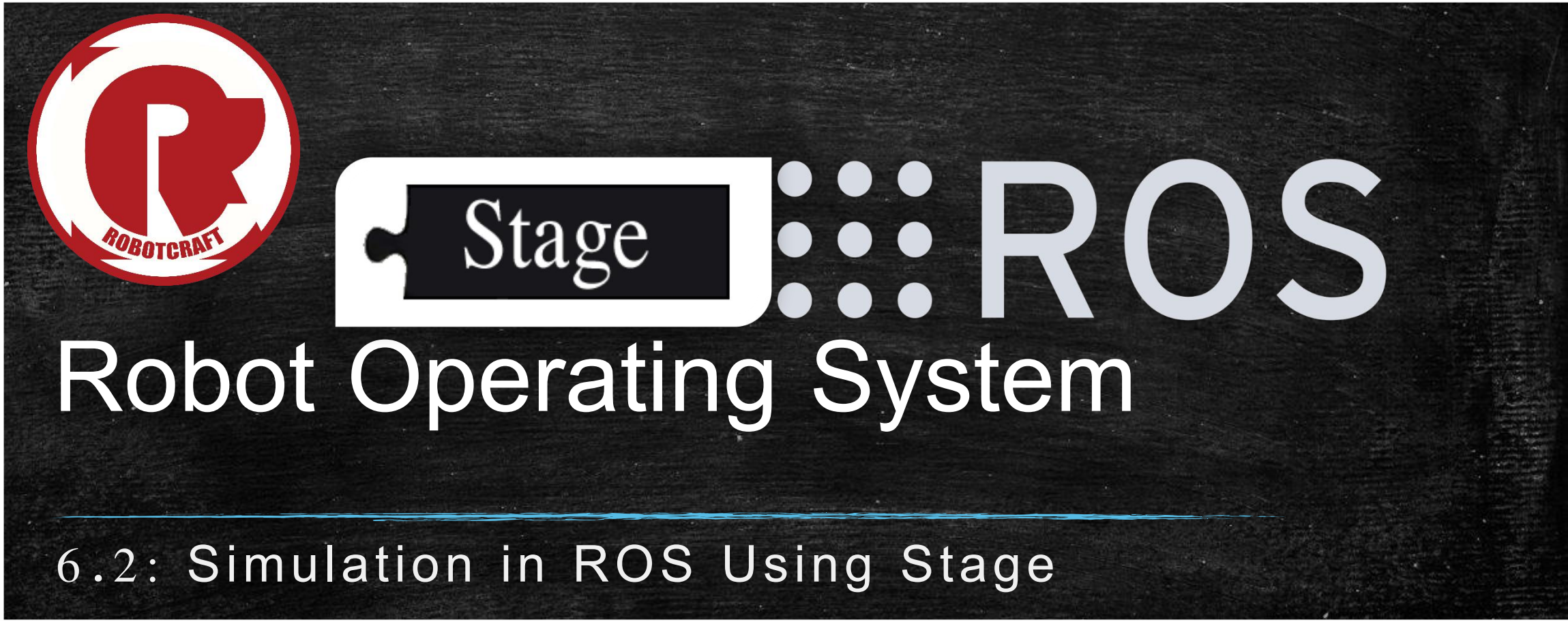
ROBOTCRAFT©

ROSin

INGENIARIUS



Craft #6

João Aguizo
joao@ingeniarius.pt

08/08/2022

Objectives and Layout (ROS Day 3)

- Last week we have:
  - Gone through the basic ROS topics (which we will recap again);
  - Implemented a very simple controller for turtlesim;
  - Got to grips with a few possible variations on node structures, topics, services, etc.
- Today we will:
  - Introduce the basic ideas behind ROS simulation;
  - Introduce one of the basic ROS simulators, stage;
- Today's schedule may be more constrained than last week, as we have a lot of material to cover. We won't make as much use of checkpoints.

Recap: Basic ROS

I hope you remember this

Why ROS ?

- What are robots?
  - Hardware + Software
- We want to have the same “intelligence” in different robots!
- We need a unified software suite that abstracts away the particularities of each robot.

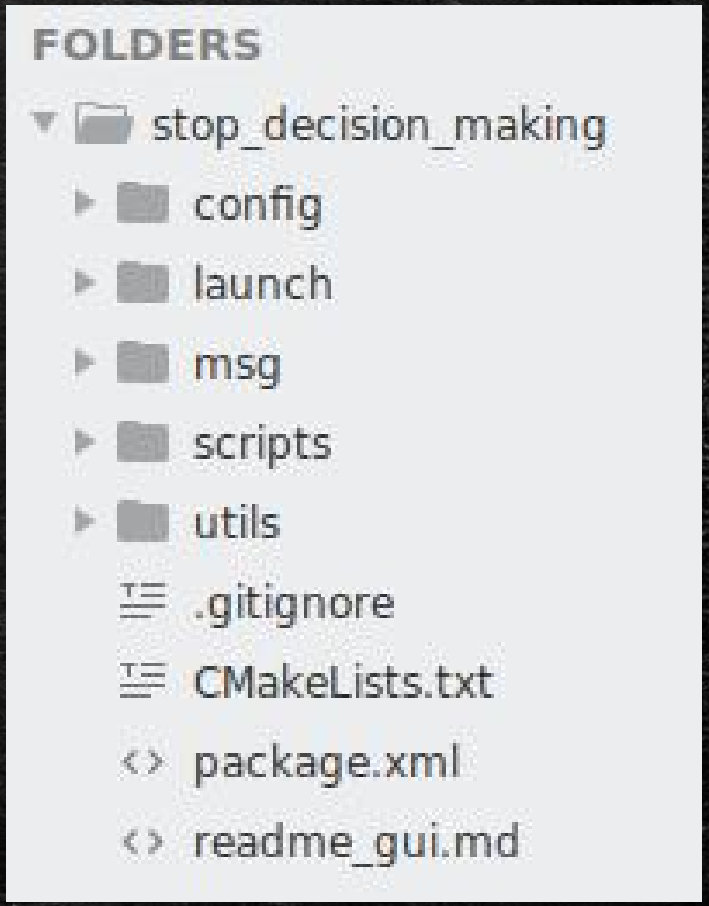
ROS

What is ROS ?

- ROS is a suite of formalisms, tools and libraries (a **framework**) for implementing robot-related software;
- The ROS conventions allow us to write robot-abstract software that can be easily ported from platform to platform;
- By greatly expediting development, it became very, very popular.

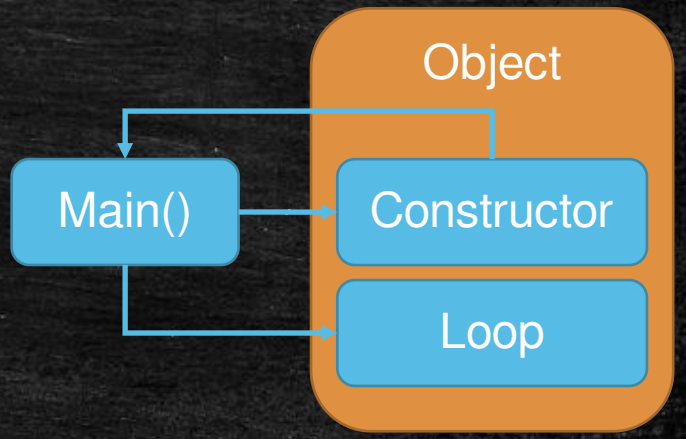
What are packages ? Why do we care ?

- ROS packages are folder structures that keep all your code neat, tidy, portable and distributable;
- Different sub-folders are created as you generate content;



Basic Structure of a ROS Node

- Now that we have something that builds against the ROS libraries, we should write a first version of our node.
- The structure of a ROS node is very up to the programmer, but I like to structure it as follows:
  - Encapsulate all functionality in a class (this allows us to elegantly maintain state, and maybe re-use our module in the future if necessary);
  - Use the constructor (or an initialization function) to create all necessary topics, etc...
  - Use a run( ) function to run the node's loop.



Topics: (kind of) Like WhatsApp Groups

- Any number of publishers can publish into the topic, and any number of subscribers can subscribe;
- **Any publisher can publish messages at any time**;
- **All subscribers will receive the messages**;
- Each topic has a pre-defined **message type**.

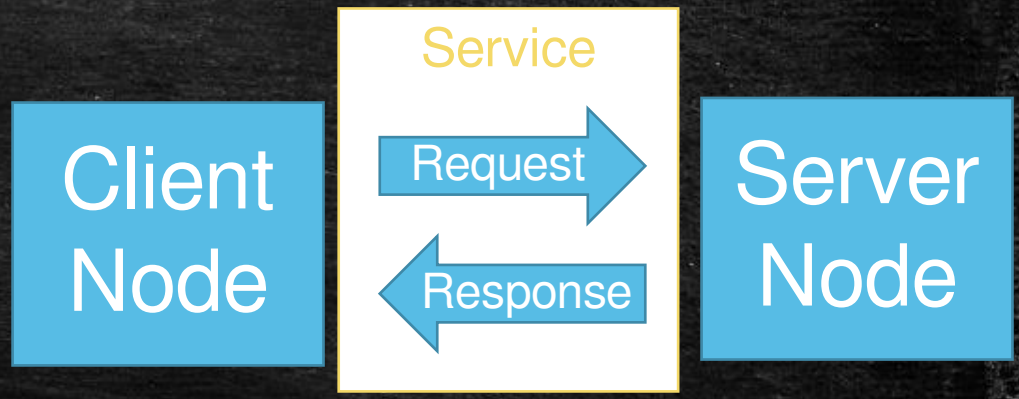
Launch Files: What are they?

```
<launch>

    <node pkg="turtlesim" type="turtlesim_node" name="turtlesim" output="screen"/>

    <!-- TODO: launch our node and remap the cmd_vel topic -->

</launch>
```

- Launch files specify a set of nodes that will run at the same time.
- Launch files can be immensely useful as your system grows.
- Each launch file can specify a number of nodes that run, and can also import other launch files.

Services: Introducing Synchronicity

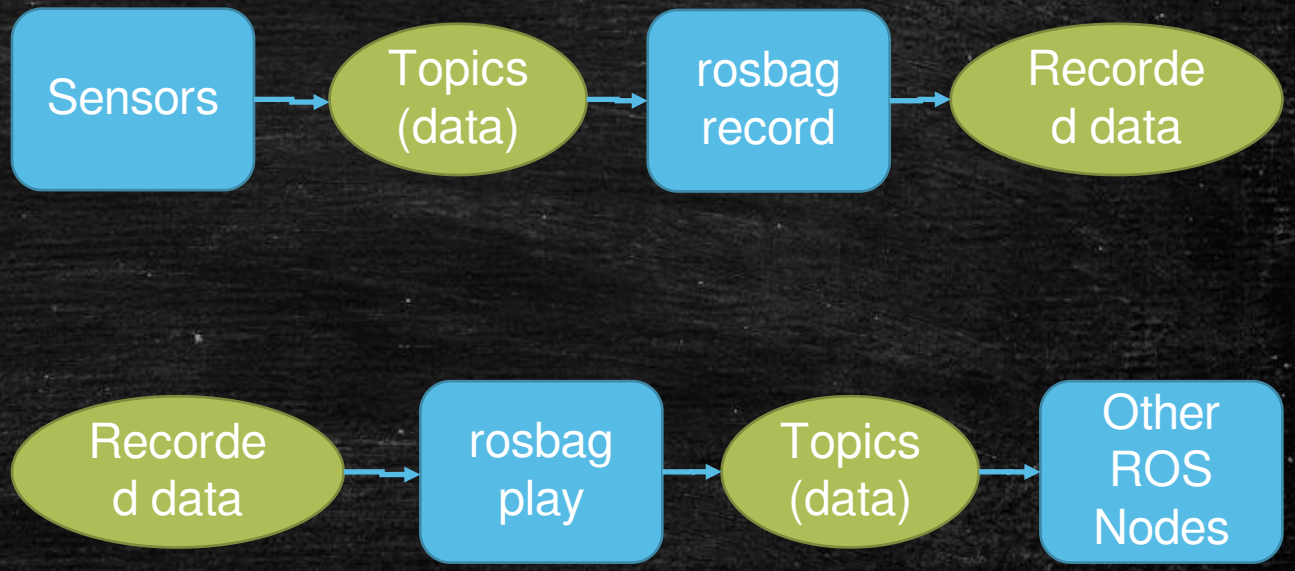
- Services allow for functionality of a different node to be called synchronously.
  - What does synchronously mean?
- Client sends a request and waits for a response.
- Can be used for (so many things):
  - Decouple nodes (dependencies, etc);
  - Offload processing power;
  - Implement synchronous client-server communication protocols.
- Can be defined with essentially the same format as messages with one key difference: a request and a response need to be defined.




```
1 # Goal: a pose where we want to be
2 geometry_msgs/PoseStamped target_pose
3 ---
4 # Result is not needed since the
5 # service response will tell us
6 # if we were successful.
```

What are bags ?

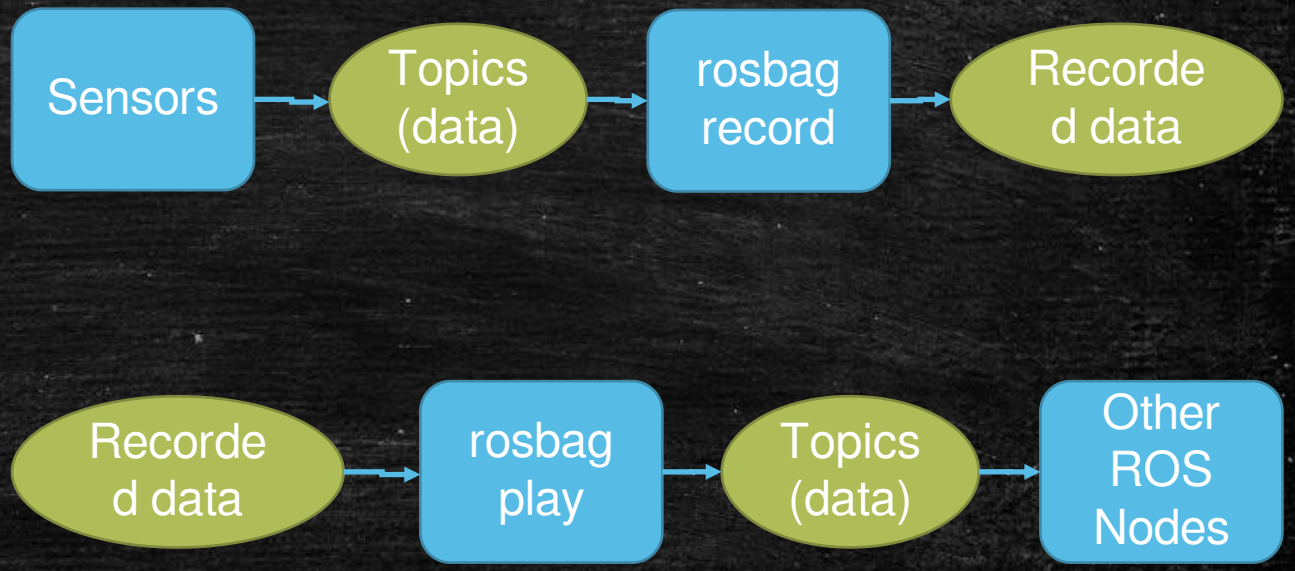
- Most information in ROS flows in topics;
- Bags allow us to record this information in a timestamped manner and play it back at will;
- Bags are extremely useful:
  - To record and play back data from an expensive experiment;
  - To allow us to develop a module on top of well-known data;
  - …



ROS Simulation Basics

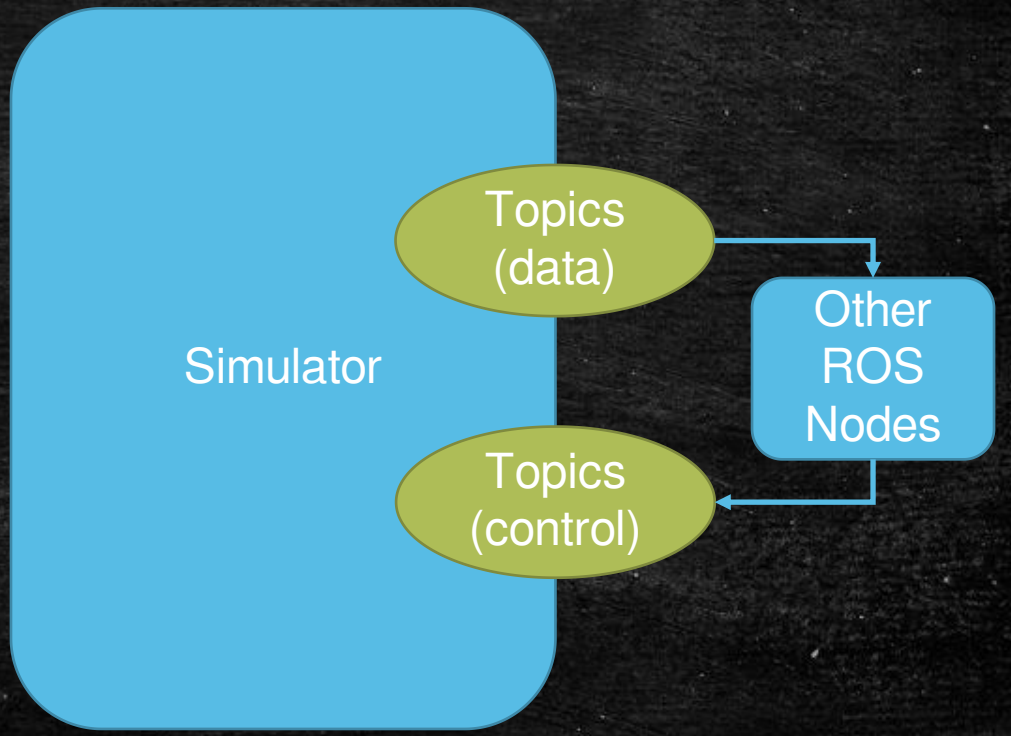
Why simulation ?

- Think about rosbags: they allow us to save resources by re-playing expensive experiments.
- What if we need to change the conditions of the experiment?
  - Control the robot differently;
  - Test new algorithms;
  - …



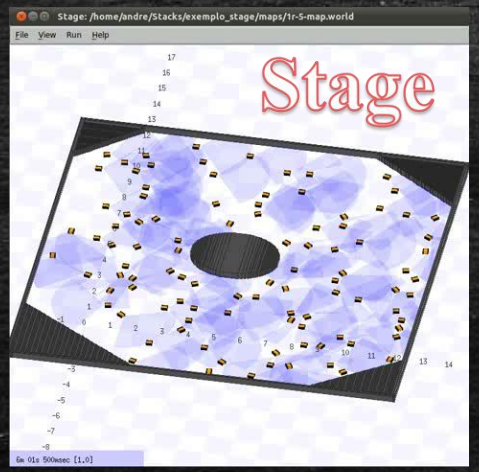
Why simulation ?

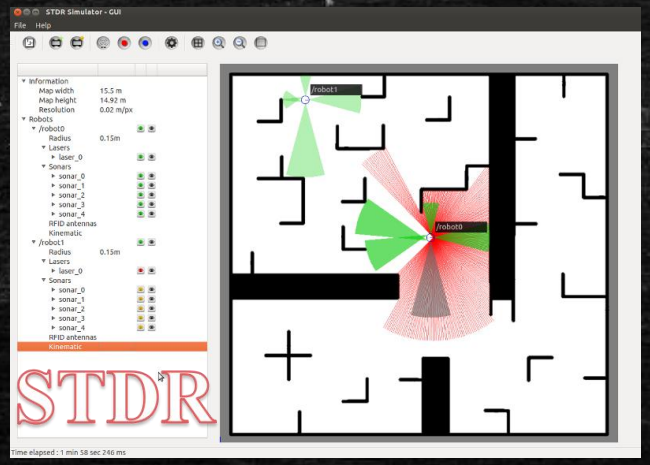
- ROS simulators seamlessly replace the robot and the environment;
- Thanks to ROS decoupling, **exactly the same code can be used in simulated and real robots;**
- ... and simulators are a lot cheaper and convenient than real robots.

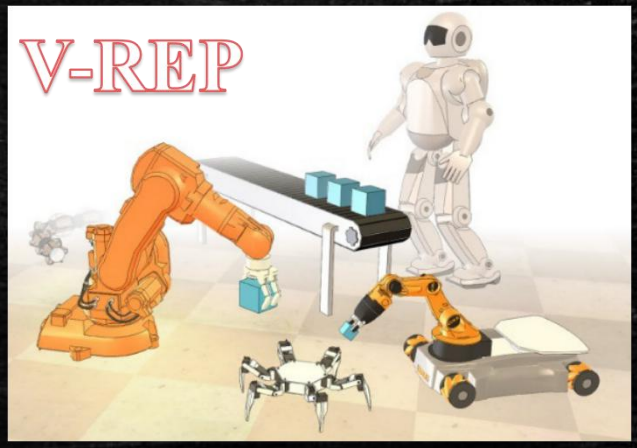


What simulators exist ?

- Many.

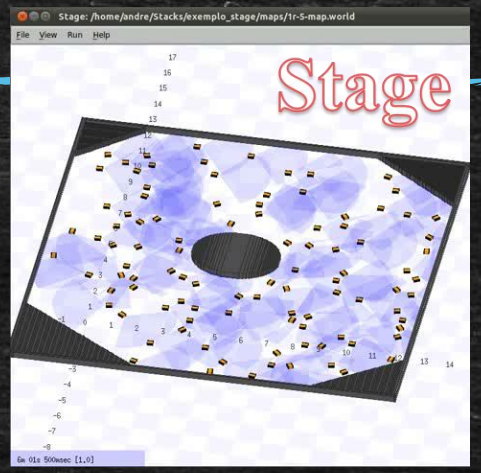


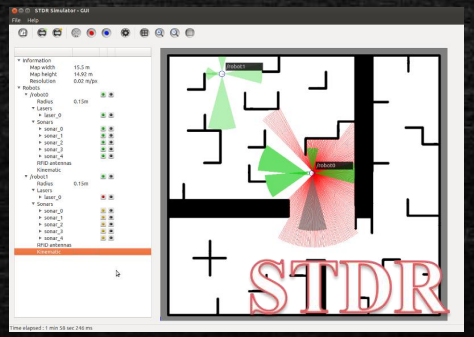


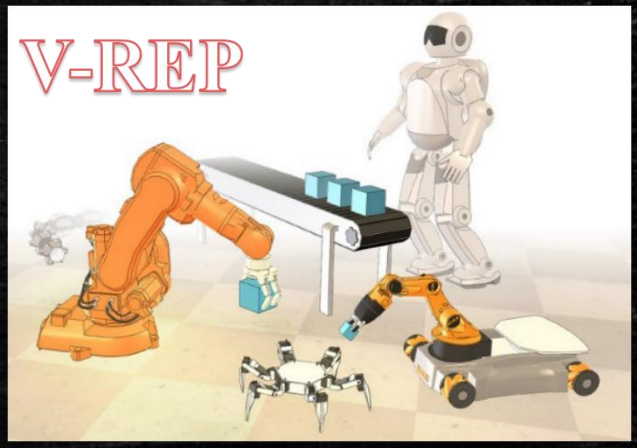


What's with all the simulators?

- Different simulators serve different purposes:
  - Some are more accurate;
  - Others fit a certain application better;
  - Others are simpler to work with;
- Will I be doomed by learning only one of them?
  - No, the basics are the same for all of them.
- We will use Stage!
  - Why?



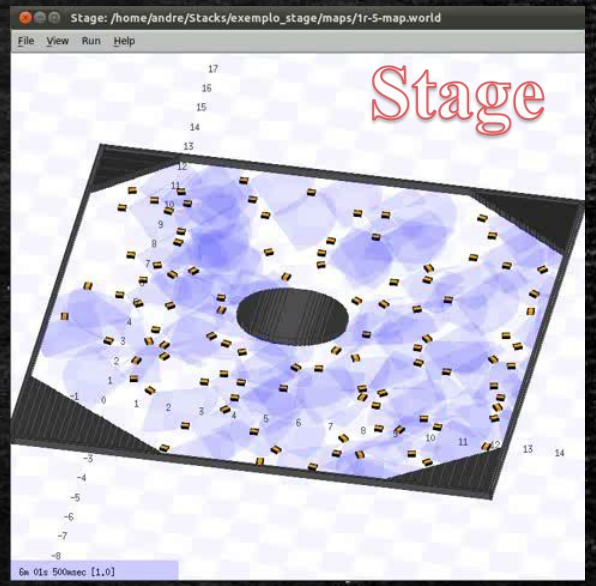




The Stage Simulator

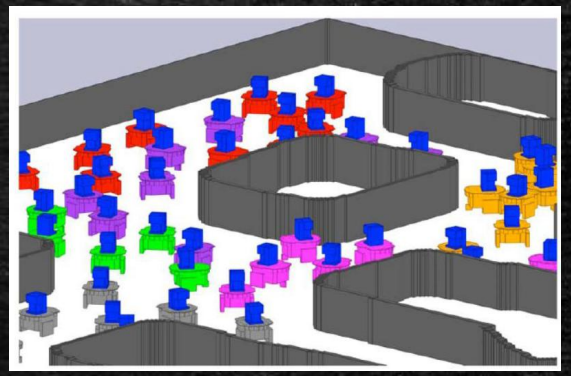
Stage Simulator

- Stage simulates a population of mobile robots, sensors and objects in 2D environments.
- 2D dynamic physics simulator with some 3D extensions: 2.5D simulator.
- OpenGL Graphical Interface.
- Leverages graphics processor (GPU) hardware -- being fast, ease to use, and having wide availability.



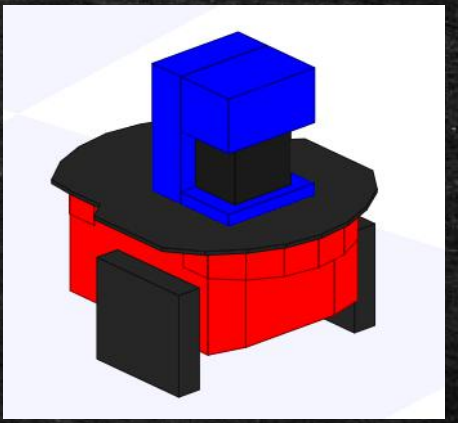
Stage Simulator

- Stage is free and open source, and it runs on Linux and other Unix-like platforms.
- It allows rapid prototyping of controllers destined for real robots.
- Originally developed by the Robotics Research Lab at
  the University of Southern California in L. A.
- It was the Simulation backend for the Player/Stage system.

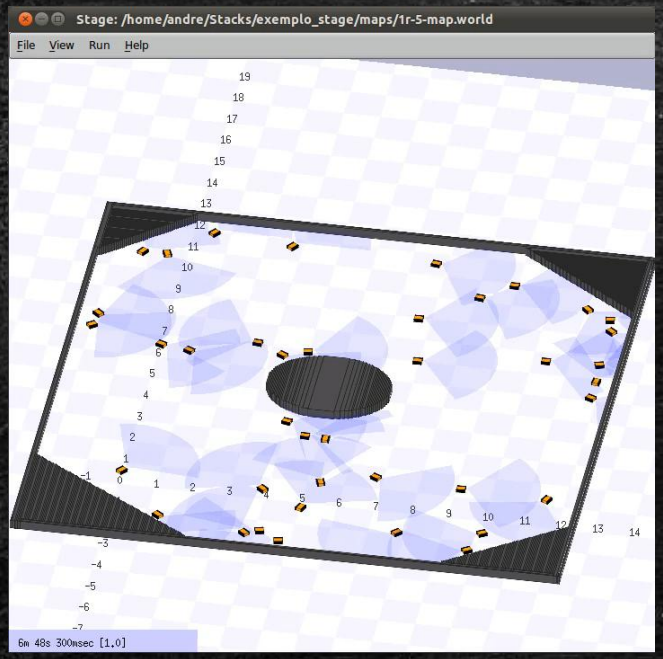


Stage Simulator

- Stage “provides fairly simple, computationally cheap models of lots of devices rather than attempting to emulate any device with great fidelity. This design is intended to be useful compromise between conventional high-fidelity robot simulations, minimal simulations, and the grid-world simulations common in artificial life research”.
- “We intend Stage to be just realistic enough to enable
users to move controllers between Stage robots and
real robots, while still being fast enough to simulate large
populations.”



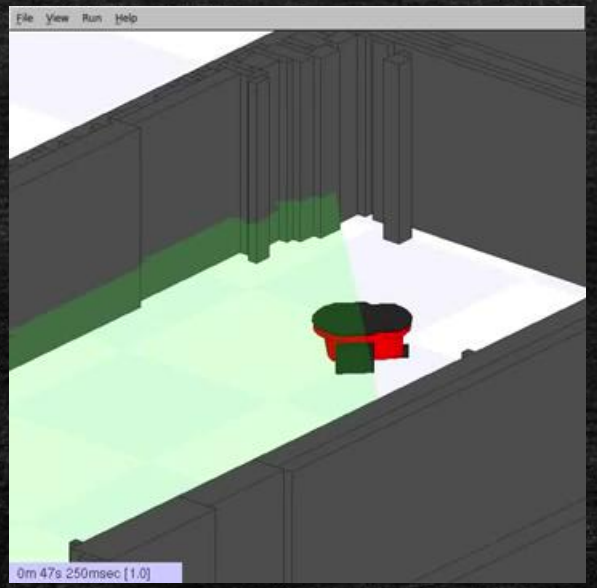
Stage Simulator: Applications



Swarm Robotics & Multi-Robot Tasks (e.g. coverage, patrolling, formation control, exploration, mapping, etc.)

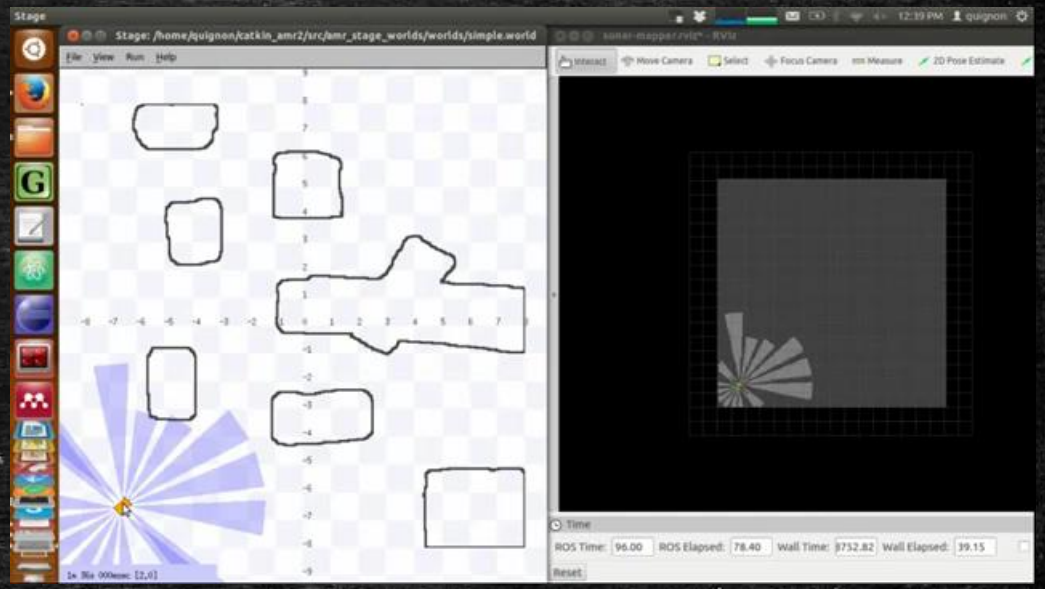
Video Link

Stage Simulator: Applications



Autonomous Exploration

Video Link



Sonar Mapping

Video Link

Stage + ROS

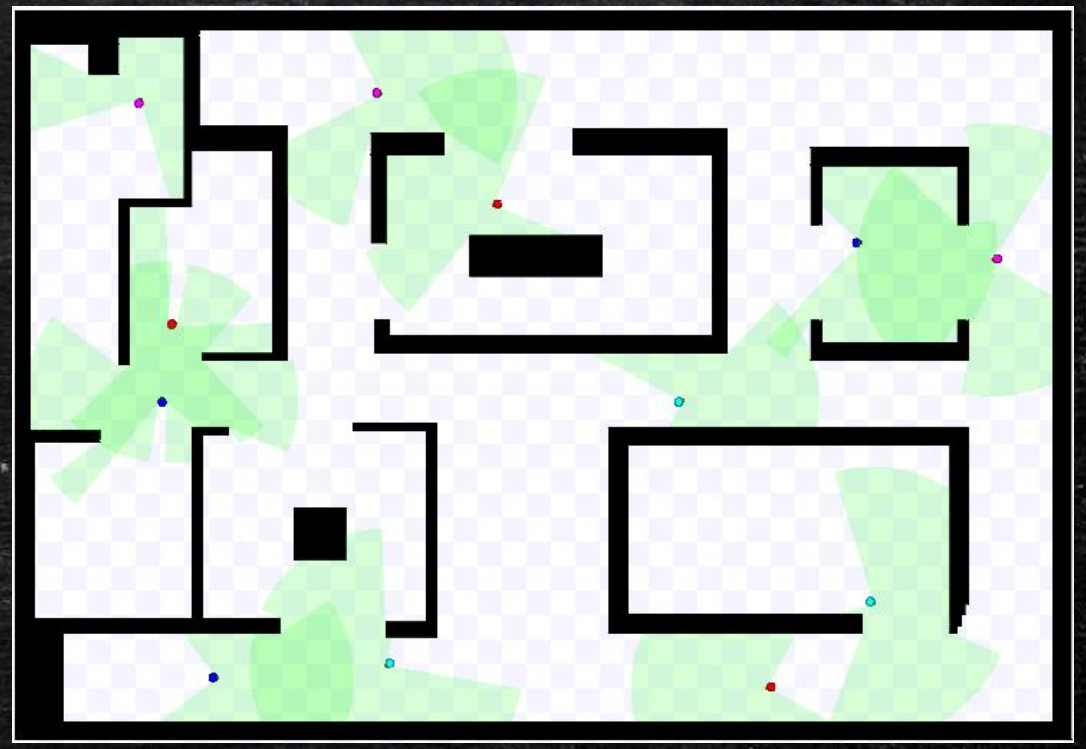
- Stage is a standalone robot simulation program. How can we run it on ROS?
- Stage is made available for ROS, through the **stageros** node from the stage_ros package, which wraps some functionalities of Stage via “libstage”.
- Using standard ROS topics, stageros provides odometry data from each virtual robot and scans from the corresponding laser model. Additionally, a ROS node may interact with Stage by sending (i.e. publishing) velocity commands to drive the virtual robot.

Stage + ROS: the *stage_ros* package

http://wiki.ros.org/stage_ros

Stage + ROS: Simulating the “World”

- Stage simulates a world as defined in a “.world” file.
- The world file is simply a description of the world to be simulated. It includes robots, sensors, actuators, moveable and immovable objects.



Stage + ROS: Simulating the “World”

- Example of a world file.
  - https://github.com/ros-planning/navigation_tutorials/blob/indigo-devel/navigation_stage/stage_config/worlds/willow-pr2.world
- The world file can also be used to control many aspects of the simulation engine, such as its speed and fidelity.

```
define block model
(
  size [0.5 0.5 0.5]
  gui_nose 0
)

define topurg ranger
(
  sensor(
    range_max 30.0
    fov 270.25
    samples 1081
  )
  # generic model properties
  color "black"
  size [ 0.05 0.05 0.1 ]
)

define pr2 position
(
  size [0.65 0.65 0.25]
  origin [-0.05 0 0 0]
  gui_nose 1
  drive "omni"
  topurg(pose [ 0.275 0.000 0 0.000 ])
)

define floorplan model
(
  # sombre, sensible, artistic
  color "gray30"

  # most maps will need a bounding box
  boundary 1

  gui_nose 0
  gui_grid 0

  gui_outline 0
  gripper_return 0
  fiducial_return 0
  ranger_return 1
)

# set the resolution of the underlying raytrace model in meters
resolution 0.02

interval_sim 100  # simulation timestep in milliseconds


window
(
  size [ 745.000 448.000 ]

  rotate [ 0.000 0.000 ]
  scale 28.806
)

# load an environment bitmap
floorplan
(
  name "willow"
  bitmap "../maps/willow-full.pgm"
  size [58.4 52.6 0.5]
  pose [ -26.300 29.200 0 90.000 ]
)

# throw in a robot
pr2( pose [ -26.068 12.140 0 87.363 ] name "pr2" color "blue")
block( pose [ -25.251 10.586 0 180.000 ] color "red")
```

Let’s Get Started!

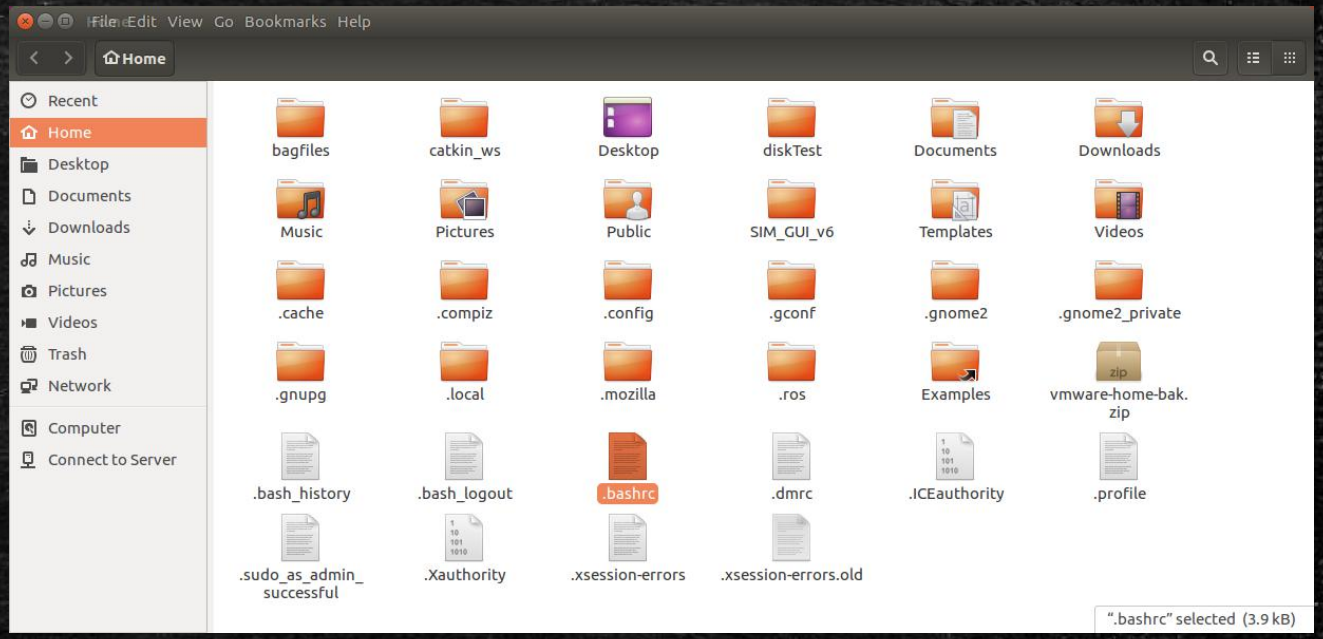
Step 1. Check ROS workspace

RobotCraft 2022: Stage/ Tutorial ROS

Step 1. Making sure you have a properly configured ROS workspace:

$ **gedit ~/.bashrc**

Open in gedit* your “.bashrc” configuration file, which is located at your home folder.



*you can use other editors (e.g. “kate”) instead of gedit, if you prefer.

RobotCraft 2022: Stage/ Tutorial ROS

Step 1. Making sure you have a properly configured ROS workspace:

You should have the following lines at the end of your ".bashrc" file:

source /opt/ros/melodic/setup.bash ← Sourcing the ROS installation

source ~/catkin_ws/devel/setup.bash ← Sourcing the ROS workspace

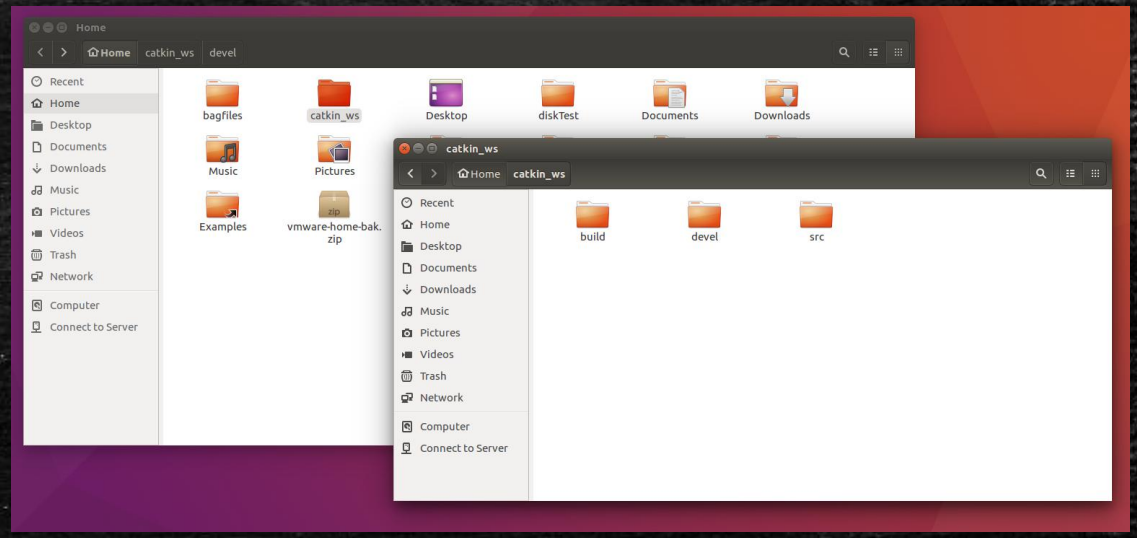
Please add them, in case you don't have them, and save the file.

RobotCraft 2022: Stage/ Tutorial ROS

Step 1. Making sure you have a properly configured workspace:

In case you don't have a working workspace already at **/home/($USER)/catkin_ws**, please do:

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/
$ catkin_make
```



Step 1. Making sure you have a properly configured workspace:

Let's confirm if everything is OK now. Open a **new terminal**, and type:

$ echo $ROS_PACKAGE_PATH

The environment variable which tell ROS where to look for packages

It should return:

/home/($USER)/catkin_ws/src:/opt/ros/melodic/share

Step 1. Making sure you have a properly configured workspace

Complete

Step 2. Check Stage

Step 2. Making sure Stage is up and running

If you installed the “Desktop-Full” version of ROS, you should already have stage_ros in your PC. You can check this by running:

$ **roscore**

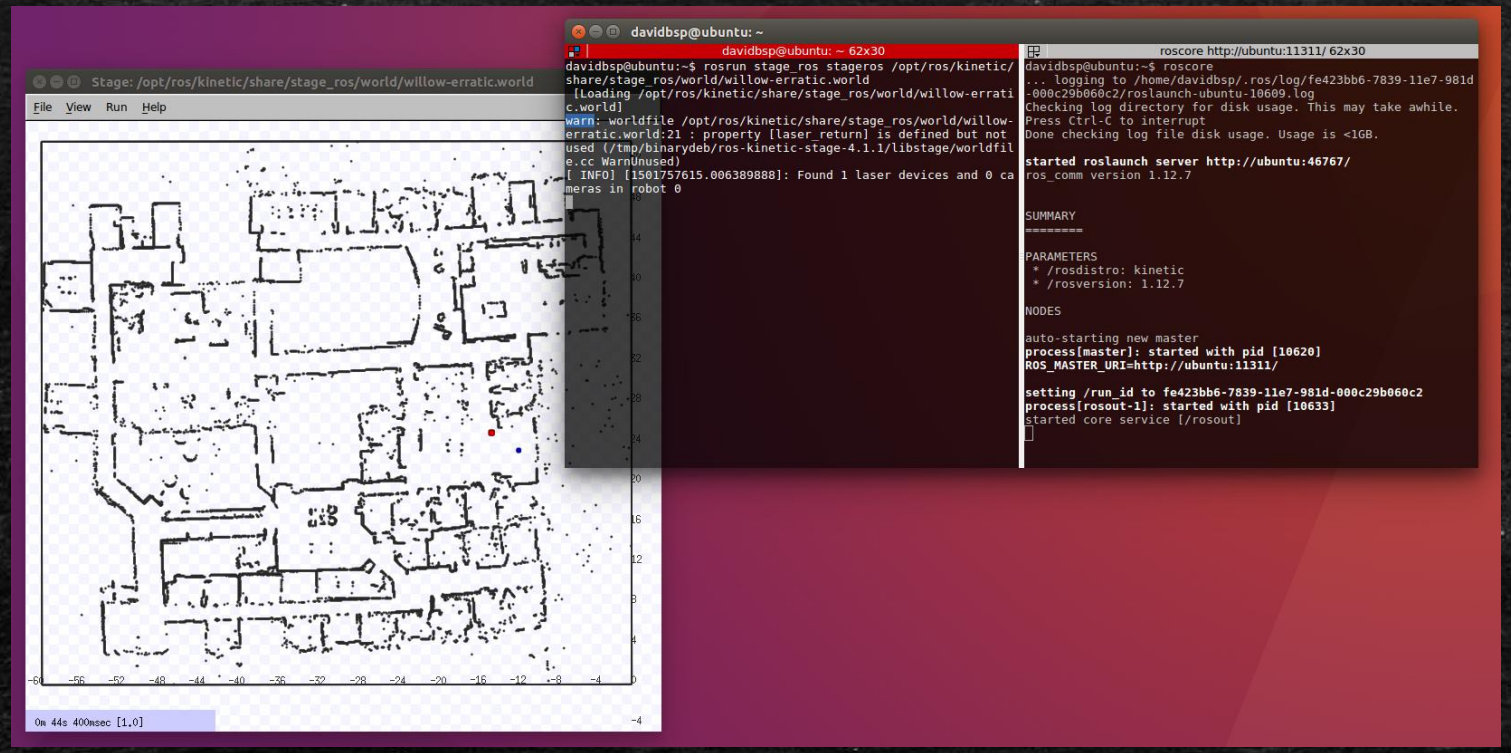
$ **rosrun stage_ros stageros /opt/ros/melodic/share/stage_ros/world/willow-erratic.world**

This will open ***stage_ros*** with a predefined world (see next slide).

RobotCraft 2022: Stage/ Tutorial

ROS

Step 2. Making sure Stage is up and running



Step 2. Making sure Stage is up and running

In case stage is not installed, you can install it by typing:

$ **sudo apt install ros-melodic-stage ros-melodic-stage-ros**

Now you can repeat the steps to open the stage simulator, and it should work.

Step 2. Making sure Stage is up and running

Complete

Step 3. ROS Package

Step 3. Creating a ROS Package for our Stage/ROS simulations

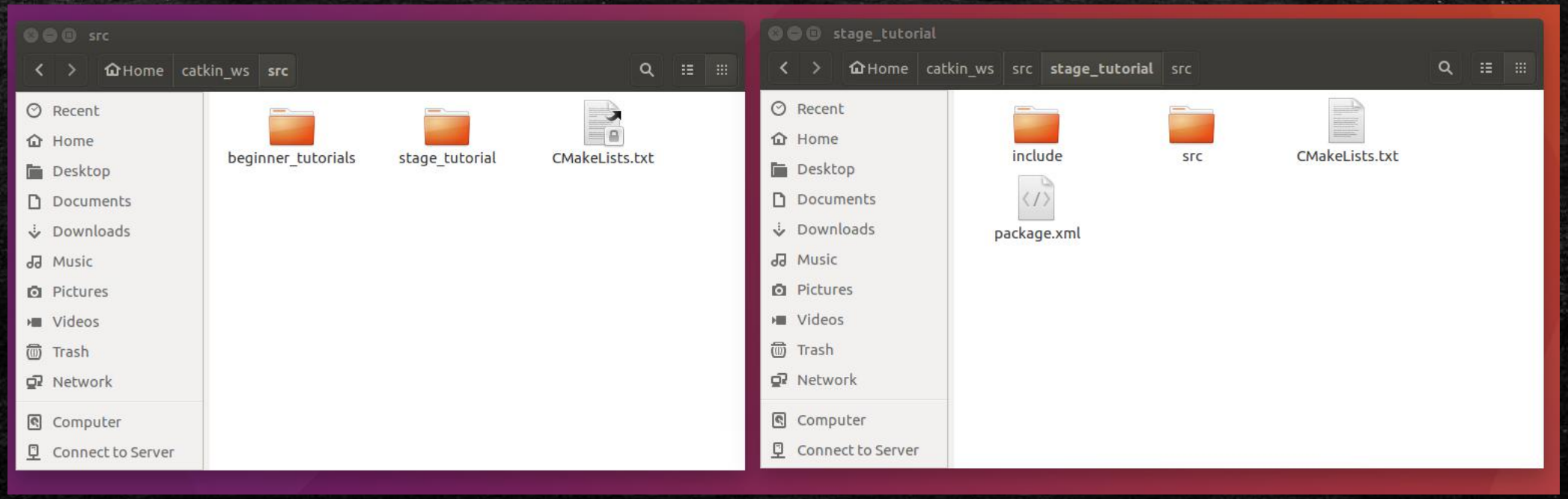
```
$ roscd
$ cd ..
$ cd src
$ catkin_create_pkg stage_tutorial roscpp rospy std_msgs
sensor_msgs geometry_msgs nav_msgs tf stage_ros
```

This will create a "**stage_tutorial**" package with dependencies on 8 other ROS packages.

RobotCraft 2022: Stage/ROS Tutorial

Step 3. Creating a ROS Package for our Stage/ROS simulations

Verifying that the "**stage_tutorial**" package was created.



Step 3. Creating a ROS Package for our simulations

Complete

Step 4. Importing a Map

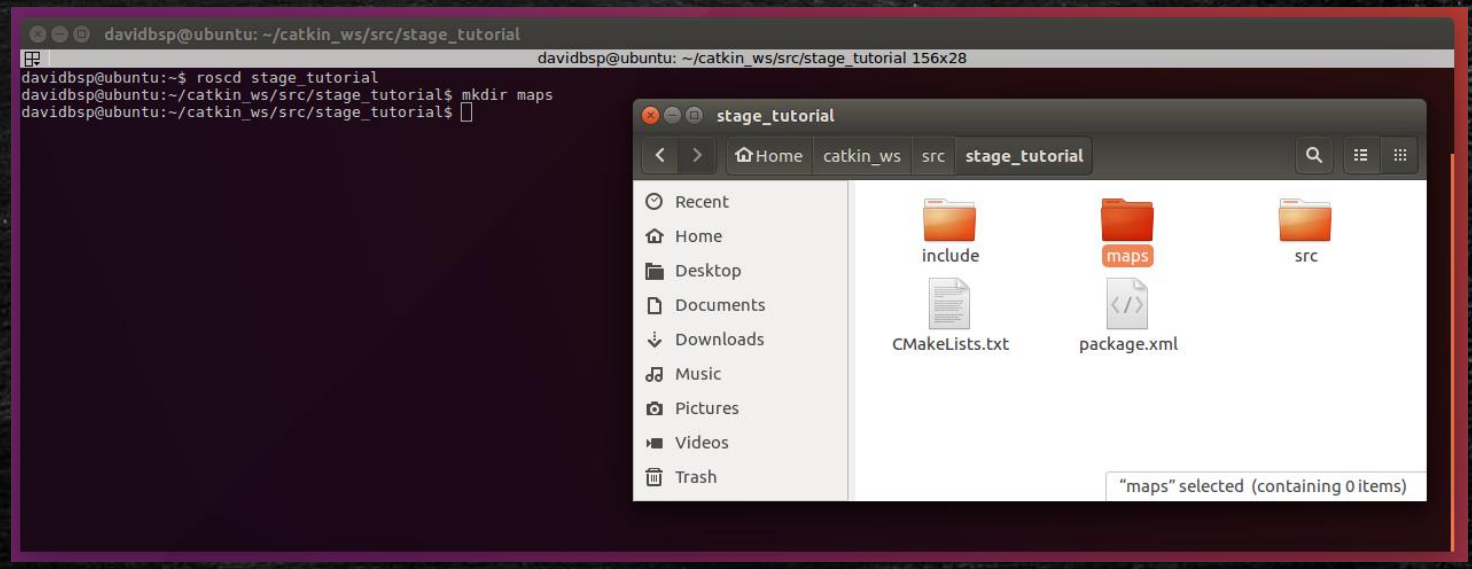
RobotCraft 2022: Stage/ ROS Tutorial

Step 4. Getting a Map and Importing it into the simulator

Prepare a folder inside your package for maps:

$ **roscd stage_tutorial**

$ **mkdir maps**



RobotCraft 2022: Stage/ROS Tutorial

Step 4. Getting a Map and Importing it into the simulator

Download the following zip file (also in Moodle):
http://ingeniarius.pt/davidbsp/robotcraft2017/rooms.zip

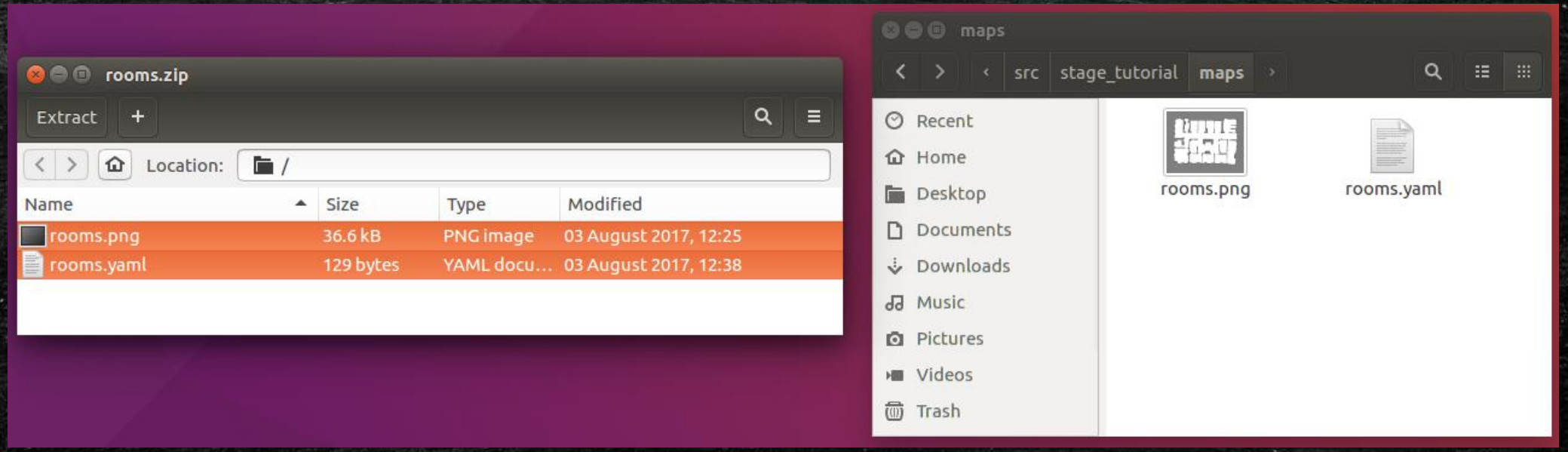
You will notice that you have two files there:

- rooms.png ← A bitmap image of an environment
- rooms.yaml ← A configuration file for the map

Extract them to your newly created “maps” folder (see next slide).

RobotCraft 2022: Stage/ ROS Tutorial

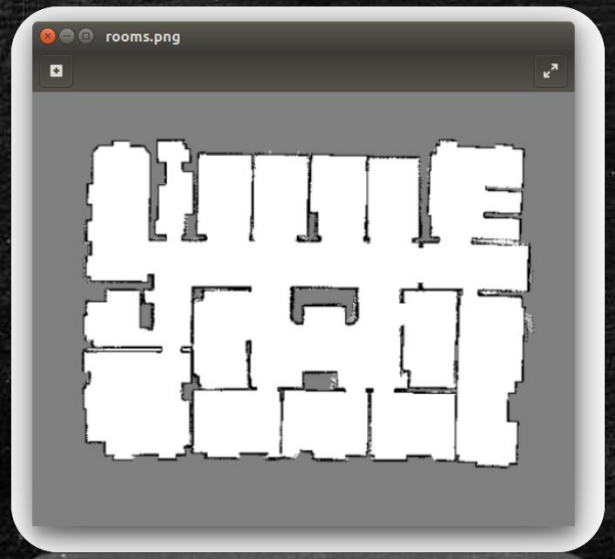
Step 4. Getting a Map and Importing it into the simulator



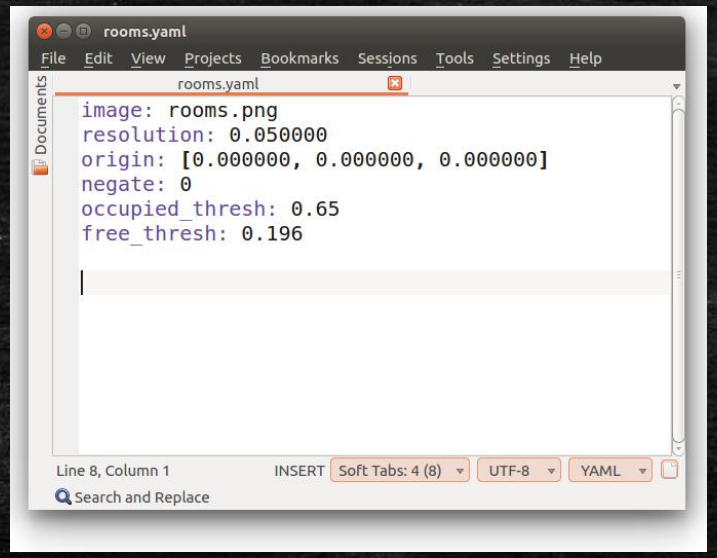
RobotCraft 2022: Stage/ ROS Tutorial

Step 4. Getting a Map and Importing it into the simulator

Examining the files:



rooms.png



rooms.yaml

Check the meaning of the ".yaml" parameters at:

http://wiki.ros.org/map_server#YAML_format

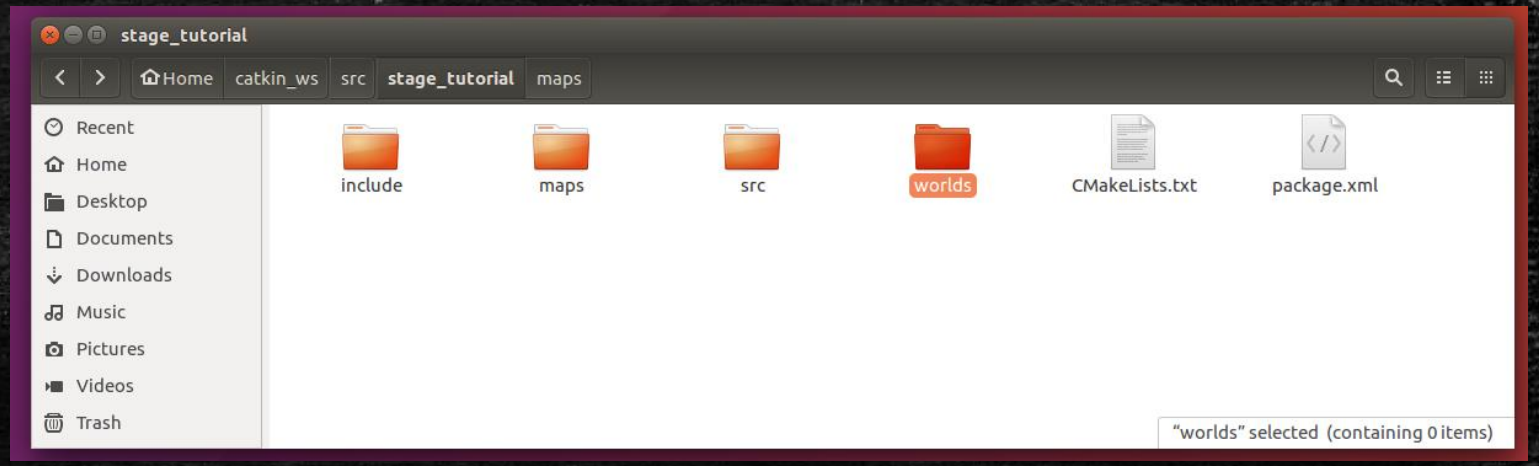
RobotCraft 2022: Stage/ROS Tutorial

Step 4. Getting a Map and Importing it into the simulator

Let's import the map into the Stage simulator now!

Prepare a folder inside your package for Stage "world" files:

```
$ roscd stage_tutorial
$ mkdir worlds
```



Step 4. Getting a Map and Importing it into the simulator

Now we need to create a world file. Let's call it "rooms.world".

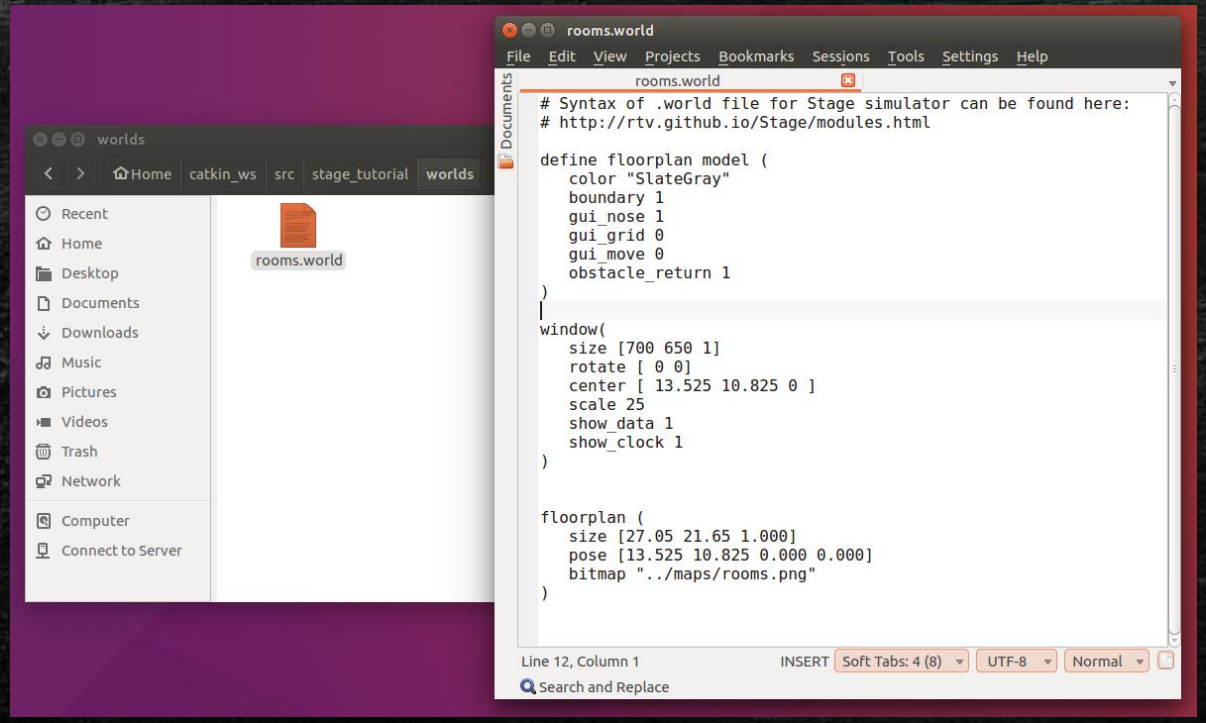
```
$ roscd stage_tutorial/worlds
$ gedit rooms.world
```

This will open an empty file in gedit (or on your preferred editor).

RobotCraft 2022: Stage/ ROS Tutorial

Step 4. Getting a Map and Importing it into the simulator

Let's put the following in the "rooms.world" file and save it:



Check the meaning of the ".world" parameters at:

http://rtv.github.io/Stage/modules.html

and:

http://rtv.github.io/Stage/group__model.html#details

RobotCraft 2022: Stage/ ROS Tutorial

Step 4. Getting a Map and Importing it into the simulator

Now that we have created the basic model, and defined the stage window properties, let's launch the stage simulation with our map:

In one terminal:

$ **roscd; catkin_make**

$ **roscore**

In another one:

$ **roscd stage_tutorial/worlds**

$ **rosrun stage_ros stageros rooms.world**

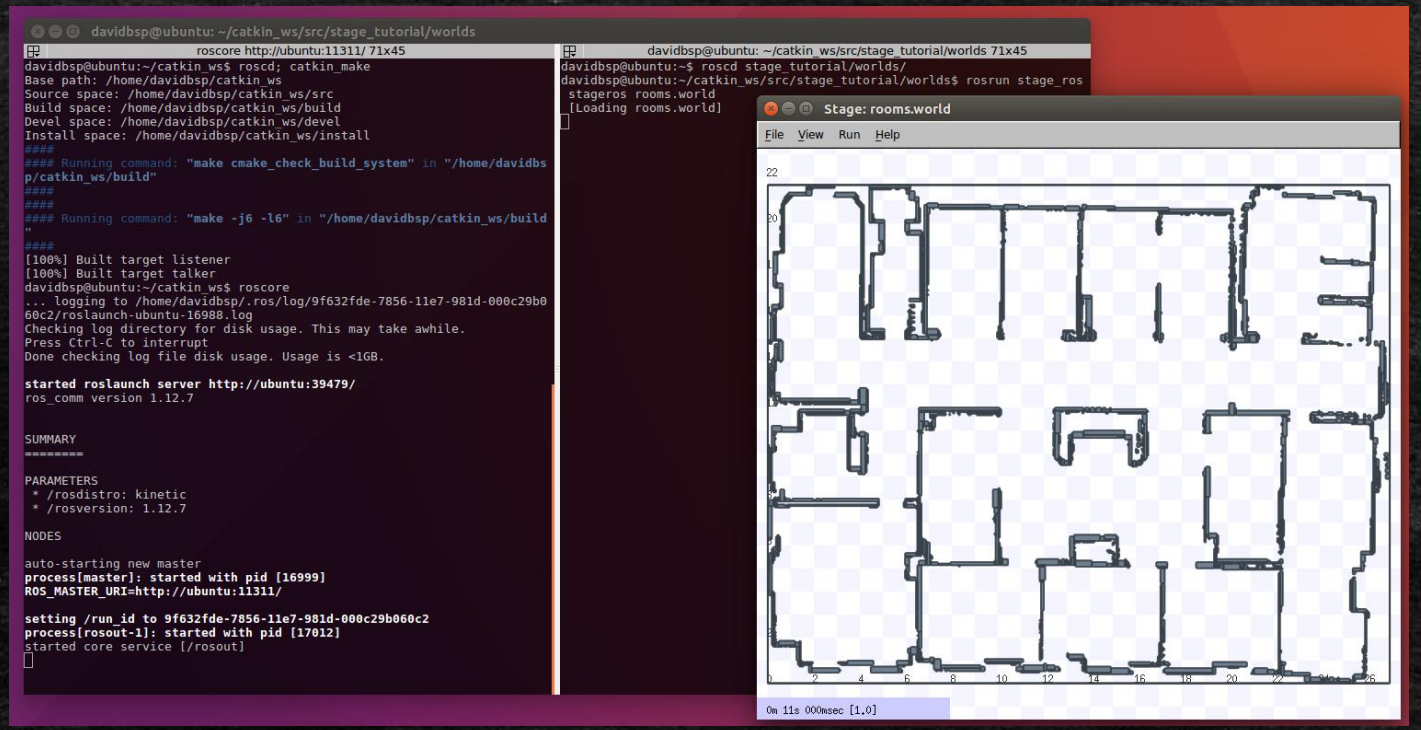
A simple compilation to register the package with ROS

Start the ROS master

Start stage from our "worlds" folder

RobotCraft 2022: Stage/ ROS Tutorial

Step 4. Getting a Map and Importing it into the simulator



Step 4. Getting a Map and importing it into the simulator

Complete

Step 5. Create a Robot

Step 5. Create a Robot and its sensors

Create a folder inside your package for robots:

```
$ roscd stage_tutorial
$ mkdir robots
```

Open a “simple_robot.inc” file in the “robots” folder:

```
$ cd robots
$ gedit simple_robot.inc
```

RobotCraft 2022: Stage/ ROS Tutorial

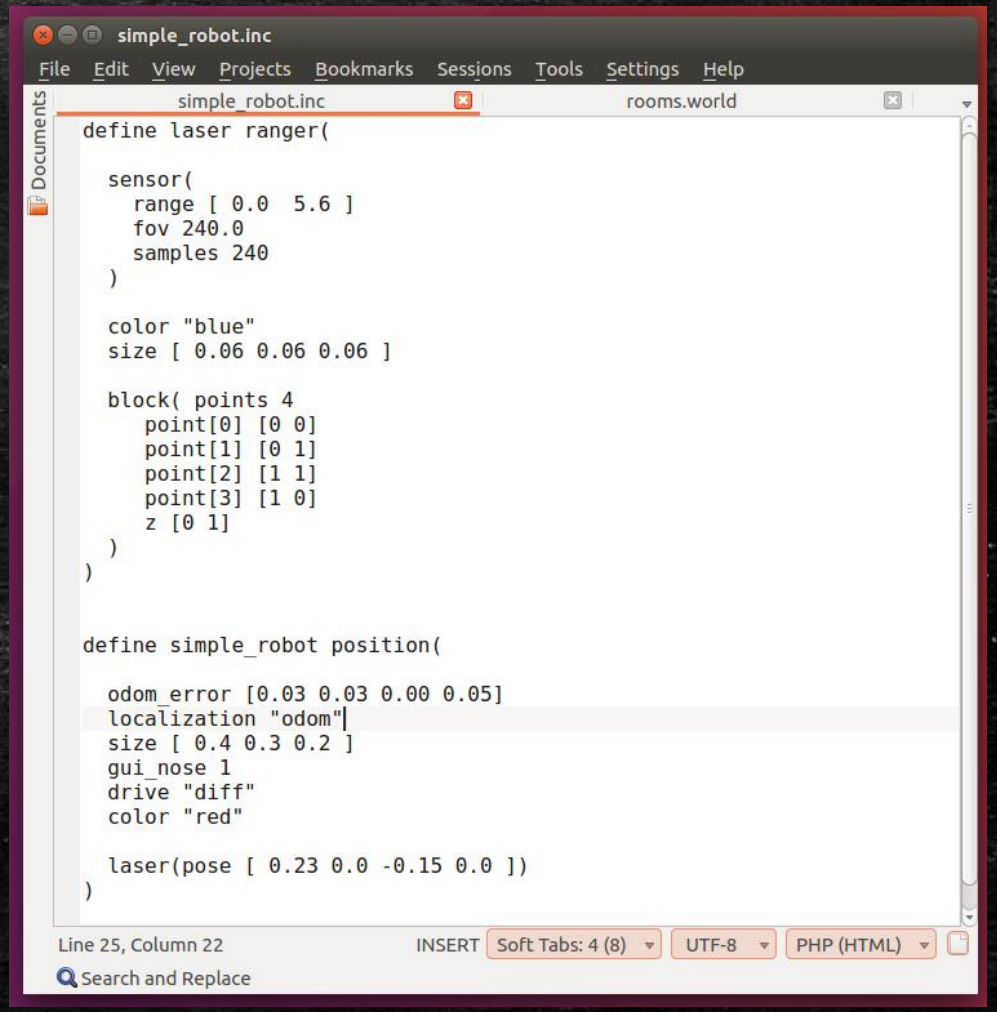
Step 5. Create a Robot and its sensors

Let's put the following in the "simple_robot.inc" file and save it:

Check the syntax of the "ranger" and "position" models at:

http://rtv.github.io/Stage/modules.html

Get an example of another robot here.



Step 5. Create a Robot and its sensors

We have defined a (very) simple red robot with a blue laser, now we need to include it in the world.

Add these two lines at the end of your “rooms.world” file:

```
include "../robots/simple_robot.inc"

simple_robot (pose [ 11.1  9.2  0.000  90.000 ] name "robot")
```

RobotCraft 2022: Stage/ ROS Tutorial

Step 5. Create a Robot and its sensors

rooms.world

File Edit View Projects Bookmarks Sessions Tools Settings Help

Documents

simple_robot.inc | rooms.world

```
# Syntax of .world file for Stage simulator can be found here:
# http://rtv.github.io/Stage/modules.html

define floorplan model (
   color "SlateGray"
   boundary 1
   gui_nose 1
   gui_grid 0
   gui_move 0
   obstacle_return 1
)

window(
   size [700 650 1]
   rotate [ 0 0]
   center [ 13.525 10.825 0 ]
   scale 25
   show_data 1
   show_clock 1
)


floorplan (
   size [27.05 21.65 1.000]
   pose [13.525 10.825 0.000 0.000]
   bitmap "../maps/rooms.png"
)

include "../robots/simple_robot.inc"

simple_robot (pose [ 11.1  9.2  0.000  90.000 ] name "robot")
```

Line 29, Column 1 INSERT Soft Tabs: 4 (8) UTF-8 Normal

Search and Replace

Step 5. Create a Robot and its sensors

Now that we have created a robot and placed it in the world, let's launch the stage simulation again:

In one terminal:

$ **roscore**

Start the ROS master

In another one:

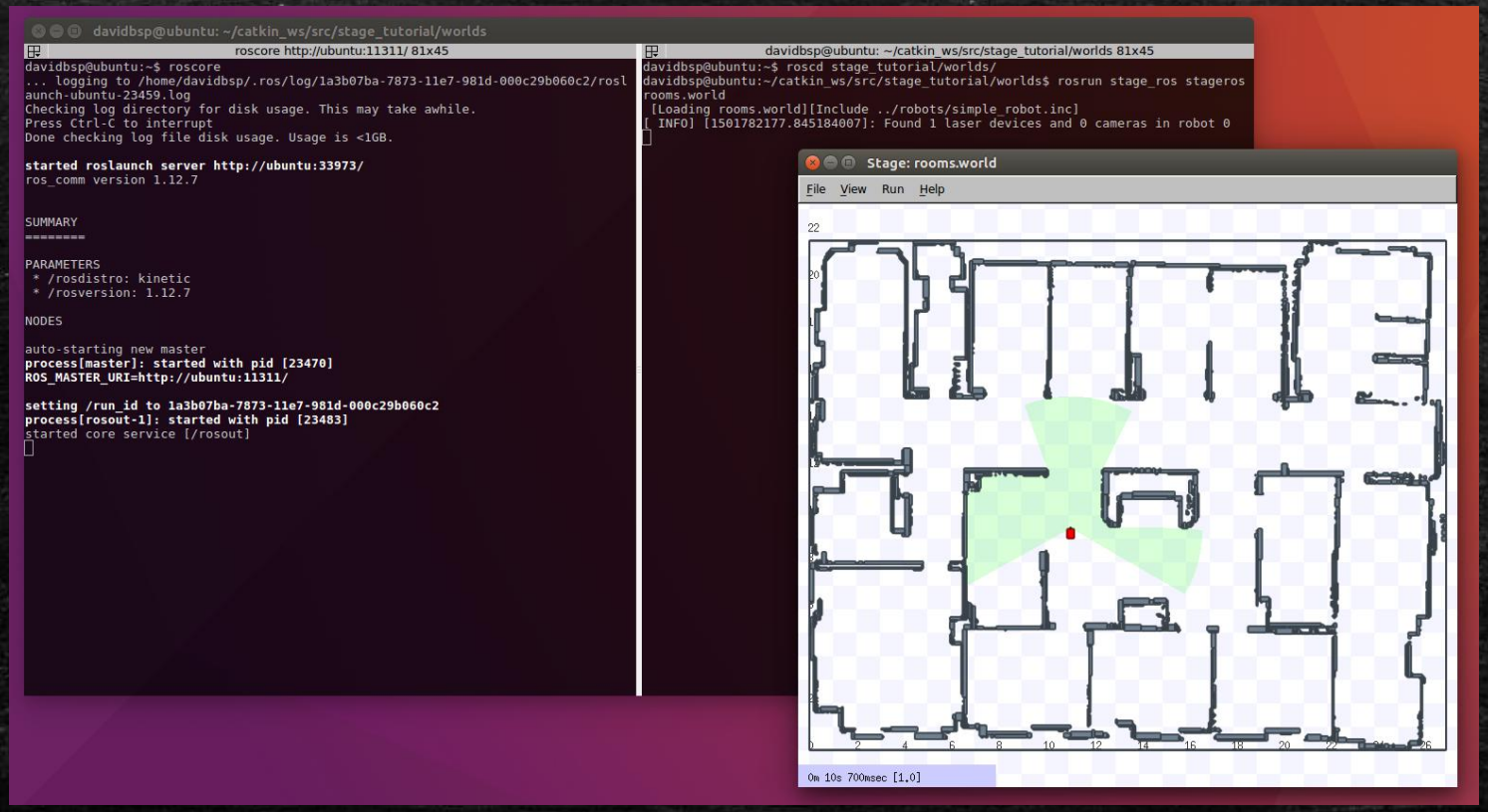
$ **roscd stage_tutorial/worlds**

$ **rosrun stage_ros stageros rooms.world**

Start stage from our "worlds" folder

RobotCraft 2022: Stage/ ROS Tutorial

Step 5. Create a Robot and its sensors

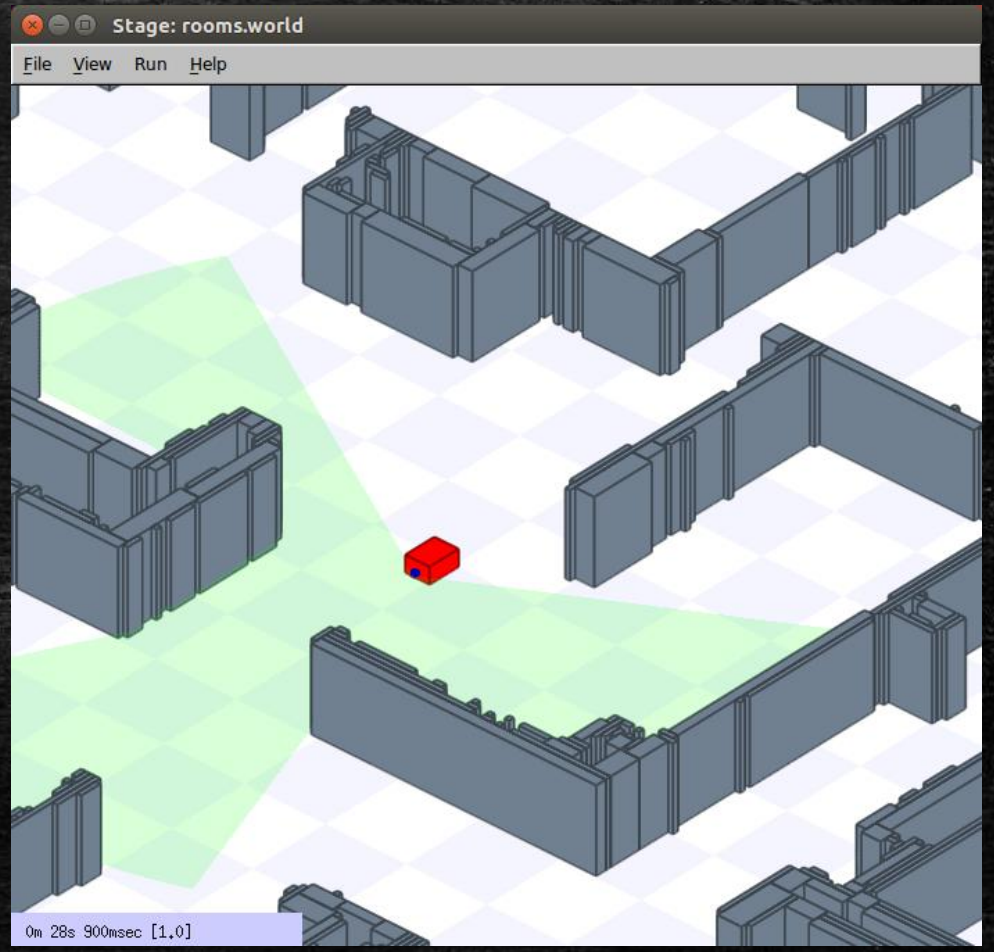


RobotCraft 2022: Stage/ROS Tutorial

Step 5. Create a Robot and its sensors

Let’s have a closer look to our robot.

Remember! Stage is 2.5D 🏝



Step 5. Create a Robot and its sensors

Complete

Step 6. Checking the Stage/ROS connection

Step 6. Checking the Stage/ROS connection

Launch the simulation and let's check the ROS nodes in our system!

$ **rosnode list**

/rosout

/stageros

Let's get more information about the **stageros** node:

$ **rosnode info stageros**

RobotCraft 2022: Stage/ ROS Tutorial

Step 6. Checking the Stage/ROS connection

```
davidbsp@ubuntu: ~/catkin_ws/src/stage_tutorial/worlds
davidbsp@ubuntu: ~/catkin_ws/src/stage_
davidbsp@ubuntu:~/catkin_ws/src/stage_tutorial/worlds$ rosnode info stageros
--------------------------------------------------------------------------------
Node [/stageros]
Publications:
 * /base_scan [sensor_msgs/LaserScan]
 * /rosout [rosgraph_msgs/Log]
 * /clock [rosgraph_msgs/Clock]
 * /odom [nav_msgs/Odometry]
 * /tf [tf2_msgs/TFMessage]
 * /base_pose_ground_truth [nav_msgs/Odometry]

Subscriptions:
 * /cmd_vel [unknown type]

Services:
 * /reset_positions
 * /stageros/set_logger_level
 * /stageros/get_loggers


contacting node http://ubuntu:37407/ ...
Pid: 24419
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
```

RobotCraft 2022: Stage/ROS Tutorial

Step 6. Checking the Stage/ROS connection

Let's check the ROS topics in our system!

$ **rostopic list**

/base_pose_ground_truth	Perfect Pose of the Robot (should not be used!)
/base_scan	**Laser Scans**
/clock	Simulated Time (for synchronization with ROS)
/cmd_vel	**Velocity commands sent to the robot**
/odom	**Pose of the Robot (with odometry error)**
/rosout	ROS Console Log Reporting
/rosout_agg	ROS Console Log Reporting (aggregated feed)
/tf	Frame Transformations

RobotCraft 2022: Stage/ROS Tutorial

Step 6. Checking the Stage/ROS connection

How fast is stageros publishing laser scans and odometry information?

$ **rostopic hz /base_scan**

$ **rostopic hz /odom**

```
subscribed to [/base_scan]
WARNING: may be using simulated time
average rate: 10.000
    min: 0.000s max: 0.200s std dev: 0.05000s window: 9
average rate: 10.000
    min: 0.000s max: 0.200s std dev: 0.04714s window: 19
average rate: 10.000
    min: 0.000s max: 0.200s std dev: 0.04629s window: 29
```

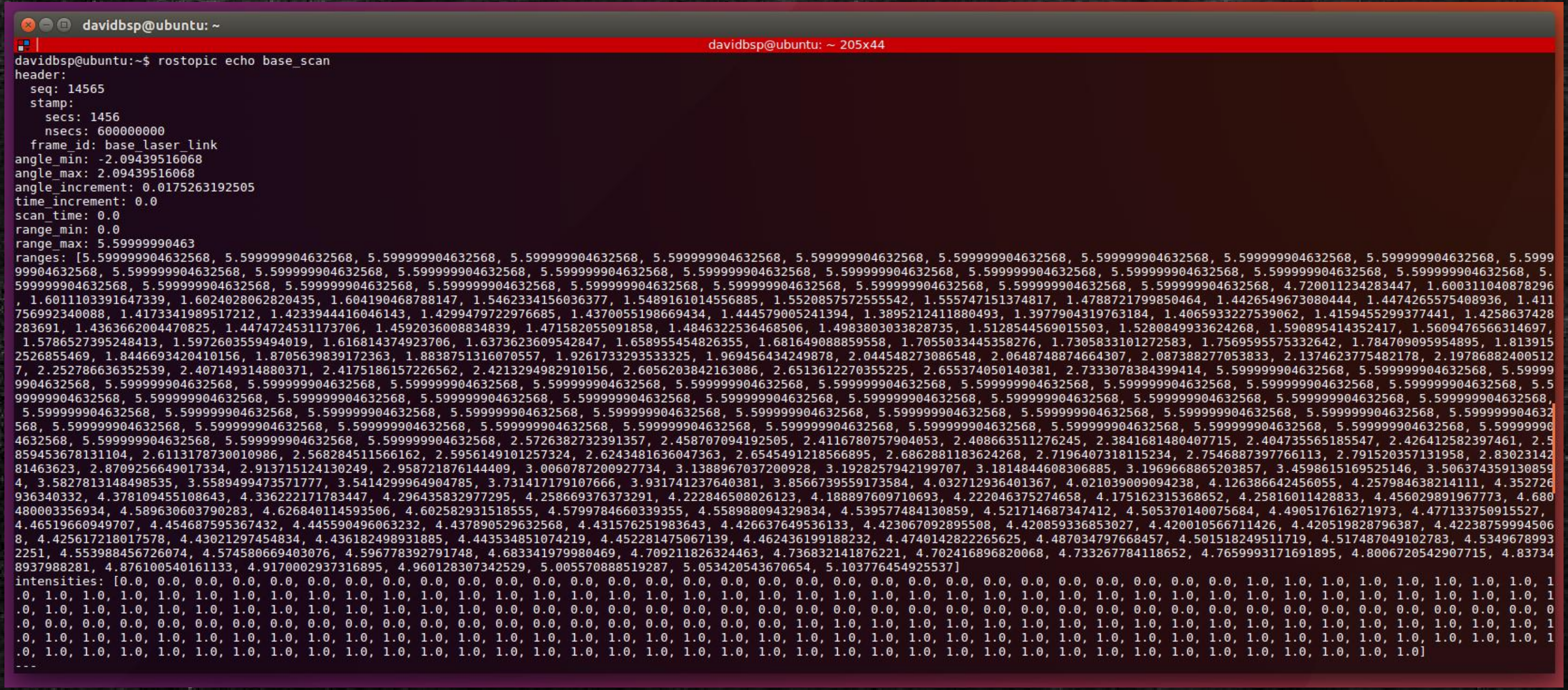
10 Hz (10 times per second). A msg is published every 0.1 seconds.

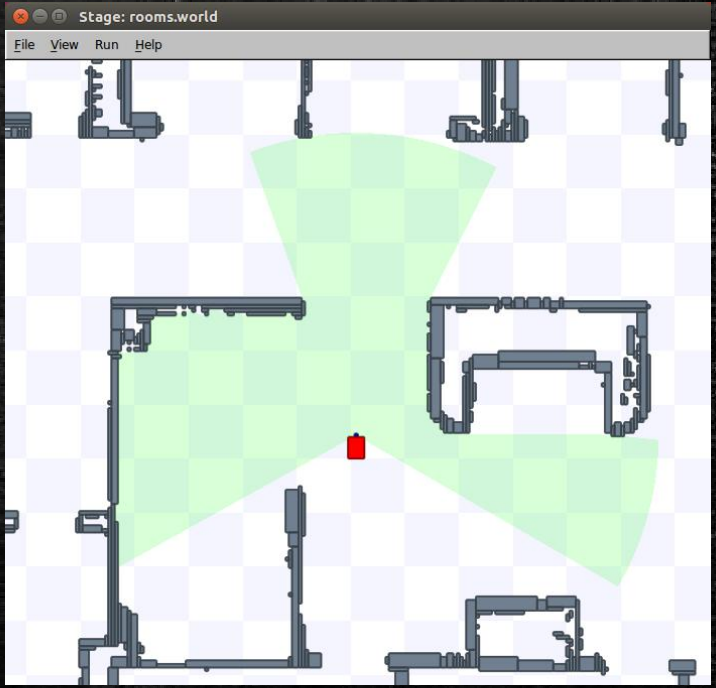
RobotCraft 2022: Stage/ROS Tutorial

Step 6. Checking the Stage/ROS connection

Let’s have a look at the laserscan messages being published in /base_scan:

$ **rostopic echo /base_scan**





Step 6. Checking the Stage/ROS connection

Let's now publish some velocity commands to /cmd_vel:

```
$ rostopic pub -r 10 /cmd_vel geometry_msgs/Twist -- '[0.75, 0.0, 0.0]' '[0.0, 0.0, 0.0]'
```

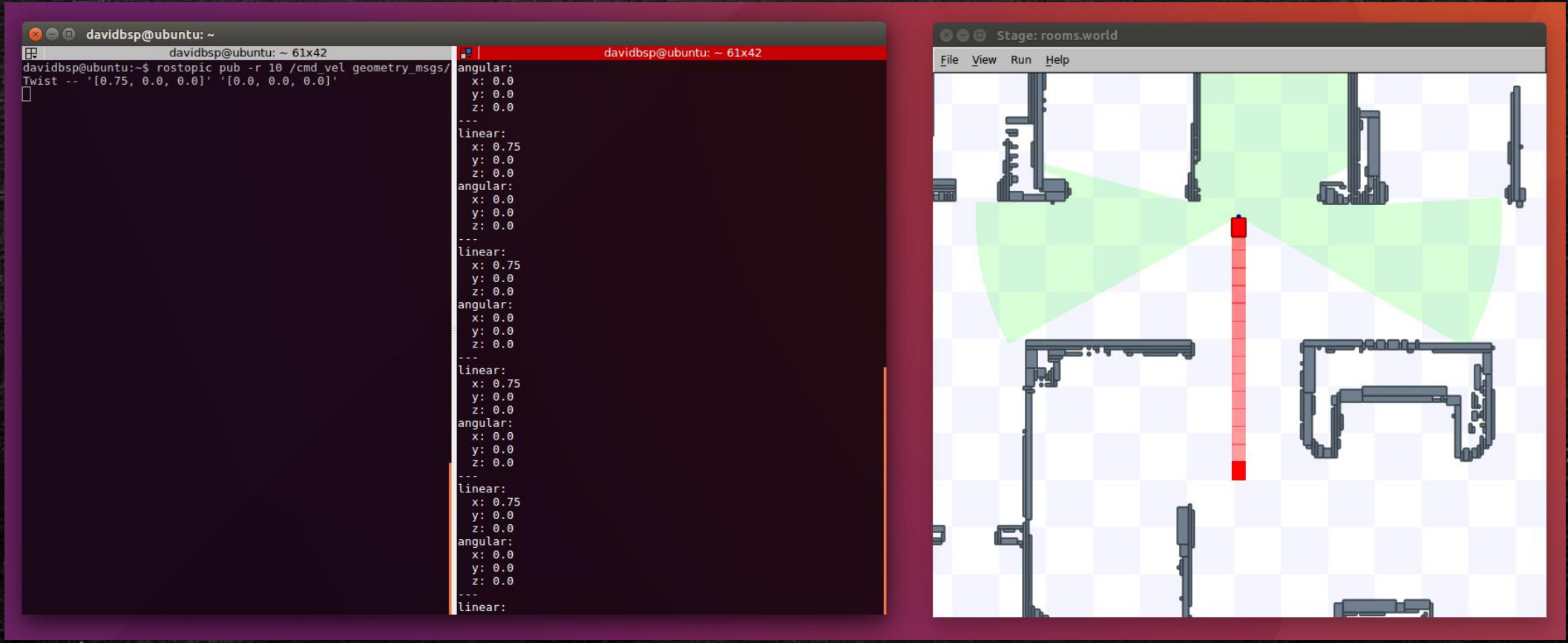
Another equivalent syntax is:

```
$ rostopic pub -r 10 /cmd_vel geometry_msgs/Twist '{linear: {x: 0.75, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}'
```

The above commands send velocity commands (move forward 0.75m/s) at 10 times per second (i.e. 10 Hz) to the robot base using the rostopic tool.

RobotCraft 2022: Stage/ROS Tutorial

Step 6. Checking the Stage/ROS connection



PS: Don't forget to type "Ctrl+C" to interrupt the "rostopic pub" command, otherwise your robot will hit a wall!

Step 6. Checking the Stage/ROS connection

Stageros also offers a service for resetting the initial state of the world

$ **rosservice list**

/reset_positions ← **Reset the initial state of the world**

/rosout/get_loggers

/rosout/set_logger_level

Logging Services provided by the /rosout node

/stageros/get_loggers

/stageros/set_logger_level

Logging Services provided by the /stageros node

Step 6. Checking the Stage/ROS connection

Let's check what is the service type of "**reset_positions**":

$ **rosservice type reset_positions**

std_srvs/Empty

reset_positions is of "Empty" type, thus no data is exchanged between the service and the client. So we can call it without arguments:

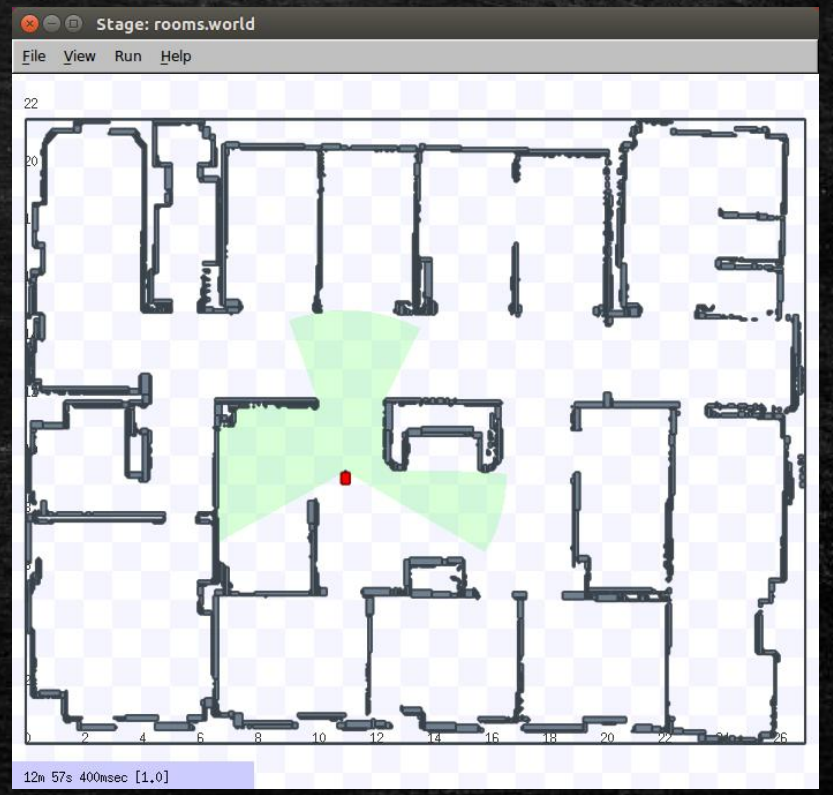
$ **rosservice call reset_positions**

RobotCraft 2022: Stage/ROS Tutorial

Step 6. Checking the Stage/ROS connection

After calling the reset_positions service,

your robot will be back at its initial position!



Step 6. Checking the Stage/ROS connection

Complete

Step 7. Teleoperation

Step 7. Adding a teleoperation node to control the robot

Teleoperation allows you to control your robot's movements
(e.g. using your keyboard, a joystick, etc.).

Which topic does the robot use to listen to commands and drive?

/cmd_vel

So, to control a robot using the keyboard, we need a ROS node that listens to keyboard events (keystrokes), and publishes velocity commands (in /**cmd_vel**) to move the robot base.

Step 7. Adding a teleoperation node to control the robot

A generic teleoperation node with the keyboard is already available for ROS*.

We just need to fetch it and install it:

$ **sudo apt install ros-melodic-teleop-twist-keyboard**

Now run the teleoperation node (assuming that stage is up and running):

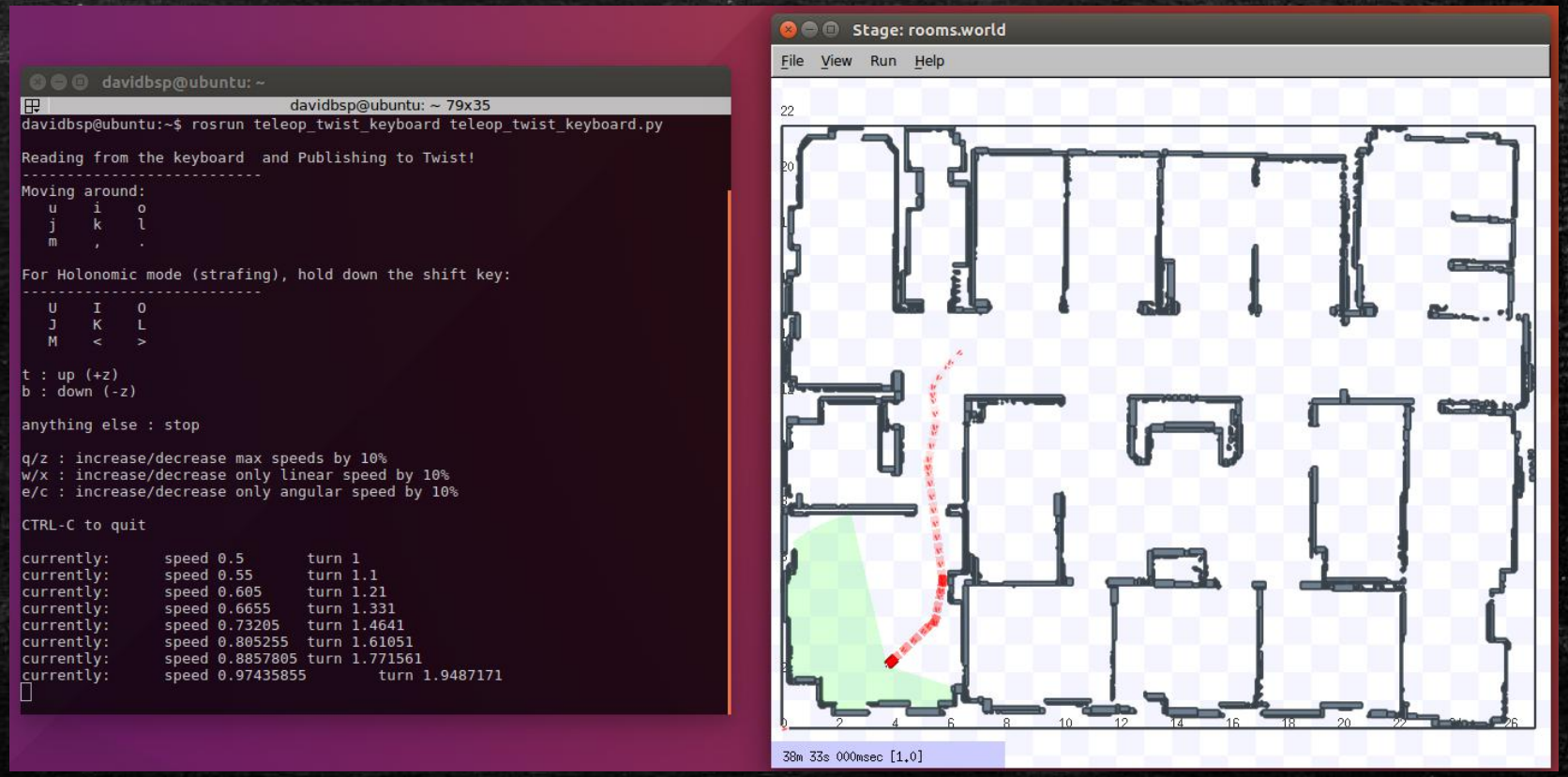
$ **rosrun teleop_twist_keyboard teleop_twist_keyboard.py**

* http://wiki.ros.org/teleop_twist_keyboard

RobotCraft 2022: Stage/ ROS Tutorial

Step 7. Adding a teleoperation node to control the robot

You can now control the robot with the keyboard!



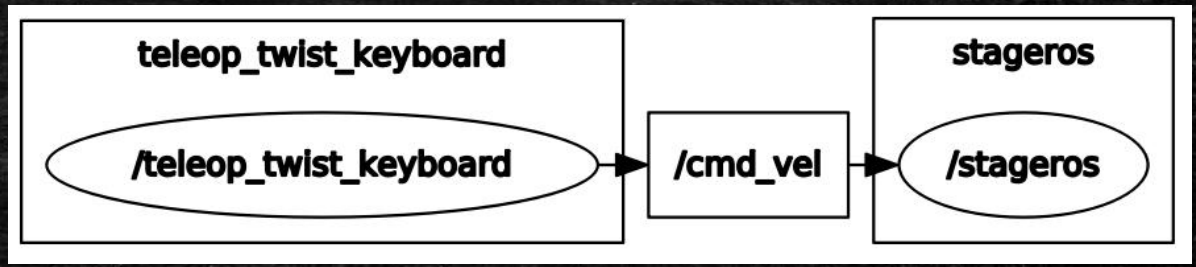
RobotCraft 2022: Stage/ROS Tutorial

Step 7. Adding a teleoperation node to control the robot

A bit more information about the **teleop_twist_keyboard** node!

```
davidbsp@ubuntu: ~
davidbsp@ubuntu: ~ 78x31
davidbsp@ubuntu:~$ rosnode info teleop_twist_keyboard
--------------------------------------------------------------------------------
Node [/teleop_twist_keyboard]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /cmd_vel [geometry_msgs/Twist]

Subscriptions:
 * /clock [rosgraph_msgs/Clock]

Services:
 * /teleop_twist_keyboard/set_logger_level
 * /teleop_twist_keyboard/get_loggers


contacting node http://ubuntu:42665/ ...
Pid: 19317
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
 * topic: /cmd_vel
    * to: /stageros
    * direction: outbound
    * transport: TCPROS
 * topic: /clock
    * to: /stageros (http://ubuntu:42431/)
    * direction: inbound
    * transport: TCPROS
```

$ rosrun rqt_graph rqt_graph



Step 7. Adding a teleoperation node to control the robot

Complete

Step 8. Reading sensor data and publishing cmd_vels

Step 8. Source code for reading sensor data and publishing cmd_vels

To replace the teleoperation node with “autonomous” navigation, we should have a ROS node that generates velocity commands to the robot base.

How should the node decide which velocity commands to send?

By reading sensor data to avoid hitting walls and obstacles!

Thus, we need to read sensor data from the robot’s laser, and drive the robot away from obstacles. → We need a “**reactive navigation**” node.

RobotCraft 2022: Stage/ ROS Tutorial

Step 8. Source code for reading sensor data and publishing cmd_vels

sensor_msgs/LaserScan Message

File: sensor_msgs/LaserScan.msg

Raw Message Definition

```
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

The **sensor_msgs/LaserScan** Message.

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

stageros publishes the LaserScan messages of the robot in the **/base_scan** topic.

RobotCraft 2022: Stage/ ROS Tutorial

Step 8. Source code for reading sensor data and publishing cmd_vels

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

geometry_msgs/Vector3 Message

File: geometry_msgs/Vector3.msg

Raw Message Definition

```
# This represents a vector in free space.
# It is only meant to represent a direction. Therefore, it does not
# make sense to apply a translation to it (e.g., when applying a
# generic rigid transformation to a Vector3, tf2 will only apply the
# rotation). If you want your data to be translatable too, use the
# geometry_msgs/Point message instead.

float64 x
float64 y
float64 z
```

Compact Message Definition

```
float64 x
float64 y
float64 z
```

linear.x
linear.y
linear.z

angular.x
angular.y
angular.z

The **geometry_msgs/Twist** Message.

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

stageros subscribes the Twist messages in the /cmd_vel topic to drive the robot around.

RobotCraft 2022: Stage/ROS Tutorial

Step 8. Source code for reading sensor data and publishing cmd_vels

Let's create the source file "reactive_navigation.cpp" in the "src" folder of your "stage_tutorial" package.

$ **roscd stage_tutorial/src**

$ **gedit reactive_navigation.cpp**

In this step, we only provide a basic node, which drives the robot forward, and stops until an obstacle is near.

You will have to code your own reactive navigation node during the week!

RobotCraft 2022: Stage/ ROS Tutorial

Step 8. Source code for reading sensor data and publishing cmd_vels

reactive_navigation.cpp

```cpp
// "reactive_navigation" node: subscribes laser data and publishes velocity commands

#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"
#include "geometry_msgs/Twist.h"

//some global variables to "perceive" the world around the robot:
double obstacle_distance;
bool robot_stopped;
//...

void laserCallback(const sensor_msgs::LaserScan::ConstPtr& msg){
  //Access the scan ranges array via: msg->ranges[0...size-1]
  //Note that:
  //msg->ranges[0] corresponds to the range with angle_min (msg->angle_min)
  //msg->ranges[size-1] corresponds to the range with angle_max (msg->angle_max)
  //msg->ranges[i] correspond to the range with angle_min + i*angle_increment (msg->angle_increment)
  //All these angles are in radians!

  if (!robot_stopped){  //print only when robot is moving...
    ROS_INFO("Received a LaserScan with %i samples", (int) msg->ranges.size() );

    //For simplicity, let's save the distance of the closer obstacle to the robot:
    obstacle_distance = *std::min_element (msg->ranges.begin(), msg->ranges.end());
    ROS_INFO("minimum distance to obstacle: %f", obstacle_distance);
  }

}

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

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

  ros::NodeHandle n;

  //Publisher for /cmd_vel
  ros::Publisher cmd_vel_pub = n.advertise<geometry_msgs::Twist>("cmd_vel", 100);
  //Subscriber for /base_scan
  ros::Subscriber laser_sub = n.subscribe("base_scan", 100, laserCallback);

  ros::Rate loop_rate(10); //10 Hz
```

reactive_navigation.cpp

```cpp
  ros::Rate loop_rate(10); //10 Hz

  //initializations:
  geometry_msgs::Twist cmd_vel_msg;
  robot_stopped = true;
  obstacle_distance = 1.0;

  while (ros::ok()){

    //fill the "cmd_vel_msg" data according to some conditions (depending on laser data)

    if (obstacle_distance > 0.5){
        //move forward:
        cmd_vel_msg.linear.x = 1.0;
        cmd_vel_msg.angular.z = 0.0;

        if (robot_stopped){ //just print once:
            ROS_INFO("Moving Forward");
            robot_stopped = false;
        }

    }else{ //stop:
        cmd_vel_msg.linear.x = 0.0;
        cmd_vel_msg.angular.z = 0.0;

        if (!robot_stopped){ //just print once:
            ROS_INFO("Stopping");
            robot_stopped = true;
        }
    }

    //publish velocity commands:
    cmd_vel_pub.publish(cmd_vel_msg);

    ros::spinOnce();

    loop_rate.sleep();
  }

  return 0;
}
```

You can use this code as a basis. Take a look at the repository linked at the end of the slides for something you can copy and paste. Are the two versions different? Why?

Step 8. Source code for reading sensor data and publishing cmd_vels

After saving the “reactive_navigation.cpp” file in your “src” folder, we need to build the node.

Open your package “CMakeLists.txt”:

$ **roscd stage_tutorial**
$ **gedit CMakeLists.txt**

...and add the following two lines at the end of the file:

```
add_executable(reactive_navigation src/reactive_navigation.cpp)
target_link_libraries(reactive_navigation ${catkin_LIBRARIES})
```

RobotCraft 2022: Stage/ROS Tutorial

Step 8. Source code for reading sensor data and publishing cmd_vels

Save your “CMakeLists.txt” file, and now let’s compile our package again:

$ **roscd**

$ **catkin_make**

```
davidbsp@ubuntu: ~/catkin_ws
davidbsp@ubuntu: ~/catkin_ws 110x22
davidbsp@ubuntu:~$ roscd
davidbsp@ubuntu:~/catkin_ws$ catkin_make
Base path: /home/davidbsp/catkin_ws
Source space: /home/davidbsp/catkin_ws/src
Build space: /home/davidbsp/catkin_ws/build
Devel space: /home/davidbsp/catkin_ws/devel
Install space: /home/davidbsp/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/davidbsp/catkin_ws/build"
####
####
#### Running command: "make -j6 -l6" in "/home/davidbsp/catkin_ws/build"
####
Scanning dependencies of target reactive_navigation
[ 33%] Built target listener
[ 50%] Building CXX object stage_tutorial/CMakeFiles/reactive_navigation.dir/src/reactive_navigation.cpp.o
[ 83%] Built target talker
[100%] Linking CXX executable /home/davidbsp/catkin_ws/devel/lib/stage_tutorial/reactive_navigation
[100%] Built target reactive_navigation
davidbsp@ubuntu:~/catkin_ws$
```

Please check your source code, in case you run into any compilation errors!

Step 8. Source code for reading sensor data and publishing cmd_vels

Now let's run our node (assuming that stage is up and running):

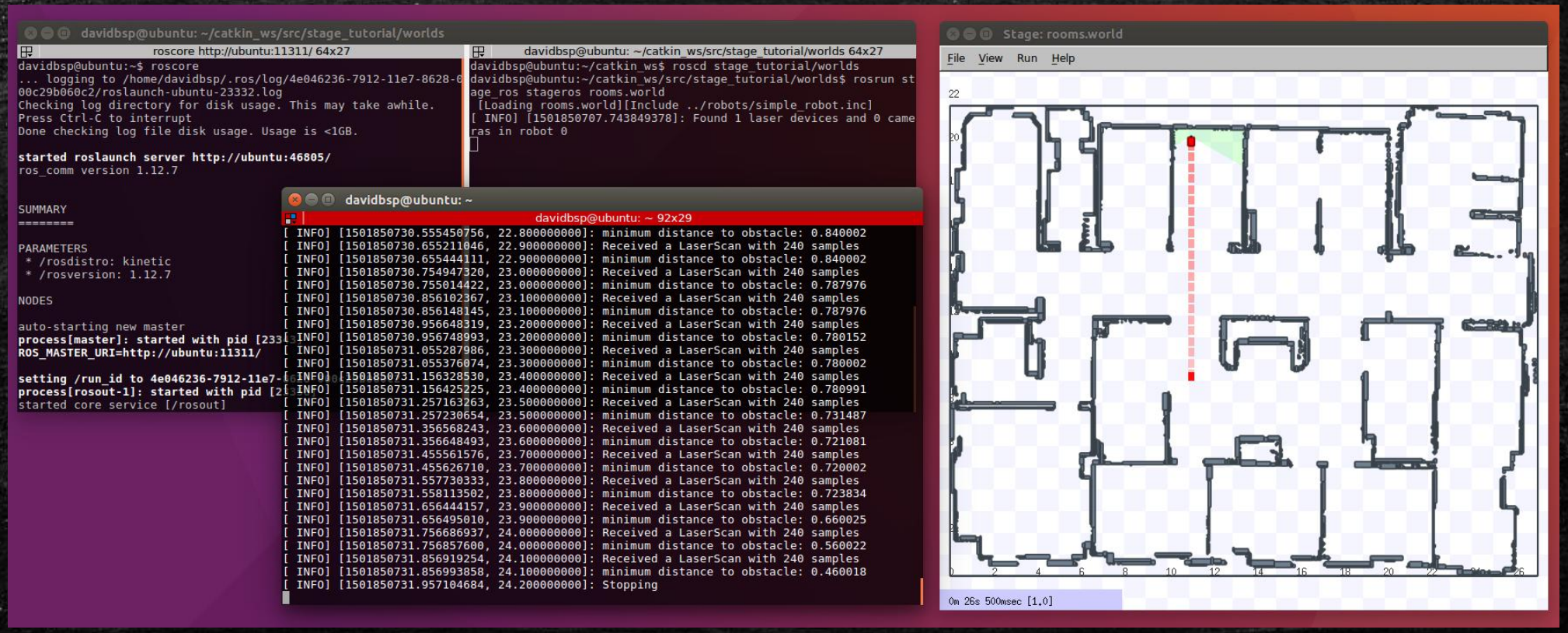
```
$ rosrun stage_tutorial reactive_navigation
```

our package name

our node name

RobotCraft 2022: Stage/ ROS Tutorial

Step 8. Source code for reading sensor data and publishing cmd_vels



Step 8. Source code for reading sensor data and publishing velocity commands

Complete

Step 9. roslaunch

Step 9. Launching everything with roslaunch

At the moment, to start our whole system we need to run in separate terminals:

```
$ roscore
```

```
$ roscd stage_tutorial/world
$ rosrun stage_ros stageros rooms.world
```

```
$ rosrun stage_tutorial reactive_navigation
```

...but it would be much more convenient to start our system with just one command!

RobotCraft 2022: roslaunch

Bringup a set of ROS nodes to provide some aggregate functionality.

Usage: **roslaunch <pkg_name> <file.launch>**

Example:

davidbsp@ubuntu:~/catkin_ws$ **roslaunch roscpp_tutorials talker_listener.launch**

talker_listener.launch file (XML description):

```
<launch>
  <node name="listener" pkg="roscpp_tutorials" type="listener" output="screen"/>
  <node name="talker" pkg="roscpp_tutorials" type="talker" output="screen"/>
</launch>
```

Read more at: http://wiki.ros.org/roslaunch/XML

RobotCraft 2022: Stage/ ROS Tutorial

Step 9. Launching everything with roslaunch

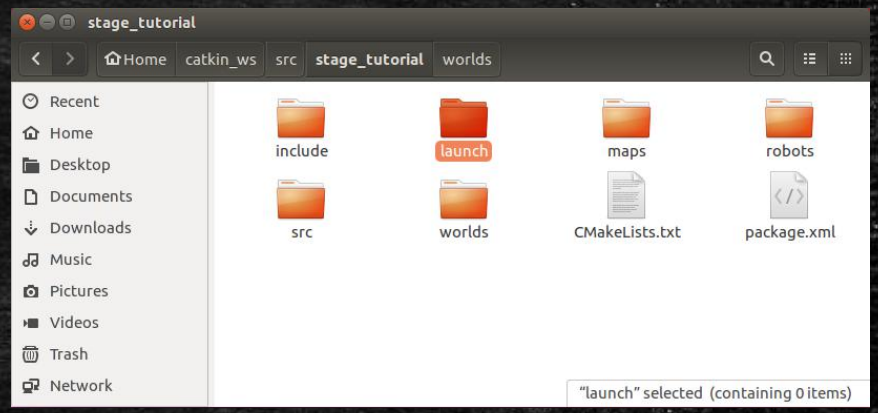
Let's create a launch file to start all our nodes!

Prepare a folder inside your package for launch files:

```
$ roscd stage_tutorial
$ mkdir launch
```

Open an "autonomous.launch" file inside the launch folder:

```
$ cd launch
$ gedit autonomous.launch
```



RobotCraft 2022: Stage/ ROS Tutorial

Step 9. Launching everything with roslaunch

Place the following lines inside your "**autonomous.launch**" file and save it:

```
<!-- autonomous.launch: launch stage with our navigation node -->

<launch>
    <node pkg="stage_ros" type="stageros" name="stageros" args="$(find
stage_tutorial)/worlds/rooms.world" />

    <node pkg="stage_tutorial" type="reactive_navigation" name="reactive_navigation"
output="screen"/>
</launch>
```

Launch files automatically start a "roscore" for us

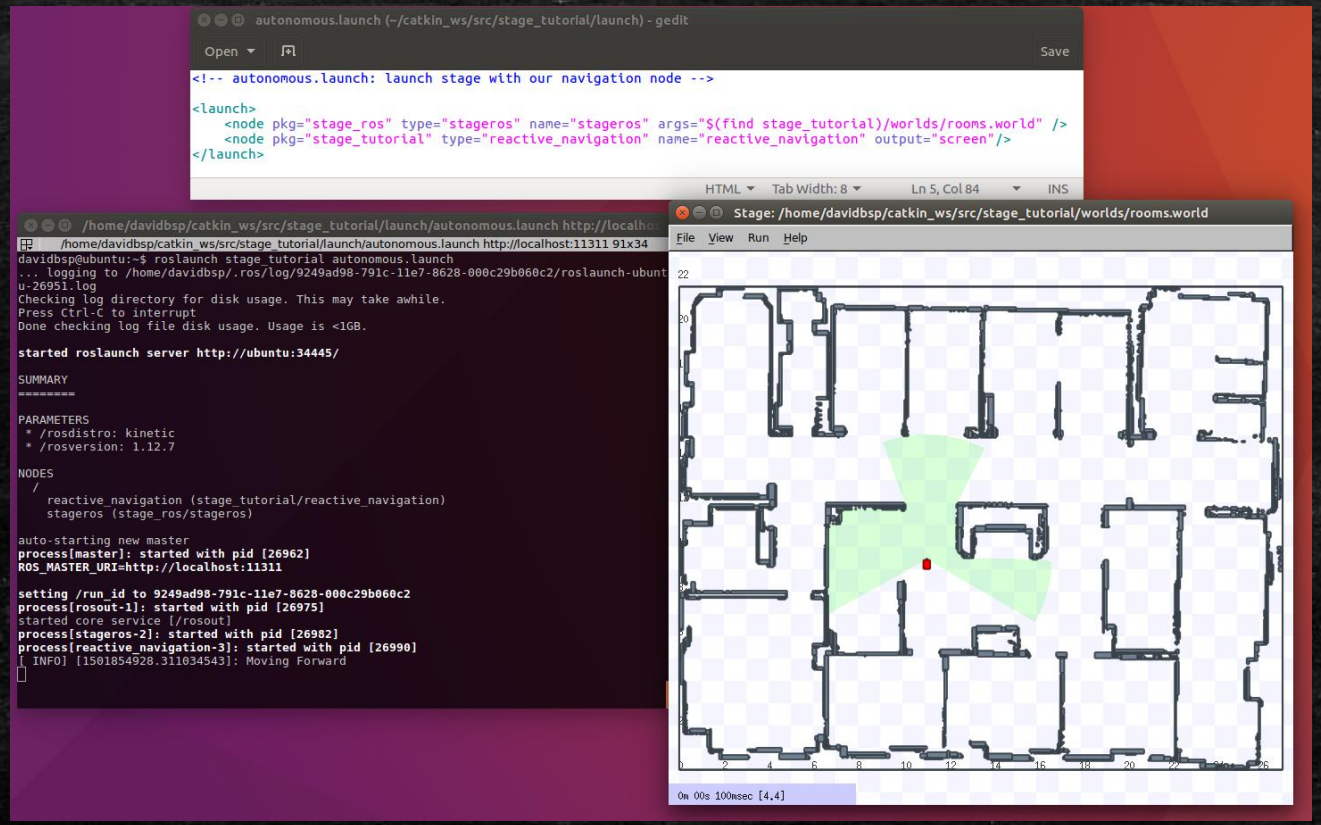
Run stage with the correct "world" file

Run our node to drive the robot (and print the node output)

RobotCraft 2022: Stage/ROS Tutorial

Step 9. Launching everything with roslaunch

Let's run our launch file: $ **roslaunch stage_tutorial autonomous.launch**



RobotCraft 2022: Stage/ROS Tutorial

Step 9. Launching everything with roslaunch

We can create another launch file, called "**teleop.launch**" to run the keyboard teleoperation instead of our navigation node:

```
<!-- teleop.launch: launch stage with keyboard teleoperation -->

<launch>
    <node pkg="stage_ros" type="stageros" name="stageros" args="$(find stage_tutorial)/worlds/rooms.world" />

    <node pkg="teleop_twist_keyboard" type="teleop_twist_keyboard.py" name="teleop_twist_keyboard" output="screen"/>
</launch>
```

Launch files automatically start a "roscore" for us

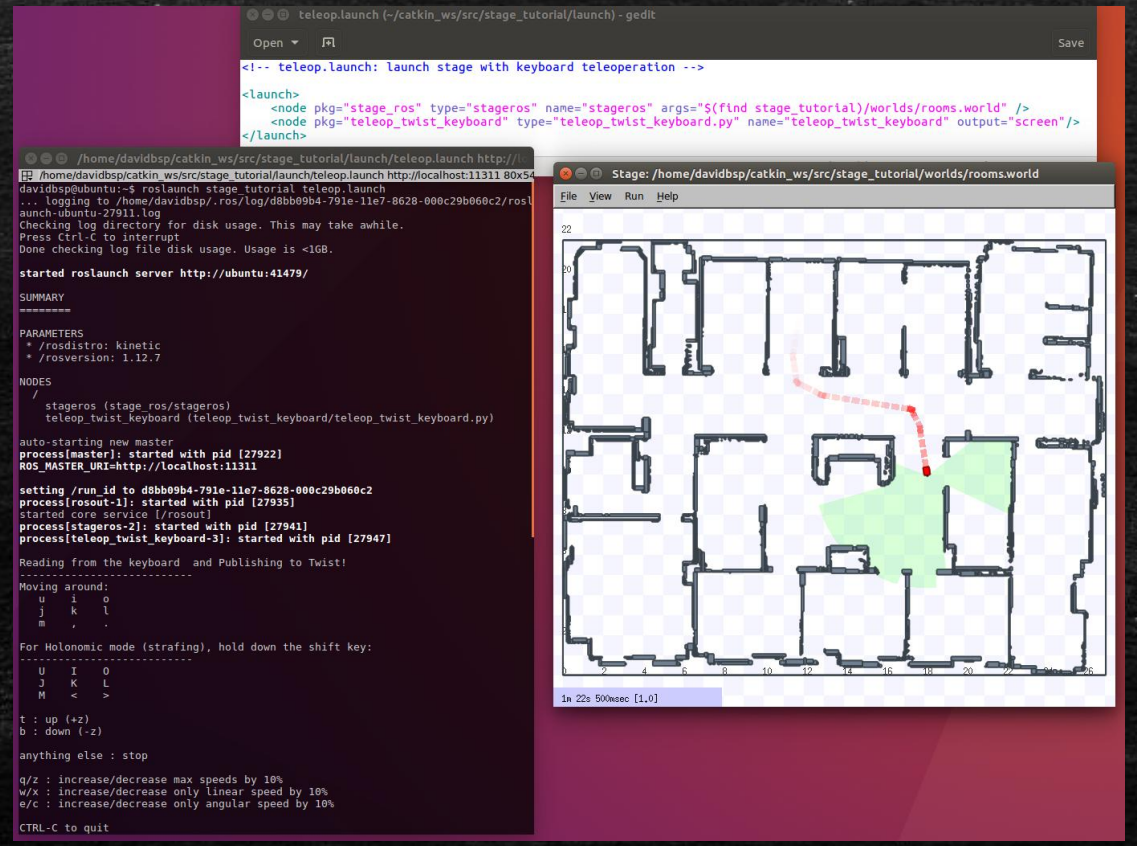
Run stage with the correct "world" file

Run the teleop node to drive the robot (and print the node output)

RobotCraft 2022: Stage/ROS Tutorial

Step 9. Launching everything with roslaunch

Let's run our new launch file: $ **roslaunch stage_tutorial teleop.launch**



Step 9. Launching everything with roslaunch

Complete

Improvements

Outset

- Again, this final part of the tutorial is meant to be hands-on.
  - The slides just ran out.
- We will basically fill the remaining time with whatever we want to do to our robot.
  - Ideas are welcome! What do you think we should do? Was there anything you did not understand?
- First ideas
  - Can we add a second robot? What happens to the topics?

Assignment

RobotCraft 2022: Stage/ ROS Tutorial

Craft 6.2 (Simulating with ROS)

#Assignment for this Week – Part 1:

- Create a ROS package named "**simstage_groupX**" (where "X" is your group number).
- Inside your package, create the needed **files** to simulate a virtual world with a robot in Stage.
- You can get any map from HERE, or create your own one, *e.g.* try searching for occupancy grids in the web.
  Note: "pgm" is just another format for bitmapped images, like "png".
- Your robot should encompass at least 3 polygonal blocks, and **robot design creativity** will be rewarded with extra points!

RobotCraft 2022: Stage/ ROS Tutorial

Craft 6.2 (Simulating with ROS)

#Assignment for this Week – Part 2:

- Add a **ROS node** inside your package, which implements a **reactive navigation** strategy.
- The robot should be able to avoid obstacles without hitting them, and drive continuously so as to (eventually) visit all areas of the environment.
- Unlike the example of the tutorial, the robot should never stop moving.
- You can gain inspiration from robots on **random walks** (e.g. here), and/or **wall-following** robots (e.g. here).

RobotCraft 2022: Stage/ ROS Tutorial

Craft 6.2 (Simulating with ROS)

#Assignment for this Week – Part 3:

- Install the SLAM* Gmapping package (sudo apt install ros-kinetic-gmapping) to create a map of the environment with the robot's laser sensor.

- Run the *slam_gmapping* node, from the *gmapping* package, in parallel with your other nodes to have your robot mapping the environment with laser scans.
  Please read the documentation of the node carefully at : http://wiki.ros.org/gmapping

- Run ***rviz*** (*rosrun rviz rviz*), and add a map element to visualize your result (next slide)!

*SLAM=Simultaneous Localization and Mapping

RobotCraft 2022: Stage/ ROS Tutorial

Craft 6.2 (Simulating with ROS)

#Assignment for this Week – Part 3:

4. Nodes

4.1 slam_gmapping

The `slam_gmapping` node takes in sensor_msgs/LaserScan messages and builds a map (nav_msgs/OccupancyGrid). The map can be retrieved via a ROS topic or service.

4.1.1 Subscribed Topics

`tf` (tf/tfMessage)
Transforms necessary to relate frames for laser, base, and odometry (see below)

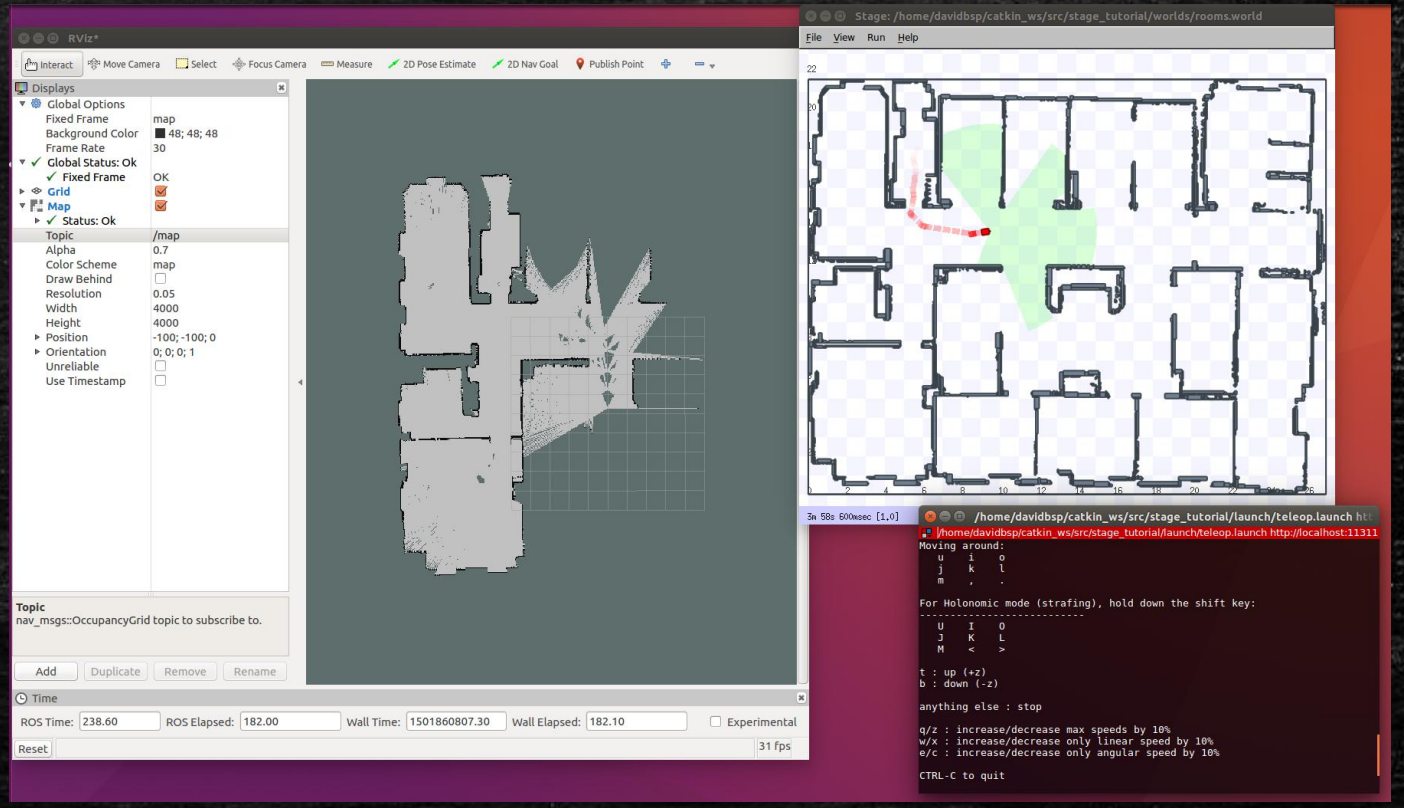
`scan` (sensor_msgs/LaserScan)
Laser scans to create the map from

4.1.2 Published Topics

`map_metadata` (nav_msgs/MapMetaData)
Get the map data from this topic, which is latched, and updated periodically.

`map` (nav_msgs/OccupancyGrid)
Get the map data from this topic

`~entropy` (std_msgs/Float64)
Estimate of the entropy of the distribution over the robot's pose (a higher value indicates greater uncertainty). New in 1.1.0.



RobotCraft 2022: Stage/ ROS Tutorial

Craft 6.2 (Simulating with ROS)

#Assignment for this Week – Final:

- Create a “reactive.launch” file to run Part 1 and Part 2 (stage + reactive navigation) in one command, using roslaunch.
- Create a “gmapping.launch” to run Part 3 (gmapping + rviz) in one command, using roslaunch.
- Record a screen **video**, while running the “reactive.launch” and the “gmapping.launch” file for at least 1 minute.
  For this, you can download gtk-recordmydesktop with: sudo apt install gtk-recordmydesktop.
- Add a “teleop.launch” file (stage + teleop) to also run a teleoperation node instead of your reactive walk with your robot on stage, using roslaunch.

RobotCraft 2022: Stage/ ROS Tutorial

Craft 6.2 (Simulating with ROS) Evaluation:

- Create the "*simstage_groupX*" ROS package (**5%**)
- Create stage robot with at least 3 blocks + World environment (**10%**)
  - *Extra for creativity in creating the robot* (**5%**)
  - *Extra for using a different occupancy grid* (**5%**)
- Implementation of the reactive navigation node (**30%**)
- Code quality (**15%**):
  - Internal documentation (**relevant** comments);
  - Structure;
  - Version Control (git);
  - Taking into account the cpp style guide (http://wiki.ros.org/CppStyleGuide)
- Launch Files (**15%**):
  - Working "reactive.launch" file (**5%**);
  - Working "*gmapping.launch" file* (**5%**);
  - Working "teleop.launch" file (**5%**);
- Screen recording showcasing a fully working reactive navigation and mapping system (**25%**)

RobotCraft 2022: Stage/ROS Tutorial

Week 5: Introduction to the Robot Operating System – Simulating with ROS

#Assignment for this Week – Submission:

- Deadline: Sunday, August 16th 2022, until 23:59
- Submission method: RobotCraft Moodle
- Submission file type: .zip
- Submission file contents:
  - Full ROS package (folder *simstage_groupX*);
  - Link to GitHub repository;
  - Video of the reactive navigation & mapping (.ogv or .mp4).

Conclusion

What did we do today? What now?

- We recapped the basic ROS concepts;
- We introduced the notion of simulation in ROS;
- We checked our ROS and Stage installations;
- We implemented a very basic robot controller;
- We were very good developers and kept everything source-controlled using git.

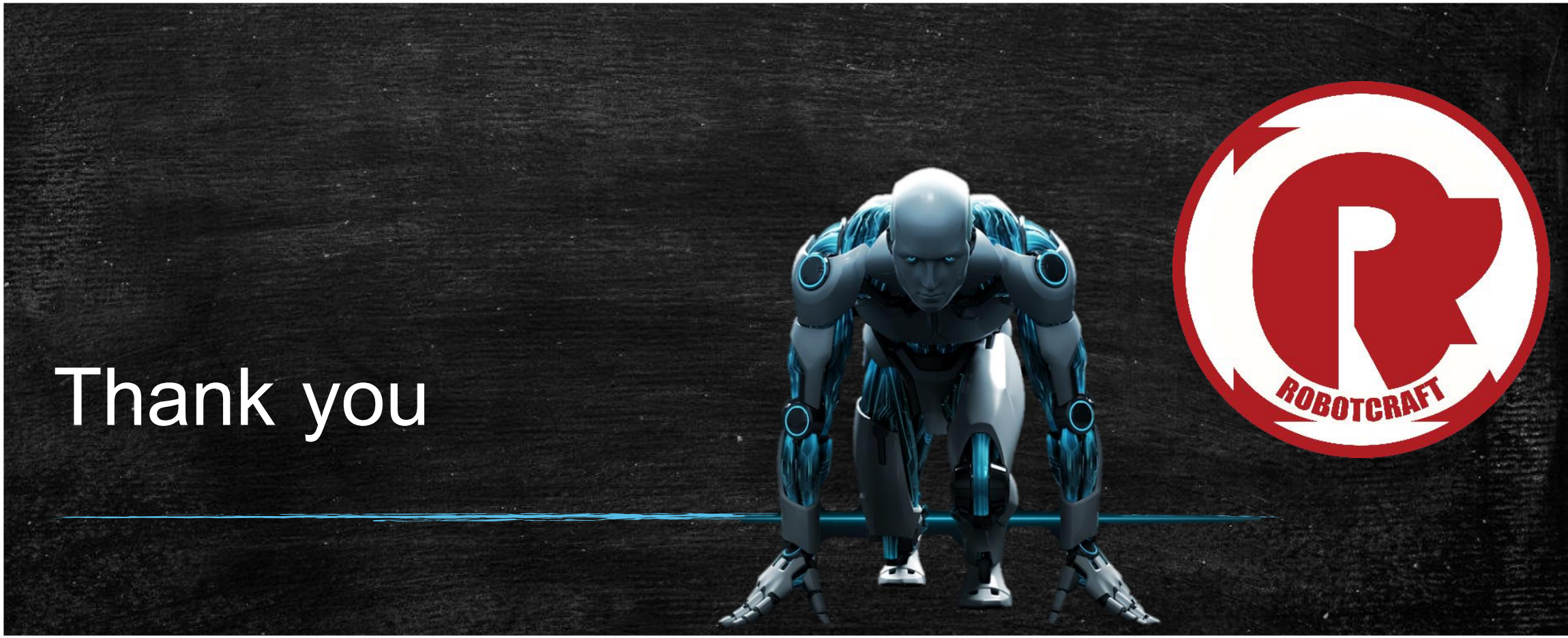
Additional Resources

- The ROS tutorials are excellent and cover most of the material:
  - http://wiki.ros.org/ROS/Tutorials
- The stage_ros package:
  - http://wiki.ros.org/stage_ros
- Stage documentation:
  - http://rtv.github.io/Stage/
- And also my own solutions, in case you need reference:
  - https://github.com/gondsm/robotcraft_turtlebot_control
  - https://github.com/gondsm/robotcraft_stage_tutorial
- Don’t forget to check the previous slides for links!

ROBOTCRAFT©

ROSin

INGENIARIUS



Craft #6

João Aguizo
joao@ingeniarius.pt

08/08/2022