Autonomy for robotics systems

Dominika Bobik

This is a summary of my work done under Dr. Anthony Pinar's supervision during the Spring 2022 semester at Michigan Technological University. It contains valuable information about robotics systems for anyone new to the field.

Table of Contents

Introduction

Robotics is a new and exciting field that combines computer science, engineering, and much more to create machines that are capable of performing a series of complex tasks automatically. Nowadays, robots are widely used everywhere. In our homes, they vacuum the floor or mow the lawn. In the factories, they are used to produce and package our favorite candies. Finally, robots fly to outer space and help us discover the universe. Applications are countless and therefore, having the ability to build and program a robot might be a crucial skill in the years to come.

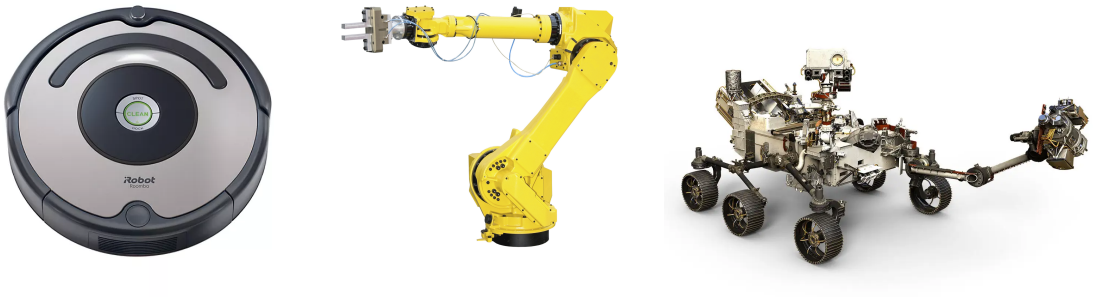


Figure 1. The abundance of robot types. From the left: iRobot Rumba - autonomous vacuum, a robotic arm used in production lines, Mars rover Perseverance

ROS

Two main components create a robot - hardware and software. In fact, the robot is nothing more than just a very specialized and sophisticated computer. The basic components are the same as a common laptop or PC.

Here is the open-source project from Jet Propulsion Laboratory with a step by step instructions on how to build a rover yourself:

https://github.com/nasa-jpl/open-source-rover

Just like a personal laptop, robots need an operating system. While Windows and iOS are great for business work, they don't provide the flexibility needed for robotics systems, where low-level system access and modifications are needed. Most robots use Linux Ubuntu as an operating system. It is a great open-source alternative to more popular cousins. There are other distributions (versions) of Linux available, but Ubuntu is widely supported by a large community of developers and is being continuously improved. There are a couple of ways you can work with Linux. You could flash it directly on your computer, use Virtual Machine, or use Windows Subsystem for Windows. See Appendix A for more details on each. Linux differs from the operating systems used for business work and therefore it could be beneficial for new users to go through introductory tutorials. The Construct has a great free tutorial targeted specifically for robotics.

ROS - Robots Operating System - is a set of libraries that abstract interaction with a robot, making it easier and more efficient. Quoting the documentation:

"*ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.*"

ROS has different distributions (versions) that are meant to work with Linux. Usually, each distribution is compatible with a different Linux version therefore it is crucial to double-check compatibility before proceeding with the installation. Details on installation can be found in Appendix B.

Table 1. Recent ROS distributions and Linux compatibility

ROS distribution	Linux Version
ROS Noetic Ninjemys	Ubuntu 20.04
ROS Melodic Morenia	Ubuntu 18.04
ROS Lunar Loggerhead	Ubuntu 17.04
ROS Kinetic Kame	Ubuntu 16.04

Initial ROS release was in 2007. Since then ROS 2 was brought to life in 2014, which improved and modernized the old infrastructure. It is not as widely used yet, as the industry adaptation takes time. ROS has better community support and there are many more problem-solving strategies available online. Due to that, guides presented in this report are targeted at ROS Noetic and Linux Ubuntu 20.04.

The main ROS resource is the wiki. It contains step-by-step tutorials and documentation, which are very useful.

NASA uses ROS for its Viper mission and many more:

https://www.nasa.gov/viper/lunar-operations

RViz, Gazebo

Two very useful and important tools while working with ROS are RViz and Gazebo.

Gazebo is a simulation tool that lets the user perform testing without a need to interact with the real robot which speeds up the debugging and development process. It needs to be installed once the ROS environment is set up and then integrated with existing packages. Details can be found here.

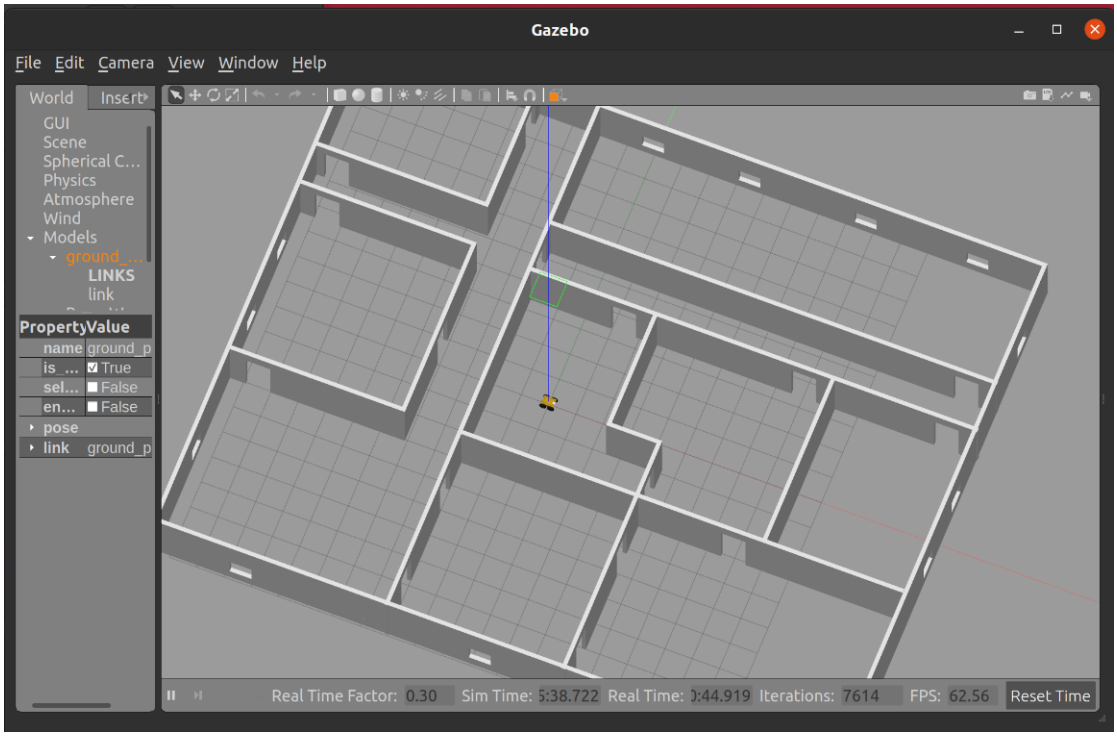


Figure 2. Gazebo simulation running

RViz is software that lets users visualize what a robot sees. Data coming from different sensors/cameras can be displayed and analyzed - the panel on the left is responsible for the data selection. As it is convenient to use the same RViz setup many times and configuration can be tedious, .rviz files can be created and saved for later use.

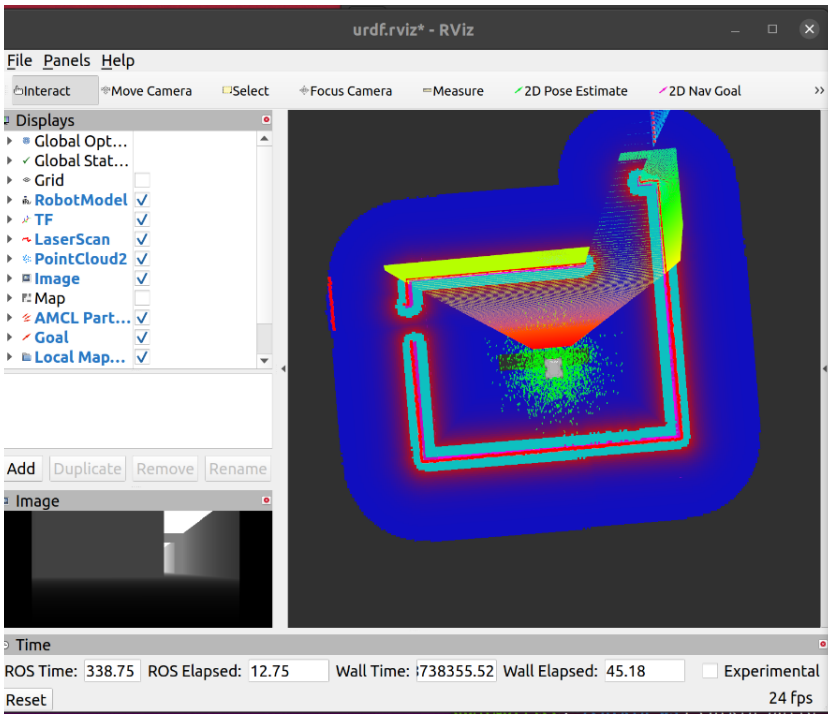


Figure 3. RViz window

Getting started

After the successful ROS installation, the best course of action is to follow the tutorials available in ROS wiki. They provide a detailed description of the concepts outlined below. Interaction with ROS starts with a catkin workspace. Catkin is a build system for ROS, which generates targets (libraries, executable programs, generated scripts, exported interfaces) from the source code. Workspace is a directory that, by default, contains four subdirectories: src, build, devel, and install. Source (src) space is where the packages "live".

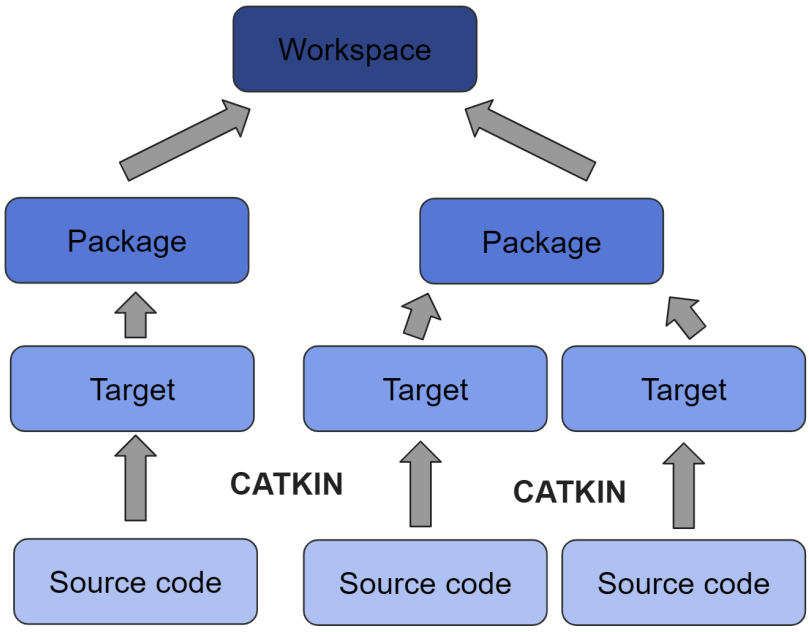


Figure 4. ROS filesystem organization

To perform a build of the source code, simply invoke the following command commands:

```
$ cd ~/catkin_ws
$ catkin_make
```

Those commands will automatically create build and devel folders inside the workspace. Build space is where the build is performed. Devel space is where targets are located before installation.

Each package consists of two configuration files package.xml and CMakeLists.txt as well as source files. Common file types used in packages:

- .launch files

Configuration files that specify nodes to launch, parameters to set, and information about the system. They allow for many nodes to be launched at the same time and use XML format. Placed in the "launch" directory within the package. Can be opened with roslaunch command.

- .msg files

Describe custom messages. Placed in the "msg" directory within the package.

- .srv files

Describe service - request and response parts. Placed in the "srv" directory within the package.

- .py/.cpp files

Custom scripts that specify the behavior of the system using the ROS libraries for hardware interaction. Placed in the “scripts” directory.

- .yaml files

Provide parameters for the system (how big the robot is, how fast it should move, etc.) that can be accessed by nodes. Collection of key-value pairs in data serialization language.

```
√ yoga ~/catkin_ws/src λ tree
.
├── beginner_tutorials
│   ├── CMakeLists.txt
│   ├── include
│   │   └── beginner_tutorials
│   ├── launch
│   │   └── turtlemimic.launch
│   ├── msg
│   │   └── Num.msg
│   ├── package.xml
│   ├── scripts
│   │   ├── add_two_ints_client.py
│   │   ├── add_two_ints_server.py
│   │   ├── listener.py
│   │   └── talker.py
│   ├── src
│   └── srv
│       └── AddTwoInts.srv
└── CMakeLists.txt -> /opt/ros/noetic/share/catkin/cmake/toplevel.cmake

8 directories, 10 files
√ yoga ~/catkin_ws/src λ
```

Figure 5. Overview of an src folder with a package in a catkin workspace

ROS setup consists of nodes that can publish/receive data by subscribing to the topics. Node is essentially an executable that uses the ROS library to communicate with other nodes in the network.

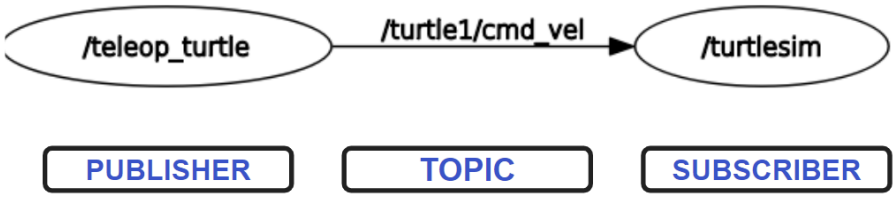


Figure 6. Interaction between the nodes

Rqt graph

As the number of nodes in a single ROS package is fairly large and grows quickly, debugging can become tedious and time-consuming. Rqt graph is a tool that visualizes all nodes and topics, providing a programmer with a clear picture of the setup.

Launching sequence when working with 4WD_description package (launch in two different terminals):

```
$ roslaunch 4WD_description gazebo.launch
$ rosrun rqt_graph rqt_graph
```

Warning! Please note, that a node will only appear on the graph if it is running - it will not display on failure.

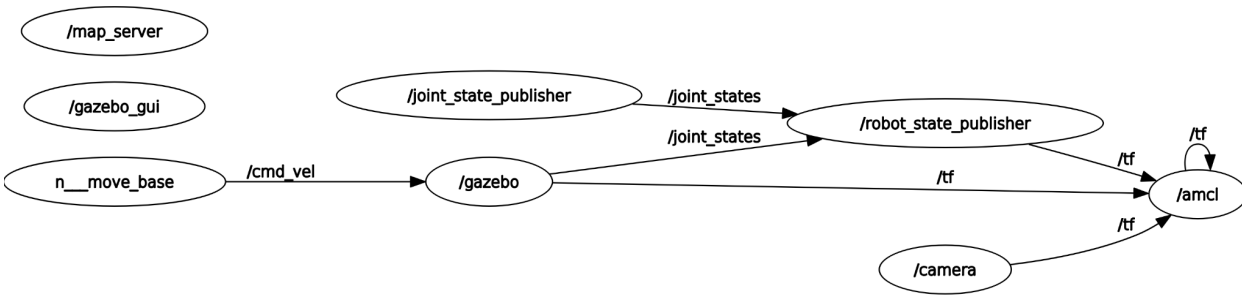


Figure 7. Rqt graph with nodes - bubbles depict nodes and arrows represent topics

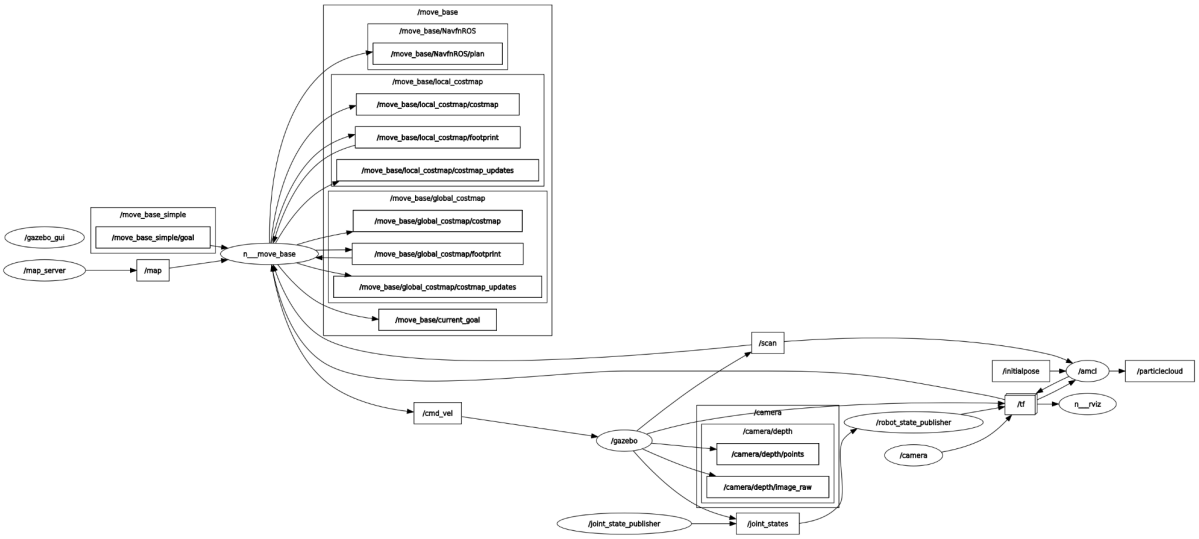


Figure 8. Rqt graph showing all nodes and topics for a 4WD_description package

Navigation Stack - How does the robot move?

We assume autonomous behavior against a known map. The robot locates itself in space using joint_state_publisher and robot_state_publisher. Those contain parameters like wheel size, diameter, overall height and length of a robot, etc. Robot_state_publisher publishes odometry data to the tf topic, where it is converted between the coordinate frames to allow for further processing in amcl. Amcl translates sensor data into a better explanation of the world around the robot and sends that data to the move_base. Move_base picks the best route and produces velocity commands (adjust speed, wheel turn, etc.) to point the robot in that direction.

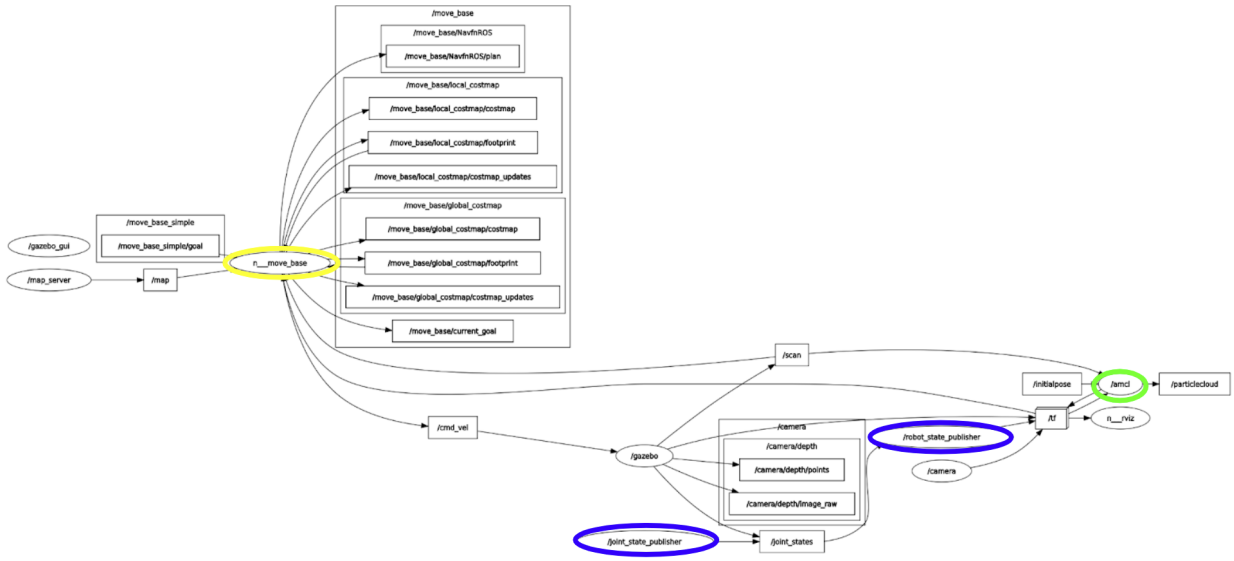


Figure 9. Joint_state_publisher and robot_state_publisher nodes are marked in blue, amcl marked in green, and move_base marked in yellow

Two important features are the 2D Pose Estimate and the 2D Nav goal at the top of rviz, which will be crucial when working with the navigation stack. Both are arrows that can be plotted on the map. The 2D Pose Estimate is an estimation of the robot's location. When moving around the known (previously mapped) environment, the robot needs a rough estimate of its initial position. It then adjusts it, however, knowledge of the starting point is crucial. The 2D Nav Goal allows the user to give the specific location where the robot should navigate. Many parameters of a navigation stack can be visualized in Rviz. Here is a brief description of the most important ones from wiki:

- Static Map

Displays the static map that is being served by the map_server if one exists.

- Particle Cloud

Displays the particle cloud used by the robot's localization system. The spread of the cloud represents the localization system's uncertainty about the robot's pose. A cloud that is very spread out reflects high uncertainty, while a condensed cloud represents low uncertainty.

- Obstacles

Display the obstacles that the navigation stack sees in its costmap. For the robot to avoid the collision, the robot footprint should never intersect with a cell that contains an obstacle.

- Inflated Obstacles

Displays obstacles in the navigation stack's costmap inflated by the inscribed radius of the robot. For the robot to avoid collision, the center point of the robot should never overlap with a cell that contains an inflated obstacle.

- Global Plan

Displays the portion of the global plan that the local planner is currently pursuing.

- Local Plan

Displays the trajectory associated with the velocity commands currently being commanded to the base by the local planner.

- Planner Plan

Displays the full plan for the robot computed by the global planner.

- Current Goal

Displays the goal pose that the navigation stack is attempting to achieve.

Appendix A - Linux Installation

Option 1. Installing Linux directly on a computer

The installation is fairly straightforward and there are many guides online.

Warning! Installing Linux on your machine will delete all the files and data from the previous operating system.

1. Download the desired Ubuntu version to your machine (.iso file) from here.
2. Install the software that will flash the Ubuntu image onto the USB. I recommend balenaEtcher, which can be downloaded from here.

Warning! Flashing the Ubuntu image onto the USB will delete all the files that are currently on the drive. Make sure to back them up before.

3. Plug the USB into your machine.
4. Change the boot order for your machine, by entering the BIOS settings and changing the boot configuration. Each manufacturer has a different key combination that allows users to enter BIOS, this information is widely available online.
5. Reboot your computer. This will cause the computer to boot from the USB and start Linux installation. Follow the instructions on the screen to set up your new operating system.

If you would like to preserve the Windows environment you can partition your disk and install Linux alongside Windows. Then on each boot, you will be prompted to choose which OS you would like to use. Here is more information on how to create a partition and install Linux alongside Windows. I recommend creating a partition beforehand and choosing to install Linux on this free space manually instead of creating it on a fly during installation - this is for safety reasons.

Option 2. Setting up Virtual Machine (VM)

Virtual Machine lets you emulate a computer system that provides the functionality of a physical machine. A common, free VM solution is a Oracle VirtualBox.

1. Install software

Installing VirtualBox guide

2. Set up a virtual machine with Linux

Setting up a Linux virtual machine guide

Common problem: VM screen is really small and doesn't resize - sample solution

Appendix B - ROS installation

Installation steps have been outlined in detail here. Ubuntu needs permission to allow an installation of open-source software (like ROS). Make sure that option is checked in the settings. Source:https://railsblogs.rohityadav.in/2019/06/e-package-aptitude-has-no-installation.html

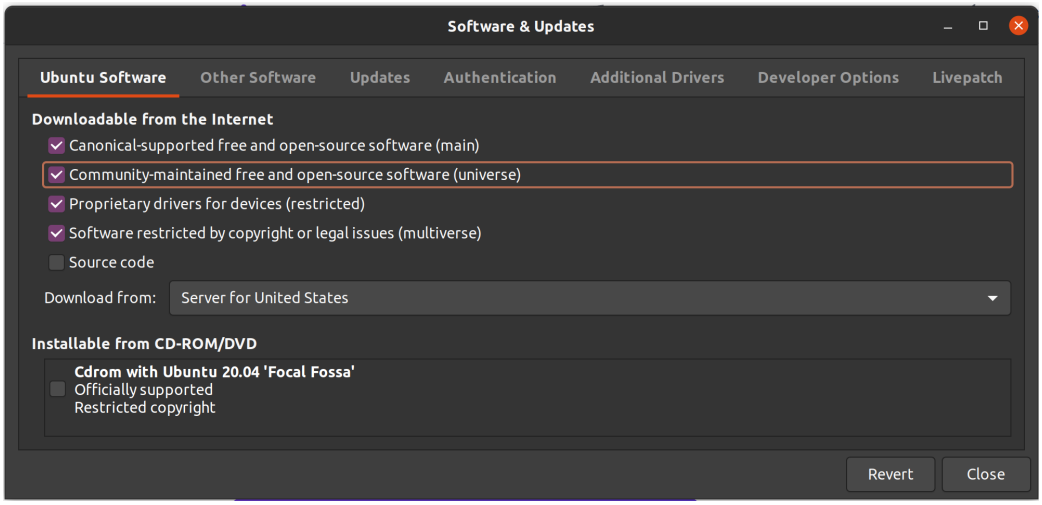


Figure 10. Settings that allow for an open-source software installation.

Appendix C - Modifying bashrc/zshrc file to source workspace automatically

A lot of work with ROS is done through the terminal. There are two most popular terminals used with Linux - the default bash and zsh that can be downloaded and installed. Bashrc/zshrc files are executed when the user logs in and provide a configuration for each terminal session. ROS requires some commands to be executed (like source) each time a new terminal is open and many times multiple terminals are needed while working. Therefore, it would be beneficial to automate repetitive tasks. To modify bashrc/zshrc file to include the source command do the following:

In the bash terminal type the following command:

```
$ nano .bashrc
```

If using zsh:

```
$ nano .zshrc
```

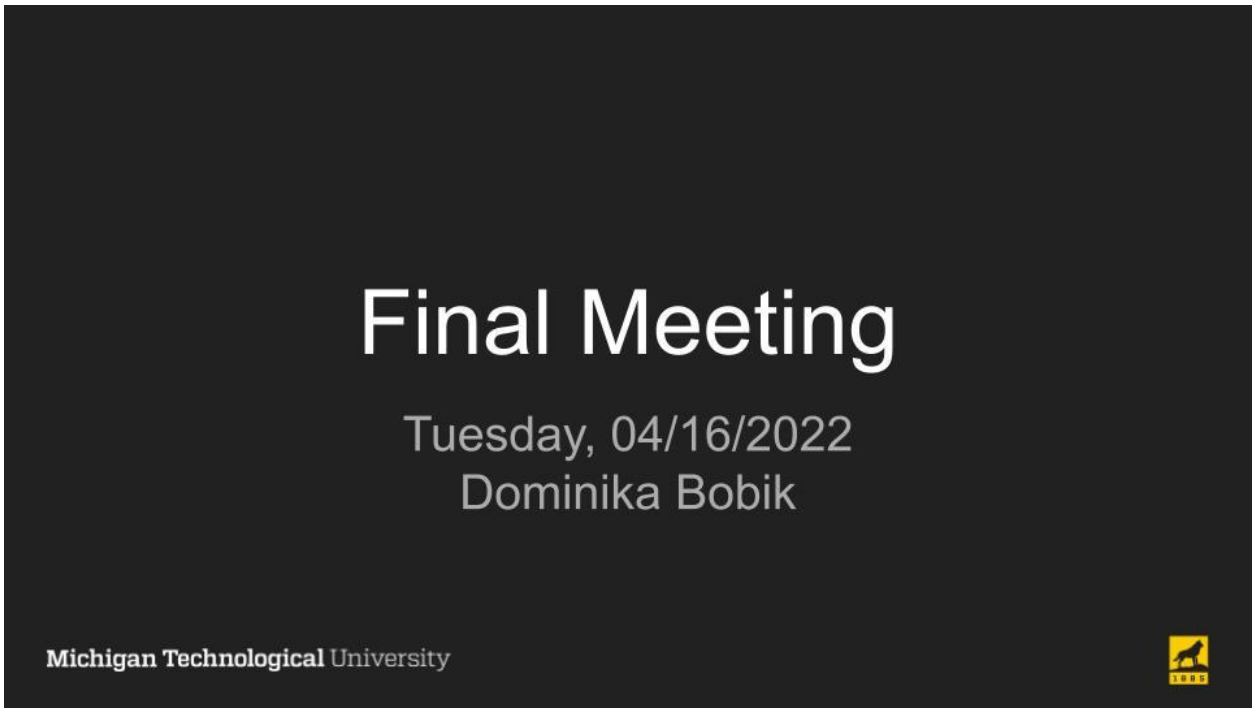
This command will open your .rc file for editing. Include the source command followed by a path to your setup.zsh/setup.bash file anywhere in the file. Here is what the sample line could look like if using zsh:

```
source /home/domi/catkin_ws/devel/setup.zsh
```

You can add as many commands as you would like including exports etc. Once the desired commands have been added save the file (CTRL+s) and exit the editor (CTRL+x).

Appendix D - Final Presentation

Final presentation can be seen using this link. Slides are attached below.



Timeline

- Meeting 1 - 2/22

ROS tutorials

- Meeting 2 - 3/1

ROS tutorials, Gazebo

- Meeting 3 - 3/15

Advanced ROS functionality

- Meeting 4 - 3/22

Remote connection, Gazebo with turtlebot

- Meeting 5 - 3/31

Autonomous behavior with 4WD_description package, Rosbag files

- Meeting 6 - 4/12

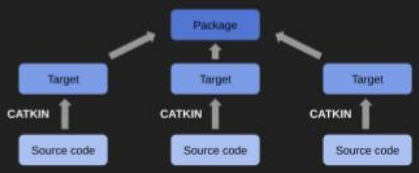
AMCL, Navigation Stack, RViz

ROS tutorials

- ROS installation (Ubuntu 20.04 + ROS Noetic)
- Introduction to Catkin

Catkin is a build system for ROS, which generates targets from the source code.

- ROS setup consists of nodes that can publish/receive data by subscribing to the topics



ROS tutorials cont., Gazebo

- Launch files - roslaunch

Allow many nodes to be launched at the same time

Specify behavior of separate nodes

- Msg files

Describe custom messages

- Srv files

Describe service - request and response parts

- Source setup.zsh/setup.bash for each new terminal

Source command can be added to the bashrc/zshrc file

.msg

```
int64 num
```

.srv

```
int64 a
int 64 b
- - -
int64  num
```


Catkin workspace

```
/ yoga ~/catkin_ws/src λ tree
.
├── beginner_tutorials
│   ├── CMakeLists.txt
│   ├── include
│   │   └── beginner_tutorials
│   ├── launch
│   │   └── turtlemimic.launch
│   ├── msg
│   │   └── Num.msg
│   ├── package.xml
│   ├── scripts
│   │   ├── add_two_ints_client.py
│   │   ├── add_two_ints_server.py
│   │   ├── listener.py
│   │   └── talker.py
│   ├── src
│   └── srv
│       └── AddTwoInts.srv
└── CMakeLists.txt -> /opt/ros/noetic/share/catkin/cmake/toplevel.cmake

8 directories, 10 files
/ yoga ~/catkin_ws/src λ
```


Advanced ROS functionality

- Creating ROS package

```
catkin_create_pkg <package_name>
```

- Including C++ class in Python
- Bundling ROS project into Snap (https://snapcraft.io/)

Remote connection

- Running ROS on multiple machines and communicating between them

Start master (hal)
- ➔ `ssh hal`
- ➔ roscore
- ➔ `export ROS_MASTER_URI=http://hal:11311`
- ➔ Rosrun [package_name] [node_name]

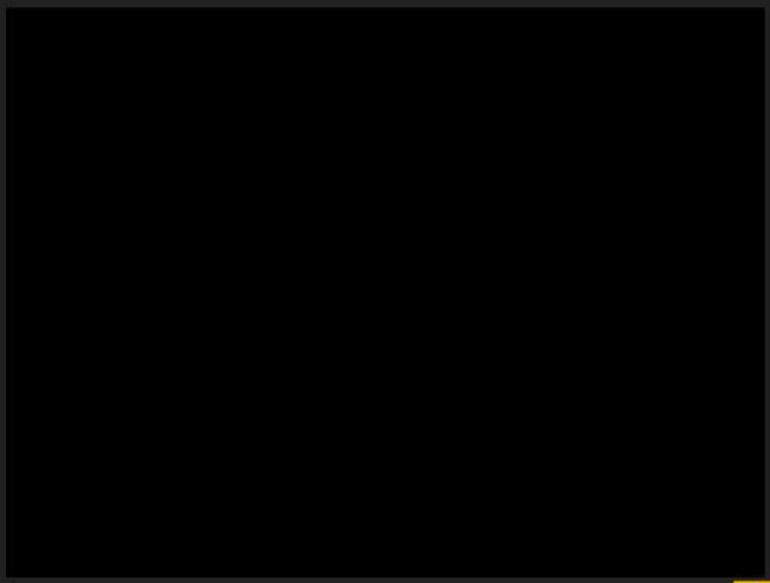
Start other machine (marvin)
- ➔ ssh marvin
- ➔ `export ROS_MASTER_URI=http://hal:1131 1`
- ➔ Rosrun [package_name] [node_name]

Remote connection

```
✓ yoga ~ λ ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 383  bytes 32667 (32.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 383  bytes 32667 (32.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp0s20f3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.226  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fd3b:334d:a511:1:a61e:670d:9e13:2ece  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::3c8a:c19b:6cf9:c992  prefixlen 64  scopeid 0x20<link>
        inet6 fd3b:334d:a511:1:8da:65d2:4a3f:296e  prefixlen 64  scopeid 0x0<global>
        ether 58:96:1d:62:74:88  txqueuelen 1000  (Ethernet)
        RX packets 5915  bytes 7714542 (7.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1570  bytes 306138 (306.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
 GNU nano 4.8                    /home/domi/.zshrc                    Modified
# Custom styling for the prompt.
source ~/.zsh-prompt
export EDITOR='nano -w'
source /home/domi/catkin_ws/devel/setup.zsh
export ROS_MASTER_URI=http://192.168.0.184:11311
export ROS_HOSTNAME=192.168.0.226

# Aliases
alias ls="ls -tr"
alias la="ls -A"
alias ll="ls -alF"
alias open="xdg-open"
alias encrypt='gpg -c --no-symkey-cache'
alias decrypt='gpg --no-symkey-cache'
alias c='xclip -selection clipboard'
alias v='xclip -o'

^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
```


Gazebo - turtlebot



Basic autonomous behavior

- Setup the robot model in gazebo

Make sure robot knows where it is: joint_states and robot_state_publisher packages - publish data to the move_base

- Configure sensors

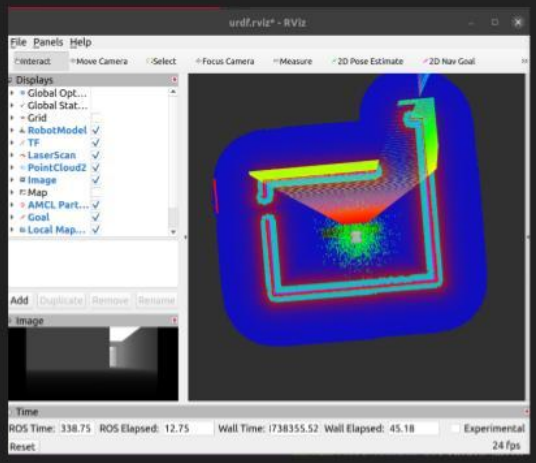
Scans of the surroundings can be published at a specific rate

- Configure navigation stack

"A 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base."

RViz

- Tool that lets developer visualize what robot sees
- Configuration can be saved to .rviz file and used multiple times



Michigan Technological University

Yaml files

- Data serialization language
- Collection of key-value pairs
- Provide parameters for the system (how big the robot is, how fast it should move, etc.) that can be accessed by nodes

```yaml
TrajectoryPlannerROS:

# Robot Configuration Parameters
  max_vel_x: 0.14
  min_vel_x: -0.56

  max_vel_theta:  40000.0
  min_vel_theta: -40000.0
  min_in_place_vel_theta: 40000.0

  acc_lim_x: 0.9
  acc_lim_y: 0.0
  acc_lim_theta: 40000.0

# Goal Tolerance Parameters
  xy_goal_tolerance: 0.3
  yaw_goal_tolerance: 0.25

# Differential-drive robot configuration
  holonomic_robot: false

# Forward Simulation Parameters
  sim_time: 4.5
  vx_samples: 20
  vtheta_samples: 50
  sim_granularity: 0.025
```

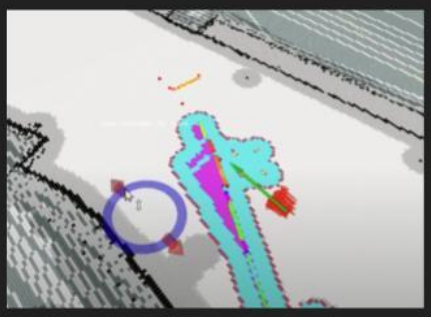

Launch files

- Configuration files that specify nodes to launch, parameters to set and information about the system
- Use XML format
- Placed in the “Launch” directory within the package
- Opened with roslaunch command

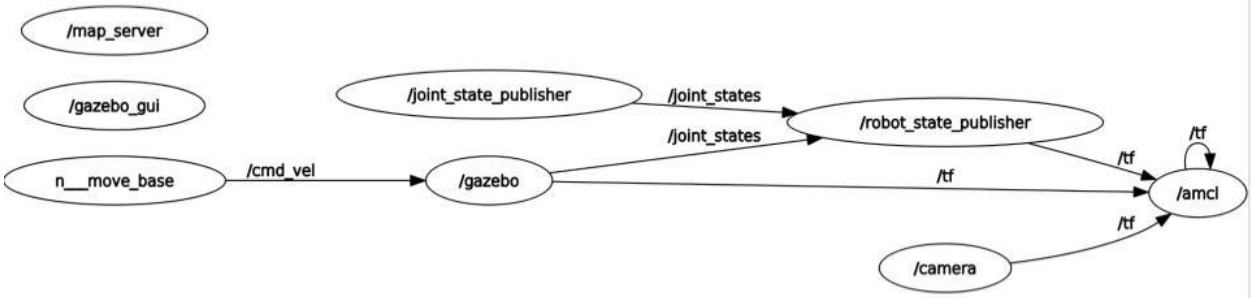
```xml
<launch>

  <group ns="turtlesim1">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <group ns="turtlesim2">
    <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
  </group>

  <node pkg="turtlesim" name="mimic" type="mimic">
    <remap from="input" to="turtlesim1/turtle1"/>
    <remap from="output" to="turtlesim2/turtle1"/>
  </node>

</launch>
```


AMCL

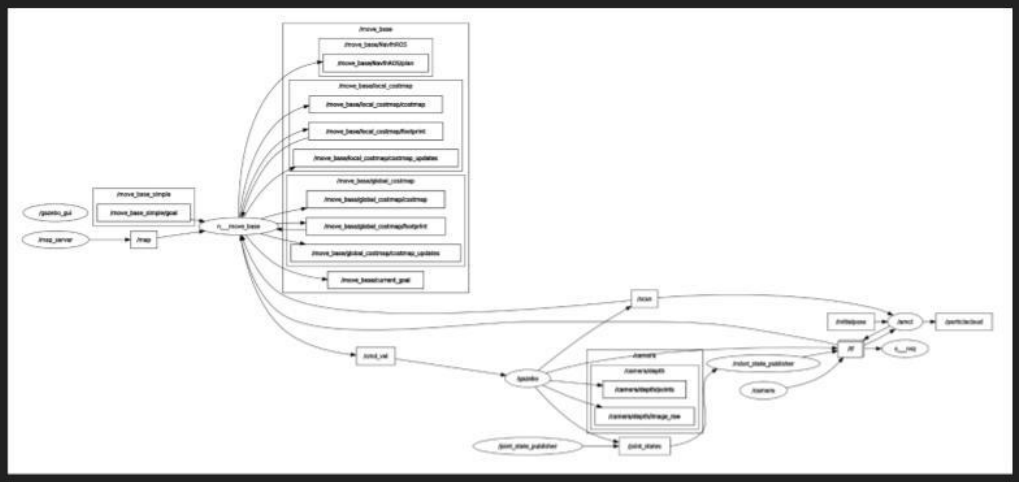
- Localizes robot in space
- Takes data from sensors and relates them to the pre-existing map
- Robot's behavior depends greatly on correct setup of sensors and amcl parameters that create odometry and laser models



Rqt Graph - Nodes only



Rqt Graph - Nodes + Topics



Thank you!