NeRF Notes

Samuel Kinstlinger

May 2025

* In real-world environments, a robot must avoid obstacles, manipulate objects, and navigate efficiently. All these tasks depend on spatial awareness—the robot needs to know what is where, including semantic and metric data. In this situation, 2D images are not sufficient: 3D structure is essential. For example, a mobile robot needs to know the 3D shape of a room to plan collision-free paths. Additionally, a manipulation robot needs 3D object geometry to grasp correctly.
* However, robots often do not perceive their entire environment at once. This can be due to sensor range limitations, inherently directional aspects of sensors, self-occlusion, or external occlusions in environments that block light of sight to other areas. The robot only has information collected by sensors, meaning that to get a full understanding, robots must often move and perceive from a variety of viewpoints and angles. It must then use the totality of information it senses to construct a representation of the relevant aspects of an environment. This representation is known as a **map**.
* Note that even though a 3D point is infinitesimally small, its observed color can still depend on the viewing direction due to material and lighting effects like reflection, transparency, subsurface scattering, and shading. Some materials exhibit transparency, refraction, or subsurface scattering. This shows the need to not just store the occupancy or color of a point, but also be able to gauge direction-independent information. This means that the map cannot simply take into account a point but also the direction from which it is viewed to output its occupancy and any color aspects for semantic reasoning or vision-based SLAM.
* In traditional mapping, a robot would align its sensor data from consecutive scans to a single, coherent, discrete map. However there are some issues with this:
  + Navigational Efficiency: In many tasks (navigation, exploration, manipulation), the robot must evaluate many possible actions—but can only move one way at a time. If it had to physically explore each one, it would be incredibly slow and inefficient. In some situations, it may even be dangerous. Reasoning about occluded areas allows for planning ahead, mental simulation, and information gain estimation. It enables reasoning about what’s likely to be behind walls, around corners, inside containers, etc.
  + Memory Efficiency: Robots often operate with limited RAM, GPU memory, and storage capacity. They also need real-time processing, which constrains the data resolution and model size. Storing every raw image, point cloud, or voxel grid is expensive as high-resolution data accumulates fast. Data is also often redundant as adjacent frames overlap heavily and nearby data points indicate the same surface. Due to inherent geometric relationships in the world, surfaces are locally smooth, planar, or follow predictable textures. Points in the environment are simply samples of these surfaces. However, functions are more memory efficient as they have infinite resolution and merely require the storage of parameters as opposed to many points. By knowing the surfaces, we can generate points on-demand. Similarly, by knowing the points, we can approximate the surfaces with piecewise functions and use these to generate points on demand. A function approximator can complete missing regions, and infer occluded geometry and semantic information.
  + Integration with Learning-Based Systems: Function approximators are differentiable, making them compatible with reinforcement learning and gradient based planning.
* This shows the necessity for a machine learning approach in which a neural network learns a function mapping a point and viewing direction to the color and occupancy/opaqueness. The robot can collect information from LiDAR/Camera throughout its episodes, train the network afterwards, then use the network for inference in future endeavors. This network is known as a **Neural Radiance Field** (**NeRF**).
* NeRF is represented as follows: F(x, d) = (c, σ)
  + Inputs:
    - x ∈ R3 - A point in 3D space—represented by its x, y, z coordinates. This tells the network where in space we’re evaluating. NeRF learns a function over the entire 3D volume, and x is the input to that function.
    - d ∈ R3 – A unit vector (direction of length 1) indicating the camera’s viewing direction as it looks at point x.
  + Outputs:
    - c ∈ R3 - The red, green, and blue color values emitted from point x in direction d. This is the appearance of the point from the camera’s perspective. It's what the renderer ultimately uses to construct the final image.
    - σ ∈ R>=0 - A scalar representing how much material exists at point x—specifically, how likely it is for light to be absorbed or emitted there. σ = 0: The point is completely transparent (like air or empty space). High σ: The point is highly opaque or solid (like the surface of a wall). It determines how much a point along a ray contributes to the final pixel color. This is crucial for volume rendering. NeRF doesn’t directly generate 3D surfaces—instead, it builds them up implicitly by assigning high σ values to points that make up visible surfaces. NeRF stores volume density —not binary occupancy—because it’s designed for differentiable, continuous, realistic view synthesis.

Get Pixel Value:

* We need a function that tells us how much color comes from each point along the ray, and how much of that light actually reaches the camera.
* A single point in continuous 3D space has zero volume. So, it cannot by itself block light or reflect it — because a ray passing through space would never “hit” an infinitesimal point. Instead, only a finite stretch of space (i.e., a region with measurable thickness) can interact with light.
* When a ray passes through space each infinitesimal segment contributes a small chance. To say anything meaningful, we must consider a small interval of space the ray passes through as only non-infinitesimally small physical segments can reflect light. For a ray passing through this length, each point makes a contribution to how much light the segment absorbs, with some points contributing more than others. A segment of some material will absorb more light if thicker. Similarly, a segment of the same thickness can absorb more or less light depending on the density and material.
* For a given point, its opaqueness is independent of the opaqueness of points around it. To construct a scene, we need to output information about each point in the scene. For a given point, for the reasons shown above, we must provide the probability density representing the likelihood that the point absorbs/scatters light per unit distance. To get the probability that a segment absorbs light, we must integrate to get the total light accumulated.

For Probabilistic LiDAR:

* Uses:
  + Path Planning: Check path occupancy
  + Localization and Loop Closure: Render a synthetic LiDAR scan from a candidate pose using the trained NeRF, compare it to the real scan from the robot’s LiDAR sensor, and use methods such as scan matching to detect loop closures and refine pose estimates.
  + Simulation and Policy Training: A realistic, efficient simulation of the environment to train vision or navigation policies, test grasping or manipulation strategies, and simulate sensor behavior.
* Traditional NeRFs assume a single surface along each ray, which can lead to artifacts when dealing with LiDAR data that often includes multiple returns due to semi-transparent surfaces or complex geometries. The probabilistic formulation models the likelihood of a return at each point along a ray, allowing the network to represent multiple potential surfaces. This model learns a probability distribution over 3D space: for each 3D point, it predicts how likely a LiDAR sensor is to return from that location. It builds an implicit, differentiable map of the environment. It handles multiple returns, transparency, soft surfaces, and noisy data, better than traditional occupancy grids or point clouds.
* Instead of integrating optical density, the loss function integrates over the probability distribution of returns along each ray. This approach enables the network to learn multiple peaks in the density function, corresponding to different possible return depths, and mitigates the issue of "phantom surfaces" that can arise from conflicting measurements.
* Inputs: A 3D point in the environment which the network will evaluate. This is sometimes inferred from LiDAR origin, ray direction, and distance along the LiDAR ray. LiDAR measurements are largely view-independent, so view direction is not a relevant input.
* Outputs: σ(x): This is the predicted probability density function (PDF) value at point x. It quantifies the likelihood of a LiDAR return occurring at that specific location in space. Some models also include color.