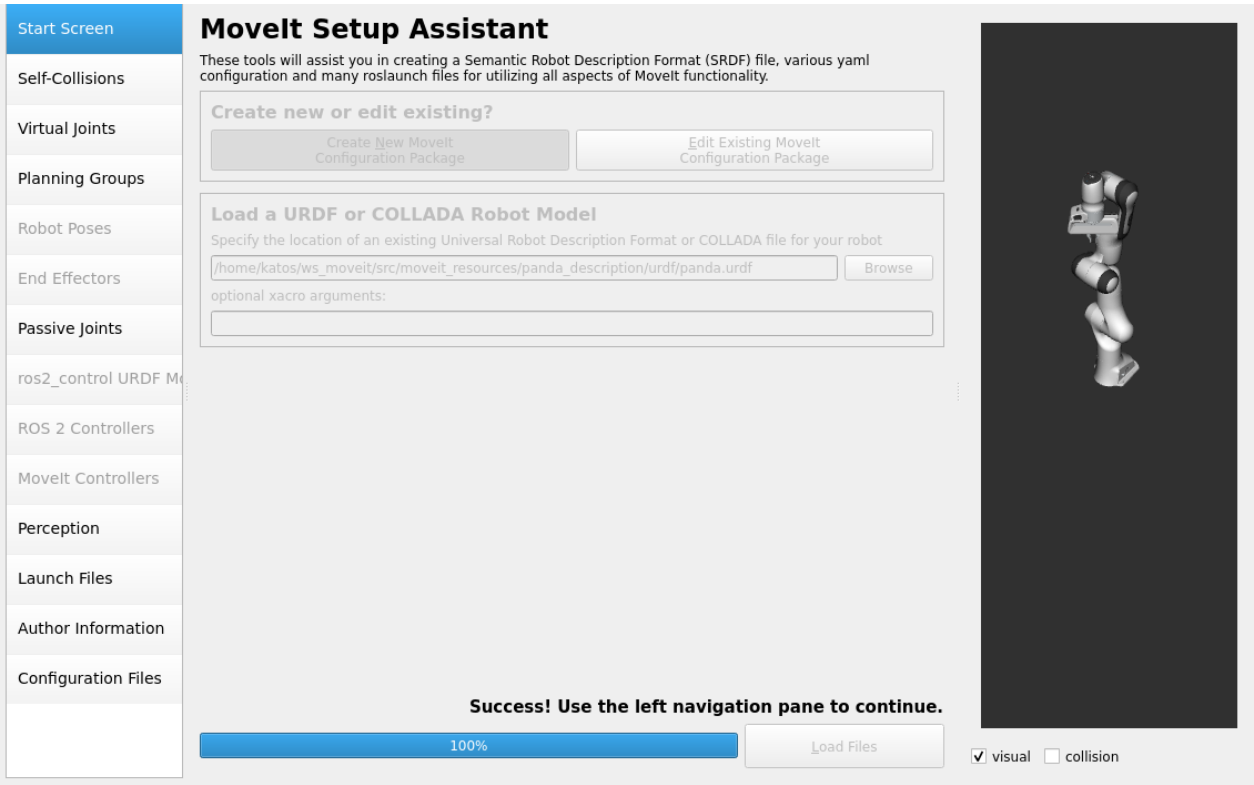
Έπειτα από την εκκίνηση του γραφικού περιβάλλοντος, επιλέγεται το **Create New MoveIt Configuration Package**. Στη συνέχεια, θα γίνει η αναζήτηση του βραχίονα στον προσωπικό υπολογιστή. Το μονοπάτι του αρχείο που είναι επιθυμητό να φορτωθεί στο Setup Assistant παρουσιάζεται στον ακόλουθο πίνακα.

| $ ~/ws_moveit2/src/moveit_resources/panda_description/urdf/panda.urdf |
|---|

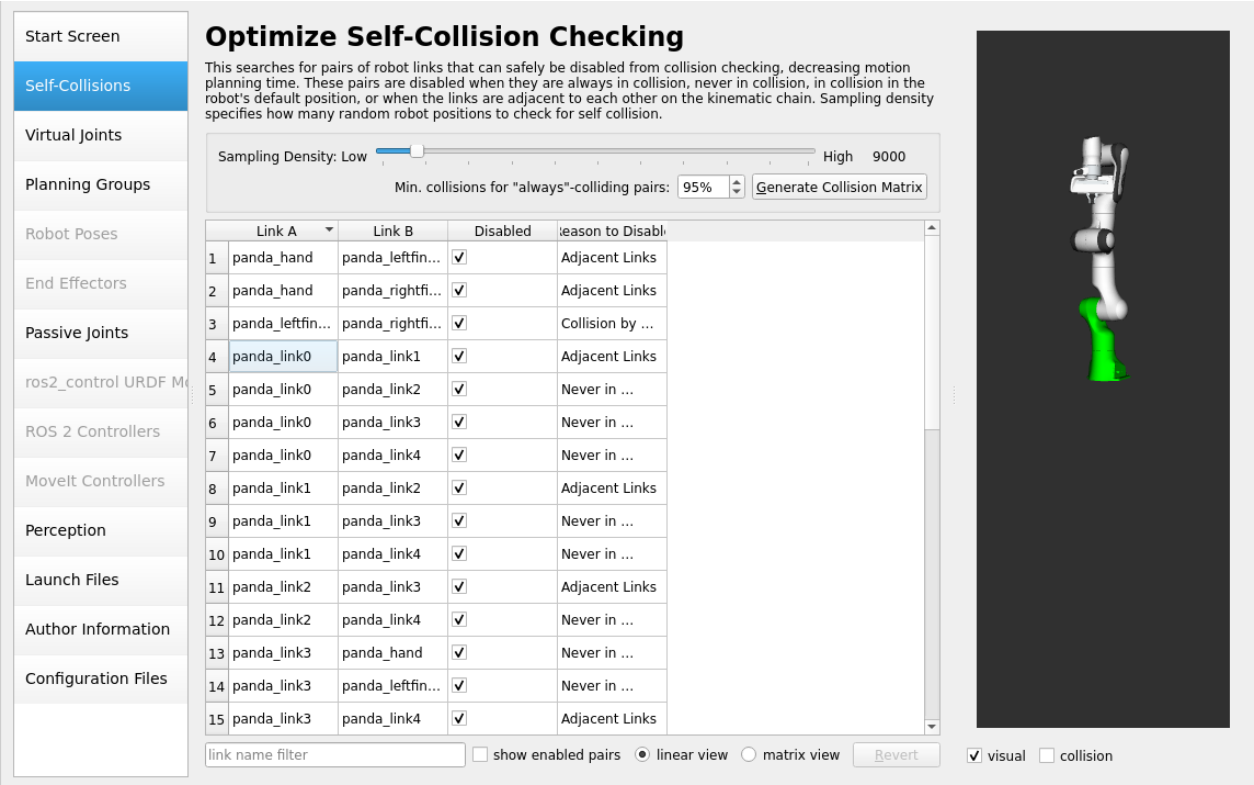
Πίνακας 116. Το μονοπάτι του ρομπότ.

Έπειτα από την εύρεση του αρχείου URDF στο προσωπικό υπολογιστή, τότε επιλέγεται το Load Files στο κάτω δεξιά μέρος του γραφικού περιβάλλοντος. Αν δεν προκύψει κάποιο σφάλμα, τότε θα πρέπει να φορτωθεί ο βραχίονας στο γραφικό περιβάλλον και αυτός να εμφανίζεται στο δεξή μέρος του παραθύρου όπως παρουσιάζεται στην παρακάτω εικόνα.

Αν προκύψει το ακόλουθο αποτέλεσμα τότε επιλέγεται το **Self-Collisions** και ακολουθούν οι διεργασίες του επόμενου βήματος.




Εικόνα 79. Επιτυχής φόρτωση του βραχίονα.




Εικόνα 80. Το περιβάλλον του Self-Collisions.

Το πεδίο του Self-Collisions στην αρχή θα είναι κενό. Αφού επιλεγεί το **Generate Collision Matrix**, τότε θα προκύψουν όλες οι επιλογές στο κεντρικό παράθυρο όπως φαίνονται στην παραπάνω εικόνα. Το Self-Collisions μπορεί να βοηθήσει στην μείωση του χρόνο προγραμματισμού κίνησης (motion planning) μέσω της απενεργοποίησης κάποιων collisions μεταξύ κάποιων joints του ρομπότ που υπάρχει η γνώση ότι είναι ασφαλής. Συνεπώς το MoveIt απενεργοποιεί κάποια collision σε περίπτωση που αντιληφθεί ότι υπάρχει ασφάλεια, δηλαδή ότι δύο κομμάτια του βραχίονα δεν θα συγκρουστούν μεταξύ τους.

Το MoveIt για να εξασφαλίσει ότι δύο κομμάτια του βραχίονα δεν θα συγκρουστούν, δοκιμάζει διάφορες τοποθετήσεις του ρομπότ. Πιο συγκεκριμένα, οι δοκιμές που θα παραχθούν είναι 10.000, ωστόσο μέσω της μπάρας που βρίσκεται στο Sampling Density, μπορεί η συγκεκριμένη ρύθμιση να αλλάξει. Όπως θα παρατηρήθηκε, έπειτα από την επιλογή του **Generate Collision Matrix** το MoveIt χρειάζεται λίγο χρόνο για να πραγματοποιήσει τις δοκιμές και στο κεντρικό παράθυρο θα εμφανιστούν όλα τα αποτελέσματα. Σε αυτά παρουσιάζονται τα ζευγάρια των links που έχουν ανιχνευθεί ως ασφαλή από το Collision checking. Τα links που είναι ασφαλή είναι τσεκαρισμένα, ωστόσο η ρύθμιση αυτή μπορεί να αλλάξει.

Στη συνέχεια γίνεται η μετάβαση στο **Virtual Joints**. Ο κύριος στόχος του συγκεκριμένου πεδίου είναι να δημιουργήσει ένα joint μεταξύ του ρομπότ με το επίπεδο, ωστόσο διατίθεται και η επιλογή να τοποθετηθεί ο βραχίονας πάνω σε ένα αυτόνομο ρομπότ. Για την προσθήκη ενός εικονικού joint, επιλέγεται το **Add Virtual Joint**, μέσω του οποίου θα γίνει η μετάβαση σε ένα νέο γραφικό περιβάλλον.

Στο συγκεκριμένο γραφικό περιβάλλον θα πρέπει να καθοριστούν κάποιοι παράμετροι οι οποίοι είναι οι ακόλουθοι:

- Στο Virtual Joint Name δίνεται ένα όνομα για το joint.
- Στο Child Link δίνεται η επιλογή του συνδέσμου παιδιού, δηλαδή το panda_link0.
- Στο Parent Frame δηλώνεται ο κόσμος, δηλαδή το επίπεδο ή το αυτόνομο ρομπότ. Στη συγκεκριμένη περίπτωση θα δηλωθεί το world.
- Στο τέλος, θα καθοριστεί ο τύπος του joint και δηλώνεται τύπου fixed.

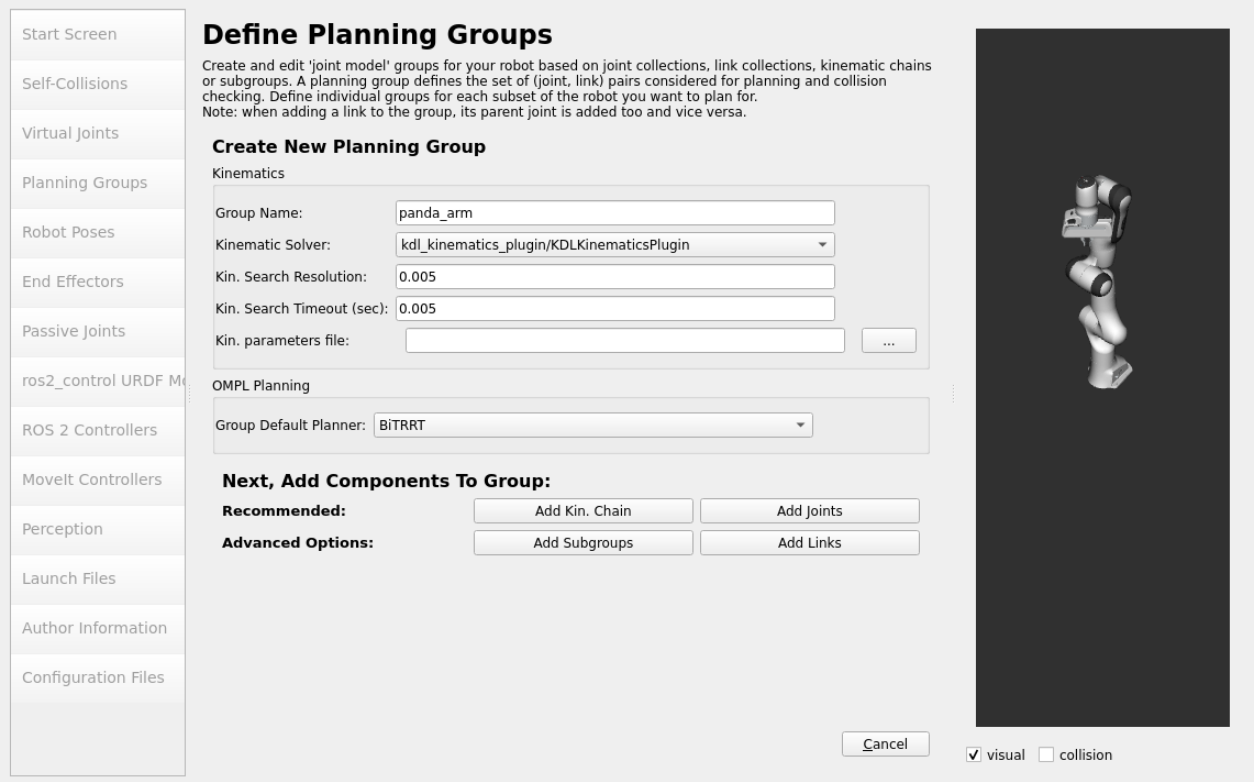
Αφού καθοριστούν όλες οι παράμετροι, τότε επιλέγεται το save και το εικονικό joint έχει δηλωθεί επιτυχώς.


Εικόνα 81. Καθορισμός των virtual joints.

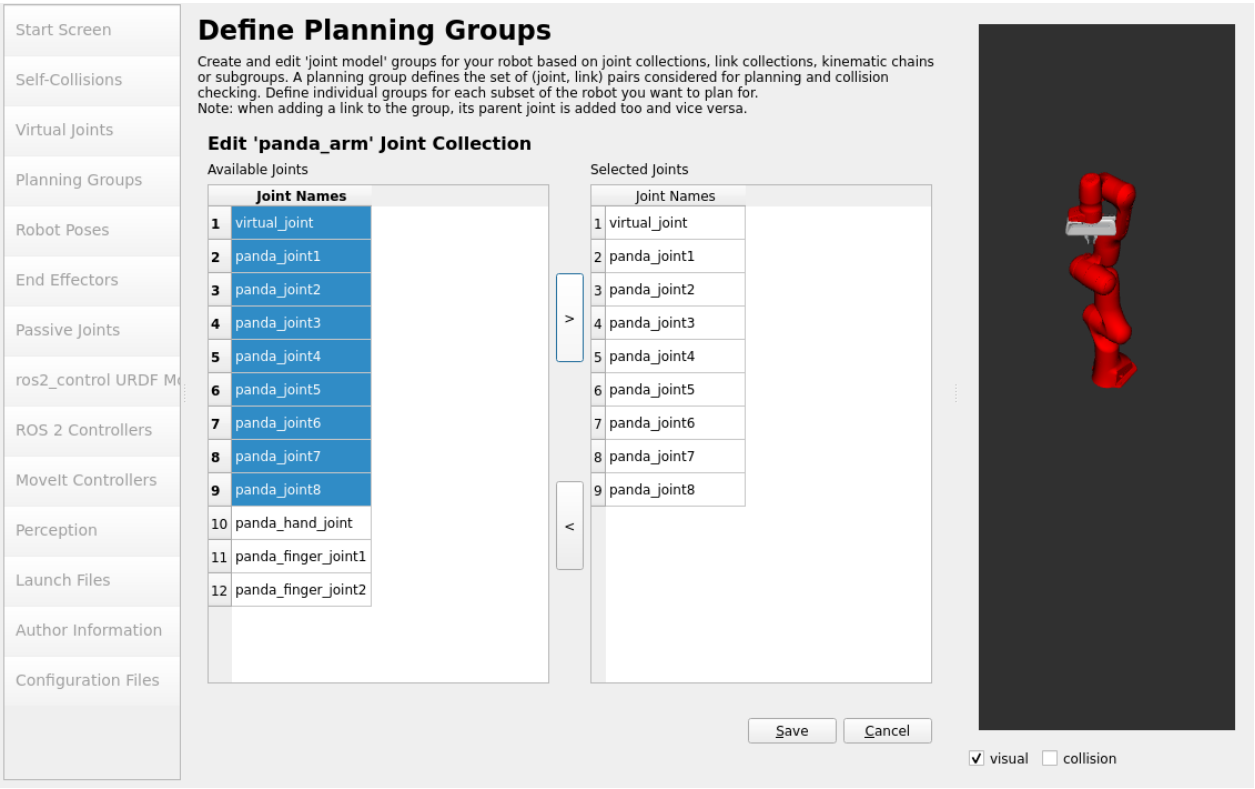
Στη συνέχεια γίνεται η μετάβαση στο πεδίο **Planning Groups**. Στο συγκεκριμένο πεδίο, περιγράφονται κάποια κομμάτια του ρομπότ όπως είναι η ο βραχίονας και το εργαλείο αυτού, με στόχο να διευκολυνθεί ο σχεδιασμός της κίνησης του ρομπότ.

Ένα κινούμενο γκρουπ μπορεί να διαμορφωθεί ώστε να είναι σε συμφωνία με μια κινηματική αλυσίδα ενός ρομπότ, το οποίο αποτελείται από διάφορα links και κάποια joints μεταξύ αυτών. Μέσω των links και joints θα καθοριστούν τα συστήματα συντεταγμένων από τη βάση του ρομπότ έως και το εργαλείο της άκρης.

Στο συγκεκριμένο πεδίο, θα πρέπει να καθοριστούν τα μέλη του ρομπότ. Για αρχή θα καθοριστεί σαν ένα κινητό μέλος, μόνο ο βραχίονας χωρίς το άκρο του. Πρέπει να δοθεί ένα όνομα για το κινητό μέλος και έπειτα θα καθοριστεί ο τρόπος με τον οποίο θα λύνεται το κινηματικό πρόβλημα. Στην συγκεκριμένη ιδιότητα θα επιλεγεί το **kdl_kinematics_plugin/KDLKinematicsPlugin** το οποίο είναι και η default παράμετρος για το MoveIt. Έπειτα θα καθοριστεί και το Group Default Planner και θα επιλεγεί το **BiTRRT**. Όλα τα παραπάνω αποτυπώνονται στην εικόνα 82. Στη συνέχεια επιλέγεται το **Add joints** και επιτυγχάνεται η μετάβαση στο γραφικό περιβάλλον της εικόνας 83.



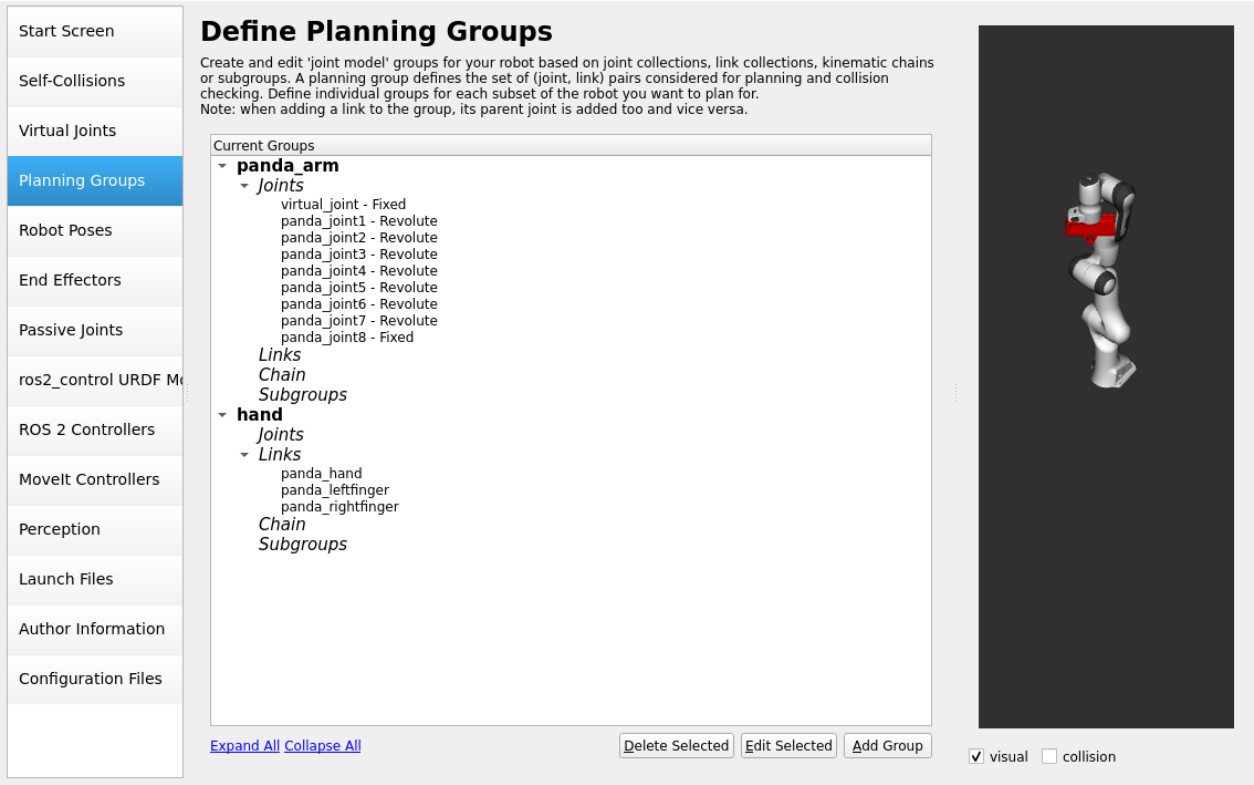

Εικόνα 82. Καθορισμός του βραχίονα στο Planning groups.




Εικόνα 83. Καθορισμός των joints για το panda_arm.

Στο παράθυρο αυτό θα πρέπει να επιλεγούν όλα τα μέλη του βραχίονα και έπειτα να γίνει η επιλογή του συμβόλου >, το οποίο βρίσκεται στο μέσο του γραφικού περιβάλλοντος. Οπότε από τη λίστα των Available Joints, έχουν μεταφερθεί κάποια στη λίστα των Selected Joints. Αφού επιλεγούν τα επιθυμητά joints τότε αποθηκεύεται η δουλειά που έχει γίνει μέχρι τώρα μέσω του **Save**.

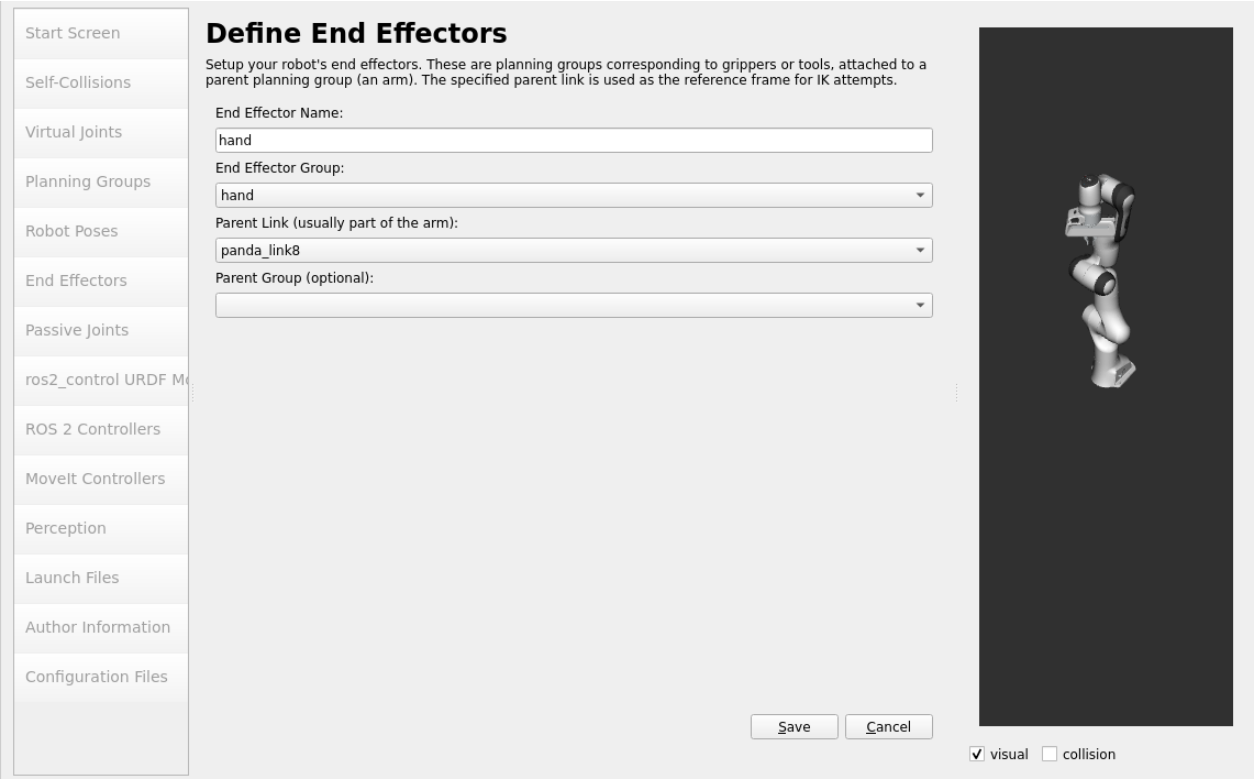
Μετά την αποθήκευση θα γίνει η μετάβαση στο γραφικό περιβάλλον του πεδίου **Planning Groups** και η διαδικασία που περιγράφεται παραπάνω θα πρέπει να πραγματοποιηθεί και για την αρπάγη του βραχίονα. Θα επιλεγεί το **Add Group** και θα προκύψει το γραφικό περιβάλλον της Εικόνας 82. Στις επιλογές που διατίθενται εκεί, θα δοθεί το όνομα hand ενώ οι άλλες επιλογές δεν θα αλλαχθούν. Σε αντίθεση με τον βραχίονα, για την αρπάγη θα γίνει η επιλογή του **Add Links**, και θα προκύψει το γραφικό περιβάλλον της Εικόνας 83. Θα επιλεγούν τα τρία τελευταία links και έπειτα θα πρέπει να επιλεγεί το σύμβολο > όπως και στην προηγούμενη περίπτωση. Ακολουθεί η αποθήκευση του και θα πρέπει να προκύψει το ακόλουθο.




Εικόνα 84. Καθορισμός του planning groups.

Στη συνέχεια γίνεται μετάβαση στο πεδίο **Robot Poses.** Εκεί μπορούν καθοριστούν κάποιες πόζες για το βραχίονα αλλά και για την αρπάγη. Το βήμα αυτό είναι προαιρετικό και για αυτό δεν παρουσιάζεται. Ακολουθεί η μετάβαση στο πεδίο **End Effector** στο οποίο θα καθοριστεί η αρπάγη ως το άκρο του βραχίονα. Το βήμα αυτό είναι απαραίτητο διότι καθορίζοντας της αρπάγη ως το άκρο του βραχίονα, το MoveIt μπορεί να εκτελέσει διάφορες εργασίες όπως για παράδειγμα να αρπάξει ένα αντικείμενο και να το μεταφέρει κάπου αποφεύγοντας τα εμπόδια που παρουσιάζονται στη πορεία του.

Στο παράθυρο του **End Effector**, θα πρέπει να δοθούν κάποιες παράμετροι οι οποίες είναι οι ακόλουθες. Δίνεται ένα όνομα στην επιλογή End Effector Name και θα δοθεί το όνομα hand. Έπειτα θα πρέπει να πραγματοποιηθούν αλλαγές στα **End Effector Group** και **Parent Link**. Θα τους δοθούν κατά αντιστοιχία οι επιλογές hand και panda_link8. Οι αλλαγές αυτές παρουσιάζονται στην επόμενη εικόνα και αφού πραγματοποιηθούν επιλέγεται το Save.

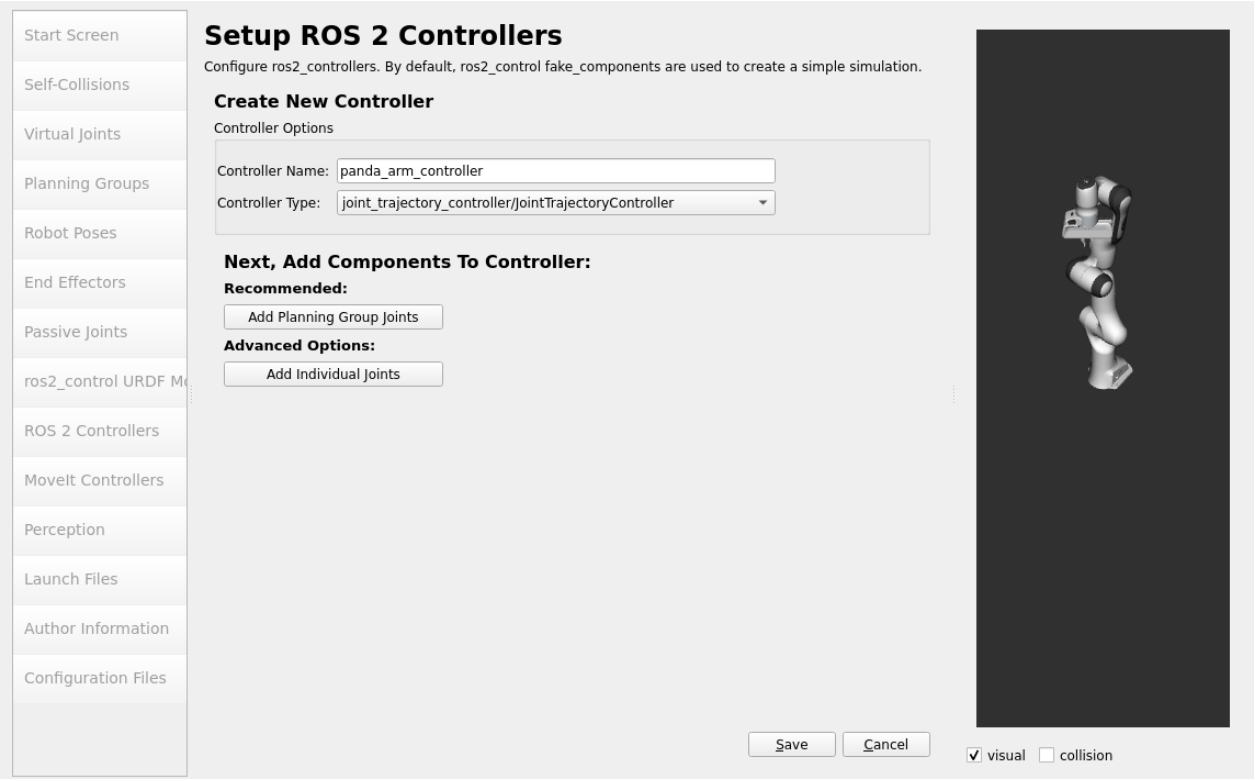



Εικόνα 85. Καθορισμός του άκρου του βραχίονα.

Έπειτα από τον καθορισμό του άκρου του βραχίονα, ακολουθεί ο ορισμός κάποιων passive joints που ίσως εμπεριέχονται στο βραχίονα. Ο βραχίονας panda που χρησιμοποιείται δεν διαθέτει ανενεργά joints και για αυτό το λόγο το βήμα αυτό είναι περιττό. Ωστόσο σε περίπτωση που ο βραχίονας εμπεριέχει τέτοιου τύπου joints τότε αυτά θα πρέπει να οριστούν στο συγκεκριμένο πεδίο.

Στη συνέχεια θα ακολουθήσει ο καθορισμός των controllers του ρομπότ. Ο βραχίονας που χρησιμοποιείται αλληλεπιδρά με το πακέτο ros2_control όπως και όλο το project του MoveIt. Το πεδίο **ros2 control URDF Modifications** είναι απαραίτητο σε περίπτωση που δεν έχει δημιουργηθεί ένα xacro file στο οποίο θα ορίζονται οι ελεγκτές του ρομπότ. Σε περίπτωση που έχει προηγηθεί η δημιουργία αυτού του αρχείου τότε το βήμα αυτό παραλείπεται.

Στη συνέχεια υπάρχει το πεδίο **ROS 2 Controllers**, το οποίο χρησιμοποιείται για την δημιουργία των ελεγκτών που θα θέσουν σε κίνηση της αρθρώσεις του ρομπότ. Στο γραφικό περιβάλλον του πεδίου επιλέγεται το Add Controller και γίνονται οι προσθήκες όπως παρουσιάζονται στην παρακάτω εικόνα.




Εικόνα 86. Δημιουργία ελεγκτή για τον βραχίονα.

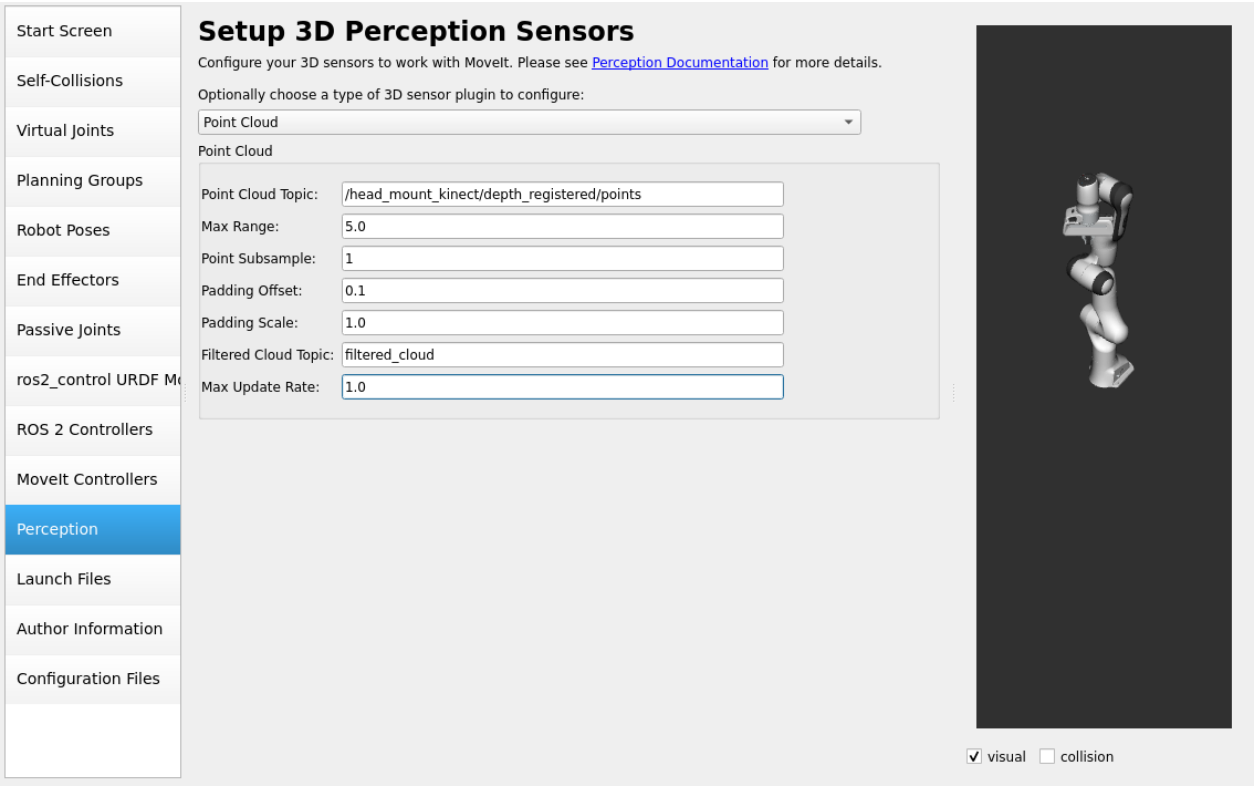
Αρχικά θα πρέπει να δοθεί ένα όνομα για τον ελεγκτή του βραχίονα και δίνεται το panda_arm_controller. Επίσης πρέπει να καθοριστεί ο τύπος του ελεγκτή και επιλέγεται το joint_trajectory_controller για τον ρομποτικό βραχίονα. Στην συνέχεια επιλέγεται το **Add Planning Group Joints** και θα ακολουθήσει η μετάβαση σε ένα άλλο παράθυρο. Στο συγκεκριμένο παράθυρο θα πρέπει να γίνει η επιλογή του panda arm και να γίνει η επιλογή του συμβόλου > που βρίσκεται ενδιάμεσα από τις στήλες **Available Groups** και **Selected Groups**. Θα ακολουθήσει το **Save** και γίνεται η επιστροφή στο περιβάλλον της Εικόνας 86.

Η παραπάνω διαδικασία θα πραγματοποιηθεί και για την αρπάγη. Θα δοθεί το όνομα hand_controller και ο τύπος του ελεγκτή θα είναι ο position_controller. Έπειτα θα επιλεγεί το **Add Planning Group Joints** και εκεί θα ακολουθήσει η επιλογή του hand και ακολούθως το σύμβολο > . Τέλος, θα γίνει η αποθήκευση και στο σημείο αυτό θα έχουν προστεθεί οι ελεγκτές του βραχίονα αλλά και τη αρπάγης.

Το πεδίο του **MoveIt controllers** είναι το τελικό στάδιο για τον καθορισμό των ελεγκτών του βραχίονα και της αρπάγης. Το πακέτο απαιτεί ελεγκτές του τύπου **FollowJointTrajectoryAction** ώστε να μπορέσει να καθορίσει τις τροχιές που θα ακολουθήσουν τα κινητά κομμάτια του ρομπότ.

Συνεπώς στο πεδίο του MoveIt Controllers, θα γίνει μια επιπρόσθετη διαδικασία για την μετατροπή των controllers στον είδος ελεγκτή που θα δέχεται το πακέτο. Πιο συγκεκριμένα, στο πεδίο του MoveIt controller θα δοθεί το όνομα panda_arm_controller, στην επιλογή **Controller Type** θα επιλεχθεί το **FollowJointTrajectory** και έπειτα θα γίνει αποθήκευση. Η διαδικασία θα πραγματοποιηθεί άλλη μια φορά, αλλά για την αρπάγη. Θα δοθεί το όνομα hand_controller και στον τύπο του ελεγκτή το **GripperCommand**. Τέλος, θα ακολουθήσει η αποθήκευση του και γίνεται η μετάβαση στο επόμενο πεδίο το οποίο είναι το **Perception**.

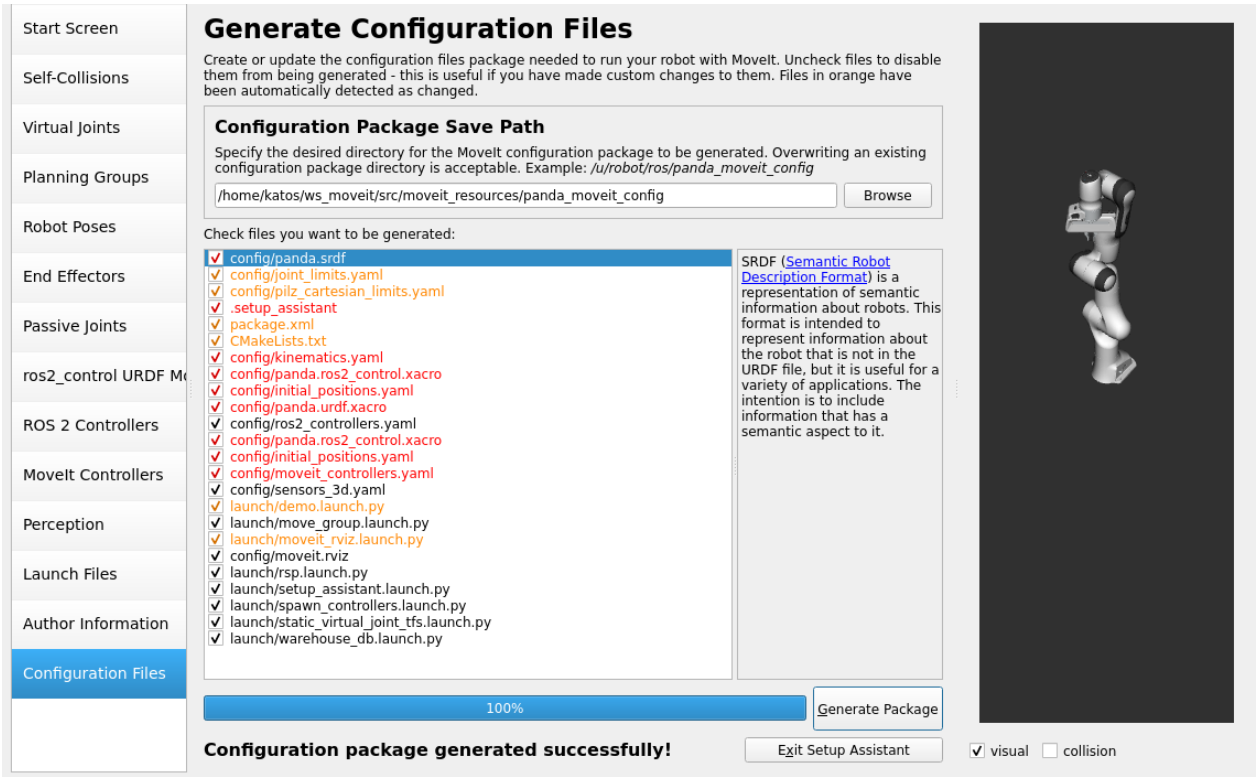
Το πεδίο του Perception είναι υπεύθυνο για την διαμόρφωση των ρυθμίσεων των αισθητήρων που διαθέτει το ρομπότ. Οι ρυθμίσεις αυτές είναι αποθηκευμένες σε αρχεία τύπου yaml και συγκεκριμένα στο αρχείο με όνομα sensors_3d.yaml. Στο παράθυρο του Perception, γίνεται η επιλογή του **Point Cloud** και ακολούθως γίνεται η προσθήκη των επιλογών όπως παρουσιάζονται στην παρακάτω εικόνα.




Εικόνα 87. Καθορισμός των παραμέτρων για τον 3D sensor.

Στη συνέχεια γίνεται η μετάβαση στο πεδίο **Launch Files**. Στο συγκεκριμένο πεδίο δημιουργείται το launch αρχείο που θα δημιουργηθεί για την εκκίνηση του MoveIt,των ελεγκτών και του Rviz. Το αρχείο αυτό θα δημιουργηθεί μέσω γραφικού περιβάλλοντος επιλέγοντας τα nodes που είναι επιθυμητό να εκκινήσουν. Στο παράθυρο αυτό δεν θα πραγματοποιηθούν κάποιες τροποποιήσεις καθώς είναι επιθυμητό να εκκινήσουν όλα τα nodes που είναι επιλεγμένα. Στο πεδίο **Author Information**, θα προστεθούν κάποιες πληροφορίες όπως το όνομα του χρήστη που δημιούργησε το πακέτο.

Το τελευταίο βήμα είναι το Configuration των αρχείων. Γίνεται η επιλογή του Browse με στόχο να του δοθεί ένα μονοπάτι το οποίο είναι το /home/<user>/ws_moveit2/src/panda_moveit_config. Έπειτα από την επιλογή του μονοπατιού, θα εμφανιστούν οι επιλογές που παρουσιάζονται στην παρακάτω εικόνα. Ακολούθως θα επιλεγεί το **Generate Package** ώστε να δημιουργηθεί το πακέτο και να παραχθούν όλα τα αρχεία που είναι απαραίτητο για την εφαρμογή του MoveIt στο βραχίονα. Στο σημείο αυτό το γραφικό περιβάλλον θα πρέπει να είναι το ακόλουθο και το πακέτο θα έχει δημιουργηθεί και θα μπορεί να χρησιμοποιηθεί.

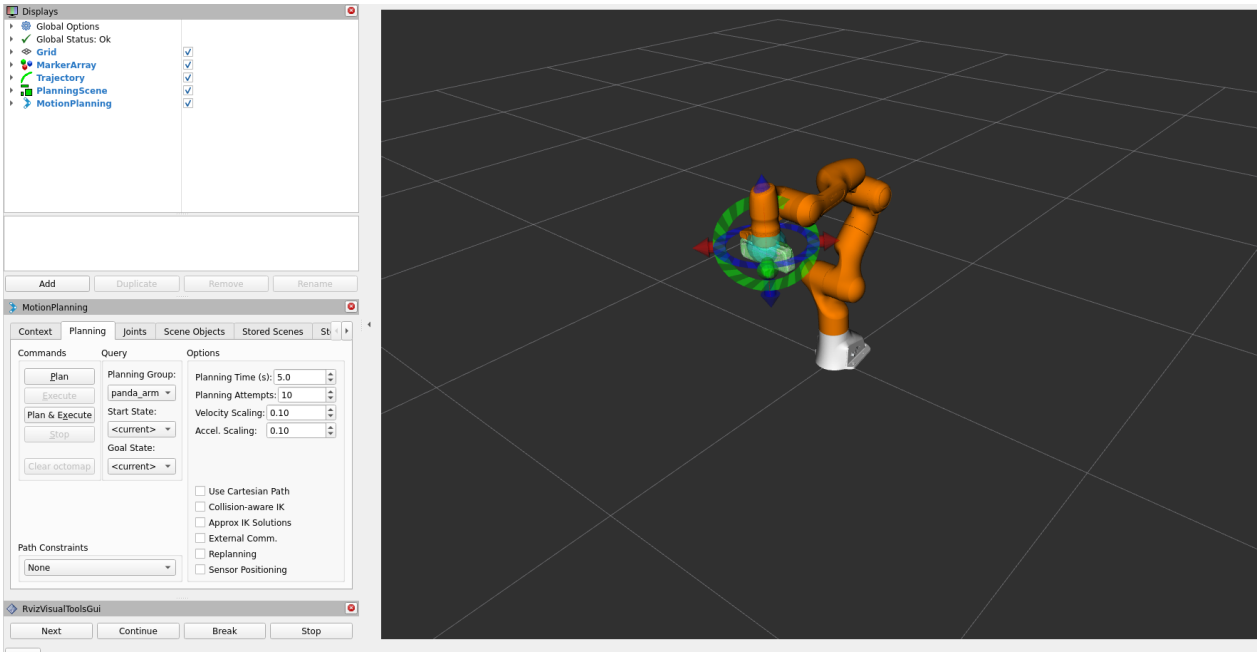



Εικόνα 88. Επιτυχής δημιουργία του πακέτου.

Στη συνέχεια, για να επιβεβαιωθεί η ορθή δημιουργία της προσομοίωσης που έχει δημιουργηθεί, θα πρέπει να εκτελεστεί το launch αρχείο που παράχθηκε μέσω γραφικού περιβάλλοντος. Θα πρέπει αρχικά να χτιστεί το πακέτο και στη συνέχεια να εκτελεστεί το αρχείο. Συνεπώς εκτελούνται οι παρακάτω εντολές.

```
$ cd ~/ws_moveit2

$ colcon build --packages-select panda_moveit_config

$ export LIBGL_ALWAYS_SOFTWARE=1 LIBGL_ALWAYS_INDIRECT=0

$ ros2 launch moveit_resources_panda_moveit_config demo.launch.py
```

Πίνακας 117. Χτίσιμο πακέτου και εκκίνηση του launch file.




Εικόνα 89. Ο βραχίονας στο Rviz.

## Κεφάλαιο 7

## Συμπεράσματα και Μελλοντική Χρήση

Το ROS 2 είναι ένα λογισμικού που αναπτύσσεται διαρκώς και η χρήση του οποίου θα έχει ραγδαία ανάπτυξη τα επόμενα χρόνια. Ο λόγος που θα συμβεί αυτό είναι ευκολία εφαρμογής του λογισμικού στην εκπαιδευτική κοινότητα αλλά και σε ερευνητικές εργασίες. Επιπροσθέτως στόχος του ROS 2 είναι οι επέκταση των πακέτων και εφαρμογών ώστε το λογισμικό να υποστηρίζεται από εταιρίες οι οποίες θα παράγουν προϊόντα βασισμένα σε αυτό.

Συνολικά τα πακέτα και ο κώδικας που δημιουργήθηκαν, έχουν σαν στόχο να εισάγωγουν τους αναγνώστες στο κόσμο του ROS 2 και της ρομποτικής. Καθόλη την εργασία, γίνεται αναφορά στις δύο βασικές κατηγορίες ρομπότ που είναι τα αυτόνομα ρομποτικά συστήματα και στους ρομποτικούς βραχίονες και σε κάποια χαρακτηριστικά αυτών. Παράλληλα, αναφέρονται και πιο σύνθετα θέματα ρομποτικής όπως είναι η πλοήγηση και η χαρτογράφηση των ρομπότ στο χώρο εργασίας τους. Μέσω αυτών των ερεθισμάτων ο αναγνώστης μπορεί να αναζητήσει τα σημεία ενδιαφέροντος του και να αποκτήσει νέες γνώσεις μέσω της αναζήτησης των θεμάτων αρεσκείας του και σε συνδυασμό με τις δυνατότητες που προσφέρει το ROS 2 να αναπτύξει τις δικές του εφαρμογές.

Συνολικά, το ROS 2 αποτελείται από αρκετά πακέτα, ωστόσο οι βασικοί πυλώνες του είναι τέσσερις βιβλιοθήκες, οι οποίες είναι οι ακόλουθες:

- Ros2_control: βιβλιοθήκη πραγματικού χρόνου αφιερωμένη στον έλεγχο των ρομπότ.
- Nav2: βιβλιοθήκη αφιερωμένη στην αυτονομία των ρομπότ. Προσφέρονται πακέτα πλοήγησης χαρτογράφησης και άλλα.
- MoveIt2: βιβλιοθήκη αφιερωμένη στην αλληλεπίδραση με ρομποτικούς βραχίονες.
- Micro-Ros: είναι μια βιβλιοθήκη μέσω της οποίας μπορεί να γίνει η χρήση του ROS2 με microcontrollers.

Τα πακέτα αυτά μπορούν να συνδυαστούν είτε ταυτόχρονα είτε ξεχωριστά και να δημιουργήσουν αυτόνομα ρομπότ παντός τύπου, τα οποία μπορούν να αναπτυχθούν από μεγάλες κοινότητες αλλά και ερευνητικές ομάδες.

Όσον αφορά το μέλλον του ROS 2, αυτό θα συνεχίσει να εξελίσσεται με ραγδαίους ρυθμούς καθώς αναμένεται σύντομα ένα νέο distribution του λογισμικού βασισμένο στην νέα έκδοση των Linux, Ubuntu

24.04. Η νεότερη έκδοση του ROS 2, Jazzy Jalisco αναμένεται να αναβαθμίσει περαιτέρω τα πακέτα των προηγούμενων εκδόσεων. Η συγκεκριμένη έκδοση του ROS θα μπορεί να υποστηρίξει πλήρως τη νέα πλατφόρμα προσομοίωσης του Gazebo.

Κλείνοντας, άξιο αναφοράς αποτελεί η διαρκώς αυξανόμενη χρήση της τεχνητής νοημοσύνης στο κόσμο της πληροφορικής. Η εφαρμογή τεχνικών όπως η μηχανική μάθηση, τα νευρωνικά δίκτυα, η βαθιά μάθηση αλλά και η τεχνητή νοημοσύνη σε ρομπότ μπορεί να οδηγήσει σε μια νέας και πιο έξυπνης γενιάς ρομπότ, που θα μπορεί να βοηθήσει τον άνθρωπο στην καθημερινότητα αλλά και σε πιο απαιτητικές εργασίες όπως ο αγροτικός ή κατασκευαστικός τομέας.

**Βιβλιογραφία**

[1] Fairchild, C., & Harman, T. L. (2017). *ROS Robotics By Example*. Packt Publishing.

[2] Siegwart, R., Nourbakhsh, I.R., & Scaramuzza, D. (2011). Introduction to Autonomous Moblie Robots (2nd ed.). The MIT Press.

[3] Ros 2 Humble Documentation. Διαθέσιμο στην ηλεκτρονική διεύθυνση:

https://docs.ros.org/en/humble/index.html.

[4] ros2_control documentation – Humble . Διαθέσιμο στην ηλεκτρονική διεύθυνση:

https://control.ros.org/humble/index.html.

[5] Nav2 documentation. Διαθέσιμο στην ηλεκτρονική διεύθυνση: https://navigation.ros.org/index.html.

[6] MoveIt2 Documentation. Διαθέσιμο στην ηλεκτρονική διεύθυνση:

https://moveit.picknik.ai/main/index.html.

[7] Οι διαφορές του ROS με το ROS 2 μπορούν να βρεθούν στο άρθρο: https://roboticsbackend.com/ros1-vs-ros2-practical-overview/.

[8] Διαφορές του ROS με το ROS2 μπορούν να βρεθούν στο άρθρο: https://medium.com/@oelmofty/ros2-how-is-it-better-than-ros1-881632e1979a.

[9] Διαφορές του ROS με το ROS2 μπορούν να βρεθούν στον σύνδεσμο:

https://design.ros2.org/articles/changes.html.

[10] Πληροφορίες για το turtlebot αντλήθηκαν από τον σύνδεσμο:

https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/.

[11] Το πακέτο micro-ROS μπορεί να βρεθεί στον σύνδεσμο: https://micro.ros.org/.

[12] Tutorials και βίντεο μπορούν να βρεθούν στον σύνδεσμο: https://articulatedrobotics.xyz/.