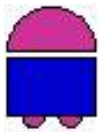
Target Localization and Relative Navigation

Mehmet Yasar Alici

Intelligent Systems Laboratory
Electric Electronic Engineering
Bogazici University

15.02.2021



Project Interest

Create a robot that can,

- localize an object using its RGB-D sensor,
- navigate towards that object.

Project's Components Overview

Which

- world
- robot
- backend
- localization algorithms
- navigation algorithm

that we use?

World

Choice: Gazebo Simulation Environment

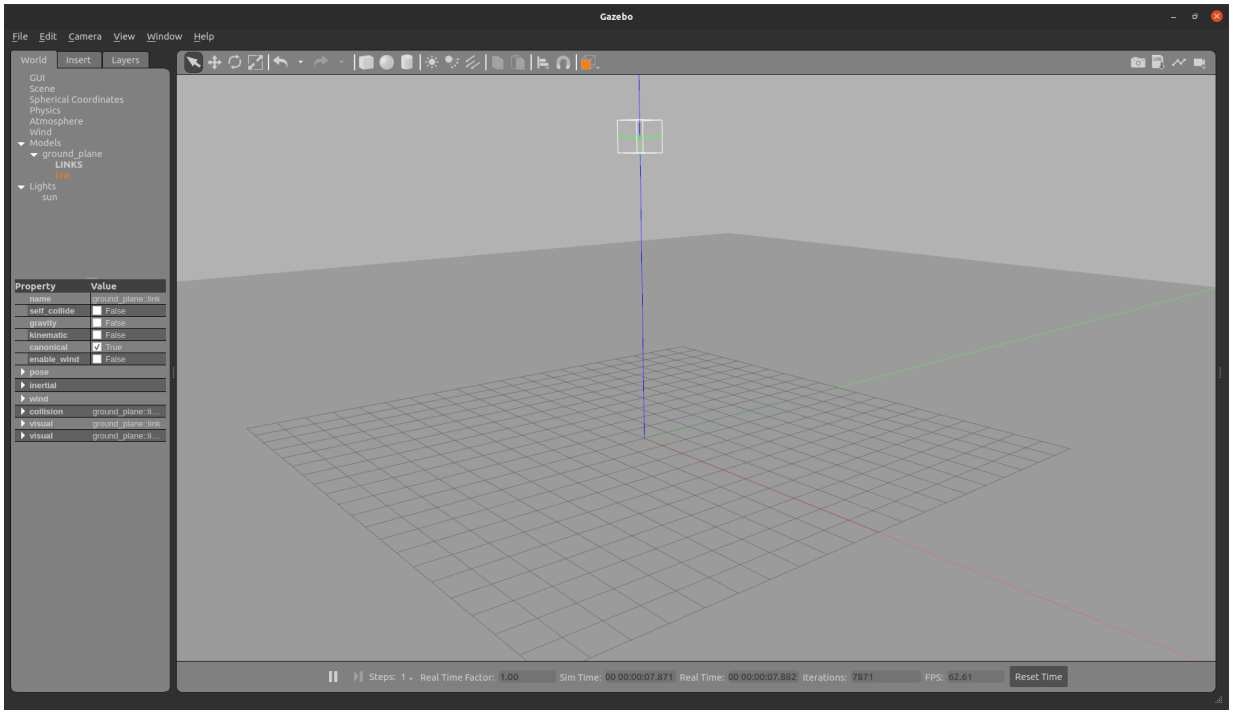


Figure: Gazebo start screen

Robot

Choice: Modified Segway RMP220 by ISL

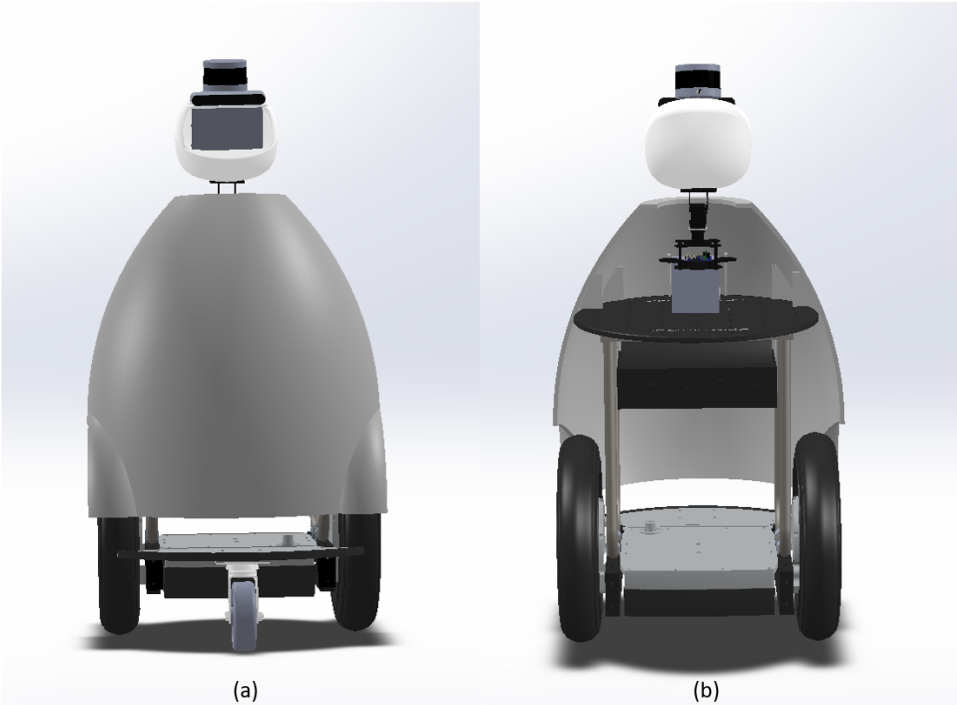


Figure: Solidworks model of our robot: (a) front, (b) back.

Robot: Parts

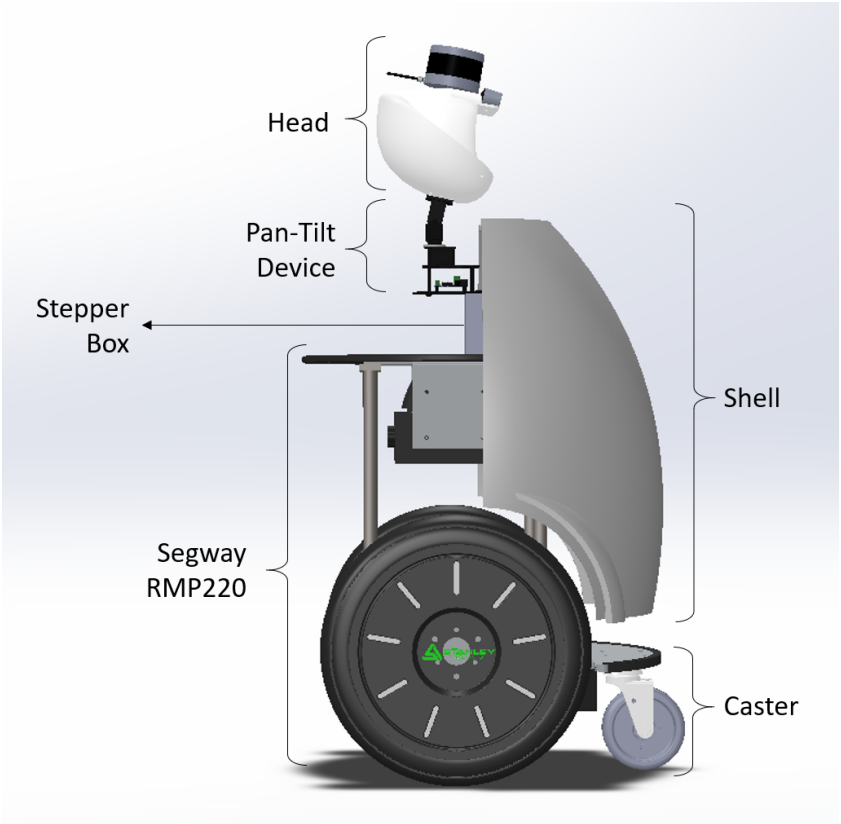
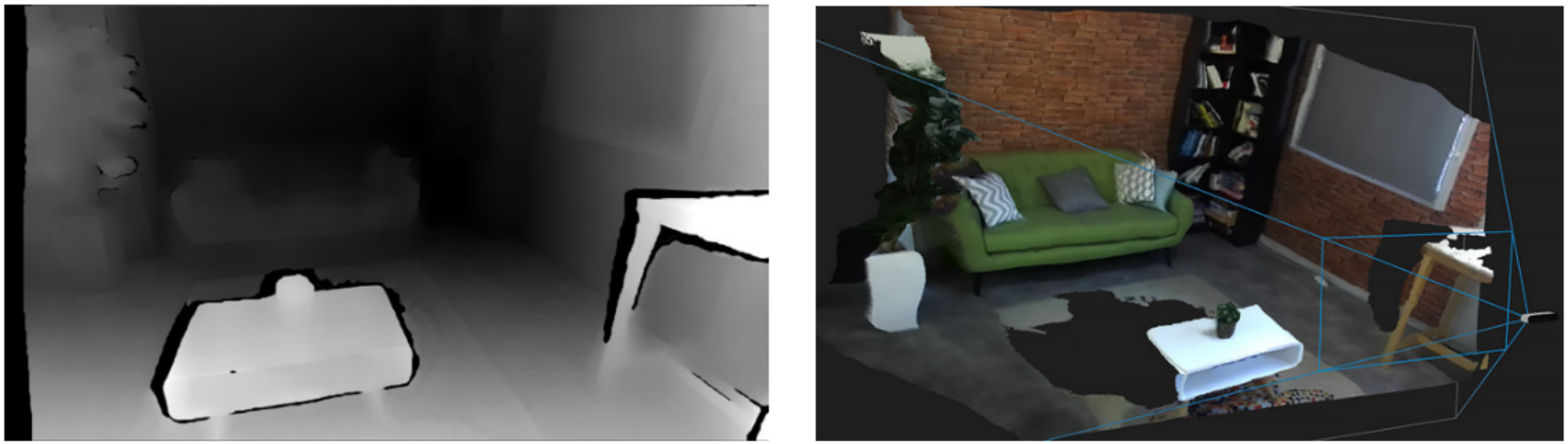


Figure: Robot's side view and its parts.

Robot: RGB-D Sensor



(a) (b)

Figure: A scene captured by RGB-D Camera: (a) Depth Map, (b) Point Cloud [1]

Backend

Choice: ROS Middleware

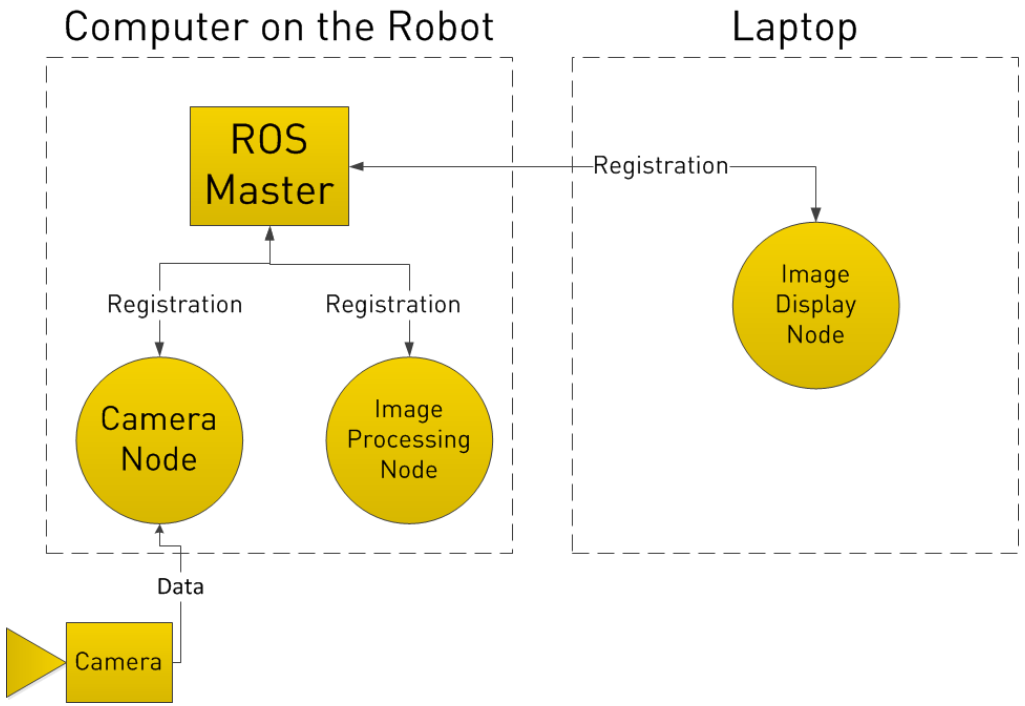
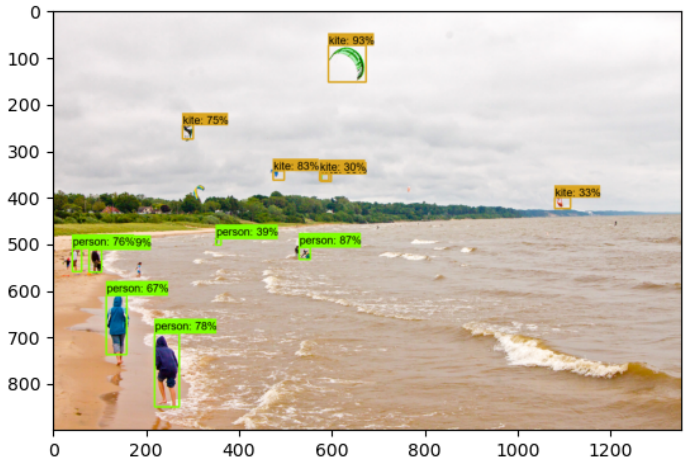


Figure: A typical ROS Usage [2]

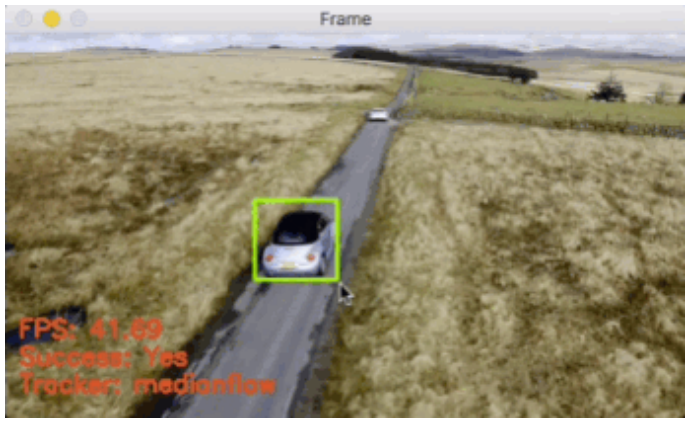
Localization Algorithms

We used two families of algorithms that works on RGB Data:

- **Object Detection**
  Find bounding boxes of objects in an image (Image [3]).



- **Object Tracking**
  Given initial object's coordinates, evaluate the location of the target in the new frame (Image [4]).



Navigation Algorithm

Choice: Modified APF Algorithm by ISL

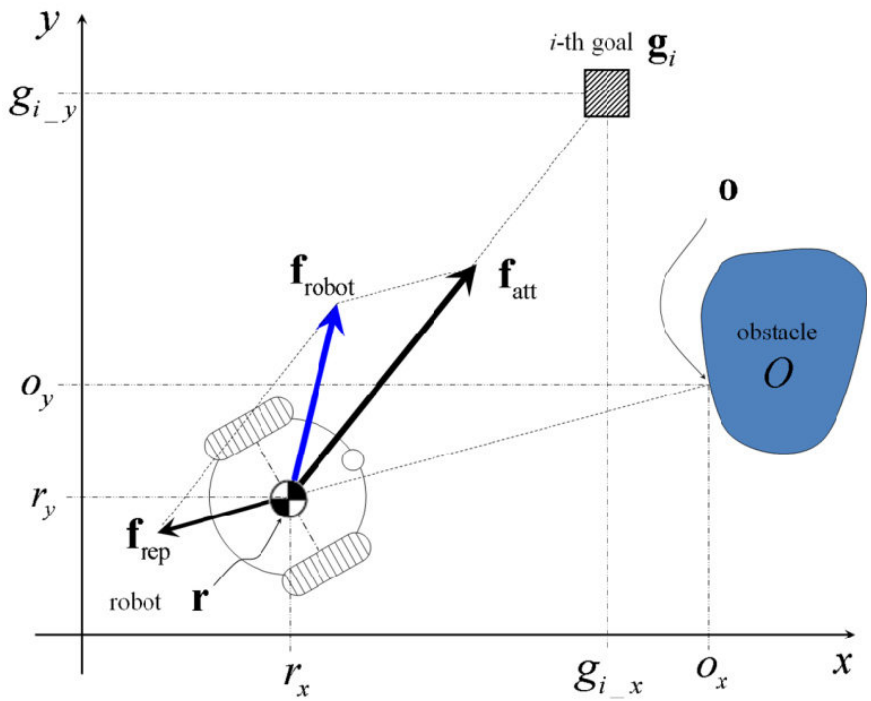


Figure: Obstacle avoidance using the potential field method [5].

Overview

We have the following components individually:

1. Gazebo
2. Modified Segway RMP220
3. ROS
4. Object detection and tracking
5. Modified APF

Now, we will explain how we integrated those components under the following sections:

- **Environment Bring-up** (1-2-3)
- **Target Localization** (4)
- **Relative Navigation** (5)

Environment Bring-up: Overview

1. **Satisfy prerequisites:**
   Install Ubuntu, ROS and Gazebo.
2. **Obtain the robot on Gazebo:**

First bring-up individually,

- Segway RMP220
- Pan-Tilt Device
- RGB-D and Lidar Sensors
- Other parts

and then assembly.

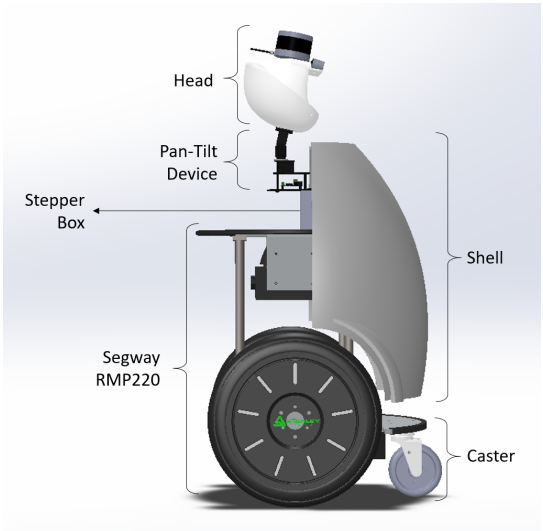
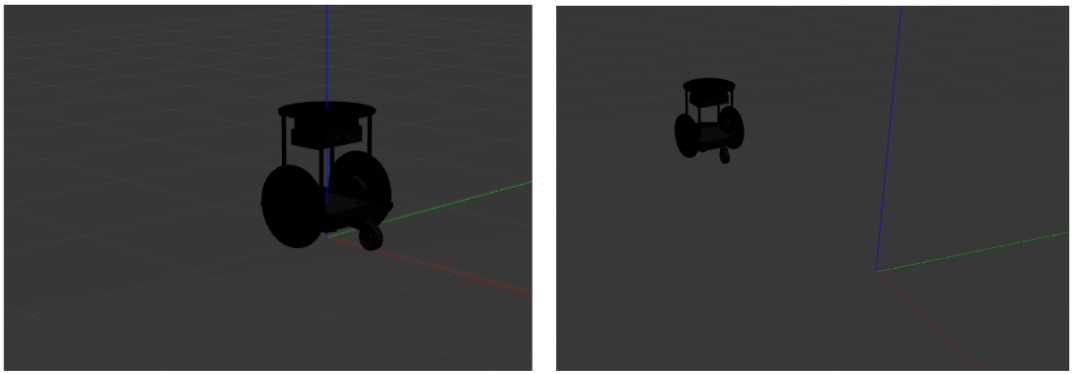


Figure: Robot's reference design on Solidworks

Segway RMP220

- Used manufacturer's packages on Github.
- Fixed version mismatches and documented as appendix to final paper.

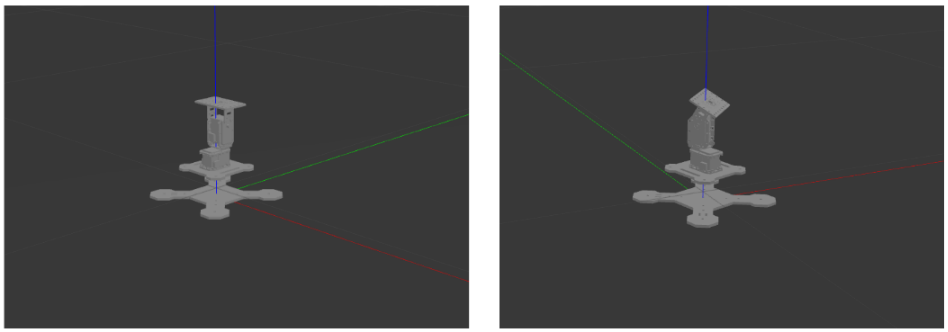


(a) (b)

Figure: Segway RMP220 bring-up on Gazebo: (a) Spawn, (b) Navigation

Pan-Tilt Device

- Used manufacturer's visual model on Github.
- Developed PID Controllers with $K_p = 20$, $K_d = 0.1$ and $K_i = 1$ for pan and tilt joints.
- Bring-up was documented and available as appendix to final paper.

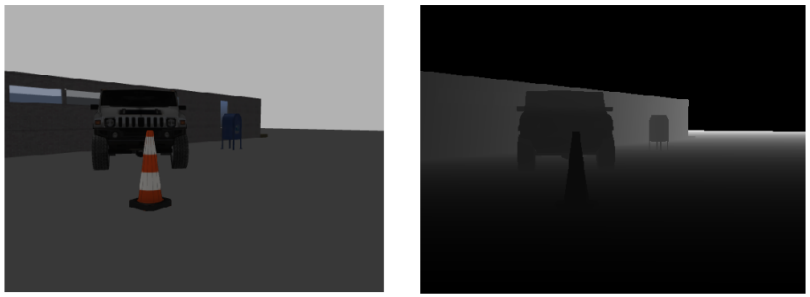


(a) (b)

Figure: Pan-Tilt bring-up on Gazebo: (a) Spawn, (b) Control

RGB-D and Lidar Sensors

- Used manufacturer's visual models on Github.
- Added sensor functionality to RGB-D Visual.



(a) (b)

Figure: RGB-D bring-up on Gazebo: (a) Spawn, (b) Control

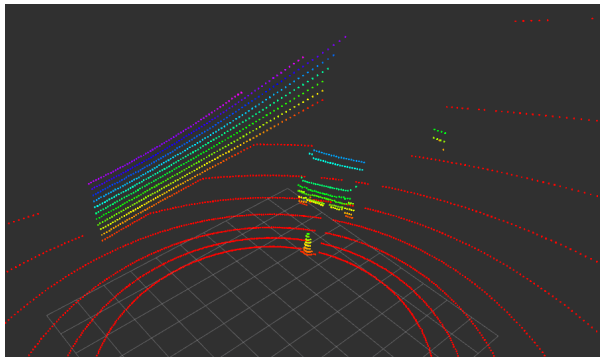
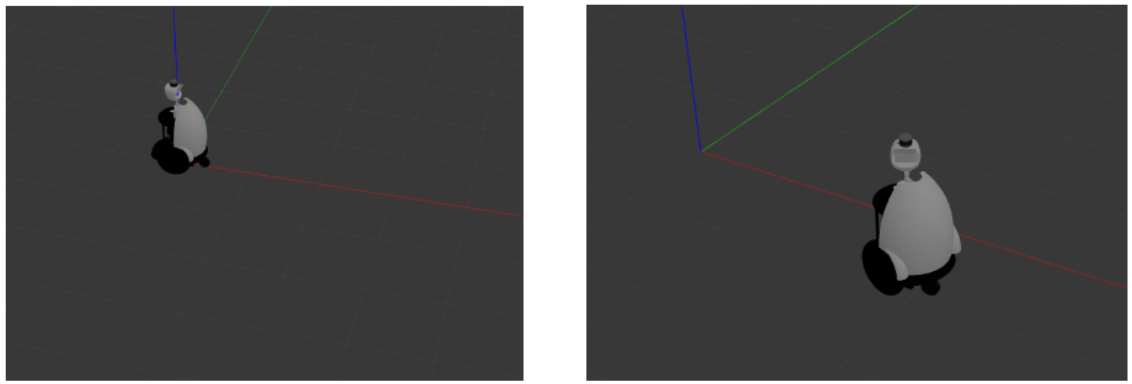


Figure: Lidar bring-up on Gazebo

Robot Integration

- We brought up remaining components by exporting them from Solidworks.
- Assembled all components into a robot.



(a) (b)

Figure: Final robot in Gazebo after Robot integration: (a) Spawn, (b) Control

Target Localization: Overview

- Fast localization is important.
- **Aim**: Develop a ROS Node that detects and tracks a target object with the following usage:

```
python detect_track.py person
```

Detection

- We used Tensorflow Object Detection API as backend.
- We selected two candidate models from the API:

Model Name	Speed (ms)	COCO mAP
SSD MobileNet v2 320x320	19	20.2
CenterNet Resnet50 V1 FPN 512x512	27	31.1

Table: Candidate model characteristics described in the API

Detection: Application

We added detection as well as real-time visualization feature to our node.

(a) (b)

Figure: Real-time detection results for the faster model.

Detection: Evaluation

We evaluated the characteristics of two models:

Model Name	Model Speed (s)		Confidence %
	Loading	Inference	
SSD MobileNet v2 320x320	19	0.89	64
CenterNet Resnet50 V1 FPN 512x512	28.94	2.38	95

Table: Actual model characteristics for human detection

Tracking

- Used OpenCV's Tracking API as backend.
- Implemented tracking in our node and obtained an average of 0.27 seconds as tracking time.

Figure: Tracking of a target

Tracking with Head

- **Aim**: Always center the target to the RGB image, therefore never lose sight of it.
- Map the target location in pixels in the RGB image to radians in the viewing angle.
- For pan controller, a reference input $r(t)$ is calculated as,

$$r(t) = f(x_T) = \frac{x_T}{W}\Theta - \frac{\Theta}{2} \tag{1}$$

where,
W is the maximum RGB width,
Θ is the maximum horizontal viewing angle.

Tracking with Head: FIX

- It is very important to wait for the current command to finish until a new command is given to pan-tilt.
- For pan controller, we define r(t) as,

$$r(t) = \begin{cases} r(t) + f(x_T) & |r(t) - y(t)| < M_{ss} \\ r(t) & \text{o.w} \end{cases} \tag{2}$$

where,
M_{ss} is a threshold limit to consider that the pan controller is not moving anymore,
$y(t)$ is the output position of the pan-tilt,
$f(x_T)$ is the map of target location from pixels to angles.

Tracking with Head

We added the feature to our node and evaluated with a test setup:

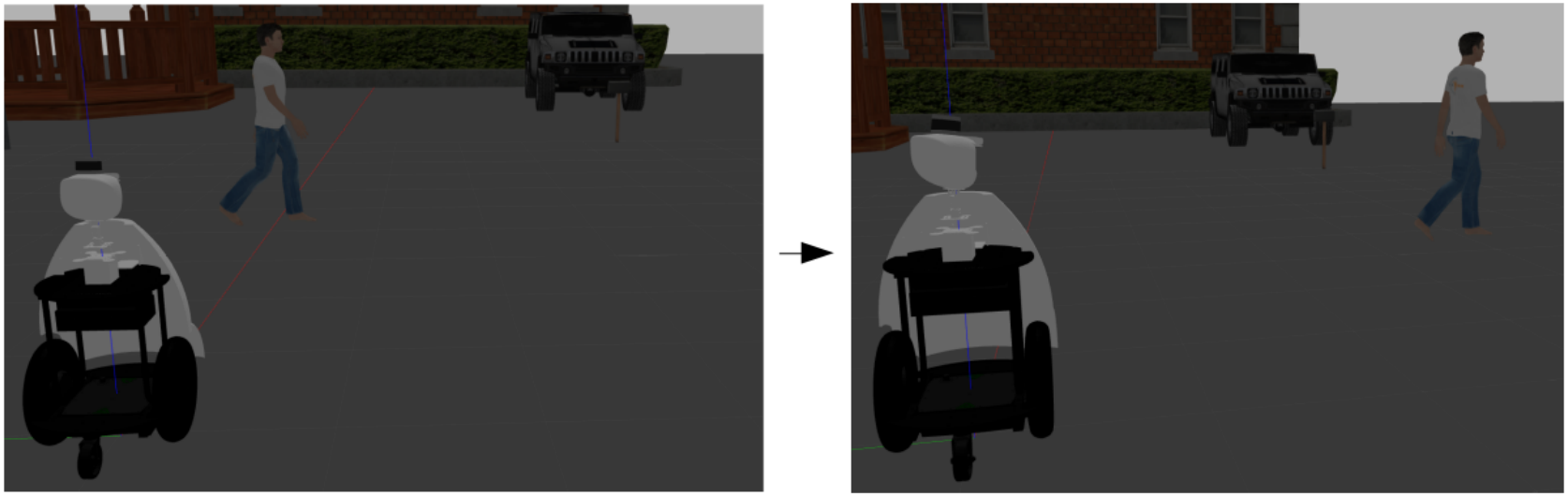


Figure: Tracking with head as target moves to the right.

Detection-Tracking State Machine

- Implemented a state-machine in the node that acts as a main loop.
- Apply detection once after every $i = 10$ tracking operation:

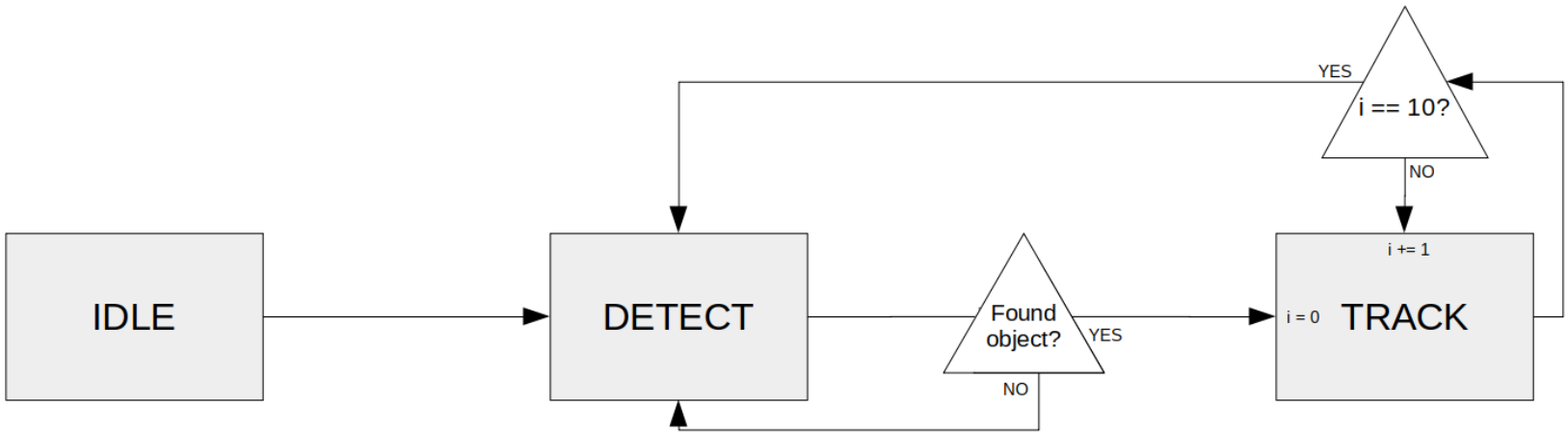


Figure: Detection-Tracking main logic

Relative Navigation

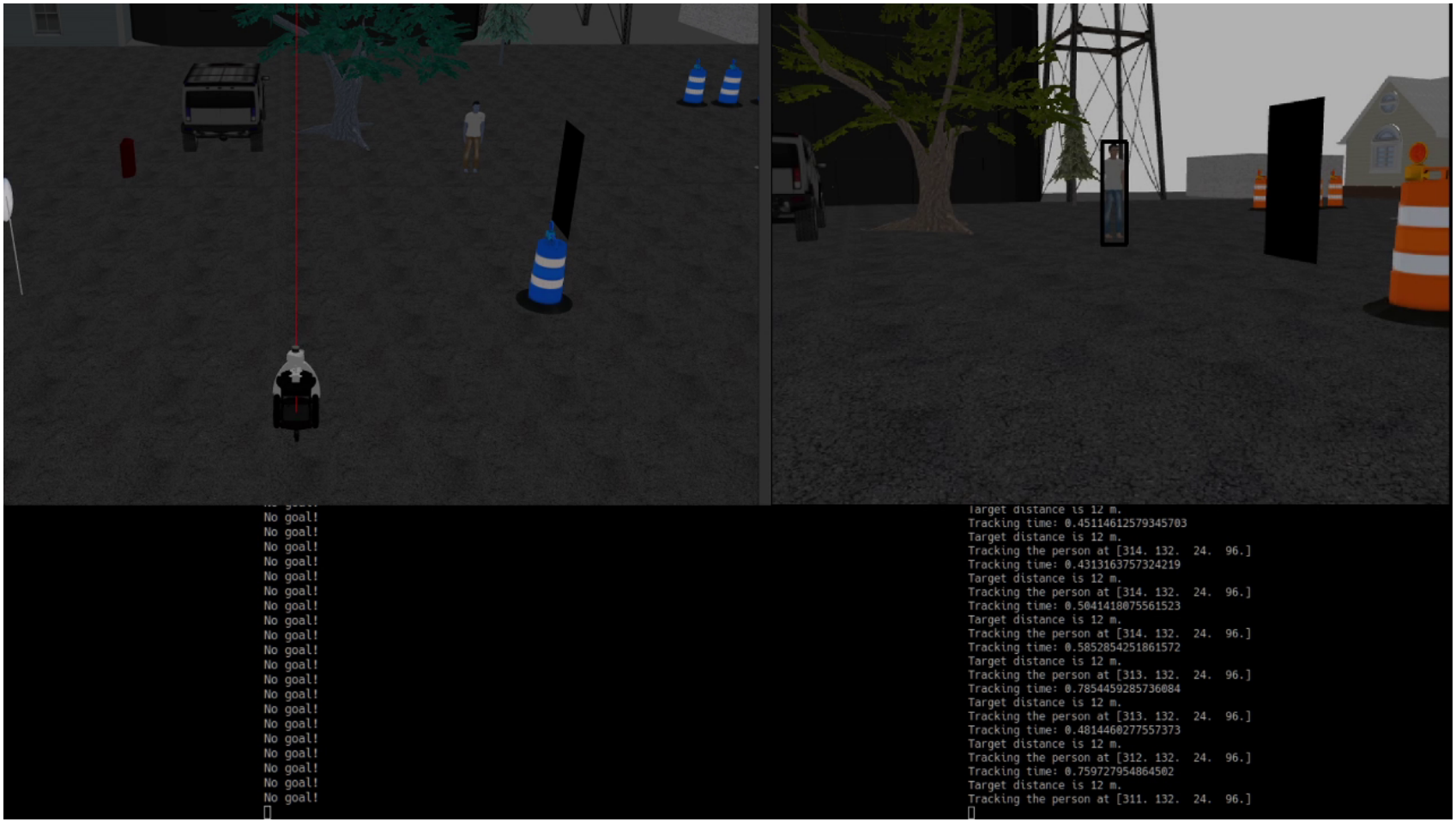
- **Aim**: Task the robot to navigate towards the object.
- Employed the APF algorithm which requires to specify,
  - distance x to target,
  - angle θ between the robot and target.
- **How can we calculate x and θ?**

Relative Navigation

- We already know θ is the pan angle thanks to tracking with head.
- We calculate x by calculating the histogram of the bounding box zone in the depth map.

(a) (b)

Experiments



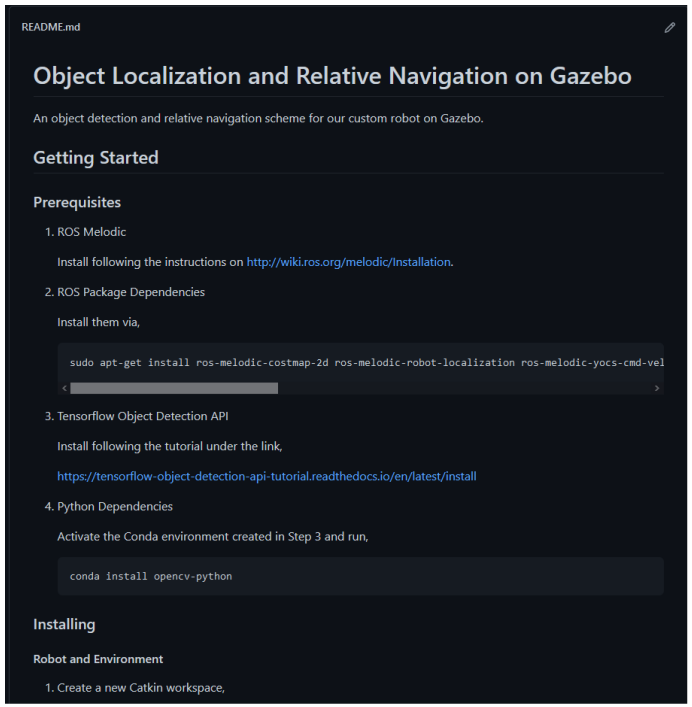
Conclusion

In this project, we addressed,

- bringing up our robot in Gazebo simulation environment,
- developing a fast detection and tracking scheme,
- tasking the robot to navigate towards the localized target.

Conclusion

- Additionally, we documented four extensive installation and usage guides publicly on Github.


README.md

Object Localization and Relative Navigation on Gazebo

An object detection and relative navigation scheme for our custom robot on Gazebo.

Getting Started

Prerequisites

1. ROS Melodic

   Install following the instructions on http://wiki.ros.org/melodic/Installation.

2. ROS Package Dependencies

   Install them via,

```
sudo apt-get install ros-melodic-costmap-2d ros-melodic-robot-localization ros-melodic-yocs-cmd-vel
```

3. Tensorflow Object Detection API

   Install following the tutorial under the link,

   https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install

4. Python Dependencies

   Activate the Conda environment created in Step 3 and run,

```
conda install opencv-python
```

Installing

Robot and Environment

1. Create a new Catkin workspace,

Figure: Main Github Readme of the Project

Future Work

A promising area of future work would be,

- Lidar incorporation to object localization,
- point cloud analysis and scene segmentation.

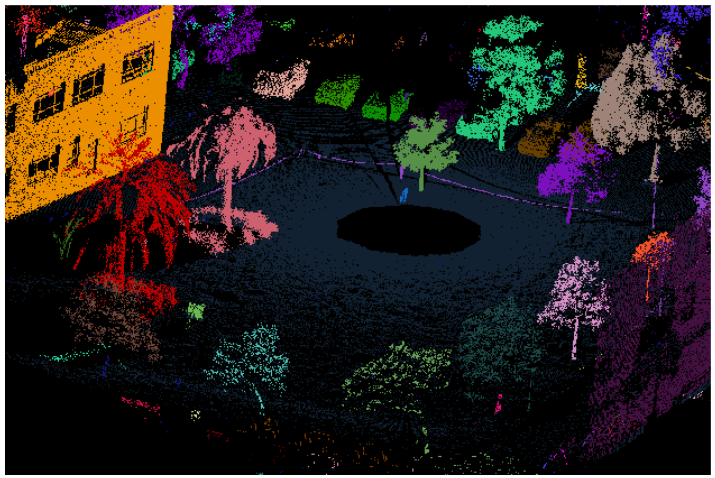


Figure: Semantic segmentation of 3D Point Clouds

Thank you for listening!

Intelligent

Systems Laboratory

Bogaziçi University
Electrical & Electronics Engineering Dep.

 URL: https://www.stereolabs.com/docs/depth-sensing/.

 URL: http://www.clearpathrobotics.com/assets/guides/kinetic/ros/Intro%20to%20the%20Robot%20Operating%20System.html.

 URL: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/.

 URL: https://learnopencv.com/object-tracking-using-opencv-cpp-python/.

 URL: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-17592008000400003.