Generalizable Robotic Manipulation

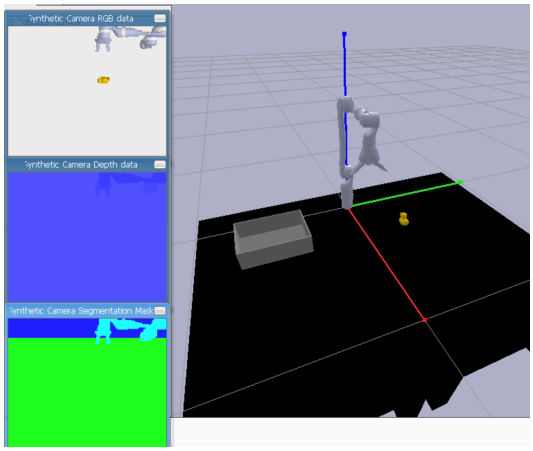
Shashwat Chakraborty

Introduction

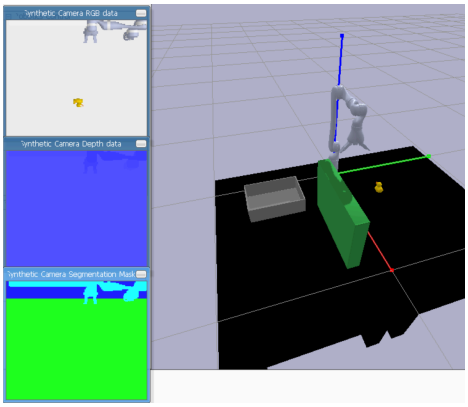
One of the most vital attributes of robots in all applications is their ability to adapt to different scenarios. This can be difficult to do, since robotic algorithms are often trained to be heavily optimized for one task, with little variation in starting conditions. However, through more generalizable machine learning methods, algorithms can be made that avoid this issue.

The following approach is made up of 2 main components: object detection and obstacle avoidance. The goal of the robotic manipulator was to bring an object from one side of the table to a box on the other side, without hitting any obstacles in the way. A variety of different objects for grasping were used for training, and a rapidly-exploring random tree algorithm was used for motion planning.

Algorithmic approach

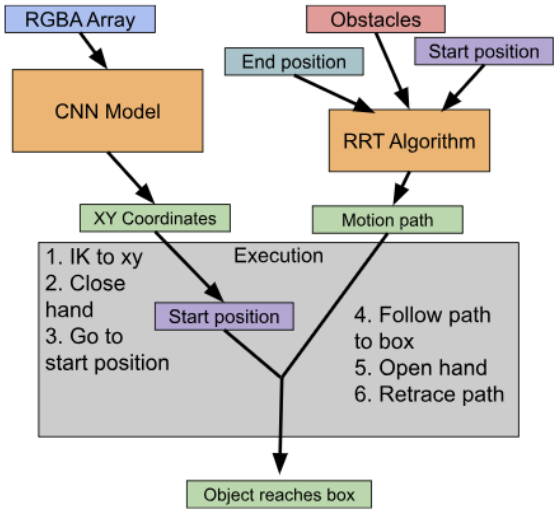


[Fig. 1] Base setup



[Fig. 2] One barrier

The base setup for the task is as shown in [Fig. 1], with the manipulator arm at the middle of the table, an object on one side, and a box on the other. [Fig. 2] shows the same setup, but with a barrier as an example of an obstacle. When started, the object and its location are randomized, and the manipulator has to deposit it in the box without hitting any of the obstacles. This process uses two separate subtasks, object detection and motion planning for avoidance.



[Fig. 3] Full model architecture

Object detection

[Fig. 4] CNN I/O

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 4, 135, 135]             148
         MaxPool2d-2           [-1, 4, 67, 67]               0
            Conv2d-3           [-1, 8, 67, 67]             296
         MaxPool2d-4           [-1, 8, 33, 33]               0
            Conv2d-5          [-1, 16, 33, 33]           1,168
         MaxPool2d-6          [-1, 16, 16, 16]               0
            Conv2d-7          [-1, 32, 16, 16]           4,640
         MaxPool2d-8            [-1, 32, 8, 8]               0
            Linear-9                 [-1, 128]         262,272
           Linear-10                   [-1, 2]             258
================================================================
Total params: 268,782
Trainable params: 268,782
Non-trainable params: 0
----------------------------------------------------------------
```

[Fig. 5] CNN model summary

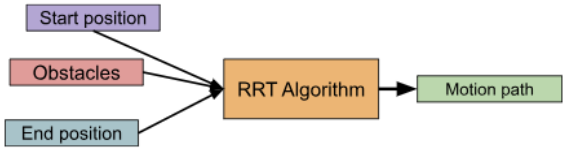
For detecting the location of objects, a convolutional neural network (CNN) was trained using RGBA data from the camera within the simulator. The model architecture is seen in [Fig. 4], where it takes in a 135x135 RGBA array, and outputs a x and y coordinate. This was done by collecting 12000 images and coordinates of randomly placed objects, from the eight shown in [Fig. 5].

[Fig. 6] Objects

The objects have a wide variety of colors, shapes, and sizes, which allows the model to be able to generalize to any object placed in the camera's field of view. After training, the model reached an average error of 2.7 mm and a max error of 7 mm, meaning that the manipulator will be able to accurately grasp the object nearly every time. This was difficult to accomplish, as most documentation on CNN models were for classification, and very few touched on regression applications such as this one.

Knowing the X and Y coordinate, and since the height of the all the objects the roughly uniform, the inverse kinematics to reach that point are calculated. After execution, the hand closes and lifts the object to a position above the table. Here is a video of the CNN in action.

Obstacle avoidance



[Fig. 7] RRT I/O

For avoiding obstacles, a rapidly-exploring random tree (RRT) algorithm is implemented. This is a fast and reliable way to find a reasonably optimal path through a space. The implementation of the algorithm was derived from the pseudocode provided in the original conception of the RRT, seen in [Fig. 8].

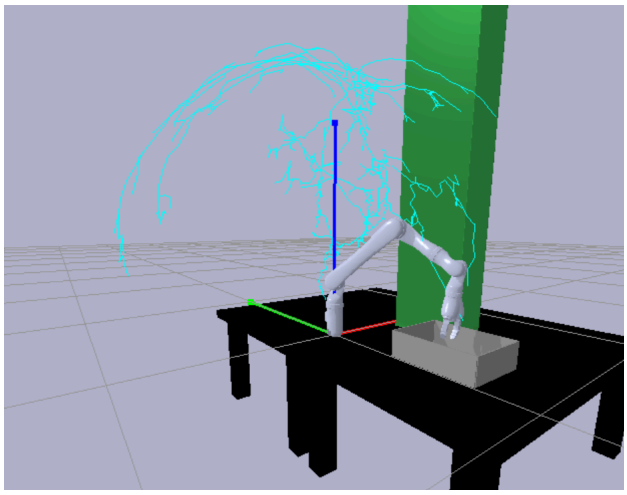
GENERATE_RRT($x_{init}, K, \Delta t$)
1 $\mathcal{T}$.init(x_{init});
2 **for** $k = 1$ **to** K **do**
3 $x_{rand} \leftarrow$ RANDOM_STATE();
4 $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$);
5 $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
6 $x_{new} \leftarrow$ NEW_STATE($x_{near}, u, \Delta t$);
7 $\mathcal{T}$.add_vertex(x_{new});
8 $\mathcal{T}$.add_edge(x_{near}, x_{new}, u);
9 Return $\mathcal{T}$

[Fig. 8] RRT Pseudocode
S. M. LaValle

First, the starting position serves as the 'root' of the 'tree'. The robot is then

randomly configured a set distance away from the current position. If that new configuration gets closer to the goal, and doesn't hit any obstacles, then a new 'branch' is added to the 'tree'. If either of those two conditions aren't met, then it repeats the process from the last point. In this way, the robot can reach its goal without hitting anything.

Using debugging lines, the 'tree' can be visualized in the simulator. Here are two examples of the RRT in action: easier barrier; harder barrier.



[Fig. 9] Example of 'tree' created

The benefit of using this algorithm rather than a predefined model is that it can adapt much easier to new scenarios that are foreign to a training dataset. A supervised learning approach to this motion planning would also be difficult, due to the tedium of collecting so many obstacle orientations and positions.

Results

Here are a few full tests of the procedure: No barrier; easier barrier; harder barrier.

Test name	**Test metric**	**Test result**
Model performance on test dataset post-training	Predicted vs. actual distance loss	2.7 mm loss on average
Manipulation of randomly placed objects present in dataset	Success rate of picking up object	96.68%
Manipulation of randomly placed objects not present in dataset	Success rate of picking up object	92.43%
RRT path, grounded obstacles	Success rate of finding path within 10k iterations	93.52%
RRT path, floating obstacles	Success rate of finding path within 10k iterations	93.26%
Full execution, random obstacles, random objects	Success rate of getting object into the box	94.55%

[Table 1] Testing of algorithm

After extensive testing of the full process, it is clear that it is an effective method for generalizing robotic motion. As seen in Table 1, there was only a 4.25% drop in accuracy once foreign objects were added, and floating obstacles only had a negligible impact on the motion planning. Overall, the algorithm works to great success.

Future work

While this procedure currently works well, there are many areas that it could improve on. Most glaringly, the time complexity of the RRT algorithm should be minimized through various means, since its worst case time complexity is exceedingly high. Second, the CNN can be further iterated on to add more functionality. Some examples of this could include being more precise, adding a second camera for height, adapting the model to find the location of one object out of multiple on the table, and more. Lastly, most of the time that the manipulator failed in its task, it was due to a gripping and orientation issue, not the models that were developed here. Thus, improvement on the mechanical side can allow for more robust testing and development of the algorithms.

Github link:
https://github.com/ArcadiusX/generalizable_robotic_manipulation

References

[1] Steven M. LaValle. "Rapidly-exploring random trees : a new tool for path planning," in *The annual research report*, 1998. Available: https://msl.cs.illinois.edu/~lavalle/papers/Lav98c.pdf

[2] P. Fischer, A. Dosovitskiy, and T. Brox, “Image Orientation Estimation with Convolutional Networks.” Accessed: Nov. 16, 2020. [Online]. Available: https://lmb.informatik.uni-freiburg.de/Publications/2015/FDB15/image_orientation.pdf