CSE 276A - Intro to Robotics - FA24

Homework 5 Report

Andrew Choi (A69033628), Nishanth Chidambaram (A69031827)

[Demo Video] [Github Repo]

- **Final Submission Files**
  - ~/cse276a_ws/src/rb5_ros2/rb5_ros2_vision/launch/yolo_detection_node.py
  - ~/cse276a_ws/src/rb5_ros2/rb5_control/src/hw3_slam.py*
  - ~/cse276a_ws/src/rb5_ros2/rb5_control/src/hw3_ekf.py*
  - ~/cse276a_ws/src/rb5_ros2/rb5_control/src/hw3_move.py*
  - ~/cse276a_ws/src/rb5_ros2/rb5_control/src/hw5_square.py
  - ~/cse276a_ws/src/rb5_ros2/rb5_control/src/mpi_twist_control_node.py
  - ***note) the files are named with 'hw3' but were all modified according to hw5***

Objective

The objective of this assignment is to design and test a Roomba-like system for coverage within a controlled 10ft × 10ft environment, including:

1. Setting up an environment with specified landmarks and virtual boundaries.
2. Implementing a localization system to ensure situational awareness.
3. Designing and executing behaviors for area coverage and virtual boundary avoidance.
4. Demonstrating system performance with trajectory visualization and video evidence.

1. Environment Setup

- **Description of Environment**:
  - Our environment was set up at UCSD Franklin Antonio Hall building 4F. We used a 3m x 3m space with a carpet floor. The main reason for changing the environment from a hard floor to a carpeted floor was that the robot's slide movement wasn't accurate on the hard floor due to less friction, and we figured sliding would be more efficient and precise than rotating 90 degrees.
- **Landmark Placement**: Specify the positioning of two landmarks on each side, including measurements from the walls.
  - Since we adopted the natural landmark detection model for SLAM with YOLOv8, we placed 'bottle', 'potted plant', 'suitcase', 'umbrella', 'teddy bear', 'keyboard', 'stop sign', and 'bowl' and around the square 3m x 3m area.
  - The ground truth coordinates (in meters) of the objects were: Bottle: (2.5, 0.7) / Potted Plant: (0.3, 0.0) / Suitcase: (2.0, 0.0) / Umbrella: (0.5, 3.3) / Teddy Bear: (0.0, 2.5) / Keyboard: (-0.3, 0.0) / Stop Sign: (2.0, 3.0) / Bowl: (2.7, 2.0)

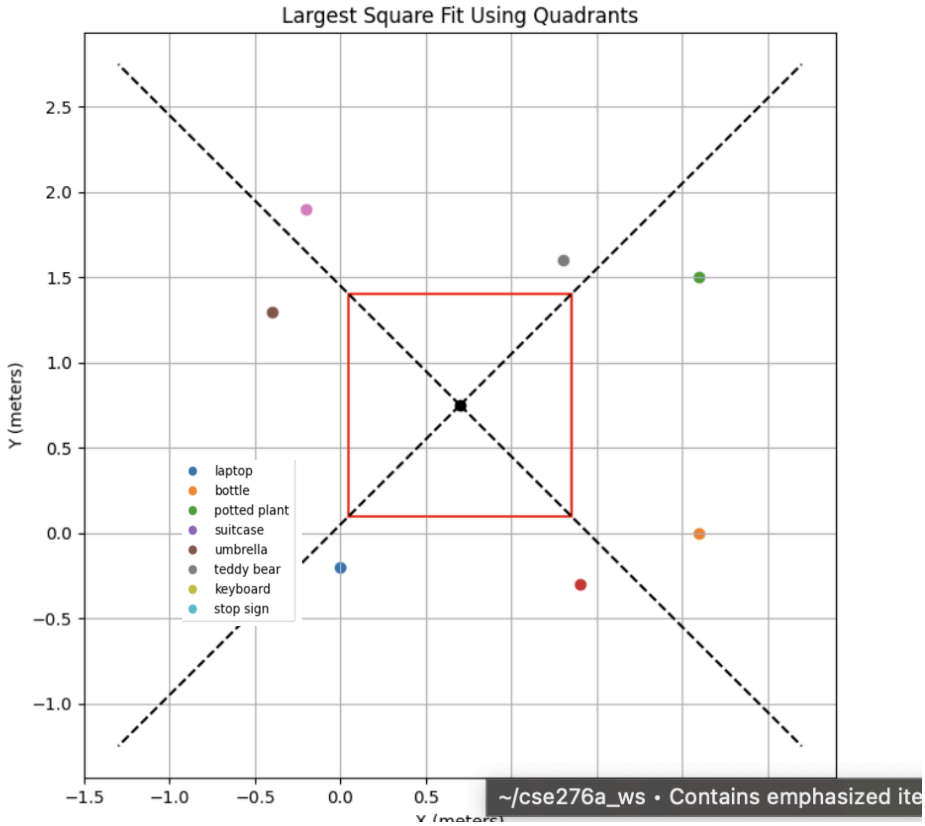
2. Localization System

- Description of how the localization system ensures situational awareness.
    - Each time the robot moves, it detects the natural landmarks relative to the robot. It then uses the robot's position in the world frame to recalculate the landmarks' positions in the world frame. Using these locations, the robot is then able to find its position in the world frame, update the EKF, and improve the accuracy of the landmarks' locations the next time they are seen by the camera.
- Explanation of how virtual boundaries are implemented using localization.
    - Once the robot has gone through the cycles of using SLAM to find the robot's position in the world frame and the natural landmarks' positions in the world frame, the largest square area surrounded by landmarks without enveloping an object is calculated, which then becomes the robot's new virtual boundary. The robot moves to the new virtual area and only moves within that virtual space.

3. Behaviors for Coverage and Avoidance

Explanation of the behaviors designed for:

- Area coverage.
    - We are finding a square area that avoids all obstacles and boundaries, and we are finding the coverage path for that area. We have a known size of the robot, so we are sweeping, traveling forward or backward, from one side of the square to the opposite side of the square. We are also traversing vertically up the area by moving the width of the robot up between each sweep.
- Avoiding virtual boundaries or obstacles.
    - We are using **SLAM** to detect the natural landmarks that mark the boundary and obstacles. Once the objects are detected, we are doing the coverage sweep to traverse across the entire mapped area and stay within the landmarks. We implemented a logic (*hw5_square.py*) where it calculates the largest square that can fit within these landmarks and ensures systematic coverage of the area.
    - The space around the centroid is divided into four **quadrants** based on angles (45°, 135°, 225°, and 315°). From each quadrant, the object closest to the center point is selected as the defining boundary point. The boundaries are adjusted by a margin (0.2m) to account for robot safety and ensure the square stays within the boundaries.



4. Implementation

- Description of the system implementation using ROS or Python.
- Architecture of the system, motivate your design choice(s)
- Brief description of your modules

System Implementation

The program is implemented in Python and ROS 2, combining SLAM, object detection, and trajectory planning. The system uses Extended Kalman Filter (EKF) SLAM for landmark mapping, YOLO-based object detection for real-time feedback, and waypoint navigation for trajectory execution. The architecture leverages modular nodes for flexibility and efficiency in task handling.

1. **Landmark Detection and Mapping (SLAM):**
    - Design Choice: EKF SLAM is selected for its robustness in managing uncertainty in noisy sensor data and dynamic robot movements.
    - Components:
        - hw3_ekf.py: Performs EKF SLAM, estimating distances and object coordinates.
        - hw3_slam.py: Handles robot movements (forward/rotate) and visualizes both the landmarks and robot trajectory.

2. **Object Detection:**
    - Design Choice: YOLO is used for its real-time object detection capabilities, enabling precise identification of landmarks and obstacles.
    - Components:
        - Camera Node: Streams live video feedback to facilitate object detection and coordinate calculations.
        - yolo_detection_node.py: Processes camera data, detects objects and maps their locations in the SLAM environment.
3. **Robot Motion Control:**
    - Design Choice: Modular trajectory execution (square and octagon patterns) is employed for systematic exploration and mapping.
    - Components:
        - hw3_move.py: Executes predefined trajectories for SLAM exploration.
        - mpi_twist_control_node.py: Controls the robot's movement and speed using feedback from mpi_control.py.
4. **Coverage Planning and Path Execution:**
    - Design Choice: The hw5_square.py module calculates the largest navigable area and implements a back-and-forth sweeping path for efficient coverage.
    - Components:
        - hw5_square.py: Determines the largest square navigable within detected landmarks and plans a systematic sweeping path for full area coverage.

Modules Description

1. **EKF(Extended Kalman Filter):**
    - Performs EKF SLAM to estimate the robot's pose and object positions.
    - Tracks landmark coordinates and updates their locations based on sensor feedback.
2. **SLAM:**
    - Provides forward/rotate movements and plots both detected landmarks and the robot's path.
    - Integrates with hw3_ekf.py for real-time mapping.
3. **MOVEMENT:**
    - Defines movements for square and octagon trajectories to facilitate SLAM-based exploration.
    - MPI_TWIST_CONTROL: A sub-module that provides speed control and basic movements.
4. **CAMERA:**
    - Launches the camera for real-time feedback, which is processed by the YOLO detection node.

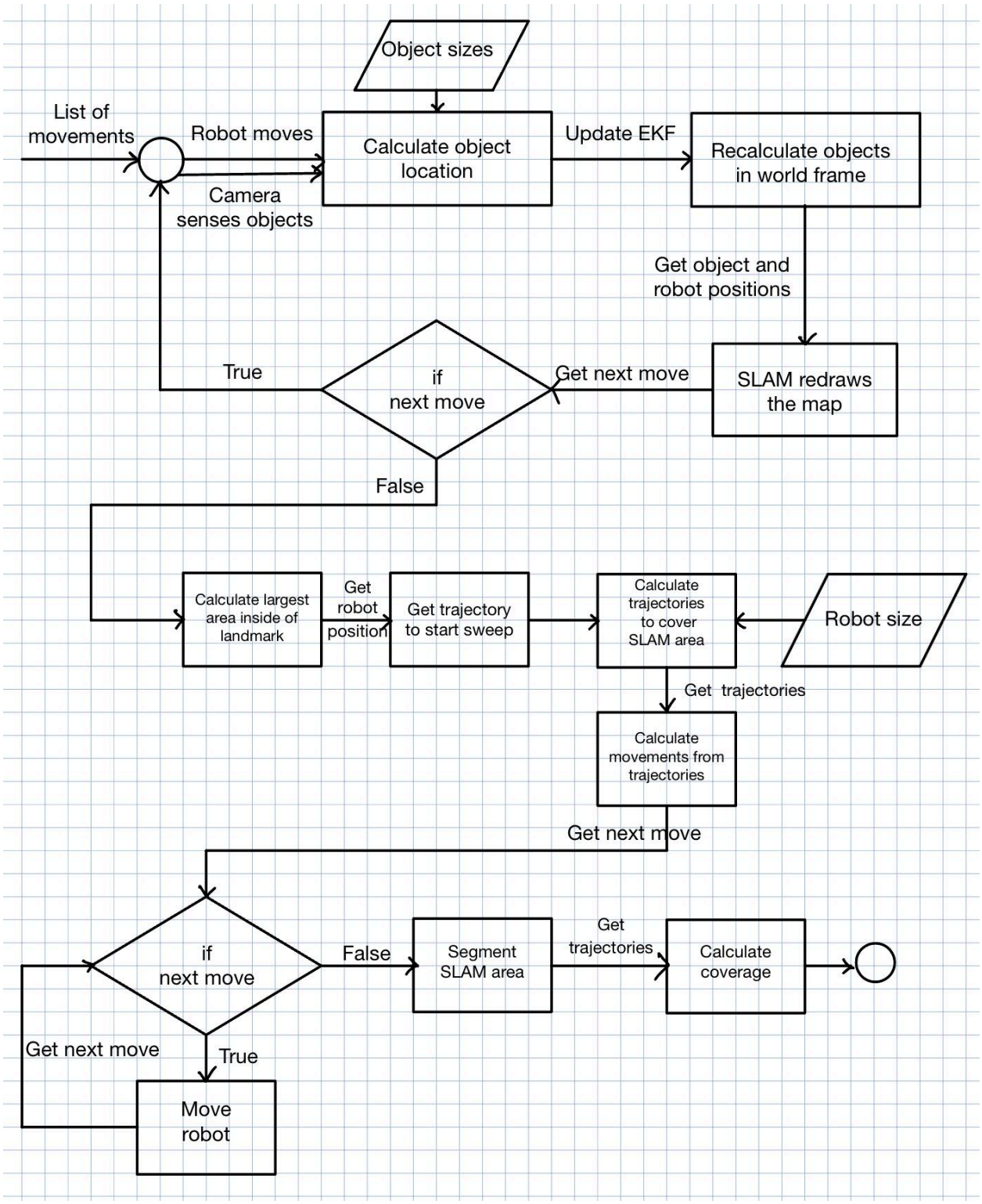
5. **OBJECT_DETECTION:**
    - Uses YOLOv8 to detect objects and their positions relative to the robot.
    - Relays detection results to SLAM for mapping.

Design Motivation

- The modular structure allows independent handling of SLAM, detection, and movement, ensuring scalability and fault isolation.
- Quadrant-based square fitting ensures efficient coverage planning within detected boundaries.
- Real-time feedback from YOLO and SLAM ensures precise and dynamic adjustments during operation.

5. Control Flow Diagram

- Diagram illustrating the control flow of the system.

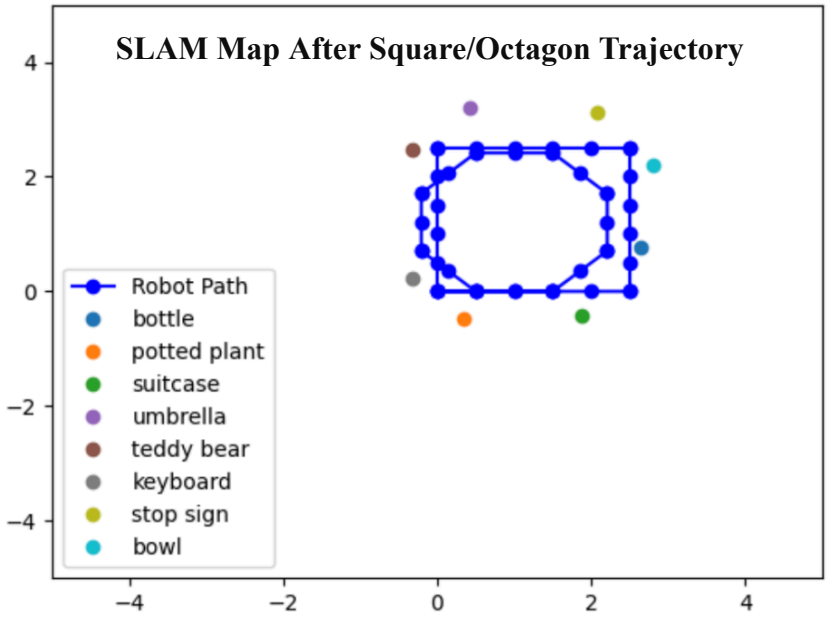


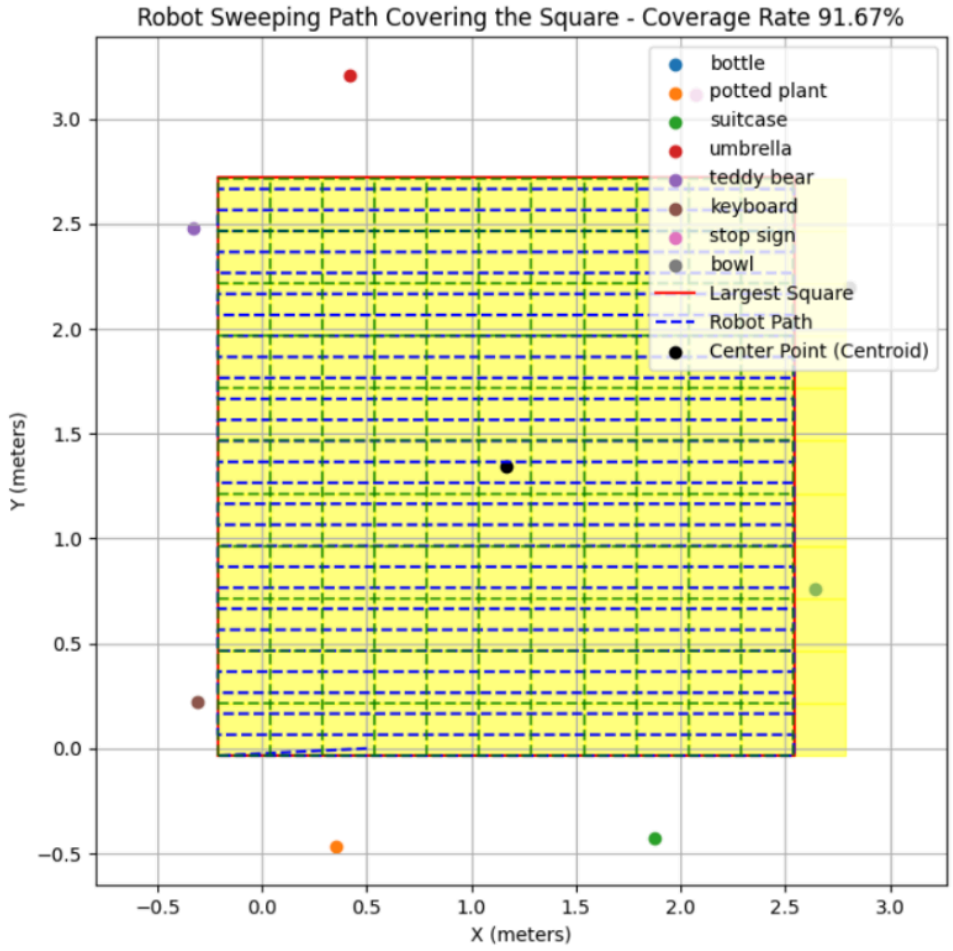
6. Performance Evaluation

Demonstration of system performance:

- Description of the trajectories generated by the system.
    - After the SLAM mapping, the robot knows its position in the world frame, and the landmarks' positions in the world frame, and has calculated the location and size of the space it needs to do the coverage for. It calculates the trajectory to go from the robot's current position to the start position for the sweeping and then calculates the trajectories the robot must take in order to cover all of the space calculated as the largest square within the landmarks by finding the size of the sides of the square, the finding the trajectories to travel along the x-axis along the width of the square, slide upwards for the size of the robot, traverse back along the x-axis, then slide upwards again. This is repeated however many times necessary to cover the entire area, and if there is room for another sweep, the trajectory is calculated to find the last sweep.

- **Graphical illustration of the trajectories.**





7. Extra Credit (Optional)

7.1 Performance Guarantees

If implemented, provide a description or derivation of performance guarantees for coverage.

- We have implemented a performance guarantee system for coverage by taking the calculated square found using SLAM and finding the largest square found inside of the landmarks. Inside the square, the robot finds the path that covers the entire area of the square based on the robot's size and begins to traverse. The trajectories are calculated so that every sweep of the robot (movement along the x-axis from one end to the opposite) will align with the previous sweep. This was done to maximize coverage but minimize movement. We split the square into a number of segments. If the robot's sweeping path travels through a segment, it marks the segment as covered, and at the end posts the final ratio of covered segments against the total number of segments. That ratio is converted to a percentage, which is the percentage guarantee of the total coverage of the mapped area, and added to the title of the chart containing the square and sweeping path.

7.2 SLAM System for Mapping

Description of the custom SLAM system used to map the environment on the fly.

- We incorporated our own solution of the SLAM system using an Extended Kalman Filter (EKF) that was used for Homework 3. So we don't provide the locations of the obstacles or the boundaries, and the robot will navigate through the 10 ft x 10 ft environment in square & octagon paths to detect the surrounding natural landmarks without knowing its starting position. The robot considers its starting point (0,0) in the world frame, and every detected object is compared to the relative position of the robot compared to its starting position, which finds both the robot's position and the landmark's position in the world frame. At the end of the SLAM calculations, the robot has accurately found the position of all objects and itself in the world frame.
- The ground truth coordinates of the objects were:
    - Bottle: (2.5, 0.7)
    - Potted Plant: (0.3, 0.0)
    - Suitcase: (2.0, 0.0)
    - Umbrella: (0.5, 3.3)
    - Teddy Bear: (0.0, 2.5)
    - Keyboard: (-0.3, 0.0)
    - Stop Sign: (2.0, 3.0)
    - Bowl: (2.7, 2.0)
- The calculated coordinates by SLAM were:

```
detected_objects = [
        (2.640951728820801, 0.7626204788684845, 'bottle'),
        (0.3525556892156601, 0.06681589782238007, 'potted plant'),
        (1.8769765138626099, -0.0272804319858551, 'suitcase'),
        (0.4195142149925232, 3.2073102951049805, 'umbrella'),
        (-0.332245796918869, 2.481550693511963, 'teddy bear'),
        (-0.312736182736811, 0.223912658127292, 'keyboard'),
        (2.0687340974807739, 3.1123993396759033, 'stop sign'),
        (2.8089545249938965, 2.1990201473236084, 'bowl')
       ]
```