PROJECT 3B: PURSUIT EVASION GAME (PART-2)

Tanmay Khandait
Student ID : 1219385830
CIDSE, Arizona State University
tkhandai@asu.edu

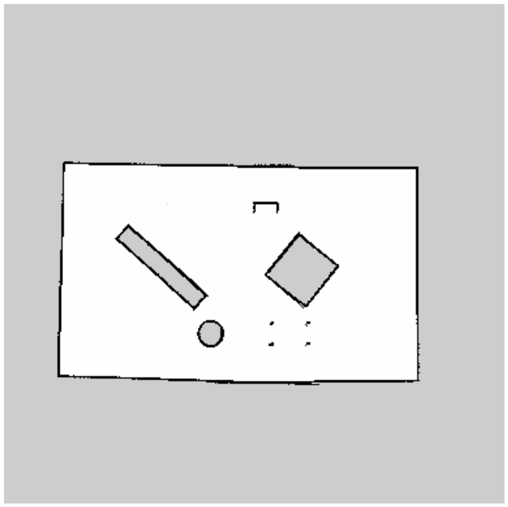
28 April, 2021

ABSTRACT

This report being submitted for Project-3b along with the necessary resources. In this project, the idea is develop code in so that the pursuer brobot can catch the evader bot. In order to do this, we start by creating maps of the environment by performing SLAM. Once we have these maps, we move towards our main project, the pursuer-evader model.

1 SLAM with GMapping in ROS (Task 1)

I used the SLAM tutorial from the robotis documentation [1]. I launched the SLAM node from the resources provided (robot_mapping.launch). Once the SLAM node was launched, the turtlebot was teleoperated to move around and perform the SLAM to create the map. The map that was obtained is show in figure 1. Once the map is created, we have to save the map. All the commands for this task are provided in the resources folder provided and the robotis documentation [1].The video of the map creation process is shown in the videos.


(a) World 0 map

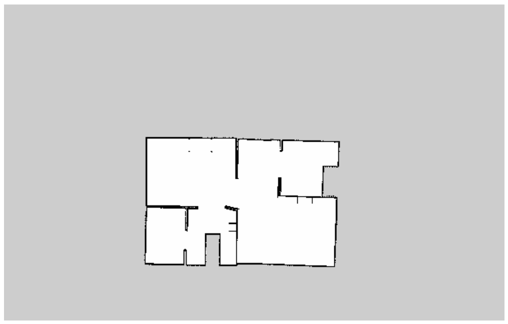

(b) World 1 map

Figure 1: Map generated using SLAM from [1].

2 Autonomous Navigation (Task 2)

The navigation stack works by taking in readings from the odometry (pose) and sensor readings and outputs a velocity that the bot must move with. In this project, we provide the navigation stack takes in the odometry readings, sensor streams and the global cost map from SLAM and determines a velocity with which the robot must be moved.

The idea is that we want to keep track of the obstacles in the environment and then move the robot in order to avoid the obstacles. The navigation stack uses both the global map and the local map. The intuition behind the global and the local map is that we want the global map to keep track of long-term plans and the local cost map to create local plans and avoid obstacles. In order to work around these, we want some criterion that must followed on the local map only, on the global map only and on both. These are formally referred to as common configuration, global configuration, and local configuration options.

In the common configuration options, we configure the options pertaining to the obstacles in the world, how the obstacles will be treated, radius of detection of detection, shape/size of robot and the sensor streams that will be available. The global configuration defines the reference coordinate frame the map should be loaded in. The local configuration option takes care of how the cost-map will look like, its resolution, width, height etc. The local planner uses these items in order to create plans to move the goal.

3 Tracking Node (Task 3)

The tracking node was created by using YOLO object detection system [2]. Specifically, I use the yolo-tiny architecture. In order to make it compatible with the python2.7 (for integration with ROS), we make use of darkflow library [3]. The idea is to detect the objects in the image and determine the "person" label objects in the image. The confidence scores of the detection and their bounding boxes are locally stored. In most of the cases, we only have one detection for the "person" labels. In case there are multiple detections, we sort the confidence scores and use the detection with the highest confidence scores. Some of the detections are shown in figure 2

The cv_bridge() is used to convert the image messages (images are captured and transmitted as messages) to images and vice-versa.

4 Control Node for Pursuit (Task 4)

We finally reach the task of pursuer and evader. Once the bounding boxes were detected by the pursuer, the centre pixel of the pursuer was calculated. Since we know that the FOV is $60\,\text{deg}$, I scaled the pixels (640) to lie in between 0-60. This helps is determining the angle which should move in. In order to detect the depth, the initial idea was to use depth maps. The problem with the depth map was that once the bounding boxes were detected, recognizing the depth of the human was difficult to estimate. This was because the space between the legs gave different depths and it was hard to localize the depth of images. To deal with the issue of depth, I took a 1 metre distance in the direction instead of determining the depth of the legs of the human.

In order to create the node, I used many methods like using the autonomous navigation (MoveBaseAction), but eventually failed. At the end, I stuck to code form the first project's rotation and moving forward code. The idea was to rotate in the direction by the angle which is determine the scaling (as explained above) and then moving forward. This is put in the call back function and run whenever the person is detected.

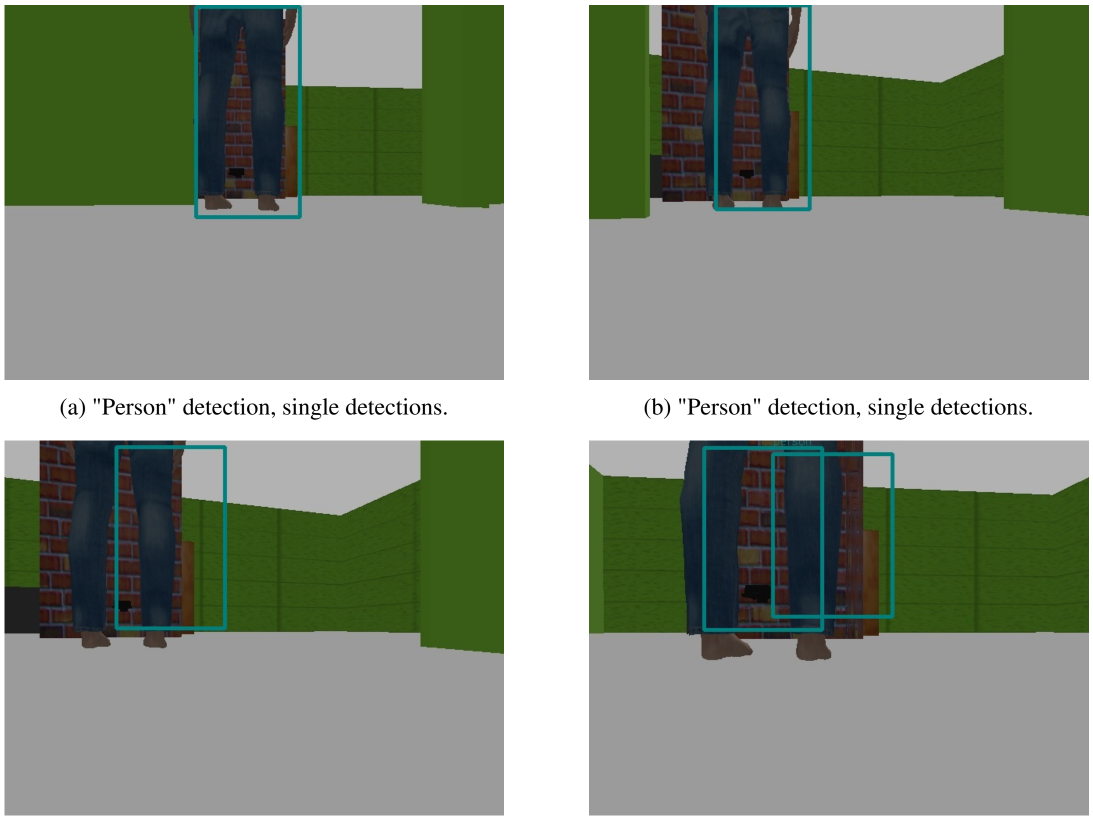
Limitations: The limitation with this approach is that it is not a very robust method. It is highly dependent on initial capturing of the human. It also is very susceptible to random movements caused when the evader loses track. Ideally, if we assume no latency between image capturing and detection, we this idea should work sensibly. However, a fair amount of lag is introduced due to the compute resources of my laptop. This causes the robot susceptible to random movements.

5 How to run the code?

This code was run using Ubuntu 18.04 and Ros-Melodic. Standard packages are needed, in the sense that we need the ROS packages and packages for turtlebot3 installed already. Also, we need the tensorflow==1.15 package for running the YOLO model. In order to run the code for map **X**, follow the following commands. **X** can be 0 or 1.

- Open the terminal window and paste the following

```
roscore
```



(a) "Person" detection, single detections.

(b) "Person" detection, single detections.

(c) "Person" detection, single detection.

(d) "Person" detection, two detections.

Figure 2: Person tracking node results on still frames. The confidence threshold used was 0.05.

- Open a new terminal window and paste the following

```
source ~/proj3b_ws/devel/setup.bash
export TURTLEBOT3_MODEL=waffle_pi
roslaunch pursuit_evasion robot_amcl.launch
    world_file:=~/proj3b_ws/src/pursuit_evasion/
        maps/map[X].yaml world_index:=[X]
```

  This will launch the rviz with the map **X** with both the pursuer and evader spawned in.

- Open a new terminal window and paste the following

```
source ~/proj3b_ws/devel/setup.bash
export TURTLEBOT3_MODEL=waffle_pi
chmod +x ~/proj3b_ws/sec/pursuit_evasion/darkflow/follower.py
rosrun pursuit_launch follower.py
```

  This will start the tracking node and pursuer will start pursuing the evader.

- Open a new terminal window and paste the following

```
source ~/proj3b_ws/devel/setup.bash
roslaunch pursuit_evasion move_evader.launch world_index:=[X]
```

  This will start moving the evader.

For example, lets say we wanto run the evader node in map **0**:

- Open the terminal window and paste the following

```
roscore
```

- Open a new terminal window and paste the following

```
source ~/proj3b_ws/devel/setup.bash
export TURTLEBOT3_MODEL=waffle_pi
roslaunch pursuit_evasion robot_amcl.launch
    world_file:=~/proj3b_ws/src/pursuit_evasion/
        maps/map0.yaml world_index:=0
```

  This will launch the rviz with the map **0** with both the pursuer and evader spawned in.

- Open a new terminal window and paste the following

```
source ~/proj3b_ws/devel/setup.bash
export TURTLEBOT3_MODEL=waffle_pi
chmod +x ~/proj3b_ws/sec/pursuit_evasion/darkflow/follower.py
rosrun pursuit_launch follower.py
```

  This will start the tracking node and pursuer will start pursuing the evader.

- Open a new terminal window and paste the following

```
source ~/proj3b_ws/devel/setup.bash
roslaunch pursuit_evasion move_evader.launch world_index:=0
```

  This will start moving the evader.

6 Acknowledgement

Most of the code was inspired or adapted from [4, 1]. The recording was done using zoom screen recording functionality. There are four videos:

- slam_map0 : File for slam mapping of world index 0.
- slam_map1 : File for slam mapping of world index 1.
- submission_world_0 : Pursuer pursuing the evader in the world index 0.
- submission_world_1 : Pursuer pursuing the evader in the world index 1.

Here is the link to the original files: `https://drive.google.com/drive/folders/1dyGdpcSjUvNzNOGcsSujnkGQrcAS4PKS?usp=sharing`.

Here is the link to sped up videos `https://drive.google.com/drive/folders/18JwGr8O7_KBCa9XGY7NGhvFcgSnQJnAU?usp=sharing`.

References

[1] Robotis turtlebot3. `https://github.com/ROBOTIS-GIT/turtlebot3_simulations`. Accessed: April 09, 2021.

[2] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[3] Darkflow. `https://github.com/thtrieu/darkflow`. Accessed: April 28, 2021.

[4] Gazebo ros demos. `https://github.com/ros-simulation/gazebo_ros_demos`. Accessed: April 09, 2021.