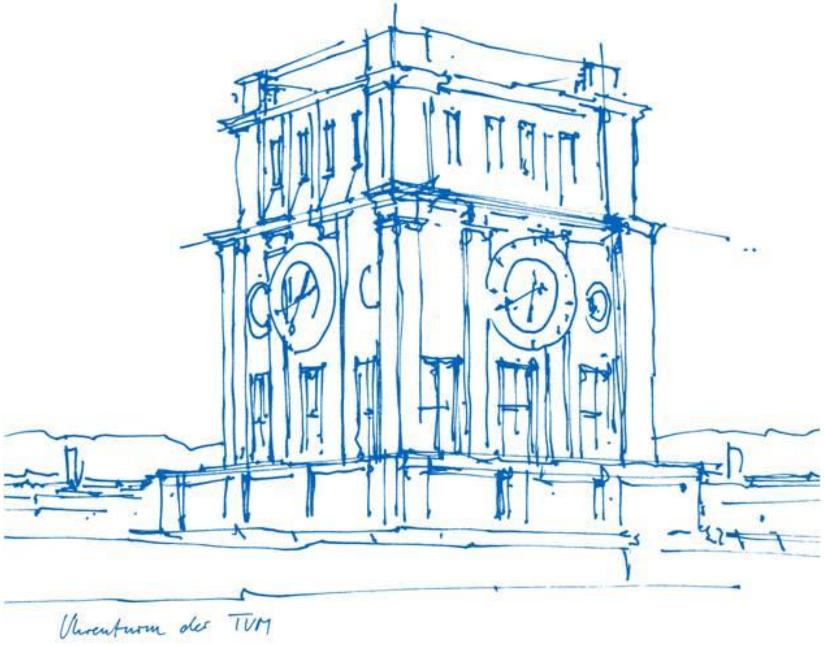
Parking Team Presentation



Zhenrui Yue

06.06.2019

Topics of the presentation

- **ROS**
- **ROS Navigation Stack**

Topics of the presentation

- **ROS**

- **ROS Navigation Stack**

ROS: Robot Operating System

- **What is ROS?**
  - **A framework for robot software development**
  - **A Collection of tools, resources and drivers**
  - **Low level hardware control possible**

- **Why do we need ROS?**
  - **For distributed, modular design => packages**
  - **Easy to program & adapt => C++ & Python**
  - **Resources and community support**

ROS
=
Plumbing
+
Tools
+
Capabilities
+
Ecosystem

ROS: Core Components

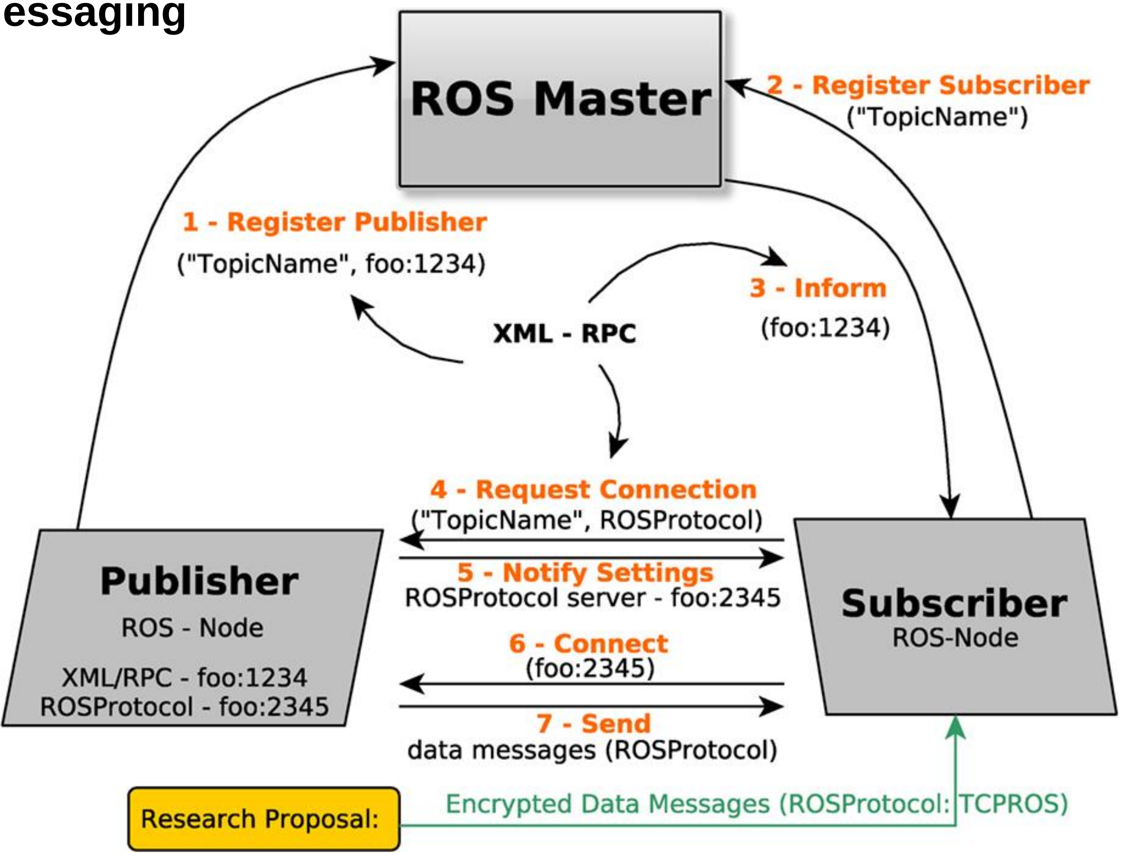
- **Messaging**
  - **Via An anonymous info publish / subscribe mechanism**
  - **Every specific Message passing under a certain topic**
  - **Asynchronous / remote / multiple communication**

- **Robot Specific Features**
  - **Standard message formats** (nav_msgs, sensor_msgs e.g. Path, LaserScan)
  - **Robot Geometry** (tf) **/ Description Language** (Unified Robot Description Format)
  - **Remote Procedure Call** (service) **/ Diagnostics** (diagnostics)
  - **Localization and Navigation** (navigation stack)

- **Tools**
  - **Rviz** (visualization)
  - **Rqt** (GUI development for ROS)

ROS: Concepts

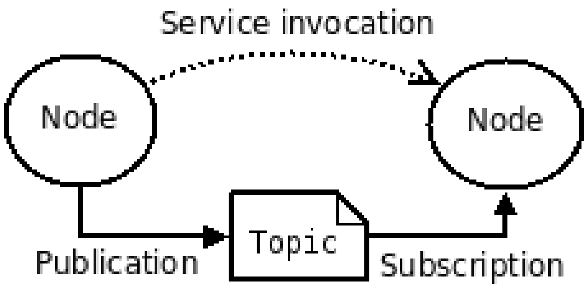
- **Master: locating nodes and enabling P2P communication**
- **Nodes: performs computation**
- **Messages: data structure for messaging**
- **Topics: messaging pipeline**

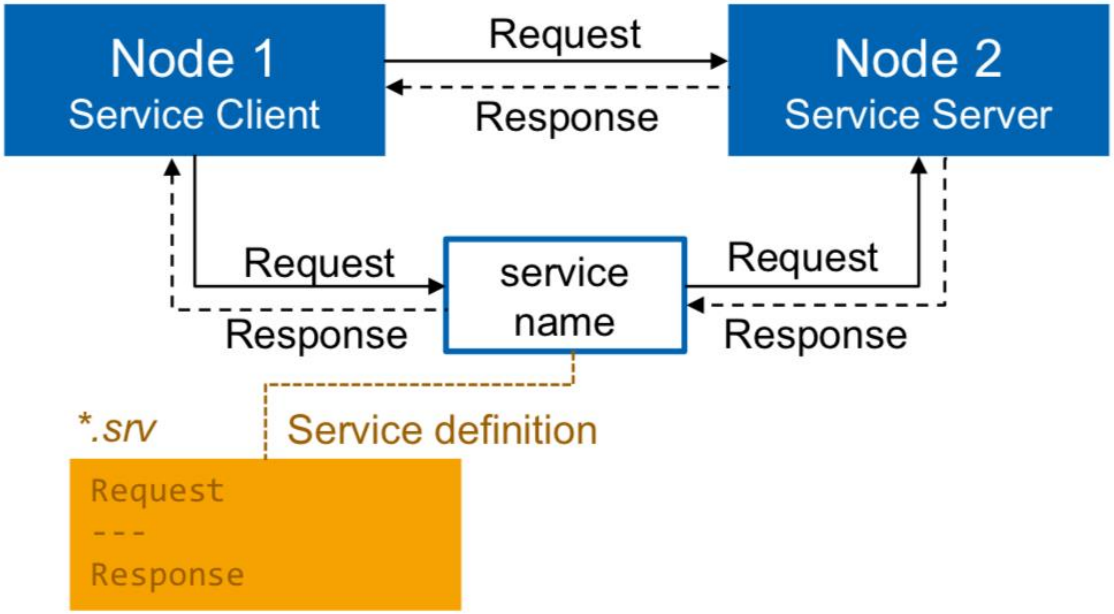


ROS: Concepts

- **Services: sending request and awaiting reply**

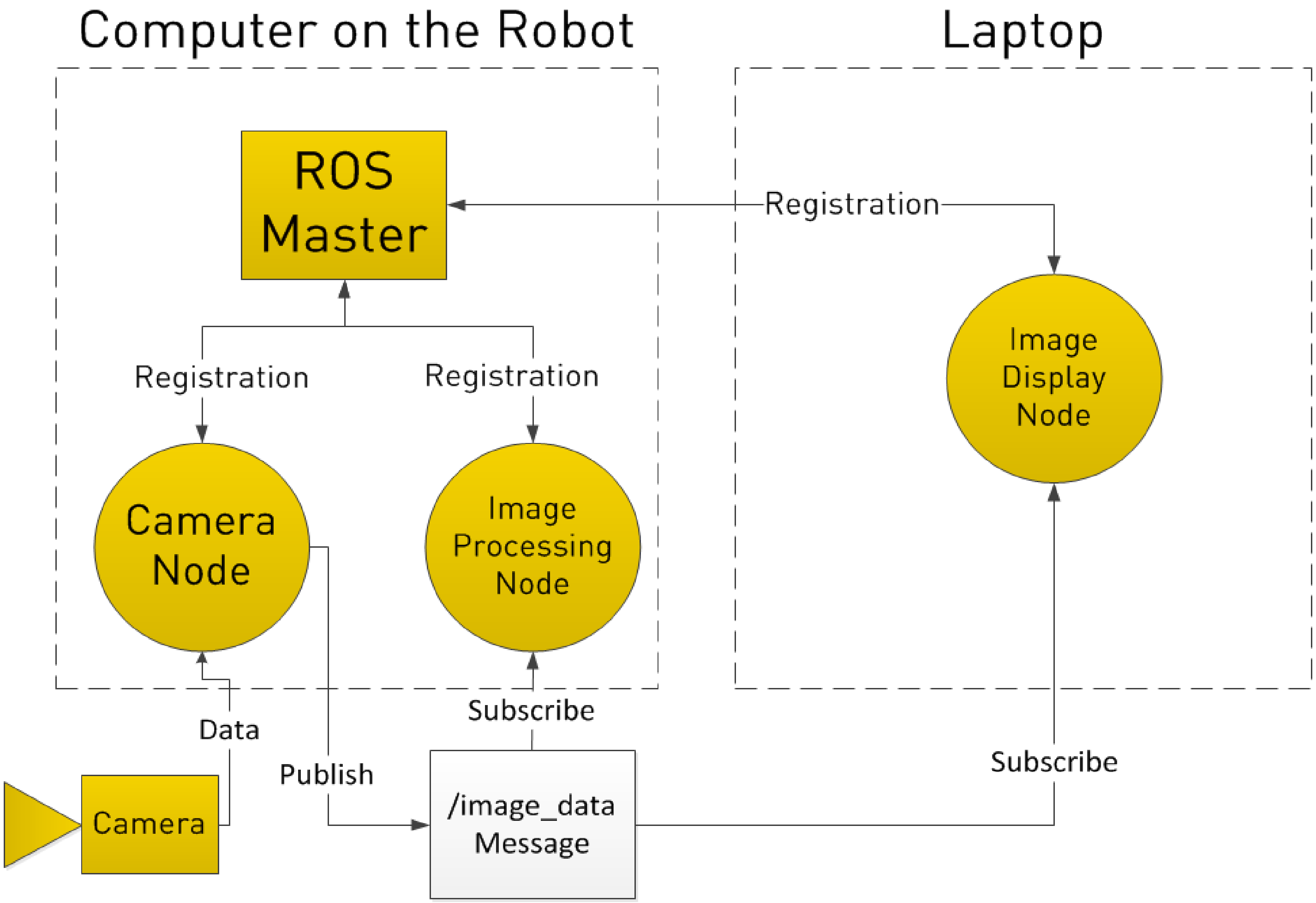
- **Bags: storing data / messages**
- **Parameter Server: storing and retriving parameter**





1. Service registers with Master
2. Service client looks up service on the Master
3. Service client creates TCP/IP to the service
4. Service client and service exchange a Connection Header
5. Service client sends serialized request message
6. Service replies with serialized response message.

ROS: A Simple Example

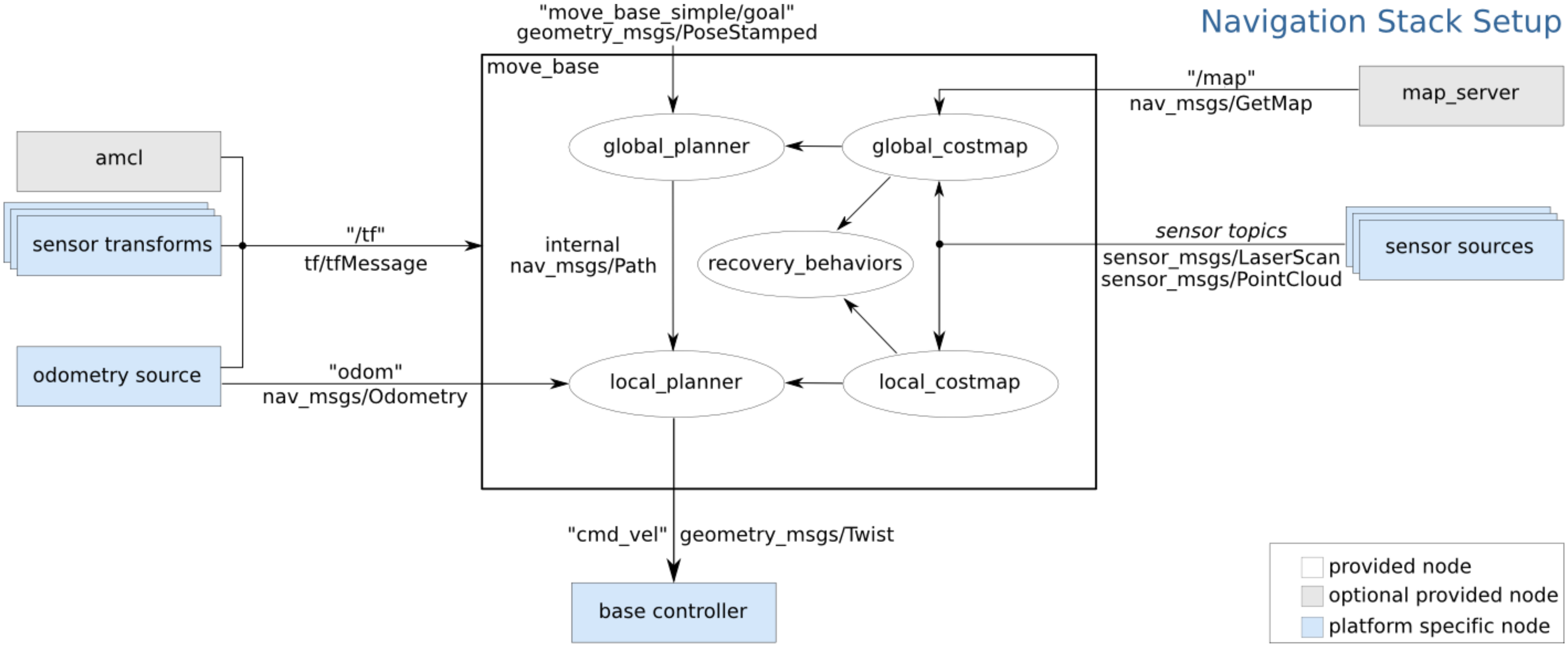


Topics of the presentation

- ROS

- **ROS Navigation Stack**

Navigation Stack: Overview



Navigation Stack: Cost Map (costmap_2d)

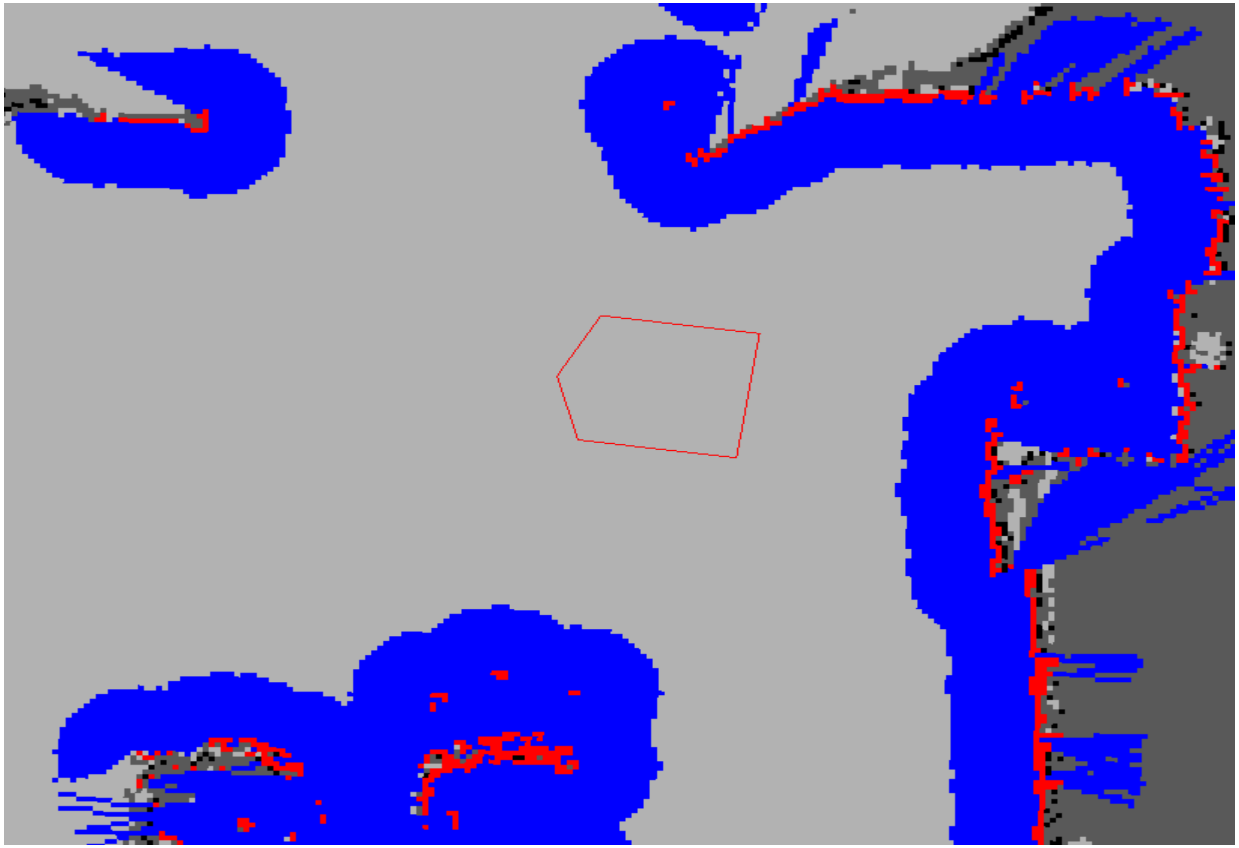
- A 2D cost map that takes in sensor and builds a grid map of the data, then inflates costs in a 2D cost map based on the occupancy grid and a user specified inflation radius.

- **Output (Map):**

nav_msgs:
/OccupancyGrid

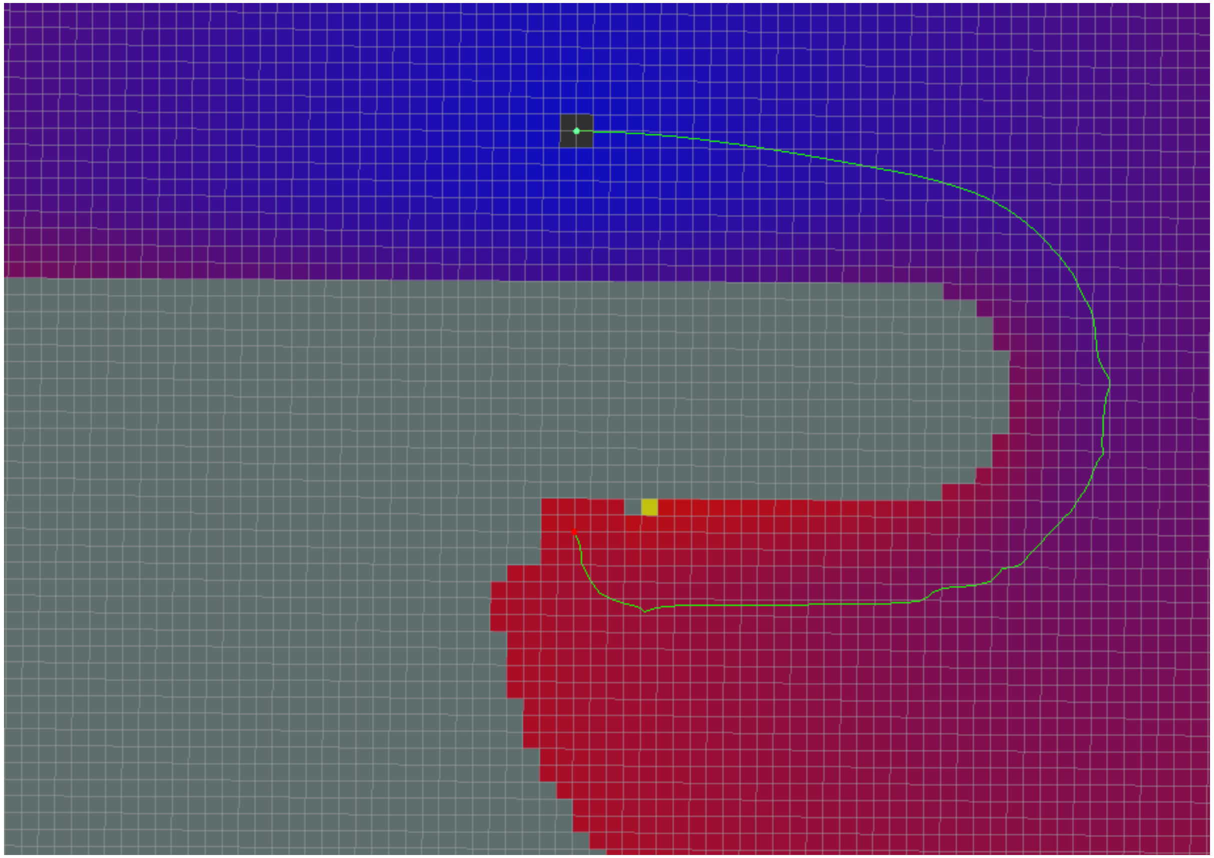
maps_msgs:
/OccupancyGridUpdate

costmap_2d:
/VoxelGrid



Navigation Stack: Global Planner (global_planner)

- An implementation of a fast, interpolated global planner for navigation. This class adheres to the nav_core::BaseGlobalPlanner interface and provides A* or Dijkstra to calculate the path.



- **Output (Path):**

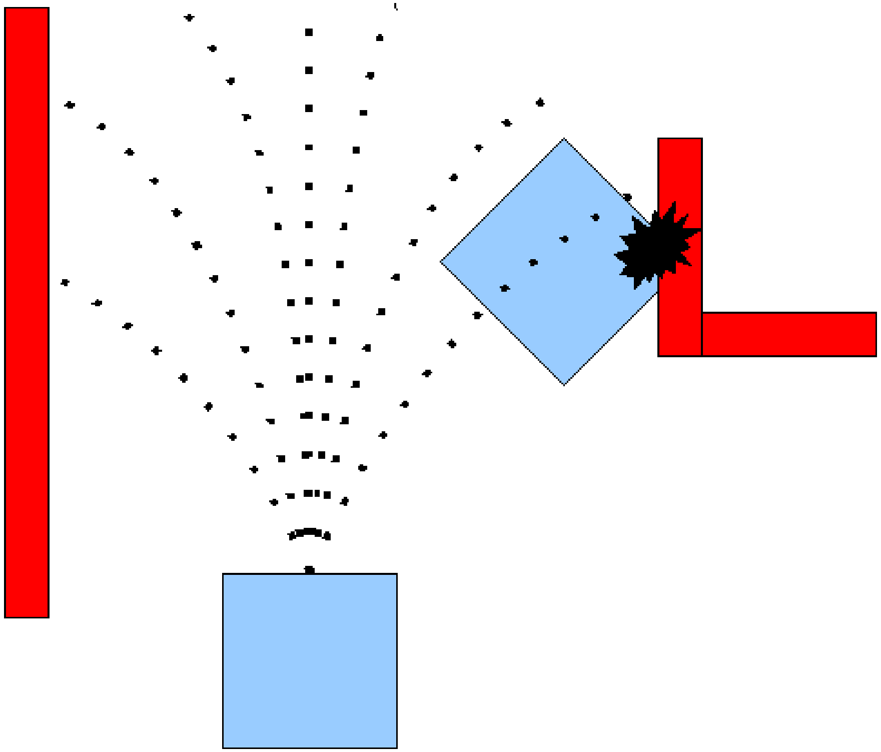
nav_msgs:
/Path

- **Parameter:**

/cost_factor
/lethal_cost
/neutral_cost
/use_dijkstra
/orientation_mode
……

Navigation Stack: Local Planner (base_local_planner)

- Using a map, the local planner creates a kinematic trajectory for the robot to get from a start to a goal location. The planner creates a value function that encodes the costs of traversing through the grid cells and use this value function to determine dx, dy, dtheta velocities to send to the robot.



- **Output (Path):**

nav_msgs:
/Path

- **Parameter:**

/acc_lim_x
/acc_lim_y
/acc_lim_theta
……

Navigation Stack: Integration (move_base)

- The move_base package provides an implementation of an action, given a goal in the world, to attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish the navigation task.

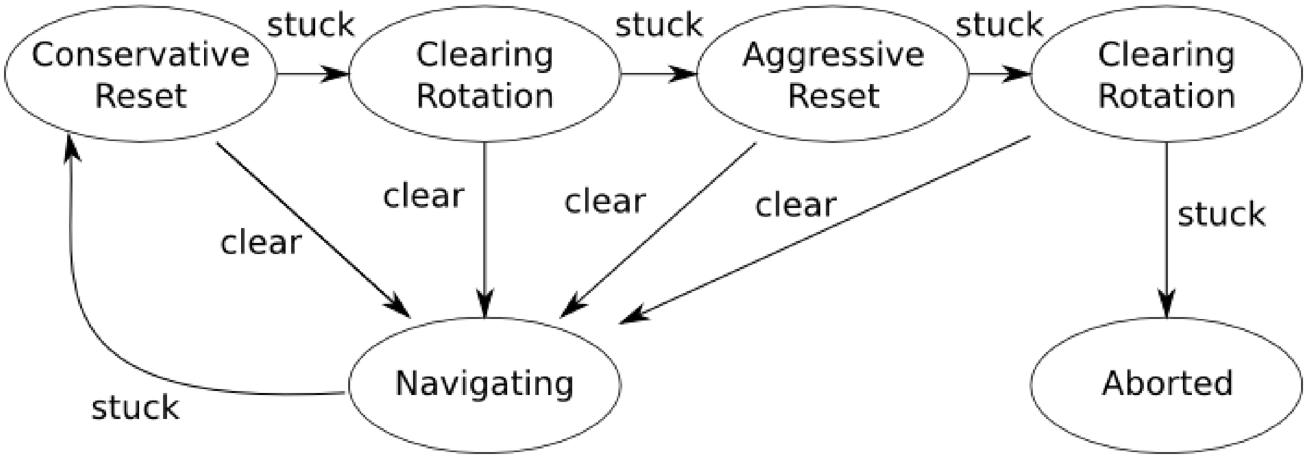
- **Input (Goal):**

Sensor Data;
geometry_msgs:
/PoseStamped

- **Output (Velocity):**

geometry_msgs:
/Twist

move_base Default Recovery Behaviors

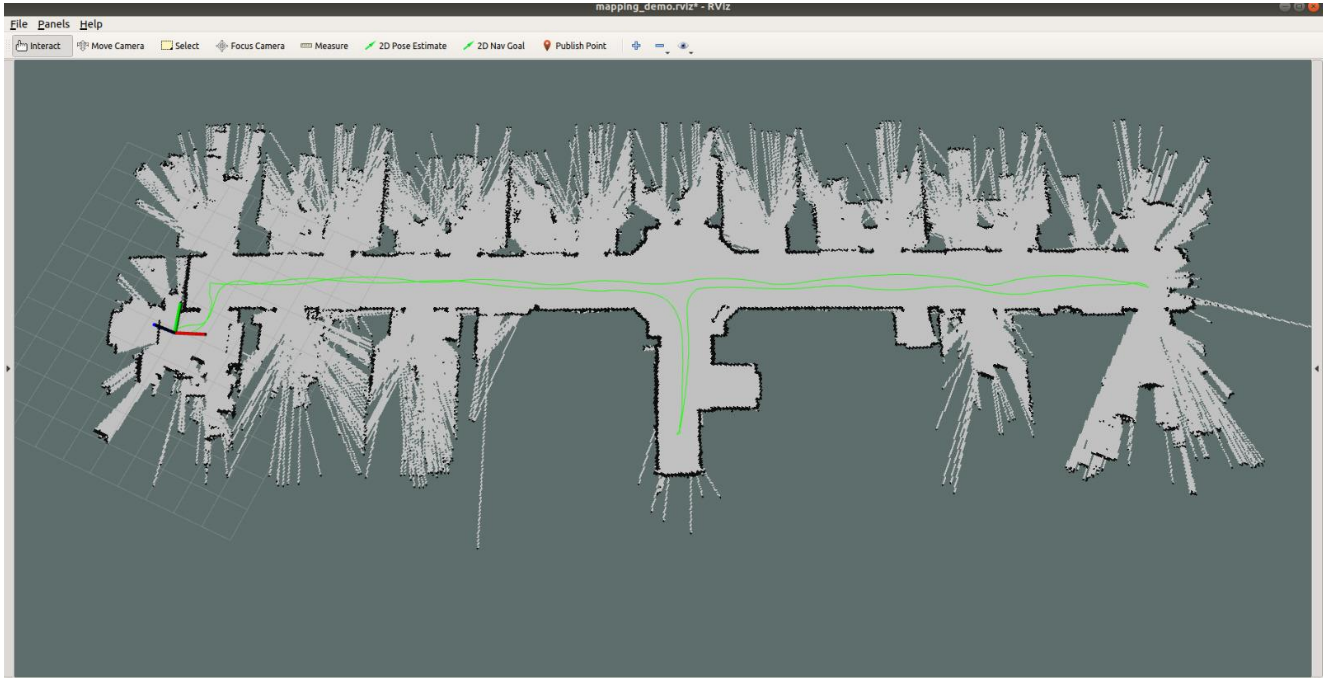


Navigation Stack: Hector SLAM

- hector_slam uses the hector_mapping node for learning a map of the environment and simultaneously estimating the platform's 2D pose at laser scanner frame rate. To use hector_slam, we will need a LiDAR sensor for /LaserScan messages to build the map.

- **Packages:**

hector_mapping
hector_map_server
hector_map_tools
hector_trajectory_server
hector_slam_launch
hector_geotiff
……



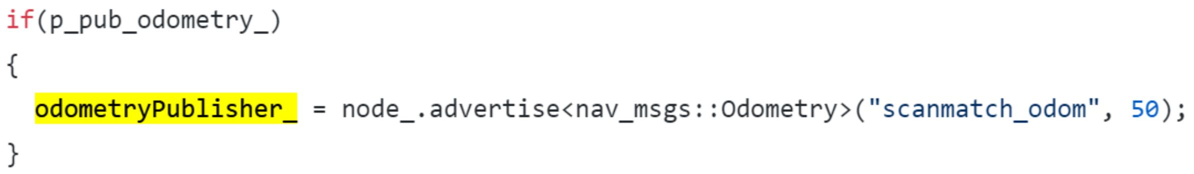
Navigation Stack: Mapping & Odom Estimation

- hector_mapping is a SLAM approach that can be used without odometry, it leverages the high update rate of modern LIDAR systems like the RPLiDAR A2 and provides 2D pose estimates at scan rate of the sensors.

- **Output:**

nav_msgs:
/MapMetaData
/OccupancyGrid

geometry_msgs:
/PoseStamped
/PoseWithCovarianceStamped

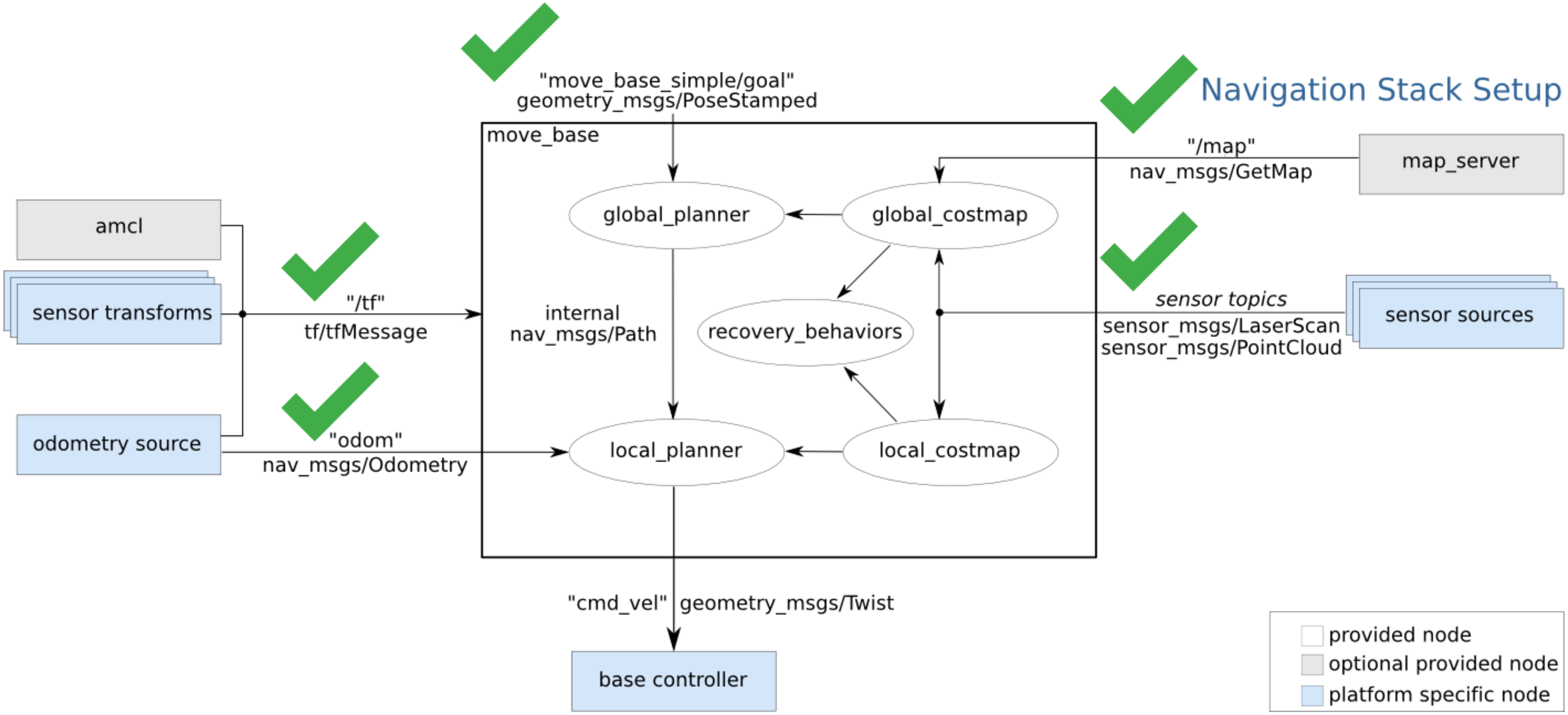


```
if(p_pub_odometry_)
{
  odometryPublisher_ = node_.advertise<nav_msgs::Odometry>("scanmatch_odom", 50);
}
```

1. Hector Mapping could also publish odometry information (Pose + Twist)
2. After modification, the hector_mapping package published odometry information under **/scanmatch_odom**

3. Another possibility for odometry information is to user **laser_scan_matcher** package, which also provides estimated odometry based on laser data

Navigation Stack: Overview



Next Step: Integrate and Test Navigation Stack

Thank you!