SAPIENZA UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA E SISTEMISTICA ANTONIO RUBERTI

Introduction to Navigation using ROS

Giorgio Grisetti

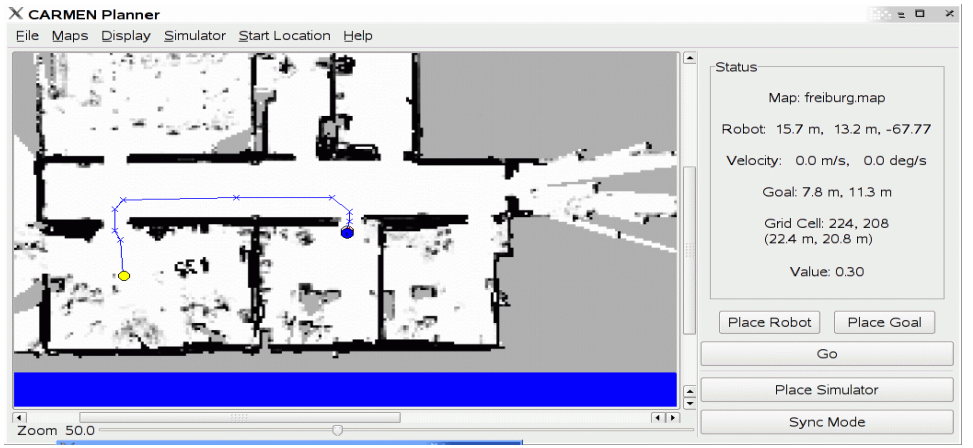
Part of the material in these slides is taken from the Robotics 2 lectures given by G.Grisetti, W.Burgard, C.Stachniss, K.Arras, D. Tipaldi and M.Bennewitz

Outline

- Navigation Concepts
  - Map, Robot Pose, and Path
  - Taxonomy of Navigation
    - Localization
    - Planning
    - Building a Map
- Building a Map
- Localizing in the map

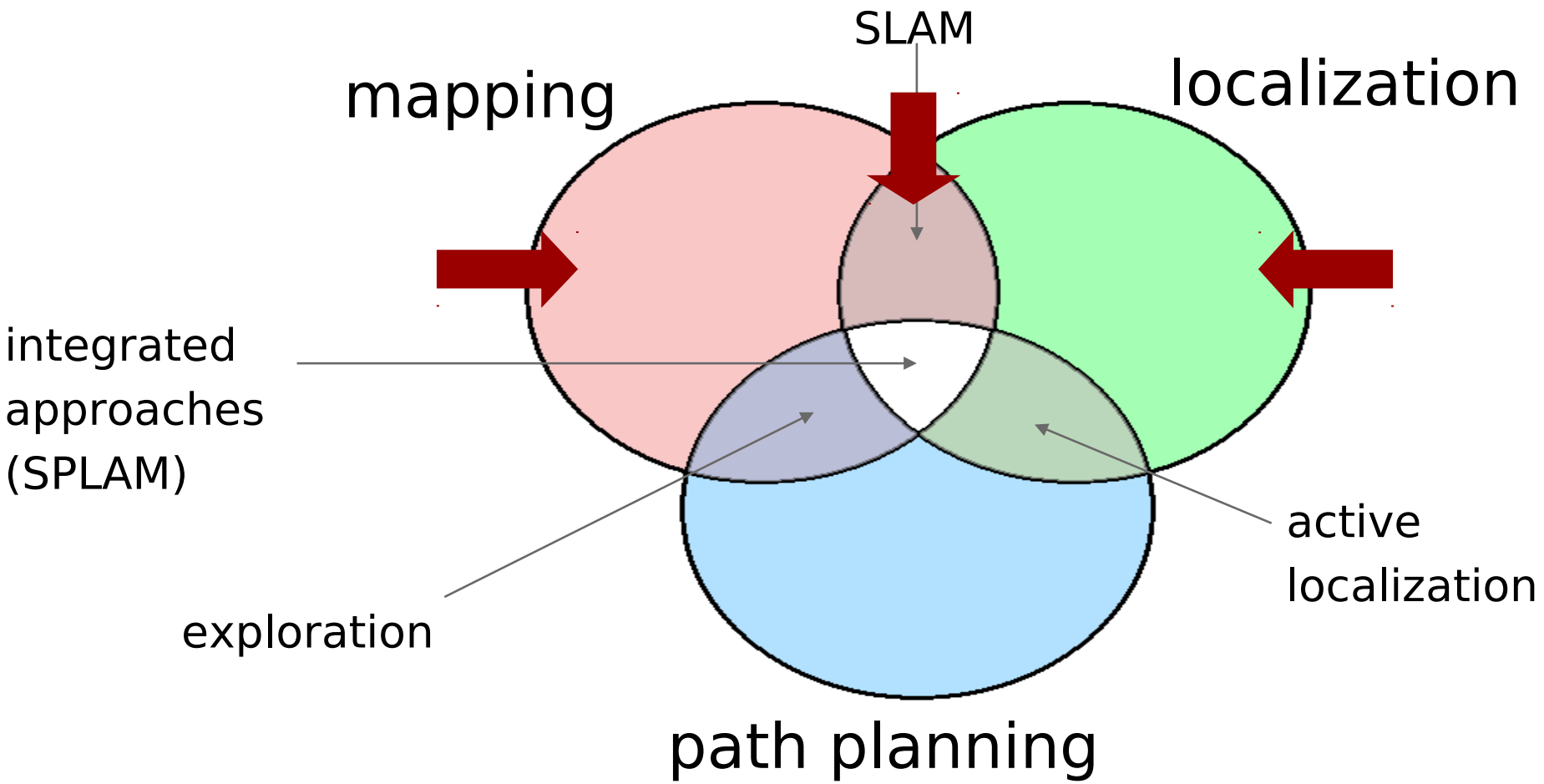
Sense - Plan - Act

- To **accomplish a task**, a robot should “understand” the environment from it sensor measurements.
- **Understanding**: capturing the information at the right level of abstraction.
- An autonomously navigating robot should know
  - what the **environment** looks like and
  - **where** it is in the environment
  - ...



[courtesy of Rainer Kuemmerle and Dirk Heahnel]

What is this Talk about?



[courtesy of Cyrill and Wolfram]

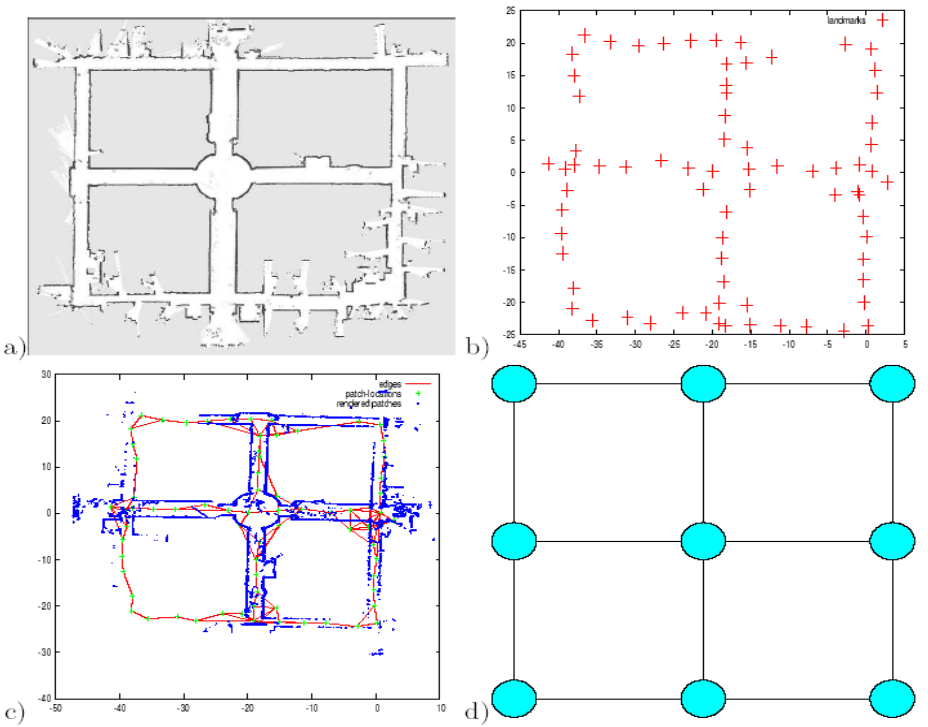
Map

- A map is a representation of the environment where the robot is operating.

- It should contain enough information to accomplish a task of interest.

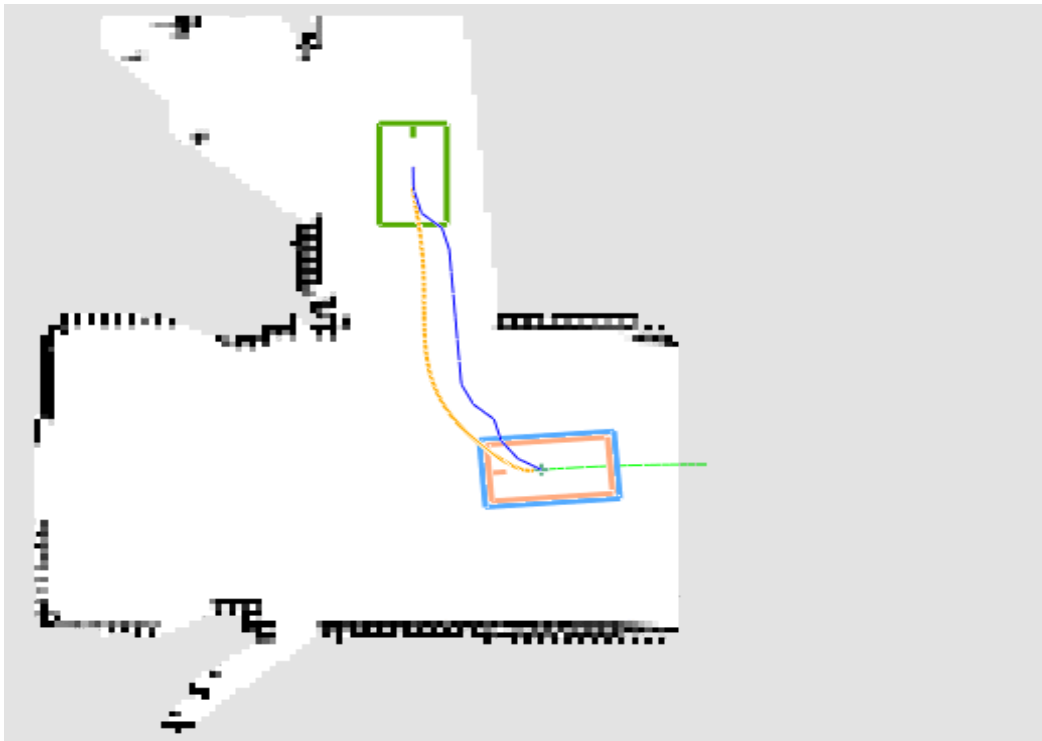
Representations:

- Metric
  - Grid Based
  - Feature Based
  - Hybrid
- Topological
- Hybrid



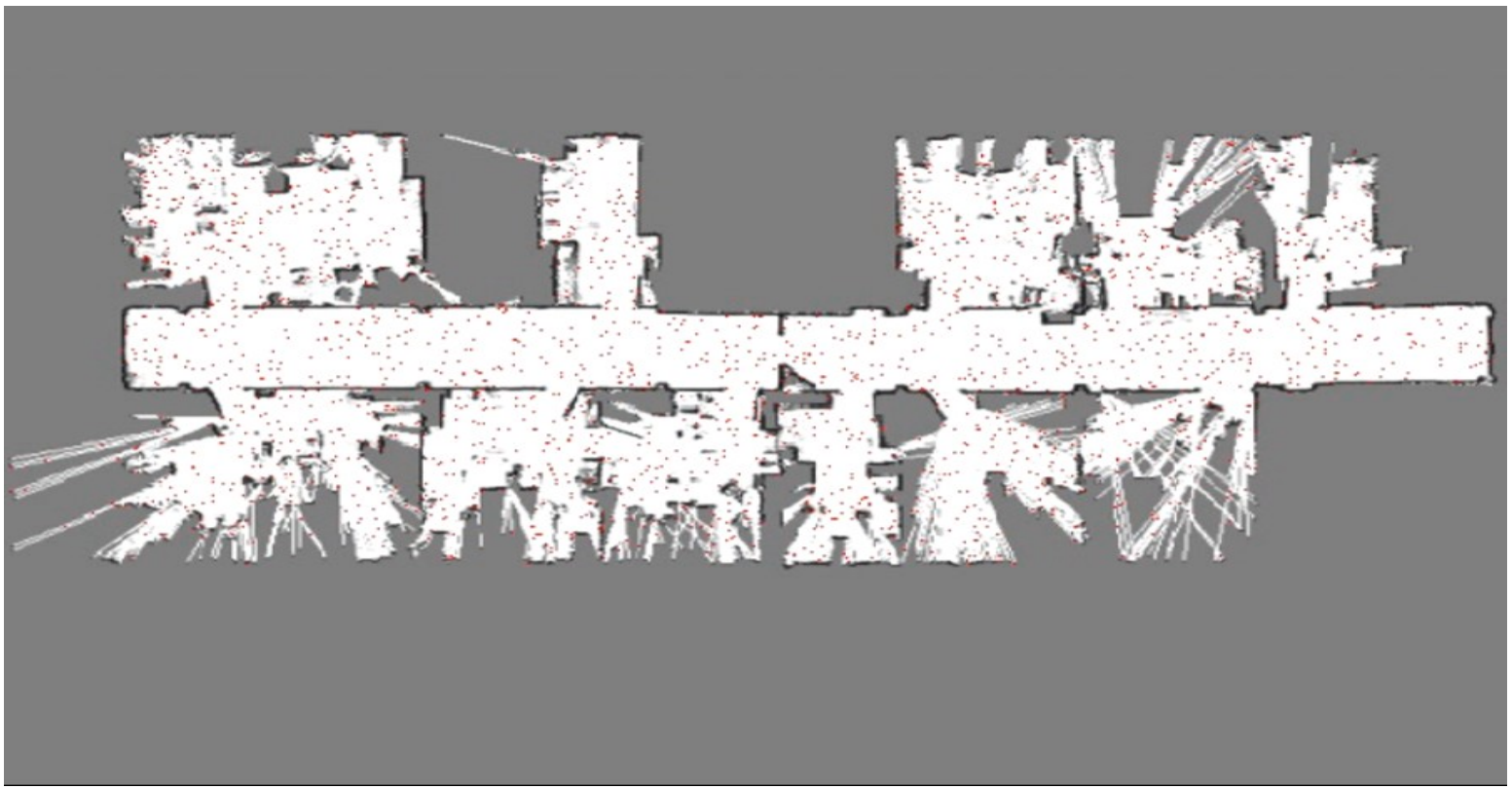
Robot Pose and Path

- A metric map defines a reference frame.
- To operate in a map, a robot should know its position in that reference frame.
- A sequence of waypoints or of actions to reach a goal location in the map is a path.



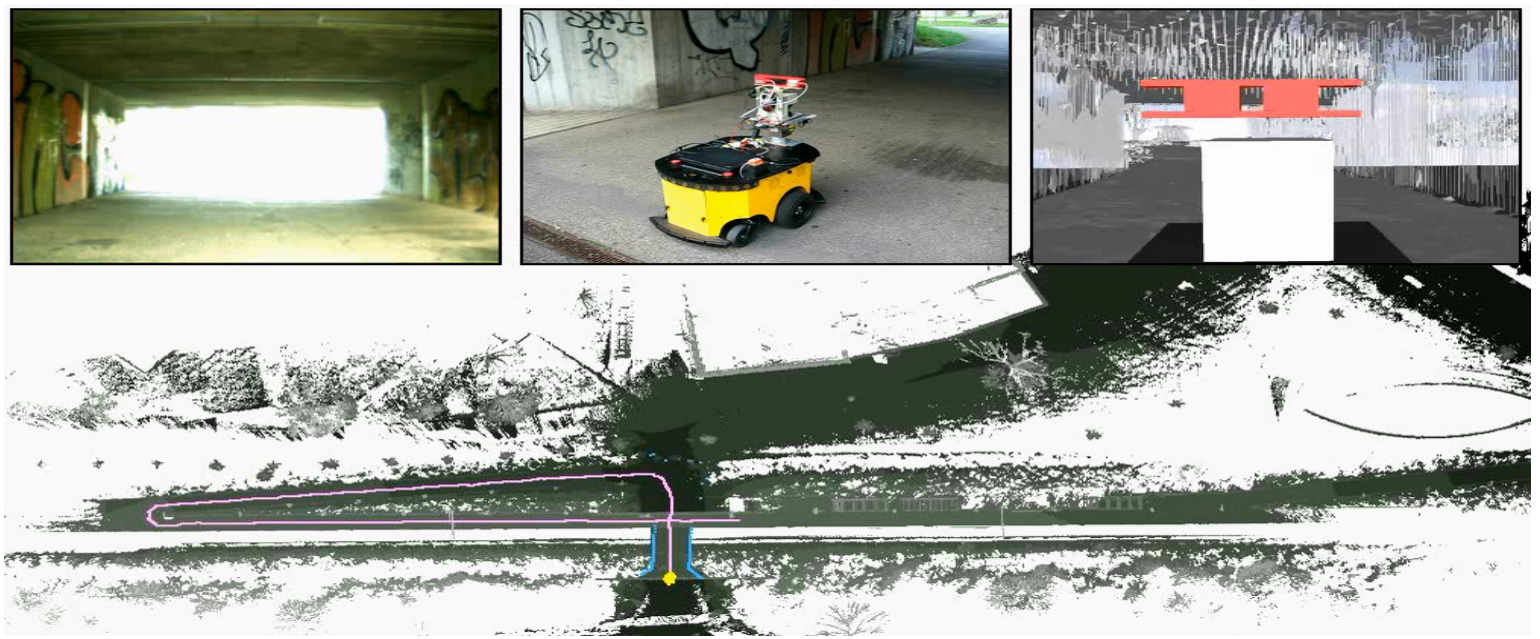
Localization

- Determine the current robot position, using the measurements up to the current instant and a map.



Path Planning

- Determine (if it exists) a path to reach a given goal location given a localized robot and a map of traversable regions.



Mapping

- Given a robot that has a perfect ego-estimate of the position, and a sequence of measurements, determine the map of the environment.
- A perfect estimate of the robot pose is usually not available.
- Instead we solve a more complex problem: Simultaneous Localization and Mapping (SLAM).

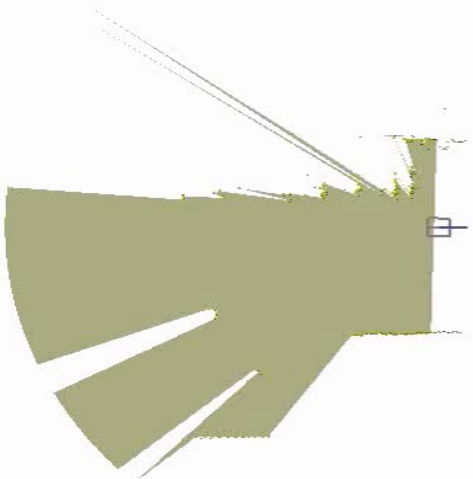
SLAM

- SLAM= Simultaneous Localization and Mapping
- Estimate:
  - the map of the environment
  - the trajectory of a moving device

these quantities are correlated

using a sequence of sensor measurements.

SLAM



Putting Parts Together

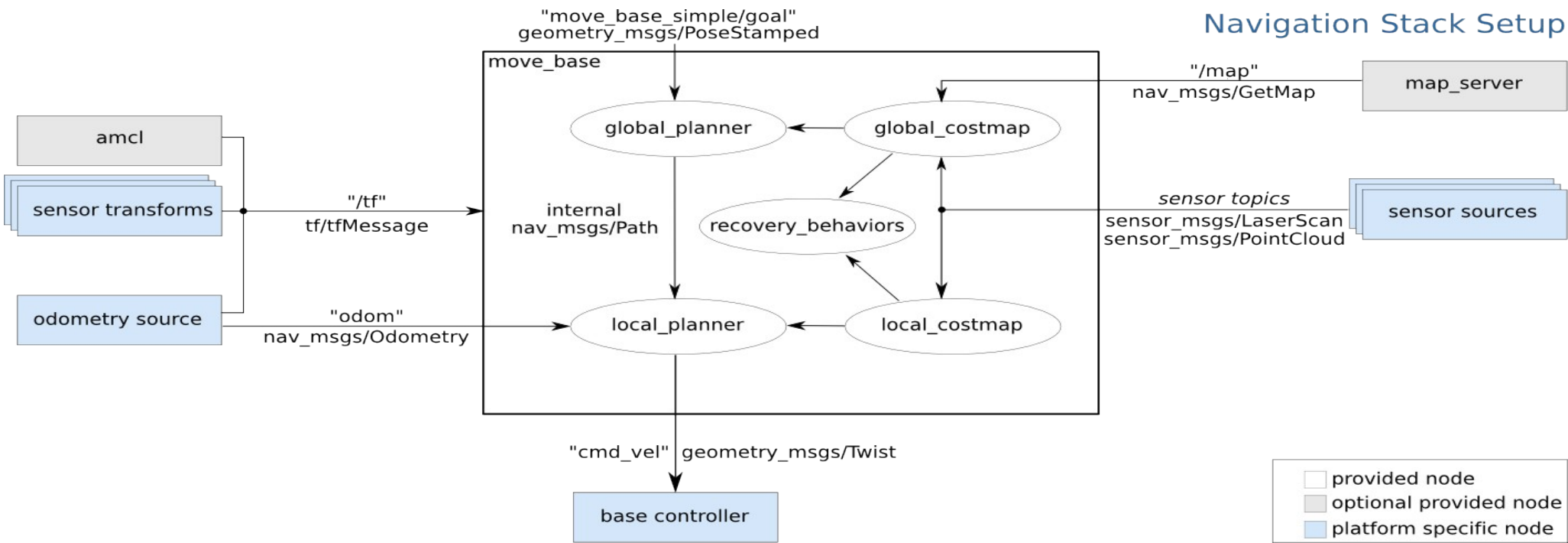
- To navigate a robot we need
  - A map
  - A localization module
  - A path planning module
- These components are sufficient if
  - The map fully reflects the environment
  - The environment is static
  - There are no errors in the estimate

- However
  - The environment changes (e.g. opening/closing doors)
  - It is dynamic (things might appear/disappear from the perception range of the robot)
  - The estimate is “noisy”

  Thus we need to complement our ideal design with other components that address these issues, namely
  - Obstacle-Detection/Avoidance
  - Local Map Refinement, based on the most recent sensor reading.

ROS Navigation Stack

- Map provided by a “Map Server”
- Each module is a node
- Planner has a layered architecture (local and global planner)
- Obstacle sensing refined on-line by appropriate modules (local and global costmap)



Building a Map

- There are gazillions of SLAM algorithms around. ROS by default uses GMapping, which implements a particle filter to track the robot trajectories.
- To build a map you need to
  - Record a bag with /odom, /scan/ and /tf while driving the robot around in the environment it is going to operate in
  - Play the bag and the gmapping-node (see the ros wiki and the live demo), and then save it.
- The map is an occupancy map and it is represented as
  - An image showing the blueprint of the environment
  - A configuration file (yaml) that gives meta information about the map (origin, size of a pixel in real world)

Localizing a Robot

- ROS standard navigation stack implements the Adaptive Monte Carlo Localization algorithm
  - AMCL uses a particle filter to track the position of the robot (see paper of Fox et al.)
  - Each pose is represented by a particle.
  - Particles are
    - Moved according to (relative) movement measured by the odometry
    - Suppressed/replicated based on how well the laser scan fits the map, given the position of the particle.
- The localization is integrated in ROS by emitting a transform from a map-frame to the odom frame that “corrects” the odometry.
- To query the robot position according to the localization you should ask the transform of base_footprint in the map frame.

Setting up a navigation Stack

Refer to the navigation tutorial

http://wiki.ros.org/navigation/Tutorials

In the practical:

- Recording data
- Building a map
- Starting localization
- Issue position commands

SRRG Ecosystem

- Alternative tools for navigation (and much more)
- You installed a snapshot with the labiagi repo

  In the remainder:
  - Building a map with srrg tools
  - Setting up a static-world navigation system
  - Setting up a dynamic-world navigation system

SRRG components

- Class loader and shell (srrg2_executor)
- Configuration editor (srrg2_config visualizer)
- Web Launcher (proc_webctl)

SRRG Shell

- Implements a running environment for the modules
- Loads a set of dynamic libraries on start (from a file dl.conf, by default)
- Allows to create/edit/start/stop/pause processing pipelines
- Allows for attaching viewers to modules

  to start

  rosrun srrg2_executor srrg2_shell

  [-vt shared] [-c <config_file>] [script]

  Type <help> for commands
- There are no “main files”, but pipelines

SRRG Config Visualizer

- Graphical editor of configurations
- Allows to edit/construct a config file
  - rosrun srrg2_config_visualizer app_node_editor -c config_file

SRRG Processing Modules

- They exist in the packages
- A processing component boils into a dynamic library that is loaded on startup

SRRG map server

- Pure ROS node
- Loads a map from a .yaml file and provides it to the rest of the nodes

  srrg2_map_server map_server <map file>

SRRG localizer

- Runs as shell component
- Script specified in run_localizer_live_ms.srrg

  (show file)

  (show configuration with config editor)

- rosrun srrg2_shell run_localizer_live_ms.srrg

SRRG planner

- Implements a complete motion planner (easier replacement of move-base)
- Publishes two paths
  - /path global
  - /local_path (local)
- Implements an anytime reaction layer that refines the local path
- Script: run_planner_live_ms.srrg

Laser SLAM

- A laser slam pipeline

  The configuration exposes stuff to run both live and from

  run_cappero_diag_slam.srrg (for bag)

  Remember to save the map at the end

Path Follower

- Pure ROS node
- Reads a path, and executes it (turns it to /cmd_vel for platform)

  srrg2_navigation_2d_ros path_follower_app

Proc Webctl

- Reads a config file to start a pool of processes, monitors them and spawns a web server for accessing them

  (for ubuntu 20.04, switch to 20.04 branch)

  proc_webctl <config file>

Joystick

- Pure ROS node
- Reads the joystick and issues /cmd_vel to the mobile base

  srrg2_joystick_teleop joy_teleop_node

Exercise

- Generate a map from the provided bag
- Generate a map from an own recorded bag (by joysticking the robot on stage)
- Bring up a navigation stack based on your new map, and verify it works with rviz
- Write a simple node that
  - receives a “goal” as x,y,theta
  - Upon commad, issues a command to reach a position to the planner and “waits” the robot to reach the location.
  - The program monitors the execution of the command, by checking the position of the robot. The goal is reached when the robot location is “close enough” to the goal. Use the TF
    - If the robot does not move for a certain amount <T_lock> of time and the goal is not reached, the program prints an error message
    - If the robot does not reach the goal within a certain amount of time <T_long>, the program prints another error message and stops
- From Rviz set a goal position on the map (**2D nav goal** in the toolbar)
- Hints :
  - *Give to the system a good initial guess about the robot location*