education_robotics Documentation

Release 0.1

Francisco J. Garcia R.

Sep 27, 2017

Contents

ROS packages for teaching robotics and control systems.

Features:

- ROS (Robotic operating system) (ubuntu 16.04 - ROS kinetic)
- Simulation of robots using Gazebo 7
- Step by step tutorials for controlling and interfacing robots

CHAPTER 1

Contents:

Description of project

This project aims to provide a simple step by step tutorials and exercises in order to start with robot programming.

Developed using ROS (Robotic operating system) Kinetic under Ubuntu 14.04 Source code was written in Python and C++

Suggested tutorials

It is advisable to complete first two basic tutorials:

- ROS (Robotic operating system) http://wiki.ros.org/ROS/Tutorials
- Gazebo simulator: http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1

Developed by:

- Francisco J. Garcia R. - garciar@rhrk.uni-kl.de
- 13. Sc. Alen Turnwald - turnwald@eit.uni-kl.de

Institute for control systems

University of Kaiserslautern 2016

How to install education robotics

In order to use this project, we must install first ROS (tested on Jade and kinectic), then compile the project and finally run some simulations and tests.

you can choose between two options:

- Use virtual machine which has already everything ready (windows users)
- Install ROS and dependencies in your own ubuntu computer

Prepare virtual machine

- Download virtual machine from: https://goo.gl/gnFGAz
- Install VMware player (free for non-commercial application) from: http://www.vmware.com/products/player/playerpro-evaluation.html
- Uncompress virtual machine and open it using VMware workstation player
- Please disable 3D acceleration graphics, it is not supported by gazebo simulator
- the user is : **ubuntu** and password: **ubuntu**

If you get access to virtual machine successfully, then you can go to last section "Test project".

If you have an ubuntu PC and want to install the project locally, please follow the next tutorial:

Install ROS and dependencies

Follow ros installation procedure:

http://wiki.ros.org/kinetic/Installation/Ubuntu

we can summarize the steps:

Open a command windows on ubuntu and run the following commands:

- Prepare ubuntu for installation:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /
↪etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 0xB01FA116
sudo apt-get update
```

- Install ROS

For PC/Laptop we should install full desktop version:

```
sudo apt-get install ros-kinetic-desktop-full
```

For Raspberry PI 2 and Odroid we can install ROS-Base:

```
sudo apt-get install ros-kinetic-ros-base
```

Install rosdep:

```
sudo rosdep init
rosdep update
```

And finally prepare environment:

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

Compile project (Only for local installation)

This step is only required if you install the project locally, please skip it if you use virtual machine.

First we need to install some dependencies and ROS packages:

```
sudo apt-get install libqwt-dev ros-kinetic-teleop-twist-joy  ros-kinetic-rviz-imu-
↪plugin python-smbus ros-kinetic-rqt-multiplot git
```

Finally we create a workspace for project, clone github repository, install dependencies and compile it:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
git clone https://github.com/francisc0garcia/education_robotics
cd ..
catkin_make
```

If you get some error during compilation, check if all dependencies are installed using:

Test project

Once the project has been compiled successfully, we can run a simulation that includes a simple robot + environment.

```
cd ~/catkin_ws
source devel/setup.bash
roslaunch education_robotics demo_robot_simple.launch
```

if everything is correct, you should see a robot moving with predefined steps.

Now you are ready to play and extend the project, let's go to section Tutorials and extensions.

Introduction tutorials

This section present some examples of how to use gazebo simulator and ROS in order to play with the robot and develop some extensions.

This tutorials assume you have already installed and tested project, see installation page for more details.

Getting started

You can run the test demo by typing in a command window:

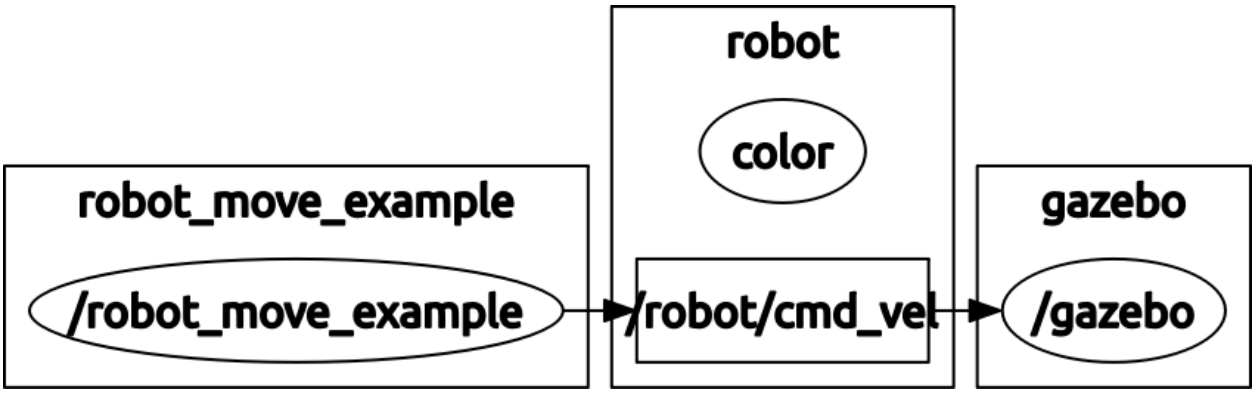
```
cd ~/catkin_ws
source devel/setup.bash
roslaunch education_robotics demo_robot_simple.launch
```

This demo (demo_robot_simple.launch) shows a simple example of how ROS interface with gazebo.

The demo is composed by the following nodes (processes):

- **robot_move_example:** A python scripts that send command velocities for robot (/robot/cmd_vel), placed on: **education_robotics/src/robot_move_example.py**
- **gazebo:** Simulator: shows and moves the robot

Now we can check how this nodes are connected using Node graph:



As you can see here, the node **robot_move_example:** sends the command **/robot/cmd_vel** to the node **gazebo**.

Let's have a look at some parts of **robot_move_example.py** script:

- The first part of the code creates a new ROS node, a publisher that send command velocities to the robot (self.cmd_vel_pub) and define the update speed (self.rate)

```
# Init ros node
rospy.init_node('robot_cmd_vel')

# create a publisher for command velocity
self.cmd_vel_pub = rospy.Publisher('/robot/cmd_vel', Twist, queue_size=1)

# define rate of transmission: 10 hz
self.rate = rospy.Rate(10.0)
```

- Now we create a infinity loop that sends commands to robot using some methods:

```
while not rospy.is_shutdown():

self.move_forward()
time.sleep(3)

self.rotate_left()
time.sleep(2)

self.stop_robot()
```

```
time.sleep(1)
.......
```

- If we take a look at one of this methods, we can realize that it only changes velocities for linear and angular components, in this case (move_forward), it defines linear component equals to 0.3 and angular = 0 (no rotation):

```
# move forward robot
def move_forward(self):
    # create Twist message
    cmd = Twist()
    cmd.linear.x = 0.3
    cmd.angular.z = 0.0

    # Send command
    self.cmd_vel_pub.publish(cmd)
```

You can use this script as a template for new programs in the future.

First task: Make some figures with the robot!

You should program the movement of robot in order to perform some geometrical figures like circles, triangles, squares etc...

We have prepared for you a script with almost everything you need for this task, you can find the script at: **education_robotics/src/tutorials/1_basic_figures.py**

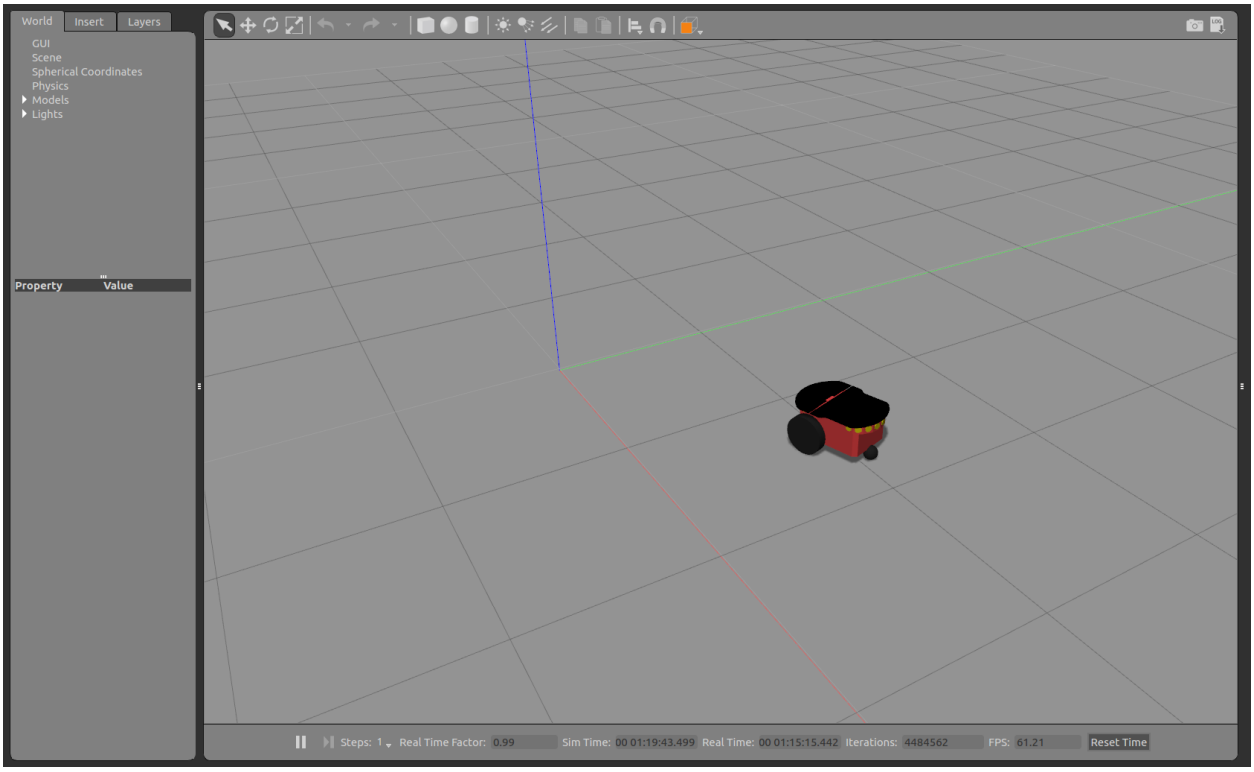
Please open the file an complete the missing parts:

```
# send commands at 10 hz
while not rospy.is_shutdown():
    # ---------------------
    # Place your code here:
    # remember you should create a set of figures with the robot
    # if you have problems, you can use robot_move_example.py as a example.

    # circle:

    # triangle:

    # square:

    # others?:

    # ----------------------
```

Once you have modified the code, you can test it using the following commands:

```
cd ~/catkin_ws
source devel/setup.bash
roslaunch education_robotics tutorials_test_figures.launch
```

- You should have something like this:



Now you are ready for more challenging tasks, lets check the next section!

Navigation tutorials

This section contains some tasks related with robot navigation and strategies for obstacle avoidance.

Reach the goal!

We have prepared for you a maze that includes robot, some goal positions and a closed map.

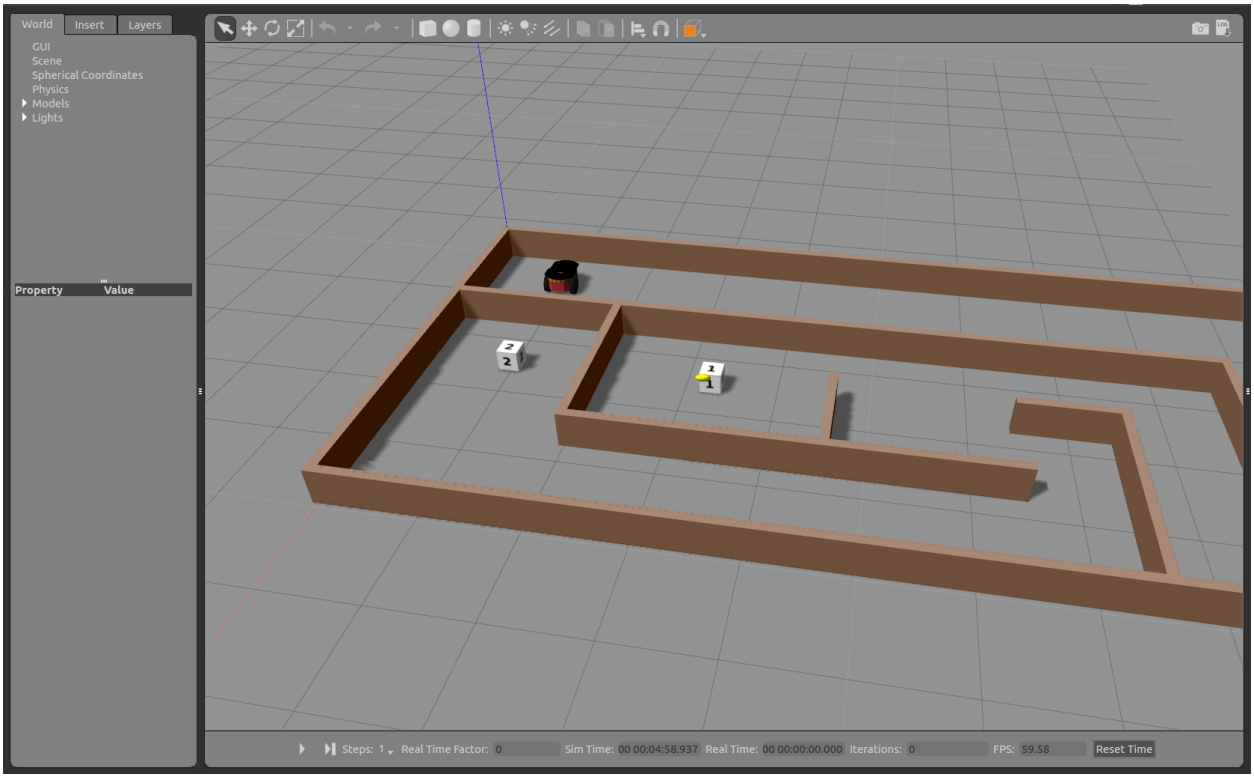
Your task is simple: Move the robot through the maze and reach the goals number 1 and 2 as faster as possible.

You can start the program by running:

```
cd ~/catkin_ws
source devel/setup.bash
roslaunch education_robotics tutorials_navigation.launch
```

In the same way as before, you can edit a template located at: **education_robotics/src/tutorials/2_navigation.py**

- You should get something like this:



Map and odometry: obstacle avoidance

Once you have succeeded with the previous task, you are ready for increasing the complexity of next task:

Now, you should create an algorithm that allows the robot to avoid obstacles and explore the map until reaches goals number 1 and 2.

In order to do that, you can use robot odometry and map for making predictions about coming obstacles.

A simple template is available on: **education_robotics/src/tutorials/3_obstacle_avoidance.py**

This script subscribes for odometry and map information, and allows you to estimate the coming obstacles based on current robot position and fixed map.

Let's have a look at part of script:

- Create a callback for map subscriber, the map is represented as an image and after processing is saved into global variable (self.map)

```
# Update map
def callback_map(self, data):
    try:
        temp_map = self.bridge.imgmsg_to_cv2(data, "bgr8")

        # convert into gray scale
        (rows, cols, channels) = temp_map.shape
        if cols > 0 and rows > 0 :
            self.map = cv2.cvtColor(temp_map,cv2.COLOR_RGB2GRAY)

    except CvBridgeError as e:
        print(e)
```

- Subscribe for odometry information and save it into global variables.

```
# update position and orientation of robot (odometry)
def process_odometry_message(self, odometry_msg):
    self.robot_position_x = odometry_msg.pose.pose.position.x
    self.robot_position_y = odometry_msg.pose.pose.position.y
    self.robot_orientation = odometry_msg.pose.pose.orientation.y

    quaternion = (
        odometry_msg.pose.pose.orientation.x,
        odometry_msg.pose.pose.orientation.y,
        odometry_msg.pose.pose.orientation.z,
        odometry_msg.pose.pose.orientation.w)

    (self.robot_roll, self.robot_pitch, self.robot_yaw) = euler_from_
↪quaternion(quaternion)
```

- Take into account that odometry is given in meters and the map is given in pixels, the relationship between them is:

1 meter = 100 pixels

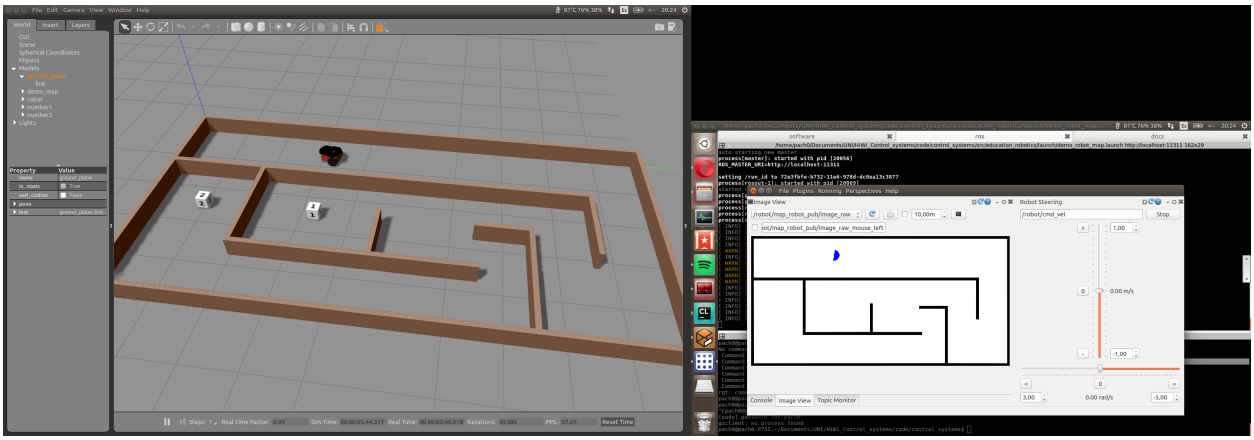
- This part does a basic obstacle detection, it considers a "safety boundary region" around the robot and a loop detect any obstacle inside this area.

```
# define a safety boundary region around robot
boundary = 40
detected_obstacle = False

# loop inside boundary, check if there are obstacles
for y in range( int((self.robot_position_y*100) - boundary), int((self.robot_position_
↪y*100) + boundary) ):
    for x in range( int((self.robot_position_x*100) - boundary), int((self.robot_
↪position_x*100) + boundary) ):
        map_point = self.map[x, y]
        if map_point < 1:
            # Obstacle detected!
            detected_obstacle = True

            # Here design your own routine for obstacle avoidance and navigation!

            # for logging: visualize using console plugin in rqt.
            # rospy.loginfo("close to obstacle y %f  x %f p %f", y , x, map_point)

            break
```

- Once you have modified the source code (python script), you can test it by running:

```
cd ~/catkin_ws
source devel/setup.bash
roslaunch education_robotics tutorials_obstacle_avoidance.launch
```

- You should have something like this: Gazebo simulator with robot and map + RQT interface with tools.



Frequent questions

This section explain some common problems and how to solve it!

Installation and compilation problems

- If you want to change default GCC compiler to version 5, open a terminal and run:

```
export CC=/usr/bin/gcc-5
export CXX=/usr/bin/g++-5
```

then you can compile using GCC 5.

ROS related problems

under development...

Interface problems

- When I close a node that uses gazebo, it takes too long before it closes.

Answer: You can close manually the process, run in a separate command window:

```
sudo killall gzserver gzclient
```

Contact us

- If you want more info about the project or you want to contribute, we are happy to contact you!

Francisco J. Garcia R. - garciar@rhrk.uni-kl.de

13. Sc. Alen Turnwald - turnwald@eit.uni-kl.de

- education robotics license:

Developed by:

- Francisco J. Garcia R. - garciar@rhrk.uni-kl.de
- 13. Sc. Alen Turnwald - turnwald@eit.uni-kl.de

Institute for control systems

University of Kaiserslautern 2016