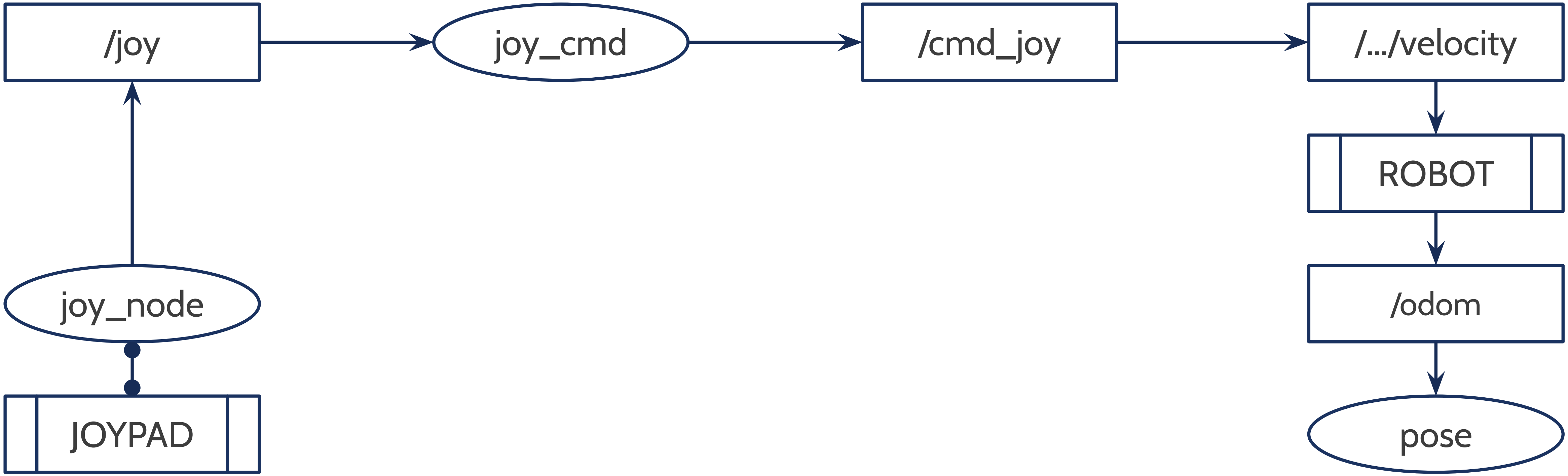
ROBOT NAVIGATION

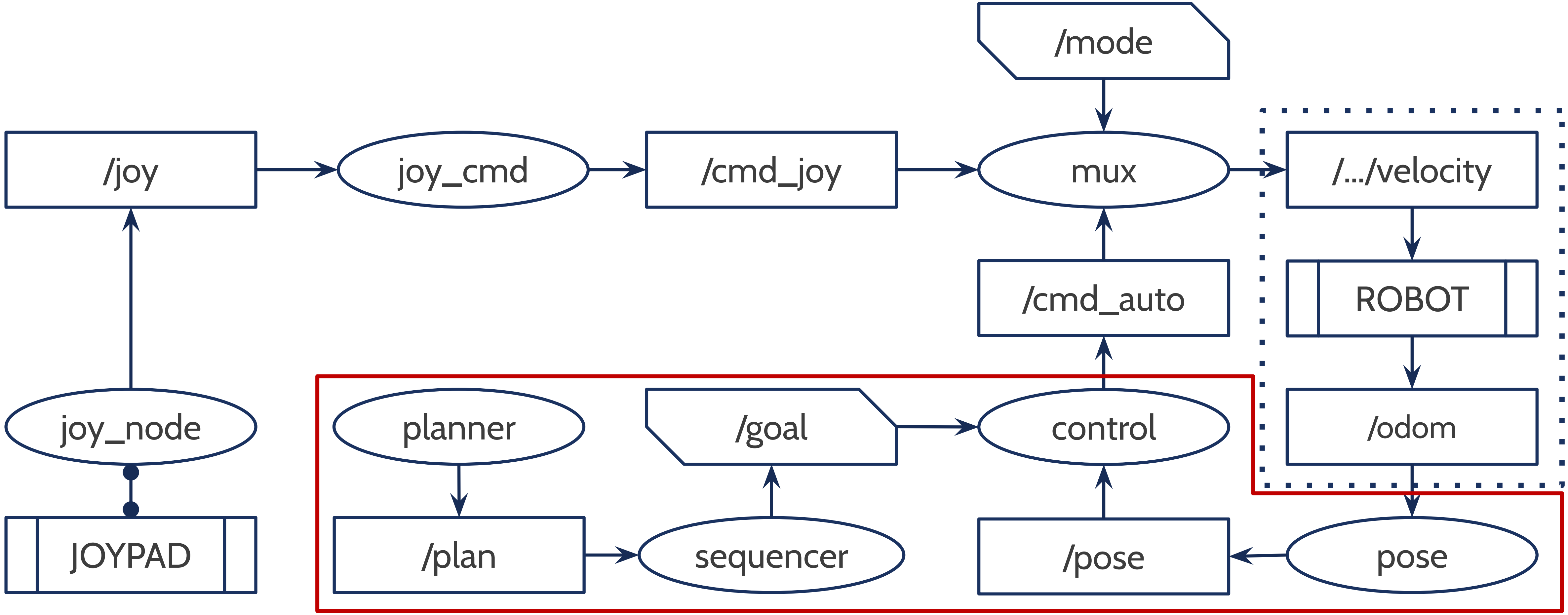
ROBOTICS

POLITECNICO
MILANO 1863

OUR IMPLEMENTATION

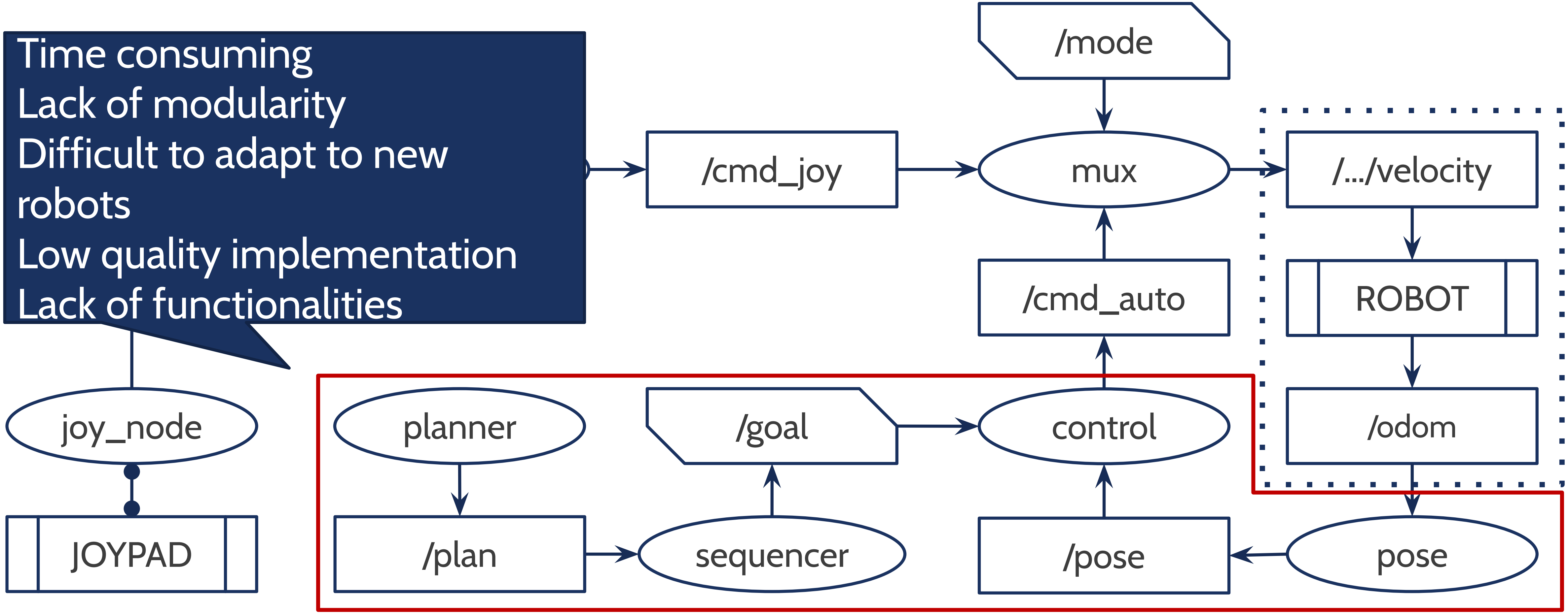


OUR IMPLEMENTATION



OUR IMPLEMENTATION

Time consuming
Lack of modularity
Difficult to adapt to new robots
Low quality implementation
Lack of functionalities



SOLUTION?

Exploit the greatest quality of ROS

already available and implemented components

SOLUTION?

Exploit the greatest quality of ROS

already available and implemented components

↓

ROS **navigation** (stack)

http://wiki.ros.org/navigation

NAVIGATION

move_base

nav_core

amcl

robot_pose_ekf

base_local_planner

carrot_planner

dwa_local_planner

navfn

global_planner

move_slow_and_clear

rotate_recovery

clear_costmap_recovery

costmap_2d

map_server

voxel_grid

fake_localization

move_base_msgs

NAVIGATION

Central element of *navigation* and the definition of the base class

move_base

nav_core

amcl

robot_pose_ekf

base_local_planner

carrot_planner

dwa_local_planner

navfn

global_planner

w_and_clear

overy

clear_costmap_recovery

costmap_2d

map_server

voxel_grid

fake_localization

move_base_msgs

NAVIGATION

move_base

nav_core

amcl

robot_pose_ekf

base_local_planner

carrot_planner

dwa_local_planner

navfn

global_planner

move_slow_and_clear

rotate_recovery

map_recovery

2d

map_server

voxel_grid

fake_localization

move_base_msgs

Robot localization using various methods

NAVIGATION

move_base

nav_core

amcl

robot_pose_ekf

base_local_planner

carrot_planner

dwa_local_planner

navfn

global_planner

move_slow_and_clear

rotate_recovery

clear_costmap_recovery

costmap_2d

move_base_msgs

Different algorithms to implement local autonomous movement

NAVIGATION

move_base

nav_core

amcl

robot_pose_ekf

base_local_planner

carrot_planner

dwa_local_planner

navfn

global_planner

move_slow_and_clear

rotate_recovery

clear_costmap_recovery

costmap_2d

map_server

voxel_grid

fake_localization

e_msgs

Global planner used to generate the trajectory on a large scale

NAVIGATION

move_base

nav_core

amcl

robot_pose_ekf

base_local_planner

carrot_planner

dwa_local_planner

navfn

global_planner

Various recovery behavior for stuck robots or critical situations

move_slow_and_clear

rotate_recovery

clear_costmap_recovery

costmap_2d

map_server

voxel_grid

fake_localization

move_base_msgs

NAVIGATION

move_base

nav_core

amcl

robot_pos

base_loca

carrot_pla

dwa_local_planner

navfn

global_planner

move_slow_and_clear

rotate_recovery

clear_costmap_recovery

costmap_2d

map_server

voxel_grid

fake_localization

move_base_msgs

Tools for 2D and 3D map representation

NAVIGATION

move_base

nav_core

amcl

robot_pose_ekf

base_local_planner

carrot_pla

dwa_local

navfn

global_planner

move_slow_and_clear

rotate_recovery

clear_costmap_recovery

costmap_2d

map_server

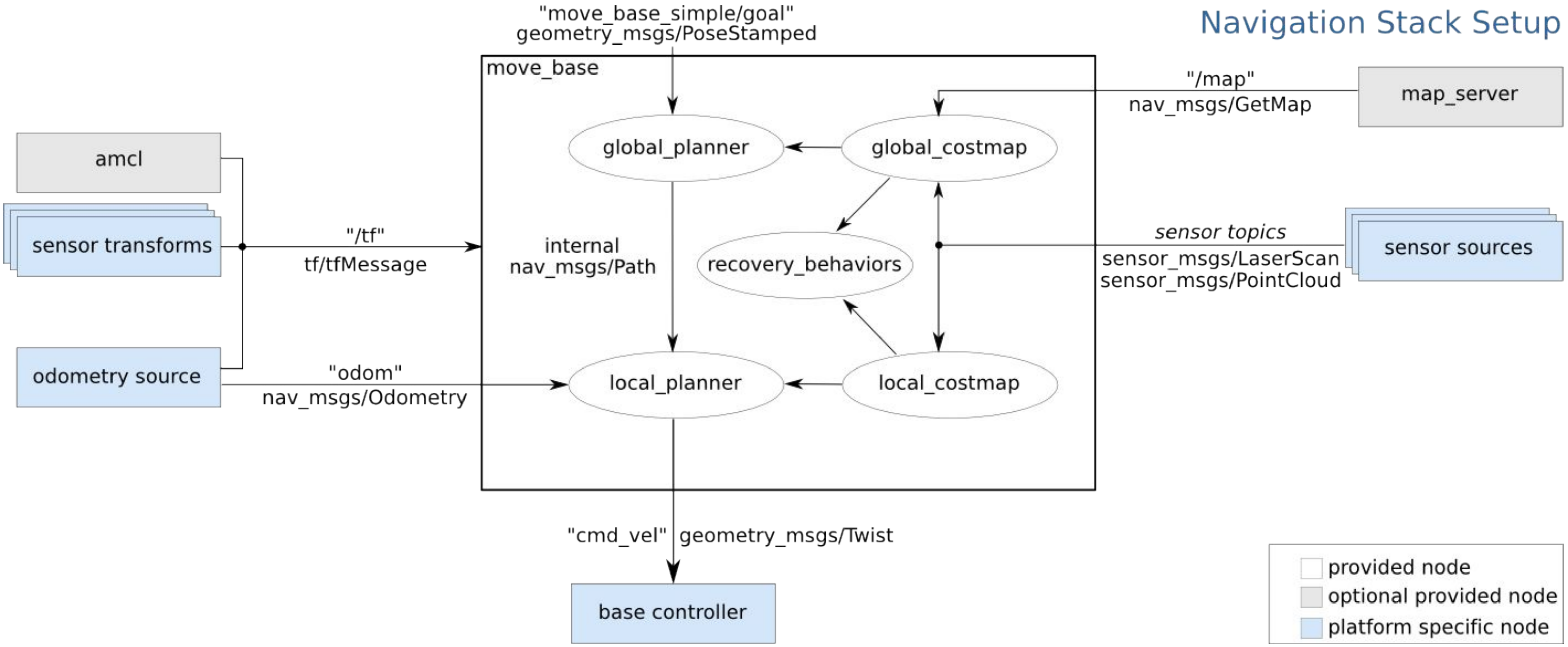
voxel_grid

fake_localization

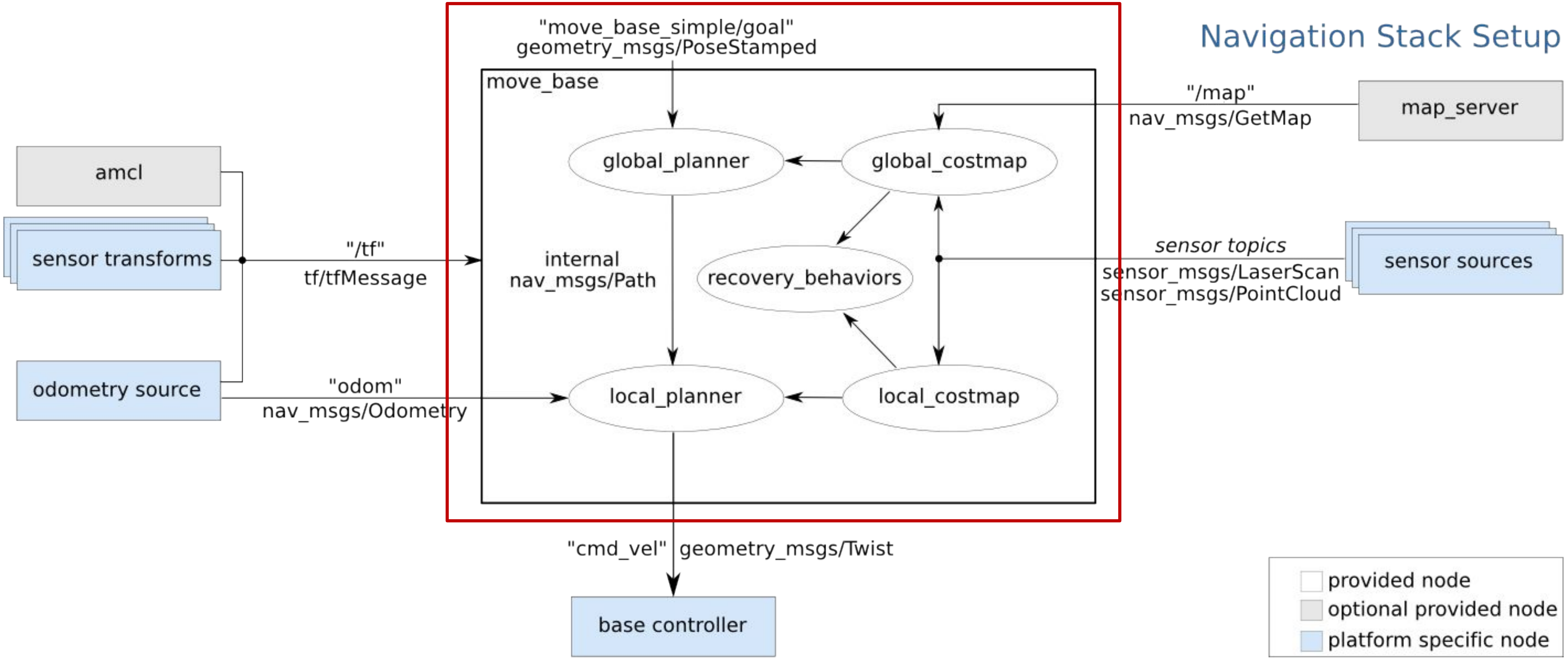
move_base_msgs

Extra utilities for testing and communication

GENERAL ARCHITECTURE



MOVE_BASE



Navigation Stack Setup
"move_base_simple/goal"
geometry_msgs/PoseStamped
move_base
global_planner
global_costmap
"/map"
nav_msgs/GetMap
map_server
amcl
sensor transforms
"/tf"
tf/tfMessage
internal
nav_msgs/Path
recovery_behaviors
sensor topics
sensor_msgs/LaserScan
sensor_msgs/PointCloud
sensor sources
odometry source
"odom"
nav_msgs/Odometry
local_planner
local_costmap
"cmd_vel"
geometry_msgs/Twist
base controller
provided node
optional provided node
platform specific node

MOVE_BASE

Single node and ***core element*** of ROS navigation.

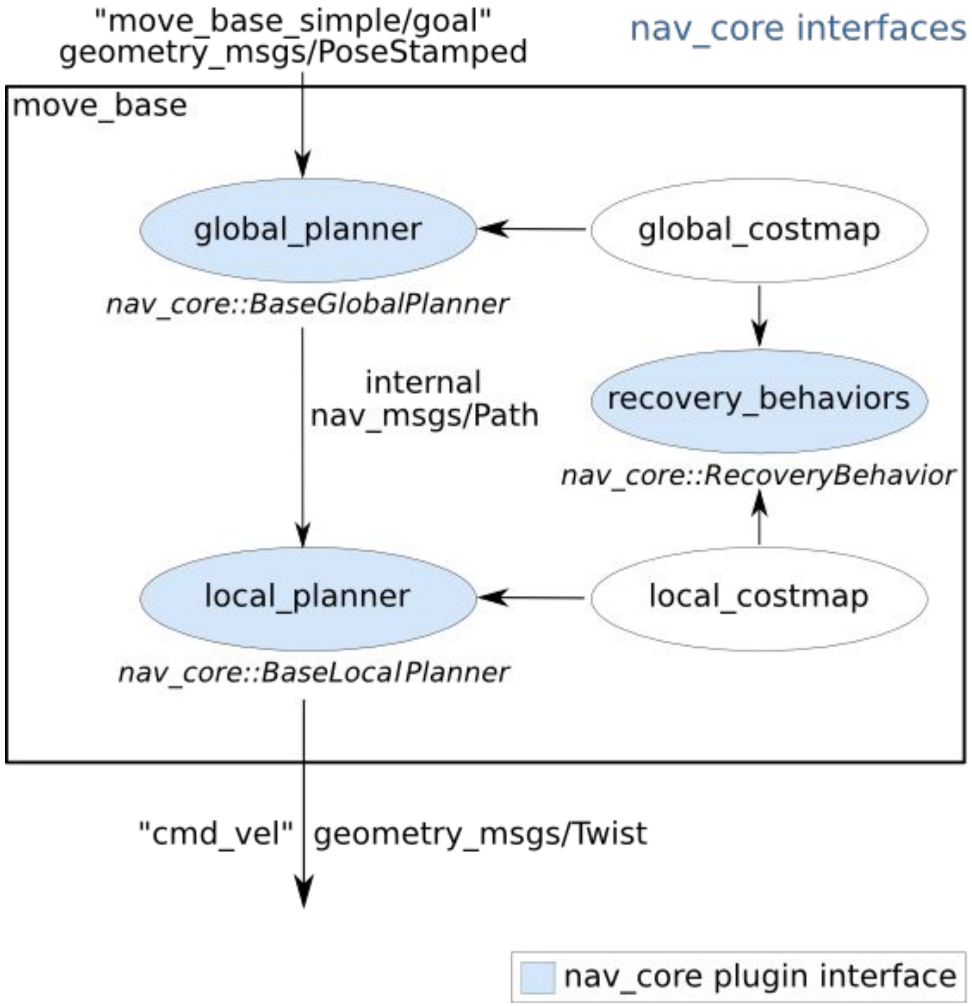
Implements all the main planning and control functionalities

based on plugins for dynamic configuration.

Easy to extend via ROS ***pluginlib***.

Based on the ***nav_core*** class.

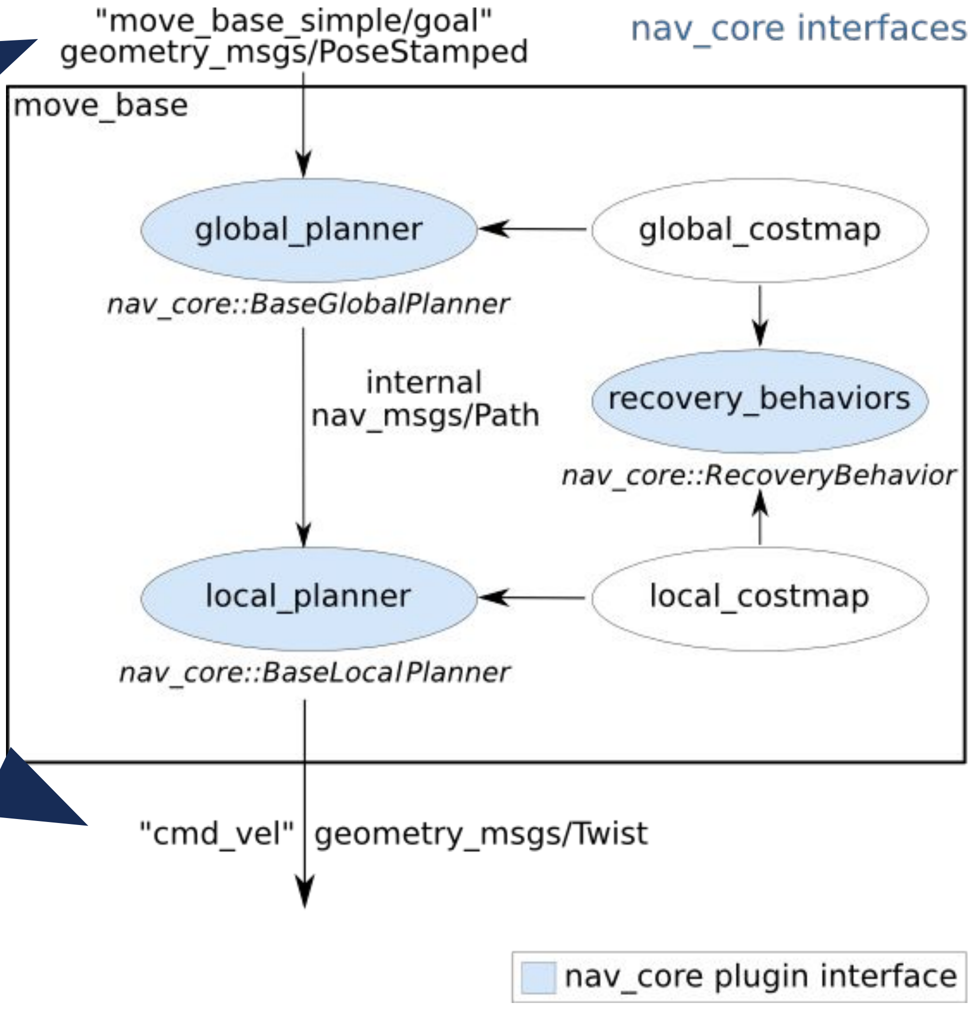
NAV_CORE



"move_base_simple/goal"
geometry_msgs/PoseStamped
nav_core interfaces
move_base
global_planner
global_costmap
nav_core::BaseGlobalPlanner
internal
nav_msgs/Path
recovery_behaviors
nav_core::RecoveryBehavior
local_planner
local_costmap
nav_core::BaseLocalPlanner
"cmd_vel"
geometry_msgs/Twist
nav_core plugin interface

NAV_CORE

Goal as a single point via topic or actions

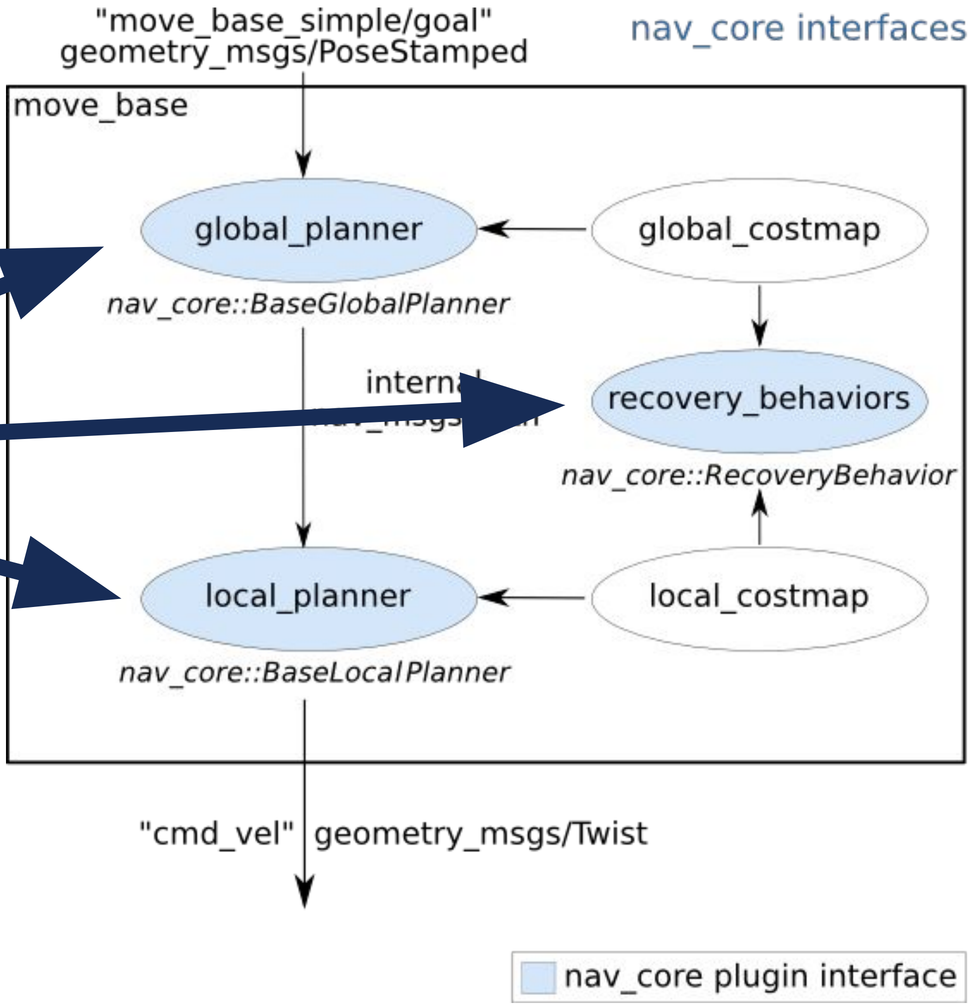


Velocity command via topic

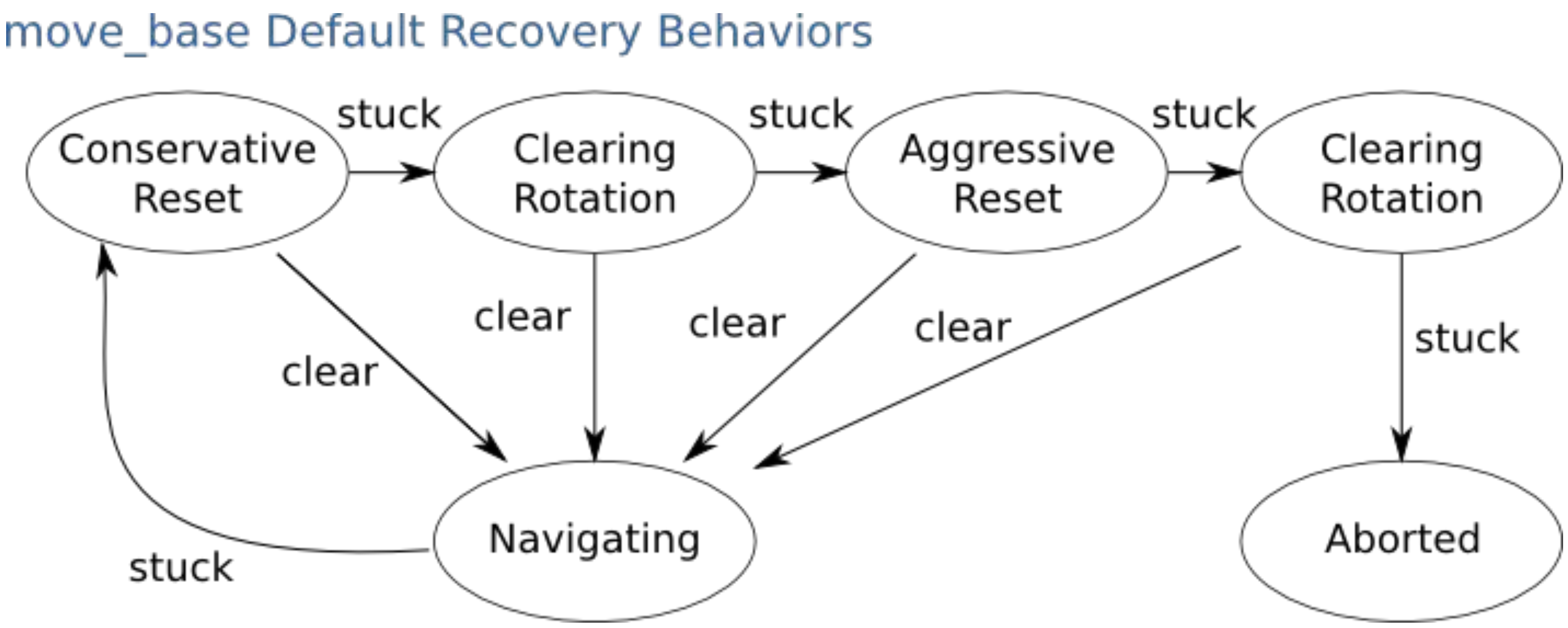
NAV_CORE

Plugins implement functionalities

Exchangeable at execution time

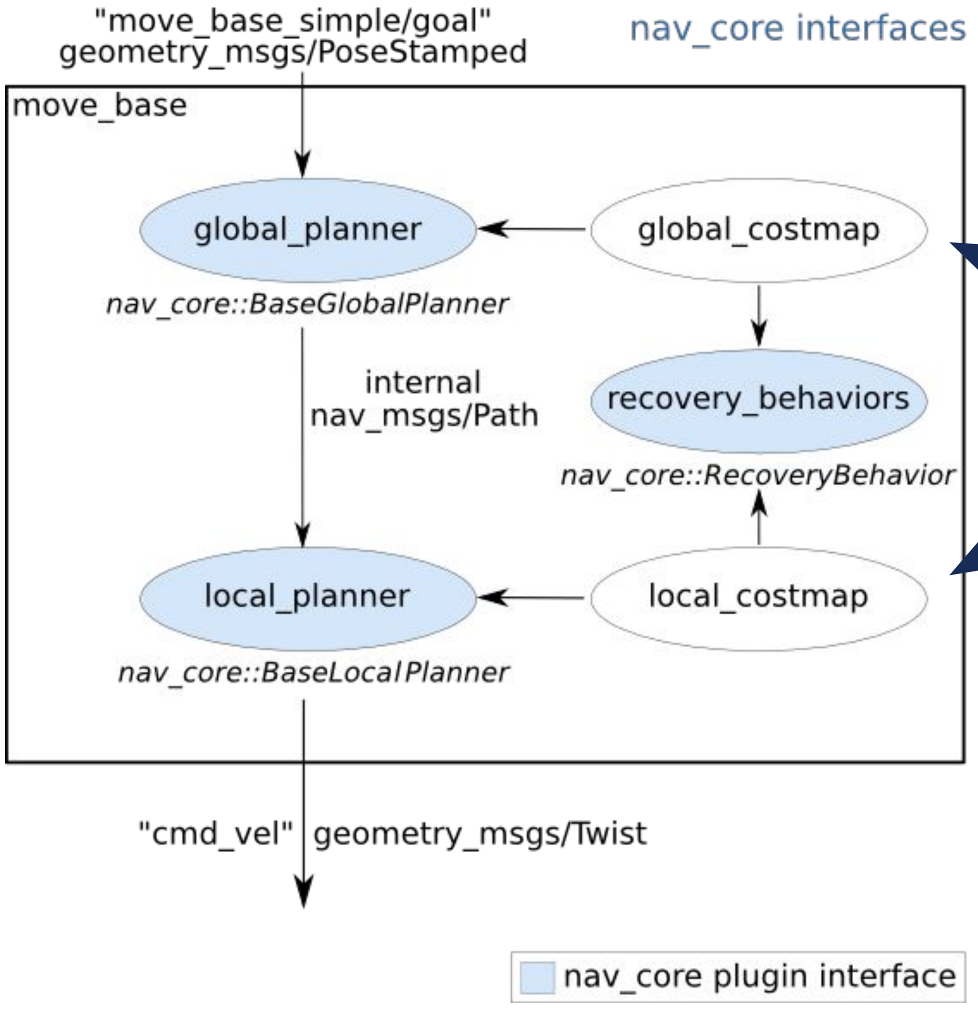


NAV_CORE



move_base Default Recovery Behaviors
Conservative Reset
stuck
Clearing Rotation
stuck
Aggressive Reset
stuck
Clearing Rotation
clear
clear
clear
clear
stuck
Navigating
Aborted
stuck

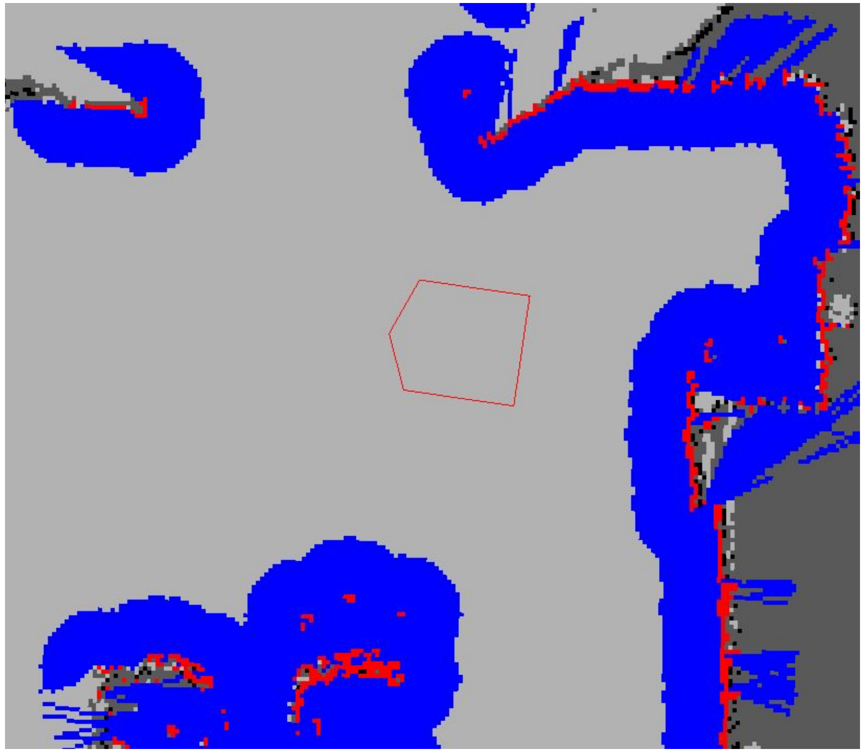
NAV_CORE



Information about the world provided by the map server and the sensors

COST MAP

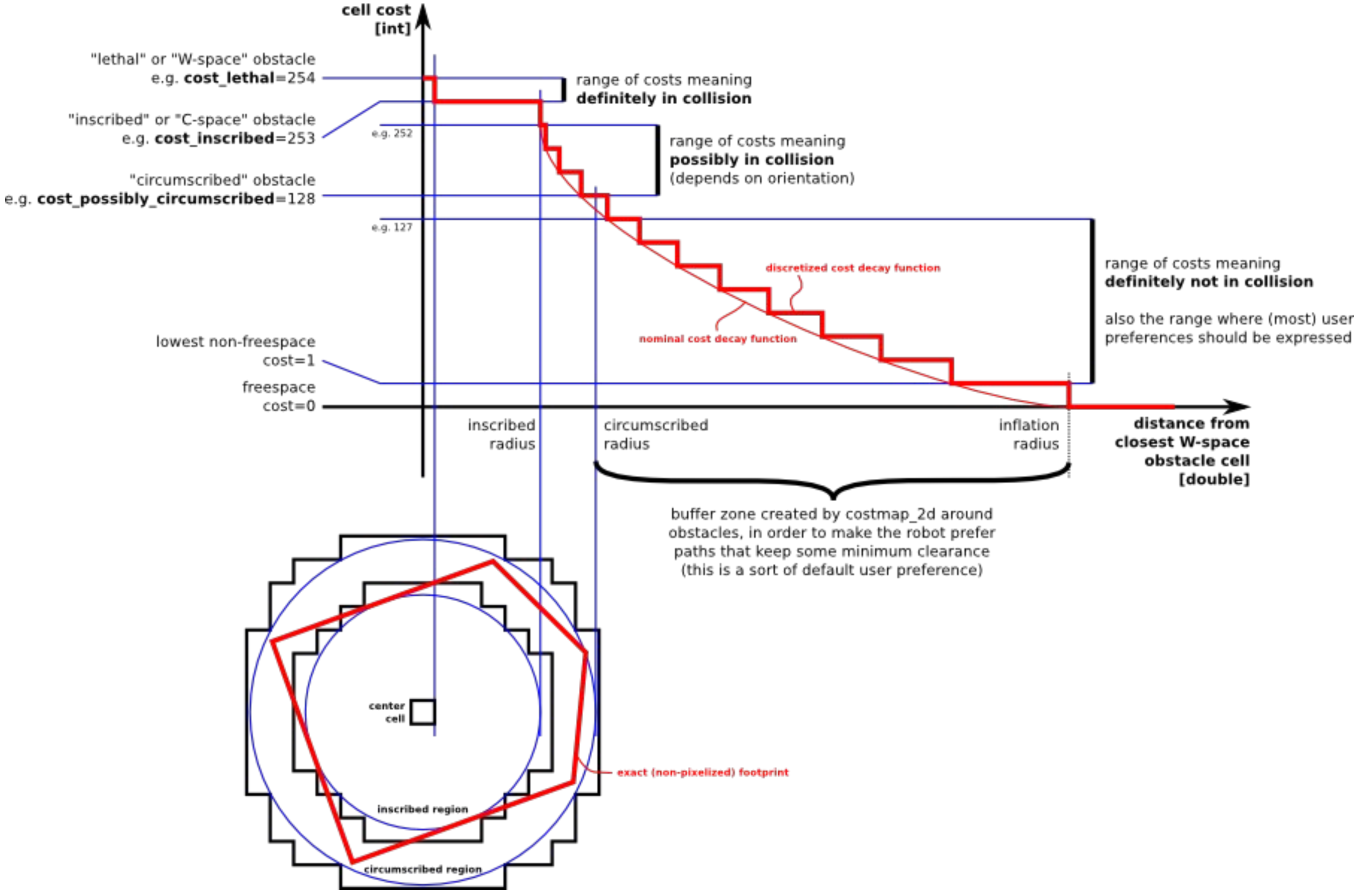
Takes in sensor data and builds a 2D or 3D occupancy grid of the data



COST MAP

Each cell can have one of 255 different cost values

Inflates costs



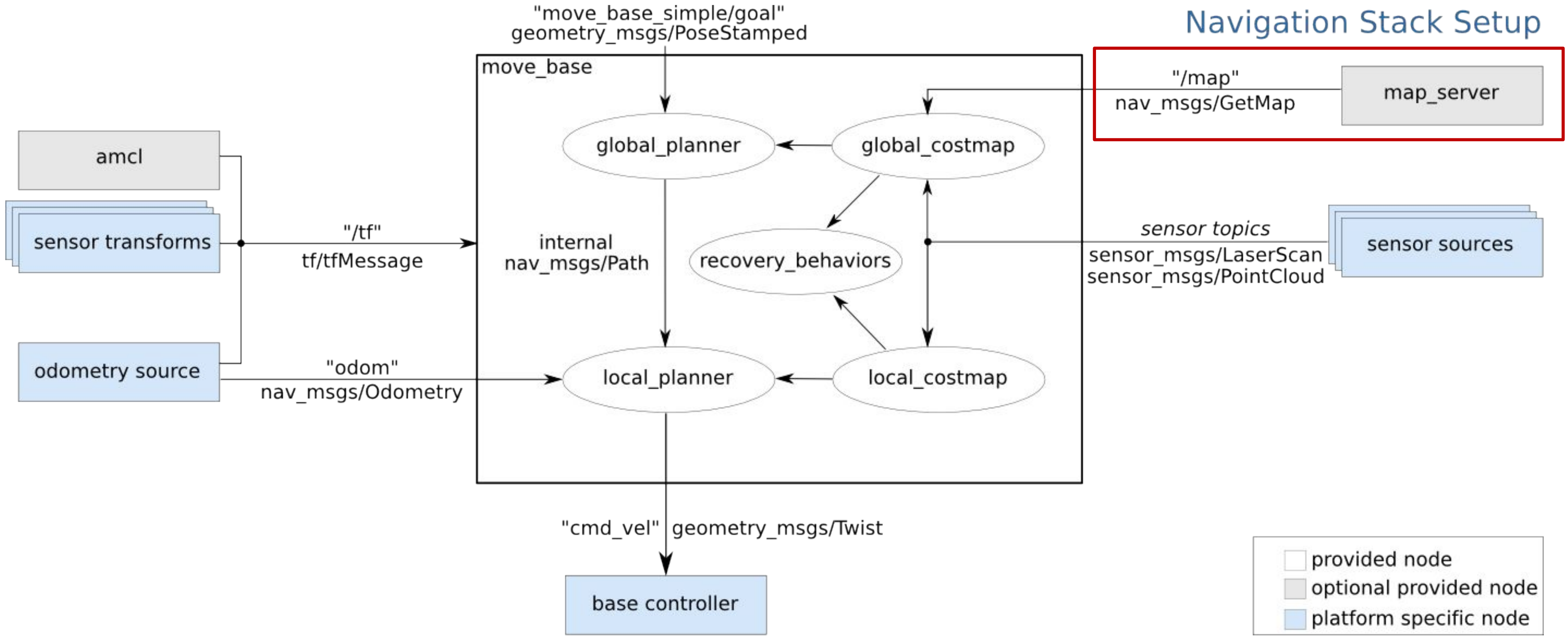
COST MAP

ROS Navigation is based on two different costmaps:

- Global: used for long-term plans over the entire environment
- Local: used for local planning and obstacle avoidance

These costmaps have specific and common configurations

MAP_SERVER


Navigation Stack Setup
"move_base_simple/goal"
geometry_msgs/PoseStamped
move_base
"/map"
nav_msgs/GetMap
map_server
amcl
global_planner
global_costmap
sensor transforms
"/tf"
tf/tfMessage
internal
nav_msgs/Path
recovery_behaviors
sensor topics
sensor_msgs/LaserScan
sensor_msgs/PointCloud
sensor sources
odometry source
"odom"
nav_msgs/Odometry
local_planner
local_costmap
"cmd_vel"
geometry_msgs/Twist
base controller
provided node
optional provided node
platform specific node

MAP_SERVER

Tool provided by ROS navigation to publish and save maps.

Offers the map both via topic and via service.

Can save dynamically generated maps.

Combined with costmap_2d:

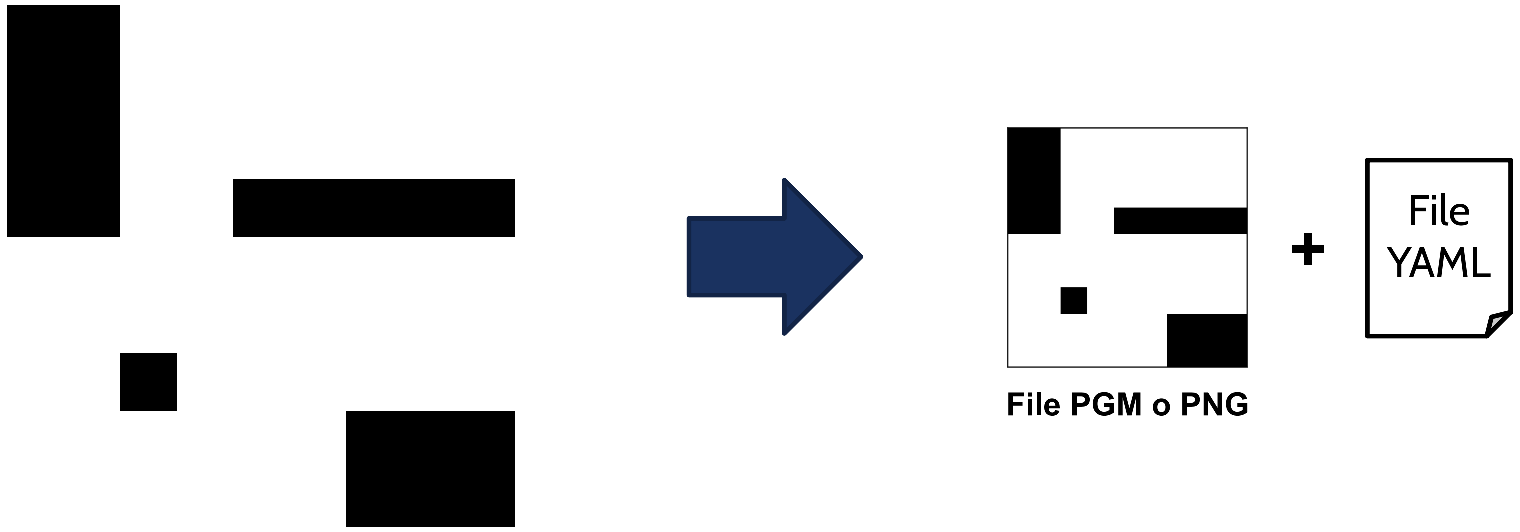
Manages multi-layered 2D maps.

Inflate obstacle according to sensor information.

MAP_SERVER

The map is composed by:

- YAML file: describes the map meta-data
- Image file: encodes the occupancy data



MAP_SERVER

maze.yaml

image: maze.png — Path to the image file containing the occupancy data

resolution: 0.05 — Resolution of the map, meters / pixel

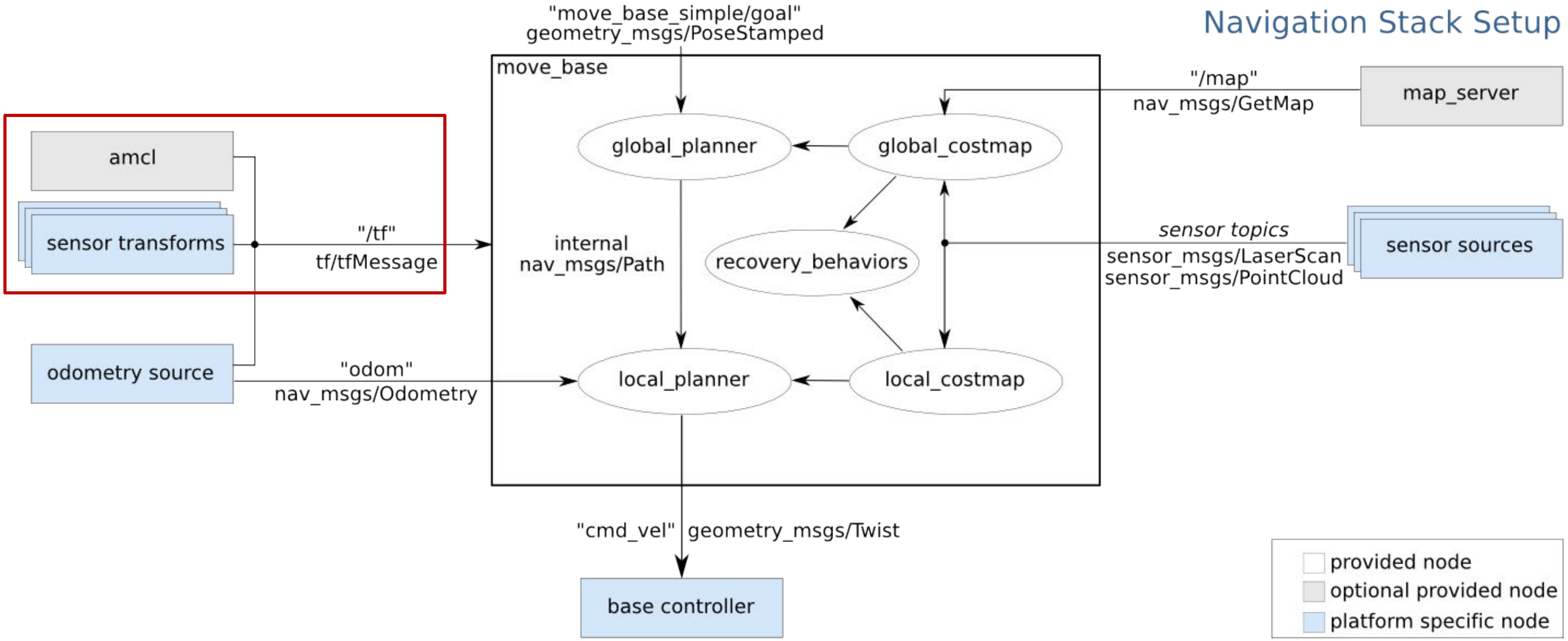
origin: [0.0, 0.0, 0.0] — The 2-D pose of the lower-left pixel in the map, as (x, y, yaw)

negate: 0 — The white/black free/occupied semantics should be reversed

occupied_thresh: 0.65 — Pixels with occupancy probability greater than this threshold are considered completely occupied

free_thresh: 0.196 — Pixels with occupancy probability less than this threshold are considered completely free

AMCL


Navigation Stack Setup
"move_base_simple/goal"
geometry_msgs/PoseStamped
move_base
global_planner
global_costmap
recovery_behaviors
local_planner
local_costmap
internal
nav_msgs/Path
"/map"
nav_msgs/GetMap
map_server
amcl
sensor transforms
"/tf"
tf/tfMessage
sensor topics
sensor_msgs/LaserScan
sensor_msgs/PointCloud
sensor sources
odometry source
"odom"
nav_msgs/Odometry
"cmd_vel"
geometry_msgs/Twist
base controller
provided node
optional provided node
platform specific node

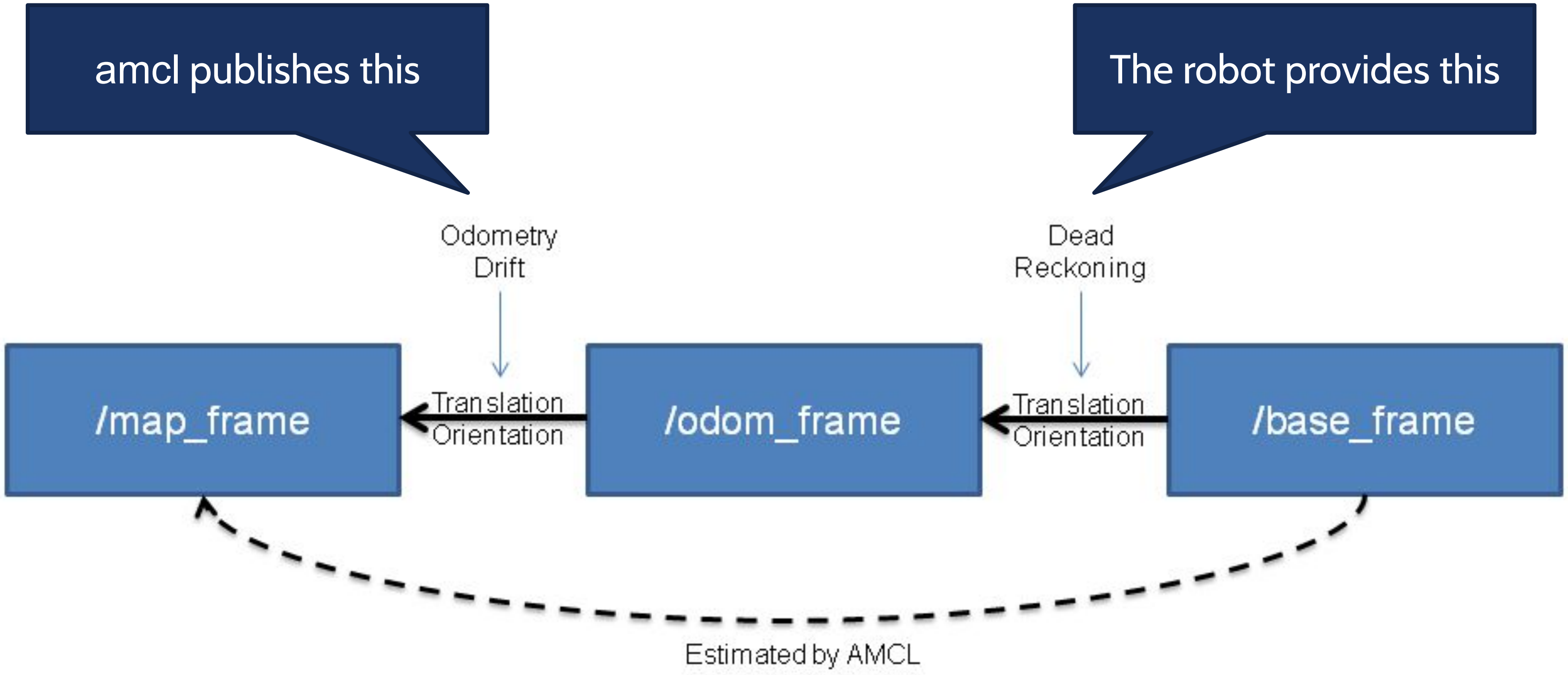
AMCL

Probabilistic localization system based on a 2D map.

Provides the estimated position using future dated tf.

Requires a laser scan and provides better result when using odometry.

AMCL (TRANSFORMATION FRAMES)



AMCL (TRANSFORMATION FRAMES)

Transforms incoming laser scans to the odometry frame

- → It requires a path from /base_scan to /odom

Estimates the position of the robot in the global frame

- → Transformation between /map and /base_link

Publishes the transformation between the global frame and the odometry frame

- → Transformation between /odom and /map
- → Correct the odometry drift

AMCL

Acml parameters

min_particles: 500
max_particles: 2000

Minimum/Maximum allowed number of particles.

update_min_d: 0.25
update_min_a: 0.2

Translational and rotational movement required before performing a filter update

resample_interval: 1

Number of filter updates required before resampling

initial_pose_x: 2.0
initial_pose_y: 2.0
initial_pose_a: 0.0

Initial pose mean (x, y, yaw), used to initialize filter with Gaussian distribution.

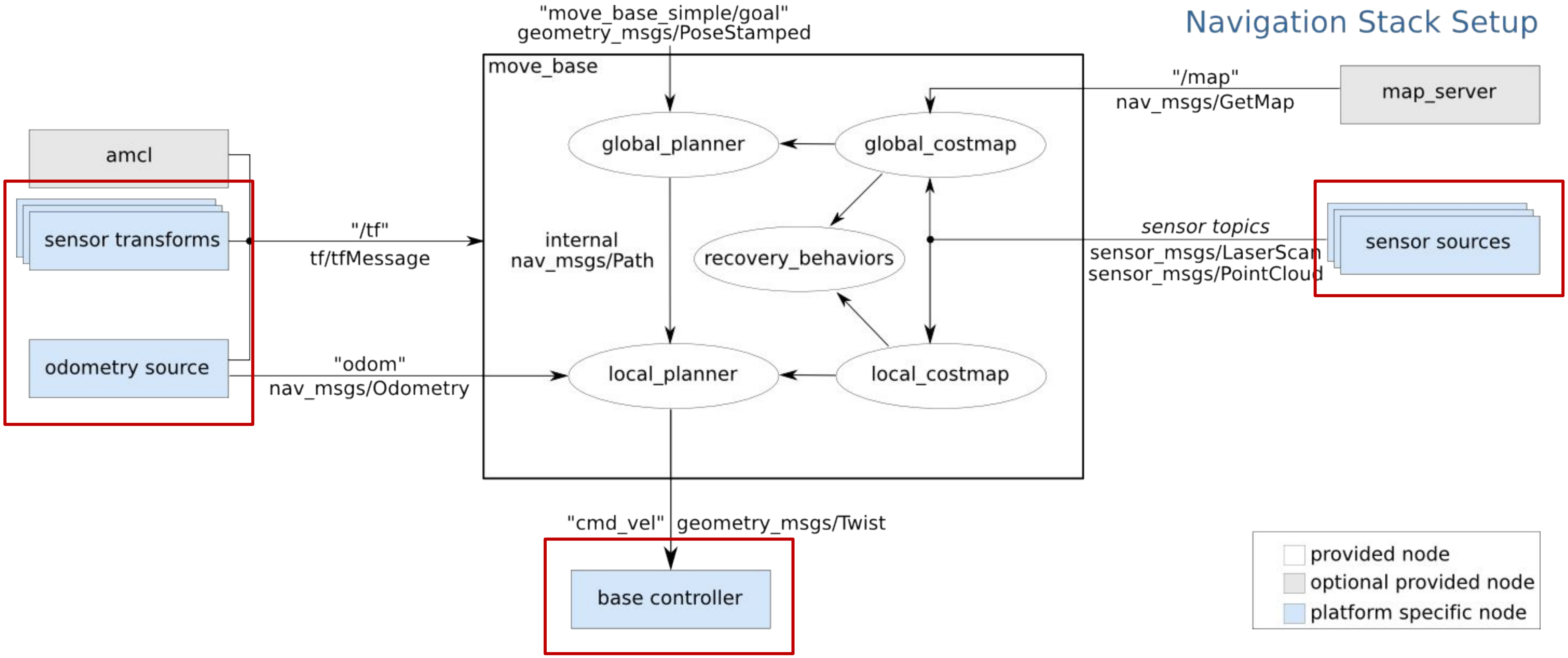
odom_model_type: "diff"

Model to use, either "diff", "omni"

odom_frame_id: "odom"
base_frame_id: "base_footprint"
global_frame_id: "map"

Frame to use for odometry, robot_base and for the localization system

WHAT'S MISSING?



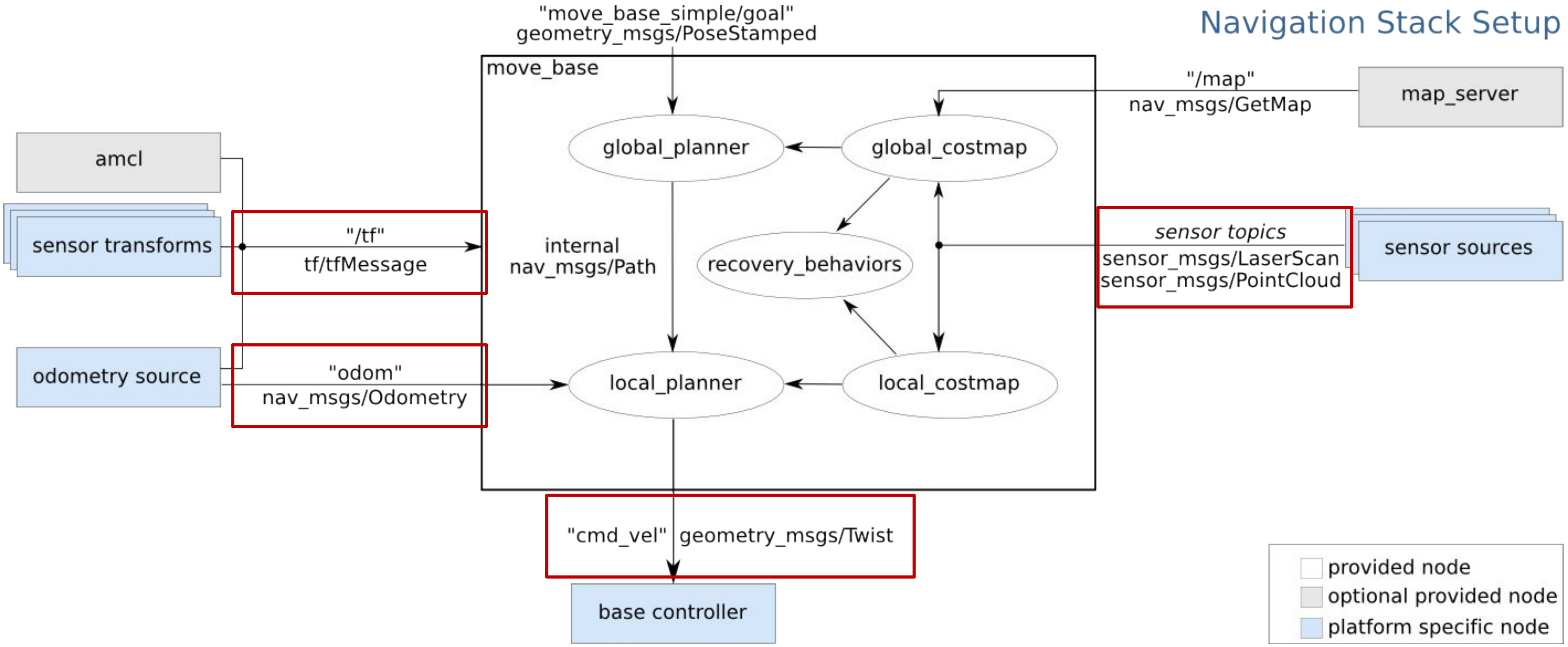
WHAT'S MISSING?

Everything platform specific need to be implemented by hand:

- Low-level robot interaction
- Sensor drivers
- Sensor measurements processing
- Odometry estimation
- High-level task planning

Most of these are already available in ROS as existing packages (i.e., drivers, robot_pose_ekf, …)

ROS NAV REQUIREMENTS



ROS NAV REQUIREMENTS

ROS Navigation has a specific architecture and needs some specific condition to work:

- Sensor source to localize and avoid obstacle, as sensor_msgs/LaserScan or sensor_msgs/PointCloud
- A source of odometry, as nav_msgs/Odometry
- Conversion from geometry_msgs/Twist to motor control
- A well formed tf tree (sensors position, robot position and map)

ROS NAV REQUIREMENTS

The ROS Navigation is quite general and adaptable, but it has a few hardware requirements:

- Works better with differential drive or holonomic robots
- Requires a planar laser for scanning and localization
- Best results with square or circular robots

ROSBAG

Is a set of tools for recording from and playing back to ROS topics

This is the current list of supported commands:

record: Record a bag file with the contents of specified topics.

info: Summarize the contents of a bag file.

play : Play back the contents of one or more bag files.

check: Determine whether a bag is playable in the current system, or if it can be migrated.

fix: Repair the messages in a bag file so that it can be played in the current system.

filter: convert a bag file using Python expressions.

compress: compress one or more bag files.

decompress: decompress one or more bag files.

reindex: reindex one or more broken bag files

ROSBAG COMMAND

Record a bag:

rosbag record (-a | <topic name>)

Records all the topics

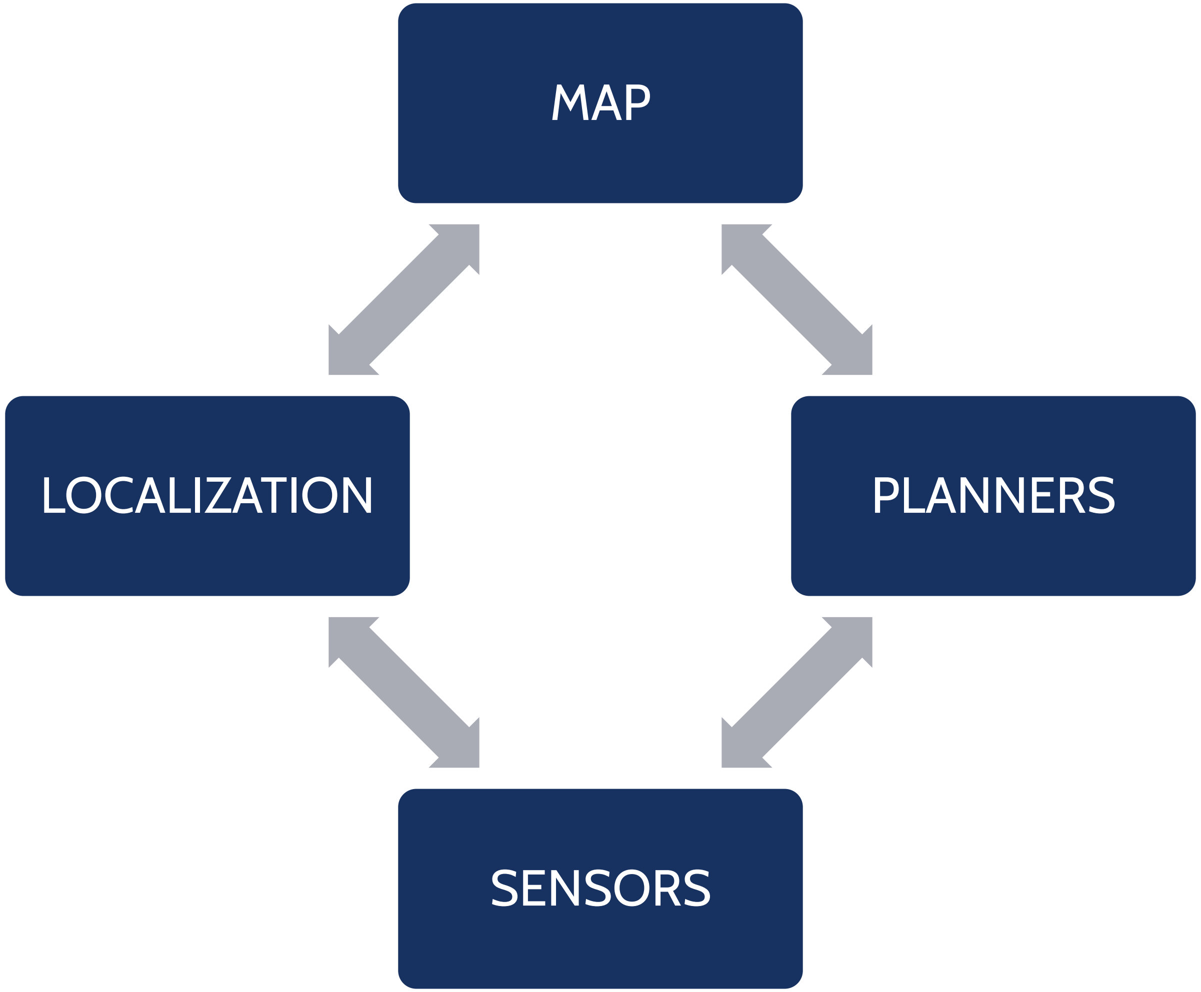
Records only specific topics

Play a bag

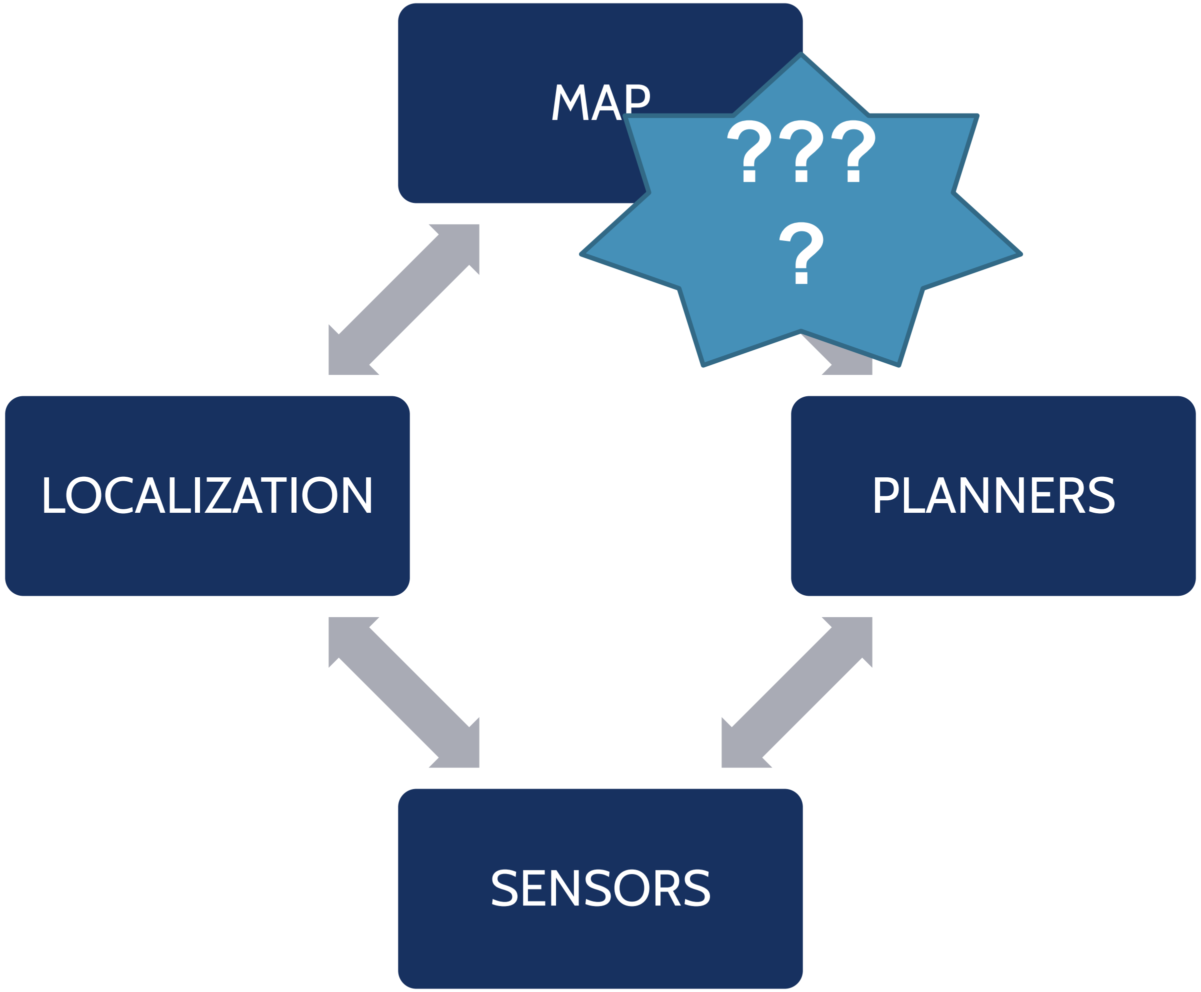
Use a simulated time

rosbag play --clock <name_of_the_bag>

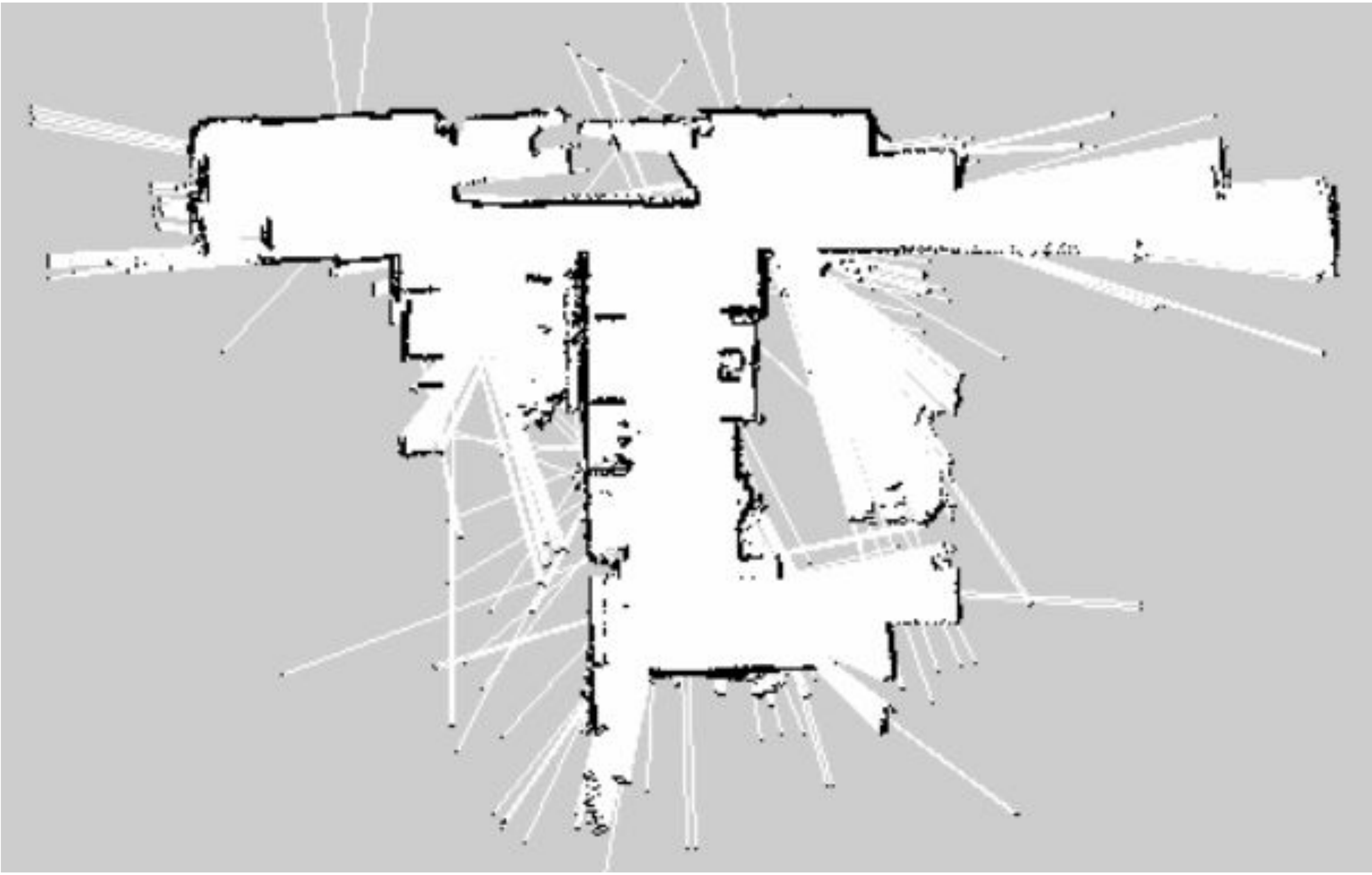
NAVIGATION MAIN ELEMENTS



NAVIGATION MAIN ELEMENTS



GMAPPING



GMAPPING

ROS wrapper for openslam gmapping

Actually a SLAM algortithm

Can be used for real time map creation and localization

Based on lasers and odometry

REQUIREMENTS

- Odometry
- Horizontally-mounted, fixed, laser range-finder
- Full tf tree with:
  - Base to laser transformation
  - Base to odometry transformation

IMPORTANT PARAMETERS

base_frame (string, default: "base_link")	the frame attached to the mobile base
map_frame (string, default: "map")	the frame attached to the map
odom_frame (string, default: "odom")	the frame attached to the odometry system

Also, topics to remap

scan (sensor_msgs/LaserScan)	laser scans to create the map from
map (nav_msgs/OccupancyGrid)	get the map data from this topic

HOW TO USE IT

1. Drive your robot around
   1. Explore all the area you want to map
   2. Try to collect as much data as possible
   3. Try to make loops and give the algorithm references
2. Save everything in a bag
3. Run the bag
4. Start gmapping and let it crunch the data
5. Save the generated map

You can skip this and run the gmapping node in real time

BAG VS REAL TIME

Using a bag

Faster

Can use data already collected

Can do different trials

Tune parameters

Processing in real time

Early stop if something goes wrong

Restart in case of problems

Can see directly the results

Assure full coverage

SOME EXAMPLES

Let's see it in practice!

ROBOT SIMULATORS

ROBOTICS

POLITECNICO
MILANO 1863

STAGE

-download from drive the folder called “stage”

-cd to the stage folder you downloaded

-to start the simulation simply use the command:

$ stage maze.world

if we want to control the robot we need to start it as a ROS node:

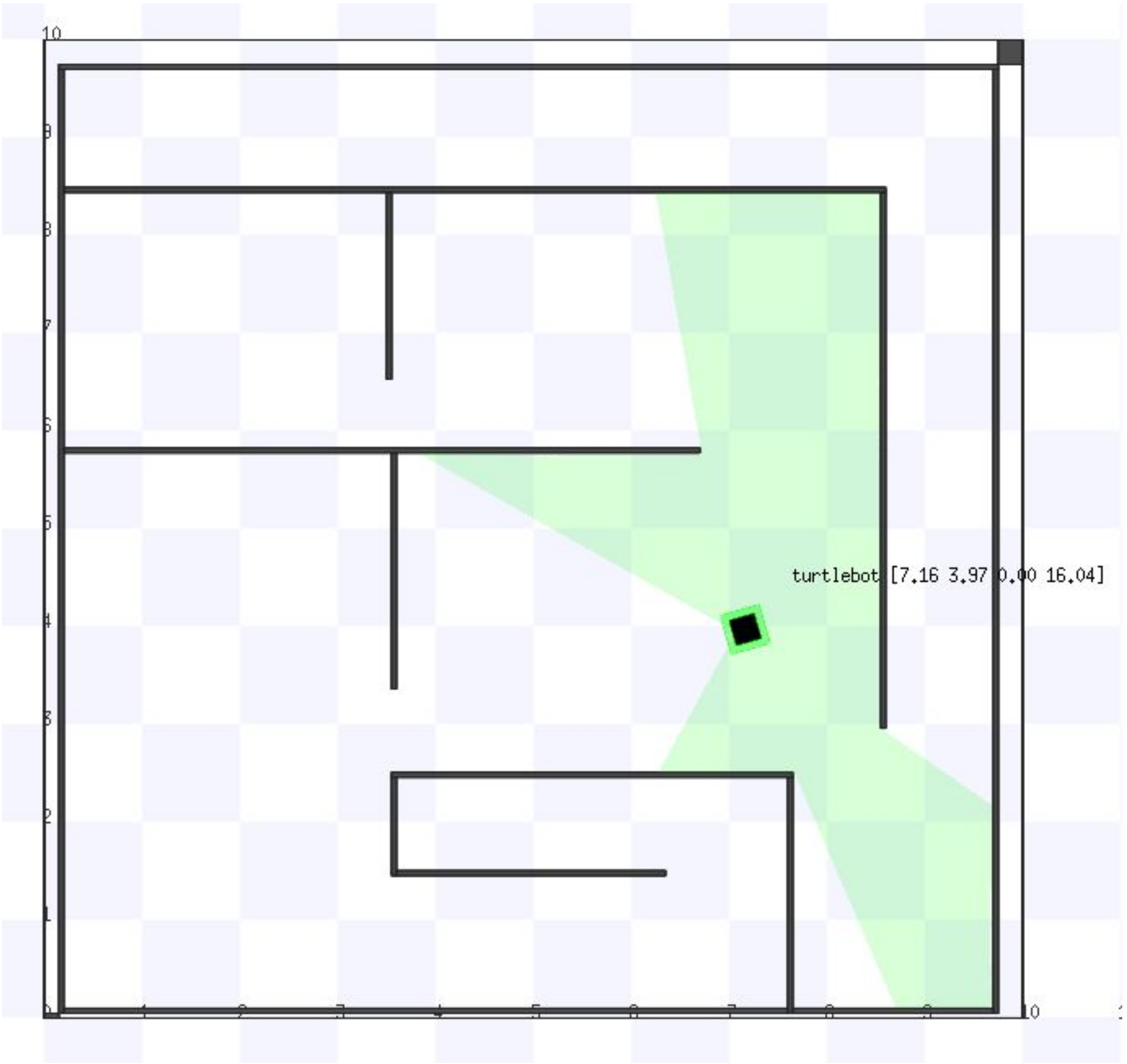
$ rosrun stage_ros stageros maze.world

STAGE

-to control the robot we can use any node publishing /cmd_vel:

$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

but before we need to export:
export TURTLEBOT3_MODEL="burger"

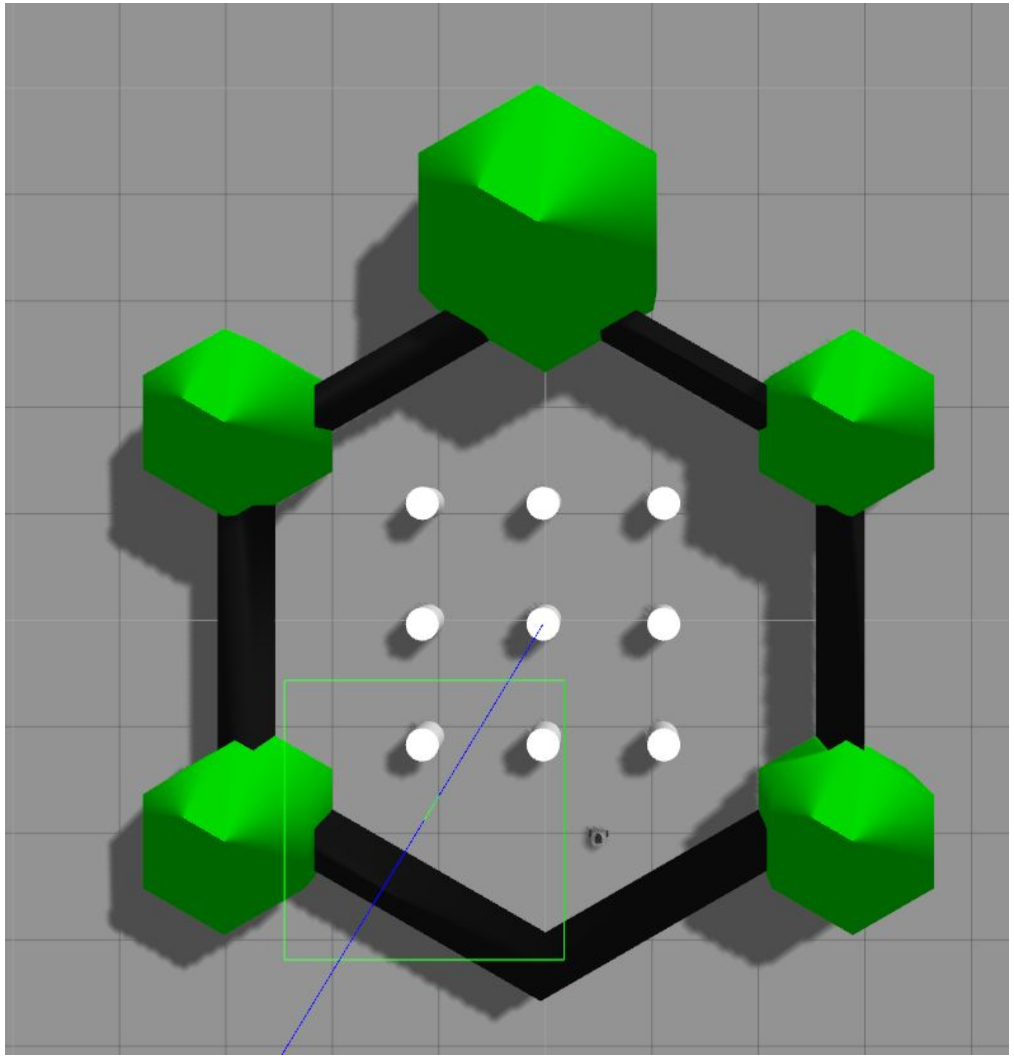


GAZEBO

$ export TURTLEBOT3_MODEL="burger"

then launch turtlebot

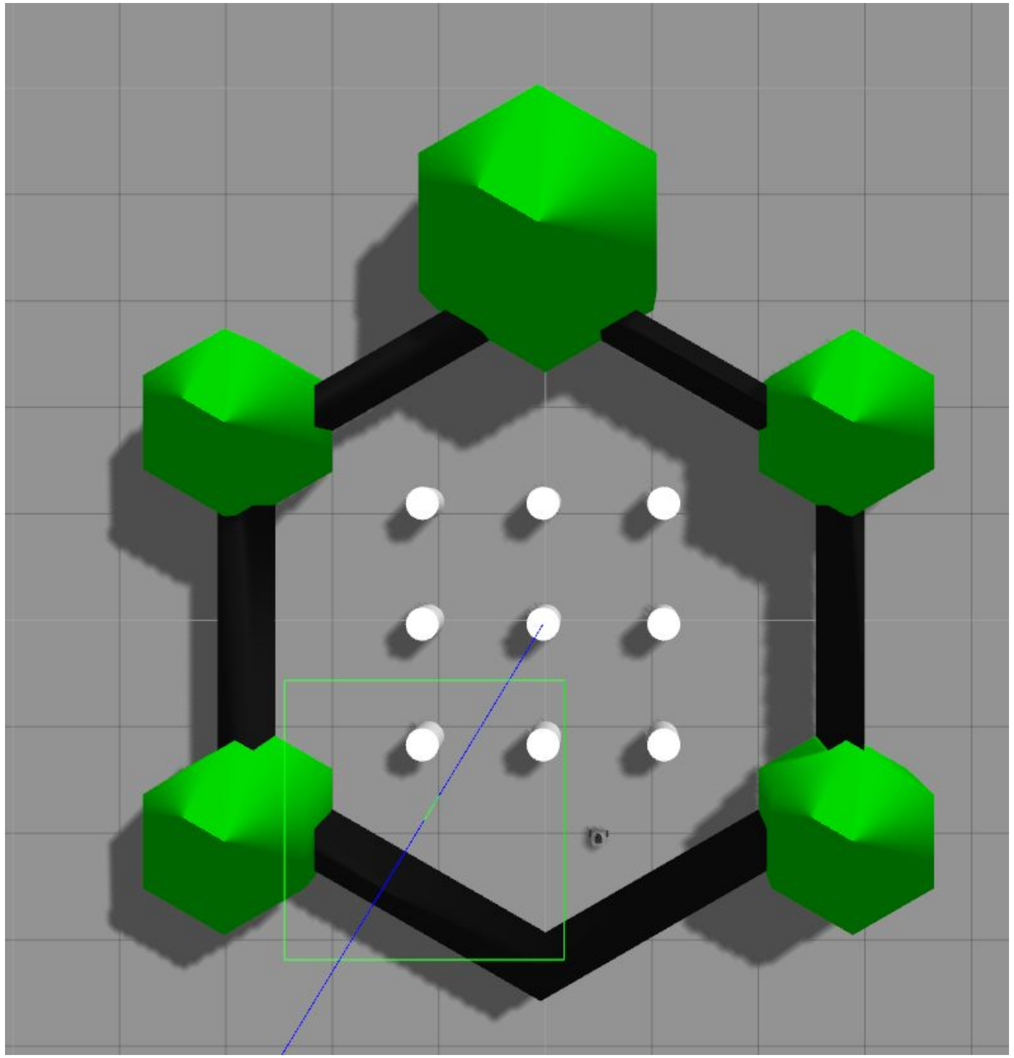
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch



GAZEBO

Now we want to control the robot, so we will launch the teleop node:

$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch



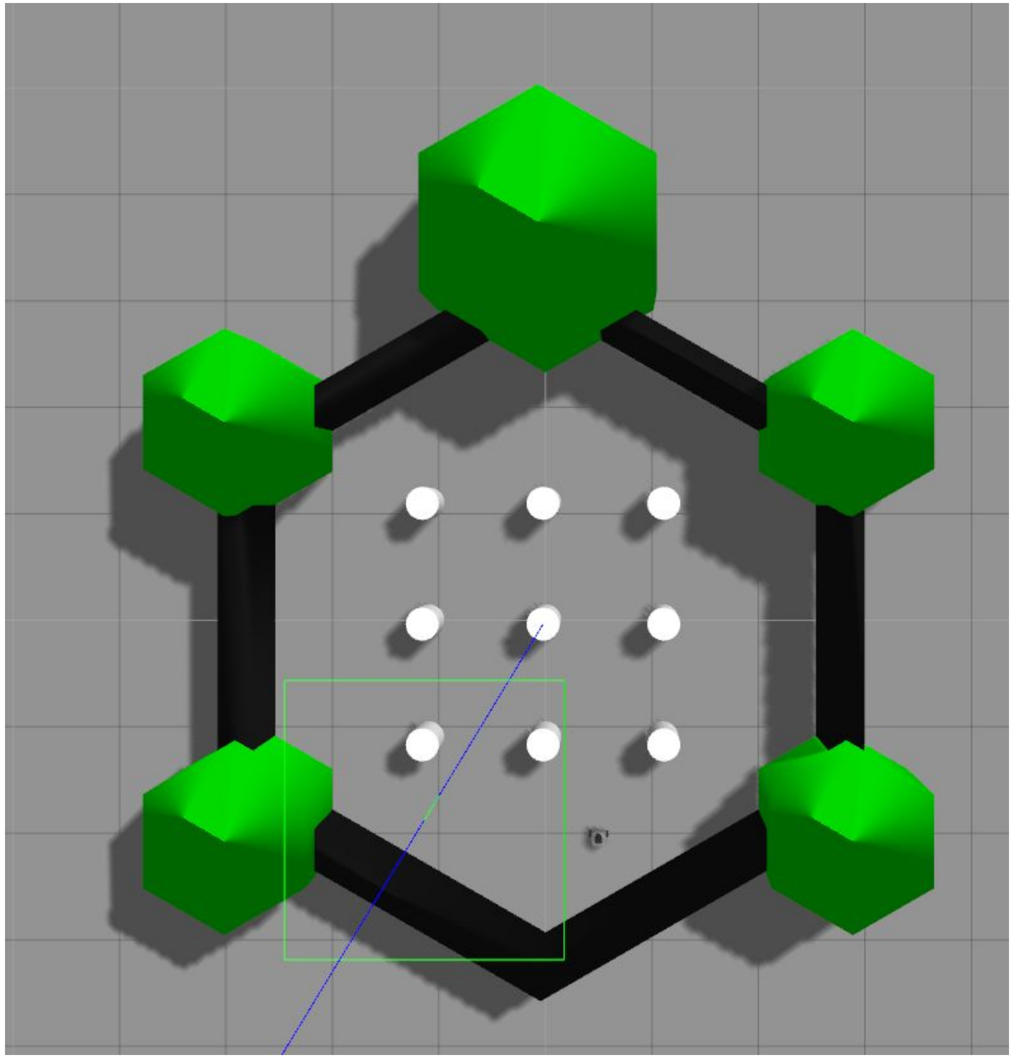
GMAPPING

Record a bag and than create a map

to record a bag we will use turtlebot3:

$ rosbag record -O turtlebot_bag -a

Now move the robot in the turtlebot world to get some data



GMAPPING

before starting gmapping we can take a look at the bag (remember to start roscore):

but first we set ros to use simulated time:

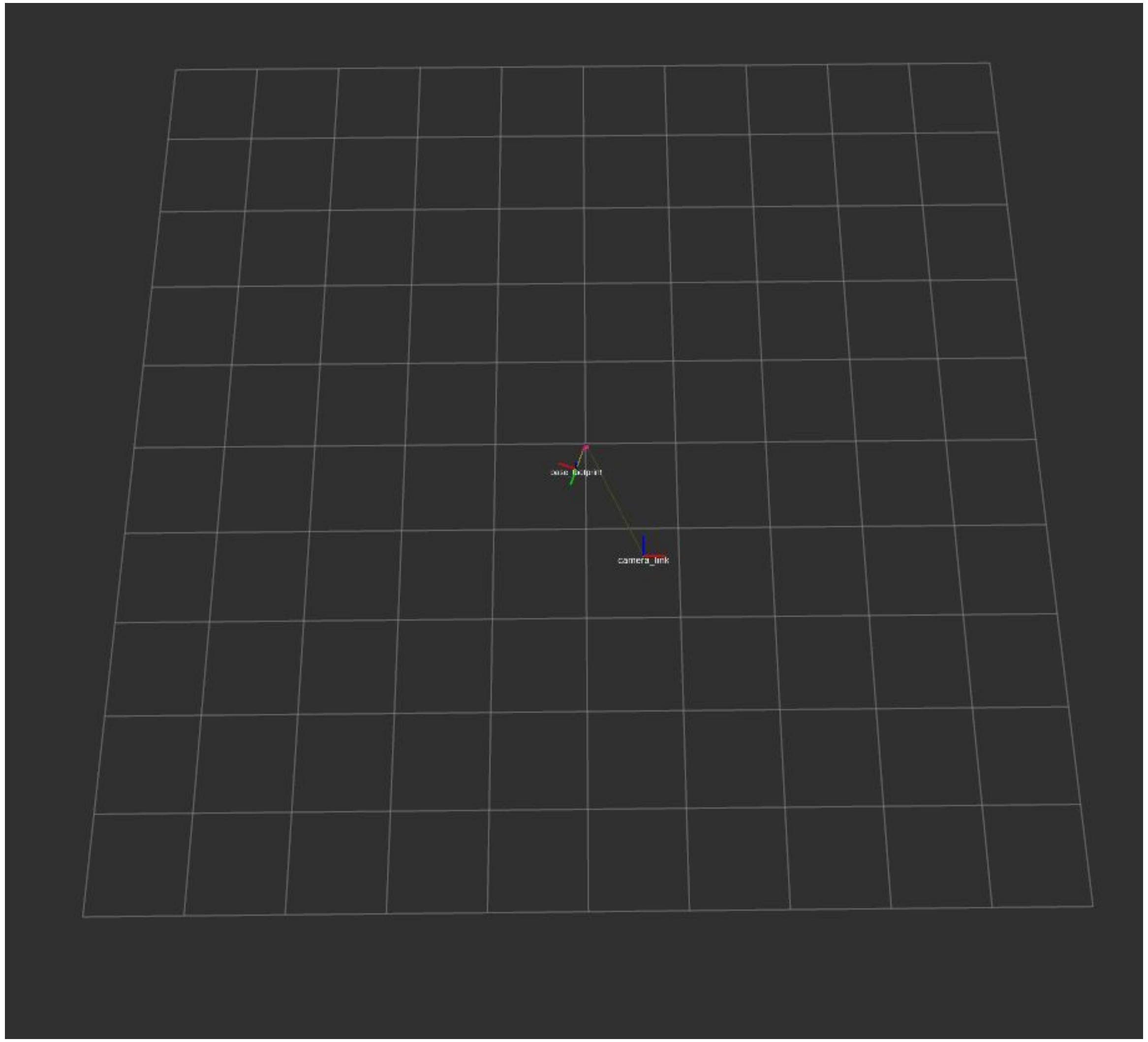
$ rosparam set use_sim_time true

then:

$ rosbag play --clock turtlebot_bag.bag

to visualize the data we will open rviz:

$ rviz



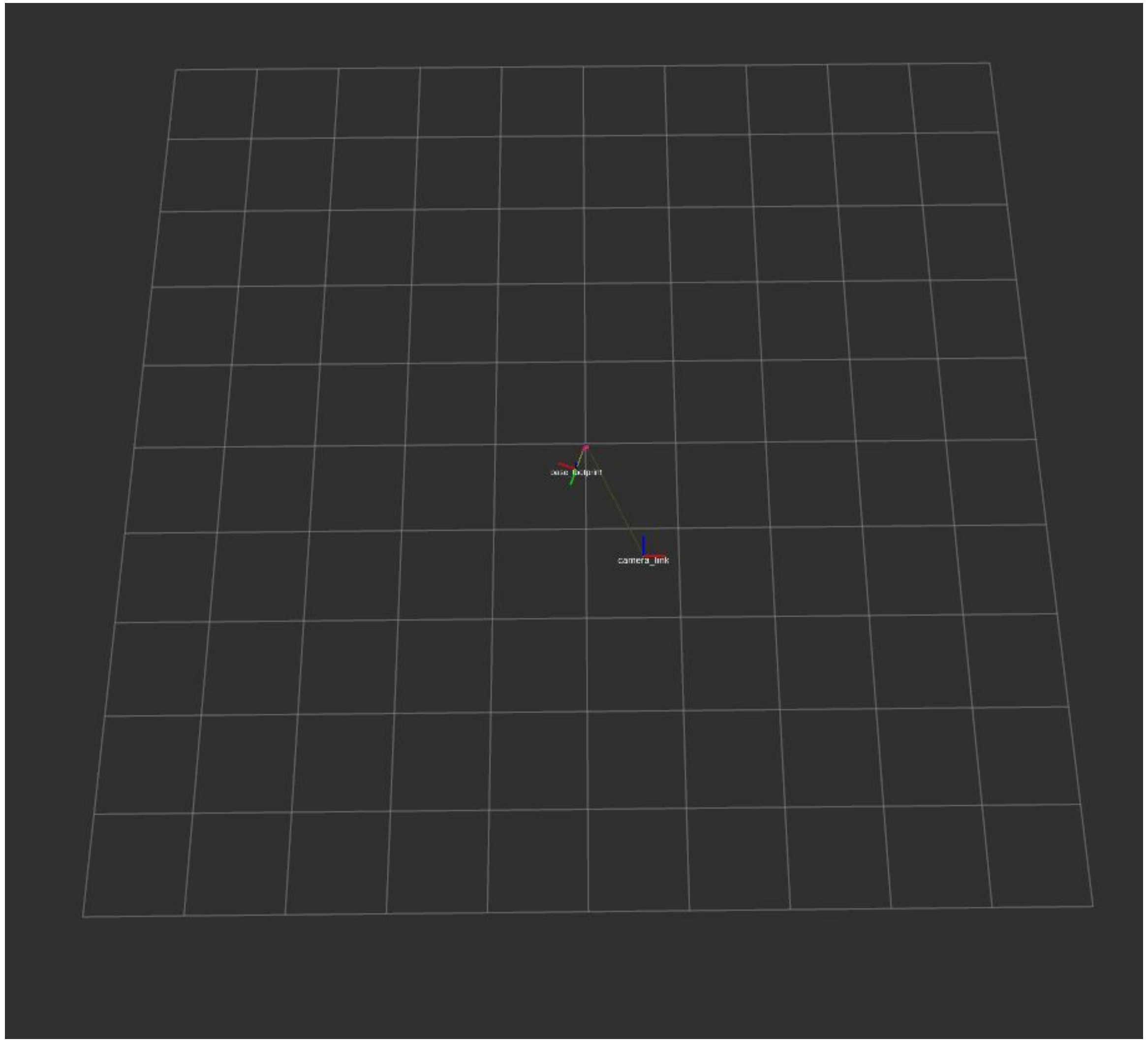
GMAPPING

if we try to add the laser data we will get the error:
“For frame [base_scan]: Frame [base_scan] does not exist”

this because we don’t have a transformation between the position of the laser scanner and the centre of the robot.
We then have to add manually the transformation, run:

$ rosrun tf static_transform_publisher 0 0 0 0 0 0 1 base_footprint base_scan 100

now we see the laser in rviz



GMAPPING

Now we can finally start gmapping; stop the bag and close rviz.

make sure the static transform is still published

then start gmapping:

```
$ rosrun gmapping slam_gmapping scan:=/scan _base_frame:=base_footprint
```

we have to specify some parameters that are not at the default value like the scan topic and the base frame

last start again the bag file

```
$ rosbag play --clock turtlebot_bag.bag
```

wait the bag to end

GMAPPING

To create the map, after the bag has finished playing run the command:

$ rosrun map_server map_saver -f map

to create the map file (both picture and yml)

GMAPPING

To run gmapping in real time:

start turtlebot:

```
$ export TURTLEBOT3_MODEL="burger"
```

```
$ roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

start the static tf publisher

```
$ rosrun tf static_transform_publisher 0 0 0 0 0 0 1 base_footprint base_scan 100
```

start gmapping

```
$ rosrun gmapping slam_gmapping scan:=/scan _base_frame:=base_footprint
```

GMAPPING

As previously to control the robot use the teleop node:

$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

We can visualize at runtime the map being created using rviz:

$ rviz

and adding the map topic

when the map is completed you can save it using the previous command:

rosrun map_server map_saver -f map