Locie: Autonomous robot for Common object localization

MAY 3 2018

Nazmus Sakib
MSc. Computing Science
University of Alberta

UNIVERSITY OF ALBERTA

Locie Autonomous robot for common object localization

Real Time Object Detection and Localization in 2D map

Objective

The autonomous robot “Locie” recognizes common objects while roaming indoor and localizing them in a 2D map. It’s a mobile robot, capable of freely moving in a known map and looks for objects. The robot is trained on 22 Common Objects and can recognize the trained objects in real time with RGB camera. It then localizes the objects with the help of depth sensor.

Background and Motivation

Real time object detection helps visualizing the environment and taking real time decision. With the help of Deep Learning it’s possible to detect multi-varied objects in the same scene. This feature when installed on a mobile platform gives rise to many more applications such as patrolling robot as a part of security, object pickup/follower robot, searching robot etc. Localizing the target objects might generate useful information, for example looking for a free parking space, when the autonomous system localizes the car in the map. The feature can be used as search and bring object in a home or industrial setting, food fetching in a restaurant, object locating in indoor for social robots.

Goal

1. **Common object detection** in real time while moving in a known map freely with dynamic path planning.
2. **Localizing** the objects in a 2D map and generating useful information/triggering predefined events during the process of navigation.

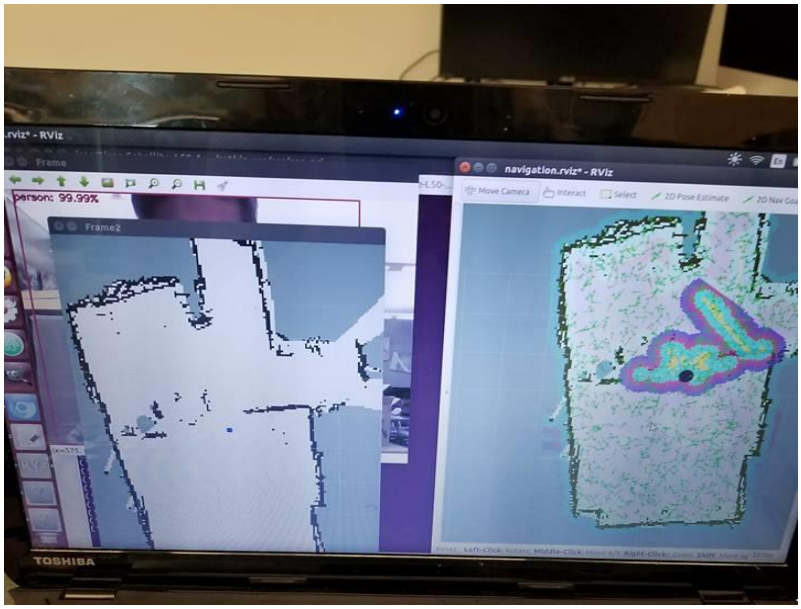


Fig 1: Sample object detection and localization

Functionalities/Hypothesis

- Create 2D map and launch software with minimum actions
- Navigate the given map avoiding the obstacle.
- Carry on board laptop for image processing.
- Reporting the updated map with detected object
- Triggering event when the condition is fulfilled. (e.g. searching for a cell phone on a table might trigger a signal(audio) to the user).

Resources/Materials

Hardware

- Kobuki turtlebot 2 as mobile base
- Asus Xtion Pro Live RGBD camera
- Laptop Running Ubuntu, TensorFlow , ROS Indigo and OpenCV
- An environment setting with common objects

Software

- **ROS packages**
  - **ROS SLAM package** http://wiki.ros.org/slam_gmapping?distro=indigo
  - **ROS navigation stack** http://wiki.ros.org/navigation/Tutorials

- **NeuralNet Model**
  - **Trained Model : mobilenet ssd**
    - **https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/**
  - **OpenCV 3.4**
  - **OpenCV DNN library**
  - **Python Socket API**

Procedures:

The real-time detection were done with OpenCV 3.4 library with the dnn class inside it. The trained model file format is in Caffe [2]. It can detect 22 objects which are : "background", "aero plane", "bicycle", "bird", "boat", "bottle", "bus", "car", "cat", "chair", "cow", "dining table", "dog", "horse", "motorbike", "person", "potted plant", "sheep", "sofa", "train", "TV monitor". It can detect with webcam running on CPU and generate bounding boxes. In our case, the mobile base ROS runs on python 2.7 and OpenCV. So the deep model is incompatible with the native environment of the pc running the ROS. So, a socket connection with TCP/IP was implemented to transfer the class of the detected objects and the bounding boxes to ros program.

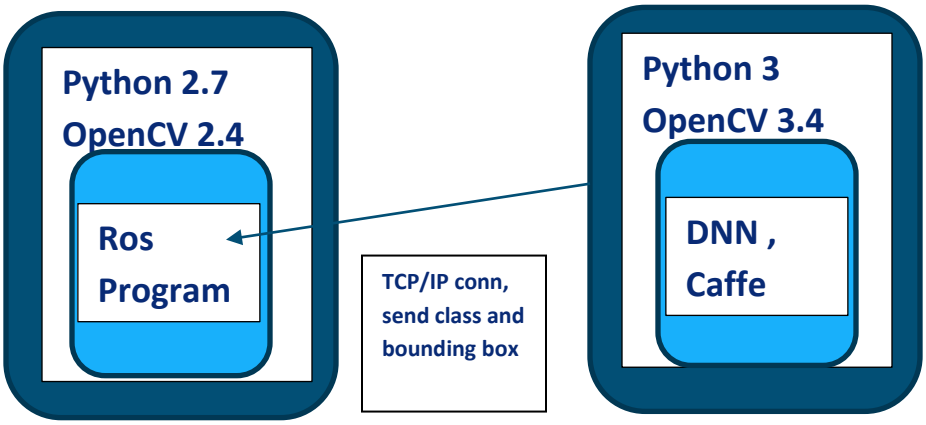
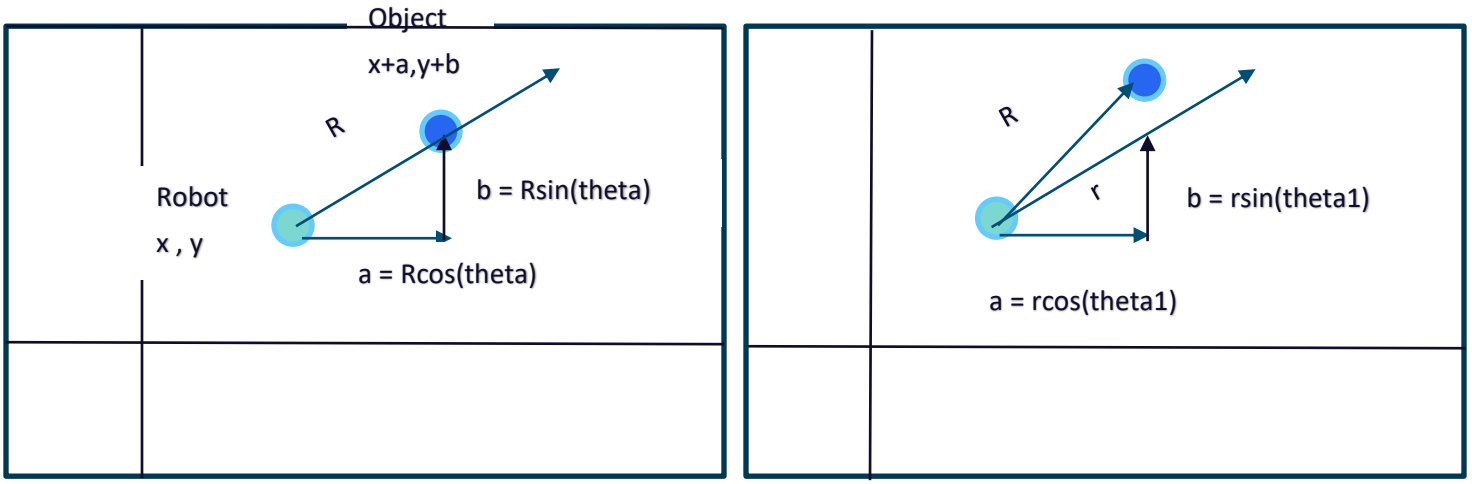


Fig 2: Bridge between 2 environments

In order to run the ros program appropriate launch file is created to launch all the necessary support from ros library. At first, the socket connection is established. Then all the necessary ros subscriptions are done. The "global_localization" service help to spread the particle filters on the pre-loaded map. The map was created using gmapping tool of the ROS SLAM package. The robot goes some sequence of activities (see next section). It rotates or take random walk to localize at the beginning and enter into searching state.

During searching it goes to some predefined goals within the map with amcl “send_goal()” function. But whenever search criteria is met it does the previously set activities. For our case we create sound where it says the class of the detected object. Then the robot follows the object (this was designed to show the robots capabilities/functionalities), From the bounding box over RGB image we get the corresponding depth scan values which help the robot to follow with simple p component of the PID control algorithm.

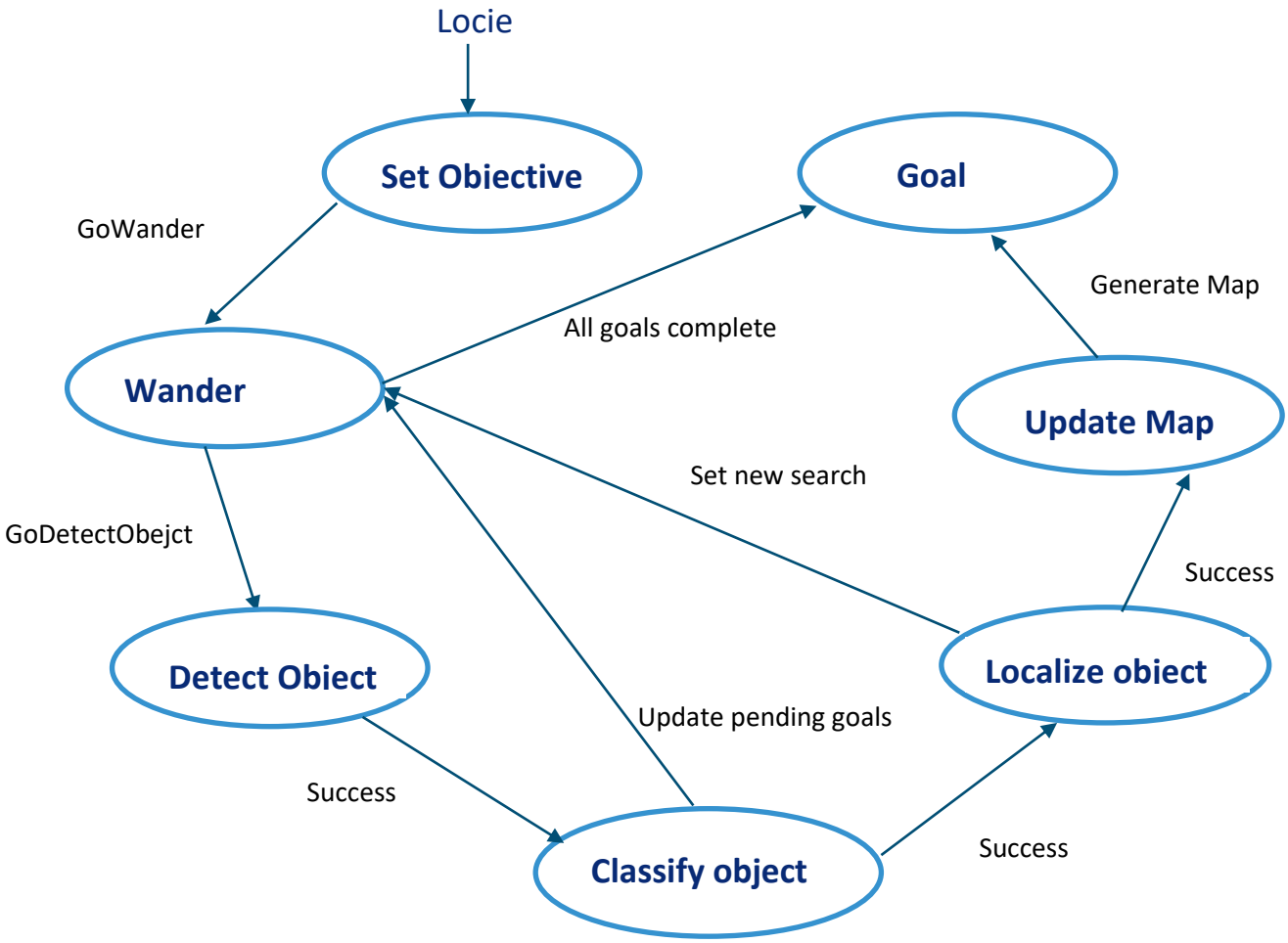
The robot also generates custom map where it localizes the robot’s location after subscribing to the amcl pose topic and map it corresponding pixel values of the map. For locating the detected object it always assumes that the object is right in front of the sensor (fig 3 , case1) and use the robots co-ordinate (x,y) from amcl pose and sensor’s depth reading R . But the robots sensor is a 2d sensor which can generate a range scanner vector of 640 values. Therefore, the depth values coming from the values other than no. 320 creates an angle with the angle of the robot’s direction. The robot’s orientation has also been calculated from the pose estimation and the quaternion values are converted into Euler’s value and yaw is found. We observe that the yaw value always range from 0 to 3 for 0 to 180 degrees and -3 to 0 for 181 to 360 degrees. So, we map this value and calculate theta to plot on our custom map.



Case 1 Case 2

Fig 3: Object Localization with respect to robot.

Activity Diagram



1. **Set Objective :** Define the goal of Locie.
2. **Wander :** Visit different locations of map avoiding the obstacle.
3. **Detect Object :** Run Object detection while wandering and when found classify into appropriate categories.
4. **Localize Object :** Take the bounding box of the object detection , filter with depth values and localize.
5. **Classify Object :** Classify in between different object and get rid of false positive detection , update search mechanism in wander.
6. **Update map :** Put the localized object in the map and publish when goal criteria met.
7. **Goal :** When all the objectives are met reach goal criteria , publish map.

Results

The real time object detection and ROS should be under the same package libraries. The bridge between different versions of python and opencv creates lag. It also reduces the robot speed. The performance of the object detection is very good considering the resource it is running on and the accuracy is also acceptable when observed over longer period. The robot localization in custom map is accurate but the object localization in custom map is not acceptable since it didn't consider the case 2 of fig 3. Also current code detects single object, plotting multiple object is a challenging one. But the robot is good for search and indoor localization. It can successfully report where it found certain object. It can successfully navigate and look for multiple objects at the same time, see the videos on [5]

Analysis

The use of deep model with CPU was challenging. The environment mismatch was a great disaster for the project progress. The algorithms used are easily reusable. The use of thread, member functions of the class, robot state parameter have made the project more robust. It can easily be extended to future work. Although it is successful for robot positioning in custom map but localizing objects is tricky. The use of custom map is not necessary but the information used to build it is the basis of more functionalities for object grab/pick robots, social bots , cleaning bots etc. This information is the result of dynamic environment change around a known map which lead us to advanced operations.

Conclusion

The project is a good example of using deep models to detect real time objects and run it on a mobile platform, retrieve the dynamic changing objects and plan accordingly. It can also be used after training own custom deep networks for image classification or analysis.

References

1. **Project Template :** https://eclass.srv.ualberta.ca/pluginfile.php/4145726/mod_page/content/22/Demo7%20Project%20Proposal.pdf
2. **Caffe** http://caffe.berkeleyvision.org
3. **Tensorflow models** https://github.com/tensorflow/models
4. **ROS sample codes** https://www.osrfoundation.org/ros-osrf/
5. https://github.com/nsa31/ROS_RealTime_CommonObject_Localization