
Αυτόνομοι Πράκτορες

Αναφορά 4ης Εργαστηριακής Άσκησης

Πέτρου Δημήτριος - 2018030070

Χανιά, Ιανουάριος 2022

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

1. Εισαγωγή

Αντικείμενο μελέτης της 4ης εργαστηριακής άσκησης ήταν η εγκατάσταση και η ενσωμάτωση του R.O.S.(Robot Operating System) στον προσομοιωτή Gazebo με περαιτέρω πειραματισμό επ αυτού. Το ROS παρέχει συλλογές από εργαλεία, διεπαφές και βιβλιοθήκες που αποτελούν βοηθό στην ανάπτυξη ρομποτικών συστημάτων λογισμικού σε ποίκιλες πλατφόρμες ρομποτικού hardware. Παρέχονται μεταξύ άλλων εργαλεία διεπαφής με τους αισθητήρες αντίληψης ενός ρομπότ που προσομοιώνεται ενεργά στο Gazebo, βιβλιοθήκες αλληλεπίδρασης με τα κινητά μέρη του ρομπότ καθώς και έτοιμο λογισμικό προς εκμετάλλευση.

2. Εγκατάσταση & Εκκίνηση Εργαλείων

Η εγκατάσταση του ROS έγινε σε Ubuntu Mate 20.04 LTS το οποίο στηρίζεται στην ίδια έκδοση Linux Kernel με αυτό του Ubuntu 20.04. Σύμφωνα με τις οδηγίες επιλέχθηκε η Noetic Ninjemys έκδοση του ROS και πραγματοποιήθηκε η εγκατάσταση ακολουθώντας το tutorial που παρέχεται. Το μοντέλο ρομπότ το οποίο χρησιμοποιήθηκε στην προσομοίωση ήταν το Husky της Clearpath Robotics. Αφού λήφθηκαν τα απαραίτητα αρχεία και πακέτα εντός του directory `catkin_ws` έγινε build με την χρήση της εντολής `catkin make`. Η εντολή πραγματοποίησε build, χωρίς ωστόσο να υπάρχει κάποιο πακέτο που να συνδέει το ROS με το Gazebo. Το αποτέλεσμα εκτέλεσης φαίνεται παρακάτω:

```
dpetrou@mate:~/catkin_ws$ catkin_make
Base path: /home/dpetrou/catkin_ws
Source space: /home/dpetrou/catkin_ws/src
Build space: /home/dpetrou/catkin_ws/build
Devel space: /home/dpetrou/catkin_ws/devel
Install space: /home/dpetrou/catkin_ws/install
Creating symlink "/home/dpetrou/catkin_ws/src/CMakeLists.txt" pointing to "/opt/ros/noetic/share/catkin/cmake/../../../src/CMakeLists.txt"
####
#### Running command: "cmake /home/dpetrou/catkin_ws/src -DCATKIN_DEVEL_PREFIX=/home/dpetrou/catkin_ws/devel --no-deps"
####
-- The C compiler identification is GNU 9.3.0
-- The CXX compiler identification is GNU 9.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using CATKIN_DEVEL_PREFIX: /home/dpetrou/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: /opt/ros/noetic
-- This workspace overlays: /opt/ros/noetic
-- Found PythonInterp: /usr/bin/python3 (found suitable version "3.8.10", minimum required is "3")
-- Using PYTHON_EXECUTABLE: /usr/bin/python3
-- Using Debian Python package layout
-- Found PY_en: /usr/lib/python3/dist-packages/en.py
-- Using empy: /usr/lib/python3/dist-packages/en.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: /home/dpetrou/catkin_ws/build/test_results
-- Forcing gtest/gmock from source, though one was otherwise available.
-- Found gtest sources under '/usr/src/gtest': gtests will be built
-- Found gmock sources under '/usr/src/gmock': gmock will be built
-- Found PythonInterp: /usr/bin/python3 (found version "3.8.10")
-- Found Threads: TRUE
-- Using Python nosetests: /usr/bin/nosetests3
-- catkin 0.8.10
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dpetrou/catkin_ws/build
####
#### Running command: "make -j2 -l2" in "/home/dpetrou/catkin_ws/build"
####
```

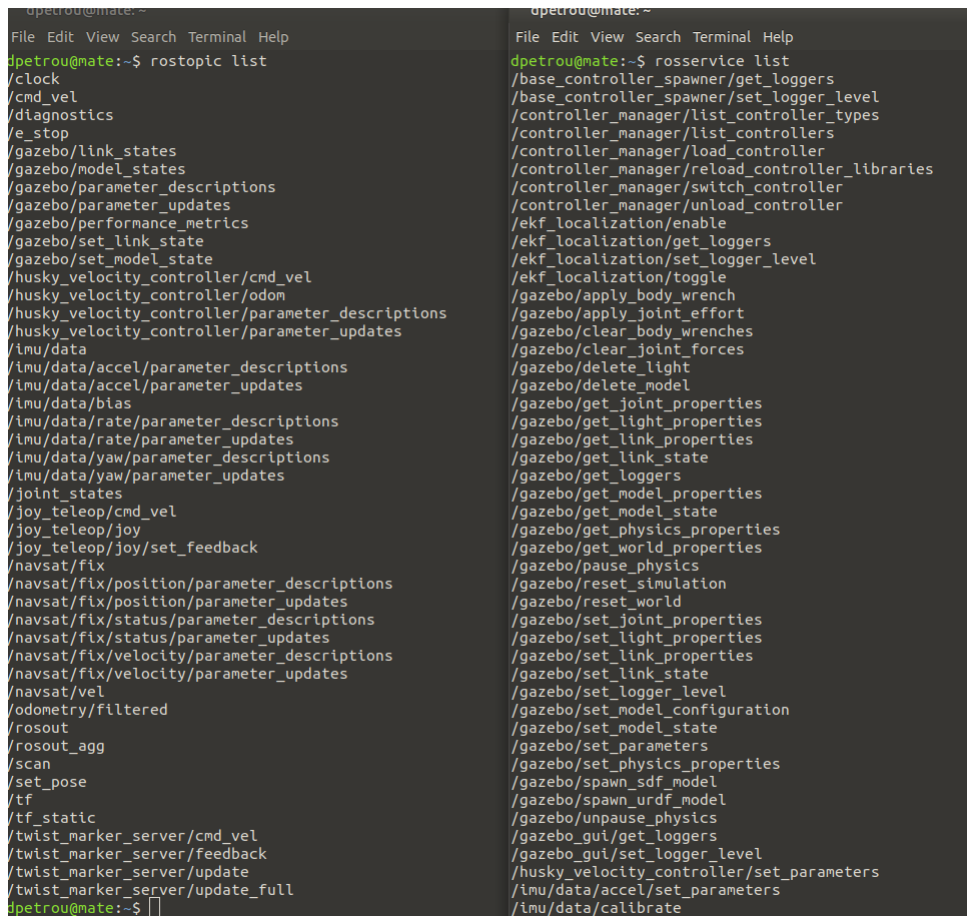
Στιγμιότυπο 1: Εκτέλεση `catkin make` χωρίς πακέτα διασύνδεσης

Στη συνέχεια αφού λήφθηκαν τα πακέτα που υλοποιούν την διασύνδεση του ROS με το Gazebo επαναλήφθηκε το build και έπειτα από μια μακροσκελή διεργασία η προσομοίωση στο Gazebo εκκινήθηκε με ενσωματωμένο το ROS χρησιμοποιώντας την εντολή:

```
roslaunch husky_gazebo husky_empty_world.launch
```

Σημειώνεται πως είχε προηγηθεί η ενεργοποίηση του αισθητήρα LiDAR (μέσω κατάλληλης μεταβλητής στο αρχείο προσομοίωσης του μοντέλου) του Husky προκειμένου να μελετηθούν τα εργαλεία απεικόνισης δεδομένων που παρέχονται από το ROS.

Αμέσως μετά την εκκίνηση, εκτελέστηκαν οι εντολές `rostopic list` και `rosservice list` οι οποίες δείχνουν όλα τα διαθέσιμα ROS topics (υπεύθυνα για μεταφορά πληροφοριών των αισθητήρων) και services (υπεύθυνα για την υλοποίηση των διεπαφών ανταλλαγής δεδομένων). Τα αποτελέσματα της εκτέλεσης φαίνονται παρακάτω:



```
dpetrou@mate:~$ rostopic list
/clock
/cmd_vel
/diagnostics
/e_stop
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/performance_metrics
/gazebo/set_link_state
/gazebo/set_model_state
/husky_velocity_controller/cmd_vel
/husky_velocity_controller/odom
/husky_velocity_controller/parameter_descriptions
/husky_velocity_controller/parameter_updates
/imu/data
/imu/data/accel/parameter_descriptions
/imu/data/accel/parameter_updates
/imu/data/bias
/imu/data/rate/parameter_descriptions
/imu/data/rate/parameter_updates
/imu/data/yaw/parameter_descriptions
/imu/data/yaw/parameter_updates
/joint_states
/joy_teleop/cmd_vel
/joy_teleop/joy
/joy_teleop/joy/set_feedback
/navsat/fix
/navsat/fix/position/parameter_descriptions
/navsat/fix/position/parameter_updates
/navsat/fix/status/parameter_descriptions
/navsat/fix/status/parameter_updates
/navsat/fix/velocity/parameter_descriptions
/navsat/fix/velocity/parameter_updates
/navsat/vel
/odometry/filtered
/rosout
/rosout_agg
/scan
/set_pose
/tf
/tf_static
/twist_marker_server/cmd_vel
/twist_marker_server/feedback
/twist_marker_server/update
/twist_marker_server/update_full
dpetrou@mate:~$
```

```
dpetrou@mate:~$ rosservice list
/base_controller_spawner/get_loggers
/base_controller_spawner/set_logger_level
/controller_manager/list_controller_types
/controller_manager/list_controllers
/controller_manager/load_controller
/controller_manager/reload_controller_libraries
/controller_manager/switch_controller
/controller_manager/unload_controller
/ekf_localization/enable
/ekf_localization/get_loggers
/ekf_localization/set_logger_level
/ekf_localization/toggle
/gazebo/apply_body_wrench
/gazebo/apply_joint_effort
/gazebo/clear_body_wrenches
/gazebo/clear_joint_forces
/gazebo/delete_light
/gazebo/delete_model
/gazebo/get_joint_properties
/gazebo/get_light_properties
/gazebo/get_link_properties
/gazebo/get_link_state
/gazebo/get_loggers
/gazebo/get_model_properties
/gazebo/get_model_state
/gazebo/get_physics_properties
/gazebo/get_world_properties
/gazebo/pause_physics
/gazebo/reset_simulation
/gazebo/reset_world
/gazebo/set_joint_properties
/gazebo/set_light_properties
/gazebo/set_link_properties
/gazebo/set_link_state
/gazebo/set_logger_level
/gazebo/set_model_configuration
/gazebo/set_model_state
/gazebo/set_parameters
/gazebo/set_physics_properties
/gazebo/spawn_sdf_model
/gazebo/spawn_urdf_model
/gazebo/unpause_physics
/gazebo_gui/get_loggers
/gazebo_gui/set_logger_level
/husky_velocity_controller/set_parameters
/imu/data/accel/set_parameters
/imu/data/calibrate
dpetrou@mate:~$
```

Στιγμιότυπο 2: Εκτέλεση `rostopic list` και `rosservice list`

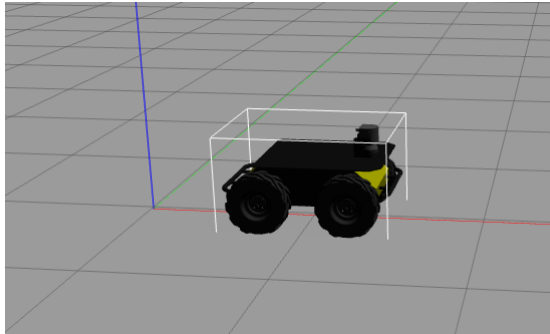
[illegible]

3

Δοκιμάζοντας σε 2η φάση τα services εκτελέστηκε η εντολή:

```
rosservice call /gazebo/reset_world
```

η οποία επαναφέρει τον κόσμο στο Gazebo στην αρχική κατάσταση, τοποθετώντας παράλληλα το Husky στην θέση 0,0,0. Προκειμένου να φανεί η διαφορά που επιφέρει η εκτέλεση της εντολής το Husky τοποθετήθηκε από πριν σε μια θέση διαφορετική της 0,0,0. Το αποτέλεσμα φαίνεται παρακάτω: Το ROS προσφέρει επίσης το



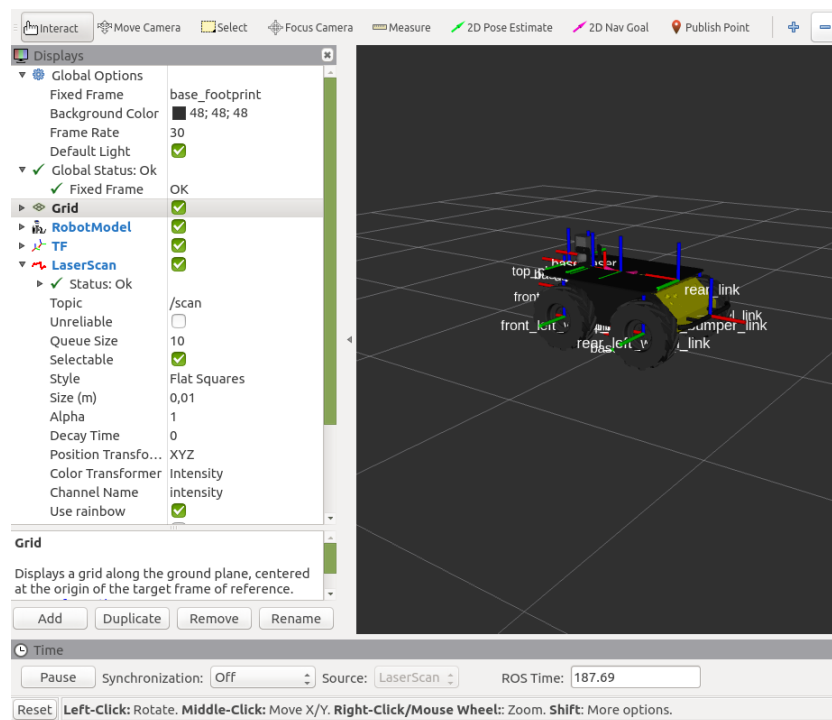
(a) Πριν



(b) Μετά

Στιγμιότυπο 4: Εκτέλεση εντολής `rosservice call /gazebo/reset_world`

εργαλείο RVIZ, το οποίο δείχνει την κατάσταση των υπαρκτών ρομπότ με τις μεταξύ τους χωρικές συσχετίσεις, όπως και τα δεδομένα που λαμβάνουν οι αισθητήρες τους, σε πραγματικό χρόνο. Με την εκτέλεση της εντολής `rviz` το εργαλείο εκκινείται. Στο γραφικό περιβάλλον προστέθηκαν με τη χρήση του Add το RobotModel, το TF (δέντρο συστήματων συντεταγμένων) και το LaserScan. Για σύστημα συντεταγμένων χρησιμοποιήθηκε το `base_footprint`:



Στιγμιότυπο 5: RVIZ με την προσθήκη όλων των απεικονίσεων

The screenshot displays the ROS2 GUI interface. On the left, the 'Displays' panel is open, showing a list of components and their status. The 'Fixed Frame' is set to 'base_footprint'. The 'Global Status' is 'Ok'. The 'Grid' is visible. The 'RobotModel' is loaded. The 'TF' is visible. The 'LaserScan' is visible. The 3D view on the right shows a robot model with labels for its parts: 'base', 'top_plate', 'front_link', 'rear_link', and 'wheel_link'. A red laser scan arc is visible on the ground plane.

Component	Status
Global Options	base_footprint
Fixed Frame	base_footprint
Background Color	48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: Ok	✓
Fixed Frame	OK
Grid	✓
RobotModel	✓
TF	✓
LaserScan	✓

Fixed Frame
Frame into which all data is transformed before being displayed.

Buttons: Add, Duplicate, Remove, Rename

Time: ROS Time: 524.89, ROS Elapsed: 32.63, Wall Time: 1641856639.51, Wall Elapsed: 83.44

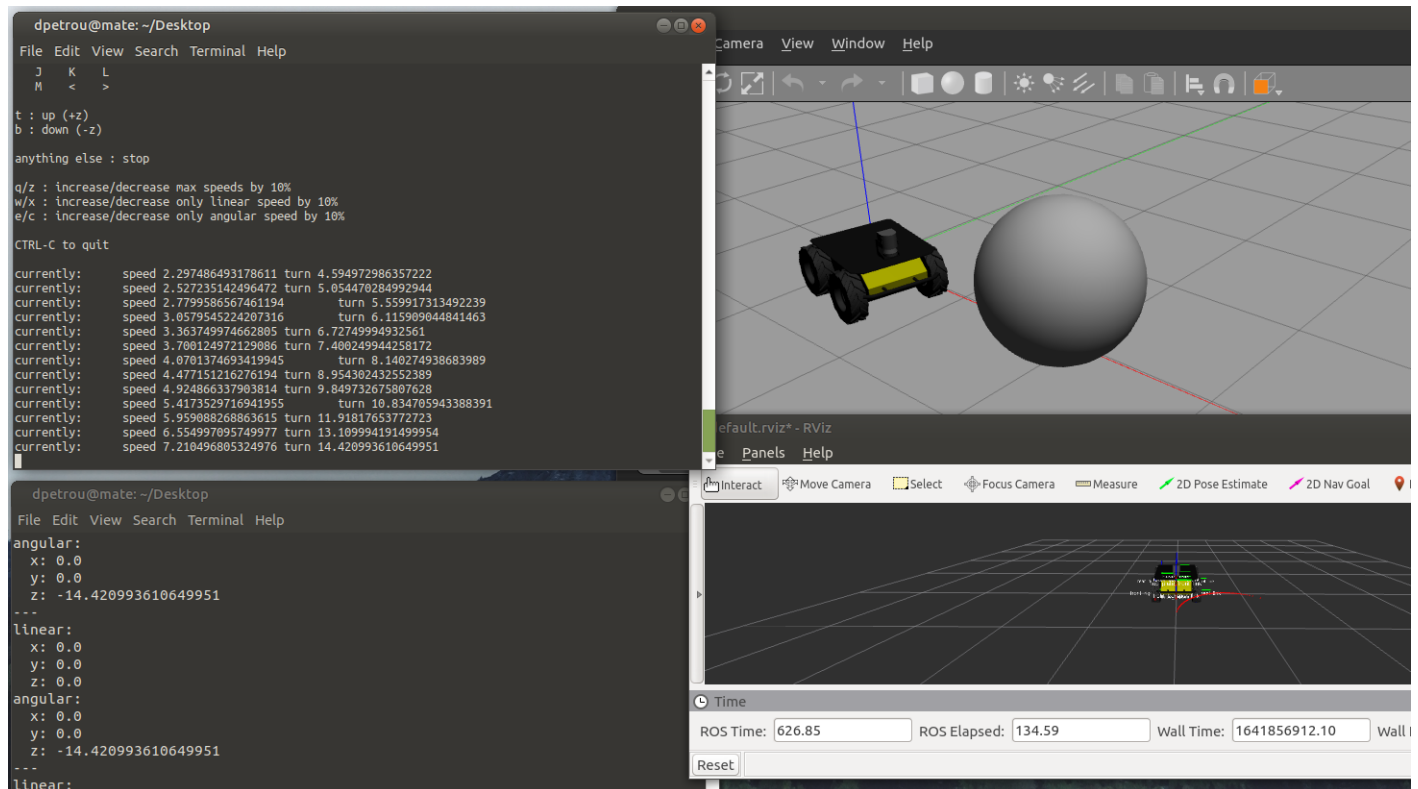
[illegible]

5

Τέλος, για τον τηλεχειρισμό από το πληκτρολόγιο και την παρακολούθηση των εκτελούμενων στο ρομπότ εντολών εκτελέστηκαν σε δύο τερματικά οι εντολές:

```
roslaunch teleop_twist_keyboard teleop_twist_keyboard.py
rostopic echo /husky_velocity_controller/cmd_vel
```

Ο τηλεχειρισμός του Husky με τη χρήση του πληκτρολογίου ήταν επιτυχής και έγινε σύμφωνα με το manual που παρέχεται στην αρχή της εκτέλεσης. Στο RVIZ απεικονίζονταν σε πραγματικό χρόνο η χωρική συσχέτιση του Husky και του εμποδίου ενώ σε άλλο τερματικό φαινόταν τα αποτελέσματα των επενεργούντων εντολών:



Στιγμιότυπο 8: Παράλληλη εκτέλεση εργαλείων και τηλεχειρισμός Husky