Robotic - Mapping/ Object Detection

Contents

Overview

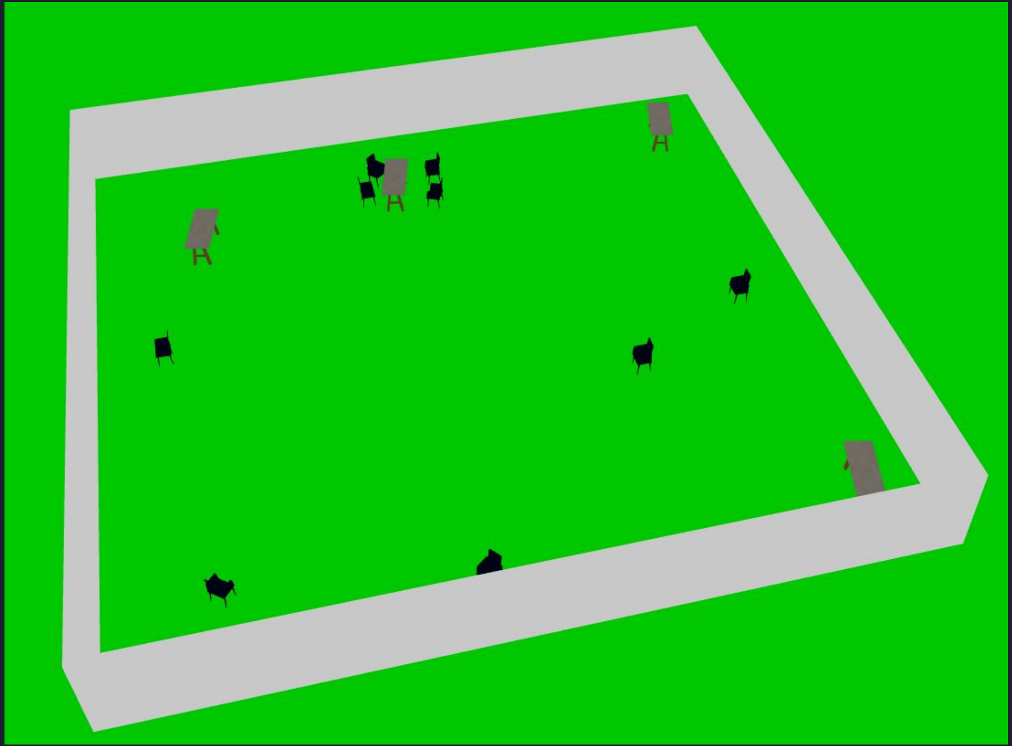
- Robot: MiR 100 (Equipped with Camera and LIDAR)
- Task:
  - Robot explores environment
  - Creates a map
  - Detects Chairs & Tables
  - Marks detected objects in map

Setup

- Single ROS package
- Working environment:
  - Docker container (Docker-ROS)
  - ROS Noetic
  - Catkin workspace
- Startup script:
  - Installing additional dependencies

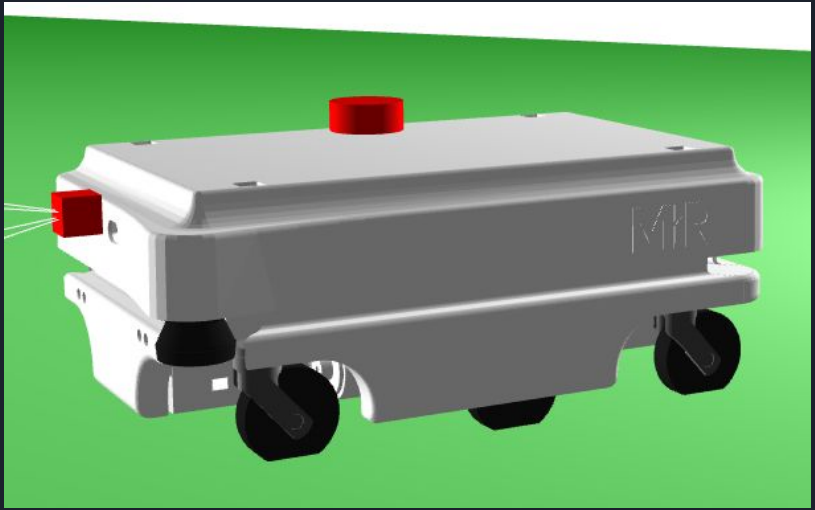
Gazebo World

Gazebo World



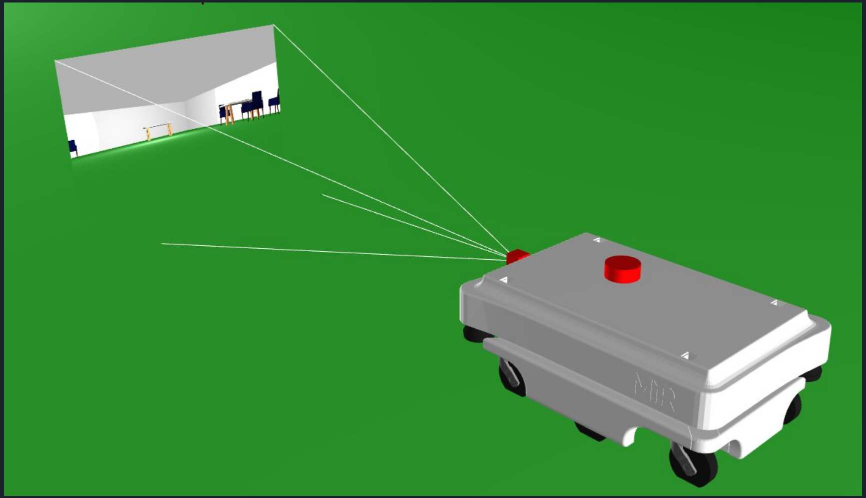
Robot - MiR100

- Package: ros-noetic-mir-robot
- Base: mir_v1
- Added sensors:
  - camera (red box)
  - lidar (red cylinder)



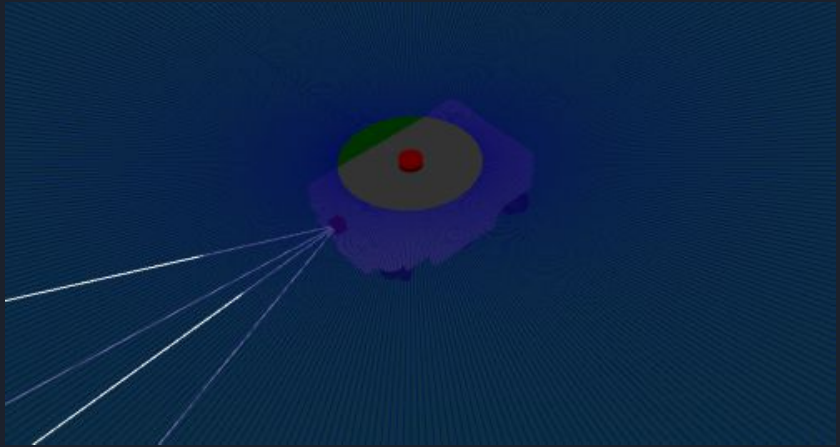
Sensors - Camera

- Connected to base_link of MiR100
- 10 Hz update rate
- Publishes to topic /image_raw
- Field of View: 1.047 rad (~60 degrees)
- Image:
  - Resolution: 640 x 480 Pixel
  - R8G8B8
- Clipping Plane: 10cm - 100m
- Plugin: gazebo_ros_camera



Sensors - Lidar

- Light Detection and Ranging
- Uses light to measure distance to obstacle
- 2D Lidar used
- Connected to base_link of MiR100
- 10 Hz update rate
- Field of View: [-3.14] - [3.14] Rad. (360°)
- Range: 100m
- Plugin: gazebo_ros_gpu_laser



Detection

- What is YOLO Tiny?
  - YOLO (You Only Look Once) is a family of real-time object detection algorithms. **YOLO Tiny** is a lighter and faster variant of YOLO designed to run efficiently on devices with limited computational resources, such as embedded systems or edge devices.
- How YOLO Works
  - YOLO is a deep learning model that frames object detection as a **single regression problem**:
    1. The input image is divided into a grid.
    2. Each grid cell predicts:
       - Bounding boxes for potential objects.
       - Confidence scores for the presence of objects.
       - Class probabilities for each detected object.
  - YOLO achieves high-speed detection by processing the entire image in a single pass through the network
- Why YOLO Tiny?
  - While the full YOLO models (e.g., YOLOv4) are powerful, they are computationally intensive. YOLO Tiny was introduced to:
    1. **Reduce Model Size**:
       - Tiny YOLO has fewer layers compared to the standard YOLO models, making it faster and requiring less memory.
    2. **Increase Inference Speed**:
       - Optimized for real-time object detection on resource-constrained devices like Raspberry Pi, Jetson Nano, or mobile devices.
    3. **Trade-off**:
       - Achieves faster performance at the cost of slightly reduced accuracy.

Detection

- What is OpenCV?
  - **OpenCV (Open Source Computer Vision Library)** is an open-source software library for computer vision, image processing, and machine learning. It provides tools to process images, videos, and real-time data streams efficiently. OpenCV supports multiple programming languages, including Python, C++, Java, and MATLAB, and is widely used in academia and industry for a variety of applications.

Key Features of OpenCV

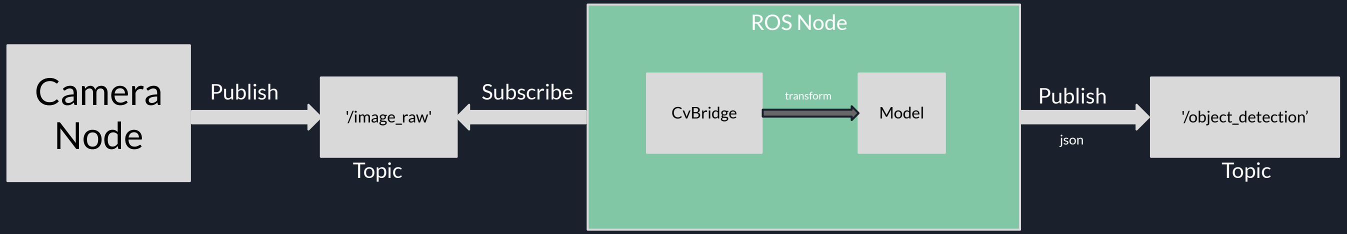
1. **Image Processing**:
   - Reading, writing, and manipulating images and videos.
   - Operations like resizing, cropping, color space conversion, and filtering.
2. **Computer Vision**:
   - Object detection, feature extraction, and recognition.
   - Optical flow, motion tracking, and gesture recognition.
3. **Machine Learning**:
   - Built-in support for common algorithms like k-means clustering and support vector machines (SVM).
4. **Integration with Deep Learning**:
   - Works with frameworks like TensorFlow, PyTorch, and Caffe to deploy pre-trained deep learning models, including YOLO.
5. **Cross-Platform**:
   - Works on various platforms, including Windows, macOS, Linux, iOS, and Android.

Detection

- **Why OpenCV is Ideal for YOLO Integration**
  1. **Built-In DNN Module**:
     - OpenCV has a dnn module that can load and run pre-trained deep learning models, including YOLO (e.g., Tiny YOLO, YOLOv4, etc.).
  2. **Lightweight and Efficient**:
     - Optimized for real-time performance, even on devices with limited computational resources.
  3. **Broad Compatibility**:
     - Works seamlessly with Python, ROS, and other frameworks.
  4. **Easy to Use**:
     - Simplifies image and video processing tasks with intuitive APIs.

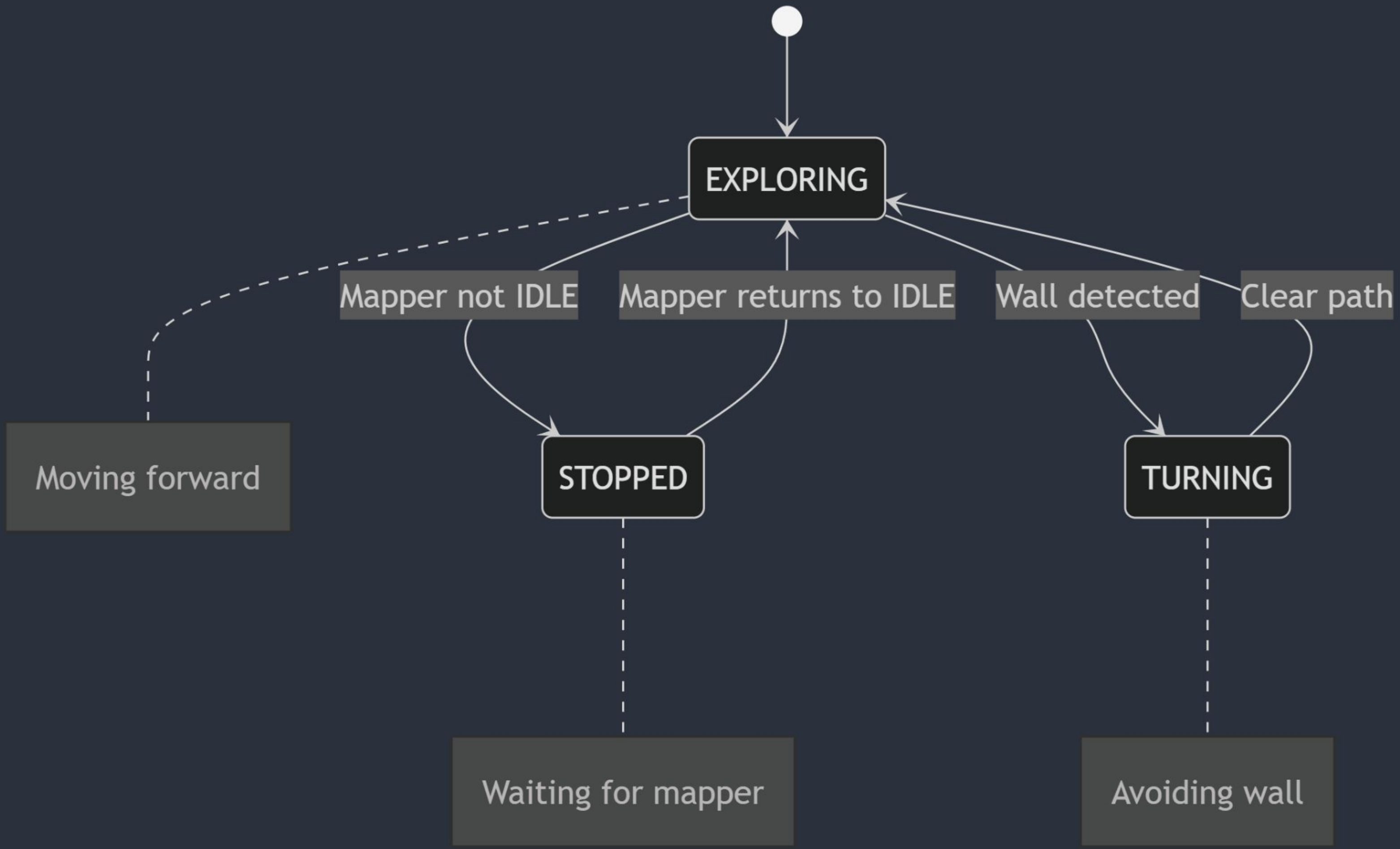
Detection

- For detection a Yolo4 Tiny model has been used
- Pre-trained on COCO dataset: COCO is a large-scale object detection, segmentation, and captioning dataset.



Exploration

- Circular movement with wall avoidance
- Constant linear & angular speed
- Check distances front, left, right sector
    - React accordingly
- Listens to mappers state topic
- Inactive when targeting an object


EXPLORING
Mapper not IDLE
Mapper returns to IDLE
Wall detected
Clear path
Moving forward
STOPPED
TURNING
Waiting for mapper
Avoiding wall

Mapping

- Hector Mapping using LIDAR data
- Map published to /map
- Upon object detection
  - Calculate offset to image center
  - Rotate to center object
  - When centered measure distance using LIDAR
  - Add marker to map
  - Move away to prevent targeting the same object again

Gmapping vs. Hector SLAM

Gmapping

- Particle Filter Based Pose Estimation
- Used odometry data (wheel encoders)
- Prone to drift

Hector SLAM

- Laser Scan matching only
- No odometry required
- Less drift

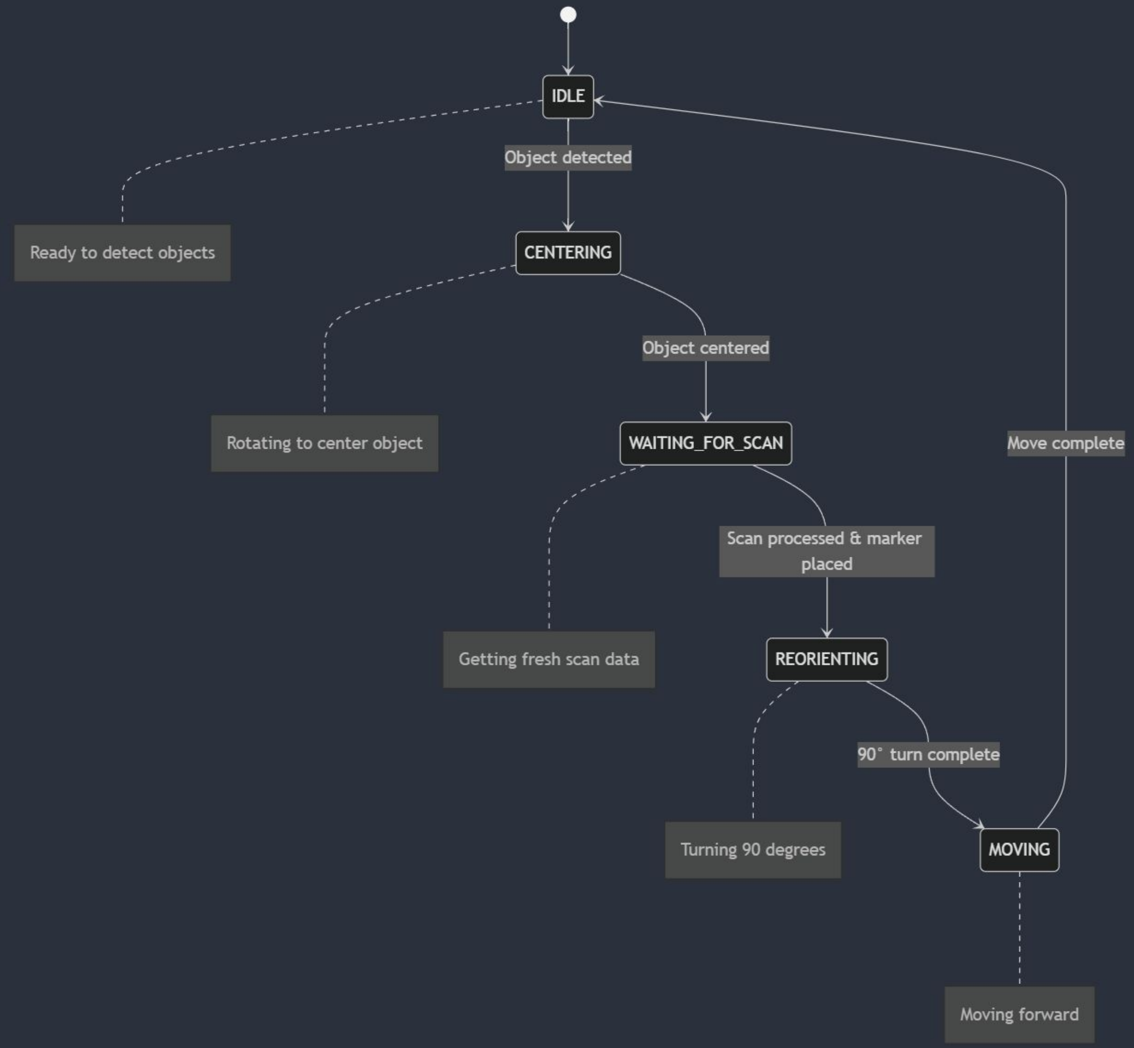
Limitations

- Limited amount of time
- Small Team (only 4 members)
- No prior experience
- 50% of team uses a Mac
- Hardware

Lesson Learned

- Don’t use a Mac (when using ROS) 🐧
- Predefining an API helps a lot when working in teams
- Dealing with robotics-related applications

Thank you for your attention!


IDLE
Object detected
Ready to detect objects
CENTERING
Object centered
Rotating to center object
WAITING_FOR_SCAN
Move complete
Scan processed & marker placed
Getting fresh scan data
REORIENTING
90° turn complete
Turning 90 degrees
MOVING
Moving forward