Learning-Based Collision Detection in Cluttered Environments

Arham Hundia, Christopher Larkin

I. Introduction

THE objective of this project is to develop a learning-based collision detection system capable of operating within cluttered environments. In traditional robotics, collision detection algorithms often rely on geometric computations, which can be computationally expensive. By leveraging machine learning techniques, particularly neural networks, we aim to improve collision detection efficiency, ultimately enhancing the speed of robotic systems operating in dynamic environments. The significance of this project lies in its potential to advance the capabilities of robots in real-world applications such as manufacturing, warehouse automation, robotic assistance, and more.

II. Methods

To implement the motion planning component of our project, we utilize the RRT* (Rapidly-exploring Random Tree) algorithm, a popular motion planning technique known for its ability to efficiently search high-dimensional configuration spaces. The algorithm will operate on a UR5 robotic arm, acting with a pybullet simulation environment. The total computation time and collision checking computation time will be compared between both the traditional RRT*, and RRT* with a neural net working as the collision checker. Multiple environments and start/goal configurations will be used to ensure accurate results. The general algorithm for RRT* is as follows:

1) Initialization: We begin by initializing the RRT* tree with the start configuration of the robot.

2) Expansion: Through iterative sampling and connecting, we expand the tree towards the goal configuration while ensuring collision-free paths.

3) Optimization: As the tree expands, we continuously optimize the paths by reassigning parent nodes to minimize path length and improve efficiency.

4) Termination: The process terminates after a set number of iterations have been completed, whether or not the goal has been found. Is it up to the user to ensure the number of cycles is sufficient for the given environment.

By implementing RRT*, we aim to generate collision-free paths for the robot within cluttered environments. By the use of a neural net, we aim to achieve these paths quickly and efficiently.

TABLE I
RRT* with Classical Collision Checker

Avg. Path Cost	Avg. Comp. Time (s)
3.19	31.98

III. Milestone 1 Results

As of milestone 1, we have successfully implemented RRT* within the Pybullet environment. The average path cost is 3.19, and the average time spent in the collision checker is 31.98 seconds (See Table I). Specifications include using N=1000 iterations. System specifications are as follows: 4.20GHz x 4 Intel core i7 processor, 16GB RAM, GeForce GTX 1070 Ti GPU.

Additionally, a file has been made that will generate training and testing data for the neural net. This file allows the user to specify the number of environments, and then generates n number of robot configurations for each environment. For each of these configurations, it records whether or not a collision occurs. All this data is then stored in the corresponding file location, which can be accessed by the neural net for training.

IV. Milestone 2 Results

In this milestone we trained a neural network on the data generated from milestone 1. Our neural nets framework allows it to take in inputs for the robots joint angles, and the xyz positions of the obstacles. It then outputs whether or not a collision has occurred. Dropout is used to prevent overfitting of the model, and the loss function used in cross-entropy loss. The activation function used is ReLu. As of now the accuracy of the neural network is 91%. We aim to improve this further for the third milestone.

V. Future Work

Moving forward to milestone 3, we will implement our neural network into the RRT* algorithm. We can then test this on multiple environments with various start and goal configurations. Additionally, code will be added to check whether or not the final path is truly collision free, allowing us to obtain a success rate for the neural net-RRT* motion planner. We can then compare how much time is saved from avoiding the classical collision checker approach.