project-division

Sub-Terrain Challenge - Team Division

Project Overview

The project aims to create an autonomous drone system that can:

1. Navigate through a cave environment
2. Find and locate four objects of interest (lights)
3. Generate a 3D representation of the environment
4. Complete the mission as quickly as possible

Team Division of Work

Person 1: Perception Pipeline

Responsibilities:

- Implement depth image to point cloud conversion using depth_image_proc
- Create voxel-grid/occupancy grid representation using Octomap
- Set up semantic camera detection for finding objects of interest (lights)
- Visualize the 3D map

Key packages to create:

- `perception_pkg`: For processing depth images and semantic camera data
- `mapping_pkg`: For building and maintaining the 3D environment model

Key files:

- `depth_to_pointcloud_node.cpp`: Converts depth images to point clouds
- `octomap_builder_node.cpp`: Manages the 3D voxel grid representation
- `object_detector_node.cpp`: Processes semantic camera data for light detection

Person 2: Path Planning

Responsibilities:

- Implement path planning algorithm for navigating through the cave

- Develop trajectory planning from path waypoints
- Create collision avoidance functionality
- Implement exploration strategy for finding lights in the cave

Key packages to create:

- `path_planning_pkg`: For generating safe, collision-free paths
- `exploration_pkg`: For coordinating the search for objects of interest

Key files:

- `path_planner_node.cpp`: Generates collision-free paths through the environment
- `trajectory_generator_node.cpp`: Converts paths to executable trajectories
- `exploration_node.cpp`: Manages the search strategy for finding lights

Person 3: Drone Control & Navigation

Responsibilities:

- Extend the provided controller_node.cpp for better performance
- Implement waypoint navigation system
- Create trajectory tracking functionality
- Manage take-off and landing procedures

Key packages to create:

- `navigation_pkg`: For waypoint following and trajectory execution
- `trajectory_tracking_pkg`: For generating smooth trajectories

Key files:

- `navigation_controller.cpp`: Handles high-level navigation commands
- `trajectory_tracker.cpp`: Follows generated trajectories
- `landing_controller.cpp`: Manages take-off and landing procedures

Person 4: State Machine & System Integration

Responsibilities:

- Develop the main state machine for the robot
- Create the main launch file and system configuration
- Set up ROS message interfaces between components
- Handle reporting and recording of object detections

- Ensure proper system startup and shutdown

Key packages to create:

- `state_machine_pkg`: For coordinating all system behaviors
- `mission_pkg`: For handling high-level mission objectives
- `utils_pkg`: For shared functionality and tools

Key files:

- `state_machine_node.cpp`: Coordinates system behavior
- `mission_coordinator.cpp`: Manages high-level mission objectives
- `main.launch`: Main system launch file
- `object_logger.cpp`: Records detected objects

Technical Interfaces

For these components to work together effectively, they should communicate through:

1. **Transform tree (tf)** - For spatial relationships between coordinate frames
2. **Point cloud messages** - For perception data
3. **Path/trajectory messages** - For planning outputs
4. **State messages** - For system coordination

ROS Topics Structure

- `/current_state_est` - Current drone state estimation
- `/desired_state` - Desired drone state from trajectory planning
- `/rotor_speed_cmds` - Motor commands
- `/depth_image` - Raw depth camera data
- `/semantic_camera` - Semantic camera data
- `/point_cloud` - Processed 3D point cloud
- `/octomap` - 3D voxel grid representation
- `/planned_path` - Generated navigation path
- `/trajectory` - Smooth trajectory for execution
- `/object_detections` - Detected light objects
- `/state_machine/current_state` - Current system state
- `/mission/status` - Mission progress information

Getting Started Guidelines

Each team member should:

1. First review the provided code, especially controller_node.cpp and simulation interfaces
2. Set up their development environment with the required dependencies
3. Create initial ROS package structures for their assigned components
4. Identify the interfaces their component will need to expose
5. Develop incrementally, testing individual components first

Development Timeline

1. **Week 1**: Setup, code review, and initial package structure
2. **Week 2**: Core functionality implementation
3. **Week 3**: Integration and testing
4. **Week 4**: Performance optimization and documentation

Documentation Requirements

- Package structure and node descriptions
- ROS graph indicating who worked on which part
- Figures and plots showing results
- Details on implementation challenges and solutions
- Attribution for any external code used

Submission Components

- Source code (with readme.md)
- Documentation (4-6 pages)
- 8-minute presentation

Communication Plan

- Daily standup meeting to share progress
- Bi-weekly integration meeting to test component interfaces
- Shared Git repository with branching strategy
- Dedicated communication channel for technical questions