COMP 580 Final Project Report: Improving Sampling-Based Planner Speed by Retrieving Previous Experiences

Arden Knoll, Thanasis Hadjidimoulas, Theodoros Tyrovouzis

December 13, 2024

1 Introduction

If robots are to autonomously navigate their environments without colliding with obstacles they must utilize efficient motion planning algorithms. One successful and heavily used group of motion planning algorithms is Sampling-Based Motion Planners (SBMPs), which handle planning by sampling a configuration space (the space of all possible robot configurations) and connecting free points in that space such that we can develop a collision-free path. With SBMPs, we are able to more efficiently plan in higher dimensional configuration spaces with complex obstacles. However, despite promises of probabilistic completeness, meaning if there exists a path the SBMP will find it in time, some spaces can be pathologically difficult for SBMPs to find a solution due to the existence of regions in the environment that are important for the solution path but unlikely to be sampled (i.e., narrow passageways) [1]. In real-time scenarios, this can cause the robot to fail to find a solution path in time and possibly crash. In contrast, we could bias our sampling such that we are more likely to sample regions that are critical to finding a solution path [2]. While computing those biased samplers in real time can be hard or impossible, generated a large corpus of them in simulation and training a model to compute a representation of them is much simpler. **By learning to detect these critical regions and sampling from them, we will enable a robot to find solution paths much more quickly than with uniform sampling, ensuring it can operate in real time.**

In this work, we train a neural network to detect critical regions represented as a low-dimensional embedding such that two scenes with similar critical regions have similar embeddings by cosine similarity and two scenes with non-overlapping critical regions have dissimilar embeddings. We then use LSH (SimHash) to store and retrieve similar environments and their critical regions so that we can bias our sampling and plan rapidly in novel environments to avoid crashing. Our specific environment consists of a point robot navigating

through a set of narrowly spaced circular obstacles meant to simulate a quadcopter flying through a forest. The goal is to fly continuously and quickly in a user-provided direction without crashing into any of the trees. As the quadcopter gains planning experience, it will store its experience in the LSH table so that when similar scenarios arise it can retrieve those experience and improve its planning speed. As such, our robot has lifelong learning capabilities that will allow it to continue to generalize to new forest scenes.

2 Methods

This work consists of three parts: the environment setup and data generation, learning similarity representations, and using our embeddings to store and retrieve relevant experiences.

2.1 Environment and Data Generation

The environment is meant to simulate a drone flying through a simple but dense forest, similar to monoculture forests used in the logging industry. As such, we take on a top-down view of the scene such that the drone is a point robot and the tree trunks are circular obstacles scattered around the drones current position. As the drone moves, obstacles come in and out of frame with no two obstacles touching each other (trees cannot grow on top of each other). The goal is for the drone to maneuver autonomously through the forest as quickly as possible for as long as possible without collisions. This problem is made difficult by the fact that the trees take up a large proportion of space and are tightly spaced around each other, forcing the drone to find narrow passageways between obstacles to continue. As discussed above, this implies that while we ensured that a solution path exists, uniform sampling could take a long time to find a solution.

In order to use SimHash to retrieve similar environments, we need to have an embedding that represents what we care about in each environment (critical regions) such that similar environments are close in the embedding space. To this end, we developed a solution for finding critical regions in our environment. First we draw all lines connecting the center of one obstacle to the center of another and find the points on along each line that intersect with the edge of the circle. Then we determine a critical region exists if the distance between the edge points along the line is less than 1.0 and define the region's center as the midpoint between these edge points. Finally, the minor axis of the ellipse is set as the distance between the edge points and the major axis is set to be the average diameter of the two circles. From there we can consider each critical region as a gaussian with its standard deviation contour defined by the ellipse. We created binary masks consisting of all the ellipses whose centers were within 60 degrees of the goal direction and saved these masks along with the original environment image and goal direction as data. We collected a total of ten thousand different (environment, goal direction, mask) groupings, each with an entirely different set of obstacles and image size 64x64.

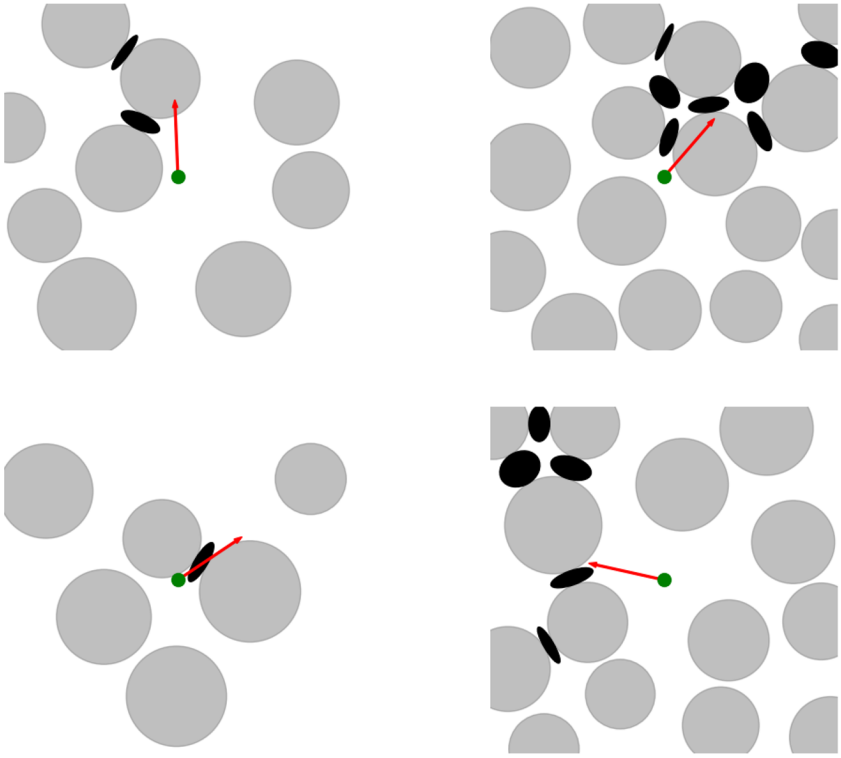

Figure 1: Four example environments, goal directions, and their respective critical regions. The gray circles are the obstacles, the green dot is the drone, the red arrow shows the goal direction and the black ellipses are the critical regions.

2.2 Learning Similarity Representations

We used a Siamese network to learn our embeddings such that the cosine similarity between two embeddings matched the value returned from our own similarity function. The cosine similarity was chosen to work nicely with SimHash which determines a hash value based on random projections in the embedding space and ultimately estimates the cosine similarity between two embeddings.

The network input is a 64x64 image representing the top down view of the environment and a vector representing the goal direction. We use a simple CNN architecture consisting of convolutions with strides that progressively lower the image dimensions, while increasing the channel count. Finally, we compute a dense linear layer and output a 16 dimensional embedding.

To determine similarity between two sets of critical regions, we first took the intersection over union of the ellipses masks. However, instead of dividing by the union of the ellipses, we divided only by whichever set of ellipses was smaller so that we obtained more highly similar pairs. Finally, since this score

maps between zero and one we scaled the results such that complete intersection mapped to +1 and no intersection mapped to -1. The direction was taken care of implicitly by the set of ellipses provided in the data, as they were originally filtered by the goal direction when the data was generated.

2.3 Storage and Retrieval

To efficiently store and retrieve environments, we implemented a *Locality-Sensitive Hashing (LSH)* approach using random projection-based hashing. This process allows us to map high-dimensional embeddings into a lower-dimensional hash space, ensuring that similar embeddings (representing similar environments) are likely to have similar hash values[3].

2.3.1 Embedding Storage

- **Embedding Network:** We used a trained neural network to extract embeddings for each environment. These embeddings capture the relationships and critical regions necessary for efficient motion planning.
- **Hashing Mechanism:** For each embedding, a hash vector is generated using random projection matrices. The process involves:
  1. Multiplying the embedding with randomly initialized projection matrices to project it into a lower-dimensional space.
  2. Computing the signs of the projection results to create a binary hash string that encodes the embedding.
  3. Converting the binary hash string into a table index for efficient storage and retrieval in the hash table.
- **Database Construction:** The hash table stores embeddings and associated data (e.g., environment identifiers) at their corresponding indices. This enables efficient lookup of similar embeddings during retrieval.

2.3.2 Retrieval Process

- **Querying with New Embeddings:** For a new environment, the embedding is computed using the same encoder network. The corresponding hash vector and table index are generated.
- **Similarity Search:** The database is queried at the computed index. Matching embeddings from the database are compared to the query embedding using cosine similarity. The most similar embeddings are retrieved and ranked.

3 Results

We conduct an experiment by generating 10000 images and gaussian distributions used for training. After splitting the dataset into 9000 training images and 1000 test images, we train our embedding network using an embedding dimension of 16. We can then insert the trained embeddings of the training set into the LSH database. For SimHash we use 8 projections with 100 table elements.

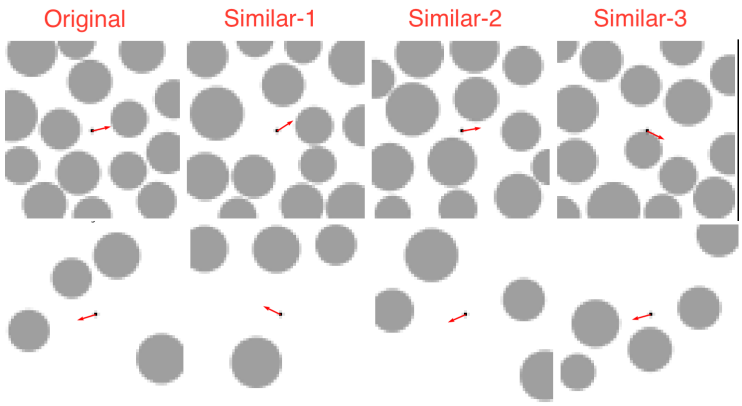

Figure 2: top-3 similar environments

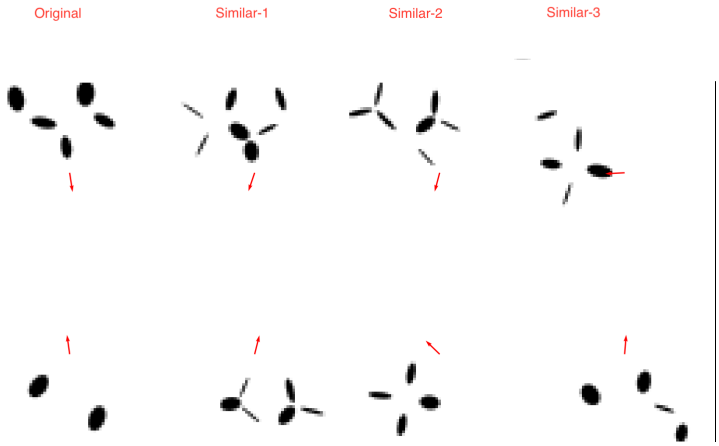

Figure 3: top-3 similar gaussian mixtures masks

Figure 2 and 3 show the top 3 matches of the LSH lookup. We observe that our embedding was indeed able to capture some notion of similarity between planning environments. When given dense images with many obstacles, environments with dense obstacles were also returned, and vice-versa for sparser

environments. Although not perfect, there is also considerable amount of overlap between the query embedding and the returned gaussian distributions. This informs us that the embedding is able to capture the critical regions that are require special caution during sampling and that LSH could be used to find distributions that can aid sampling towards those regions.

4 Conclusion

We presented a methodology for efficiently retrieving biased sampling distributions given only an image as input. Although as a proof of concept we use simplistic 2D environments, conceptually this method can be applied to any scenario where it is desirable to compute fast biased samplers and a simulation is available. The only requirement in order to benefit from our method is that it is easier to compute the biased sampling gaussian mixture in simulation rather that during real time perception. This is most often the case since when simulating we know the precise position and extent of obstacles.

As future work, we would like to test our approach in more complex planning scenarios such as real drone images and measure the performance when planning compared to using uniform distributions. There are plenty of photorealistic simulations that can be used to compute biased distributions offline[4]. Another application could be for continual learning. While an embedding has already been trained, we can keep adding biased samplers computed using traditional methods and add them to our LSH data structure. As the robot encounters similar environments, the probability that it has already planned on a similar distribution increases resulting in faster and faster performance due to query hits in the LSH retrieval.

References

[1] D. Hsu, J.-C. Latombe, and R. Motwani. "Path planning in expansive configuration spaces". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 3. 1997, 2719–2726 vol.3. DOI: 10.1109/ROBOT.1997.619371.

[2] B. Ichter, J. Harrison, and M. Pavone. "Learning Sampling Distributions for Robot Motion Planning". In: *CoRR* abs/1709.05448 (2017).

[3] M. S. Charikar. "Similarity estimation techniques from rounding algorithms". In: *Proceedings of the 34th annual ACM symposium on Theory of computing (STOC)*. ACM. 2002, pp. 380–388.

[4] S. Shah, D. Dey, C. Lovett, and A. Kapoor. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles". In: *Field and Service Robotics*. 2017.