An Efficient Probability-Based Method of Fusing Mapping Data with Neural Network Object Detection Labels

(Revised 3/11/2026)

Steven Berdak

Computer Science Department, California State University Stanislaus
1 University Circle, Turlock, CA 95350, United States of America
sberdak@csustan.edu

***Abstract*— A simple probability-based set of methods for robots to perform efficient mapping of neural network detected objects by fusing aggregated map data with neural network detections using weighted distance from stored points as a way to map intermediate points with key observations and possible optimizations.**

***Keywords*— Computer Vision, CNN, Convolutional Neural Network, EfficientNet, Midpoint Circle Algorithm, MobileNet, Neural Network, Object Detection, Probability-based, Probabilistic, Radial Basis Function, Robotics, SLAM, vSLAM, YOLO**

I. Introduction

The implementation of mapping algorithms in robotic systems is undergoing a transformation with the rise of neural networks and machine learning. Modern robotic systems commonly use SLAM (Simultaneous Localization and Mapping) which combines data streams from various sensors such as optical, LiDAR and IMU (Internal Measurement Unit) sensors to generate a map of its surroundings. These mapping capabilities have applications in overland, underwater and extraterrestrial exploration due to their ability to map regions without any prior knowledge. However, these systems do not take into account contextual clues that result from taking into account the various types (classes) of objects and their positions. The map representations store map information in various forms such as occupancy grids or points clouds, and sometimes a system may make use of multiple data formats for storing data gathered from sensors. Improved performance and decision making in robotic systems can be achieved by leveraging recent advancements in machine learning. These advancements provide novel ways to increase contextual clues by fusing sensor data with object labels generated by neural networks to achieve better decision making capabilities and more robust maps that can improve productivity for researchers and operators. The object labels generated by the neural network can be used, for example, to create a searchable database of locations of objects for further study or to provide a reasonable estimate of the number of occurrences of objects within mapped regions.

Given the increased complexity in things like mapping and the various related auxiliary tasks, maintaining good performance will become increasingly important as the size and complexity of the data increases. Efficiency can be upheld by using techniques and methodologies that are designed to be efficient from the ground up. Probability-based methods can provide many benefits by allowing inference to be made not based on any single measurement from a sensor, but based on many observations to form statistical distributions that allow good approximation and inference of things like object positions and boundaries. The sensors in a robot may take many readings in any specific zone within a region and therefore this naturally lends to using statistical methods over other methods such as in-place spatial clustering, which may need to be repeated over many iterations. It can also reduce computations by allowing approximation based on a reduced sample size by using statistical approximation to get a good overall view of the surrounding environment, especially where some degree of error is not detrimental to the overall process.

II. Core Concepts

The concepts that this paper will cover are outlined here to give a brief general understanding of what systems are necessary for building a system using the methods and techniques outlined in this paper. Note that this is not an exhaustive and modern robotic systems may incorporate a wide variety of sensors, equipment and software.

A. Simultaneous Localization and Mapping (SLAM)

Simultaneous localization and mapping is an algorithm which provides a way to blend sensor data into a common spatially mapped data structure, the purpose of which is that it can be used to determine object boundaries and locations of objects in regions in which the robot has traversed. Common SLAM implementation may use camera, LiDAR and other sensor information to build a map of the environment, although there are various issues with processing the map information that must be addressed such as drift and loop closure [1]. The map which holds stored data and/or is used for further processing contains the position of detected objects based on sensor readings and also additional meta-information about these positional values. Common types of map representations include occupancy grids, which contain cells of numerical values that represent certain states, or point clouds which store individual points rather than aggregating sensor readings over some specified resolution.

B. Convolutional Neural Networks (CNN)

Convolutional neural networks are a type of neural network that uses a specialized layer structure to achieve accurate object classification from image data gathered using optical sensors. A paper released in 2017 by Krizhevsky et al contained experimental results, stemming from an annual well-known contest, that showed a reduction of the error rate of over 10% compared to other top architectures [2]. This result is considered to be a tipping point for which machine learning began to rapidly gain popularity at scale [3].

The rise of neural network based artificial intelligence has spurred research into new methods for processing data using images. A convolutional layer network (CNN) is a type of neural networks that use specialized layers to store abstract representations of features within images using an extensive training process which uses loss and optimization to update the model [4]. A trained convolutional neural network can output object classes inferred from new images which are passed into the model via the input layer. Trained convolutional neural networks can provide bounding boxes and pixel mapping of detected objects with a relatively high degree of accuracy. It should be noted that a CNN cannot classify objects that it wasn't trained on and they are sensitive to variations in image properties like contrast, lighting and jitter.

For any given input image a CNN will output classification scores for all objects detected within the frame, though bounding boxes can be output from the model for which it is possible, although complicated, to extrapolate the exact position of the objects relative to the frame due camera distortion caused by the lens. The focal length of a lens, for example, affects both the field of view and view depth within an image which the network may be sensitive to, despite the effect not being apparent to a human observer unless the amount of distortion is high.

Selecting a model architecture is a task that is due some consideration before implementing any model within a processing pipeline. There are many pretrained models available for inference and/or as a basis for further training. These models are trained on many common classes of objects and provide out-of-the-box inference capability with little configuration required. Some of the various pretrained models that are readily-available currently include YOLO, EfficientNet and MobileNet [5]. Using pretrained models saves time and energy for developers by being able to leverage the extensive training already performed on the model which the need for complicated and extensive training of classes as a prerequisite to having object classification capabilities. In cases where objects requiring detection are not already present within the model's set of pretrained classes these models may be insufficient. However, it is possible to augment the model's training with additional classes. The process can be intensive, often requiring thousands of training examples. Regardless, pretrained models can still save time in

the training process by reducing the time required for convergence in detecting new objects by leveraging its existing knowledge about objects stored within its internal state..

C. Internal Measurement Unit (IMU)

It is important to keep track of the orientation of the robot and also forces experienced due to things like linear and rotational acceleration. The IMU of a robot provides this data using a combination of accelerometers and gyroscopes which gives the robot a way to detect its position and rotation, known as the *pose* of the robot, within a given map. This pose information is determined from physical forces that act on the sensors which is then translated into positional and rotational values, sometimes referred to as Euler axes [6]. From the forces on these sensors the roll, pitch and yaw of the robot can be determined. Certain software implementations may use quaternion format, although tools exist that can translate from one method to the other.

D. Image-based sensors

The use of images created from optical-based sensors to infer information about its surroundings is useful in environmental mapping by robots. Images themselves contain a lot of information and contextual clues within the contrast between different objects, as they interact with the sensor in different ways. Machine-learning based CNNs take matrix or vector representations of images and convert them into object classes. Consequently image-based sensors enable the use of machine-learning based methods for mapping a traversable environment [7]. Stereo cameras make it possible to infer depth information by taking images at different angles, unlike monocular cameras.

III. Fusing Aggregate Data

The fusion of data from existing methods such as SLAM algorithms, sensors within the IMU, and from artificial intelligence is a logical step in enhancing the capabilities of robots. Data can be fused from both raw and refined data for downstream processes [8]. Object classifications which are output from a trained CNN model give the robot a way to see its surroundings with additional context by taking into account classification labels generated from images captured by image sensors and passed into the network. These output labels can be combined with positional information of objects to store the position of detected objects. These technologies enable new ways to make downstream decisions during the mapping process.

A robot could conceivably be designed to locate a certain tree species, for example, within a given environment and to autonomously navigate to regions in which tree density is outside a certain level of variance to take samples required for study. This example illustrates how these systems can enable new capabilities for robots.

IV. Mapping Object Positions

Positioning objects within a stored map requires careful management of detected objects and their placement within a map. Conceivably the placement of detected objects should be resistant to noise, or in other words, the system should be able to correct itself where an object was placed at an incorrect location or mislabeled. Optimally the system would be able to remove points from locations where objects are no longer present, have been moved to other locations, or are the result of incorrect sensor readings. Thus in certain respects an object's location within a map is based on statistical samples and so mapping environments using sensors naturally lends itself to statistical thinking and statistical treatment. These readings can enable the use of bayesian or other statistical methods to generate probabilities of object positions and classes. Simplistically, the probability that an object is at a certain position given that it was detected at a given position can be inferred over overlapping sensor scans, and can be updated with subsequent scans.

Note that due to the complexity of training of a CNN it would be difficult to assign proper classification of, for example, a backpack belonging to a specific person, due to the extensive training that would be required for the CNN to ‘memorize’ the details between backpacks that can be inferred such as the brand, color and distinctive markings. Fine-grain categorization of object instances

requires the basic functionality of CNNs to be extended and has been researched to some degree [9].

A. Reconciling Differences in Distances From Sensors

The position of objects detected using camera sensors and other sensors, such as LiDAR, must undergo specific treatment in order for the readings from one to be congruent with the other in terms of spatial reference with respect to the camera frame and the real-world object positions. Specifically, camera images require the use of a calibration matrix which provides a way to apply linear transformations that account for distortion caused by the lens [10]. Stereo cameras can augment images with additional depth values that can be used to accurately place objects. Cameras that do not provide this depth information can use software methods to provide an estimate of the depth of objects within an image frame, though these estimates may not be as reliable as those using other methods.

B. Simple Software Method for Determining Depth of Detected Objects

It is possible to use pose information combined with a stored map generated from sensor readings to determine the position of objects. Conceptually a beam extending from the front of a robot which reports the position of the first object it intersects can be combined with bounding boxes located in the central region of an image processed by the CNN to correlate a position with an object label. The requirement for the object to be centered relative to the map and camera frame is due to the fact that regardless of the lens used, the center point perspective of an object relative to the frame will be a good approximation of position due to lens symmetry.

This method would only require taking into account the pose of the robot, along with the current map state, to assign object labels to points by walking along a straight-line path extending from the robot's forward position to the object. For brevity this paper will consider the case of a 2-dimensional map, although these ideas can be extended to 3-dimensional space by considering the pitch ('upward' or 'downward' angle) of the robot. Note that this method presupposes that the robot is placed within an occupancy grid, but if a point cloud were used the system could instead rely on euclidean distances to points within some distance of the terminal point, $\hat{p}_n$, at each step, n, of a walk. The resolution (cell width) of each grid cell, labeled as R, is also necessary to adjust the step-size accordingly. The appropriate step size will not skip cells, although certain cells may be referenced twice due to the hypotenuse formed by the triangle in the bounded circular region being smaller than the bounds of the squares that are being traversed. The formula to walk along the grid cells is as follows:

$$\hat{p}_n = n * \left(\left\lfloor \frac{cos(\theta)}{R} \right\rfloor \hat{x} + \left\lfloor \frac{sin(\theta)}{R} \right\rfloor \hat{y} \right)$$

Using this method, finding an object that obstructs the beam becomes a simple process of checking, at each terminal point within each step, whether an object is contained within the cell. If an object is detected then this will be the terminal point for the beam and the goal has been achieved of obtaining the distance for which to place the object.

The next step in this process is to take the cell at this terminal point and combine it with a label from a bounding box that encompasses the central position of an image, and assign it as a *post* which will store the label along with the position. It is important to ensure that the image is taken from the same pose frame as that used to generate the virtual beam through the map. Note that a CNN often provides many prediction scores for a single object, so the top k predictions could be stored instead and the aggregate overall predictions among many posts can be used to infer the object's true label, though that is beyond the scope of this paper.

This method provides a very simple way to project a beam within a map that corresponds to a point from an image generated by a camera sensor. However, this method could be extended using a calibration matrix to project multiple beams through the grid, each of which could be treated independently in assigning object labels to points and thereby making the overall process of assigning posts faster. This could greatly decrease the overall

time required for assigning object labels in large environments.

C. The Post Method for Placing Objects in a Map

Taking collections of posts placed within a map, each having its own label, it is logical to try to define the boundaries for which individual objects inhabit space on a given map. Because the objects are placed based on detected labels correlated with the object's perimeter points, additional processing will be required to map collections of related posts. In order to infer which points on the map belong to which objects, such as those which lie within the interior of the object, it will be necessary to calculate the probable label of a class given nearby posts. The perimeter of objects may have holes due to an object perimeter not being completely mapped, and therefore the total perimeter of the object must be inferred from nearby posts with some assumptions.

Firstly it can be assumed that if two posts with the same label are close to one another then there is some likelihood that they are part of the same object. Secondly, assume that within the perimeter of an object, at least some of the space near the post of a given class will belong to the object represented by the post label, the likelihood which increases based on the distance to the posts. It can also be assumed that the likelihood of a point belonging to a class increases proportionally in relation to the number of posts it is close to. In the case of multiple posts with multiple object types a simplistic assumption can be made that the class with the strongest influence is the class that a point should be assigned to.

With these assumptions in mind it is necessary to calculate a weighted distance from each post, which also takes into account the number of nearby posts. Two methods for calculating the weighted object distance which also account for the number of posts of each class are as follows:

1. Non-linearized average euclidean fractional distance nth root formula

This method relies on the fact that a fractional amount of influence, I, can be calculated by considering the distance away from a point over some maximum distance, ensuring to offset the point by the map center. This formula extends from the simple idea of a decaying probability based on linear distance from the post to some defined maximum distance D:

$$I(d) = 1 - \frac{d}{D}$$

This value can then be divided by the number of posts (cardinality) in the set of posts, P, to get the average weighted probability which will be bounded by zero and one. Non-linearity can then be applied by increasing the score using a power function which weights the average sum accordingly based on the number of posts, in which an increase in the number of posts increases the score at each probability in the range of I:

$$I_e = \left(\frac{1}{|P| \cdot D} \sum_{P_O}^{P_n} \left[D - \sqrt{(x - c_x)^2 + (y - c_y)^2} \right] \right)^{\frac{1}{|P|}}$$

To gain a better understanding of this formula, consider the influence of a post based on distance, where a point exactly at the center of the post has the maximum value of 1. The influence of the post decreases from the center point to the edge of a circle specified by the max distance, or the radius, of the area of influence, and decreases linearly before applying the exponential term. If multiple posts are nearby then taking the nth root will provide a curve that is always greater than the case of a single post (except possibly at the extreme values of every curve under consideration for various values of $1/|P|$), as shown in Figure 1. If the cell is within the influence of only a single post then the calculation will be linear when taking the first root, resulting in the influence on the cell from the post being linear as shown in Figure 2(a). If the cell is within the area of influence of multiple posts then taking the nth root will provide a curve that is always greater than the case of a single post representing the increased influence from being in range of multiple posts.

The influence at a given distance from a point to a single post at $|P| = 5$ is shown in Figure 2(b) to illustrate how the change in number of posts affects the influence probability score.

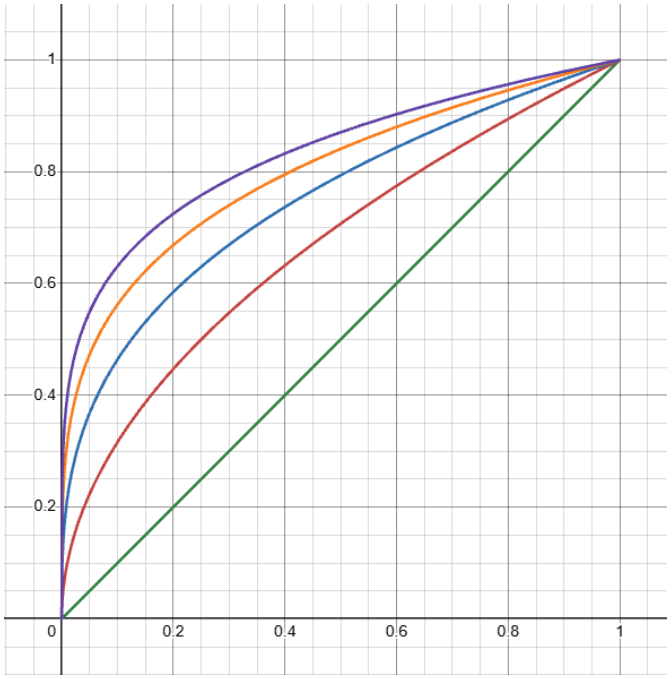

Fig. 1. The area under the graph of f(x) increases as n increases when taking the nth root of an increasing linear function bounded by [0, 1].

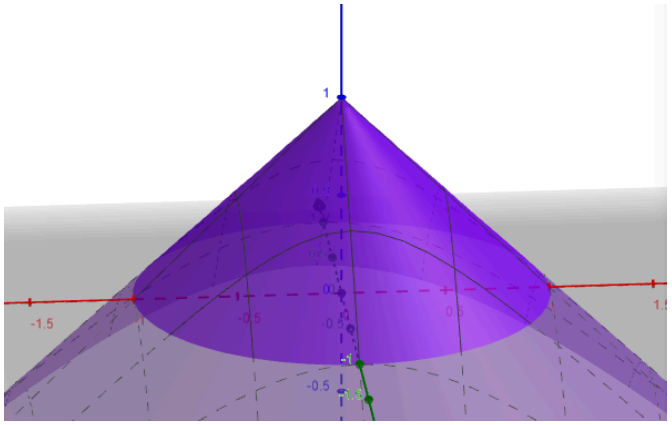

Fig. 2(a). Influence as function of distance from a single post, $|P| = 1$

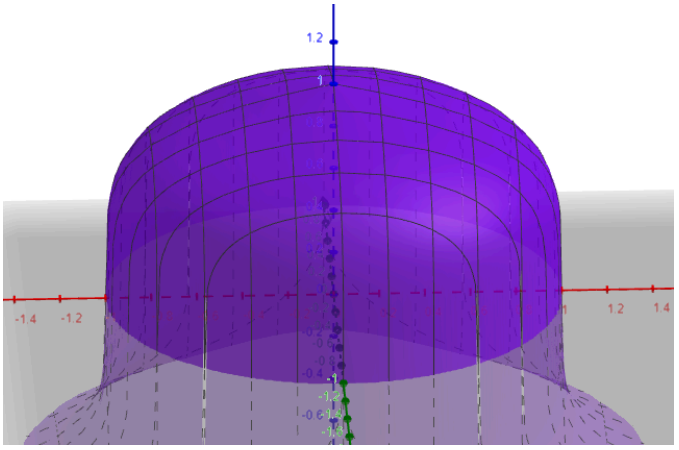

Fig. 2(b). Influence as function of distance from a single post, $|P| = 5$

The increased area under the curve represents the increased likelihood that the point of interest belongs to a given class based on distances from nearby posts. In the limit, or as the value of $|P|$ goes to infinity, probability of a point belonging to a specific class is maximal regardless of the distance within the specified max distance.

2. Gaussian RBF average euclidean distance formula

An extension of a gaussian radial basis function (RBF) can also be used to calculate the influence from posts. Gaussian processes are used extensively in machine learning and other fields [11] [12]. The advantages of an RBF over the non-linearized average euclidean distance formula shown previously is that it is implicitly non-linear and also that the curve of the generated gaussian can be controlled via the epsilon parameter which can be used to further modify the slope of the graph.

The gaussian RBF raised to a negative exponent is an exponential decay function, with parameters for epsilon and distance squared. Epsilon can be modified to change the slope of the curve, where a decreasing value in the reciprocal term will increase the total area under the curve. The reciprocal term should decrease as the number of posts increase because influence, and thus the area under the curve, should increase based on the number of nearby posts. The average value of this exponential formula from the point of interest to each post is used to give the influence on the point. This gives us the following exponential decay formula:

$$I_r = \frac{1}{|P|}\sum_{P_0}^{P_n} exp\left(-\frac{1}{\sqrt{|P|}} \cdot \left[(x-c_x)^2 + (y-c_y)^2\right]\right)$$

This method of calculating distance does not require a maximum distance to be explicitly defined. The influence probability score is increased based on the average post distance and weighted by the cardinality of the nearby posts of a given class. The increased influence from a point to a single post as $|P|$ grows larger is shown in Figure 3(a) and (b).

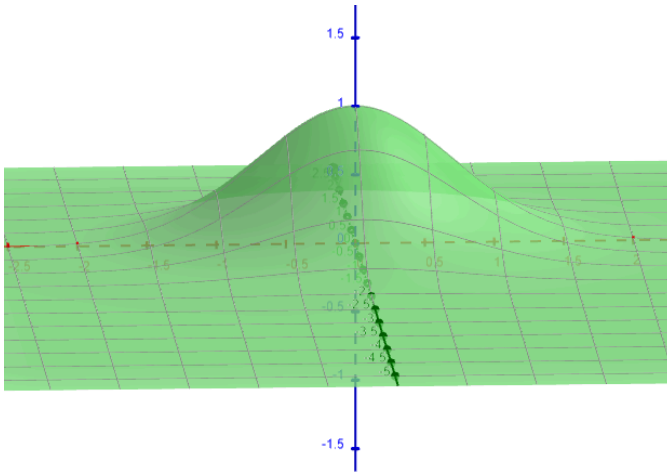

Fig. 3(a). Gaussian RBF influence as function of distance from a single post, |P| = 1

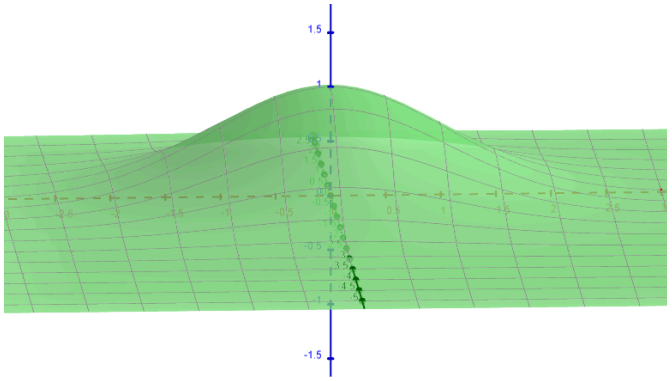

Fig. 3(b). Gaussian RBF influence based on distance from a single post, |P| = 5

An increase in the value of $|P|$ results in a more gradual curve which applies a higher weighted score at greater distances when there are more posts of a given class nearby. Note that the maximal amount of influence does not have to be 1, and instead be the label prediction score from the neural network on at each post, though that is beyond the scope of this paper.

D. Filtering points based on the Midpoint Circle Algorithm

The size of a map will grow as the robot explores more territory and thus it is a good idea to limit the number of points in consideration within any operation which relies on proximity of points. The alternative method of checking every point on the overall grid may become computationally infeasible over time. The midpoint circle algorithm (MCA), originally described by Dr. Michael Pitteway in the 1960's for rasterizing two-dimensional ellipsoids, can be extended to give a list of all points within an area of influence given a certain radius, *R* [13].

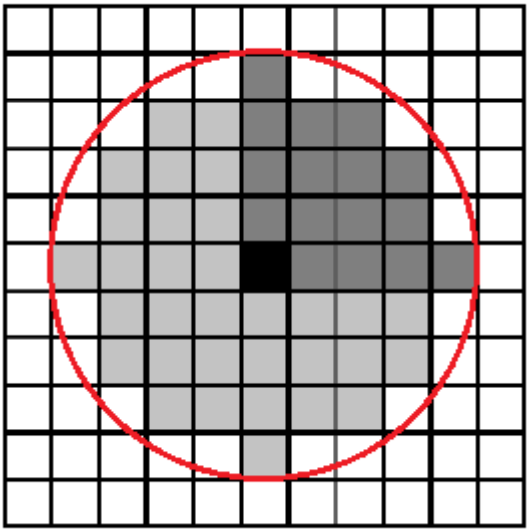

Fig. 3. Example of a grid generated using a filled midpoint circle algorithm. The dark gray area represents cells used to calculate offsets to generate all other cell positions using symmetry.

To get all points within the radius, the MCA can be extended by calculating the width of each row within the first first quadrant of the circle and storing them as offsets. Using symmetry the remaining points of the grid can be constructed by iterating over the range of points at each level from the negative of the offset (relative to the center line) to the positive of the offset. In this way every cell within range of a given cell can be checked to determine the value, or presence, of the cell. The increase in the area of the circle is quadratic and so

this process is increasingly beneficial in terms of run-time performance as the radius increases (or the resolution is increased). A sample implementation of this process in Python can be found in *Appendix A*.

E. Logical segmentation and clustering

Mapped posts can be used to generate the full perimeter of objects using perimeter closing algorithms using a combination of posts and/or points. Simple methods might use distance-based techniques to correlate points or perhaps algorithms like k-means clustering which becomes viable if the number of mapped objects are known. Additionally, resolution on the object's perimeter is enhanced when more posts are mapped to perimeter points on the object, thus the perimeter can be recalculated after more sensor readings to obtain refined results.

A further, more abstract, observation may be made that grouping together sensor readings to aggregate posts, spaced by fixed intervals, could reduce the overall number of posts required to be stored which in turn would increase overall storage efficiency. However, the details of this process adds complexity and is beyond the scope of this paper.

F. Time and space efficiency for the posts method

A very efficient set of operations can be constructed using the ideas and methods outlined in this paper to manage detected objects constructed from labels stored in posts. Storing object detections as posts means that the values of surrounding cells can be recalculated in-place using relatively simple distance calculations on a small set of points from the midpoint circle algorithm.

One of the most time intensive operations within this system is the searching of posts from a given point to calculate the influence, which has not been discussed in this paper. However, given that post positions can be hashed and stored within very flexible time constraints (e.g. as string hashes stored in sorted order in an async queue), the process of searching for posts can be as simple as a binary search. Binary search algorithms have a time complexity upper-bound of O(n), and a lower-bound of O(log n). A data structure that is efficient for storing a large number of hash keys would be requisite to index a map which could contain upwards of hundreds of millions of posts.

Storing the posts with metadata will have, pessimistically, a per-post footprint in memory that grows linearly with the number of posts that are stored, so in terms of space complexity the upper-bound is O(n). However, it is conceivable that optimizations could be made to reduce this footprint, for example by grouping posts together that are similar. Note that the posts method saves storage space by only storing the posts themselves although on certain platforms it may be desirable to cache the labels of non-posts along with the influence values, which would decrease time complexity at the cost of increasing space complexity. Caching data in this manner could greatly reduce the overall time complexity and therefore may be very desirable if memory is cheap and/or compute cost is expensive.

V. Conclusions

The ideas and methods that have been introduced in this paper can serve as the foundation for building a system that can map objects using object labels inferred from convolutional neural networks. Probability-based statistical methods provide a way to infer information that does not have full resolution and are mathematically sound, therefore providing a way to maintain computational efficiency even in global spaces which are continuous or contain many cells. Broad overarching concepts have been introduced in lieu of specific implementation so they can be implemented in many languages and platforms. They are also designed to be extensible and open to modification for niche use-cases. The hope is that given this set of tools efficient robotic mapping systems can be created that provide great utility to researchers at scale.

Appendix A

Variation of Midpoint Circle Formula

```
# A variation of the midpoint circle formula, where offsets are calculated in
quadrant 1 and then extended to all other quadrants to get all grid cells within
range.
# map_msg: The occupancy grid message
# center_x: The x axis center point of the circle in grid cells
# center_y: The y axis center point of the circle in grid cells
# radius: The radius (in grid cells) to search around the center point (default: 8)
# target_vals: List of values representing occupied cells in the occupancy grid
(default: [1, 100])
def occupied_cells_in_radius(
        map_msg,
        center_x: int,
        center_y: int,
        radius: int = 8,
        target_vals: list[int] = [1, 100],
        debug: bool = False
        ) -> list[tuple]:
    x_current = radius
    y_current = 0
    offsets = []
    while y_current <= radius:
        offsets.append((int(x_current), y_current))
        inner = x_current**2 - 2 * y_current - 1
        if inner < 0:
            break
        x_current = math.sqrt(inner)
        y_current += 1
    points = []
    for x, y in offsets:
        for dx in range(-x, x + 1):
            candidates = [
                (center_x + dx, center_y + y),
                (center_x + dx, center_y - y)
            ]
            candidates = filter(lambda point: get_grid_val(map_msg, point[0],
point[1]) in target_vals, candidates)
            points.extend(candidates)
    return points
```

References

[1] T. Taketomi, H. Uchiyama, and S. Ikeda, ‘Visual SLAM algorithms: a survey from 2010 to 2016’, *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, p. 16, June 2017.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ‘ImageNet classification with deep convolutional neural networks’, *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[3] Y. LeCun, Y. Bengio, and G. Hinton, ‘Deep learning’, *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[4] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, ‘A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects’, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, Dec. 2022.

[5] A. Younesi, M. Ansari, M. Fazli, A. Ejlali, M. Shafique, and J. Henkel, ‘A Comprehensive Survey of Convolutions in Deep Learning: Applications, Challenges, and Future Trends’, *IEEE Access*, vol. 12, pp. 41180–41218, 2024.

[6] Sarvesh Pimpalkar, “Enhancing Robotic Localization with IMUs: A Fundamental Technology for Precise Navigation,” Analog.com, 2024. https://www.analog.com/en/resources/analog-dialogue/articles/enhancing-robotic-localization.html

[7] Ignas Andrijauskas, Marius Šumanas, Andrius Dzedzickis, Wojciech Tanaś, and Vytautas Bučinskas, “Computer Vision-Based Optical Odometry Sensors: A Comparative Study of Classical Tracking Methods for Non-Contact Surface Measurement,” Sensors, vol. 25, no. 19, pp. 6051–6051, Oct. 2025, doi: https://doi.org/10.3390/s25196051.

[8] F. Castanedo, “A Review of Data Fusion Techniques,” The Scientific World Journal, vol. 2013, pp. 1–19, 2013, doi: https://doi.org/10.1155/2013/704504.

[9] D. Wang, Z. Shen, J. Shao, W. Zhang, X. Xue, and Z. Zhang, “Multiple Granularity Descriptors for Fine-Grained Categorization,” Thecvf.com, pp. 2399–2406, 2015, Accessed: Dec. 16, 2025. [Online]. Available: https://openaccess.thecvf.com/content_iccv_2015/html/Wang_Multiple_Granularity_Descriptors_ICCV_2015_paper.html

[10] K. Liao *et al.*, ‘Deep Learning for Camera Calibration and Beyond: A Survey’, *arXiv [cs.CV]*. 2025.

[11] G. E. Hinton and D. van Camp, ‘Keeping the neural networks simple by minimizing the description length of the weights’, in *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, Santa Cruz, California, USA, 1993, pp. 5–13.

[12] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005.

[13] M. L. V. Pitteway, 'Algorithm for drawing ellipses or hyperbolae with a digital plotter', *The Computer Journal*, vol. 10, no. 3, pp. 282–289, 01 1967.

[14] C. E. Shannon, 'A mathematical theory of communication', *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.