Knot-Tying from Learning from Demonstration

Humanoid Robotics Project

Team 8
Members: Sneha Silwal (ss4547), Yanrong Wo (yw2513)

Project Abstract

Manipulation of deformable objects, such as rope and clothes, is a common household activity that has been a challenge to robotics for the past 3 decades. As an attempt to tackle this problem, this semester we attempted to implement and run a learning from demonstration approach proposed by a group at Berkeley, who described their approach in a paper titled "Learning From Demonstrations Through the Use of Non-Rigid Registration" [2].

Their main contribution was a technique called *trajectory transfe*r which computes transformations for a new test scene given data on an existing training scene. To apply this new technique, the authors suggested segmenting a task in multiple steps to create the training scenes. The robot would then repeatedly consider its own scene during the test phase and find the nearest neighbor to the previously recorded training scenes.

They broke down this process into (1) registering the point clouds with unknown correspondences from the training scene to the test scene (using TPS-RPM from Chui and Ragnarajan[3]) to find the most similar test scene, (2) creating a transformation to warp the most similar training scene's point clouds into a similar position as the current test scene's, and (3) then applying the transformations to the end effector and trajectories (followed by an optional minimization/simplification of this path). The end goal was to be able to take previously recorded demonstrations and be able to carry out the same actions on the new run in which the geometry of the shape was different from before.

Related Work

As previously mentioned, our main paper in which we will take our algorithm and implementation from will be "Learning from Demonstrations Through the Use of Non-Rigid Registration" which we will now refer to as the "LfD paper" [2]. The technique and algorithm were previously discussed and will not be repeated. The paper can be found here: http://joschu.net/docs/2013-ISRR-LFD.pdf and an accompanying video of their implementation in action on the Berkley PR2 can be found here: https://www.youtube.com/watch?v=AJllD3AiSqs.

In the related works of the LfD paper, Schulman et al. mention that there have been numerous other works in the learning from demonstration field, but of interest, there are two works with similar goals to theirs by Calinon et al. which also allows for different initial configurations in manipulation tasks (using learning from demonstration) [5][6]. However, Schulam et al. were not able to use this approach to tie ropes as Calinon et al. relied on a fixed length vector of landmarks which is not present in knot-tying tasks. Since this implied that this would not be helpful in our goal either, we did not explore this paper in further detail.

Another more recent paper involving manipulation of rope was “Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation” by Nair et al. in which the goal of the robot was actually to learn the effects of an action on a rope [1]. Generally, their approach was to have the robot repeatedly select a point to pick the rope up at and then a point to drop the rope off at and then observe effect. However, this approach would require a lot of time as the robot needs to do the pick/drop many times to be able to learn the effects of its actions. Since we did not have this extended access or time to let the robot do this, we decided against this paper.

The last paper we considered was “Folding Deformable Objects using Predictive Simulation and Trajectory Optimization” [4]. This paper focused on the task of folding clothes and in particular, trajectory optimization using an online optimization algorithm that combines manipulation of mathematical models and predictive thin shell simulation. However, this paper was originally done at Columbia University’s Robotics Lab, so we decided we wanted to try something different.

What We’ve Done

Progress

We were able to get the robot to record demonstrations and generate annotations for the recordings, by modifying the code given in LfD. The robot also is able to identify the nearest neighbor of the demo and pick up the rope and work on stages for tying the knot.

Our modified code currently does the following:

When creating a demonstration, the program looks for a “master” file to keep track of all the files involved. A folder will be created for color and depth images received from the Kinect. Two subprocesses are then run to do the following: (1) creating a rosbag to record messages for the joint states and joysticks and (2) listen to color and depth rostopics and sync them with timestamps. Using the joystick, the demonstration is recorded in repetitions of three parts, in a (“Look”, “Start”, “Stop”) fashion :

1) **Look**: to signal the a look at the state of the rope,
2) **Start**: to start recording the joint states of the arms (which are in mannequin mode) as they carry out an action,
3) **Stop:** to signal the end of this action, and (repeat 1-3 as required)
4) **Done:** to signal the end of the demonstration once the rope has been tied.

Once the recording is complete, annotations are generated for the demo and saved in a yaml file. These annotations consist of the timestamps of the (“Look”, “Start”, “Stop”) in addition to one for the Done stage. After this step, we generate h5 files which involves generating point clouds from the Look stages of the demo. Finally, we run the program that will make the robot execute the same task on a different rope configuration by itself!

As of now, the robot is able to grasp the rope and perform the general motions of the task, but at times, will miss.

Problems encountered

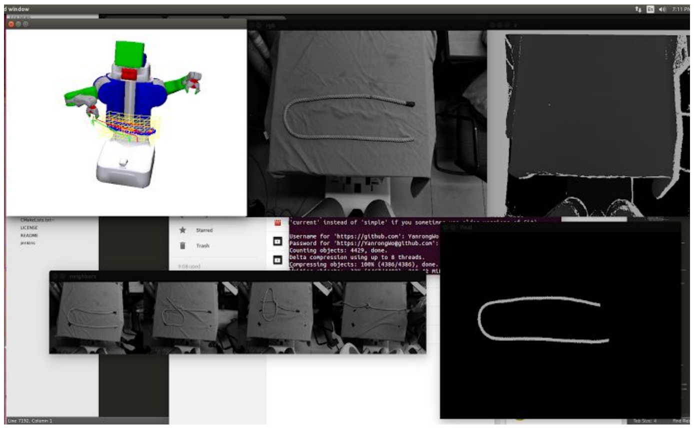
The following is a list of problems that we encountered while trying to get the code to run correctly. In addition to describing the problem, we list our attempts to fix the problem as well as the final working solution.

- **OpenRave, Trajopt, Gurobi**: The code base relied on the previously listed libraries. We downloaded the libraries from the official sources and went through the process of installing them as listed by their documentation. However, the currently released version of Openrave were not compatible with Linux 14 and we had to revert to an older branch (on github) of Openrave in order for it to be compatible with our operating system. When compiling Trajopt, we had to add an extra flag “BUILD_CLOUDPROC” so that it would build a supplementary library for cloud points.
- **OpenSceneGraph Viewer**: The osg viewer was not opening when we were attempting to run the simulation because we were importing an opencv module. To work around this, we imported cv2 only when used, rather than importing it at the beginning.
- **Syncing Time with Basestation:** Initially while running through the execution of the task, the robot’s arms were not moving. However, the PR2 would sometimes continue the action expected of it after we had quit the program, we realized this was because the time of the base station was not synced with c1. To remedy this, we made sure to sync the times upon start-up of the PR2 in every session.
- **Kinect on Robot:** Unlike the LfD authors, we have our Kinect mounted on the robot instead of a computer, so we were unable to use the Openni library. We instead changed our code to record rgb and depth information from two ros topics: kinect2/hd/image_color_rect and kinect2/hd/image_depth_rect. However, when testing, we found that the robot did not seem to have the right depth. So we compared the depth values we got to those from the given sample data and found that we had drastically different depths. We looked into fixing this by possibly downloading KinectSensor (a driver needed by Openni to interface with Kinect), a driver that would allow us to receive the depth in mm from the head. We began to install this software, but was unable to get it running successfully. In the end. we found that we just needed to save the depth files as .npy (numpy array) files instead of as cv2 (image) files.
- **Downloaded new system clock synchronization daemon on pr2-head.** When downloading the previously mentioned KinectSensor, we also updated the clock synchronization daemon. After doing so, we found that the clock on the robot head was no longer synced with c1 (the base robot computer). We fixed this by turning off the wifi on the robot head so that it could sync with c1 instead of web sources.
- **Hand eye calibration:** After fixing our depth, the robot’s actions were still not accurate enough. We ran tests to calibrate the PR2’s cameras and kinematic parameters with this tutorial. We ran into issues with the calibration when the robot’s wrists were not correctly rotated so that it could face the checkerboards used for calibration. We originally tried to change the rotation by modifying the robot’s URDF file, but this did not fixed the problem. In the end, we had to take changes from another fork of the github repository that correctly rotated the wrists. (We ran into additional issues and were not able to finish the kinect calibration process).

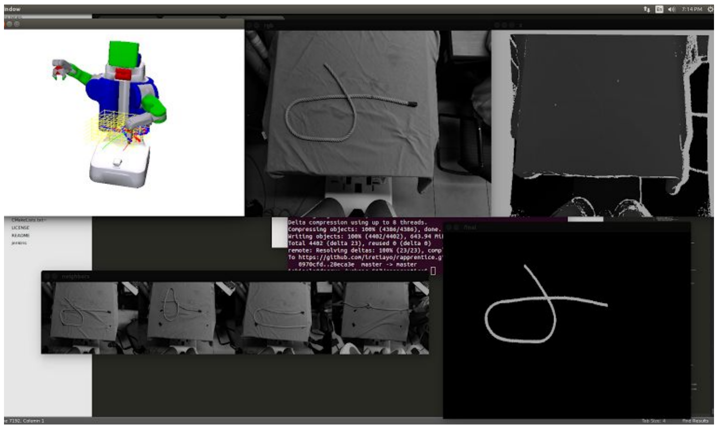
- **Camera Intrinsic Parameters:** We tried both running the simulation with our parameters for f, cx, and cy as well as theirs. We initially thought using our own would be more accurate, but we found that their parameters resulted in a better execution of the task (and display of the rope in the simulation).
- **Masking colors:** Our rope color was different from the LfD authors'. Initially, we just changed the HSV values of the masks they provided. However, while testing, we noticed that the colors from the environment that matched with the rope would get picked up. To solve this issue, we created a mask such that only the table/workspace would be considered (and the robot's arms, even if they were in the frame, would be disregarded). The program allows the user to click and drag a rectangle on the first image to select the area that will have the rope. Anything outside this area will be ignored, when considering point clouds.
- **Rope Type:** We initially started off with a yellow rope, but the material was plastic and gave too much resistance for the robot to work with. We then changed to a cloth rope that was more flexible, but this rope was both blue and white which made our color selection less robust. The robot's arms were also white. (We overcame this later in the previous step). We then switch to a t-shirt, but we found that the robot had a hard time picking it up because it was so flat. Finally, we used a rope that was provided by Professor Allen which didn't have any of the previously mentioned problems.

Results

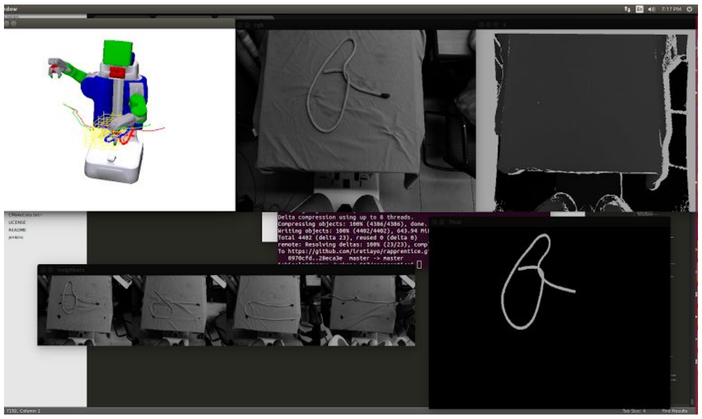
We were able to successfully record and execute the tying of a knot (see videos here). When recording the demo, we made sure to take into account the offset in depth (which we think could be fixed if we had time to calibrate the machine) and so we pressed the grippers slightly more into the foam. We also used tape to mark the setup of the rope so that the runs were consistent. We found that the robot could repeatedly tie the rope when we used these guides, and even without the tape if we use the same shape (smooth lines and curved with the ends on the right side. However, if we curve the rope, the robot does recognize the nearest neighbor but does not grab the rope itself, so it does not work. We think this might have to do with the point cloud registration or the transformation done on the grippers.



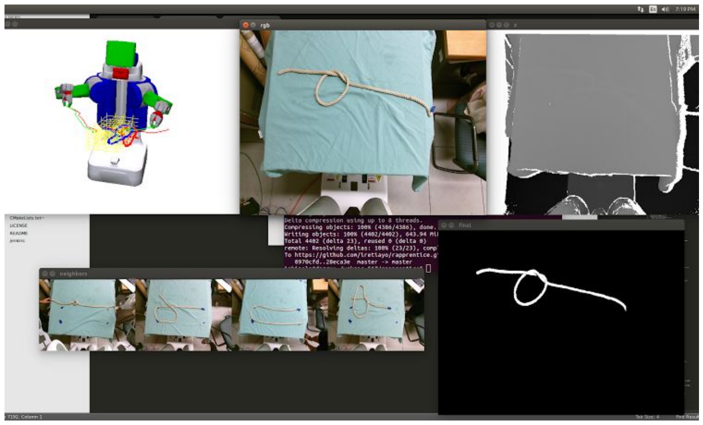
```
'current' instead of 'simple' if you sometime
Username for 'https://github.com': YanrongWo
Password for 'https://YanrongWo@github.com':
Counting objects: 4429, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4386/4386), done.
```



```
Delta compression using up to 8 threads.
Compressing objects: 100% (4386/4386), done.
Writing objects: 100% (4402/4402), 643.94 Mi
Total 4402 (delta 23), reused 0 (delta 0)
remote: Resolving deltas: 100% (23/23), comp
To https://github.com/iretiayo/rapprentice.g
   0970cfd..20eca3e  master -> master
```



```
Delta compression using up to 8 threads.
Compressing objects: 100% (4386/4386), done.
Writing objects: 100% (4402/4402), 643.94 Mi
Total 4402 (delta 23), reused 0 (delta 0)
remote: Resolving deltas: 100% (23/23), comp
To https://github.com/iretiayo/rapprentice.g
   0970cfd..20eca3e  master -> master
```



```
Delta compression using up to 8 threads.
Compressing objects: 100% (4386/4386), done.
Writing objects: 100% (4402/4402), 643.94 Mi
Total 4402 (delta 23), reused 0 (delta 0)
remote: Resolving deltas: 100% (23/23), comp
To https://github.com/iretiayo/rapprentice.g
   0970cfd..20eca3e  master -> master
```

Above are the screens that come up. The top left is the osg viewer, where the training rope points red and the testing rope points are blue. The trajectories are red and green for old and new trajectories, respectively. The program requires you to step through the simulation before it executes on the robot. In the bottom window, we see the nearest neighbors ranked. And we also see the original image from the kinect, greyed out as well as the masked result.

Github Link

Our current code base can be found on github: https://github.com/iretiayo/rapprentice

Future Work

There are many possible paths to take with this project going forward. If we had more time we would have liked to take a look at the following:

1) The LfD authors mentioned in their paper that their algorithm was not very successful with large rotations of the scene. We would have liked to rotated the images/point clouds and create transformations for these rotated scenes for each of the training "Look" stages and add these as point clouds to be used as training scenes. We imagine that this would have made the program more robust to rotations.
2) Given the time, we would have liked to finish the Kinect calibration. This would have hopefully made the grasping a bit more accurate.
3) With more accurate grasping, we would have liked to attempt the same process on clothes to allow the robot to fold clothes.
4) It would be interesting to explore the idea of having "memory" of past actions, instead of or in conjunction with the nearest neighbor method, would make the process smoother if the robot had prior knowledge of the sequence of steps.

Division of Labor

Both team members worked together in the lab on all parts of the project. We had met together to learn to operate the robot with Iretiayo. We also set up a weekly meeting to check in with him. However, we found that it was easier for both members to be aware of what is going on and more helpful for debugging if both people were present in the lab.

References

[1] Nair, Ashvin, Pieter Abbeel, Dian Chen, Phillip Isola, Pulkit Agrawal, Jitendra Malik, and Sergey Levine. "Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation." (n.d.): n. pag. Cornell University Library. Web. 30 Apr. 2017.

[2] Schulman, J., Ho, J., Lee, C., and Abbeel, P. Learning from Demonstrations Through the Use
of Non-rigid Registration. Springer International Publishing, Cham, 2016, pp. 339-354.

[3] H. Chui and A. Rangarajan, “A new point matching algorithm for non-rigid registration,” Computer Vision and Image Understanding, vol. 89, no. 2, pp. 114–141, 2003.

[4] Li, Yinxiao, Yonghao Yue, Danfei Xu, Eitan Grinspun, and Peter K. Allen. "Folding Deformable Objects Using Predictive Simulation and Trajectory Optimization." 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2015): n. pag. Web.

[5] S. Calinon, F. Guenter, and A. Billard, “On learning, representing, and generalizing a task in a humanoid robot,” Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 37, no. 2, pp. 286–298, 2007.

[6] S. Calinon, F. D’halluin, D. Caldwell, and A. Billard, “Handling of multiple constraints and motion alternatives in a robot programming by demonstration framework,” in Humanoid Robots, 2009