Implementation of Collision Avoidance Through Neural Implicit Probabilistic Scenes

Madhav Rawal
madhavr@umich.edu

I. Introduction

Collision Avoidance Through Neural Implicit Probabilistic Scenes or CATNIPS [1] is a framework for robot trajectory planning through a NERF scene represented as a probabilistic map. This is done by establishing a mathematical transformation of a NERF scene to a Poisson Point Process. CATNIPS proposed an equivalency of a NERF scene and a Poisson Point distribution and this distribution was then used to calculate robot collision probabilities in the scene. This is obtained by convolving a kernel representing the robot geometry with the voxelized representation of densities of the NERF scene. The resulting map is called the Probabilistically Unsafe Robot Region or the PURR. The trajectory planner uses this PURR thresholded with a user-defined probability value to ensure a collision free trajectory between waypoints.

II. Methodology

This implementation is done on python using libraries such as pytorch and numpy. We also assume that a pre-trained NERF scene representation is given. The CATNIPS algorithm involves two major parts: Generation of PURR and Trajectory Planner.

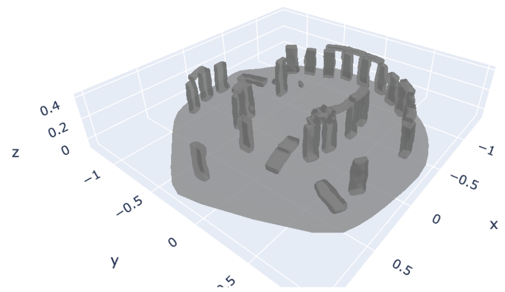

Fig. 1: Stonehenge NERF Render

A. Generation of PURR

A Probabilistically Unsafe Robot Region or PURR is a voxelized representation of the world in R3. This represents the probability of a collision in the scene. If a robot occupies a free voxel in the map then it can be assumed that it is not in collision with any surroundings. To obtain the PURR, the occupied space on the voxel map is "inflated" keeping into account the geometry of the robot and the chance of collision of a robot given it is at an arbitrary location in the map. The generation of PURR from a NERF is a stepwise process described below:

1) Cell Intensity Grid: The cell intensity for a voxel can be calculated by integrating the density function and averaging over the voxel space. Since the density function is described as a neural network, the integral is not trivial to compute. Therefore, to compute an approximation of this integral, we use a trilinear interpolation scheme. Given a user-defined dimension of a voxel and the world, we first create a voxelized grid as a torch tensor of dimension (HxWxD) with the origin lying in the top left corner. The coordinate of a voxel corresponds to the top left front corner. We then evaluate NERF densities at vertices of each voxel. However, the nerf model needs coordinates between -2 to 2 so we calculate the local coordinates of each vertex of a voxel in the grid and scale it to a value between -2 and 2. This array of coordinates shaped ((HxWxDx8)x3) is then sent to the nerf model to get the resulting densities at these points. To get the density of the voxel, we need to fit a function considering the density values at the corners using trilinear interpolation. According to the paper, the equation can be described as follows:

$$\hat{\rho}(x,y,z) = c_1+c_2x+c_3y+c_4z+c_5xy+c_6yz+c_7xz+c_8xyz \tag{1}$$

The coefficients of the equations can be obtained by constructing a least squares problem $A_{1:8}c_{1:8} = \rho_{1:8}$. After obtaining the coefficients, the closed form solution of the integral can be found by using the equation mentioned in the appendix A of the CATNIPS paper. The resulting integral solution for all cells gives the cell intensity grid.

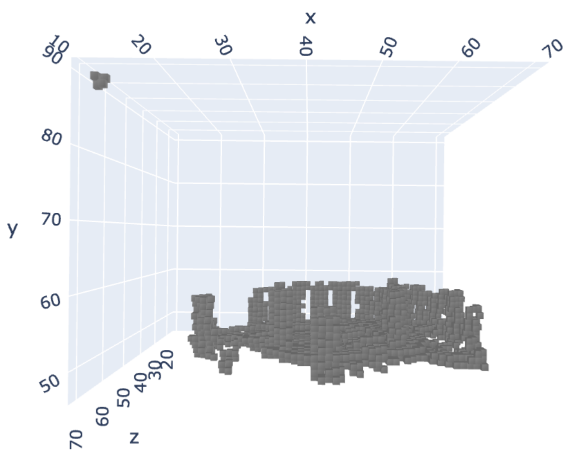

Fig. 2: Cell Intensity Grid

2) Robot Kernel: A robot kernel is a mask of voxels that are considered in the collision computation when the robot is in the vicinity of any voxel in the map. In this implementation, the robot kernel is calculated by taking a bounding sphere around the robot of the largest dimension of the robot in any direction. This sphere is then voxelized in a kernel of an arbitrary dimension. This kernel can be convolved with the voxelized map by assigning the kernel as the weights of a 3D pytorch convolution layer.

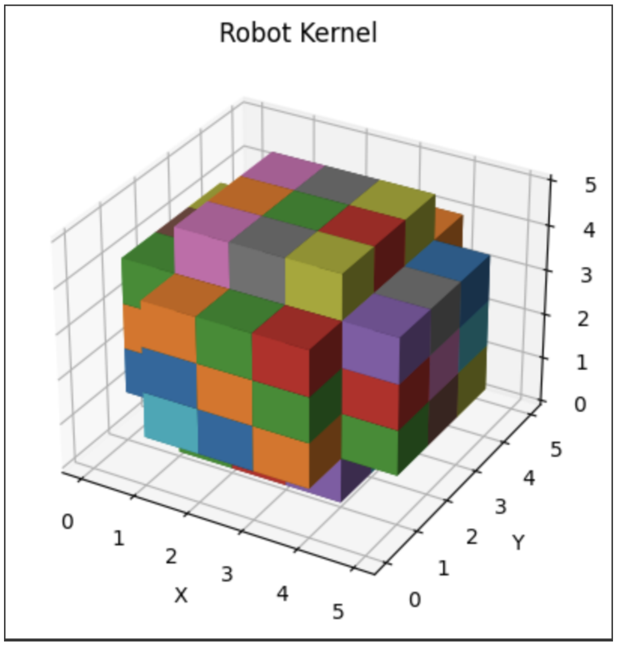

Fig. 3: Robot Kernel

3) Robot Intensity Grid and PURR: To get the robot intensity grid, we need to convolve the robot kernel with the cell intensity grid. Finally, we get the PURR by thresholding the map with a user defined threshold.

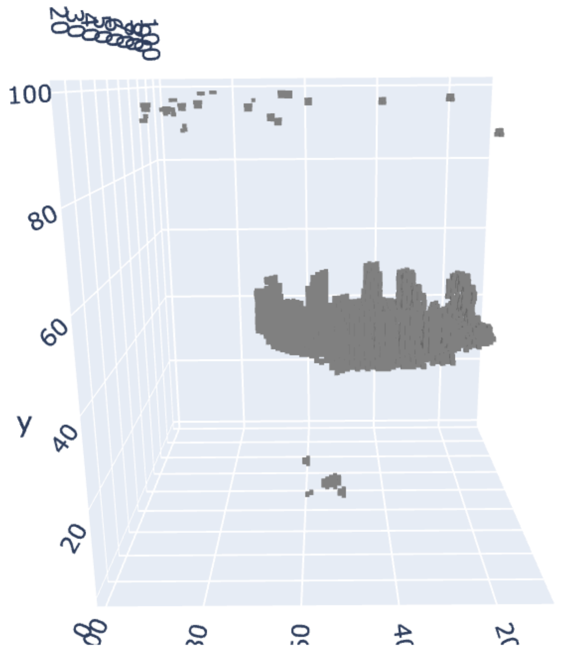

Fig. 4: PURR

B. Trajectory planner

The goal for trajectory generation is to get a trajectory that satisfies the chance constraint of collision at all points along the trajectory as opposed to only a set of knot points. This trajectory also needs to be dynamically feasible. To get an initial estimate of a path in the PURR, we run a vanilla a-star algorithm with movement possible only along the axes. This gives us an optimal path between waypoints that exists in the free space of the PURR and hence is collision free. However, this path may not always be dynamically feasible. The following sections describe the generation of a smooth, dynamically feasible and probabilistically safe path.

1) Bounding box generation: We seek to generate a dynamically feasible path, we inflate a tube around the astar path that is large enough to not constrain the optimization problem and be in the free space of the PURR. To generate these tubes, we first split the astar path into straight line segments along one of the axes. Then for each line, we iterate over each pixel and find an obstacle closest to it in the normal directions of the line. The minimum distance in each direction is multiplied by the corresponding normal direction and added to the first and last voxel coordinates to get the coordinates of the corners of the bounding box of the line. We then calculate the dimensions of the box with the minimum dimension being one voxel length to handle edge cases. We then add these dimensions to the top left corner of the bounding box. We find the top left corner by finding the corner with the least distance from the origin at the top left corner of the grid. This computation gives us the coordinates of the bounding box in local coordinates, where the corners of the bounding box are the outermost vertices of the voxels.

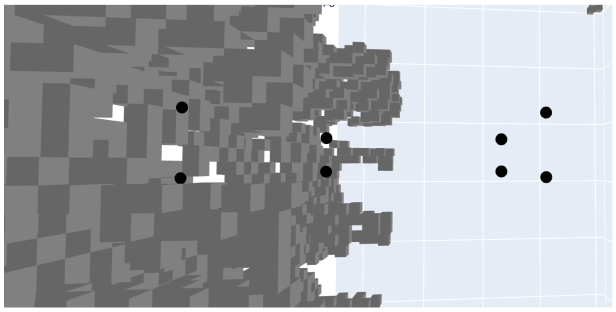
Fig. 5: Example of a Bounding box lying in the complement of the PURR

2) *Quadratic program formation:* We now seek to generate a smooth Bezier curve that connects the start and end waypoints and lies completely in the bounding box. Since the bounding boxes lie in the complement of the PURR, any path lying inside the bounding boxes is probabilistically safe by default. The bezier curve can be represented in terms of matrices as shown in the equation of a degree 3 bezier curve below:

$$B(t) = \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}. \tag{2}$$

Therefore the bezier curve can be expressed as a linear function of the form:$B(t)$ where t is the curve parameter. The optimization problem described in the paper is of the form:

$$J(s^1, \dots, s^L) = \sum_{i=1}^{L} \Bigg(\int_0^1 \left\| \beta^{(d)}(t) s^i \right\|^2 dt + \sum_{k=0}^{N-1} \left\| s_k^i - s_{k+1}^i \right\|^2 \Bigg) \tag{3}$$

where $\beta^{(d)}(t)$ is the derivative of order d of the bezier curve and the s_k is a concatenated vector of control points. The constraints include the continuity constraint where the last point of a bezier curve in one box is the starting point of the curve in the next box and the start and end waypoint correspond to the start and end points of the total bezier curve. Additionally, all these constraints must lie in the respective bounding boxes. These can be expressed as follows:

$$\begin{aligned} \min_{s^1,\dots,s^L} \quad & J(s^1, \dots, s^L) \\ \text{s.t.} \quad & s_j^i \in \mathbb{B}^i, & \forall i \le L, j \le N \\ & \beta^{(d)}(1) s^i = \beta^{(d)}(0) s^{i+1}, & \forall i < L, d \le D \\ & \beta^{(0)}(s^1) = p_0, \\ & \beta^{(1)}(s^L) = p_f. \end{aligned} \tag{4}$$

The integral term is calculated using sympy and then a jacobian is computed of the integral solution with respect to a concatenated vector of all s_k (s_{vec}). The constraints are computed symbolically first and then jacobians are calculated with respect to the s_{vec}. All jacobians are concatenated to make a huge matrix that can be multiplied to the s_{vec} to minimize the quadratic problem. The quadratic problem solver cvxopt is used to solve the optimization problem. The result is an optimized set of control points that link the start and the end point

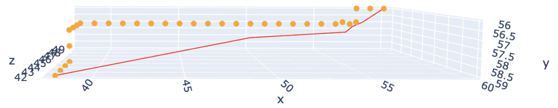

Fig. 6: The optimized bezier curve(red) as opposed to the dynamically unfeasible astar path(orange)

III. Results and Discussion

Figure 1 shows the nerf model used in this implementation. It is a widely used nerf dataset that is also used in the main paper. Figure 2 shows the cell intensity grid with a voxelized representation of the stonehenge model. The figure 4 shows the PURR generated after convolving the robot kernel with the cell intensity grid. Notice how the walls are thickend as compared to the cell intensity grid. This comes with the consideration of the robot geometry and collision chance constraint consideration. Figure 6 shows the astar path as a collection of voxels; it can be seen that it is not dynamically feasible as it involves sharp turns as opposed to the path in red which is a smoother bezier curve approximation. We can also see in figure 7 how the final path is generated through the PURR.

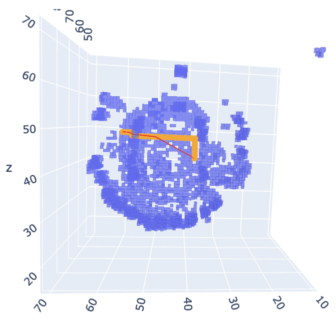

Fig. 7: Path Through PURR. Here the orange path is the result of the a star algorithm and the red is the optimized bezier curve

IV. Conclusion

In concluding the implementation of CATNIPS, there is need for efficiency enhancements. The use of sympy for integral computation has shown time constraints, urging a search for more efficient alternatives. Future work should focus on code studies, evaluating compatibility with various NERF models and multiple waypoints. Additionally, fine-tuning hyperparameters like collision thresholds and bounding box overlaps is crucial for optimal performance. This groundwork provides a solid foundation for further refinements, ensuring CATNIPS evolves into a more versatile and efficient tool for dynamic robot trajectory planning in diverse environments.

References

[1] T. Chen, P. Culbertson, and M. Schwager, "Catnips: Collision avoidance through neural implicit probabilistic scenes," 2023.