Lecture 1 | ROS Basics

Introduction

▼ **What is ROS?**

> 💡 ROS is Robot Operating System 🤪

Is it an operating system?

Why ROS?

To answer this question, we have to ask the following question:

▼ **How would you write a code to read data from multiple sensors attached to your system and using this info control the actuators attached to another SBC?**

```
# On main system
while True:
  img = realsense.read()
  points = lidar.read()
  orientation = imu.read()
  control = some_processing(img, points, orientation)
  send_to_SBC(control)  # Possibly via socket

# On SBC
while True:
  control = get_control_from_socket()
  apply_control_to_motors(control)
```

> 💡 What are the points of failures here?

Formal Definition

> ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some respects to 'robot

> frameworks' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.

Philosophy

- `Peer-To-Peer`
  Individual programs are nodes that communicate over a standard API
- `Distributed`
  Programs can be run on multiple computers and communicate over network
- `Multi-Lingual`
  Python, Java, C++
- `Light Weight`
  The core ROS layer is light weight
- `Open Source`
  Yaaay!

Basic Structure

Master

- Consider it as a server
- Every node registers at startup with the master
- Manages communication between different nodes

```
# Starting a ROS master
roscore
```

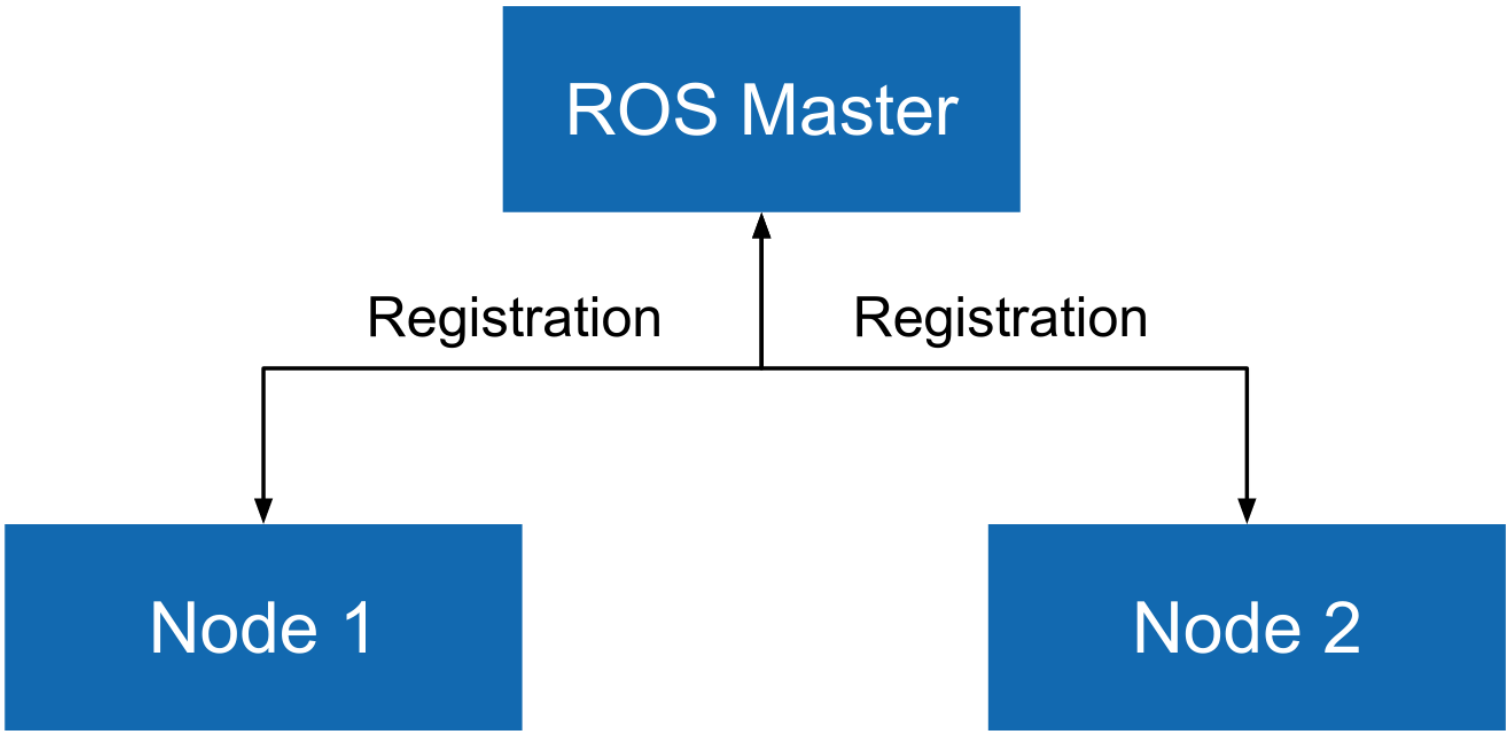
Nodes

- Consider node as client that connect to server (Master)
- Stand alone executable programs
- Individually compiled, executed, and managed
- Generally part of a package

```
# Starting a node that is part of a package
rosrun <package_name> <node_name>

# Listing all running nodes
rosnode list

# Get info about a node
rosnode info <node_name>
```



Topics

- Nodes communicate over topics
  - Nodes can *publish* or *subscribe* to a topic
  - Typically 1 publisher and n subscribers

```
# List all active topics
rostopic list

# Subscribe and print contents of a topic
rostopic echo /topic

# Show information about a topic
rostopic info /topic
```

Messages

- Data structures that define how nodes will communicate over a topic
- They are analogous to *'variable type'* in programming language. (And more!)
- ROS comes with it own set of standard messages
- You can define your own in *.msg files

```
# See type of a topic
rostopic type /topic

# Publish a message to a topic
rostopic pub /topic type data
```

- Some common message types:
  - Point
  - Image
  - Pose
  - Twist
  - Pointcloud

geometry_msgs/Point.msg

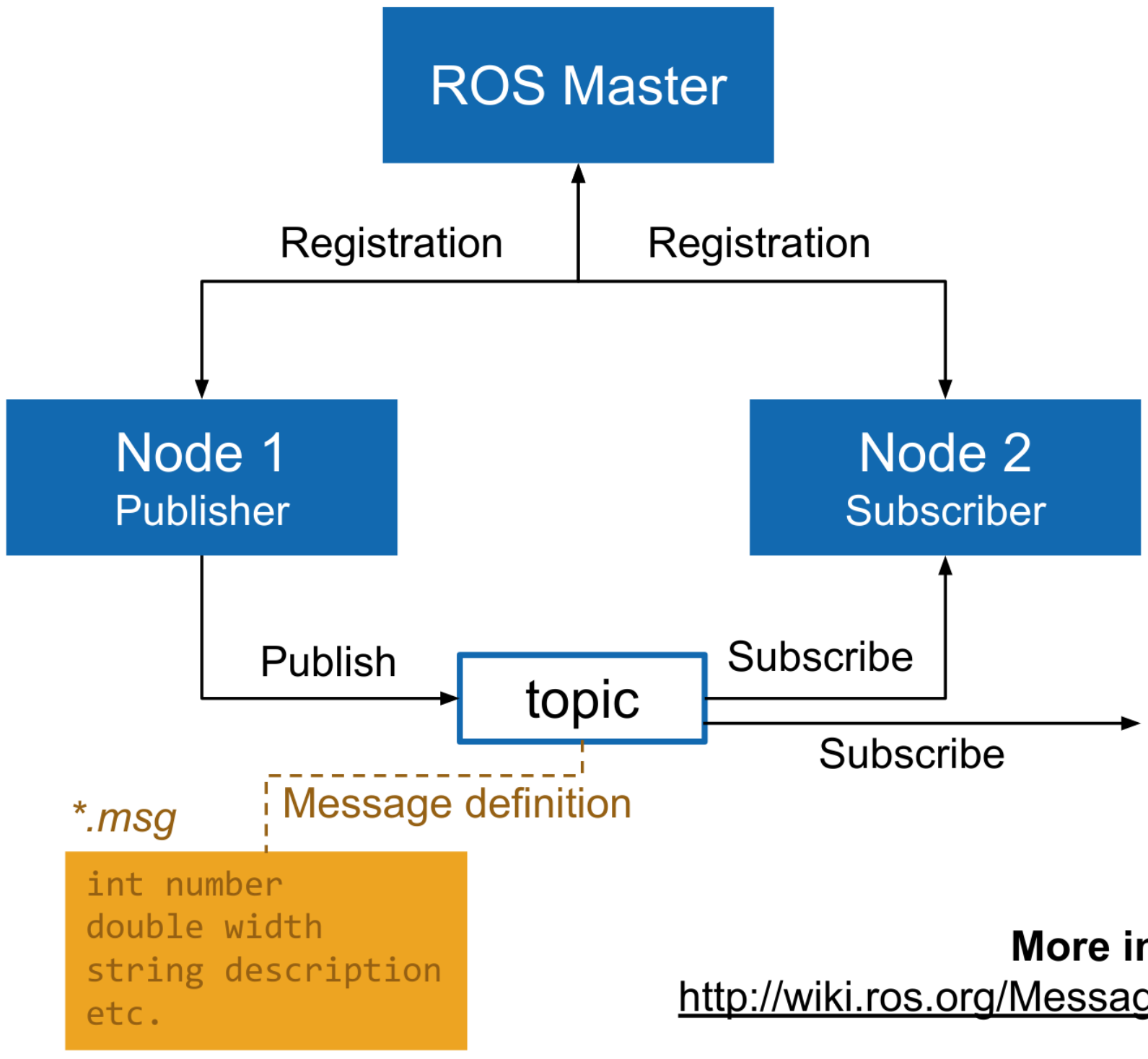
```
float64 x
float64 y
float64 z
```

sensor_msgs/Image.msg

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header
 uint32 seq
 time stamp
 string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```



More info
http://wiki.ros.org/Messages

Catkin Build System

- catkin is the ROS build system to generate executables, libraries, and interfaces.
- All your ROS packages will be in a catkin workspace and built using catkin

```
# Create a direcotry that will be your catkin workspace
mkdir ~/catkin_ws && cd ~/catkin_ws

# Make the required directories
mkdir src build devel
```

- Workspace Structure:

catkin Build System

The catkin workspace contains the following spaces

Work here

The *source space* contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

Don't touch

The *build space* is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

Don't touch

The *development (devel) space* is where built targets are placed (prior to being installed).

- Usage:

```
cd ~/catkin_ws

catkin_make

# This will overlay the workspace in your current shell.
source devel/setup.bash # or setup.zsh
```

Package

```
catkin_create_pkg <package_name> {dependencies}

# Example
catkin_create_pkg rrc_tutorial rospy roscpp std_msgs
```

- The package.xml file defines the properties of the package

package.xml

```xml
<?xml version="1.0"?>
<package format="2">
   <name>ros_package_template</name>
   <version>0.1.0</version>
   <description>A ROS package that...</description>
   <maintainer email="tlankhorst@ethz.ch">Tom Lankhorst</maintainer>
   <license>BSD</license>
   <url type="website">https://github.com/leggedrobotics/ros_...</url>
   <author email="tlankhorst@ethz.ch">Tom Lankhorst</author>

   <buildtool_depend>catkin</buildtool_depend>

   <depend>roscpp</depend>
   <depend>std_msgs</depend>

   <build_depend>message_generation</build_depend>
</package>
```

ROS Launch

- launch is a tool for launching multiple nodes (as well as setting parameters)
- Are written in XML as *.launch files

```
# When you are inside the folder containing the launch file
roslaunch <file_name.launch>

# If you have already build the package and overlayed the workspace
roslaunch <package_name> <file_name.launch>
```

- File structure

```xml
<launch>
  <node name="listener" pkg="rrc_tutorial" type="listener.py" output="screen"/>
  <node name="talker" pkg="rrc_tutorial" type="talker.py" output="screen"/>
</launch>
```

- **`launch`**: Root element of the launch file
- **`node`**: Each *<node>* tag specifies a node to be launched
- **`name`**: Name of the node (free to choose)
- **`pkg`**: Package containing the node
- **`type`**: Type of the node, there must be a corresponding executable with the same name
- **`output`**: Specifies where to output log messages (`screen`: console, `log`: log file)

Parameter Server

- Nodes use the parameter server to store and retrieve parameters at runtime
- Parameters can be defined in launch files or separate YAML files

```
# List all pramaeters
rosparam list

# Get the value of a particular parameter
rosparam get <parameter_name>
```

```
# Set a parameter
rosparam set <parameter_name> <value>
```

Code Walk-Through

- Code is available on GitHub

Advanced Stuff

RQT

- User-Interface based on QT
- Lots of tools exist
- Custom tools can be created

```
# Visualize graph of nodes and topics
rosrun rqt_graph rqt_graph

# Display and filter ROS messages
rosrun rqt_console rqt_console
```

- Explore:
  - rqt_logger_level
  - rqt_multiplot
  - rqt_gui

RVIZ

- 3D visualization tool for ROS
- Subscribes to topics and visualizes the message contents
- Save and load setup as RViz configuration

```
# Run Rviz
rviz

# Alternate
rosrun rviz rviz
```

Bag Files

```
# Record topics
rosbag record -O <bag file name> <topic names>

# Play recorded bags
rosbag play <name of bag file> --topics /topic1 /topic2 /topic3
```

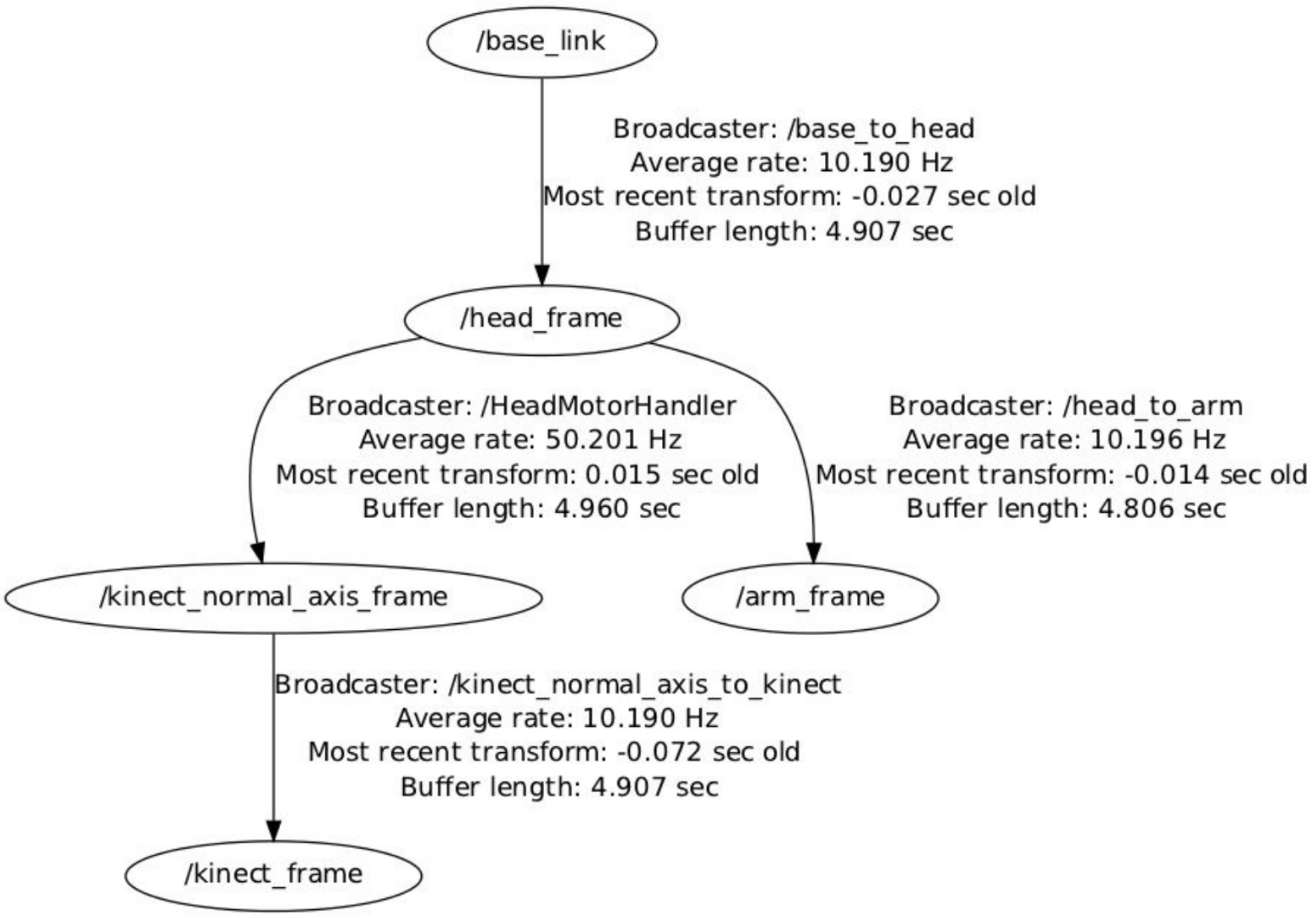
Transformations

- `TF Transformation System` is a tool for keeping track of coordinate frames over time
- Maintains relationship between coordinate frames in a tree structure buffered in time
- Lets the user transform points, vectors, etc. between coordinate frames at desired time
- Implemented as publisher/subscriber model on the topics `/tf` and `/tf_static`

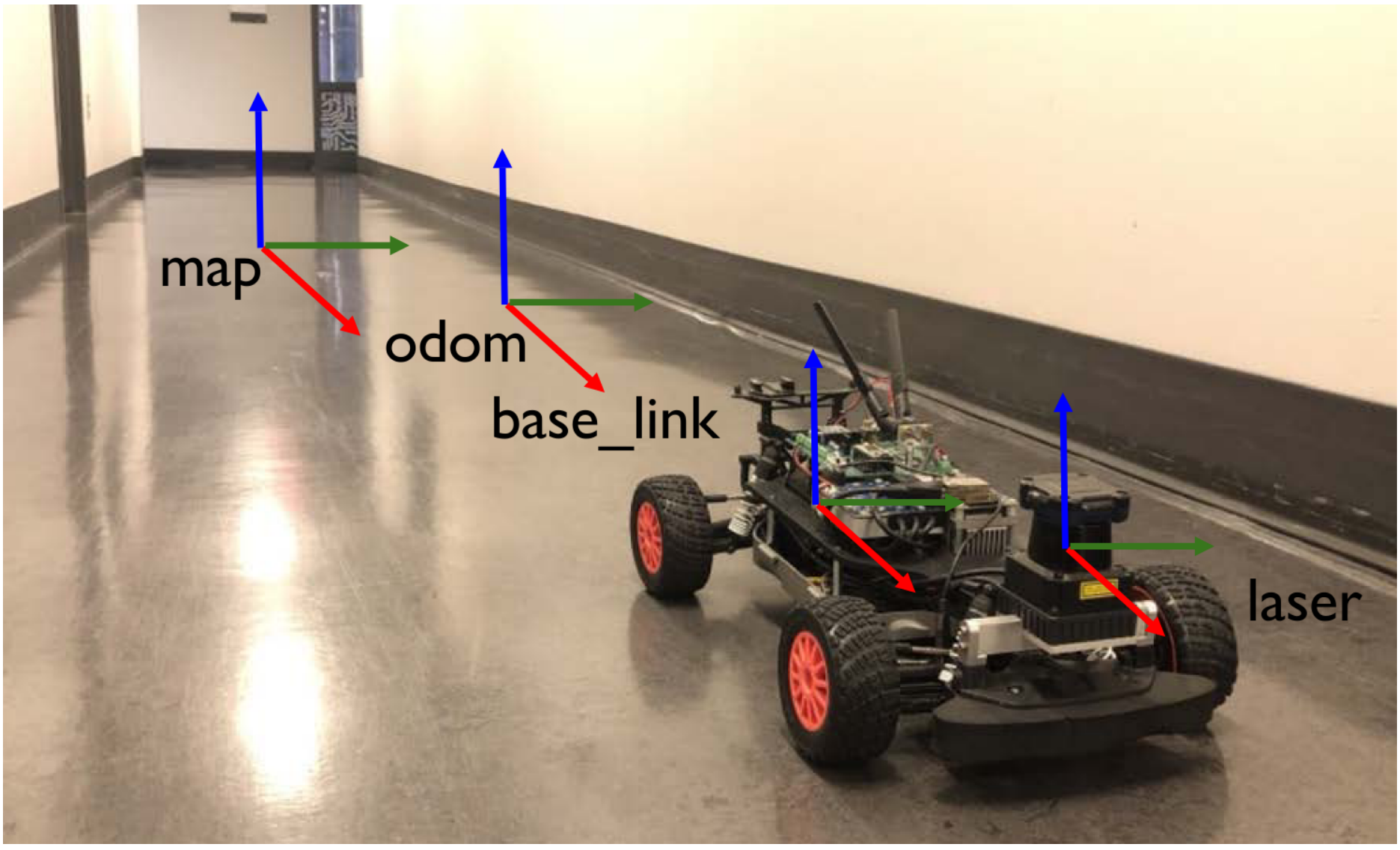
```
# Prints information about the transform tree at the moment
rosrun tf tf_monitor
```

```
# Visulaize the TF Tree
rosrun rqt_tf_tree rqt_tf_tree
```

- An example of a TF Tree:



- Some common frames: `map`, `odom`, `base_link`



- `map`: Represents the environment in which the robot works
- `odom`: Represents the starting point of the robot
- `base_link`: Represents the frame that is rigidly attached to the body of robot.
  - All sensors transformations are then calculated with respect to the `base_link` frame

Installation

References

- Most of the content is from ETH ROS Course: https://rsl.ethz.ch/education-students/lectures/ros.html
  - Its a 5 part course. I recommend doing it.
- A nice blog: https://towardsdatascience.com/what-why-and-how-of-ros-b2f5ea8be0f3
- A good book: https://cse.sc.edu/~jokane/agitr/