Learning in Mobile Service Robots

He Chen, Varun Gupta, Seungwon Lee, Shashank Shivkumar
Monica Vyavahare, Christopher Zawacki, Hongrui Zheng
CIS700II - Integrated Intelligence for Robotics
University of Pennsylvania

Abstract

Intelligent service robots require the capability of learning new knowledge from experience and exploiting the knowledge to improve the way of completing the given task. To achieve such capability, the learning focus group developed database API for collecting training examples from experience, systems of learning the characteristics of environment, especially spatial distribution of objects and temporal/spatial distribution of people in corridors, and system of learning actions from demonstration.

INTRODUCTION

The learning group focuses on developing the functionality of the robot to collect data, process data to learn the knowledge, and provide new knowledge back to the user in a more refined form. As this is the first learning focus group of the course, the major goal of learning group in this semester was to set foundation of work in learning by providing API of database for collecting data and sample capabilities on the basis of learning features. These sample abilities are learning appropriate action from the given demonstration and learning the pattern of the environment from the observation such as the distribution of objects. We expect that API of database will be necessary part for the work of future learning group and the sample capabilities will provide useful functions for more complex tasks. This report describes accomplishments of the learning group in this semester and required amendments for the better usage.

Database API for Learning

One of Learning focus group's achievements this semester is the implementation of an interface between a MongoDB database and ROS for future development for the learning focus group. Learning new behavior from demonstration, learning traffic patterns, etc. require collection of the data from the sensors on-board the service robot prior to the application of any learning algorithm. Therefore, a ROS package which subscribes to ROS topics and stores data in and retrieves data from a database is necessary.

The database API has four core functionalities: 1) subscribing to ROS topics and writing data into the database, 2) retrieving data from the database and publishing to ROS topics, 3) transferring data from a bag file to the database, and 4) transferring data from the database to a bag file. These functionalities are wrapped by launch files.

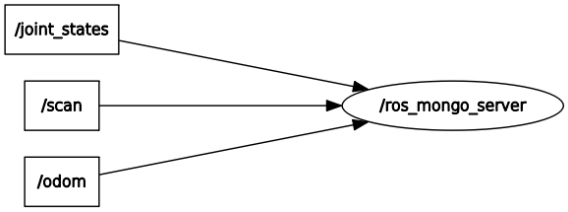

Figure 1: rqt graph of node ros_mongo_server

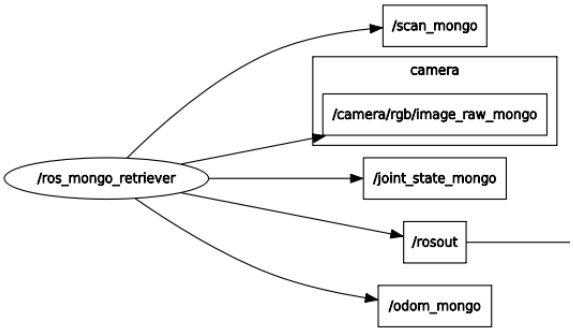

Figure 2: rqt graph of node ros_mongo_retriever

ROS node ros_mongo_server(*ros_mongo_server.py*) subscribes to four ROS topics(four at the moment and can be expanded as needed) to record the data from sensors on the service robots. The interaction between the node and the topics are shown in Figure 1. In each of the topic callback methods, the data is parsed according to MongoDB conventions and inserted as id-ed post into the connected database.

ROS node ros_mongo_retriever(*server_to_ros.py*) regenerates ROS messages and publishes them to new topics according to the query given. The node determines how the MongoDB posts are parsed back into ROS messages based on the tags of the post. The messages are published to ROS topics created just for this node as shown in Figure 2 to avoid conflict. These topics also come in handy when some of

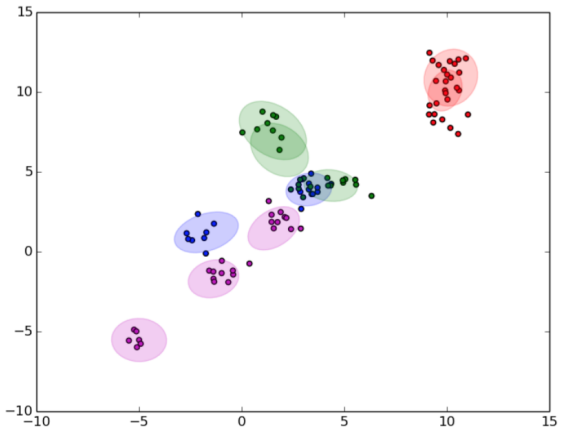

Figure 3: IGMM clusters and objects on the space. According to the distance between newly observed object and clusters (small circle), the object is included into an existing cluster or makes new cluster for itself. This is demonstrated for simulated dataset.

these data are needed, because other node can just subscribe to these new topics. The number of topics can be easily expanded to accommodate new sensor in the similar way of modifying ROS node ros_mongo_server for new sensor.

The usage of the database API can be found in the README on the repository. The database API is designed for MongoDB. Since the map data server and all other database-related code migrated to CouchDB, the database API should also migrate to CouchDB.

Learning Distribution of Objects

The learning of the distribution of objects is an integrated system providing information about where objects are located at around the map. As the robot navigates around the building, its vision system can detect objects. By combining the vision data with the navigation's localization and orientation data, it is possible to approximate where the observed object lies on the map (object localization).

Currently, we are using an incremental gaussian mixture model (IGMM) (Engel and Heinen 2010) and some of the code courtesy of the ESE 650 report on Semantic Map Layering by Alex Baucom, Sakthivel Sivaraman, and Sai Krishnan Chandrasekar. Ideally, each distinct object is represented as a cluster on the map of clusters. To add to the terminology, a group of all the objects (clusters) in a floor is called a layer.

Finding a cluster of object from observation is required because the system should be able to associate different observations of the same object in the presence of noise and the slight changes of the object's position. This reasoning is possible if the mean and covariance of accurate cluster is learned from the observation.

The IGMM maintains a standard mixture model representation

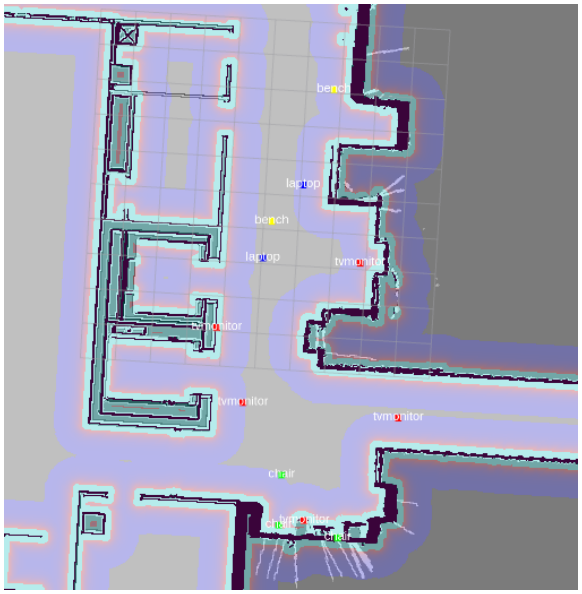

Figure 4: IGMM clusters visualized on the map using data from the object detection and localization pipeline.

$$p(\mathbf{x}) = \sum_{j=1}^{M} p(\mathbf{x}|j)p(j) \tag{1}$$

where $p(j)$ is the prior, or weight, of the jth cluster and $p(\mathbf{x}|j)$ is the Gaussian probability that $\mathbf{x}$ belongs to cluster j, which is defined as:

$$p(\mathbf{x}|j) = \frac{1}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}} exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{C}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right) \tag{2}$$

where D is the dimensionality of the data, and $\boldsymbol{\mu}_j$ and $\mathbf{C}_j$ are the mean and covariance matrix of the jth cluster respectively.

Then a *novelty criterion* is defined as:

$$p(\mathbf{x}|j) < \frac{\tau_{nov}}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}} \forall j \tag{3}$$

where τ_{nov} is a user defined fraction that determines how novel a new data point must be to create a new cluster. If the *novelty criterion* is met, a new cluster is created; otherwise, an update step is performed. The full update step and component creation equations are presented in Section 2 of (Pinto and Engel 2015), and Figure 3 shows location of objects and their clusters.

A key feature to note of this algorithm is that, in addition to maintaining the standard GMM values of $\boldsymbol{\mu}_j$, $\mathbf{C}_j$, and $p(j)$ for all j, two other values v_j and sp_j are maintained and updated for each cluster. The value of v_j keeps track of how many updates have occurred since the cluster j was created and the value of sp_j is a running sum of $p(j|\mathbf{x})$. In essence, v_j measures how long the cluster has existed and sp_j measures how important the cluster is. These values are

used for both updating the clusters during the update step and pruning out clusters that are not important. A cluster is pruned when $v_j > v_{min}$ and $sp_j < sp_{min}$ where v_{min} and sp_{min} are user defined values.

Building upon the IGMM for learning region-wise object distributions, we tried to study and infer pairwise object relations which we call object association. The idea is that if the robot is looking for a marker pen, it may know that it can find a marker in a conference room but may not be able to see a small marker from a distance. But after associating objects the robot would know that the marker will be extremely close to a writing board and this will help guide the search process of the robot. Similarly, the robot would learn simple things like chairs and tables go together, chairs and monitors are often together, etc. Keeping these kinds of application in mind, we decided to infer the following relations between object.

- Which object is almost always next to another object?
- How likely is it to find object A near object B?
- How far apart are objects A and B if they are likely to be in the same room?

To answer these questions, we first calculate the probability of finding object cluster i in another objects cluster, say object j using the expression below

$$p(\mathbf{x_i}|j) = \frac{1}{(2\pi)^{D/2}\sqrt{|\mathbf{C}_j|}} exp\left(-\frac{1}{2}(\mathbf{x_i} - \boldsymbol{\mu}_j)^T \mathbf{C}_j^{-1}(\mathbf{x_i} - \boldsymbol{\mu}_j)\right) \tag{4}$$

where all the conventions are similar to IGMM with the only difference that x_i here denotes the mean position of the i^{th} object cluster. Here we assume that the mean of the object cluster is a good representative of its position in the real world since the parameters in the IGMM have been tuned accordingly. This probability is averaged up over all clusters of object i and object j that are within a specified distance range, selected to be 5 meters in our implementation, and weighted by an inverse exponential distance function. This weighting ensures that objects that are relatively far apart have their probability scaled down and make it easier to compare different pairs. The obtained probability is a good measure of how likely an object can be found near another object as has been observed from simulated experiments.

To incorporate all these features, we provided a ROS launch-able package with services using the information contained within the IGMM layers. Figure 4 visualizes one outcome of our package which shows location of detected objects on the map via ROS rviz.

1. Simple lookup provides a way to retrieve the location of an object given the name of that object. It can either return the location this object exists at with the highest probability, or return the closest object with that name to a given point, depending on the search mode.
2. Object pair likelihood lookup takes two object names and returns the probability of finding the two objects near each other
3. Alternate object search takes an object name and returns the name of a different object which is likely to be found near the given object
4. Object location lookup returns all the locations of an object, given the name of that object
5. NLP region recommender takes in a phrase such as "drinking fountain" and returns the semantically closest region which most closely associates with that phrase, which may be, for instance, the area outside of a bathroom right next to the label "water fountain". This interface currently uses a word2vec model trained on the Brown corpus to convert words into a semantic vector.
6. Count objects returns the count of all of the given objects within a particular time frame and within a particular distance of a given location

In addition to these services, the package also provides 2 additional services that allow the object clusters to be stored and retrieved. This allows the robot to retain knowledge from past runs and use it for performing tasks better in the future. To allow this, the package provides services that allow:

1. Update Data Manager with object cluster information: This service writes the object cluster means onto the database with 2 tags, namely *objects* and current timestamp, in order to uniquely identify these clusters and not to confuse with the earlier stored clusters since the information contained in the earlier clusters is carried with the new set of clusters.
2. Initialize IGMM model with object clusters from database: This service reads in all the clusters from the database based on the appropriate tag, in our case, the string *objects* and the latest time stamp, publishes it onto a topic that is subscribed by the IGMM and initializes or updates the model with this old knowledge in the form of obejct clusters.

The object distribution model as well as the object association model could be further improved by:

1. Using batches of information for IGMM updates which would make it invariant to the order of observation.
2. Using confidences from the object detection output.
3. Adding hierarchies to the objects in relevance to their frequency of detection in an environment and the size of the object itself. This would be more realizable in application front.
4. Including the sizes of the objects in the likelihood estimation model.

Learning Traffic Pattern

When traversing common foot traffic routes, the cost associated with a path is impacted by the number of people a robot needs to navigate around. No matter how robust the navigation stack is, if a crowd can be avoided with minor increase in the distance of path, the longer path should take priority in terms of both safety of the navigation and traversal time. Thus, a given hallway should have a cost associated to the

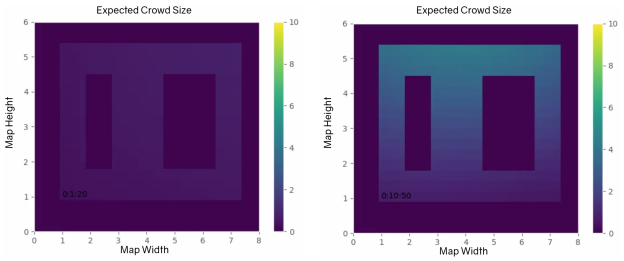

Figure 5: Expected crowd size at 1:00am (left) and 10:50am (right)

estimated time of travel, not the distance only, to determine the desirable path among alternatives. To achieve this, it becomes necessary to predict the number of obstacles which an agent is expected to run into at a given time.

For this task, we choose to implement a Gaussian Process Regression model on the basis of the python GPFlow library. The goal of the current system is to learn the traffic distribution of people over a map throughout the course of a day. The current system utilizes discretized space and time. The output of the regression model is the expected number of people in a given hall region at a given time of day. The current system works on a single cyclical day; however, it could be expanded to any number of days or a whole week. To allow online and offline training of the model, the observations can asynchronously and randomly occur. The current simulation assumes that an observation is made every few minutes at a random location. This assumption would allow multiple robots to periodically update the internal belief of the system based on the most recent observations.

Due to the low demand for this capability, this work is still a proof of concept and it is not wrapped as ROS package. Currently, the system loads a map as a binary array where a wall corresponds to 1 and a hall as 0. Every n minutes a random sample representing the number of people is observed as a function of location and time. This was set up to reflect the rush of students before the hour a class starts.

As seen in Figure 5 the left image reflects a time with minimal energy. The image on the right however occurs 10min before a class and so the top hall, made to reflect the main hall of Towne, is much busier than the back hall.

The next steps for this work would be wrapping the model in a ROS capable node to interface with the map server and vision system. The model becomes slower as the space is more finely discretized, so the way of storing and loading pre-trained models is necessary.

Unfinished Components

Learning from Demonstration

Learning from demonstration is very useful in the case of problems where it is difficult to mathematically formulate desired behavior in terms of the robots state (or relevant features extracted from it). There were a few possibilities that we had discussed in terms of what a suitable goal for implementation would have been. Some of these were:

1. Manipulator control
   (a) Grasping
   (b) Pick and place
   (c) Pushing a button
   (d) End-to-end learning from images
2. Local navigation and obstacle avoidance (moving naturally around dynamic obstacles).
3. User intention recognition.

Some of the major issues with pursuing this technique that we anticipated were:

1. Most algorithms were created for discrete state and action spaces which are not suited to solving these problems in robotics which are usually high dimensional.
2. Reward structures are highly nonlinear: especially, manipulation tasks using simple features such as the arm state.
3. Running multiple reinforcement learning episodes within the inverse reinforcement learning loop would be impractical without the simulated system.

Additionally, it is still an open problem in inverse reinforcement learning to dynamically switch between complex sequences of tasks which are useful for a practical scenario. For instance, the possible subtasks of opening a door with a manipulator are positioning near the door, grasping the handle and turning the handle while pulling the door. It is difficult to both giving demonstrations of this sequence of tasks, finding useful feature representations for learning, and dealing with failure, so even algorithms of hierarchical learning have been used over basic actions.

Besides aforementioned difficulties of inverse reinforcement learning, we decided to begin with problem of navigating through small environment with obstacles because this problem has the least dependence on work of other focus groups like vision and manipulation which was incomplete at that time. Specifically, the problem that we aimed to solve was to navigate to a fixed and known goal in the presence of a single obstacle which can be either static or dynamic. This setting of goal location can be extended to the problem navigating longer distance in environment with obstacles by selecting a single local way-point on the path sequentially for the goal of this learned behavior.

We used feature set which consists of followings: radial vectors pointing goal and obstacle and the angle between them. For the algorithm of inverse reinforcement learning, we focused on the learning from demonstration algorithm of paper (Levine and Koltun 2012). This paper proposed the approximation of maximum entropy (MaxEnt) loss function which allowed the MaxEnt inverse reinforcement learning to work with continuous state and action spaces. We chose this algorithm because this method is also compatible with other high dimensional robotics problems such as manipulator control above.

Demonstrations were collected by teleoperating a robot in the environment with a fixed goal and a static obstacle but different starting locations. Control of the robot for giving demonstration was achieved by an Xbox 360 controller

and the existing turtlebot package for Xbox controller inputs. The reason why Xbox 360 controller was used instead of keyboard is that keyboard teleoperation of the turtlebot responds only to a single keypress at a time which makes demonstration more choppy and imprecise.

We attempted to use an available MATLAB implementation for the method in paper (Levine and Koltun 2012). This was not completed since it was not an immediate requirement of other focus group and team; however, ROS-based controller for navigating the robot according to the given specific policy and feature was implemented and it is now in master branch of github repository. Some future steps that we envision to achieve would be:

1. Creating a dedicated simulator for learning from demonstration.
2. Finding a compatible C++/Python implementation which can be used directly or using Matlab-ROS plugin to use existing implementation of the learning algorithm.
3. Using raw LIDAR readings as features to a nonlinear reward learning algorithm (possibly using Gaussian processes or deep learning). This would be able to generalize better to new environments and incorporate features from vision and depth sensing.

Open Issues

In this semester, our efforts were directed toward developing basic functionality in navigation, vision and manipulation. It was difficult to simultaneously implement learning algorithms that utilized these core operations. Developing learning algorithms in upcoming semesters will be easier because the core functionality of the service robots are close to fully-developed. There are several open issues which can be starting points for upcoming semesters.

1. The data collecting API needs to be migrated to CouchDB.
2. The NLP region recommender currently uses a locally hosted, self trained word2vec model on the brown corpus. However, this is not reliable. Instead, we should use the Google word2vec model (3.5+ gigabytes file size), host it on a server, and make it available as a service.
3. Currently, the layer server in object distribution only works with one region ID, meaning one region of the map. It would be nice if the layer server additionally keeps a dictionary for each region ID that maps to the actual layer for that region ID. Additionally, all of the observed locations in object localization only writes to a single topic for all of the region IDs. This is more of a vision change, but it should write to a different topic depending on the region ID, so that we can update different models pertaining to different regions.

References

Engel, P. M., and Heinen, M. R. 2010. *Incremental Learning of Multivariate Gaussian Mixture Models*. Berlin, Heidelberg: Springer Berlin Heidelberg. 82–91.

Levine, S., and Koltun, V. 2012. Continuous inverse optimal control with locally optimal examples. *arXiv preprint arXiv:1206.4617*.

Pinto, R. C., and Engel, P. M. 2015. A fast incremental gaussian mixture model. In *PloS one*.