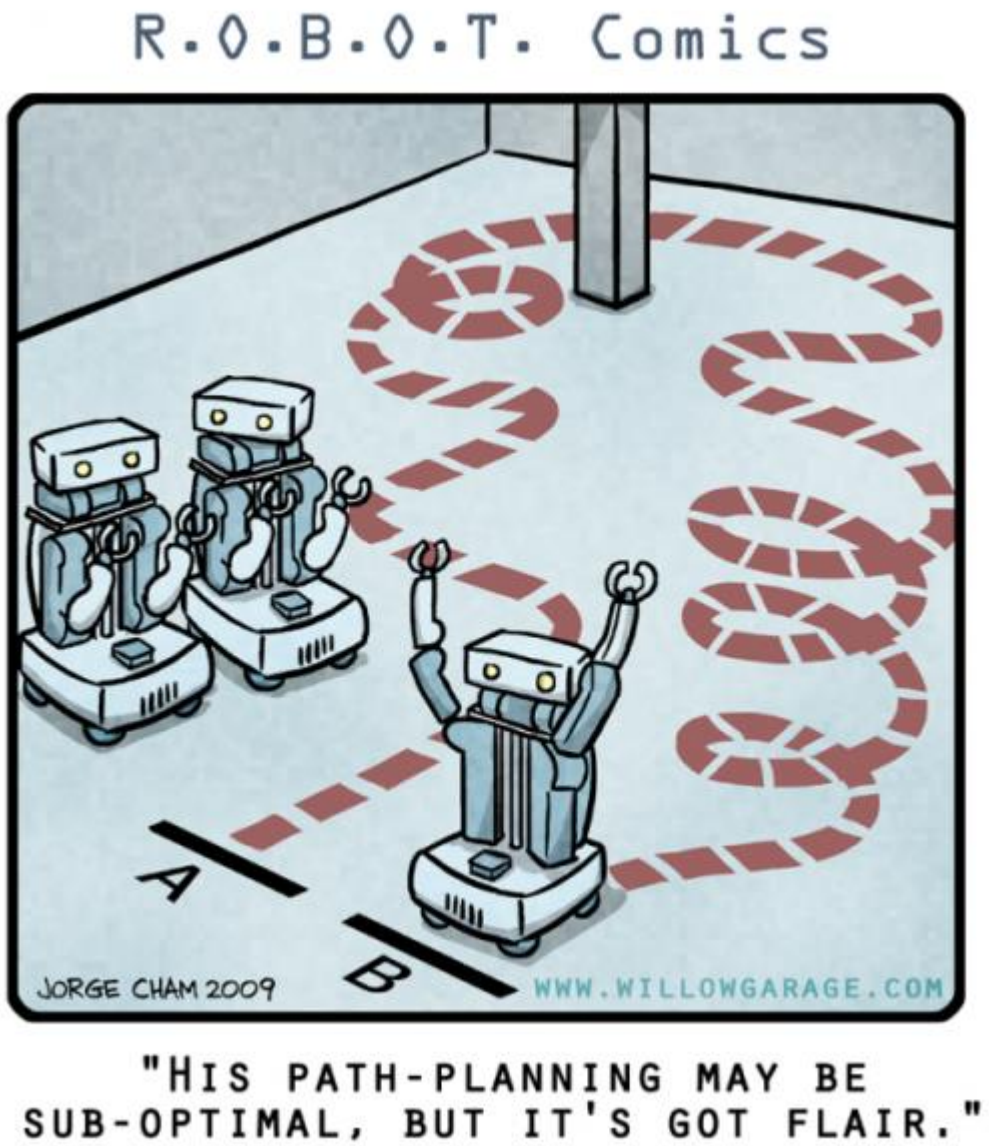
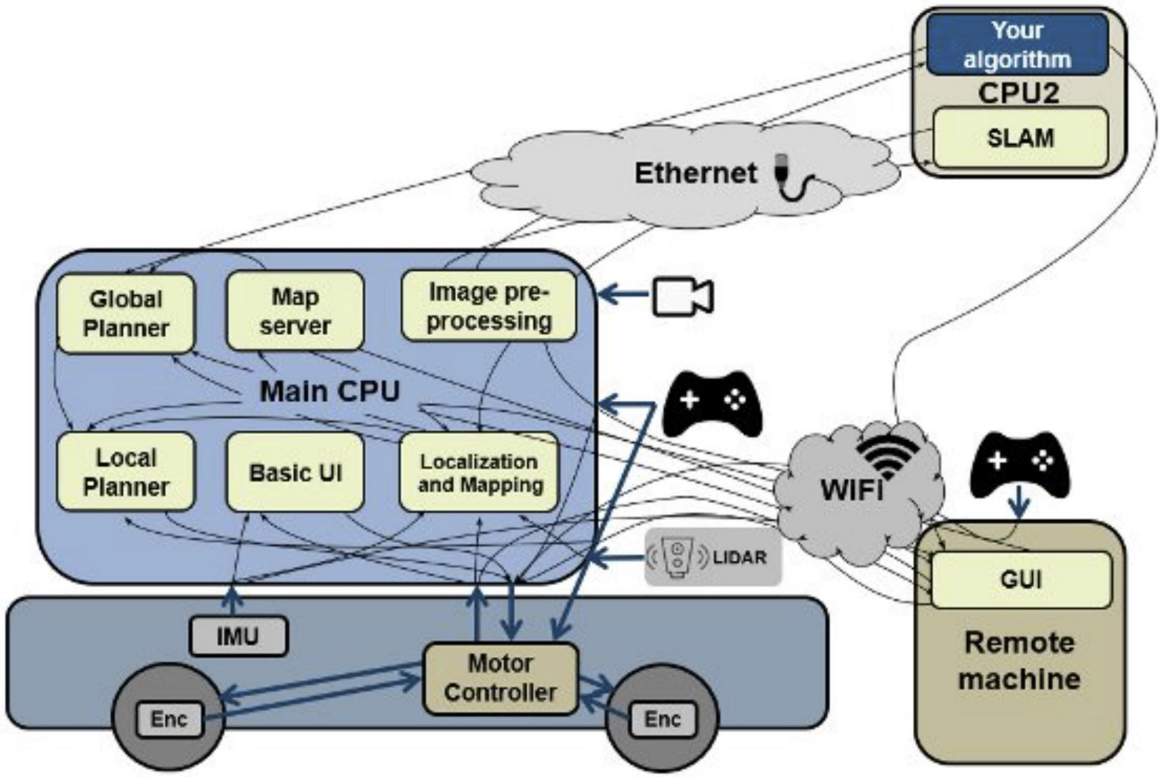
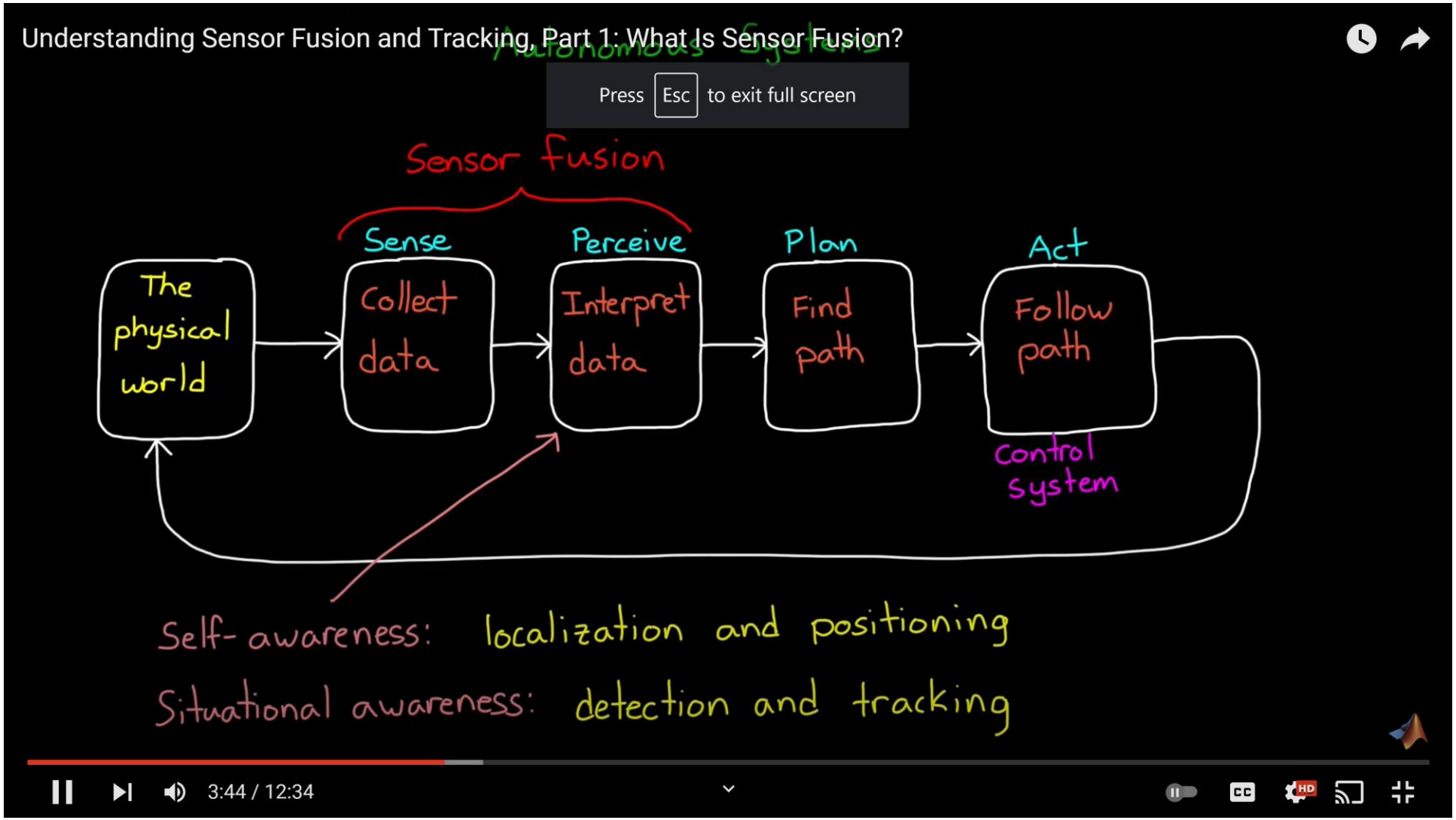




Architecture of AMRs.

https://www.youtube.com/watch?app=desktop&v=6qV3YjFppuc&t=678s



A John Deere 8R fully autonomous tractor is displayed ahead of CES on Jan. 4, 2022, in Las Vegas. John Deere and agricultural robot start-up Naio combined the popular 8R tractor, a plow, GPS and 360-degree cameras to create a machine a farmer can control from a smartphone. PATRICK T. FALLON/AFP VIA GETTY IMAGES

Founded in 2003, **METECS** is an engineering and applied technology company. We have a long history of providing high-fidelity simulation, software, robotics and analysis to NASA with contributions to the Space Shuttle, International Space Station, Orion, Lunar Gateway, and Artemis programs. Our customer base also includes clients from many other industries including agriculture, energy, construction, and sports marketing.

We are headquartered in Houston, Texas just outside the gates of NASA's Johnson Space Center.

We view two short videos to define the operations to follow:

Melonie Wise explains Odometry and the IMU for Turtlebot.

4:26 https://www.youtube.com/watch?v=3S8MXsnNe3U

Melonie explains Localization and AMCL 2:24

https://www.youtube.com/watch?v=Mv1mbsMfbmI

VP Robotics AutomationVP Robotics Automation
Zebra TechnologiesZebra Technologies Aug 2021 - Present · 8 mo Aug 2021 - Present · 8 mo San Jose, California, United States

Mapping & Navigating Using a RealSense R200 Camera & TurtleBot 3:18

https://www.youtube.com/watch?v=UQT9kakt60g

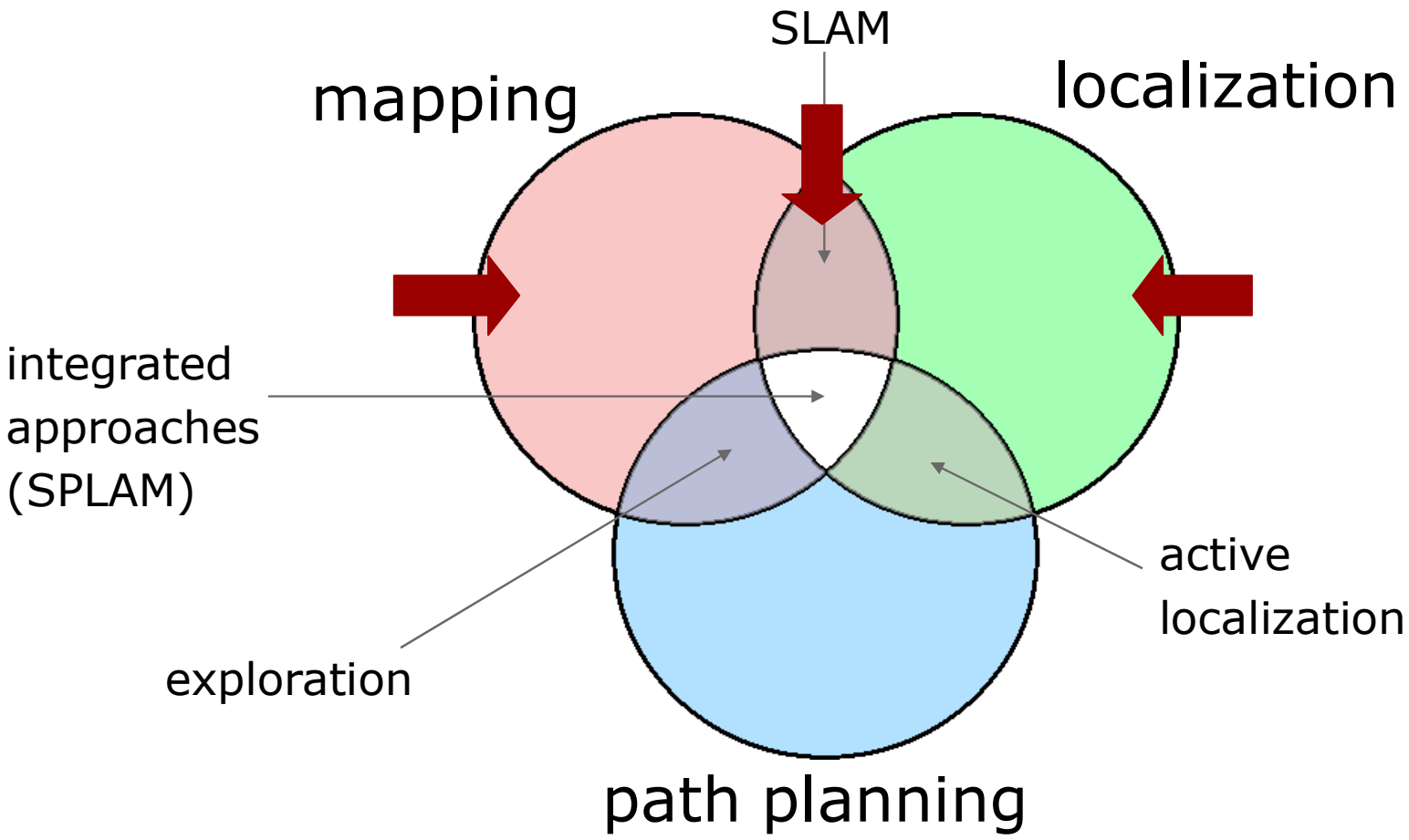
Note the path lines after the map is made and the robot navigates!

Turtlebot Monte Carlo Localisation (AMCL)

Ali Nagaria Watch TurtleBot navigate with a Map. 2:29

https://www.youtube.com/watch?v=u5n2jhF3UrU

This is the Navigation process using SLAM.

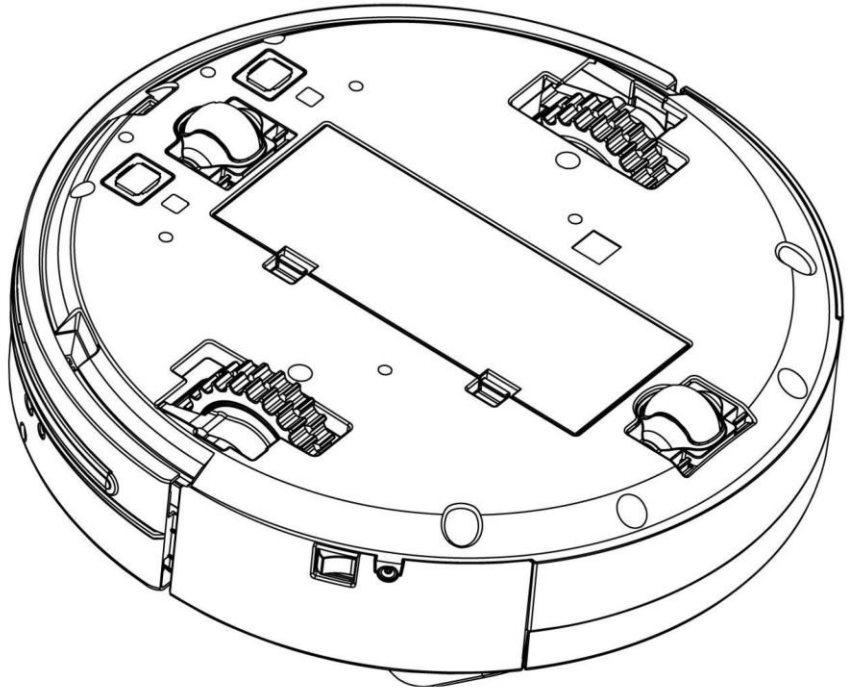


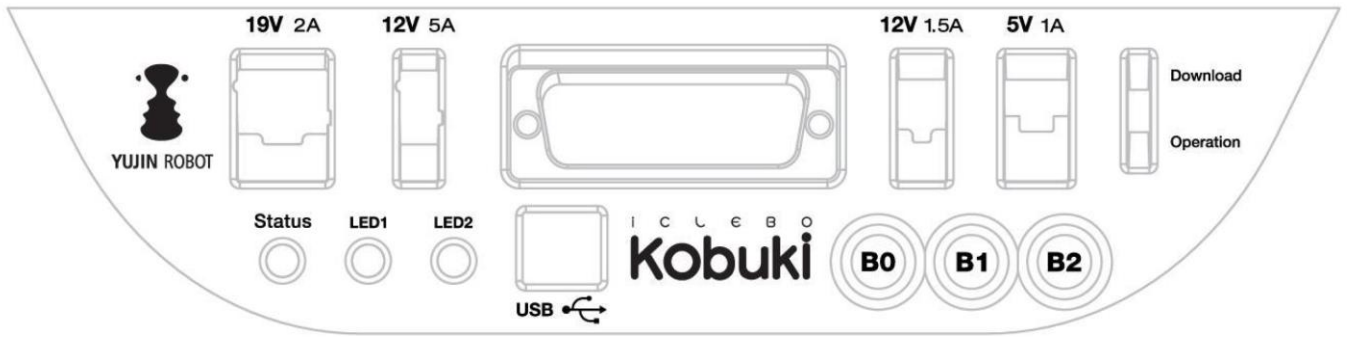
[courtesy of Cyrill and Wolfram]

In this section, we explore the TurtleBot's **odometry.** The general definition of odometry is the use of data from motion sensors, such as wheel encoders, to estimate change in Turtlebot's position over time. **Odometry** is used by the TurtleBot to estimate its position and orientation relative to its starting location given in terms of an *x* and *y* position and an orientation around the *z* (upward) axis as the TurtleBot moves.

Course Textbook Page 78

http://kobuki.yujinrobot.com/about2/





CLEARPATH ROBOTICS™

Driving Around with TurtleBot Chapter 3

- Loading the TurtleBot simulation software and using Gazebo with TurtleBot
- Setting up your system to control a real TurtleBot from its own netbook computer or wirelessly from a remote computer
- **Controlling the movement of the TurtleBot with ROS terminal commands or using the keyboard for control in teleoperation**
- **Creating a Python script which, when executed, moves TurtleBot**
- Using rqt tools to provide a GUI that aids the user in analyzing robot programs and also monitoring and controlling the robot
- **Exploring an environment using TurtleBot's odometry data**
- Executing the automatic docking program of TurtleBot
- Introducing a newer version of TurtleBot, called TurtleBot 3, and describing the simulation and keyboard control of a real TurtleBot 3

```
$ rosservice call gazebo/get_model_state '{model_name: mobile_base}'
```

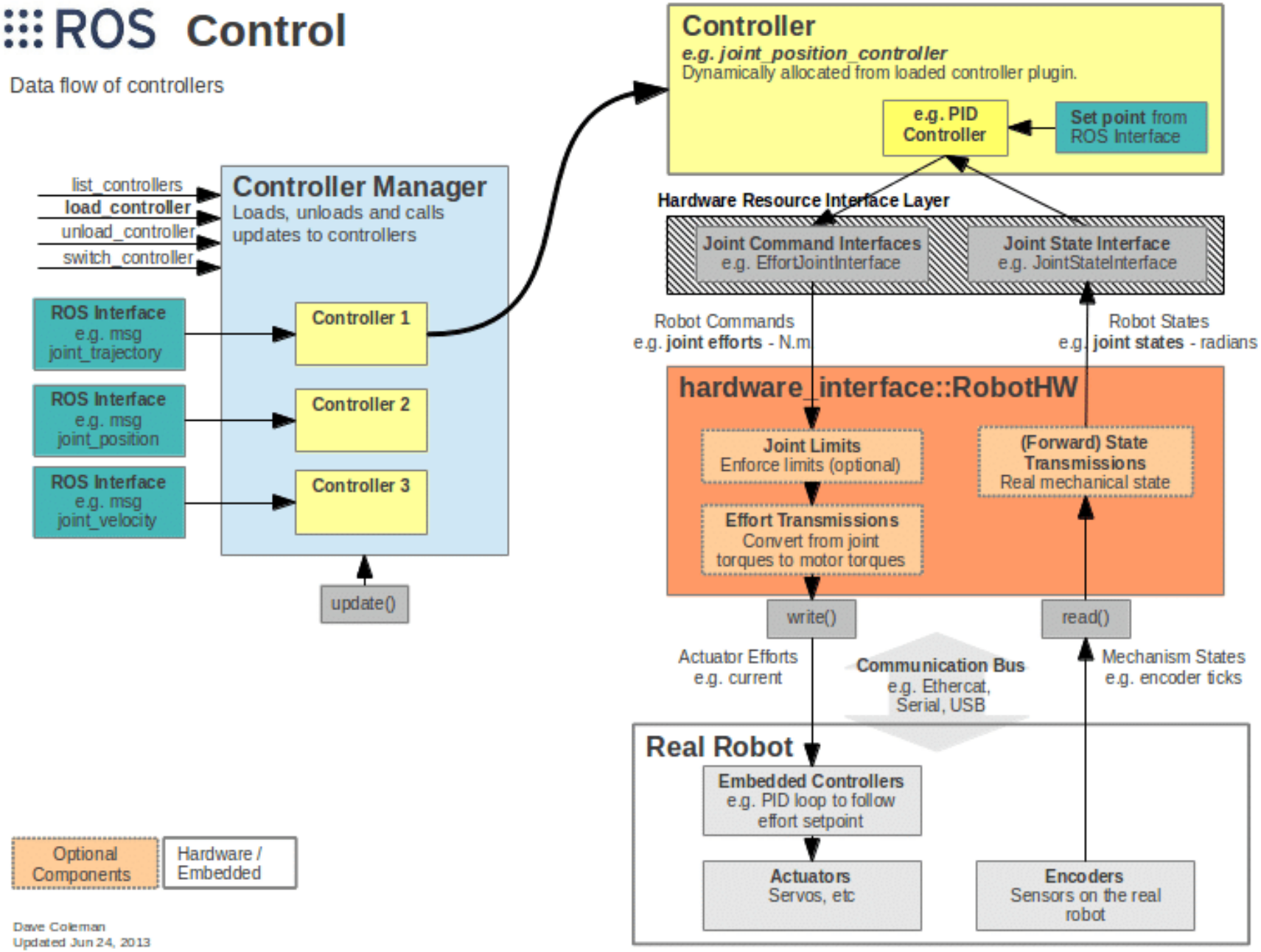
The output of the preceding command is similar to the following if TurtleBot is at the origin:

```
pose:
 position:
  x: 0.00161336508139
  y: 0.0091790167961
  z: -0.00113098620237
 orientation:
  x: -5.20108036968e-05
  y: -0.00399736084462
  z: -0.0191615228716
  w: 0.999808408868
 twist:
  linear:
   x: 9.00012388429e-06
   y: 6.54279879125e-05
   z: -1.4365465304e-05
  angular:
   x: -0.000449167550145
   y: 0.000197996689198
   z: -0.000470014447946
success: True
status_message: GetModelState: got properties
```

FROM THE ROBOT'S POINT OF VIEW!

ROS Control

Data flow of controllers



Dave Coleman
Updated Jun 24, 2013

https://github.com/turtlebot/turtlebot/blob/melodic/turtlebot_teleop/launch/keyboard_teleop.launch

```
import rospy
from geometry_msgs.msg import Twist
import sys, select, termios, tty
msg = """
Control Your Turtlebot!
---------------------------
Moving around:
u i o
j k l
m , .
31 32 33 34 35 36 37 38 39 40 41 42
```

http://wiki.ros.org/diff_drive_controller

https://github.com/turtlebot/turtlebot/blob/melodic/turtlebot_teleop/src/turtlebot_joy.cpp

```
finally:
twist = Twist()
twist.linear.x = 0; twist.linear.y = 0;
twist.linear.z = 0
twist.angular.x = 0; twist.angular.y = 0;
twist.angular.z = 0
pub.publish(twist)
166 167 168 169
```

https://github.com/turtlebot/turtlebot/blob/melodic/turtlebot_teleop/scripts/turtlebot_teleop_key

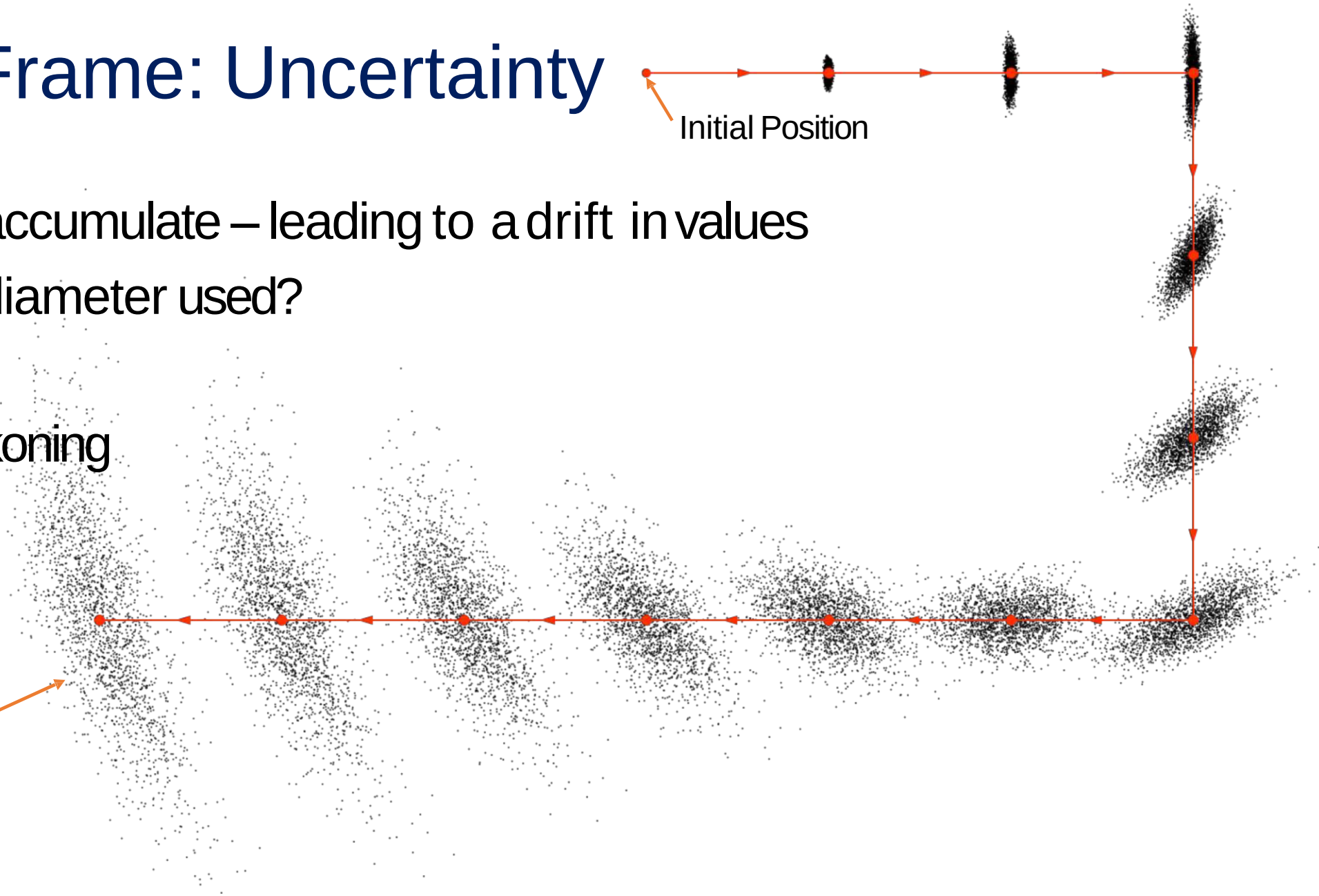
```
import rospy
from geometry_msgs.msg import Twist
import sys, select, termios, tty
msg = """
Control Your Turtlebot!
---------------------------
Moving around:
u i o
j k l
m , .
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly
CTRL-C to quit
"""
```

The termios functions describe a general terminal interface that is provided to control asynchronous communications ports.

select() allows a program to monitor multiple file descriptors

Odom Frame: Uncertainty

- Error can accumulate – leading to a drift in values
- Incorrect diameter used?
- Slippage?
- Dead Reckoning



(A few slides borrowed.)

Mudhar Behl F10 Slides

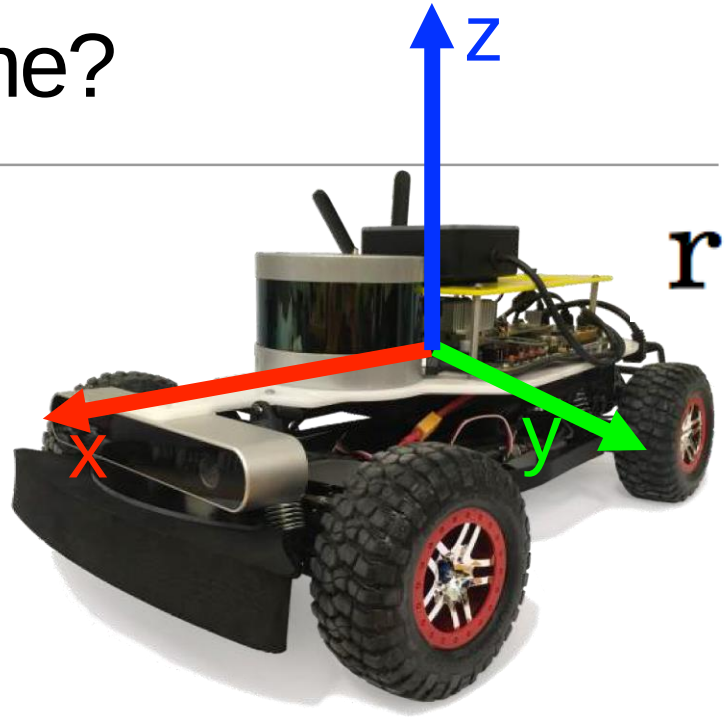
Introduction to Navigation using ROS

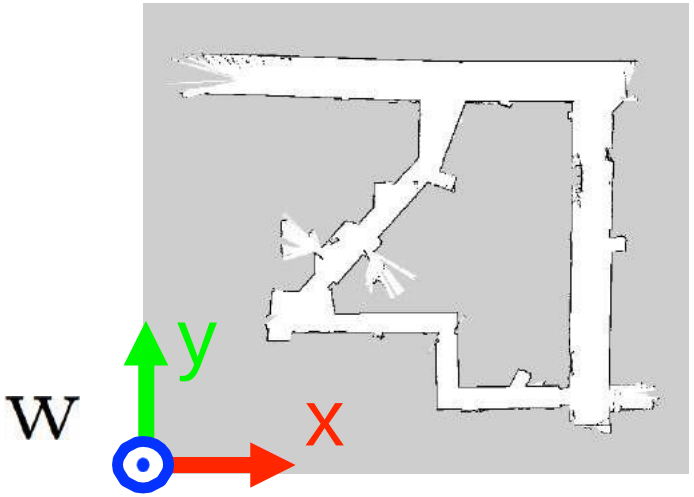
Giorgio Grisetti

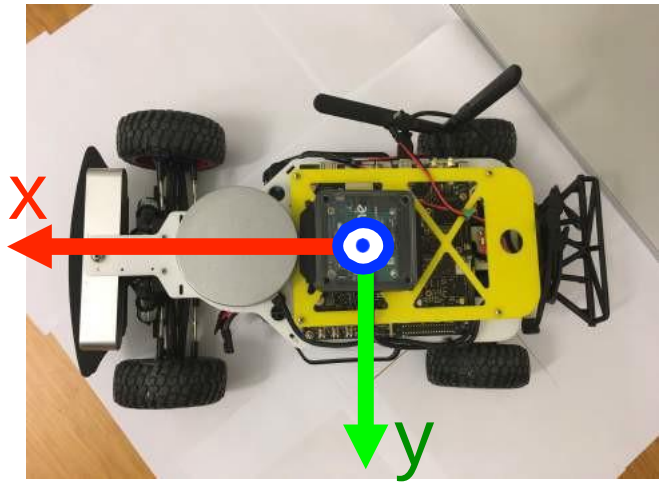
The material of this slides is taken from the Robotics 2 lectures given by G.Grisetti, W.Burgard, **Cyrill Stachniss**, K.Arras, D. Tipaldi and M.Bennewitz

What is a Coordinate Frame?

- "set of orthogonal axes attached to a body that serves to describe position of points relative to that body"
- **axes** intersect at a point known as the **origin**



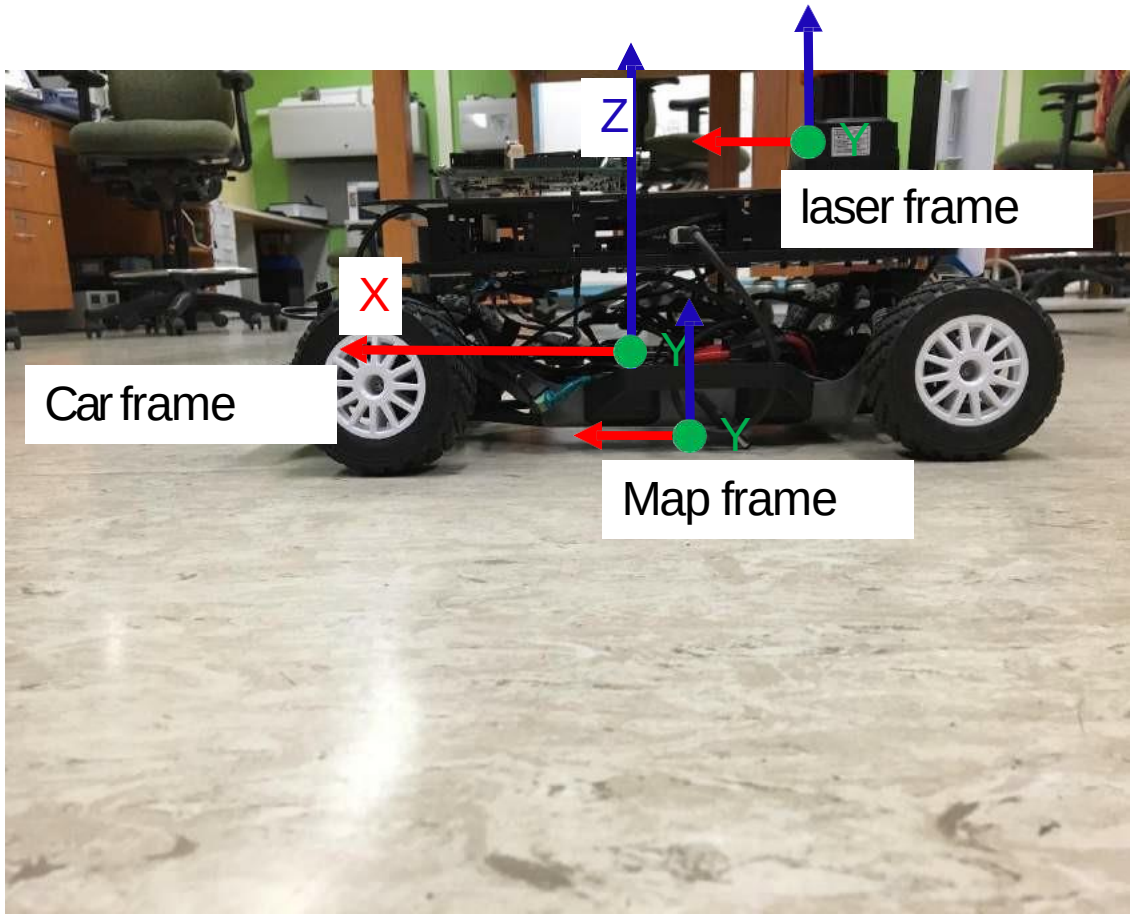


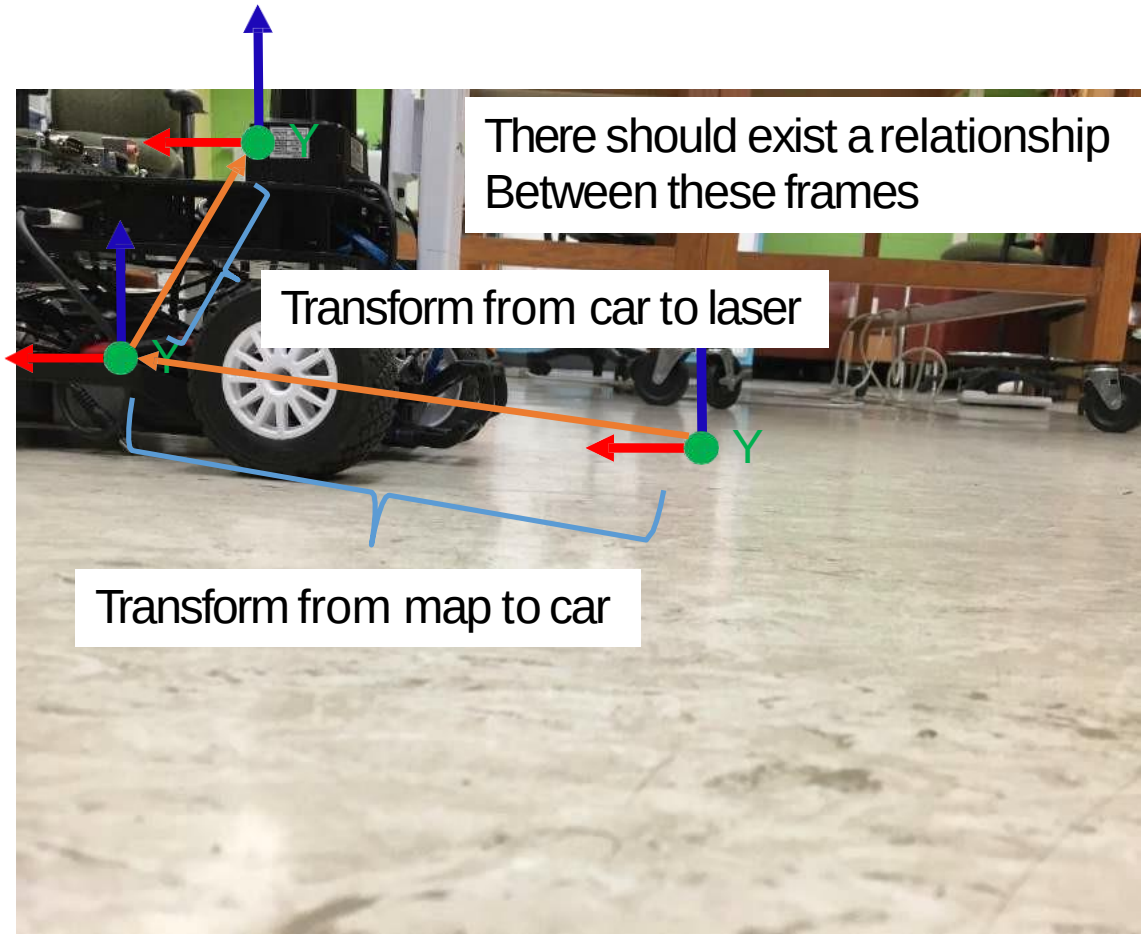


In many robotics problem, the first step is to assign a coordinate frame to all objects of interest

Transformations and Frames

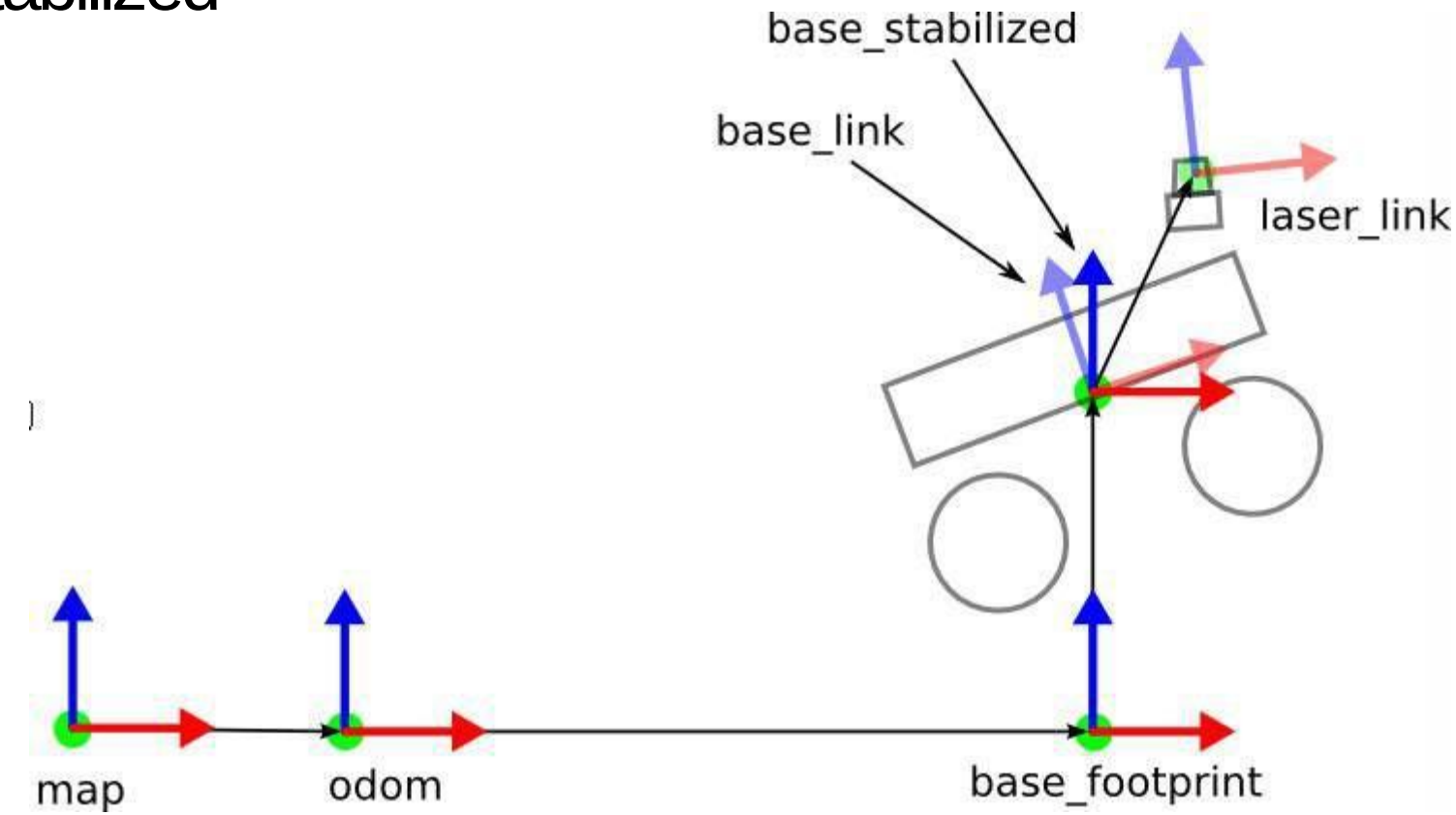
Between frames there will exist transformations that convert measurements from one frame to another





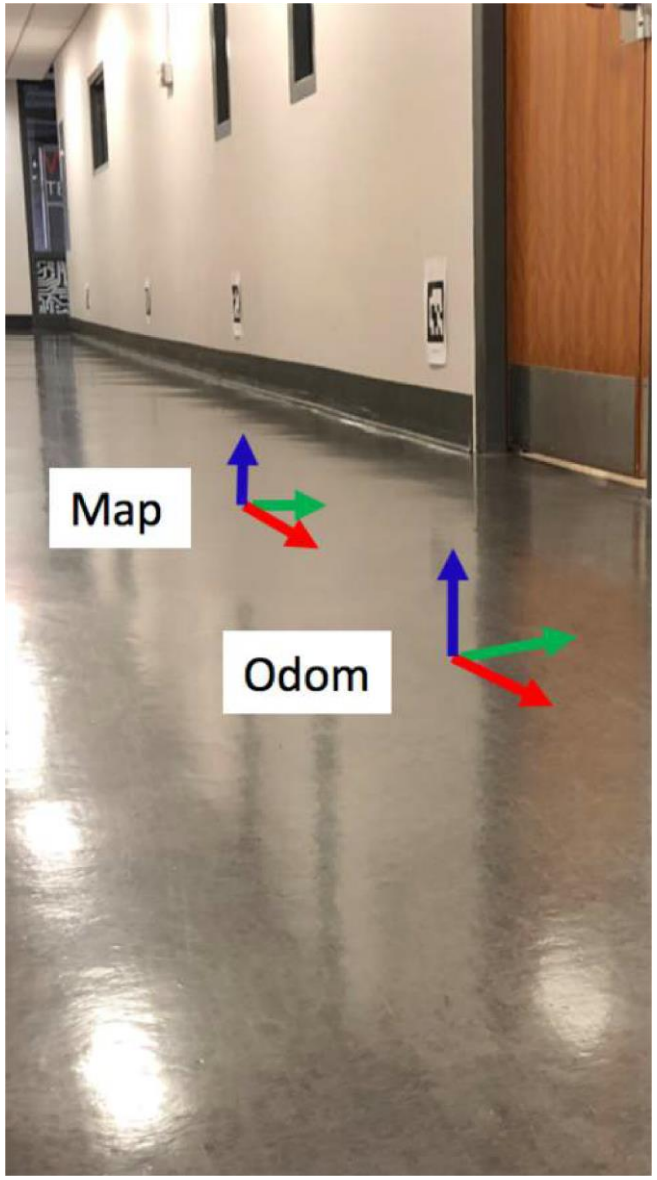
Base link: What is it

- Attached to the robot itself – base_footprint;base_link; base_stabilized



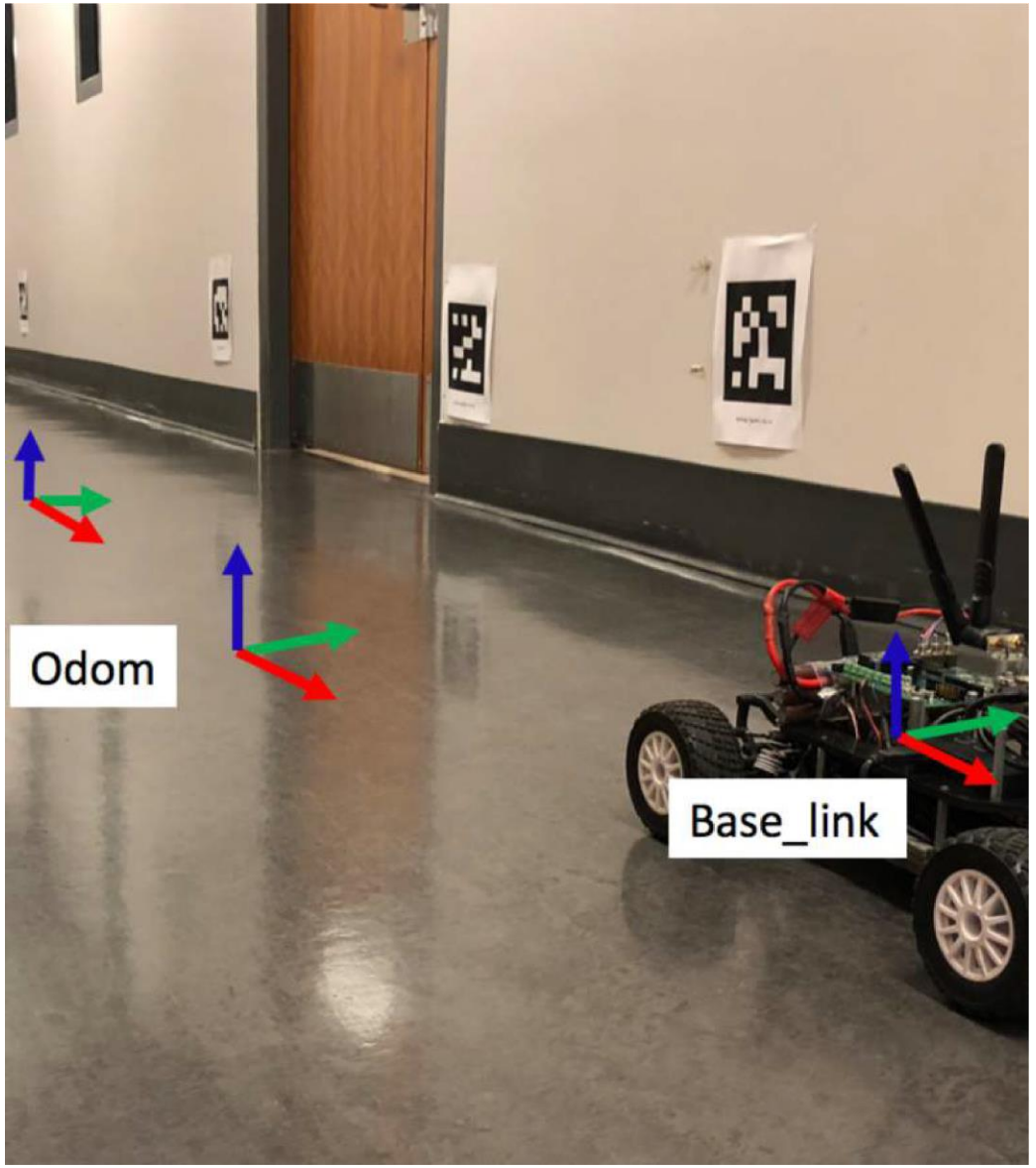
Odom Frame

- Odom frame originates where the car starts at. In this case the car started a meter in front of the map origin. Hence map -> odom transform is 1m in x direction.
- Odom frame stays fixed – it does not move with the robot.



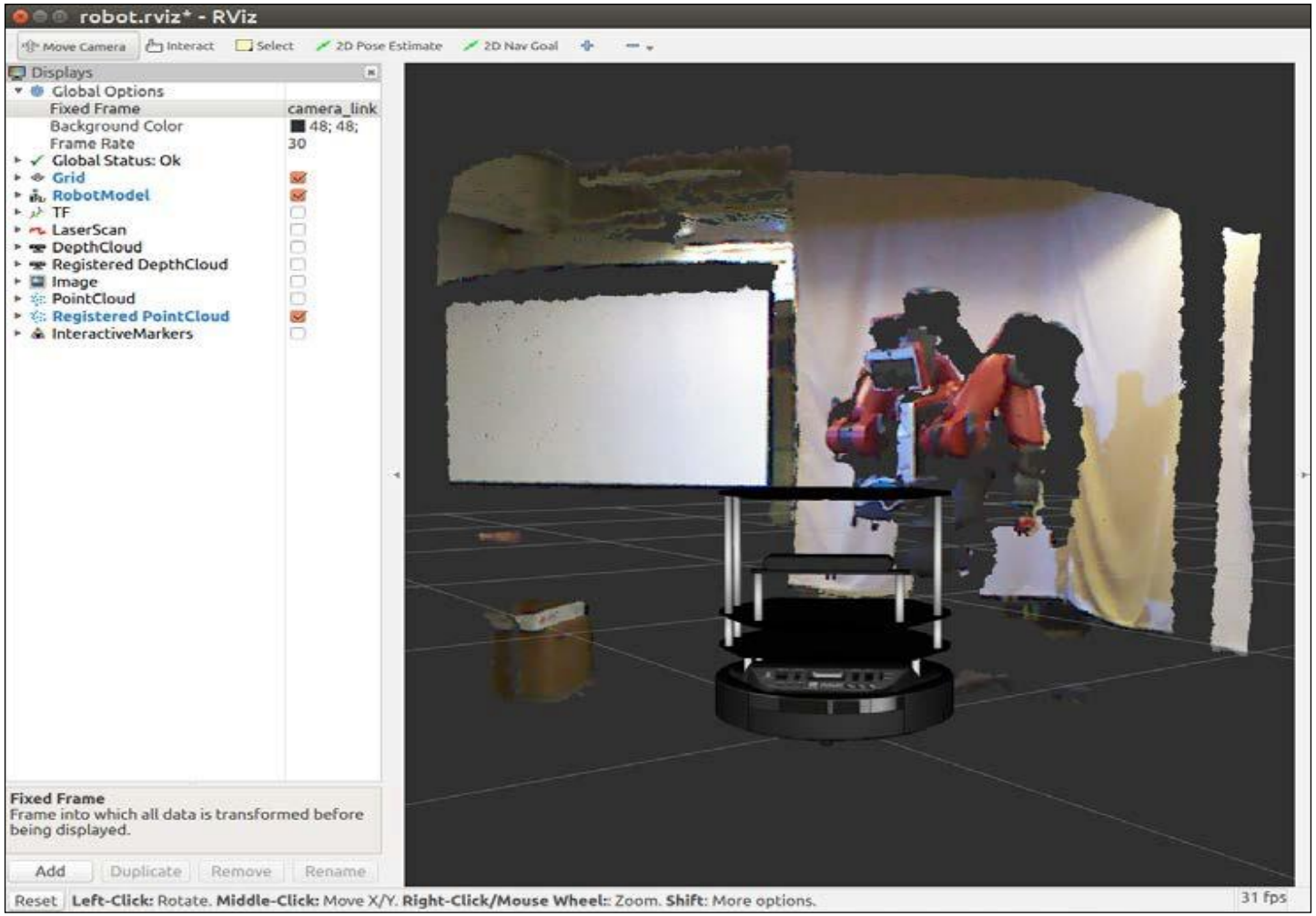
Base_link Frame

- Defined as the center of the car's rear axle
- Base_link frame moves with the car (it is not fixed)
- You really control where to place the base_link on the car. Just be consistent, and be aware that algorithms you re-use have their assumptions (usually, center of car's rear axle).



Navigating the World with TurtleBot

CHAPTER 4

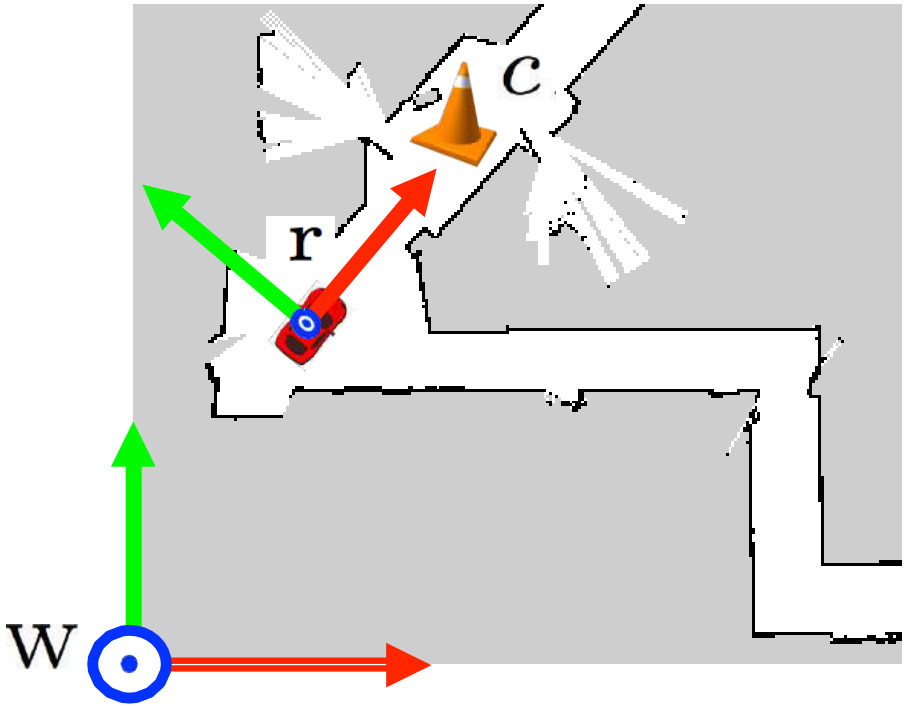


In this chapter, you will learn the following topics:

- How 3D vision sensors work
- The difference between the four primary 3D sensors for TurtleBot
- Details on a 2D vision system for TurtleBot 3
- Information on TurtleBot environmental variables and the ROS software required for the sensors
- ROS tools for the rgb and depth camera output
- How to use TurtleBot to map a room using **Simultaneous Localization and Mapping** (**SLAM**)
- How to operate TurtleBot in autonomous navigation mode by **adaptive monte carlo localization** (**amcl**)
- How to navigate TurtleBot to a location without a map
- How to navigate TurtleBot to waypoints with a Python script and a map

Rigid-body Transformations

- How to transform points from one coordinate frame to another?

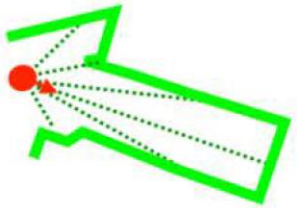


Given point “c” in coordinate frame “r” what is the position of the point in frame “w”?

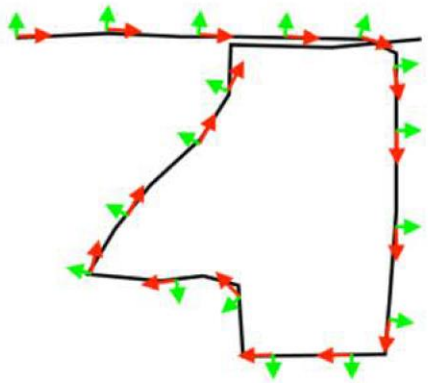
$$\boldsymbol{p}_{\mathrm{c}}^{\mathrm{W}} = \boldsymbol{R}_{\mathrm{r}}^{\mathrm{W}} \boldsymbol{p}_{\mathrm{c}}^{\mathrm{r}} + \boldsymbol{p}_{\mathrm{r}}^{\mathrm{W}}$$

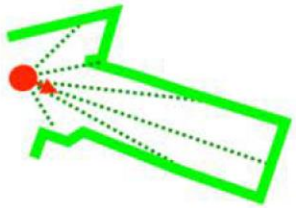
- **why rigid**? transformation does not change distance between points (does not deform shapes)
- translations, rotations, (reflections)

Problem Setting

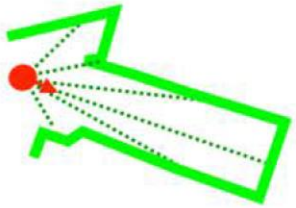


Localization: given a ***map***, use *sensor data* to estimate the current *pose* of the robot



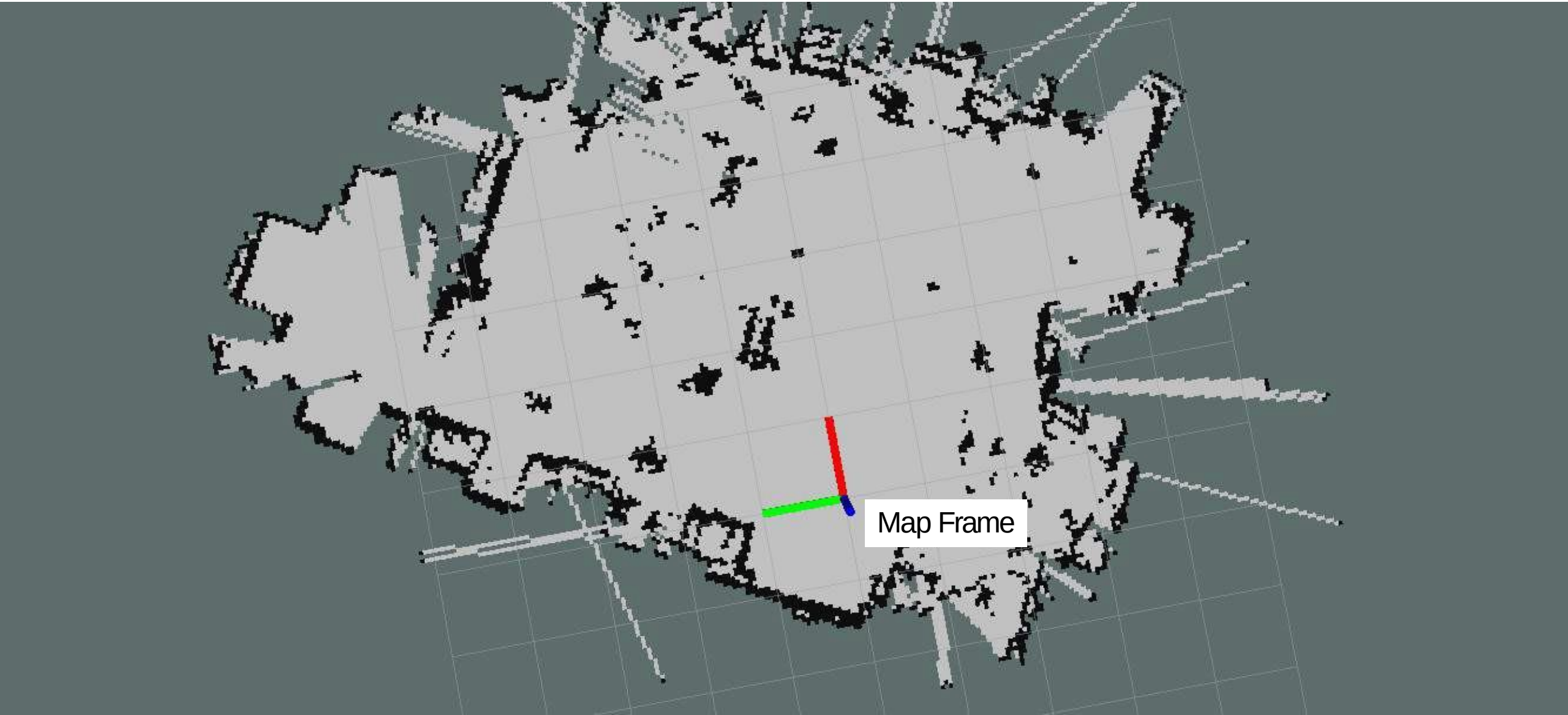


Mapping: given robot pose at each time (**trajectory**), use *sensor data* to build map



Simultaneous Localization and Mapping (SLAM): use sensor data to build map and estimate robot trajectory

Map frame

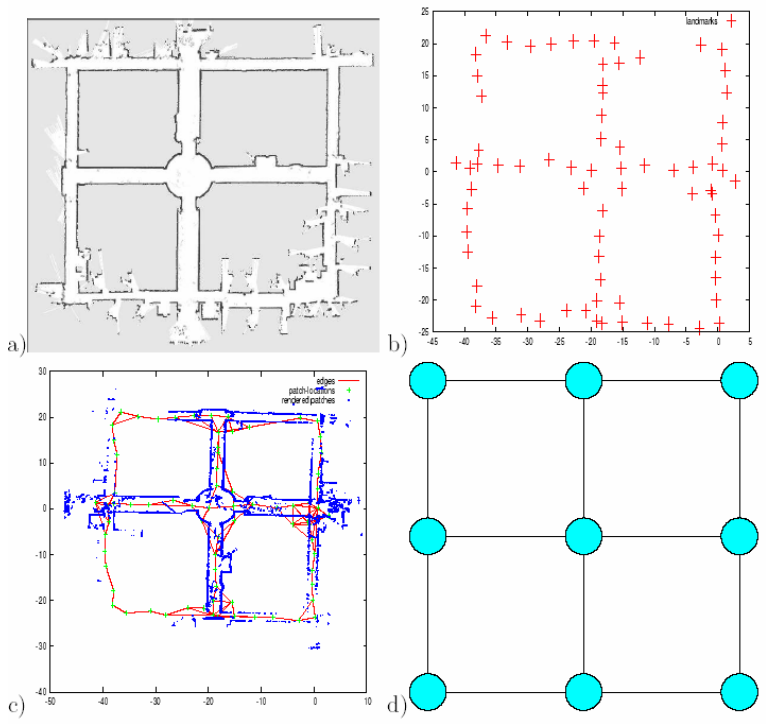


Map

- 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:

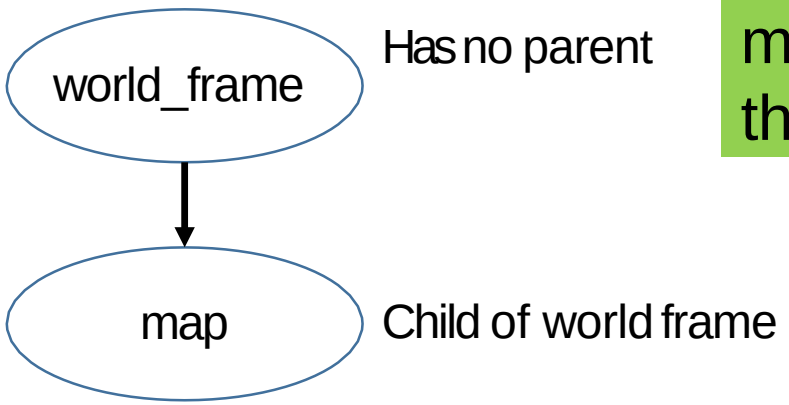
- Metric
  - Grid Based
  - Feature Based
  - Hybrid
- Topological
- Hybrid



Map Frame: ROS

- The tf package – tracks multiple 3D coordinate frames - maintains a tree structure of frames – access relationship of any 2 frames at any point of time
- ROSREP (ROS Enhancement Proposals) 105 describes the various frames involved.

- Normal hierarchy



A tf tree is a structure that maintains relations between the linked frames.

Note:
Tf = transformer class

Odom Frame: Calculation

- Difference in count of ticks of wheels – orientation
- Integrating the commanded velocities/accelarations
- Integrating values from IMU
- Scan Matching

The laser_scan_matcher package is an incremental laser scan registration tool.
The package allows to scan match between consecutive sensor_msgs/LaserScan messages,
and publish the estimated position of the laser as a geometry_msgs/Pose2D or a tf transform.
The package can be used without any odometry estimation provided by other sensors.
Thus, it can serve as a stand-alone odometry estimator. Alternatively, you can provide several types of odometry input to improve the registration speed and accuracy.

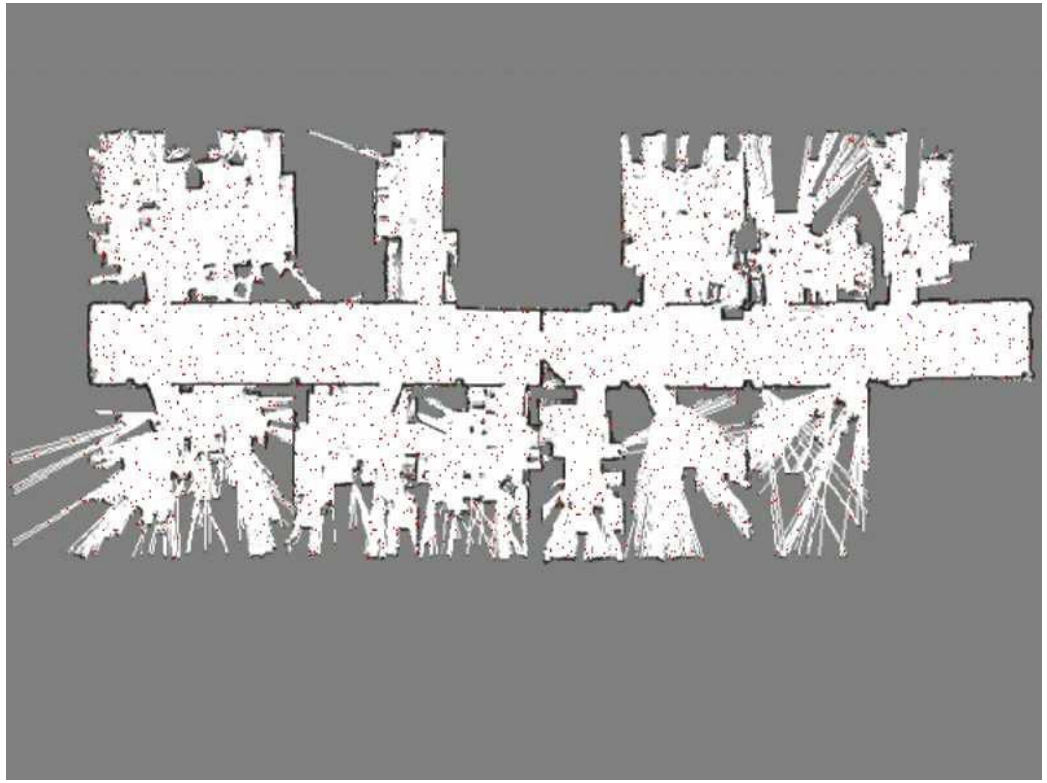
REMEMBER: The system must also coordinate times. If the odom position is “late”,
it should not be used with the laser distance data.

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, the measurements up to the current instant and a map.



Path Planning

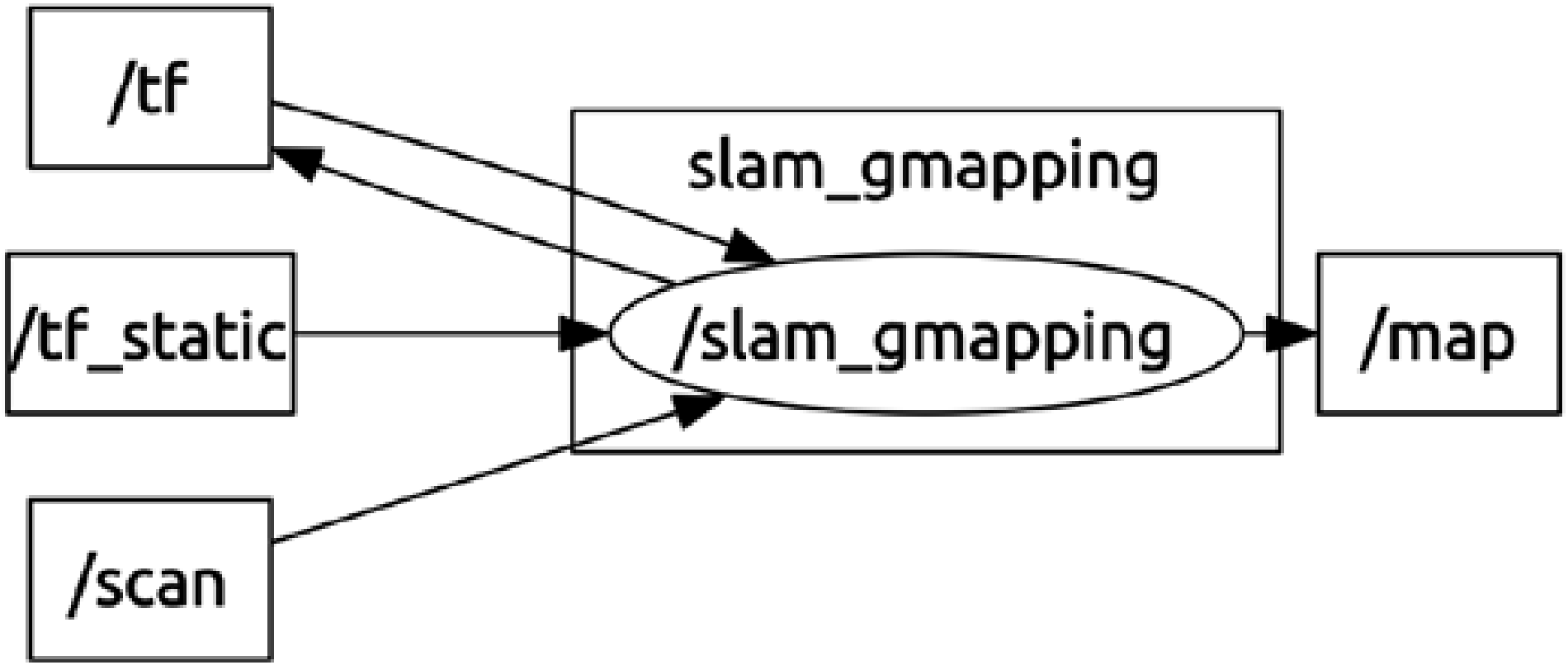
- Determine (if it exists) a path to reach a given goal location given a localized robot and a map of traversable regions.

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”

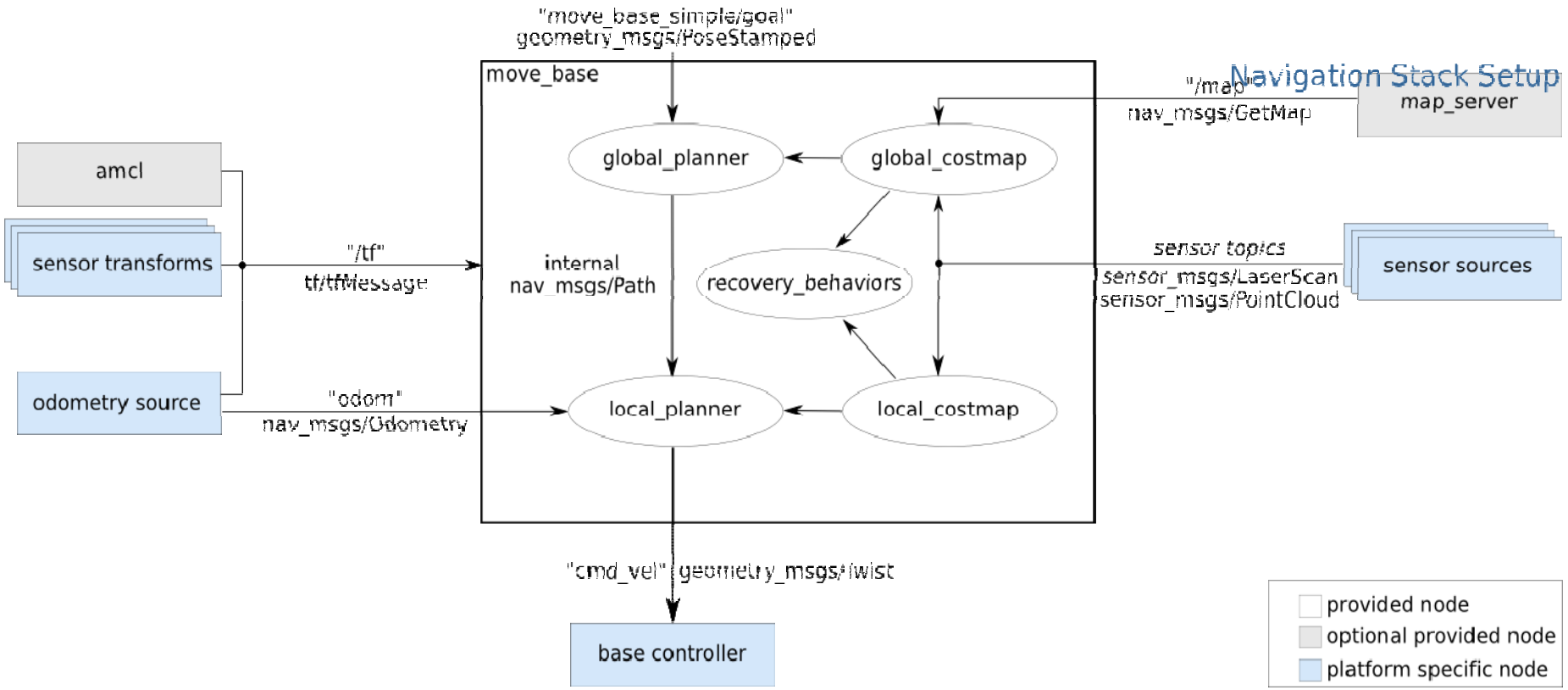
  Thus we need to complement our ideal design with other components that address these issues, namely
  - Obstacle-Detection/Avoidance
  - Local Map Refinement, based on the most recent sensor reading.


/tf
/tf_static
/scan
slam_gmapping
/slam_gmapping
/map

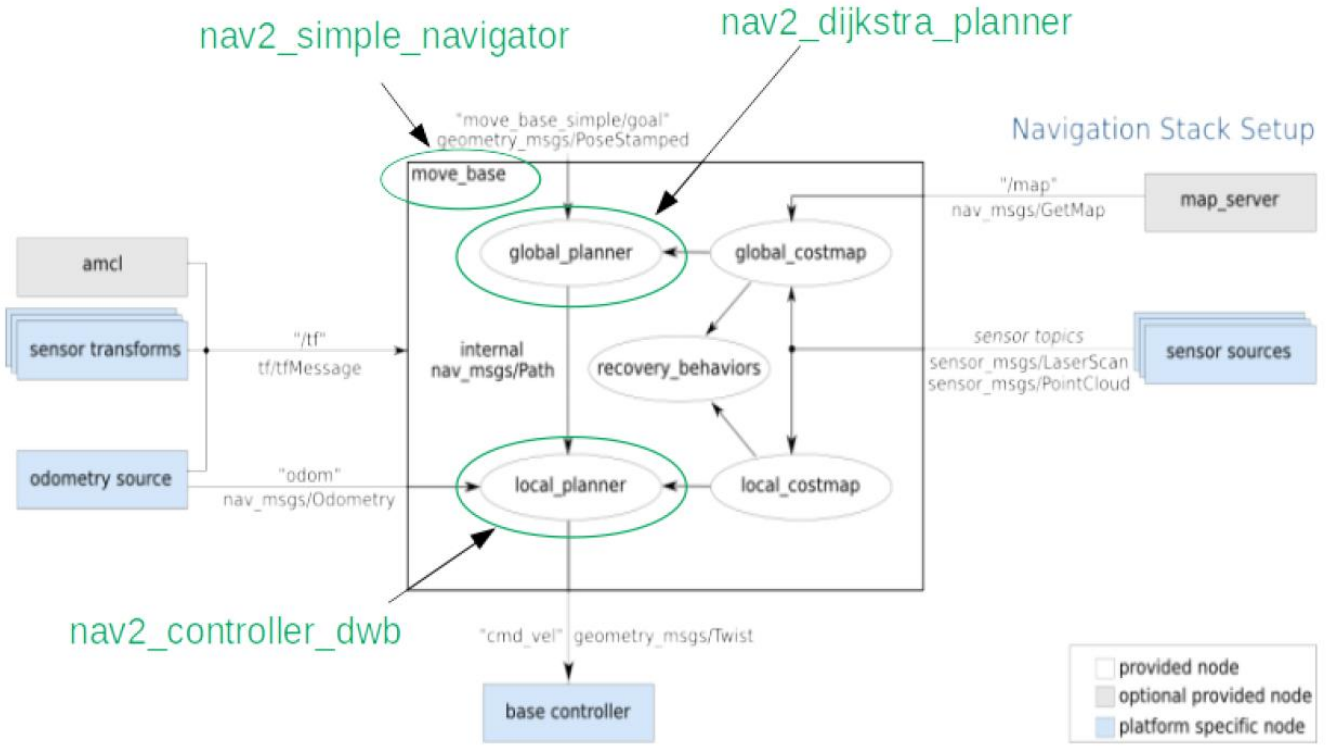
ROS Navigation Stack

- 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)

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



https://navigation.ros.org/about/ros1_comparison.html#ros1-comparison



Note: `nav2_simple_navigator` no longer exists, it has been replaced by `nav2_bt_navigator`.

Building a Map

- There are many versions of SLAM algorithms around.
- ROS uses GMapping, which implements a particle filter to track the robot trajectories.
- To build a map you need to
  - Record the map with /odom, /scan/ and /tf while driving the robot around in the environment it is going to operate in
  - Play mapping demo 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 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.

http://robots.stanford.edu/papers/fox.aaai99.pdf

Monte Carlo Localization: Efficient Position Estimation for Mobile Robots

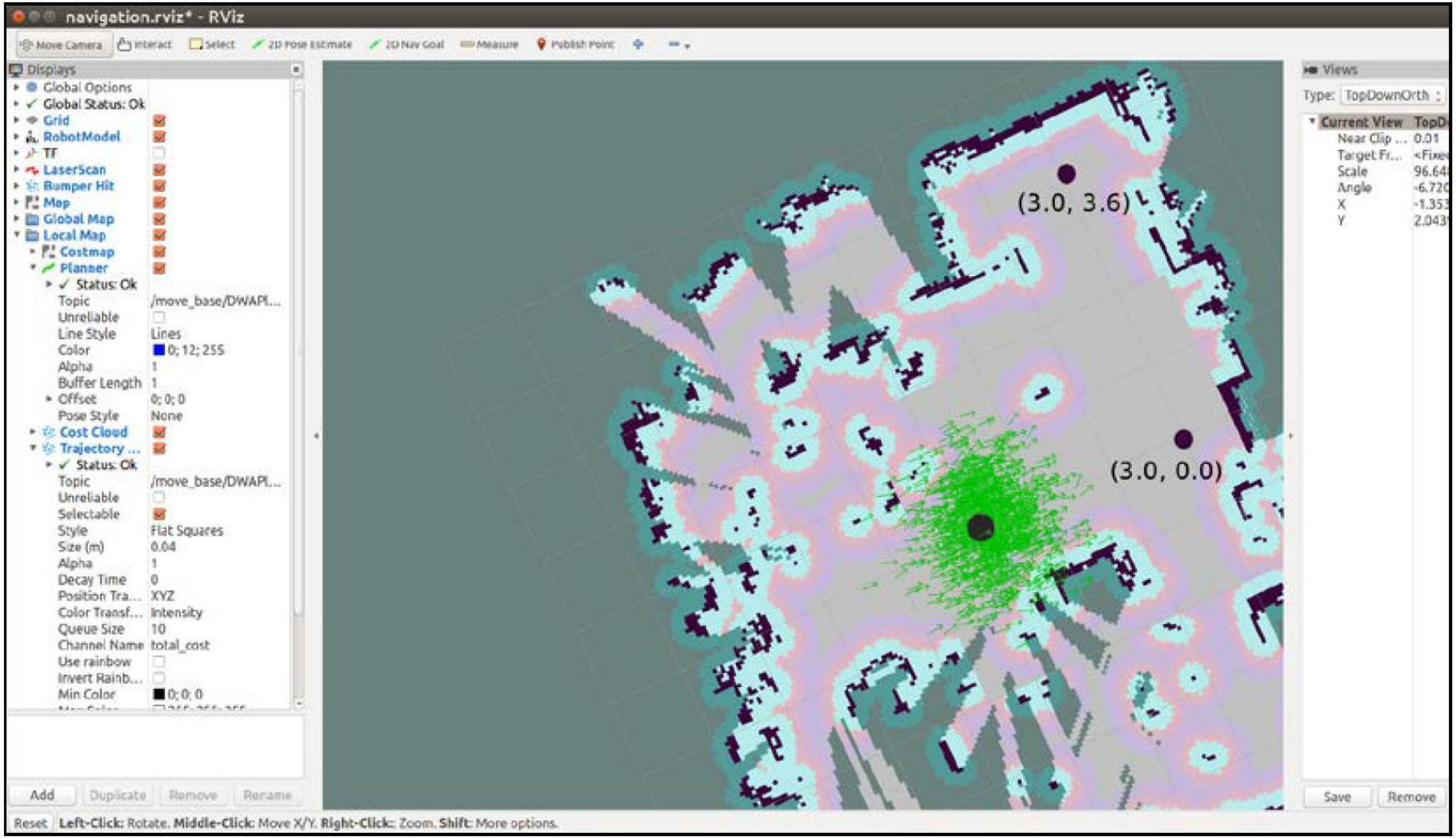
Dieter Fox, Wolfram Burgard†, Frank Dellaert, Sebastian Thrun

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

†Computer Science Department III
University of Bonn
Bonn, Germany

Localization

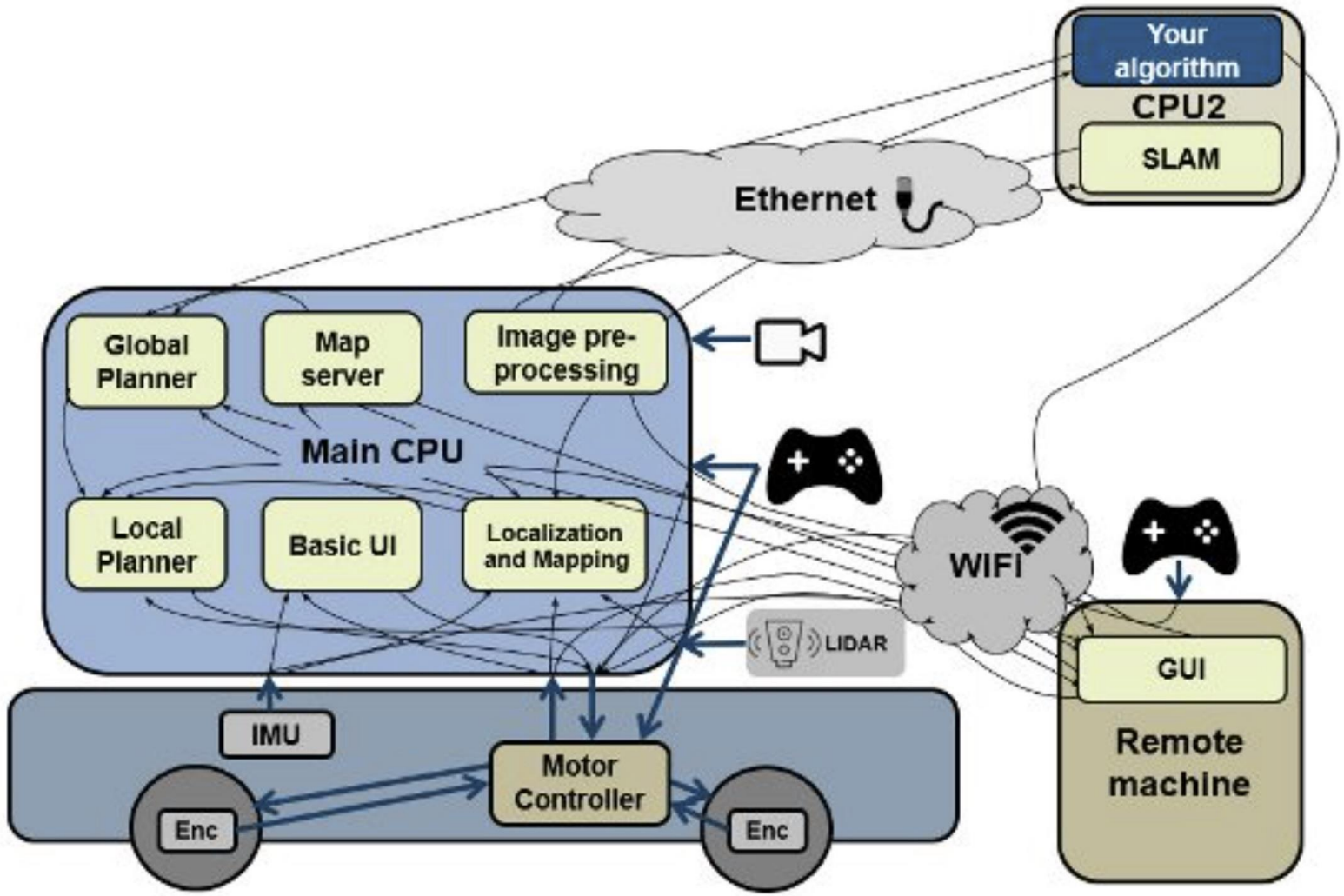
- AMCL relies on a laser – Or Depth Sensor
- Unless you want to spend $$, you will not get a laser. ,
- However your robot will localize with a Kinect or Asus or Real Sense
- The output is translated to laser scan data.
- These data can then be plugged in AMCL et voila’ you get your system running.



Goal locations chosen in rviz

Our Glorious Result

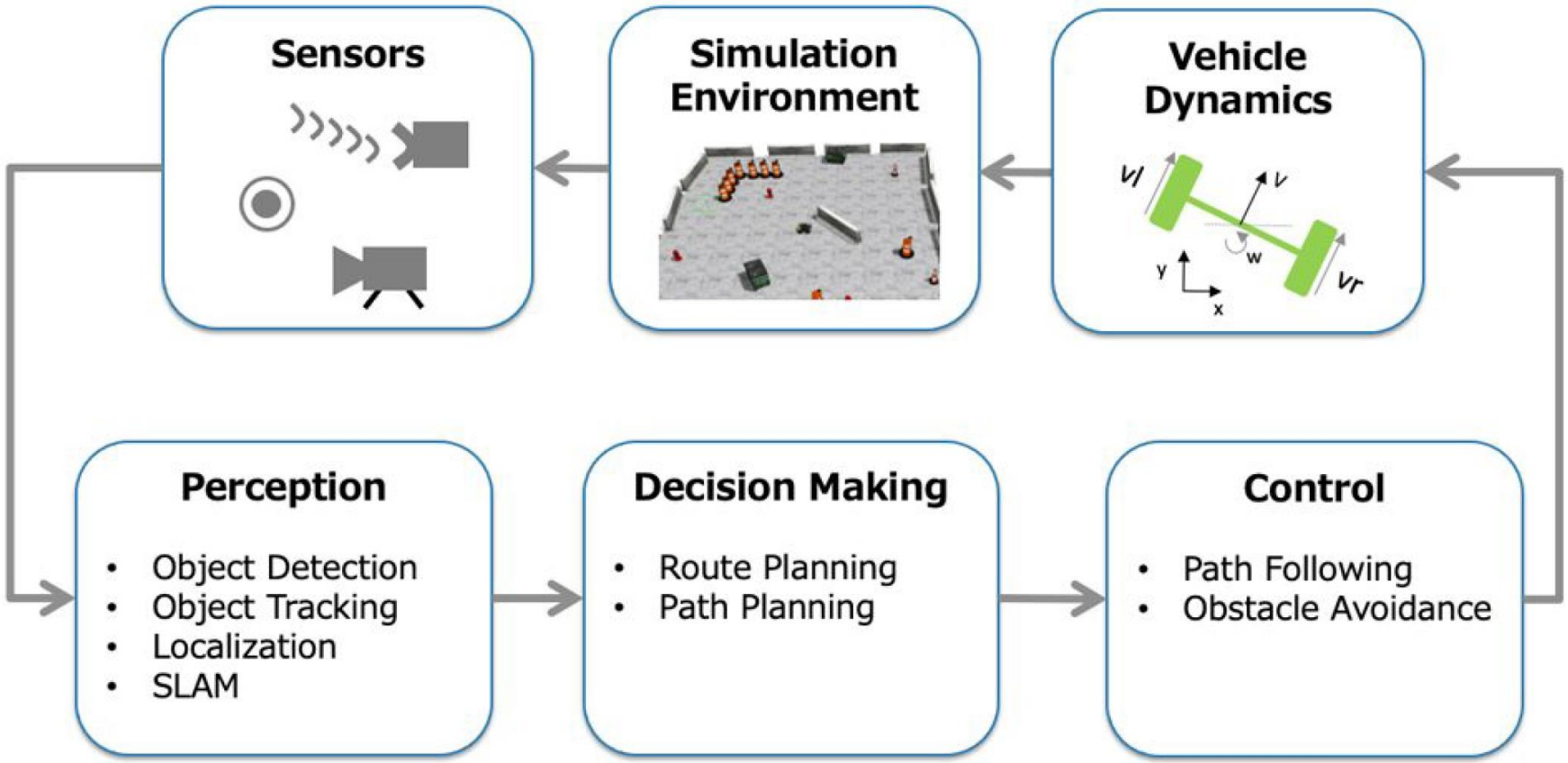
https://www.mathworks.com/campaigns/offers/next/autonomous-mobile-robots.html



Architecture of AMRs.

https://www.mathworks.com/campaigns/offers/next/autonomous-mobile-robots.html

Developing Autonomous Mobile Robots UsingMATLAB and Simulink



Autonomous mobile robot (AMR) system.

System

Sense

Perceive

Decide & Plan

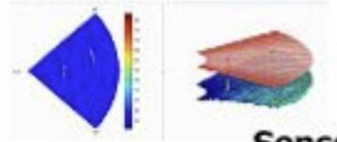
Control

Connect

Control System Toolbox™

HW Support Packages

Phased Array System Toolbox™



Data Acquisition Toolbox™

Simscape™

Aerospace Blockset™

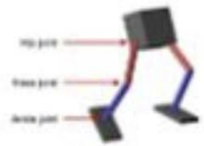
Computer Vision Toolbox™

Statistics and Machine Learning Toolbox™

Deep Learning Toolbox™

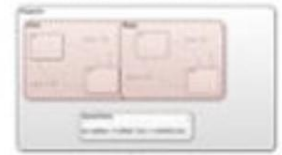
Automated Driving Toolbox™

Reinforcement Learning Toolbox™



Sensor Fusion and Tracking Toolbox™

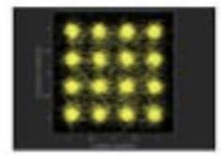
Stateflow®



Model Predictive Control Toolbox™

Communications Toolbox™

WLAN Toolbox™



Robotics System Toolbox™

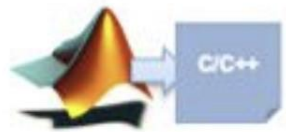
Navigation Toolbox™

ROS Toolbox

ROS 2

Simulink Real-Time™

MATLAB Coder™



Simulink Coder™

Embedded Coder™

HDL Coder™

PLC Coder™

GPU Coder™

- **ON TO CHAPTER 3 DRIVING TURTLEBOT**
- **ON TO CHAPTER 4 NAVIGATING WITH TURTLEBOT**
- **DETAILS OF TRANSFORMS, PROBABILITY, STATE ESTIMATION**