ELEC-E8111 PROJECT WORK 2019

AUTONOMOUS MOBILE ROBOTS L

**ROBOT PROJECT INSTRUCTIONS**

**Login to the teleoperator computer:**

1. Choose an account corresponding to your group number. Accounts for this lab start with “AMR”.
2. All unused accounts have password:

1hailee

1. Change the password after you have found your group, otherwise someone else might login to your account and destroy all of your data.
	1. To change the password, type *passwd* in terminal and do as instructed.

**RELATIVELY EASY QUICK START GUIDE**

Notes:

* Always keep the power cable of the robot connected when not in use!
* Robot drains its battery quickly and it takes an hour to recharge.
* To reset the computer, hold down the small red button in front of the robot next to the small blue led.
* To reset the motor, press the red reset button on the lower left side of the robot.
* To test the wireless connection between the robot and teleoperator use command *ping*.
* When turning on the main power it takes some time for the laser scanner to initialize. Scanner is initialized when the green OK light is on.
* You can find more or less extensive ROS documentation at wiki.ros.org.
* Skills in Linux may prove useful.
* **Do not allow the robot to hit obstacles, the gears for the wheels can break. If the robot hits an obstacle and cannot be stopped, use the main power switch to stop it.**
* Ctrl+Shift+T opens a new tab in the terminal window.
* Tab completion can be very useful.
* You should not make any changes in the pioneer computer system.

IF YOU NEED ASSISTANCE, CONTACT:

Mika Vainio | Room 2568 | mika.vainio@aalto.fi | 0505052156 |

**1. INITIALIZE THE PIONEER ROBOT:**

1. Turn on the power from the main power switch (black arrow) on the lower left side of the robot.



Fig. 1. Motor reset and main power switch.

* 1. Do not press the reset button (white arrow). If you do, you will need to restart the base from the main switch.
1. To start the computer, press the small red button in front of the robot next to the small blue led.



Fig. 2. Computer start / reset.

1. In the teleoperation computer, open a terminal and make a SSH-connection from the teleoperator computer to the robot with the following command:

*ssh pioneer@192.168.1.110*

* 1. Password: pioneer19
1. Now we need to start roscore, laserscanner, camera, rosaria packages, etc. All will be done by executing the launch file of the robot:

*roslaunch pioneer pioneer.launch*

* 1. You can safely ignore error messages *[Error: [Walltime:...]* in this window.
1. Keep this window open and running: if you need to restart roscore, use the command Ctrl+C to stop the process and then relaunch the file from step 4.
2. Open a new tab/window and run the following command locally:

*rostopic pub /manip\_servo\_angles geometry\_msgs/Vector3 "{x: 0.0, y: 0.0, z: 30.0}"*

* 1. This will set the laser in horizontal position by publishing to the correct topic. Notice how the command is run locally, but the topic belongs to a node running on the robot. If the laser does not move, use Ctrl+C and then try re-running again the command.

**DRIVE MANUALLY:**

1. Disconnect the power cable from the robot.
2. Open a new terminal tab and start the manual drive script with the following command:

*rosrun teleop\_twist\_keyboard teleop\_twist\_keyboard.py*

1. Follow the instructions displayed on the screen.
	1. The given starting speed is very high so it is a good idea to lower it, for example to 0.1
2. Open a new terminal window and run the following command:

*rostopic echo /RosAria/cmd\_vel*

* 1. This will display data from the topic /RosAria/cmd\_vel, corresponding to low level control of the robot's wheels.

**RVIZ APPLICATION:**

Rviz is used to visualize the data the robot produces, such as camera and laser scanner feeds. When adding new displays, it is important to rename them exactly as instructed here. The topic names need to correspond to the ROS topics, otherwise the displays will not work.

1. Run the program:

*rosrun rviz rviz*

1. Change *Fixed Frame* listed on the left side of the screen under *Global Options* to *map*.
2. You can add new displays with *Add* button at the left bottom of the window. You can ignore any error/warning messages for now.
	1. Add *TF*
		1. *odom* frame corresponds to the initial position of the robot, right after it has been started.
	2. Add *LaserScan* and rename its topic to */scan*
		1. Make sure you change the topic name and not the display's name.
		2. Since the node providing the topic has already been started, you can select it from the drop down menu.
	3. Add *Map* and rename its topic to */map*
	4. Optionally add *Camera* and rename its topic to *camera/rgb/image\_rect\_color*
	5. Optionally add *PointCloud2* and rename its topic to *camera/depth/points* or *camera/depth\_registered/points*
		1. If you add it, run the following command as well in a new terminal to adjust the RGB-D's pose:

*rostopic pub /ptu\_servo\_angles geometry\_msgs/Vector3 “{x: 0.0, y: 105.0, z: 85.0}”*

**SLAM:**

SLAM generates an occupancy grid from the laser scan feed in real time. This process is run on the pioneer robot and you can monitor it with RVIZ.

1. In a new terminal tab/window, open a new ssh connection to robot

(*ssh* *pioneer@192.168.1.110**)*and run:

*rosrun gmapping slam\_gmapping*

1. All the errors should disappear and you should see all displays correctly.
2. If you experience decreased performance and data transmission for SLAM map you can disable the optional displays.
3. Drive around the robot manually to acquire as detailed map as you can.
	1. You can see the currently generated map in Rviz.
4. Save the current map as a .PGM image to your working folder by running (in a new terminal):

*rosrun map\_server map\_saver*

1. You can also use recorded data for slam.
	1. Start recording with: *rosbag record –O mylaserdata /scan /tf*
	2. Stop recording by terminating the program with Ctrl+C
	3. Play the bag contents with: *rosbag play --clock mylaserdata.bag*
	Hint: check out rqt\_bag

Fig. 3. The map should look something like this.

**PATH PLANNING:**

For this task there are two options:

**A.** Follow the guide and use the navigation stack as described next in order to implement a working path planner.

**B.** Implement/use a different path planner than described in this guide. A working path planner will reflect on your grade as discussed in the lecture. It is, however, recommended to follow this guide and implement a working path planner, after which you may try and implement a different one.

Until now you have used pre-installed packages in the ROS environment. Next, you will learn how to create your working space, “install” new packages and run them.

1. The latest ROS versions use a catkin workspace structure. First create your own catkin working space under your home directory.
	1. ROS wiki will help you in this: http://wiki.ros.org/catkin/Tutorials/create\_a\_workspace
2. Also, run *catkin\_make* to make sure you have correctly created your workspace.
3. Next, you will need to download all the required packages.
	1. First, go to:

<http://wiki.ros.org/move_base>

* + 1. From there, open the link from “Source” (to github) and download the branch called “indigo-devel”.

https://github.com/ros-planning/navigation/tree/indigo-devel

* + 1. Copy its contents to your */Home/catkin\_ws/src/*
		2. Run again *catkin\_make* from within your catkin workspace.
		3. Lastly, remember to source your newly created workspace:

*source devel/setup.bash*

* + 1. You will need to source your setup in every new terminal you open.
1. First, you will need to create the required launch files, which automate the starting and parametrization of the node. **Make sure the formatting in your files follow the one presented here (especially the spaces at the beginning of lines).** We will use *catkin\_create\_pkg* to create the folder (package) containing the launch and configuration files. Navigate to *catkin\_ws/src/* and run (**replace \* with your group number**):

*catkin\_create\_pkg group\* catkin roscpp move\_base*

1. Inside the package folder, create the files as described next.

(HINT: COPY TEXTS TO THESE FILES DIRECTLY FROM THIS WORD DOCUMENT BY USING FOR EXAMPLE GEDIT (OR PICO). YOU CAN CREATE THOSE IN ADVANCE WITH ANY TEXT EDITOR AND THEN JUST MOVE THEM TO THE YOUR FOLDER)

* 1. Create a file named *move\_base.launch* and fill it up with the following text.Note that you should replace bold words with the same name as used in the previous step.

*<launch>*

 *<!--- Run AMCL -->*

 *<include file="$(find* ***group\*****)/amcl.launch" />*

 *<node pkg="move\_base" type="move\_base" respawn="false" name="move\_base" output="screen">*

 *<remap from="/cmd\_vel" to="/RosAria/cmd\_vel" />*

 *<rosparam file="$(find* ***group\*****)/costmap\_common\_params.yaml" command="load" ns="global\_costmap" />*

 *<rosparam file="$(find* ***group\*****)/costmap\_common\_params.yaml" command="load" ns="local\_costmap" />*

 *<rosparam file="$(find* ***group\*****)/local\_costmap\_params.yaml" command="load" />*

 *<rosparam file="$(find* ***group\*****)/global\_costmap\_params.yaml" command="load" />*

 *<rosparam file="$(find* ***group\*****)/base\_local\_planner\_params.yaml" command="load" />*

 *<param name="base\_global\_planner" type="string" value="navfn/NavfnROS" />*

 *<param name="conservative\_reset\_dist" type="double" value="3.0" />*

 *<param name="controller\_frequency" type="double" value="10.0" />*

 *</node>*

*</launch>*

* 1. Create a file named *amcl.launch* and fill it up with:

<launch>

 <node pkg="amcl" type="amcl" name="amcl">

 <!-- Publish scans from best pose at a max of 10 Hz -->

 <param name="odom\_model\_type" value="diff"/>

 <param name="odom\_alpha5" value="0.1"/>

 <param name="transform\_tolerance" value="0.2" />

 <param name="gui\_publish\_rate" value="10.0"/>

 <param name="laser\_max\_beams" value="30"/>

 <param name="min\_particles" value="500"/>

 <param name="max\_particles" value="5000"/>

 <param name="kld\_err" value="0.05"/>

 <param name="kld\_z" value="0.99"/>

 <param name="odom\_alpha1" value="0.2"/>

 <param name="odom\_alpha2" value="0.2"/>

 <!-- translation std dev, m -->

 <param name="odom\_alpha3" value="0.8"/>

 <param name="odom\_alpha4" value="0.2"/>

 <param name="laser\_z\_hit" value="0.5"/>

 <param name="laser\_z\_short" value="0.05"/>

 <param name="laser\_z\_max" value="0.05"/>

 <param name="laser\_z\_rand" value="0.5"/>

 <param name="laser\_sigma\_hit" value="0.2"/>

 <param name="laser\_lambda\_short" value="0.1"/>

 <param name="laser\_lambda\_short" value="0.1"/>

 <param name="laser\_model\_type" value="likelihood\_field"/>

 <!-- <param name="laser\_model\_type" value="beam"/> -->

 <param name="laser\_likelihood\_max\_dist" value="2.0"/>

 <param name="update\_min\_d" value="0.2"/>

 <param name="update\_min\_a" value="0.5"/>

 <param name="odom\_frame\_id" value="odom"/>

 <param name="resample\_interval" value="1"/>

 <param name="transform\_tolerance" value="0.1"/>

 <param name="recovery\_alpha\_slow" value="0.0"/>

 <param name="recovery\_alpha\_fast" value="0.0"/>

 </node>

</launch>

* 1. Create a file named *base\_local\_planner\_params.yaml* and fill it with:

TrajectoryPlannerROS:

 max\_vel\_x: 0.3

 min\_vel\_x: 0.05

 max\_rotational\_vel: 0.4

 min\_in\_place\_rotational\_vel: 0.3

 escape\_vel: -0.1

 sim\_time: 2.0

 path\_distance\_bias: 0.6

 goal\_distance\_bias: 0.6

 acc\_lim\_th: 1.2

 acc\_lim\_x: 1.0

 acc\_lim\_y: 1.0

 holonomic\_robot: false

* 1. Create a file named *costmap\_common\_params.yaml* and fill it with:

*obstacle\_range: 2.5*

*raytrace\_range: 3.0*

*inflation\_radius: 0.35*

*#---standard pioneer footprint---*

*#---(in meters)---*

*footprint: [ [0.3302, -0.0508], [0.254, -0.0508], [0.254, -0.254], [-0.254, -0.254], [-0.254, 0.254], [0.254, 0.254], [0.254, 0.0508], [0.3302, 0.0508] ]*

*transform\_tolerance: 0.2*

*map\_type: costmap*

*observation\_sources: laser\_scan\_sensor*

*laser\_scan\_sensor: {sensor\_frame: laser, data\_type: LaserScan, topic: scan, marking: true, clearing: true, expected\_update\_rate: 0.2}*

* 1. Create a file named *global\_costmap\_params.yaml* and fill it with:

global\_costmap:

 global\_frame: /map

 robot\_base\_frame: base\_link

 update\_frequency: 2.0

 publish\_frequency: 10.0

 static\_map: true

* 1. Create a file named *local\_costmap\_params.yaml* and fill it with:

local\_costmap:

 global\_frame: /odom

 robot\_base\_frame: base\_link

 update\_frequency: 5.0

 publish\_frequency: 10.0

 static\_map: false

 rolling\_window: true

 width: 5.0

 height: 5.0

 resolution: 0.025

1. Navigate back to *catkin\_ws* and run *catkin\_make.*

Remember to source your setup: *source devel/setup.bash*

1. Launch your newly created files with:

*roslaunch group\* move\_base.launch*

1. Add the following displays in RViz:
	1. Add a new *Map* display and change its topic to

*/move\_base/global\_costmap/costmap*

* + 1. You can change the *Color Scheme* to *costmap* as well.
	1. Add a *Path* and change its topic to */move\_base/NavfnROS/plan.*
	2. Add an *Odometry* display and change its topic to */RosAria/pose.*
1. In a new terminal window/tab, run:

*rosrun rqt\_tf\_tree rqt\_tf\_tree*

* 1. This will present you the current transformation frames configuration tree of the robot. Use the *button* on the top-right to save it to your folder and include it in your report.

**MOTION CONTROL:**

The motion control introduced here is the base motion control. All path planning algorithms feed their waypoints to this motion control module. You can test this module via RVIZ interface.

1. Give first a *2D Pose Estimate* to the robot (initial pose of the robot at start-up) by using the RViz tab.
2. Now you can give a target pose for the robot by drawing a vector arrow in RViz using the tab *2D Nav Goal*. The robot should react immediately to the command. You should see the trajectory
3. If you run into problems, you can debug the process by testing the Twist connection with manual drive. You may also want to echo the nodes /move\_base\_simple/goal and RosAria/cmd\_vel to ensure messages are passed correctly.
4. Lastly, run the following command, which will present you with all the active nodes inside ROS:

*rosrun rqt\_graph rqt\_graph*

* 1. Save the image and include it in your report with a proper description. To save it as an image, use the “blank” button on the top right corner.

**Useful commands:**

 rostopic list  list all active nodes

 rosnode info /<name>  display node information

 rostopic echo /<name>  listen to node

 rospack find <name>  find out if a specific rospack exists