Christian Loucka

Lakiel Wade

Emmanuel Manu

Autonomous Soda Searcher

Initial Review:

In our proposal, we originally stated that our goal was to search for and positively identify a soda can of a certain type within the robotics lab. After attempting this and finding difficulties along the way, we were able to achieve a modified version of this goal. In the end, our Turtlebot was able to identify and face the text of a queried can name within a rough estimate of its final position. Similar to our proposal, our findings for the two main stages of development can be found below.

Search Pattern: Lakiel Wade

For the robot to search it requires a known map of the area and knowledge of specific places to search for soda. For this project ROS Navigation stack is used to build a map, localize and navigate.

To build a map we used ROS navigation stack Gmapping, this package is a wrapper for SLAM. Simulation Localization and Mapping (SLAM) is the most up to date construction model of the environment, and the estimation of the local state of the robot moving inside the environment, the SLAM community has been working on this model for over 30 years. The concept of SLAM is described as the robot state from its on-board sensors used to perceive the environment and pose (position and orientation). The map is a representation of objects of interest that describe the environment in which the robot operates. The need for a map in navigation is major, the map offers support for tasks such as path planning and visualization for human operators. The map also limits the error percentage in estimating the state and pose of the robot. Lastly the map can also help distinguishes landmarks; this can help the robot re-localize by revisiting known areas or loop closures.

Furthermore, for the robot to autonomously navigate around the world we used ROS navigation amcl, this package is a probabilistic localization system for a robot moving in 2D it implements the Monte Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map. Monte Carlo Localization (MCL) is a version of Markov localization, and a family of probabilistic approaches that have recently been applied to many practical approaches. MCL is a version of particle filtering the underlying idea for this method is to represent the posterior belief by a set of weights, random samples or particles. A sample set constitutes a discrete approximation of a probability distribution. In practice when the robot moves, MCL generates new samples that approximate the robot's position after the motion command. An interesting property of the MCL algorithm is that it can universally approximate arbitrary probability distributions.

Lastly, combing these to algorithms and approaches we can give the robot knowledge about where the soda pop might be. Currently the robot navigates to a specific location and spins in a circle to scan the surrounding area. A later improvement can be to give the robot a probability distribution of multiple location of where the soda can be found and have it autonomously decide which location is best to observer first and increase the reward for finding the object in that location. more information for these to package can be found at the following links:

ROS Navigation - http://wiki.ros.org/navigation

ROS Gmapping - http://wiki.ros.org/gmapping

ROS Amcl - http://wiki.ros.org/amcl

SLAM - https://arxiv.org/pdf/1606.05830.pdf

MCL - http://www.aaai.org/Papers/AAAI/1999/AAAI99-050.pdf

Finding Can: Emmanuel Manu

For this project, I was assigned to design a database framework by the team. I designed a database using ORK DB to store data. ORK will uses couchDB to manage soda search database.

We will feed the database with 2D models.

When you first installed ORK, my database was empty. ORK tutorials comes with 2D model of a coke can:

```
git clone https://github.com/wg-perception/ork_tutorials
```

then uploaded it to the ORK database:

```
rosrun object_recognition_core object_add.py -n "coke " -d "A universal can of coke"
rosrun object_recognition_core mesh_add.py <the object id that previous command returned> <path to the ork_tutorials/data/coke.stl>
```

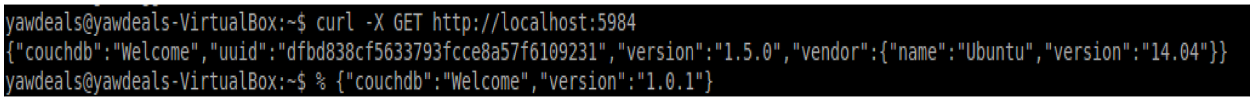
Setting up the couchDb:

First, to set up the local instance, I installed couchdb and ensure that the service has started.

I used the below command to download and install:

```
sudo apt-get install couchdb
```

To test that it is working properly, I used the command below:



```
yawdeals@yawdeals-VirtualBox:~$ curl -X GET http://localhost:5984
{"couchdb":"Welcome","uuid":"dfbd838cf5633793fcce8a57f6109231","version":"1.5.0","vendor":{"name":"Ubuntu","version":"14.04"}}
yawdeals@yawdeals-VirtualBox:~$ % {"couchdb":"Welcome","version":"1.0.1"}
```

I configured and set the port to 5894 and set the bind_address to 0 . 0 . 0 . 0 . 0 to allow couchdb to access on an interface.

After the database is setup. I run the below command in the terminal to see how find_object_2d package will work.

```
rosrun object_recognition_core detection -c  `rospack find
object_recognition_find_object_2d `/conf/detection.object.ros.ork
```

After following the steps above. The link:
http://localhost:5984/or_web_ui/_design/viewer/objects.html

Shows where the can object will be listed in our database. (which can be accessed locally on my server)

On RViz, we added ORK Object display. This enable a can placed on the detected planes to be seen in the RViz interface. For this to work. I placed the topics in con fork file and in RViz and change the topic to qhd.

This framework took a different path because we did not need it for this project. However, this will be used for future and advance projects. That will need a 3D model to work

Some problem I encounter with this database was, the 2D inputs seems to wait for the ROS topic forever. This happened a couple of times during my testing. I found out it was due to the fact the 2D camera was not listening to the topics that was published.

We decided we do not need the database for this project. So I help Lakiel and Chris on their respective projects.

Can Recognition: Christian Loucka

To achieve the task of recognizing a can within a given environment, the original proposal was to first use RGB-D processing to identify the shape of a 12oz aluminum can and then using RGB processing to analyze its label/color to determine its type. This idea was quickly scrapped in favor of using strictly RGB processing. This decision was not without its own pros and cons, however. One of the largest cons was performance. By analyzing every pixel within a given frame, the amount of time required for each frame input was quite large. Because of this, the maximum frequency that could be consistently processed was 5Hz. However, a major pro to using 2D recognition was the ability to use a very versatile and pre-existing ROS package name find_object_2d. Within this package, we were able to easily utilize an object within a given scene provided by the camera as our training data. Through visual analysis using both the FAST and BRIEF algorithms provided by OpenCV, the given training data was able to recognized later on in the frame. The necessity for these two algorithms is for FAST to handle Corner Detection while BRIEF is used to analyze the individual features within the corners detected by FAST. By combining these two methods of evaluation, reliable object detection can occur.

After pulling in examples of different soda cans a node was created to interact with the topic published to via find_object_2d. This topic, defaulted to “objects”, was used to host information regarding objects detected by the find_object_2d package. Using a custom node of our own, we were able to subscribe to this topic and, therefore, retrieve information pertaining to any objects being detected. The sole purpose of the image detection node was to subscribe to two topics and publish to one. One of the subscriptions as stated earlier was the information being published by find_object_2d. The other subscription being the queried object that is currently being searched for. This topic was published to via the node created by Lakiel as explained in his section.

Finally, the can detection node would publish to a “detection” topic when a queried can was found within the input frame. The message here would simply be a string set to “Found”.

As for issues experienced with the actual implementation, one was consistently encountered. This issue being a lack of recognition of a small can when analyzing a small resolution video feed. The initial solution to this was using a webcam that was cable of providing a 1080p video feed, however, the find_object_2d and/or usb_cam package appears to limit the input video feed to 640x480. This would make sense seeing as RGB processing time dramatically increases with the number of pixels being presented to the various algorithms.

Aside from a lack of the resolution, the frequency at which a frame of video could be interpreted also caused issues. In maxing out at 5Hz consistently, the window of time that the objection recognition software had to identify a can within a moving frame was quite low. Our solution to this was to dramatically slow down the speed at which the robot would rotate. By providing the object recognition with a slowly changing environment, the chance of a proper detection was greater.

Finally, a second issue regarding lack of resolution was identifying a small can at distances further than one or two feet. Because of the can’s size in addition to the lack of detail captured when interacting with low resolutions, the ability for a can to be recognized was not great at far distances. To combat this, the recognition of a can was scrapped in favor of text written on cardboard. By spelling out the soda’s name in large and distinct letters, the number of successful detections increased greatly.