



EasyNav: un sistema de navegación flexible y modular para ROS 2

Francisco Martín Rico, Francisco Miguel Moreno Olivo, Juan Carlos Manzanares Serrano, José Miguel Guerrero Hernández, Juan Sebastián Cely Gutiérrez, Esther Aguado, Francisco José Romero Ramírez y Juan Diego Peña



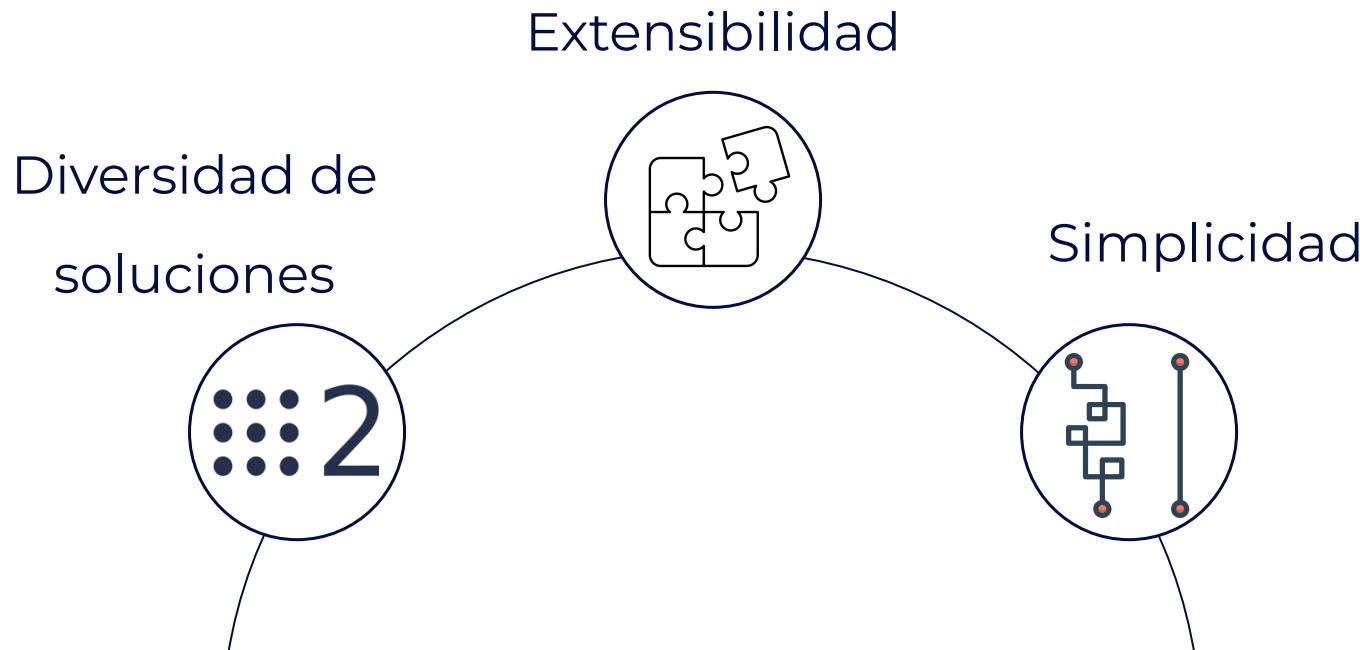
Contenido

- ◇ Motivación y criterios de diseño
- ◇ Arquitectura
 - Estructura de datos: **NavState**
 - Modelo de ejecución
 - Modularidad: Estructura de un plugin
 - Gestor de objetivos
- ◇ **Ejercicios:** Navegación en **interiores**
 1. Lanzamiento con configuración básica
 2. Desarrollo de un Plugin
 3. Gestión de objetivos: Patrolling

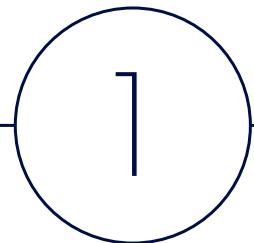
Contenido

- ◇ Nueva representación: **NavMap**
- ◇ **Caso de uso:** Uso de las representaciones en exteriores
- ◇ **Ejercicios:** Uso de NavMap y Bonxai
 - 4. Localización en NavMap y visualización en Bonxai

Alternativa a Nav2



Diseño



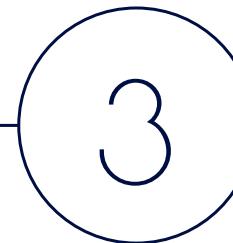
Flexibilidad

- **Representación variable:** costmaps, superficies navegables, octomap, etc.
- **Diseño modular:** funcionalidad por **plugins** combinables y extensibles.



Determinismo

- Sincronía por llamadas directas a **funciones**: predecibilidad y baja latencia.
- Procesado en **tiempo real**: baja latencia percepción - acción.



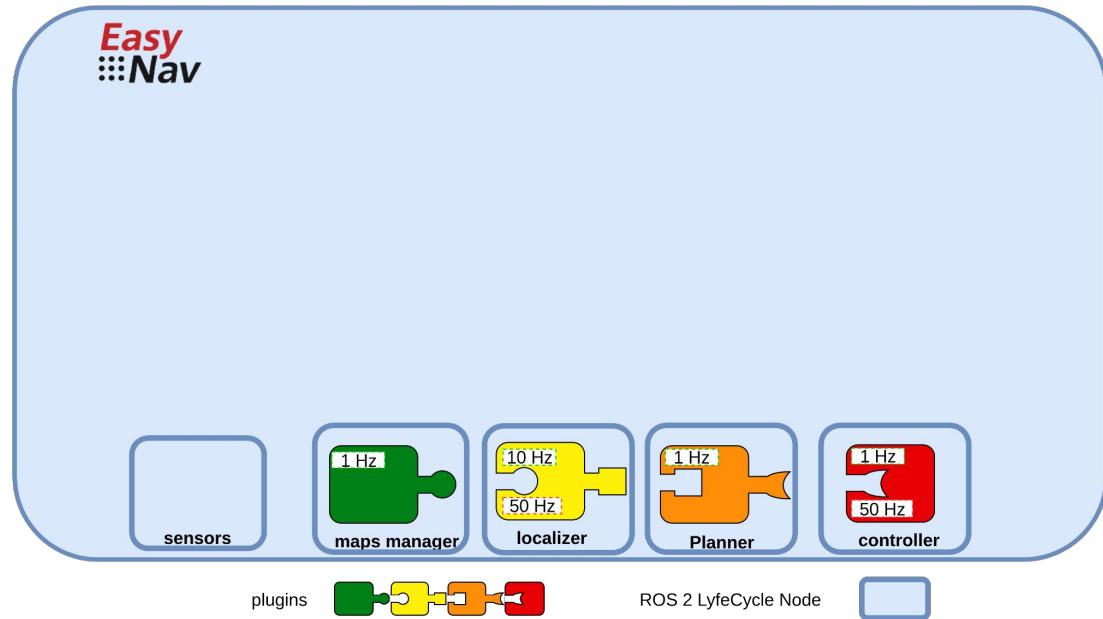
Simplicidad

- **Ligero** y fácil de desplegar: solo un nodo y un archivo de parámetros.
- **Simulación** lista para usar: amplia disponibilidad de PlayGrounds con **distintos robots y entornos**.

Arquitectura

Un **nodo principal** (ROS 2 Lifecycle)
que **coordina cinco** nodos
subordinados:

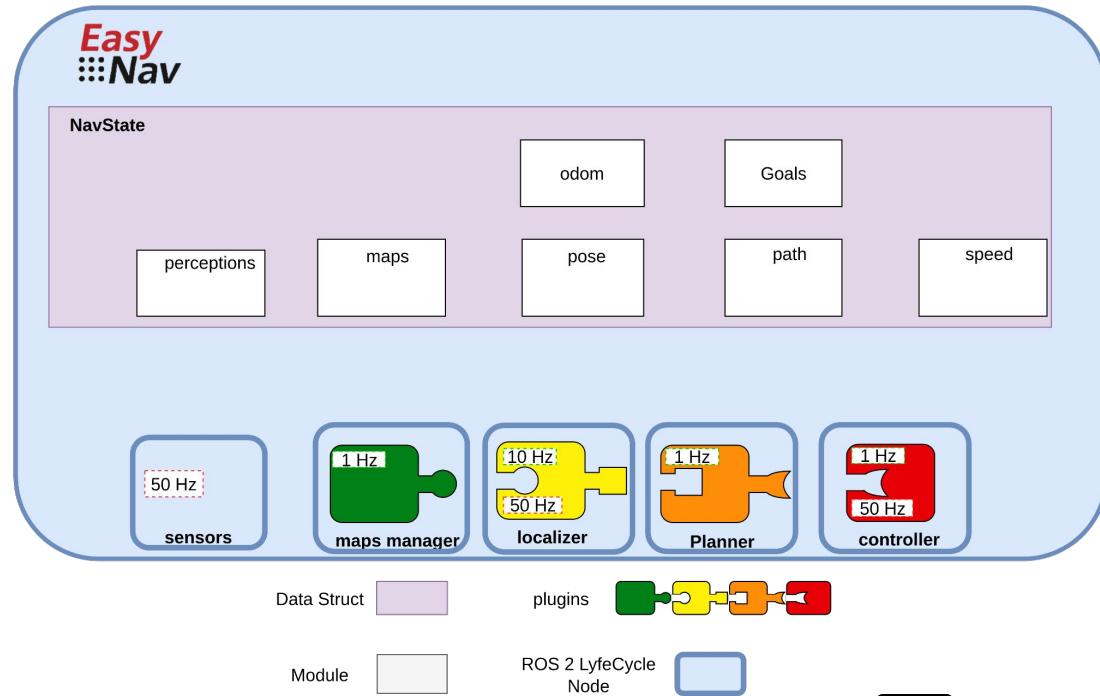
- Gestión **sensorial**
- Gestión de **mapas**
- **Localización**
- **Planificador** de rutas
- **Control**



Estructura de datos

○ **NavState**: Pizarra compartida

- Información perceptual
- Posición estimada del robot
- Mapa
- Comandos de velocidad
- Objetivo de navegación actual
- Diagnóstico



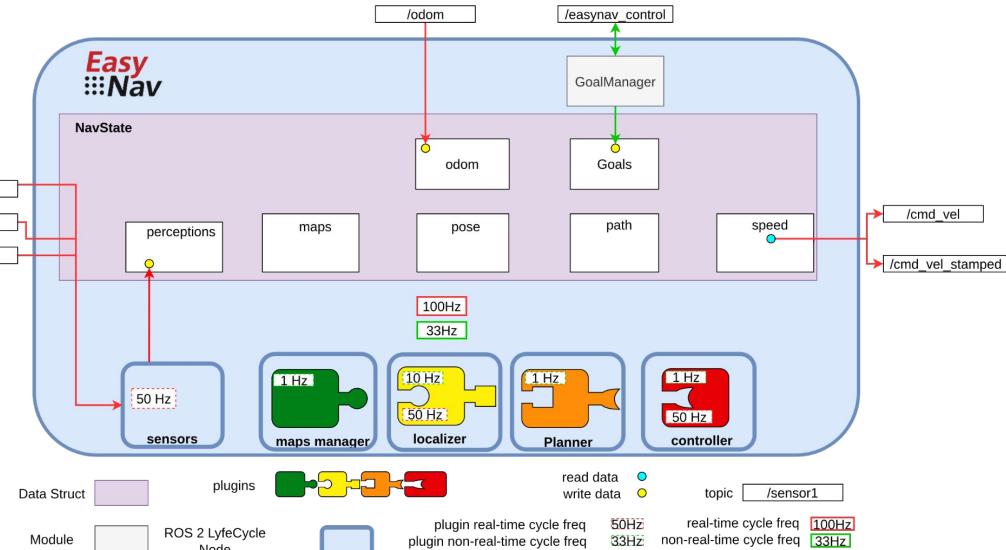
Modelo de ejecución

○ Dos ciclos de control

- Tiempo real:
percepción - posición - velocidad
- Fuera de tiempo real:
mapa - corrección localización - planificación

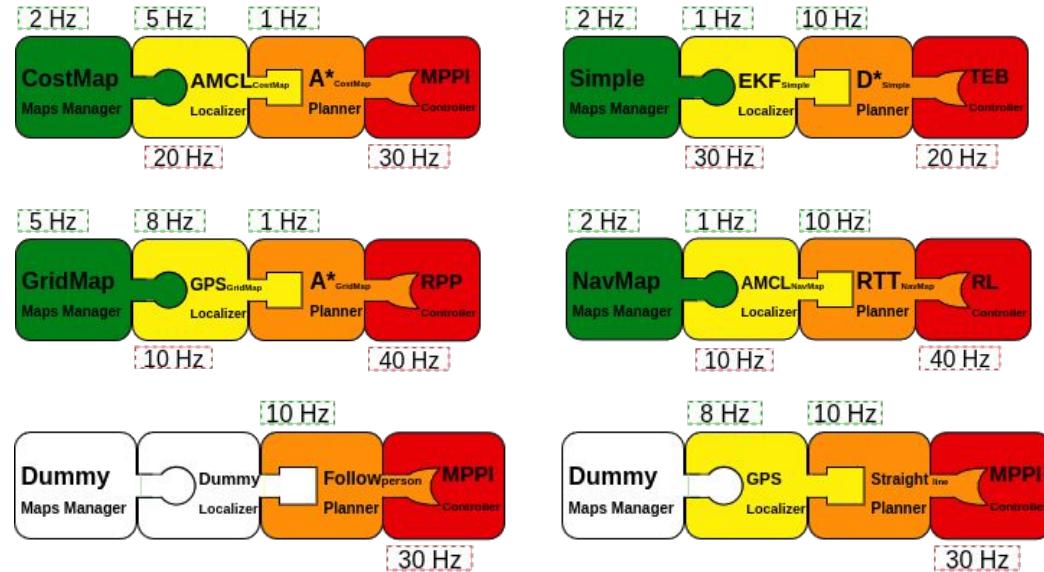
○ Nueva **percepción** desencadena ciclo tiempo real

○ Frecuencias **parametrizables** en ambos ciclos



Plugins

- Diferentes **combinaciones** de componentes EasyNav
- Importante: asegurar **compatibilidad** entre plugins
 - Representación
 - Plugins independientes
 - Dummy plugins



Plugins: Configuración

```
system_node:  
  ros_parameters:  
    use_sim_time: true  
    position_tolerance: 0.1  
    angle_tolerance: 0.05
```

Plugins: Configuración

```
maps_manager_node:  
  ros__parameters:  
    use_sim_time: true  
    map_types: [simple]  
    simple:  
      freq: 10.0  
      plugin: easynav_simple_maps_manager/SimpleMapsManager  
      package: easynav_indoor_testcase  
      map_path_file: maps/home.map
```

```
planner_node:  
  ros__parameters:  
    use_sim_time: true  
    planner_types: [simple]  
    simple:  
      freq: 0.5  
      plugin: easynav_simple_planner/SimplePlanner  
      robot_radius: 0.3
```

```
sensors_node:  
  ros__parameters:  
    use_sim_time: true  
    forget_time: 0.5  
    sensors: [laser1]  
    perception_default_frame: odom  
    laser1:  
      topic: /scan_raw  
      type: sensor_msgs/msg/LaserScan  
      group: points
```

```
controller_node:  
  ros__parameters:  
    use_sim_time: true  
    controller_types: [simple]  
    simple:  
      rt_freq: 30.0  
      plugin: easynav_simple_controller/SimpleController  
      max_linear_speed: 0.6  
      max_angular_speed: 1.0  
      look_ahead_dist: 0.2  
      k_rot: 0.5
```

```
localizer_node:  
  ros__parameters:  
    use_sim_time: true  
    localizer_types: [simple]  
    simple:  
      rt_freq: 50.0  
      freq: 5.0  
      reseed_freq: 1.0  
      plugin: easynav_simple_localizer/AMCLLocalizer  
      num_particles: 100  
      noise_translation: 0.05  
      noise_rotation: 0.1  
      noise_translation_to_rotation: 0.1  
      initial_pose:  
        x: 0.0  
        y: 0.0  
        yaw: 0.0  
        std_dev_xy: 0.1  
        std_dev_yaw: 0.01
```

Plugins : Dummy

```
maps_manager_node:  
  ros_parameters:  
    use_sim_time: true  
  map_types: [costmap]  
  costmap:  
    freq: 10.0  
    plugin: easynav_costmap_maps_manager/CostmapMapsManager  
    package: my_maps_pkg  
    map_path_file: maps/office.yaml
```

```
sensors_node:  
  ros_parameters:  
    use_sim_time: true  
    forget_time: 0.5  
  sensors: [laser1]  
  perception_default_frame: odom  
  laser1:  
    topic: /scan_raw  
    type: sensor_msgs/msg/LaserScan  
    group: points
```

```
system_node:  
  ros_parameters:  
    use_sim_time: true  
    position_tolerance: 0.1  
    angle_tolerance: 0.05
```

```
controller_node:  
  ros_parameters:  
    use_sim_time: true  
  controller_types: [dummy]  
  dummy:  
    rt_freq: 30.0  
    plugin: easynav_controller/DummyController  
    cycle_time_nort: 0.01  
    cycle_time_rt: 0.001
```

```
localizer_node:  
  ros_parameters:  
    use_sim_time: true  
  localizer_types: [dummy]  
  dummy:  
    rt_freq: 50.0  
    freq: 5.0  
    reseed_freq: 0.1  
    plugin: easynav_localizer/DummyLocalizer  
    cycle_time_nort: 0.01  
    cycle_time_rt: 0.001
```

```
planner_node:  
  ros_parameters:  
    use_sim_time: true  
  planner_types: [dummy]  
  dummy:  
    freq: 1.0  
    plugin: easynav_planner/DummyPlanner  
    cycle_time_nort: 0.2  
    cycle_time_rt: 0.001
```

Objetivos de Navegación

○ Interfaz mediante **topics /easynav_control**

- Mensaje propio con:
 - Comandos
 - Información de estado: solicitud, aceptación, rechazo, feedback, etc.

○ **GoalManagerClient**: Clase que implementa el protocolo → **ej patrolling**

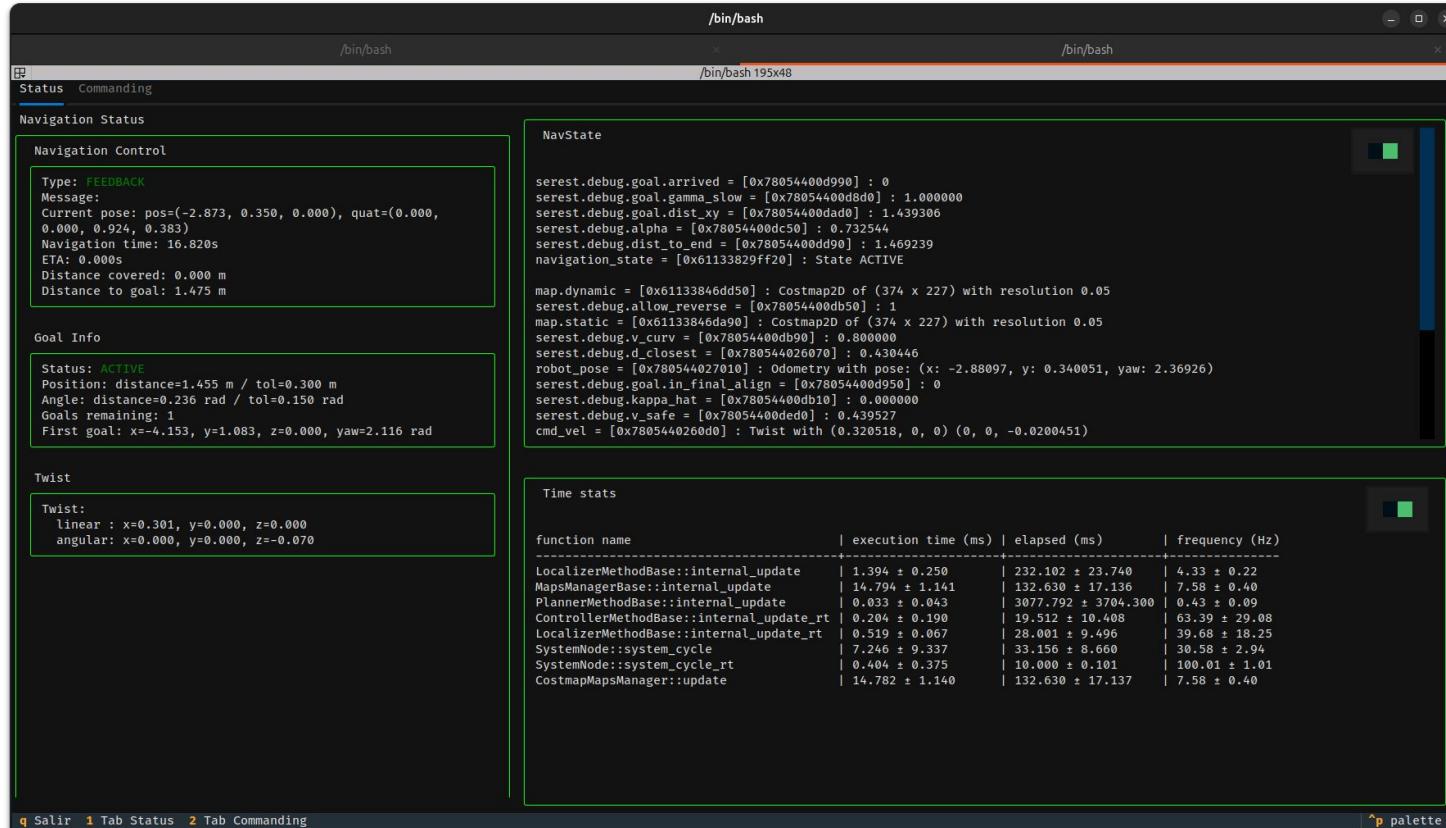
○ Topic **/goal_pose**

- Eq. request en /easynav_control

```
uint8 REQUEST=0  
uint8 REJECT=1  
uint8 ACCEPT=2  
uint8 FEEDBACK=3  
uint8 FINISHED=4  
uint8 FAILED=5  
uint8 CANCEL=6  
uint8 CANCELLED=7  
uint8 ERROR=8
```

```
uint8 type  
std_msgs/Header header  
int64 seq  
string user_id  
string nav_current_user_id  
nav_msgs/Goals goals  
string status_message  
geometry_msgs/PoseStamped current_pose  
builtin_interfaces/Duration navigation_time  
builtin_interfaces/Duration estimated_time_remaining  
float32 distance_covered  
float32 distance_to_goal
```

Terminal User Interface



Command Line Interface

```
ros2 easynav
```

```
goal_info|nav_state|twist|navigation_control|timestats|plugins
```

```
EasyNav Navigation Control Info:  
  
Type: FEEDBACK  
Message:  
Current pose: pos=(-0.874, 2.726, 0.000), quat=(0.000, 0.000, 0.936, 0.352)  
Navigation time: 31.805s  
ETA: 0.000s  
Distance covered: 0.000 m  
Distance to goal: 0.135 m
```

Desarrollo de plugins

Interfaces (Topics and Services)

Publications

Direction	Topic	Type	Purpose	QoS
Publisher	<node_fqn>/<plugin>/path	nav_msgs/msg/Path	Publishes the computed path from start to goal.	depth=10

This plugin does not create subscriptions or services directly; it reads inputs via NavState.

NavState Keys

Key	Type	Access	Notes
goals	nav_msgs::msg::Goals	Read	Goal list used as planner targets.
map.dynamic	Costmap2D	Read	Dynamic costmap used for A* search.
robot_pose	nav_msgs::msg::Odometry	Read	Current robot pose used as the start position.
path	nav_msgs::msg::Path	Write	Output path to follow.

Uso de plugins

Plugin (pluginlib)

- **Plugin Name:** easynav_costmap_planner/CostmapPlanner
- **Type:** easynav::CostmapPlanner
- **Base Class:** easynav::PlannerMethodBase
- **Library:** easynav_costmap_planner
- **Description:** A default Costmap2D-based A* path planner implementation.

Parameters

All parameters are declared under the plugin namespace, i.e., /<node_fqn>/easynav_costmap_planner/CostmapPlanner/....

Name	Type	Default	Description
<plugin>.cost_factor	double	2.0	Scaling factor applied to costmap cell values to compute traversal costs.
<plugin>.inflation_penalty	double	5.0	Extra penalty added to inflated/near-obstacle cells to push paths away from obstacles.
<plugin>.cost_axial	double	1.0	Base movement cost for axial (N/E/S/W) steps.
<plugin>.cost_diagonal	double	1.41	Base movement cost for diagonal steps (approx. $\sqrt{2}$).
<plugin>.continuous_replan	bool	true	If true, re-plans continuously as the map/goal changes; if false, plans once per request.

Caso de Uso : Navegación en **interiores y exteriores**

Ejercicios

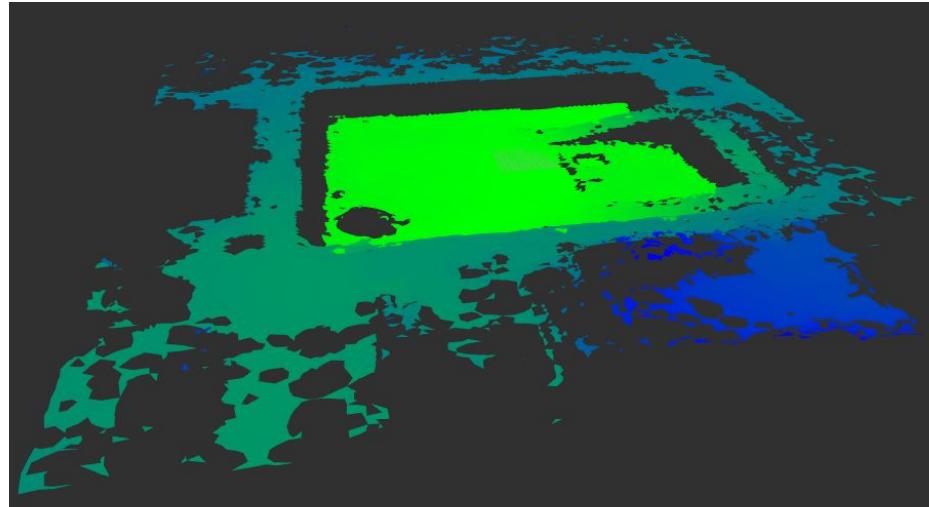
- Los siguientes ejercicios están detallados en el [repositorio](#) del workshop:
 0. Navegando la **documentación**: <https://easynavigation.github.io/>
 1. Lanzamiento con **configuración** básica
 2. Desarrolladores: Plugins y escritura de datos en **NavState**
 3. Gestión de **objetivos**: Patrolling en GridMap

NavMap :

Una representación basada
en superficies navegables

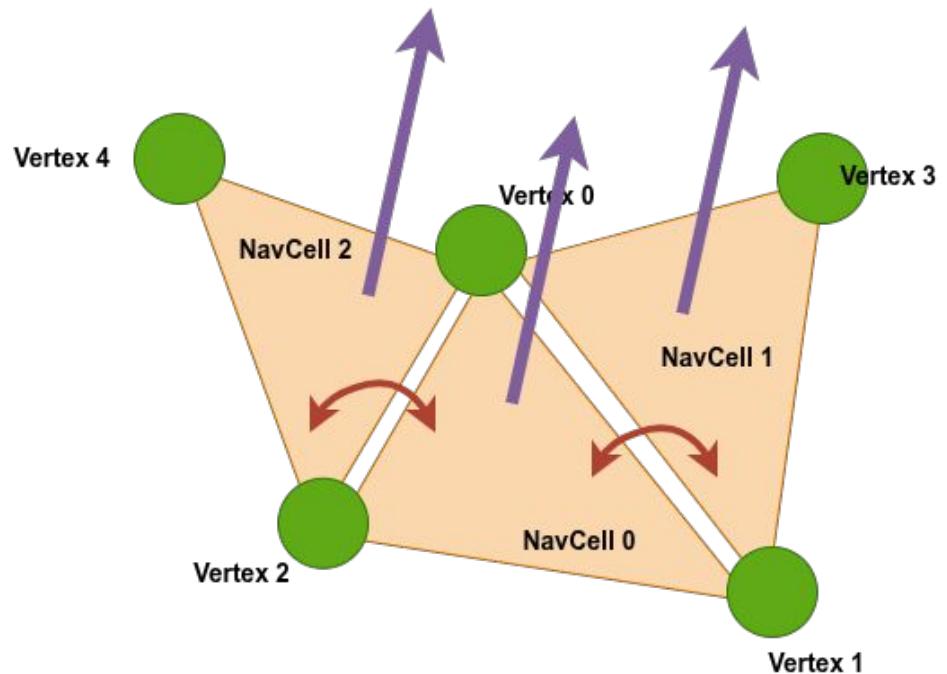
NavMap

- Representación del mundo basada en superficies navegables
- Cada superficie tiene la posibilidad de almacenar información geométrica y de datos a través de capas (layers)



NavCell

- Es la unidad sobre la cual se construyen las superficies
- Se construye a partir de 3 vértices
- Una de sus características más importantes es la vecindad, se puede saber que NavCells están adyacentes
- Otra característica importante es la normal a cada NavCell que permite obtener información de su orientación



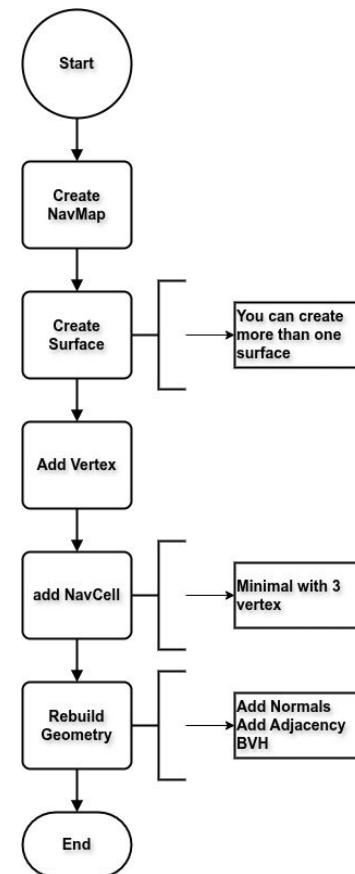
Crear un NavMap

- La jerarquía de elementos en NavMap

es:

- NavMap
 - Surface
 - NavCell
 - Vertex
 - Layers

- Esta organización permite la creación del componente geométrico de la representación



Añadir layers

- Se crean añadiendo el valor de la capa asociado a cada NavCell
- Se puede añadir tantas layer se quieran al NavMap
- Con esta implementación se puede configurar la información

```
// Crear una capa de tipo float llamada "cost"
auto cost = nm.add_layer<float>("cost", "Traversal cost", "", 0.0f);

// Asignar valores a cada NavCell
nm.layer_set<float>("cost", c0, 1.0f);
nm.layer_set<float>("cost", c1, 5.0f);
```

Interactuar con NavMap

- A partir de estos elementos básicos, se puede implementar diferentes tipos de interacción con la representación, siendo los más representativos:
 - ❑ Leer los valores para la capa de una NavCell en dada en específico
 - ❑ Localizar la NavCell de un punto 3D específico
 - ❑ Obtener el valor de una capa con un punto referencia en el mundo
 - ❑ Hacer trazado de rayos que interactúen con las NavCell.
 - ❑ Marcar áreas sobre las capas
 - ❑ Guardar o Cargar la información del NavMap en/desde disco

Más info:

<https://github.com/EasyNavigation/NavMap>

Caso de uso :

Uso de representaciones en exteriores



EasyNav: un sistema de navegación flexible y modular para ROS 2

Francisco Martín Rico, Francisco Miguel Moreno Olivo, Juan Carlos Manzanares Serrano, José Miguel Guerrero Hernández, Juan Sebastián Cely Gutiérrez, Esther Aguado, and Francisco José Romero Ramírez, Juan Diego Peña

