ROS2 Navigation & Linorobot2

(including SLAM)

Paul Bouchier, MSEE, MSCS

- Embedded systems engineer - HW & SW design
- 9 years professional robotics experience
  - Autonomous full-size trucks
  - System test & calibration
  - HW & SW subsystem design
- 15 years personal experience with ROS
- Past & present DPRG president
- 2024 winner - Robocolumbus competition

Overview

Linorobot is a set of SW packages that enables compatible robots to navigate using ROS' map-based navigation with no programming required.

- What is ROS?
- What is Linorobot?
- ROS navigation concepts, including SLAM
- Linorobot architecture
- Demos
- How to build your own linorobot

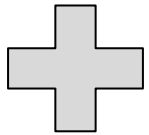
What is ROS?

- Plumbing: messaging & service rqst infrastructure connecting nodes
- Tools: visualize navigation, topics & messages, introspection, log/debug
- Capabilities: Navigation, Path planning (arms & vehicles)
- Social Ecosystem: Community of developers, adjacent products (OpenCV), Technical Steering Committee, Open source, Discourse forum, StackExchange for questions, github for collaboration & issue tracking, annual conference
- 1700 SW packages released on a lifecycle plan with CI testing infrastructure

The ROS Community

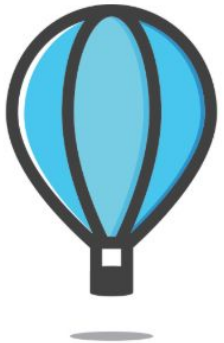
Special Interest Groups:

- ROS-Ag (Agricultural)
- ROS-Space
- Open-RMF (building frameworks, elevators, etc)



open robotics

ROS industrial™



MoveIt

NAV2

GAZEBO

OpenCV

ROS microROS

Contributors, Maintainers

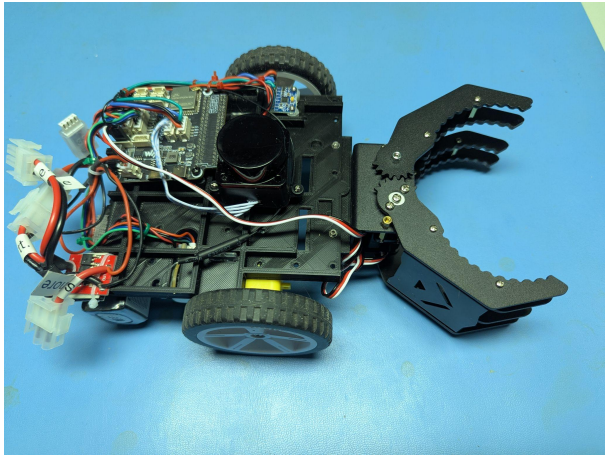
What is linorobot2?

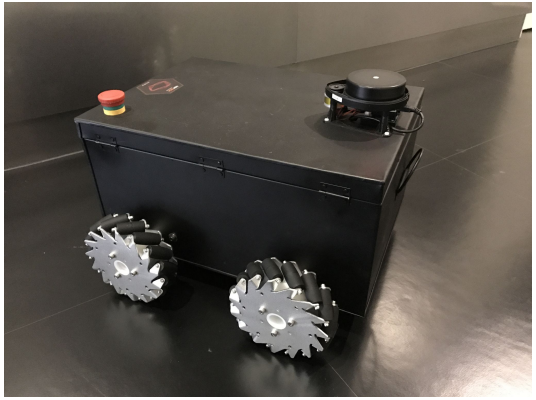
- A SW platform configured by a script & config files for wide HW flexibility.
  - HW flexibility can lead to low cost
- User apps can ask nav2 to navigate the robot somewhere on a map, avoiding obstacles as it goes
- Linorobot2 nodes help nav2 nodes navigate through a map

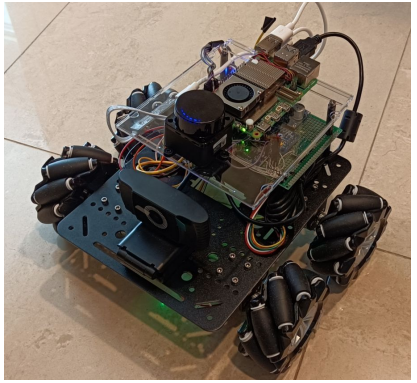
HW Flexibility

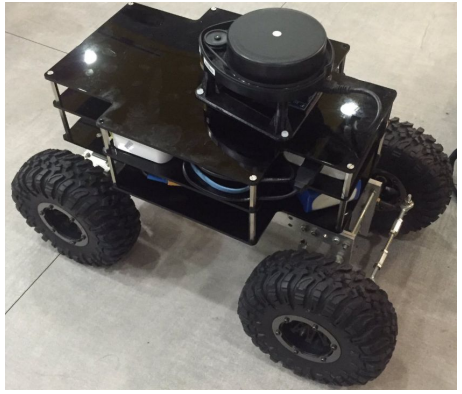
- Robot types: 2WD, 4WD, Mechanum
- Microcontrollers: ESP32, Pico, Teensy (Arduino environment)
- Motor drivers: PWM based (INA/INB/ENA e.g. L298, BTS7960, TB6612, etc, or ESC types e.g. brushless DC)
- Motors: with quadrature encoder
- Lidars over wifi: ldlidar LD19/LD06/STL27L
- Lidars over serial/USB: RPLidar, LDlidar, STlidar, YDlidar, depth cameras
- IMU: GY85, MPU6050 (GY521), MPU9150, MPU9250, QMI8658

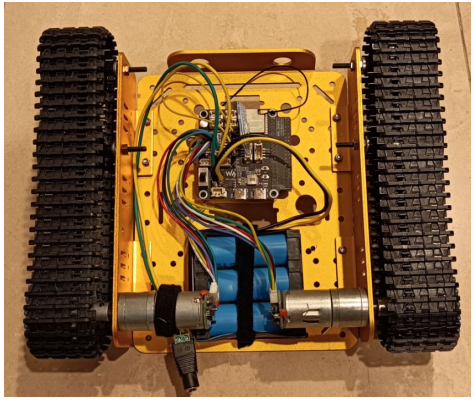
Linorobot2 instances









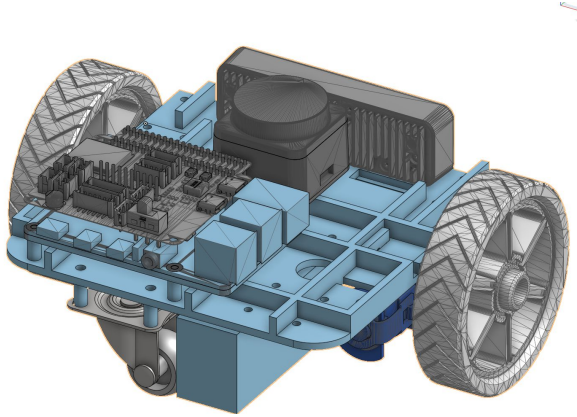


You build the HW, linorobot provides the SW (for compatible robots)

- Created by Juan Miguel Jimeno (Singapore) 2016
- Jazzy + ESP32 port by Thomas Chou (Taiwan) 2024

Minniebot BOM cost

Minimal Parts List			
Description	**Qty**	**Price ea.**	**Cost**
Gear motor w/encoder	2	$7.40	$14.80
Wheel	2	$1.50	$3.00
Waveshare General robot ctrlr	1	$27.99	$27.99
Lidar LD19	1	$59.00	$59.00
Battery, Ovonic 3S 2200mAh	1	$18.99	$18.99
IMU - MPU6050	1	$6.98	$6.98
Totals			**$130.76**



Config shown would be ~$350

Other options for a ROS2 Navigable Robot

- Turtlebot-4 $2k (lite - $1300)
- Makerspet + kaia.ai ~$120
- Yaboom $400. Yahboom has AutomaticAddison tuts.
- HiWonder $600
- Ali Express & others

kaia.ai SW is closest SW package in terms of flexibility. New SW in a container running on Windows.

IMHO there are other reasonable options for a ROS2 robot with nav support, but they all deliver a fixed config. Linorobot2 seems good for flexible HW configuration, BYOR, with a good out-of-the-box experience (not turn-key, but close).

Turtlebot 4 (lite & regular)

Makerspet HW + kaia.ai SW

Yaboom

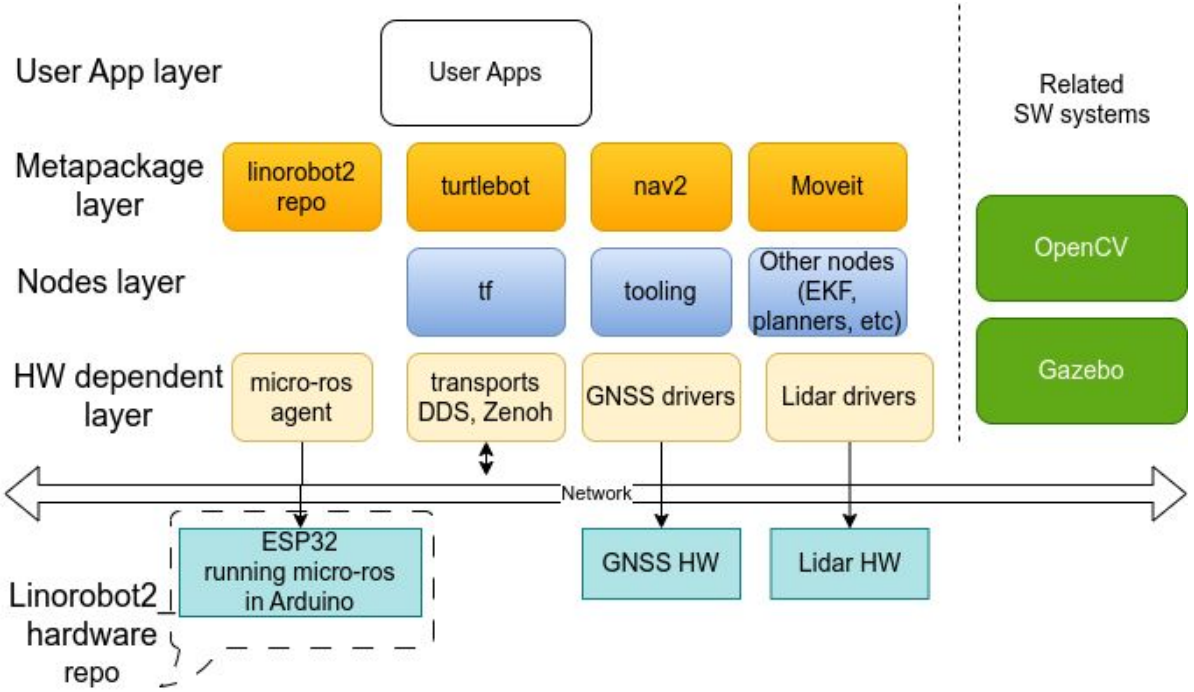
HiWonder

Linorobot in the ROS Package context

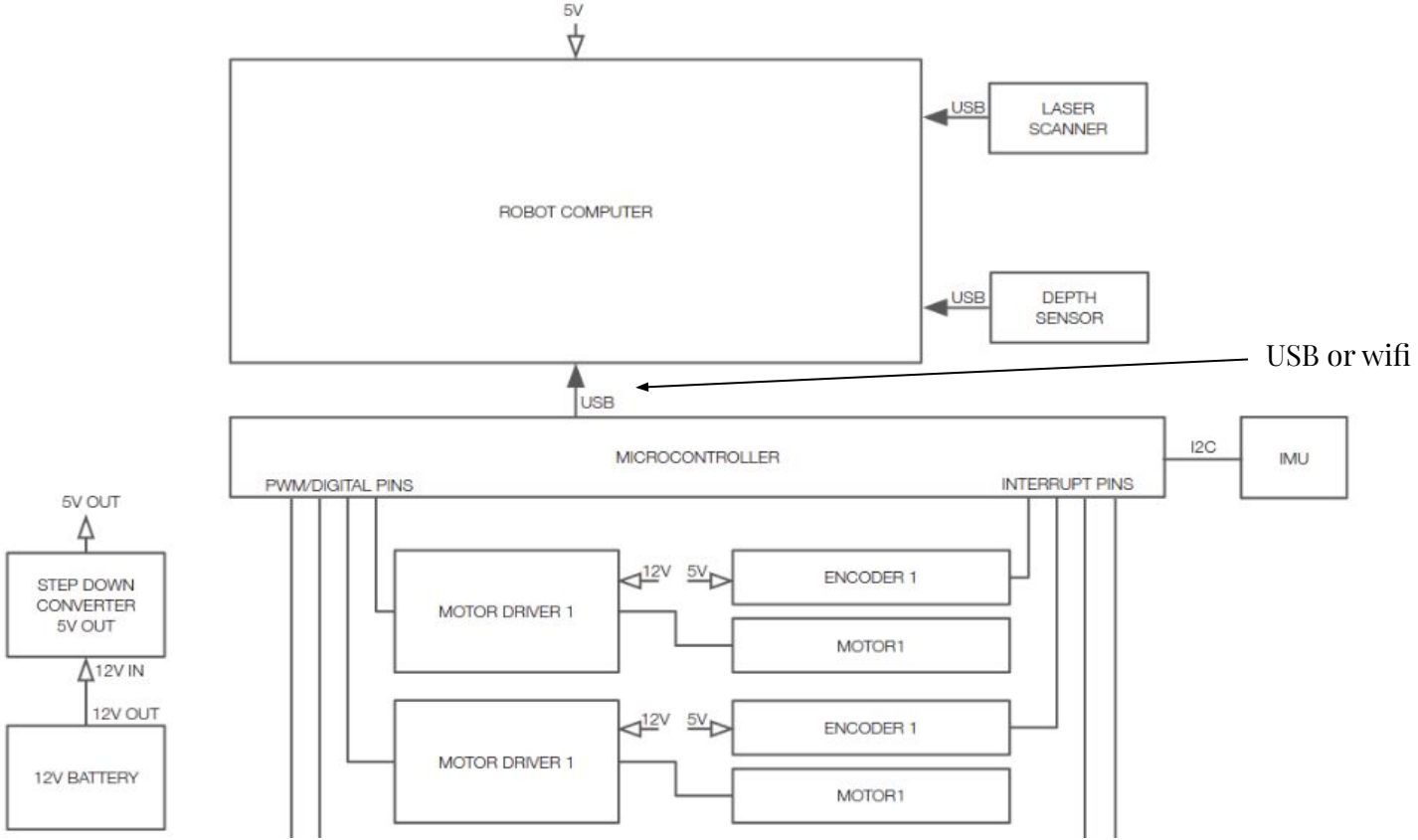
- Metapackage is a collection of packages, each of which may contain nodes that run at the node layer (binaries/processes). Metapackage is a packaging concept.
- Nodes do the work, process data
- HW-dependent nodes expose specific devices to ROS
- **index.ros.org: 1700 packages**
- **2700 repos tagged with #ROS!**

Expect to deal with a large SW system. (You already do: Windows or Linux, compilers & libraries, python, cell phone, Arduino)

- Two repos contain linorobot packages & firmware

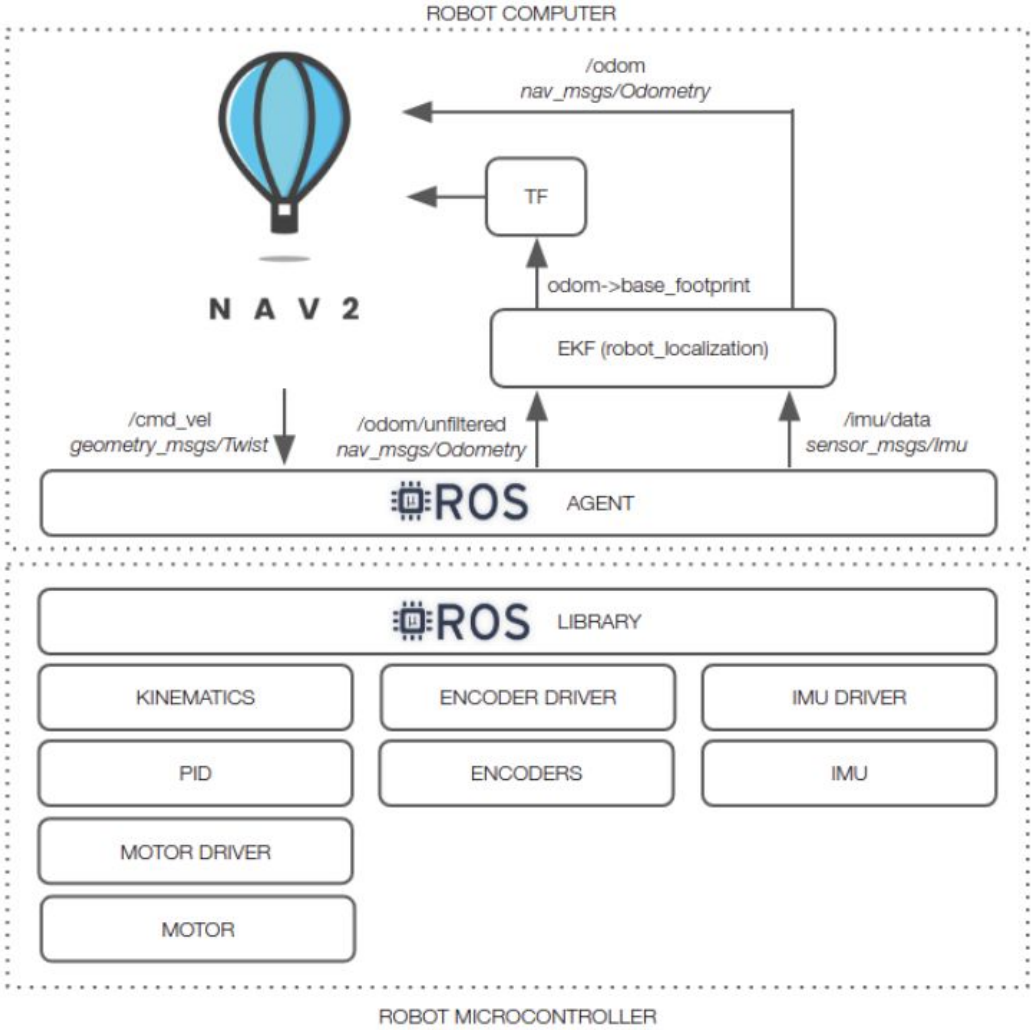


Linorobot System Design (onboard computer version shown)



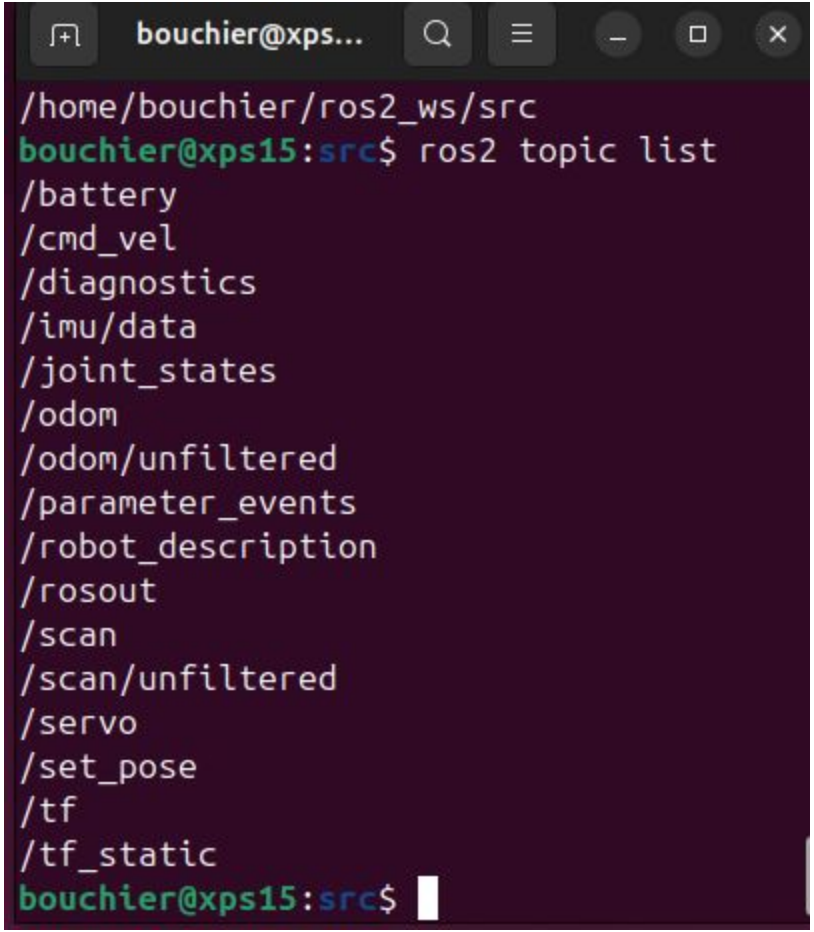
Software Architecture

- ROS nodes run on robot computer
- IMU & motor control (& lidar forwarding) run on robot microcontroller



Demo - micro-ros

- micro-ros agent topics for Linorobot
- Joystick control available
- Minniebot_wifi_bringup
  - ros2 launch linorobot2_bringup bringup.launch.py micro_ros_transport:=udp4 micro_ros_port:=8888 lidar_transport:=udp_server lidar_server_port:=8889
- Servo control:
  - ros2 topic pub /servo example_interfaces/msg/Int32 "{data: 1100}"

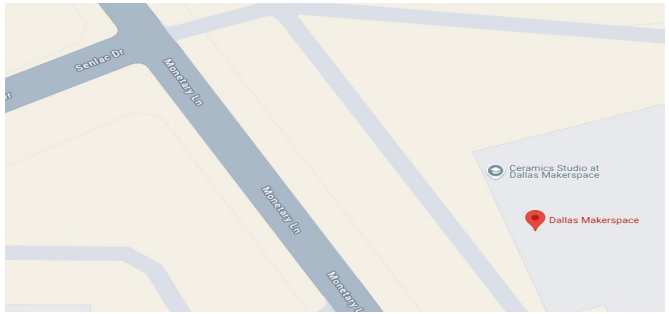


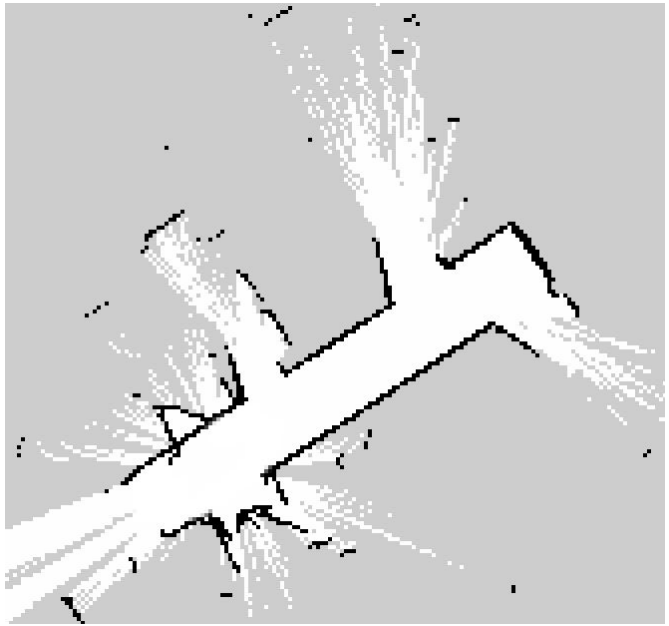
Navigation Levels

- Navigation primary goal: move a robot from point A to point B
  - Respond to obstacles - ODOA (Obstacle Detection / Obstacle Avoidance
- Level 1: Dead reckoning: odometry/IMU within robot-local frame
  - Example: Four Square contest
  - Waypoints are relative to robot start position/heading
  - Issue: distance & heading drift, initial heading
- Level 2: Navigate within a global frame (e.g. GPS)
  - Example: RoboColumbus navigating between waypoints& cones, 6-can localizing bots
  - Eliminates drift & initial heading issues
  - Issue: Route must be manually planned
- Level 3: Navigate within a map that's in a global frame (e.g. Google maps, robo-vacuum in a house), with automatic path planning.
  - Level 3 and Linorobot is focus of this talk

Map-based navigation

- What is a map?
  a. A 2d representation of the environment: fixed obstacles (walls, furniture, road boundaries) and possible destinations (kitchen, bedroom) and navigable areas, generally with some resolution (e.g. 5cm default in nav2)
- 3 Issues:
  a. How to make a map (SLAM)
  b. How to plan a path within a map
  c. ODOA (the map does not include dynamic obstacles)
- There are well-known algorithms for global path planning within a map (A* etc)
- ODOA is handled by a local path planner which plans deviations from the planned global path to avoid an obstacle



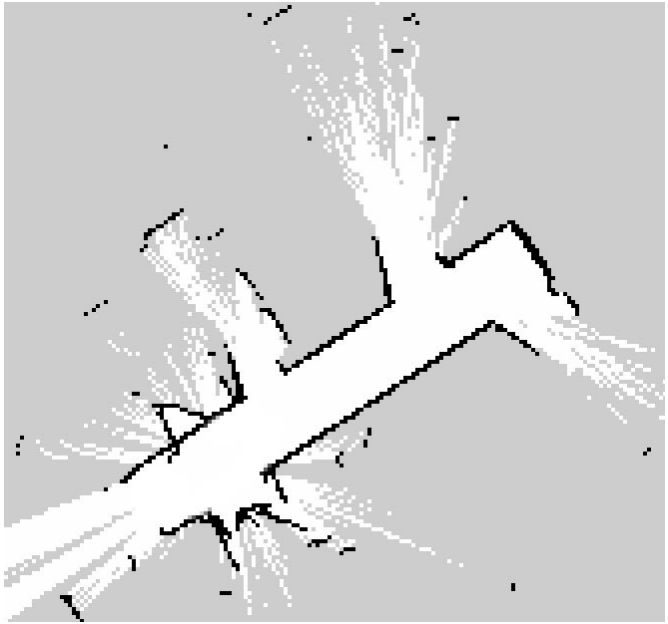


SLAM - Simultaneous Localization & Mapping

- Demo SLAM with linorobot
  - ros2 launch linorobot2_bringup bringup.launch.py micro_ros_transport:=udp4 micro_ros_port:=8888 lidar_transport:=udp_server lidar_server_port:=8889
  - ros2 launch linorobot2_navigation slam.launch.py rviz:=true
- Concept: SLAM discovers a map as user manually drives robot OR asks it to navigate within its map
- After map is made you would save the map
  - ros2 run nav2_map_server map_saver_cli -f <map_name> --ros-args -p save_map_timeout:=10000.
- Thereafter navigator would plan a path for a mission (and replan for ODOA) using the map. The path would only go through navigable areas. UNMAPPED by policy
- How SLAM works: tracking the relative motion of key points in lidar scan, while using odometry, IMU to help keep robot localized relative to those points, and discovering more points as it goes.
  - BUT if it loses localization to IMU/odometry it can “discover” new lidar features that it lost track of, and make a wrong map

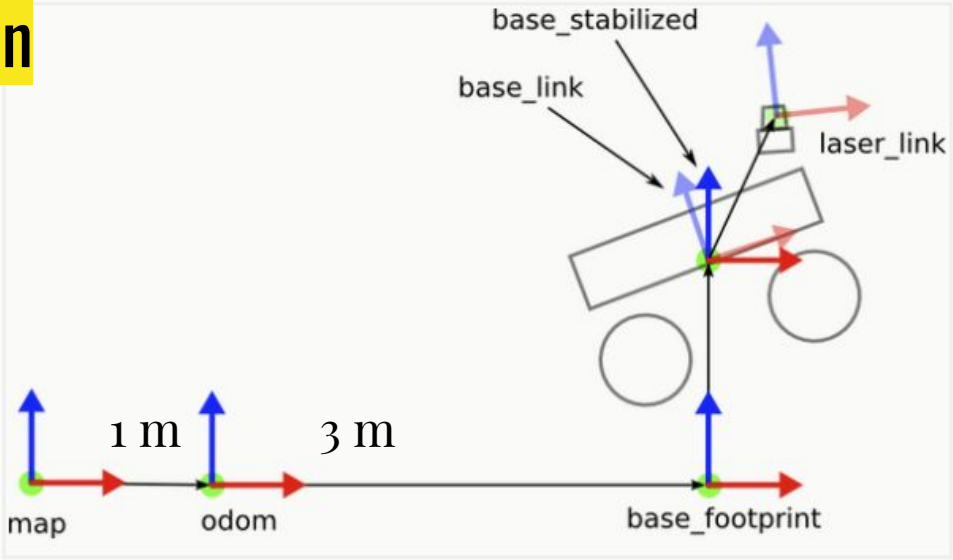
Global Costmap, Local Costmap

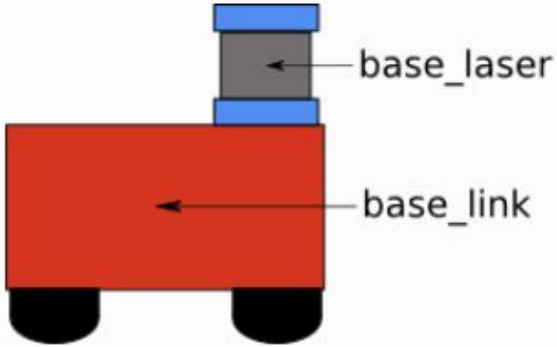
- **Global costmap:** This costmap is used to generate long term plans over the entire environment....for example, to calculate the shortest path from point A to point B on a map.
  - Created with SLAM or editor
  - Used by global planner
- **Local costmap:** This costmap is used to generate short term plans over the environment....for example, to avoid obstacles.
  - Dynamically created from lidar
  - Used by local planner
- Costmaps need to be aligned!!!

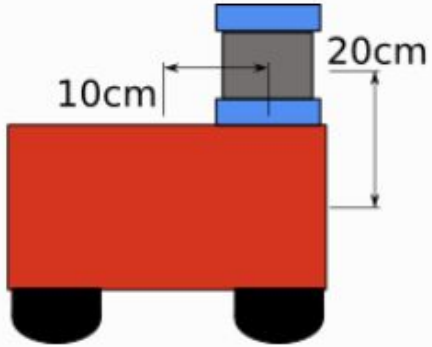


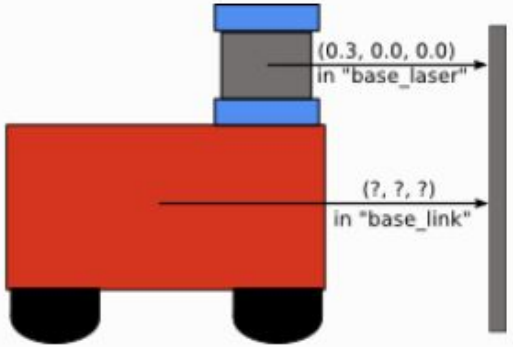
Coordinate Frames & Localization

- Pose: Position & Orientation within a frame
- Wall pose within lidar frame
- Lidar pose within robot frame
- Robot pose within odom frame
- Odom frame pose within map frame



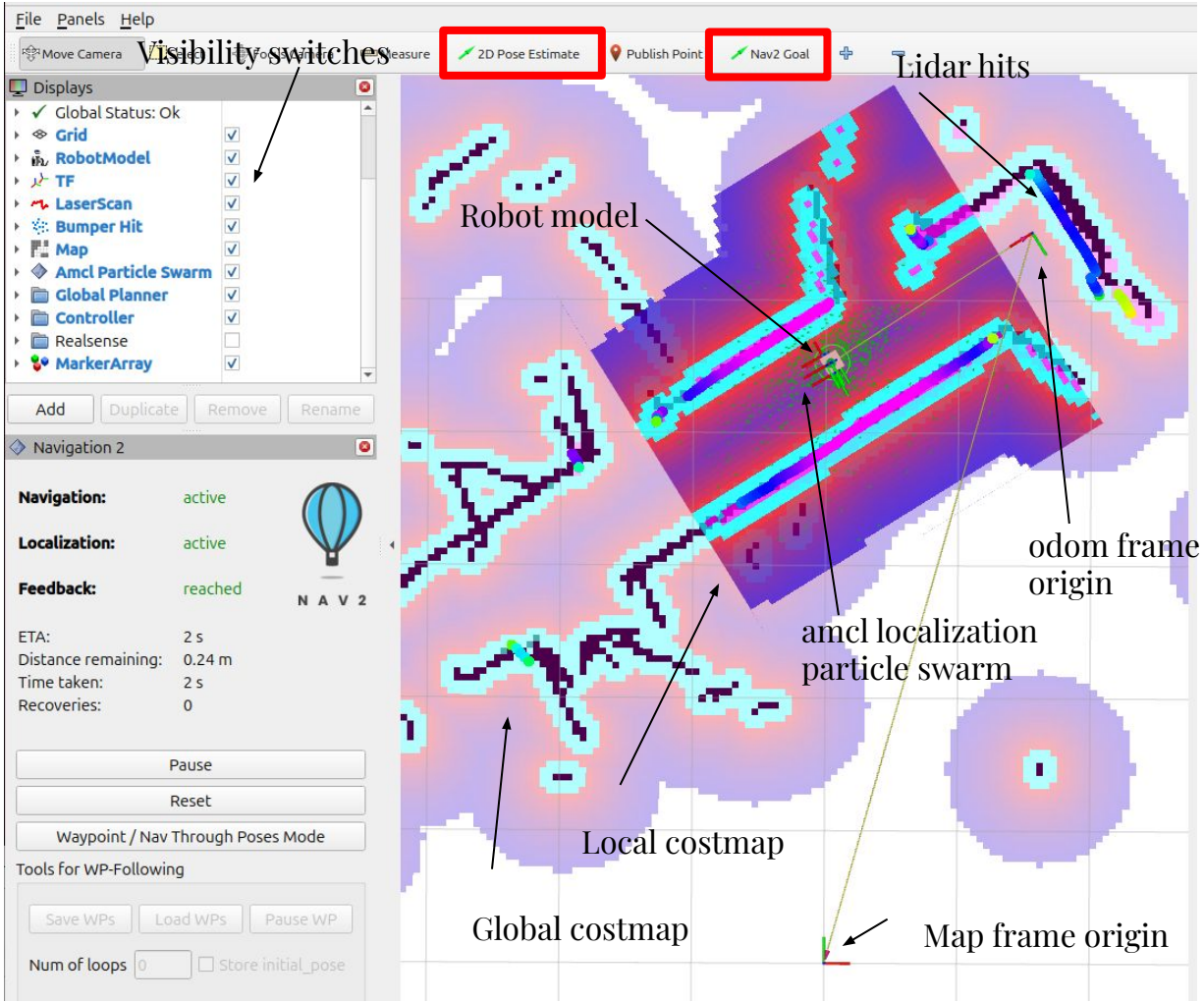






Navigation demo: understanding rviz

- Global costmap has “potential gradient”



Navigation demo

- Starting nav:
  - Robot bringup: ros2 launch linorobot2_bringup bringup.launch.py micro_ros_transport:=udp4 micro_ros_port:=8888 lidar_transport:=udp_server lidar_server_port:=8889
  - ros2 launch linorobot2_navigation navigation.launch.py rviz:=true map:=<path to yaml>
- Setting (and resetting) initial pose in a map
- Navigate to a point
- Navigate through waypoints with obstacle avoidance

Nav2 API

Nav2 wouldn't be much use if it required rviz to move the robot. Nav2 has an API for user apps. A little programming is all that's required.

- Instantiate navigator
- Define goal_pose
- Ask navigator to goToPose

```
def main():
    rclpy.init()

    navigator = BasicNavigator()
    navigator.waitUntilNav2Active()

    goal1_pose = PoseStamped()
    goal1_pose.header.frame_id = 'map'
    goal1_pose.header.stamp = navigator.get_clock().now().to_msg()

    goal1_pose.pose.position.x = 0.0
    goal1_pose.pose.position.y = -1.0
    goal1_pose.pose.orientation.w = 1.0
    goal1_pose.pose.orientation.z = 0.0

    navigator.goToPose(goal1_pose)
```

- Loop waiting for completion or ^C

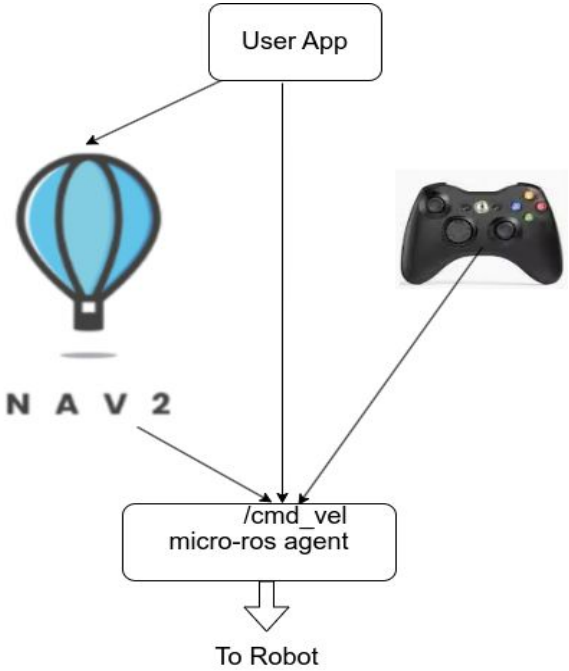
```
try:
    while not navigator.isTaskComplete():
        feedback = navigator.getFeedback()
except KeyboardInterrupt:
    print('Navigation interrupted by user!')
    navigator.cancelTask()

finally:
    result = navigator.getResult()
    if result == TaskResult.SUCCEEDED:
        print('Goal succeeded!')
        return True
    elif result == TaskResult.CANCELED:
        print('Goal was canceled!')
    elif result == TaskResult.FAILED:
        print('Goal failed!')
    else:
        print('Goal has an invalid return status!')
    return False

exit(0)
```

Mix'n'Match with Nav2: the pièce de résistance!

- Demo: joystick-driven motion while nav2 continues to track pose, intermixed with nav goals
- User app can ask Nav2 to goToPose(pose1), then take the reins and do a custom motion (e.g. advance and grab can), then hand back to Nav2 to goToPose(pose2)



Is that magic or what???

Workflow for making a linorobot2

https://github.com/hippo5329/linorobot2_hardware/wiki

1. Install the software (ROS-jazzy, linorobot2 repos).
2. Test micro-ROS agent connection before wiring and building the robot.
3. Connect IMU and run test_sensors.
4. Connect micro-ROS agent again and check /imu/data topic.
5. Design & build your robot and customize configuration file.
6. Test the motors and encoder. Test the sensors. Test the Lidar.
7. Launch robot bringup.
8. Use a keyboard to move the robot. Use a gamepad to move the robot.
9. Launch slam and save the map.
10. Launch navigation with the map.
11. Write some application SW to navigate as needed. 😂

Examine config file

```
//define your robot' specs here
#define MOTOR_MAX_RPM 150                    // motor's max RPM
#define MAX_RPM_RATIO 0.85                   // max RPM allowed for each MAX_RPM_ALL
#define MOTOR_OPERATING_VOLTAGE 6           // motor's operating voltage (used to ca
#define MOTOR_POWER_MAX_VOLTAGE 12           // max voltage of the motor's power sou
#define MOTOR_POWER_MEASURED_VOLTAGE 12      // current voltage reading of the power
#define COUNTS_PER_REV1 1800                 // wheel1 encoder's no of ticks per rev
#define COUNTS_PER_REV2 1800                 // wheel2 encoder's no of ticks per rev
#define COUNTS_PER_REV3 450                  // wheel3 encoder's no of ticks per rev
#define COUNTS_PER_REV4 450                  // wheel4 encoder's no of ticks per rev
#define WHEEL_DIAMETER 0.08                  // wheel's diameter in meters
#define LR_WHEELS_DISTANCE 0.160             // distance between left and right whee
#define PWM_BITS 10                          // PWM Resolution of the microcontrolle
#define PWM_FREQUENCY 20000                  // PWM Frequency
#define SERVO_BITS 12                        // Servo PWM resolution
#define SERVO_FREQ 50                        // Servo PWM frequency

// INVERT ENCODER COUNTS
#define MOTOR1_ENCODER_INV true
#define MOTOR2_ENCODER_INV true
#define MOTOR3_ENCODER_INV false
#define MOTOR4_ENCODER_INV false

// INVERT MOTOR DIRECTIONS
#define MOTOR1_INV true
#define MOTOR2_INV true
#define MOTOR3_INV false
#define MOTOR4_INV false

// ENCODER PINS
#define MOTOR1_ENCODER_A 34
#define MOTOR1_ENCODER_B 35
```

Linorobot2: The Good

- Map-based navigation
- Variety of supported motor drivers, sensors
- Good tools for logging & analyzing data from your robot
- ROS is built around composable nodes. (Nodes should do one thing well, and be able to be composed into a system) - this makes it extensible.
- Network-based

Linorobot2: The Bad & The Ugly

- There isn't a branch labeled "Jazzy". BAD.
- ROS2 & micro-ros are currently based on DDS. It may not work with some home routers. UGLY
- Thomas Chou's forks of the upstream repos have diverged from upstream. I've made changes to Chou's repos for real robots while Jemeno has made changes to the upstream repo for simulation. THEY NEED TO RE-CONVERGE! UGLY

Why should I use ROS2 Navigation with Linorobot2?

It makes you more powerful!

- Come join me exploring linorobot2 at DPRG Robot Builders Night Virtual meetings
- All my robots will run linorobot2 soon

Support info

References

Thomas Chou's Jazzy linorobot2_hardware repo & wiki:

https://github.com/hippo5329/linorobot2_hardware

https://github.com/hippo5329/linorobot2_hardware/wiki

Thomas Chou's Jazzy linorobot2 repo

https://github.com/hippo5329/linorobot2

Automatic Addison Tutorials:

- All tutorials: https://automaticaddison.com/tutorials/
- ROS2 Navigation: https://automaticaddison.com/tutorials/#Navigation

Nav2 Simple Commander API: https://docs.nav2.org/commander_api/index.html

Optional configuration

- Set up syslog server to record logs from microcontroller
- Set up OTA FW upgrade