**BLG456E, Robotics, Fall 2014**

# Assignment #1: Obstacle avoidance

Lect. Dr. Damien Jade Duff
Res. Asst. Çağatay Koç

**Due:** Oct 10 (Friday) 11:59pm

**Assignment summary:** Program a ROS node to control a Turtlebot to move and avoid obstacles. The Turtlebot will be simulated in Gazebo.

**Submission type:** An archive file of the source code. The archive will contain the following files (replacing **{student ID}** with your ID):

```
a1_{student ID}/src/a1_{student ID}_node.cpp
a1_{student ID}/CMakeLists.txt
a1_{student ID}/package.xml
```

It should be possible to compile the package by placing the files into a catkin workspace and running the catkin compilation.

**Assignment: In this assignment, you will design a ROS node which is able to move the robot platform freely in 2D in an unknown environment. The robot should move around autonomously while avoiding obstacles using it's laser scanner. You can use the `/scan` topic to read laser scans and the `/cmd_vel_mux/input/navi` topic to send movement messages to the robot platform. You should create your own package which is named `a1_{student ID}` where {student ID} is replaced by your own student ID. For this assignment, a report is not necessary, but your code needs to be well commented and clear.**

## Step-by-step

### ROS & Turtlebot Setup

Before starting, make sure you have the necessary tools. For this assignment you will need the Ubuntu 14.04 operating system, ROS Indigo and the Turtlebot Gazebo package package. For instructions on how to get these, see the document "How to install ROS Indigo and the Turtlebot Simulation" under course files on the Ninova courseware website.

Once this this is completed, restart your terminal, because the scripts provided will add a line to your startup files to set up your ROS environment, including the Turtlebot packages.

For more information about what the provided scripts did for you, you could read:

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

http://wiki.ros.org/turtlebot/Tutorials/indigo/Installation

## Basic knowledge

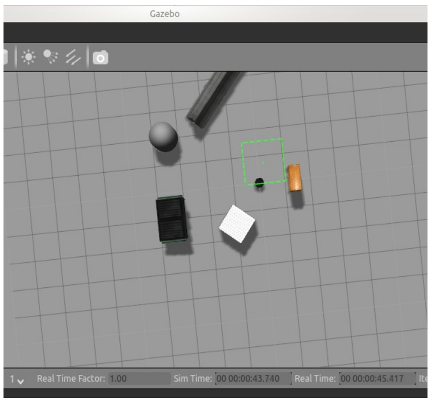
It is advised that you go over the beginner level tutorials in the [ROS web site](). In particular, tutorials 1, 2, 3, 4, 5, 6, 11 and 13.

## Run the simulator

The robot controller that you write will send messages to and receive messages from the simulated Turtlebot robot. To start the simulation, run the following command in a terminal:

```
roslaunch turtlebot_gazebo turtlebot_playground.launch
```

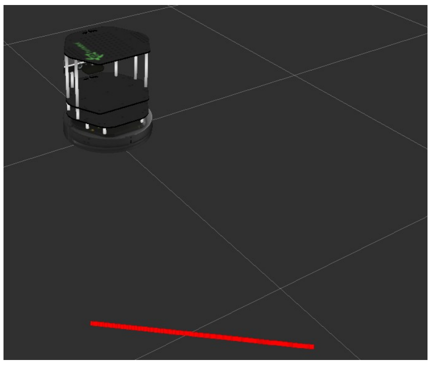
You should see the Turtlebot robot in the Gazebo GUI amongst a scattering of objects:



In order to see what your robot sees, in a *new terminal* window run

```
roslaunch turtlebot_rviz_launchers view_robot.launch
```

This allows you to visualise the contents of the robots sensors and its own self-model. This same visualiser would be used even if it were a real and not simulated robot that you were using. If you turn on the LaserScan display you will also see the result of the robot's laser scanning as a line of red markers:



In order to drive the robot around in its simulated world, you can run the following command and enter directions from the keyboard according to the displayed instructions:

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

Only keyboard commands that you enter into that terminal will be registered. It is normal to place the rviz window, the simulator window, the teleoperation terminal, and other terminals alongside each other on your computer display.

## Examine the ROS environment

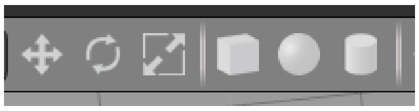
To get a deeper understanding of what is happening when the simulation and rviz is running, read:

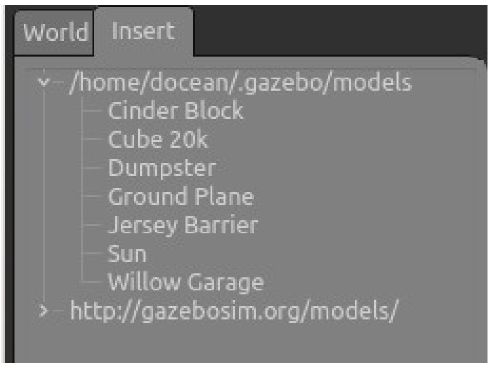
> http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes
>
> http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

Try running the `rqt_graph`, `rostopic list`, `rostopic info`, `rostopic echo`, `rosnode list`, `rosnode info`, etc. commands while the simulation and/or rviz is running to get an idea for what is going on behind the scenes.

## Editing the simulated world

In order to produce more exciting scenarios for your robot, you can use the simple object insertion and manipulation toolbar  or the "Insert" panel to insert new objects in Gazebo:





/home/docean/.gazebo/models
- Cinder Block
- Cube 20k
- Dumpster
- Ground Plane
- Jersey Barrier
- Sun
- Willow Garage

http://gazebosim.org/models/

In order to save and re-use these new scenarios, however, you will have to create your own "launch" file, which is beyond the scope of these instructions.

## Buggy Gazebo

If Gazebo crashes, it may not have fully crashed. It may just be the GUI that crashed. You usually can get the GUI back by running, in yet another terminal window:

```
rosrun gazebo_ros gzclient
```

## Set up your ROS catkin workspace

Read:

> http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment

In particular, follow the instructions in par t 4 of that tutorial. While you should read part 3, be aware that the Turtlebot installation scripts described above have already done this part for you, so you are not advised to follow the instructions in part 3.

## Create your package

Among the tutorials on the ROS web site are tutorials on creating and building ROS packages (tutorials 3 & 4):

http://wiki.ros.org/ROS/Tutorials/CreatingPackage

http://wiki.ros.org/ROS/Tutorials/BuildingPackages

Note that tutorials offer a choice between the older **rosbuild** and the newer **catkin**. In this course we will create catkin packages.

**Creation**: The package that you create for your assignment will be called `a1_{student ID}` where {student ID} is replaced by your student ID. It will depend only on the packages `roscpp` and `std_msgs`. You can create the package with dependencies using the `catkin_create_pkg` command.

**Building**: Note that you do not need to run the command beginning with "`source`" mentioned at the start of the building tutorial, as this has been taken care of you by the Turtlebot installation script – in fact, running that command would make the Turtlebot packages unavailable until you restart the terminal.

## Create the node

Follow the tutorial on writing a publisher and subscriber in the ROS tutorials:

http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

http://wiki.ros.org/ROS/Tutorials/ExaminingPublisherSubscriber

The differences with this assignment from that tutorial are that:

- Your one node will be both a subscriber and publisher as it will subscribe to laser scan messages and publish movement messages.

- Your node will be in a file called **`src/a1_{student ID}_node.cpp`** inside the package you create. You will need to uncomment out the line in `CMakeLists.txt` that asks for the node executable to be built and the line that links the node executable to the core ROS libraries:

  ```
  add_executable(a1_{student ID}_node src/a1_{student ID}_node.cpp)
  target_link_libraries(a1_{student ID}_node ${catkin_LIBRARIES})
  ```

- Your node will subscribe to laser scans from the `/scan` topic.

  These messages will be of type sensor_msgs/LaserScan.

  Within the C++ code, the type for the laser scans in the callback will be:

  ```
  const sensor_msgs::LaserScan::ConstPtr&
  ```

  More information about this type could be found by examining the header files.

- Your node will publish to twists on the `/cmd_vel_mux/input/navi` topic.

  These messages will be of type geometry_msgs/Twist.

  Within the C++ code, the type for the twists to be published will be:

  ```
  geometry_msgs::Twist
  ```

  More information about this type could be found by examining the header files.

After you have finished creating the node, try compiling it with catkin_make as described in http://wiki.ros.org/ROS/Tutorials/BuildingPackages. If the current directory is your catkin workspace, the node executable file should have been built according to the path `../devel/lib/a1_{student ID}/a1_{student ID}_node`. You can run it by typing its path.

## How to use laser scan data

Laser scan data will be available within the callback function that your program registers when it subscribes to the `/scan` topic by creating a `ros::Subscriber` object. As the datatype provided to the callback is usually in the form of a pointer, you will need **`->`** to access it.

The data structures available withn the message object are described in the message documentation:

http://docs.ros.org/indigo/api/sensor_msgs/html/msg/LaserScan.html

In particular, the `float32[]` ranges datatype is available as a normal `std::vector` inside your program and consists of a series of numbers representing how far away each part of the scan found an object. You can find the size and contents through normal debugging methods.

## How to send movement messages

You can publish a message through a `ros::Publisher` object. The message that you will be publishing is a “twist”, which is a combination of linear and angular velocity, but for three dimensional objects. As we are only concerned with two dimensions, you will only need to set the **x** and **y** compoments of the linear velocity and the **z** component of the angular velocity.

As with the laser scan, you can access the fields of the `geometry_msgs/Twist` message, which will be “linear” and “angular”, which themselves are messages of type `geometry_msgs/Vector3`. To see what fields are available in a Vector3 message, see the message definition:

http://docs.ros.org/api/geometry_msgs/html/msg/Vector3.html

## Marking criteria

- Creation of a package.
- Creation of a node.
- Reading laser scans.
- Processing laser scans.
- Publishing movements.
- Moving the robot.
- Moving the robot intelligently.
- Successful object avoidance.
- Clear code.
- Lack of errors.

**Bonuses avaiable for**: Interesting robot behaviour (e.g. wall-following), brief intelligent analysis (readme.txt) of problem, etc.

*Assignments not submitted according to requirements will not be evaluated.*