Eurobot Navigation Package Documentation

1. Package Overview

1.1 High-Level Goal

The **eurobot_navigation** package provides the complete navigation stack and mission control system for the Eurobot autonomous robot. It enables the robot to:

- **Map** its environment using SLAM (Simultaneous Localization and Mapping)
- **Localize** itself within a known map using AMCL (Adaptive Monte Carlo Localization)
- **Plan** collision-free paths from current position to goal positions
- **Execute** smooth trajectory tracking with dynamic obstacle avoidance
- **Coordinate** high-level mission logic to collect crates and deliver them to pantries
- **Integrate** perception data to build and maintain a world model
- **Make decisions** autonomously based on task priorities and robot state

1.2 System Architecture

The package consists of five main components working together:

1. **SLAM Node** - Builds maps from LiDAR data during exploration
2. **Localization Node** - Estimates robot pose within known maps
3. **Global Planner** - Computes optimal paths avoiding static obstacles
4. **Local Planner** - Executes trajectory tracking with real-time obstacle avoidance
5. **Task Manager** - High-level mission coordinator that orchestrates the entire system

These components wrap and integrate standard ROS 2 Nav2 tools (slam_toolbox, AMCL, planner_server, controller_server) with custom mission logic designed specifically for the Eurobot competition.

1.3 Hardware and Simulation Environment

- **Platform**: ROS 2 Humble
- **Simulation**: Gazebo (testing phase)
- **Sensors**: 2D LiDAR (for mapping/localization), RGB Camera (via perception package)
- **Actuators**: Differential drive base (receives /cmd_vel commands)
- **External Dependencies**: Nav2 stack, slam_toolbox, AMCL

1.4 Design Philosophy

The navigation package is designed with modularity in mind:

- **Wrapper Nodes**: SLAM and planner nodes are thin wrappers around proven Nav2 components, allowing easy configuration and potential extension
- **Centralized Configuration**: All Nav2 parameters managed through a single YAML file
- **Clear Interfaces**: Communication happens through well-defined ROS 2 topics and messages
- **Autonomous Operation**: Task manager runs fully autonomously once started, requiring no human intervention
- **Coordinate Frame Management**: Handles transformations between camera-relative, robot-relative, and map-global frames

2. Package Summary

Package Name

eurobot_navigation

Goal

Provide complete autonomous navigation capabilities including mapping, localization, path planning, motion control, and mission coordination for the Eurobot competition robot.

Dependencies

ROS 2 Packages:

- rclpy - ROS 2 Python client library
- std_msgs - Standard message types
- nav_msgs - Navigation messages (Path, OccupancyGrid, Odometry)
- geometry_msgs - Pose and velocity messages
- eurobot_interfaces - Custom detection and manipulation messages

External Packages (not in package.xml, launched separately):

- slam_toolbox - SLAM implementation
- nav2_amcl - AMCL localization
- nav2_planner - Global path planning

- nav2_controller - Local trajectory control
- nav2_bt_navigator - Behavior tree navigation

Main Nodes

Node Name	Purpose	Type
slam_node	SLAM Toolbox wrapper for mapping	Wrapper
localization_node	AMCL wrapper for localization	Wrapper
nav2_global_planner	Planner server wrapper	Wrapper
nav2_local_planner	Controller server wrapper	Wrapper
task_manager	Mission coordinator and decision logic	Full Implementation

Configuration Files

nav2_params.yaml - Contains configuration for:

- SLAM Toolbox parameters (mapping/localization mode)
- AMCL parameters (particle filter, update thresholds)
- Planner server parameters (planning frequency)
- Controller server parameters (control frequency)
- Behavior tree navigator parameters

Interaction with Other Packages

Subscribes from eurobot_perception:

- /crate/detections (CrateDetectionArray) - Detected crate positions
- /pantry/detections (PantryDetectionArray) - Detected pantry positions

Publishes to (future):

- /manipulation/command (CrateManipulation) - Pick/place commands for manipulator package

Internal Communication:

- Coordinates between SLAM/AMCL, Nav2 planners, and task manager
- Manages navigation goals and feedback

3. Node-Level Documentation

3.1 Node: slam_node

Node Name

slam_node

Goal / Functionality

Wrapper node for SLAM Toolbox that builds 2D occupancy grid maps from LiDAR scan data. This node serves as a configuration and monitoring interface for the underlying slam_toolbox node.

Subscribed Topics

(Handled by slam_toolbox)

Topic	Type	Purpose
/scan	sensor_msgs/LaserScan	2D LiDAR data for mapping

Published Topics

(Handled by slam_toolbox)

Topic	Type	Purpose
/map	nav_msgs/OccupancyGrid	Generated occupancy grid map
/map_metadata	nav_msgs/MapMetaData	Map dimensions and resolution

Services

(Provided by slam_toolbox)

- /slam_toolbox/save_map - Save current map to file
- /slam_toolbox/serialize_map - Serialize map for later use
- /slam_toolbox/dynamic_map - Get current map

Main Logic Flow

1. **Initialization**: Node starts and logs "SLAM Node started - wrapper for slam_toolbox"
2. **Spin**: Maintains ROS 2 event loop for potential future extensions
3. **Delegation**: All actual SLAM functionality handled by slam_toolbox node

Future Extensions (currently minimal wrapper):

- Monitor mapping quality metrics
- Log loop closure events
- Trigger automatic map saving at intervals
- Publish mapping status messages

3.2 Node: localization_node

Node Name

localization_node

Goal / Functionality

Wrapper node for AMCL (Adaptive Monte Carlo Localization) that estimates the robot's pose within a known map using particle filtering.

Subscribed Topics

(Handled by AMCL)

Topic	Type	Purpose
/scan	sensor_msgs/LaserScan	LiDAR data for localization
/map	nav_msgs/OccupancyGrid	Reference map for localization
/initialpose	geometry_msgs/PoseWithCovarianceStamped	Initial pose estimate

Published Topics

(Handled by AMCL)

Topic	Type	Purpose
/amcl_pose	geometry_msgs/PoseWithCovarianceStamped	Estimated robot pose in map frame
/particle_cloud	geometry_msgs/PoseArray	Particle filter visualization
/tf	tf2_msgs/TFMessage	Transform from map to odom frame

Main Logic Flow

1. **Initialization**: Node starts and logs "Localization Node started - wrapper for AMCL"

2. **Spin**: Maintains event loop
3. **Delegation**: All localization handled by AMCL node

Future Extensions:

- Monitor localization confidence/covariance
- Detect and handle kidnapped robot scenarios
- Log pose uncertainty metrics
- Trigger relocalization if confidence drops

3.3 Node: nav2_global_planner

Node Name

nav2_global_planner

Goal / Functionality

Wrapper for Nav2 planner_server that computes collision-free global paths from the robot's current position to a goal pose using algorithms like A* or Dijkstra.

Subscribed Topics

(Handled by planner_server)

Topic	Type	Purpose
/map	nav_msgs/OccupancyGrid	Costmap for path planning

Published Topics

(Handled by planner_server)

Topic	Type	Purpose
/plan	nav_msgs/Path	Computed global path (sequence of poses)

Action Interfaces

(Handled by planner_server)

- ComputePathToPose - Request path to a goal pose

Main Logic Flow

1. **Initialization**: Logs "Nav2 Global Planner started - wrapper for planner_server"
2. **Delegation**: Planner server handles all path computation

Future Extensions:

- Log path computation time and success rate
- Monitor replanning frequency
- Publish path quality metrics (length, smoothness)
- Implement custom cost functions

3.4 Node: nav2_local_planner

Node Name

nav2_local_planner

Goal / Functionality

Wrapper for Nav2 controller_server that executes trajectory tracking along the global path while avoiding dynamic obstacles and respecting kinematic constraints.

Subscribed Topics

(Handled by controller_server)

Topic	Type	Purpose
/plan	nav_msgs/Path	Global path to follow
/odom	nav_msgs/Odometry	Robot odometry for feedback control
/scan	sensor_msgs/LaserScan	Real-time obstacle detection

Published Topics

(Handled by controller_server)

Topic	Type	Purpose
/cmd_vel	geometry_msgs/Twist	Velocity commands to robot base

Main Logic Flow

1. **Initialization**: Logs "Nav2 Local Planner started - wrapper for controller_server"
2. **Delegation**: Controller server handles trajectory execution

Typical Controller Behavior (DWB or TEB):

- Sample velocity space (linear and angular velocities)
- Simulate trajectories forward in time

- Score trajectories based on: path tracking, obstacle clearance, goal alignment
- Select best trajectory and publish velocity command
- Repeat at control frequency (~20Hz)

Future Extensions:

- Monitor control errors and oscillations
- Log obstacle avoidance events
- Tune controller parameters adaptively
- Implement emergency stop triggers

3.5 Node: task_manager

Node Name

task_manager

Goal / Functionality

High-level mission coordinator that integrates perception data with navigation to autonomously plan and execute the Eurobot competition mission. The task manager:

- Receives object detections from perception nodes
- Transforms detections from robot-relative to map-global coordinates
- Maintains a world model of crates and pantries
- Decides which tasks to execute (pick crates, place in pantries)
- Publishes navigation goals to Nav2 stack
- Monitors task completion and triggers manipulation commands
- Coordinates the complete pick-and-place workflow

Subscribed Topics

Topic	Type	Purpose
/crate/detections	eurobot_interfaces/CrateDetectionArray	Crate positions from perception
/pantry/detections	eurobot_interfaces/PantryDetectionArray	Pantry positions from perception

/amcl_pose	geometry_msgs/PoseWithCovarianceStamped	Robot localization for coordinate transforms

Published Topics

Topic	Type	Purpose
/task_manager/nav_goal	geometry_msgs/PoseStamped	Navigation goals sent to Nav2
/manipulation/command	eurobot_interfaces/CrateManipulation	Pick/place commands (planned, for future manipulator)

Parameters

Parameter	Type	Default	Description
crate_topic	string	/crate/detections	Input topic for crate detections
pantry_topic	string	/pantry/detections	Input topic for pantry detections
amcl_topic	string	/amcl_pose	Input topic for robot localization
max_crates_per_pantry	int	3	Maximum capacity of each pantry
goal_threshold	float	0.25	Distance threshold (meters) to trigger manipulation

Internal State Variables

Robot state

robot_pose: (x, y, theta) # Current position in map frame

Task state

current_task: {

"type": "crate" or "pantry",

"id": crate_id or None,

"x": goal_x,

```
    "y": goal_y
}

# World model
crates: [
    {
        "id": int,
        "x": float,        # map frame
        "y": float,        # map frame
        "color": string,   # "blue" or "yellow"
        "collected": bool
    }
]

pantries: {
    pantry_id: {
        "x": float,        # map frame
        "y": float,        # map frame
        "crates": [crate_ids]  # list of placed crates
    }
}
```

Main Logic Flow

1. Initialization

```
def __init__(self):
```

- Declare and load parameters
- Initialize state variables (robot_pose=None, current_task=None)
- Initialize world model (empty crates list, empty pantries dict)

- Create subscriptions to perception and localization topics
- Create publishers for navigation goals and manipulation commands
- Log startup message

2. AMCL Pose Callback (amcl_cb)

def amcl_cb(msg):

1. Extract pose from PoseWithCovarianceStamped message
2. Convert quaternion orientation to yaw angle (using atan2)
3. Update robot_pose = (x, y, theta)
4. Call check_goal_reached() to monitor task progress

Quaternion to Euler Conversion:

q = pose.orientation

siny_cosp = 2 * (q.w * q.z + q.x * q.y)

cosy_cosp = 1 - 2 * (q.y**2 + q.z**2)

theta = atan2(siny_cosp, cosy_cosp)

3. Crate Detection Callback (crate_cb)

def crate_cb(msg: CrateDetectionArray):

```
IF robot_pose is None:
    RETURN  # Cannot transform without localization


FOR each detection in msg.detections:
    # Transform robot-relative to map frame
    map_x, map_y = robot_to_map(det.x, det.y)


    # Check for duplicate (within 5cm tolerance)
    exists = any crate in crates where:
        abs(crate.x - map_x) < 0.05 AND
        abs(crate.y - map_y) < 0.05
```

```
        IF not exists:
            crate_id = len(crates) + 1
            ADD to crates: {
                id: crate_id,
                x: map_x,
                y: map_y,
                color: det.color,
                collected: False
            }
            LOG: "New crate detected: ID={id}, color={color}, pos=({x},{y})"

    process_task()  # Trigger task planning
```

4. Pantry Detection Callback (pantry_cb)

```
def pantry_cb(msg: PantryDetectionArray):
    IF robot_pose is None:
        RETURN

    FOR each detection in msg.detections:
        pantry_id = det.id

        IF pantry_id NOT in pantries:
            # Transform to map frame
            map_x, map_y = robot_to_map(det.x, det.y)

            ADD to pantries: {
                pantry_id: {
```

```
            x: map_x,
            y: map_y,
            crates: []  # empty list
        }
    }
    LOG: "New pantry detected: ID={id}, pos=({x},{y})"
```

5. Coordinate Transformation (robot_to_map)

```
def robot_to_map(rel_x, rel_y):
    """Transform from robot-relative to map-global coordinates"""
    rx, ry, theta = robot_pose

    # 2D rotation and translation
    map_x = rx + rel_x * cos(theta) - rel_y * sin(theta)
    map_y = ry + rel_x * sin(theta) + rel_y * cos(theta)

    return map_x, map_y
```

Explanation:

- rel_x, rel_y are in robot frame (x forward, y lateral)
- Robot is at (rx, ry) with heading theta in map frame
- Apply rotation matrix and translation to get map coordinates

6. Task Processing (process_task)

```
def process_task():
    IF robot_pose is None:
        RETURN  # Cannot plan without localization

    IF current_task is not None:
        RETURN  # Already executing a task
```

```
    # Priority 1: Pick nearest uncollected crate
    crate = select_nearest_crate()
    IF crate is not None:
        send_goal(crate.x, crate.y)
        current_task = {
            type: "crate",
            id: crate.id,
            x: crate.x,
            y: crate.y
        }
        LOG: "Navigating to crate {id} at ({x},{y})"
        RETURN

    # Priority 2: Drop at nearest available pantry
    pantry = select_nearest_pantry()
    IF pantry is not None:
        send_goal(pantry.x, pantry.y)
        current_task = {
            type: "pantry",
            id: None,
            x: pantry.x,
            y: pantry.y
        }
        LOG: "Navigating to pantry at ({x},{y})"
```

7. Nearest Crate Selection (select_nearest_crate)

```
def select_nearest_crate():
```

```
    # Filter uncollected crates
    uncollected = [c for c in crates if not c["collected"]]

    IF uncollected is empty:
        RETURN None

    robot_pos = (robot_pose.x, robot_pose.y)

    # Find closest by Euclidean distance
    nearest = min(uncollected, key=lambda c:
        hypot(robot_pos[0] - c.x, robot_pos[1] - c.y)
    )

    RETURN nearest
```

8. Nearest Pantry Selection (select_nearest_pantry)

```
def select_nearest_pantry():
    robot_pos = (robot_pose.x, robot_pose.y)

    # Filter pantries with available capacity
    available = [p for p in pantries.values()
                 if len(p["crates"]) < max_crates_per_pantry]

    IF available is empty:
        RETURN None

    # Find closest
    nearest = min(available, key=lambda p:
```

```
        hypot(robot_pos[0] - p.x, robot_pos[1] - p.y)
    )

    RETURN nearest
```

9. Goal Reached Check (check_goal_reached)

```
def check_goal_reached():
    IF current_task is None OR robot_pose is None:
        RETURN

    robot_pos = (robot_pose.x, robot_pose.y)
    target_pos = (current_task.x, current_task.y)
    distance = hypot(robot_pos[0] - target_pos[0],
                     robot_pos[1] - target_pos[1])

    IF distance < goal_threshold:  # Default 0.25 meters
        IF current_task.type == "crate":
            # Arrived at crate - trigger pickup
            trigger_manipulation(current_task.id, command=0)  # 0=PICK

            # Mark crate as collected
            FOR c in crates:
                IF c.id == current_task.id:
                    c.collected = True

            current_task = None  # Clear task
            process_task()       # Plan next task
```

```
    ELSE IF current_task.type == "pantry":
        # Arrived at pantry - trigger placement
        # Find the crate we're carrying
        carried_crate = first crate where collected == True

        IF carried_crate exists:
            trigger_manipulation(carried_crate.id, command=1)  # 1=PLACE

            # Update pantry memory
            nearest_pantry = select_nearest_pantry()
            IF nearest_pantry:
                nearest_pantry.crates.append(carried_crate.id)

        current_task = None
        process_task()
```

10. Manipulation Command (trigger_manipulation)

```
def trigger_manipulation(crate_id, action):
    """Publish pick/place command to manipulator"""
    msg = CrateManipulation()
    msg.crate_id = crate_id
    msg.command = action  # 0=PICK, 1=PLACE

    manip_pub.publish(msg)

    IF action == 0:
        LOG: "Published PICK command for crate {id}"
    ELSE:
```

LOG: "Published PLACE command for crate {id}"

11. Navigation Goal Publishing (send_goal)

```
def send_goal(x, y):
    """Send navigation goal to Nav2 stack"""
    goal = PoseStamped()
    goal.header.frame_id = "map"
    goal.header.stamp = current_time
    goal.pose.position.x = x
    goal.pose.position.y = y
    goal.pose.position.z = 0.0
    goal.pose.orientation.w = 1.0  # No specific orientation


    nav_goal_pub.publish(goal)
```

Decision Logic Summary

Task Priority:

1. **Pick crates** - Always prioritize collecting uncollected crates
2. **Place in pantries** - Only after picking a crate (when carrying one)

Selection Strategy:

- **Greedy nearest-first** - Always select closest target to minimize travel distance
- **Capacity management** - Respect pantry capacity limits (max_crates_per_pantry)

State Transitions:

IDLE → NAVIGATE_TO_CRATE → ARRIVED_AT_CRATE → PICK → NAVIGATE_TO_PANTRY →

ARRIVED_AT_PANTRY → PLACE → IDLE → (repeat)

4. Topic Communication Summary

4.1 Node Communication Table

Node	Publishes	Subscribes	Message Type	Description
slam_node	/map	/scan	nav_msgs/OccupancyGrid	2D occupancy grid map
	/map_metadata	-	nav_msgs/MapMetaData	Map properties
localization_node	/amcl_pose	/scan	geometry_msgs/PoseWithCovarianceStamped	Robot pose estimate
	/particle_cloud	/map	geometry_msgs/PoseArray	Particle filter visualization
	/tf (map→odom)	/initialpose	tf2_msgs/TFMessage	Transform tree
nav2_global_planner	/plan	/map	nav_msgs/Path	Computed global path
nav2_local_planner	/cmd_vel	/plan	geometry_msgs/Twist	Velocity commands
		/odom	-	Odometry feedback
		/scan	-	Obstacle detection
task_manager	/task_manager/nav_goal	/crate/detections	geometry_msgs/PoseStamped	Navigation goals

	/manipulation/command	/pantry/detections	eurobot_interfaces/CrateManipulation	Pick/place commands
		/amcl_pose	-	Robot localization

4.2 Message Definitions

CrateManipulation.msg (Planned)

```
int32 crate_id          # ID of target crate
int32 command           # 0=PICK, 1=PLACE
```

Note: This message is published by task_manager but not yet defined in eurobot_interfaces. It will be implemented when the manipulator package is developed.

4.3 Critical Data Paths

Localization Data Flow:

LiDAR (/scan) → AMCL → /amcl_pose → Task Manager

- Task manager uses pose for coordinate transformations

Navigation Command Flow:

Task Manager → /task_manager/nav_goal → Nav2 → /cmd_vel → Robot Base

- Goals trigger path planning and execution

Perception Integration (from eurobot_perception package):

Camera → Perception Nodes → /crate/detections, /pantry/detections → Task Manager

- Detection messages transformed to map frame