ROS Tutorial & Navigation

CS/EE/ME 134

Richard Cheng

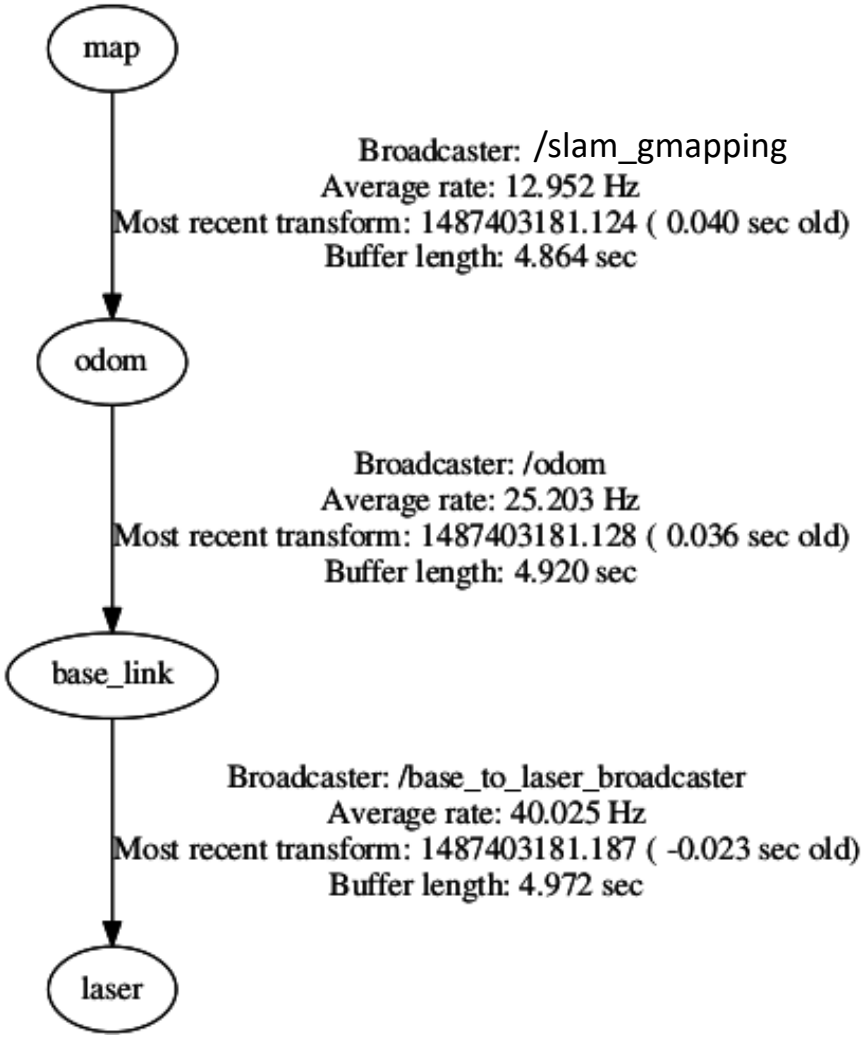
04/25/18

Outline

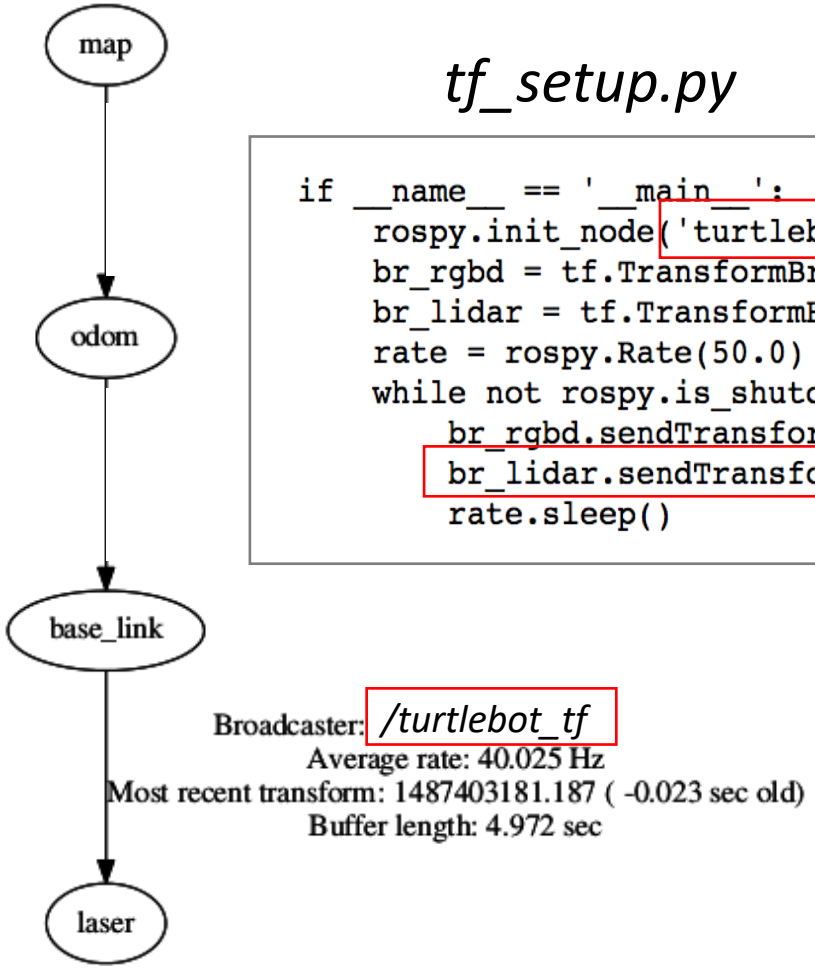
- *Transform Trees + Nodes/Topics/Messages*
- *Robot Navigation (Navigation Stack)*
- *Sending Messages*
- *Questions*

Defining Robots in ROS

- We define our system using a *tf structure,* which defines the parts of our robot(s) and the relation between them
- Important landmarks and parts of the robot are encoded in the *tf tree*
- Transforms are ***broadcasted*** by a specific **node**
- Give us **relative positioning** between frames
- Relations can be fixed or variable



Transform Tree in ROS



tf_setup.py

```
if __name__ == '__main__':
    rospy.init_node('turtlebot_tf')
    br_rgbd = tf.TransformBroadcaster()
    br_lidar = tf.TransformBroadcaster()
    rate = rospy.Rate(50.0)
    while not rospy.is_shutdown():
        br_rgbd.sendTransform((0.0, 0.0, 0.18), (0.5, 0.5, 0.5, 0.5), rospy.Time.now(), "rgbd", "base_link")
        br_lidar.sendTransform((0.06, 0.0, 0.23), (0.0, 0.0, 0.0, 1.0), rospy.Time.now(), "laser", "base_link")
        rate.sleep()
```

Nodes/Topics in ROS

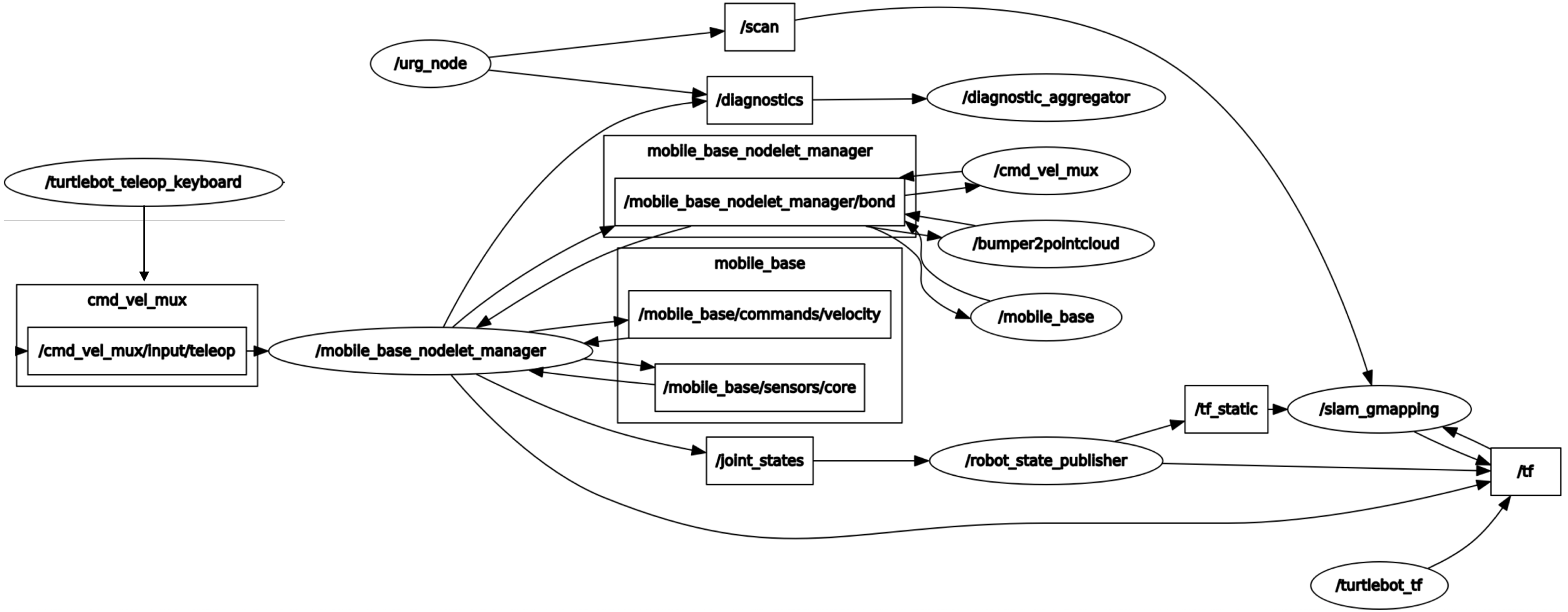
- Using the *tf* tree, we can encode position/orientation of all important parts of our robot(s)
- How do we get different components of our system to communicate with each other?
  (e.g. LIDAR sensor, Optitrack sensors, Turtlebot actuators)

Nodes and Topics

- *Nodes* represent separate processes that can communicate with each other by passing *messages* across *topics*

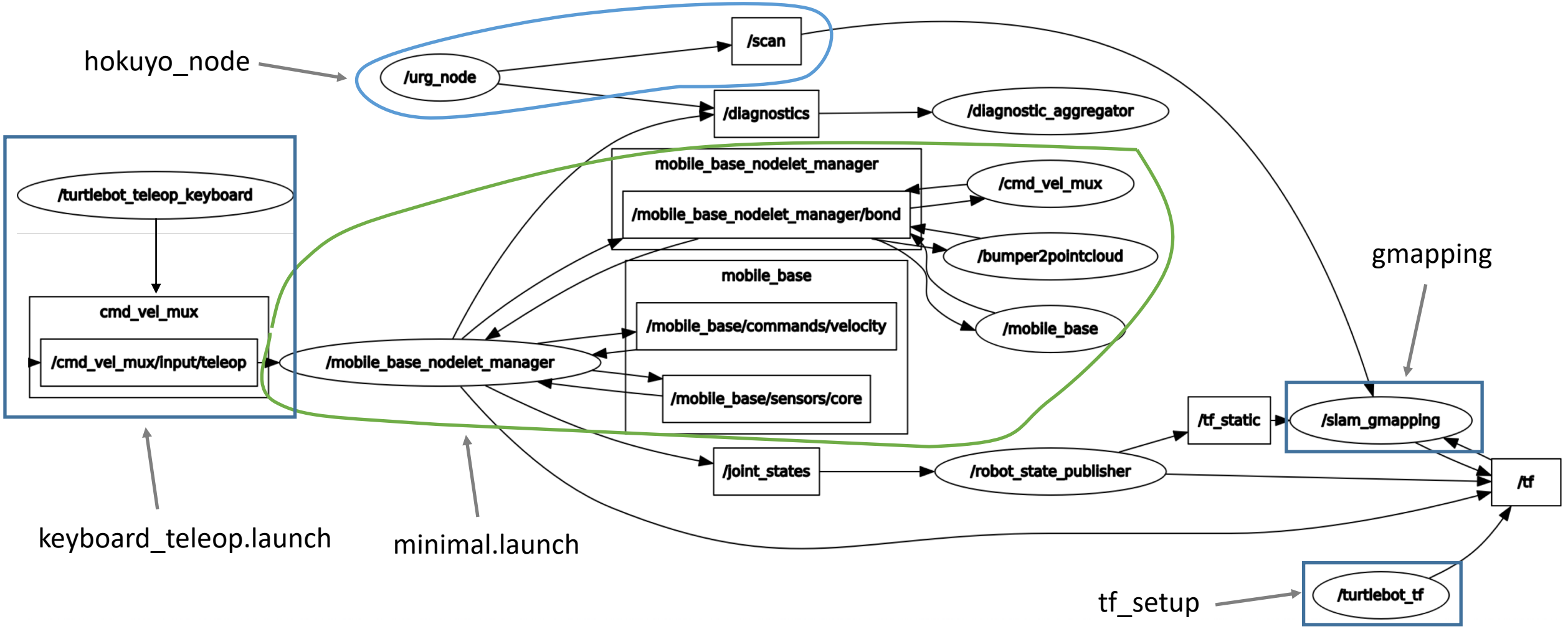
Lab #1 Node/Topic Graph

- *Nodes* represent separate processes that can communicate with each other by passing *messages* across different *topics*



Lab #1 Node/Topic Graph

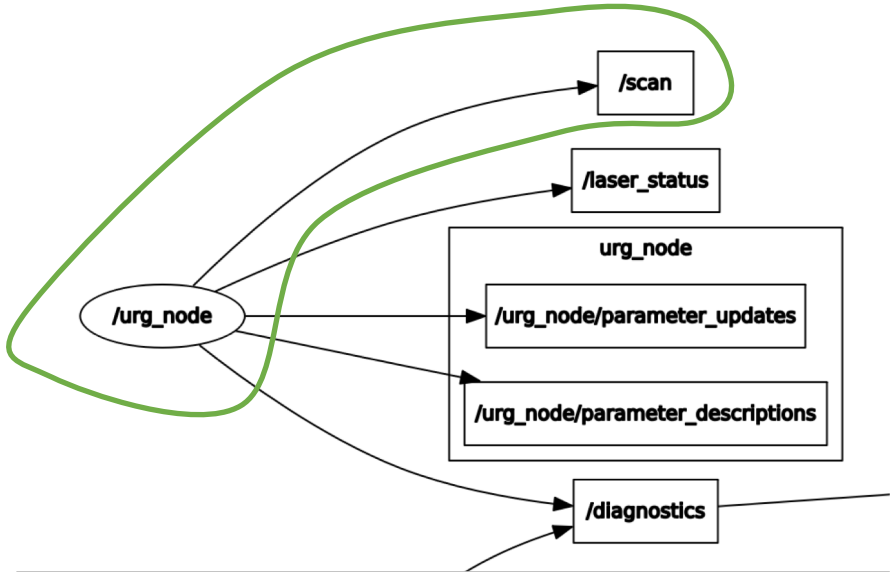
- *Nodes* represent separate processes that can communicate with each other by passing *messages* across different *topics*

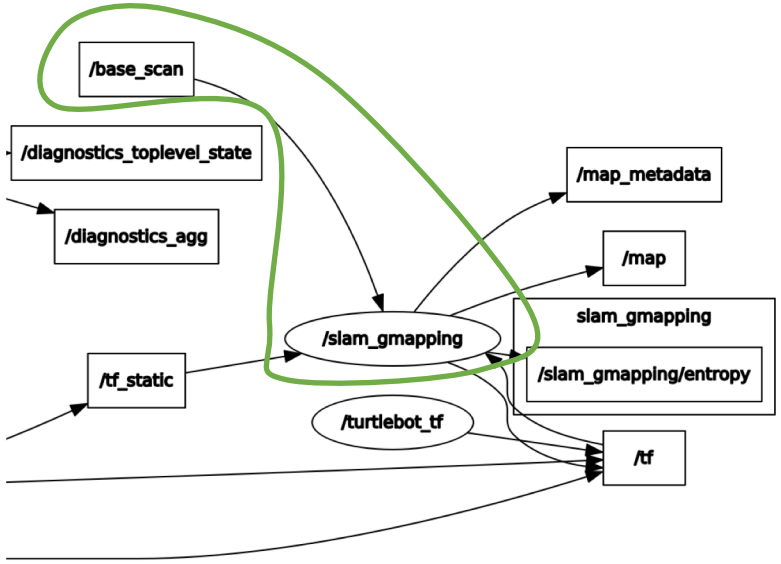


Things You Need to Get Right

- *Nodes* pass *messages* across different *topics*

1. Are our nodes communicating with each other properly over the right topics?
   - *$ rosrun gmapping slam_gmapping scan:=base_scan*

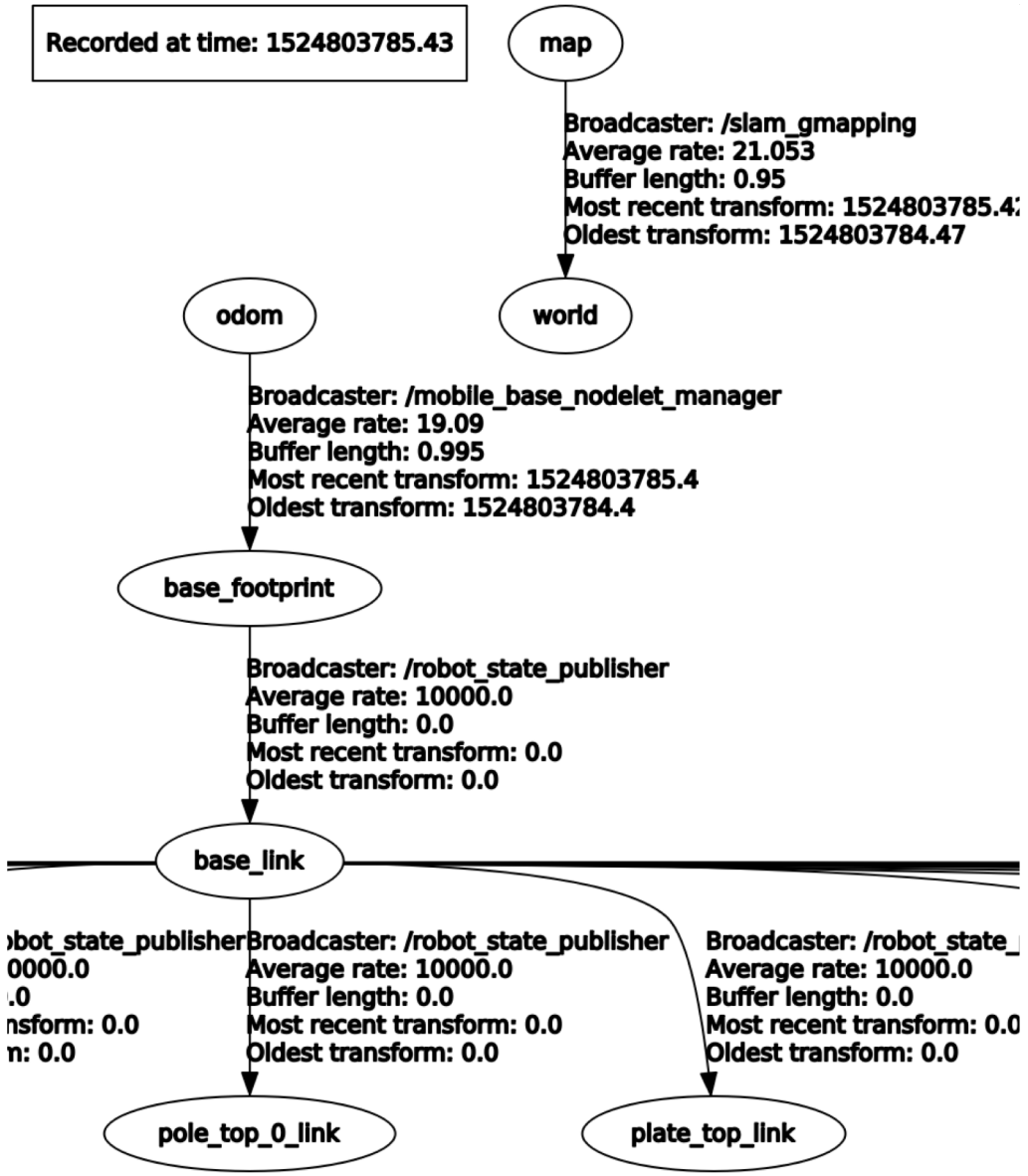




Things You Need to Get Right

- *Nodes* pass *messages* across different *topics*

1. Are our nodes communicating with each other properly over the right topics?
   - *$ rosrun gmapping slam_gmapping scan:=base_scan*
2. Are our nodes broadcasting the right transforms to the tf tree?
   - e.g. *$ rosrun gmapping _odom_frame:=world*



Laser_scan_matcher:

4.2.1 Required tf Transforms

`base_link` → `laser`

the pose of the laser in the base frame.

4.2.2 Provided tf Transforms

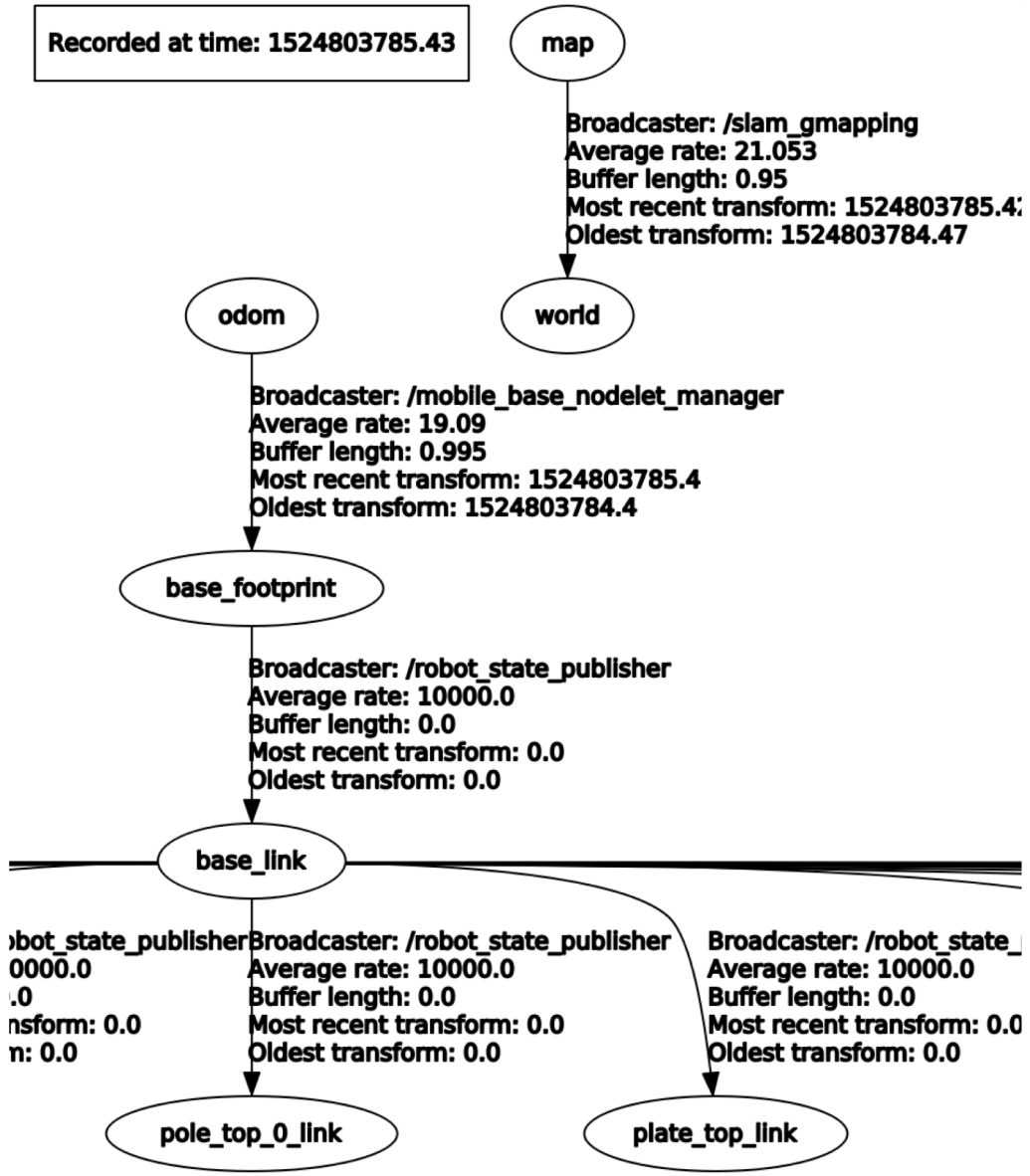
`world` → `base_link`

the pose of the robot base in the world frame.

Things You Need to Get Right

- *Nodes* pass *messages* across different *topics*

1. Are our nodes communicating with each other properly over the right topics?
   - *$ rosrun gmapping slam_gmapping scan:=base_scan*
2. Are our nodes broadcasting the right transforms to the tf tree?
   - e.g. *$ rosrun gmapping _odom_frame:=world*
3. Are there missing links in our *tf_tree?*
   - e.g. did we forget to run tf_setup.py



Things You Need to Get Right

- *Nodes* pass *messages* across different *topics*

1. Are our nodes communicating with each other properly over the right topics?
   - *$ rosrun gmapping slam_gmapping scan:=base_scan*
2. Are our nodes broadcasting the right transforms to the tf tree?
   - e.g. *$ rosrun gmapping _odom_frame:=world*
3. Are there missing links in our *tf_tree?*
   - e.g. did we forget to run tf_setup.py
4. Are we broadcasting messages of the right type?

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 s
# array), please find or create a different message, since applicatio
# will make fairly laser-specific assumptions about this data

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

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 yo
                         # is moving, this will be used in interpolati
                         # 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
float32[] intensities    # intensity data [device-specific units].  I
                         # device does not provide intensities, please
                         # the array empty.
```

Launch Files

Launch file: Tool for easily launching multiple ROS nodes and setting parameters

keyboard_teleop.launch *roslaunch turtlebot_teleop keyboard_teleop.launch*

```
File Edit Options Buffers Tools Help
<launch>
  <!-- turtlebot_teleop_key already has its own built in velocity smoother -->
  <node pkg="turtlebot_teleop" type="turtlebot_teleop_key" name="turtlebot_teleop_keyboard"  output="screen">
    <param name="scale_linear" value="0.5" type="double"/>
    <param name="scale_angular" value="1.5" type="double"/>
    <remap from="turtlebot_teleop_keyboard/cmd_vel" to="cmd_vel_mux/input/teleop"/>
  </node>
</launch>

-UU-:%%--F1  keyboard_teleop.launch   All L9     (Fundamental) ----------------------------------------------------
End of buffer
```

Equivalent to:

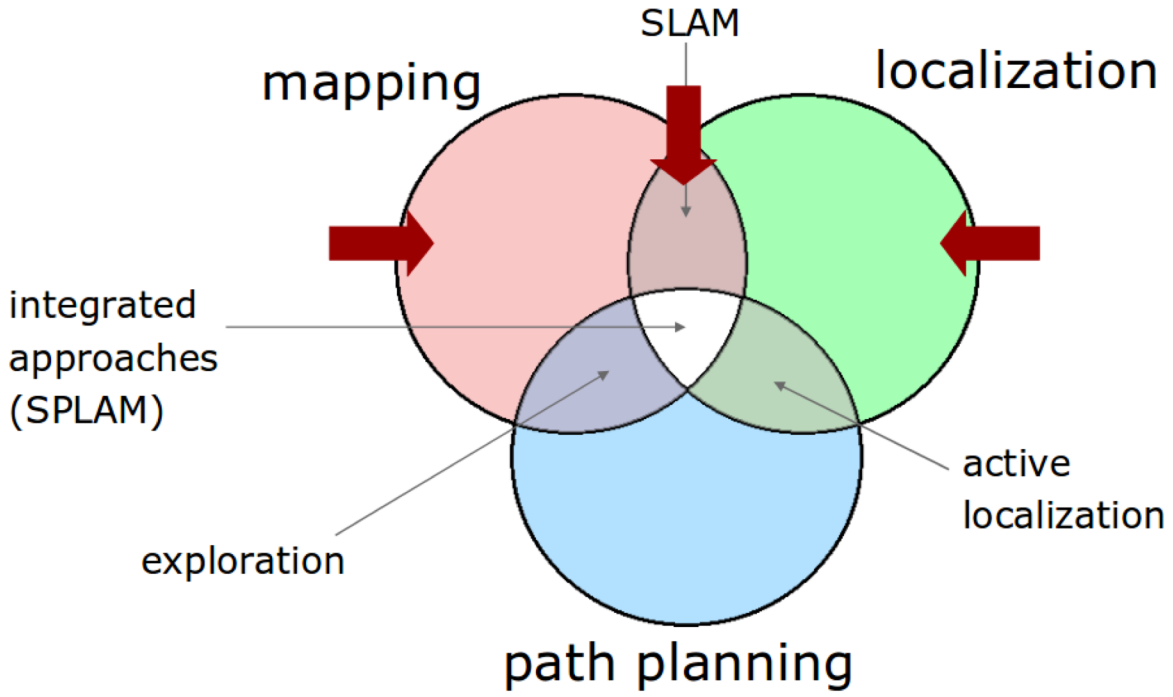
rosrun turtlebot_teleop turtlebot_teleop_key turtlebot_teleop/cmd_vel:=cmd_vel_mux/input/teleop _scale_linear:=0.5 _scale_angular:=1.5

Why do we use launch files (*roslaunch*) instead of directly running nodes (*rosrun*)?

Robot Navigation

Navigation requires:

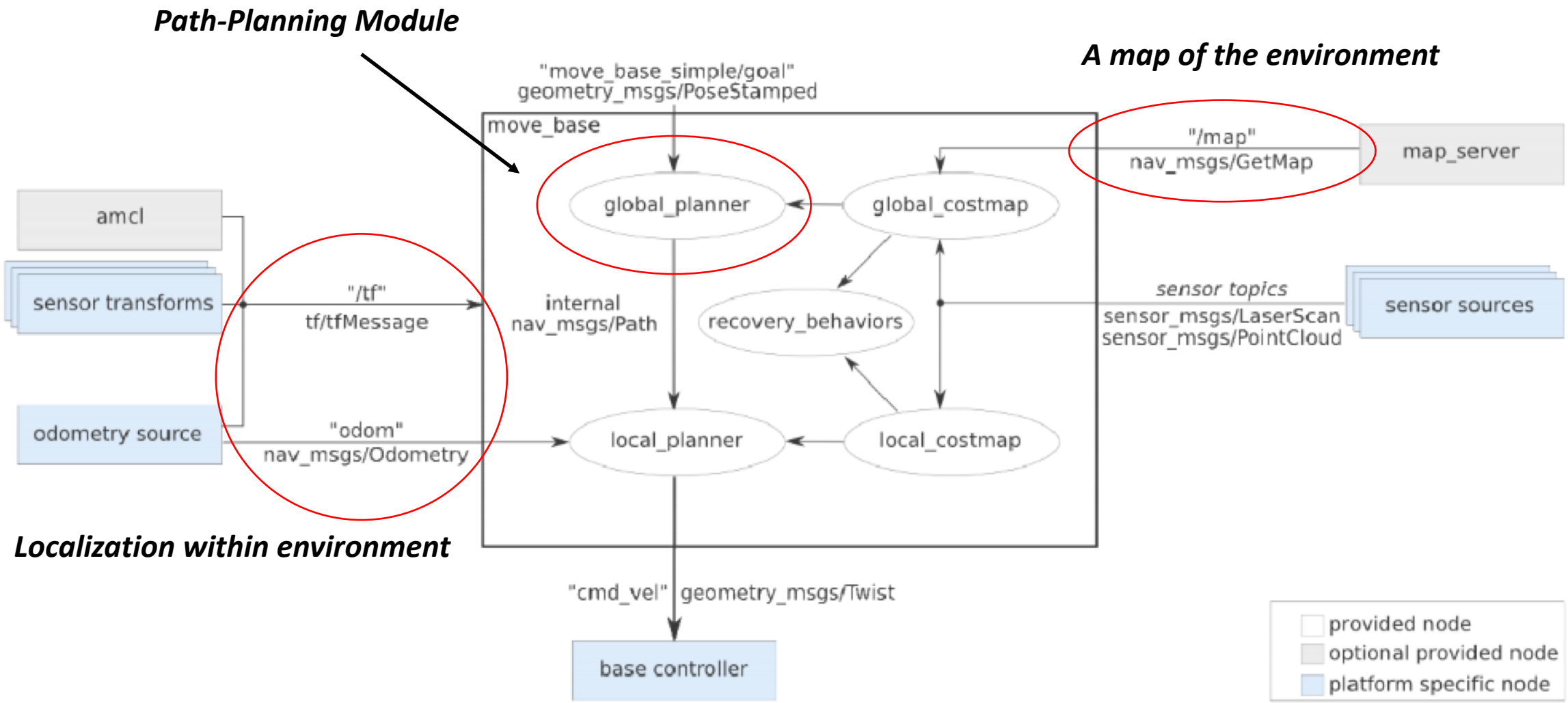
- A map of the environment (e.g. *gmapping*)
- Localization within the environment (e.g. *odometry, laser scan matching, Optitrack*)
- A path-planning module (e.g. *A*, D*)*



[courtesy of Cyrill and Wolfram]

Navigation Stack

Brings together all the components we need for navigation!!



Interfacting with Navigation Stack

Read ROS API!!

1.1.2 Action API

The `move_base` node provides an implementation of the `SimpleActionServer` (see actionlib documentation), that takes in goals containing `geometry_msgs/PoseStamped` messages. You can communicate with the `move_base` node over ROS directly, but the recommended way to send goals to `move_base` if you care about tracking their status is by using the `SimpleActionClient`. Please see actionlib documentation for more information.

Action Subscribed Topics

`move_base/goal` (move_base_msgs/MoveBaseActionGoal)

A goal for `move_base` to pursue in the world.

`move_base/cancel` (actionlib_msgs/GoalID)
A request to cancel a specific goal.

Action Published Topics

`move_base/feedback` (move_base_msgs/MoveBaseActionFeedback)

Feedback contains the current position of the base in the world.

`move_base/status` (actionlib_msgs/GoalStatusArray)
Provides status information on the goals that are sent to the `move_base` action.

`move_base/result` (move_base_msgs/MoveBaseActionResult)
Result is empty for the `move_base` action.

1.1.3 Subscribed Topics

`move_base_simple/goal` (geometry_msgs/PoseStamped)
Provides a non-action interface to move_base for users that don't care about tracking the execution status of their goals.

1.1.4 Published Topics

`cmd_vel` (geometry_msgs/Twist)
A stream of velocity commands meant for execution by a mobile base.

Navigation Stack

1.1.3 Subscribed Topics

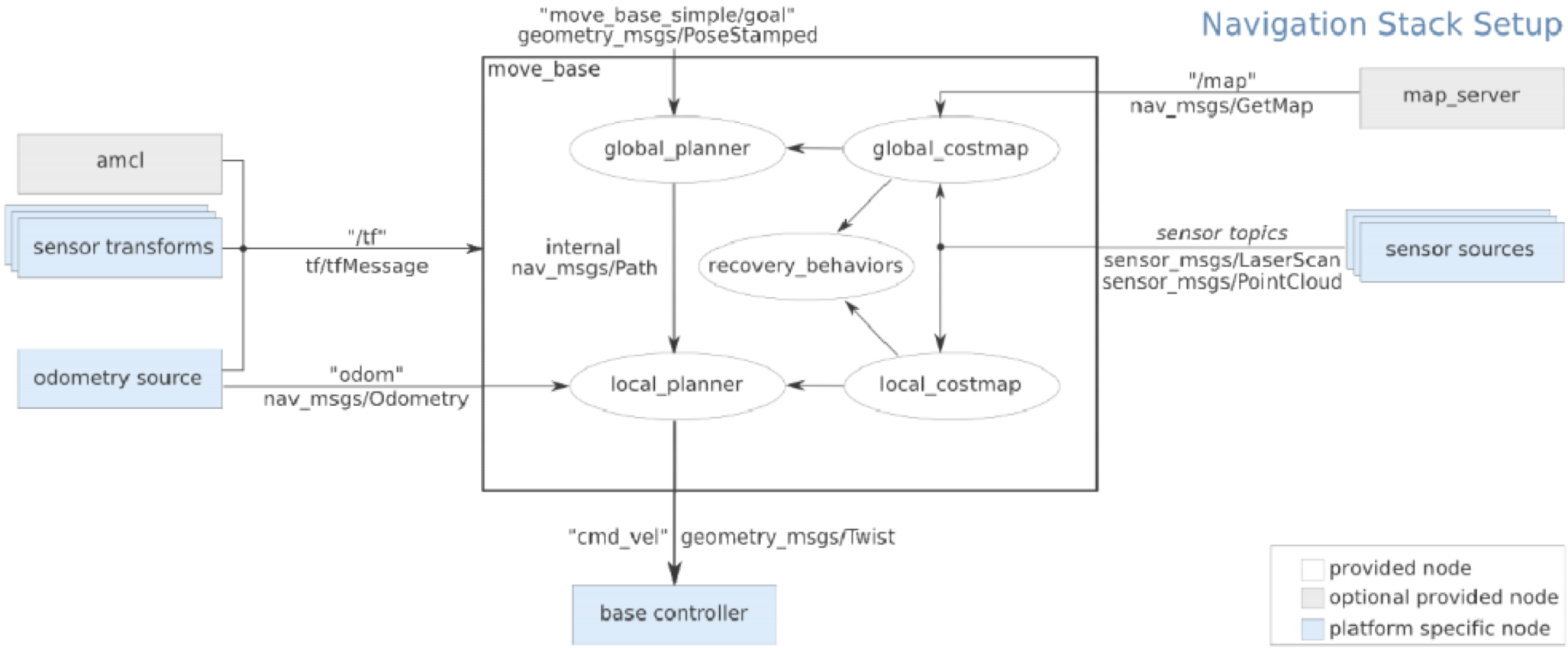
`move_base_simple/goal` (geometry_msgs/PoseStamped)

Provides a non-action interface to move_base for users that don't care about tracking the execution status of their goals.

1.1.4 Published Topics

`cmd_vel` (geometry_msgs/Twist)

A stream of velocity commands meant for execution by a mobile base.

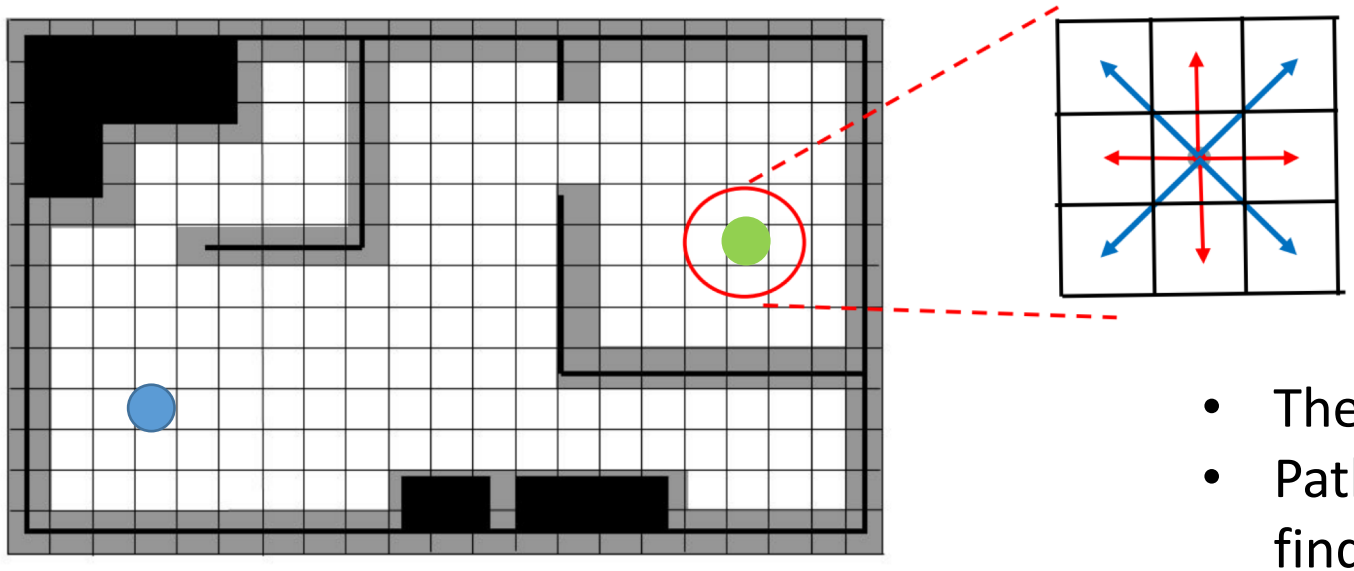


Motion Planning with MDPs

Definition: A *Mark Decision Process* (MDP) consists of

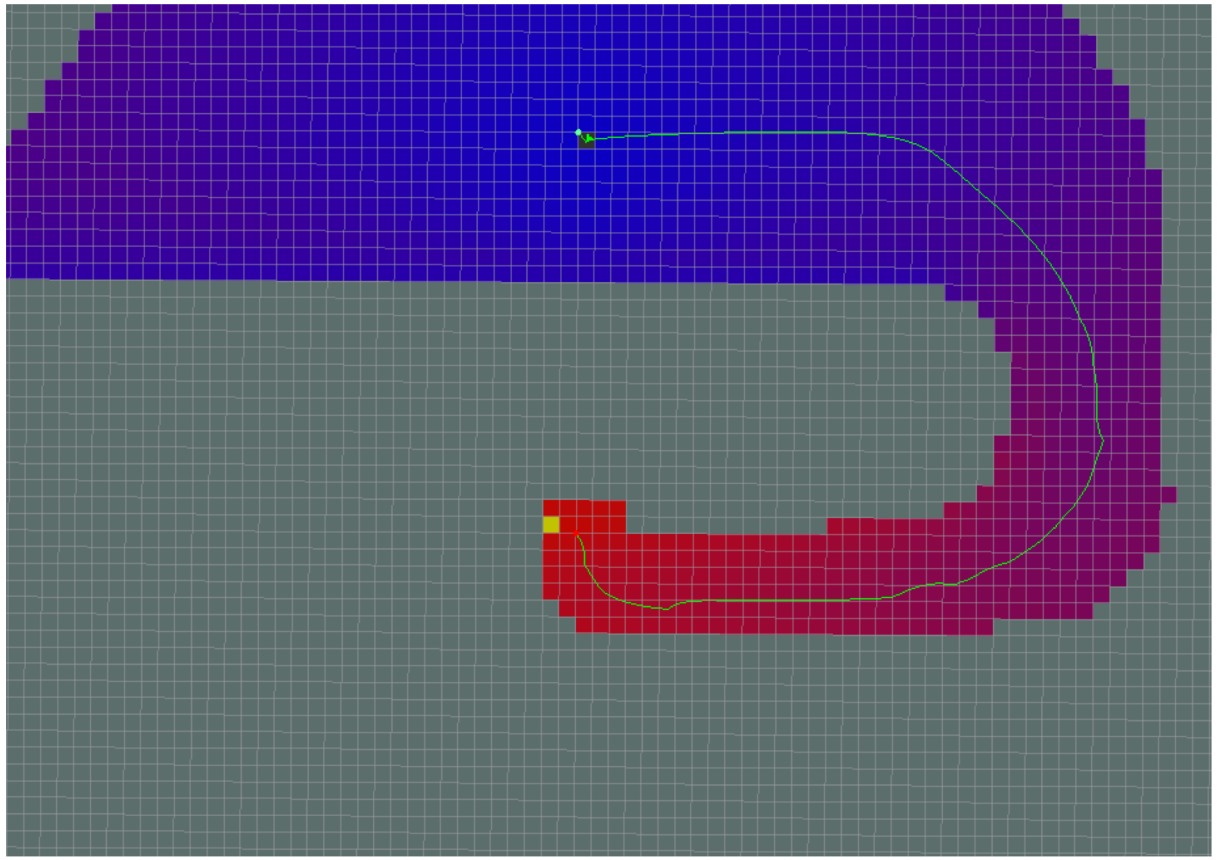
- A discrete set of states, $S = \{x_1, x_2, \dots, x_N\}$
- A set of possible actions to take in each state: $U = \{u_1, \dots, u_k\}$
  - Set of actions can be state dependent: $\mathrm{U_i} = \mathrm{U(x_i)}$
- A *reward function* $r(x, u) \to \mathbb{R}$
  - Reward can incorporate *goal information*

$$r(x,u) = \begin{cases} +100 & if\ u\ leads\ to\ the\ goal \\ -1 & otherwise \end{cases}$$



- The reward captures the robot's overall goal
- Path planning algorithms *solve* the MPD to find the path with highest reward

Path Planner



Ingredients:

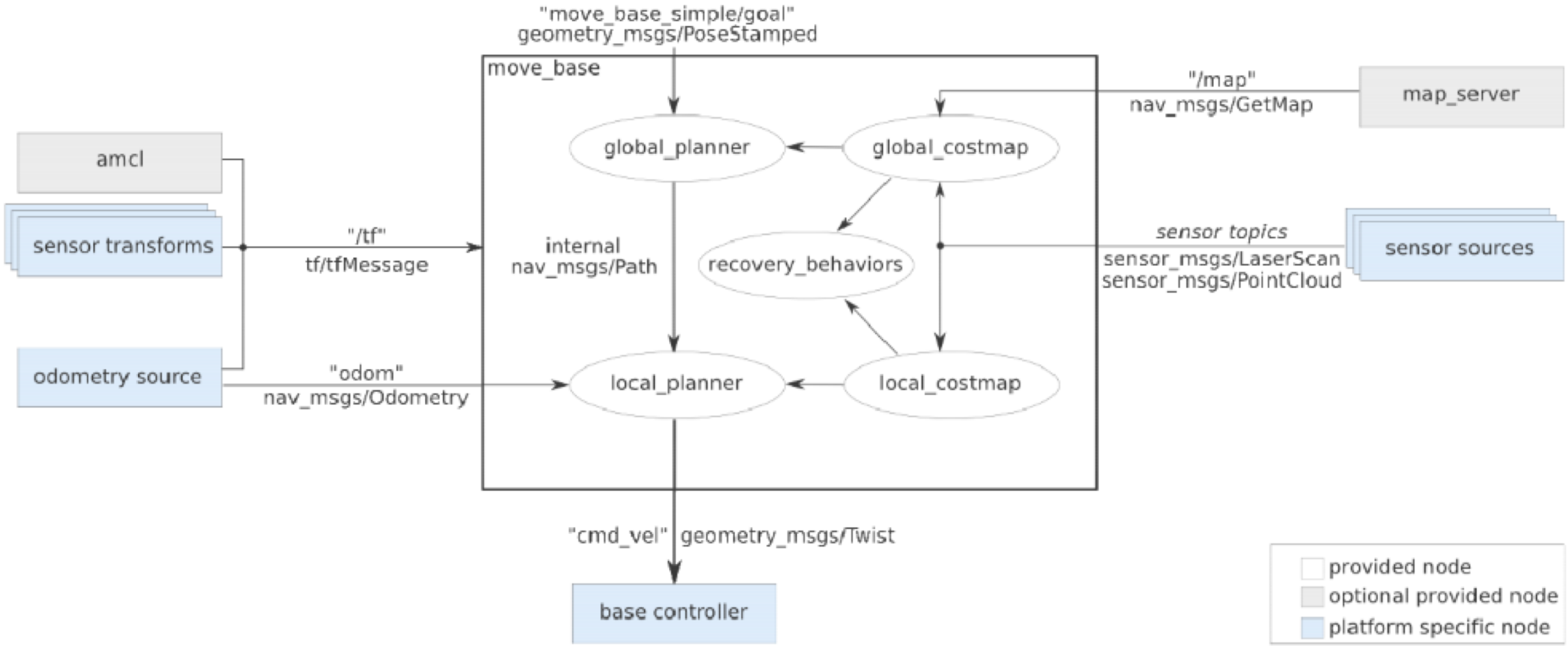
1. Map of environment
2. Localization in map
3. Reward function

Color represents accumulated reward/cost if starting from that state

- Red means you accumulate less reward (more cost) from that state
- Blue means you accumulate lot of reward (little cost) from that state

Path planner travels towards region of higher return to reach goal

ROS Messages



ROS Messages

Every type of message sends different information in different format

- To function, nodes must send/receive the proper information in the proper format

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.
```

rostopic echo /scan

```
---
header:
  seq: 301
  stamp:
    secs: 1524730484
    nsecs: 846581583
  frame_id: "laser"
angle_min: -2.35619449615
angle_max: 2.09234976768
angle_increment: 0.00613592332229
time_increment: 9.76562732831e-05
scan_time: 0.10000000149
range_min: 0.019999999553
range_max: 5.59999990463
ranges: [0.01899999938905239, 0.01899999938905239, 0.01899999938905239,
9999938905239, 0.01899999938905239, 0.01899999938905239, 0.018999999389
0.01899999938905239, 0.01899999938905239, 0.01899999938905239, 0.018999
5239, 0.01899999938905239, 0.01899999938905239, 0.01899999938905239, 0.
9938905239, 0.01899999938905239, 0.01899999938905239, 0.018999999389052
984009, 3.385999917984009, 3.4089999198913574, 3.4130001068115234, 3.42
, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan,
```

ROS Debugging Tools

1. rosrun rqt_tf_tree rqt_tf_tree
   - Do necessary components exist, and are they connected properly?
2. rqt_graph
   - Are all nodes communicating properly across the right topics?
3. rostopic echo … (or rostopic info ...)
4. rviz → add in the frames/topics you would like to read from)
   - If you want to see laser scans, add the “laser” frame to your rviz and display the /scan topic associated with it
   - *$ rviz rviz –d turtlebot_rgbd_lidar.rviz*
5. Read APIs for ROS nodes you are using!!
   - Make sure parameters are defined as you want them
   - Some nodes require certain transformations in the *tf* tree to exist

Questions?