Dynamic Continuous Semantic Counting Sensor Model for Mapping Dynamic City Street Environments

Chengyang Huang, Craig Knuth, Hojun Sun, Isabel Taylor, Ziyue Zhou

***Abstract*— This paper develops a method that improves on existing environment mapping algorithms by explicitly accounting for dynamic objects. Existing approaches to environment mapping utilize semantic information and local correlations in order to efficiently and accurately map static environments. In cases where the scene contains dynamic obstacles, these methods either exclude those objects or assume that they are static which yields inaccurate maps of the environment. However, we note that available semantic information indicates whether or not a section of the map is dynamic e.g. by classifying a point as part of a tree or part of a car. We utilize the dynamic information in 3 ways: 1) a local update that accounts for the motion of objects within the field of view, 2) a class-based refinement of the local update, and 3) a decay approach that updates dynamic obstacles that have left the field of view. Qualitative evaluation of our method shows that it is capable of tracking dynamic objects as well as mapping static obstacles in the scene with only a slight increase in additional computational load.**

I. INTRODUCTION

Environment mapping is an active research area in robotics that seeks to represent the real world as structured information which computers can understand and construct. It is essential in many applications for robots or self-driving cars to perform simultaneous localization and mapping (SLAM) and path planning. One of the most popular approaches to solve this problem can be classified as occupancy grid mapping [1], [2], [3]. These approaches divide the world into axis aligned cells (voxels) with parameters that indicate whether or not a voxel is occupied or, in the case of semantic mapping, what class of object (tree, road, car, pedestrian, etc.) occupies a voxel. Recently, proposed methods exploit local correlations in the map by use of a kernel that describes the similarity of two different points in space [4], [5], [6], [7].

However, many existing methods of environment mapping focus on constructing a map without considering dynamic objects in the scene which can lead to poor performance depending on the number and size of dynamic objects in the scene, the density of sensor measurement, and the approach to building the map. This poor performance can lead to situations where regions of the map are believed to be occupied by a dynamic object but in reality were occupied in the past but has since been vacated by the dynamic object. For these instances, the occupied space is inherently different than the real world which can yield challenges in downstream processes such as localization and path planning. For these reasons, a strategy that maps static and tracks dynamic obstacles may improve the quality of the occupancy map. For instance, the benefits in localization may be achieved when point cloud matching (ICP and similar methods [8]) result in greater number of associated points, or benefits in planning may be achieved by removing unnecessary detours caused by inaccurate occupancy in the map.

In this paper, we integrate dynamic semantic infomation to improve the method presented by Gan, Lu, et al. [4], Continuous Semantic Mapping via Bayesian Kernel Inference (BKI-CSM). In our approach, we hope to be able to track dynamic objects in the scene by virtue of which voxels are occupied. We seek to accomplish this with three adjustments to BKI-CSM. The first is a local update method that tracks dynamic objects in the field of view. The second is a class-based refinement of the local update method which uses the knowledge that different classes move at different velocities, e.g. a car moves faster than a bicycle which moves faster than a pedestrian. The third is a decay approach that updates occupancy for dynamic obstacles that are outside of the field of view. Either local update or decay can be included independently of one another, but we show that including both improves the mapping of dynamic obstacles significantly. In particular we will show that the free space of the resulting map is more accurate than the map produced by BKI-CSM with respect the ground truth.

To summarize, the contributions of this work are:

1) A local update algorithm to update occupancy of voxels for dynamic objects within the field of view
2) A class-based refinement of the local update algorithm that accounts for different velocities of dynamic obstacles
3) A decay algorithm that updates the occupancy of voxels for dynamic objects outside of the field of view
4) qualitative and quantitative evaluation of our approach on a city driving sequence provided by the SemanticKITTI dataset

The structure of the rest of the paper is as follows. Related work is discussed in Section II. The preliminaries of BKI-CSM and our adaptation to dynamic occupancy is described in Section III. The details of our method are described in Section IV. Section V shows the results of constructing a semantic occupancy grid map with our approach. Discussion and limitations of this work are presented in Section VI and the conclusion is in Section VII.

II. RELATED WORK

A grid-based map is a common approach to building maps of the environment. Among grid-based maps, approaches can be further classified by whether or not they are discrete or continuous and whether or not they utilize semantic information. For a discrete map, the occupancy of voxels

is generally updated when a scan either passes through a voxel or ends in a voxel. It does not account for where the scan is in the voxel, so a measurement at the edge of a voxel is given the same weight as a measurement at the center of a voxel [2]. However, other recent approaches utilize local correlations [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [5], [6], [7] by kernel functions, conditional random fields, or other methods. The kernel function accounts for the distance between the measurement and the center of the voxel. Additionally, a map may be built only identifying which voxels are occupied or not, or by identifying which class a voxel is occupied by. The semantic mapping approach requires some method of labeled 3D points either via labeled point clouds [4] or semantic images of stereo cameras [7].

Methods in continuous semantic mapping based on Gaussian process and semantic information [5], [6], [7] show improved map quality. The continuous mapping considered each cell in the occupancy grid map as correlating with its neighboring via a kernel. These local correlations promote smoothness of the map which in turn suppresses outliers in the measurements. Semantic information in turn is able to determine what parts of space are occupied by different classes of obstacles.

Very few approaches address dynamic information in mapping the environment. Often, dynamic information is ignored and methods rely on more measurements to eventually remove occupancy in regions where dynamic obstacles used to be. In [4], dynamic motion or obstacles were dealt with as stationary while constructing occupancy grid map. Another approach in [7] is to simply ignore mapping dynamic obstacles, which provides an accurate map of the static environment but is not capable of also tracking dynamic objects.

III. PRELIMINARIES

Our approach builds on BKI-CSM in [4]. Let $\mathcal{K} = \{1, \dots, K\}$ be a set of semantic class labels partitioned into sets $\mathcal{D} \subset \mathcal{K}$, $\mathcal{S} \subset \mathcal{K}$, and $\mathcal{F} \subset \mathcal{K}$ where $\mathcal{D}$ is the subset of classes that are considered to be dynamic, $\mathcal{S}$ is the subset of static classes, and $\mathcal{F}$ is the set of free space classes. Note that in the presence of at least one dynamic obstacle, free space is also dynamic. This is due to fact that as a dynamic obstacle moves through free space, free space necessarily changes. However, our approach in handling dynamic scenes are addressed from the perspective of dynamic obstacles moving through space. Therefore, we exclude free space from our dynamic updates. Let $x_i \in \mathcal{X} \subset \mathbb{R}^3$ be a point in 3D space and let $y_i = (y_i^1, \dots, y_i^K)$ be a one-hot measurement tuple with exactly one of $y_i^k = 1$ and all others equal to zero. Measurements (in practice provided by laser scans) are assumed to come in the form of a dataset $D = \{(x_i, y_i)\}_{i=1}^M$.

Let $x_j \in \mathcal{M}$ be the jth point in the map. x_j takes on one of the K semantic labels according to probabilities $\theta_j = (\theta_j^1, \dots, \theta_j^K)$ (i.e. a Categorical distribution). We adopt a Dirichlet distribution as a conjugate prior over θ_j described by the corresponding parameters $\alpha_0 = (\alpha_0^1, \dots, \alpha_0^K)$, $\alpha_0^k > 0$. The continuous semantic counting sensor model uses a Gaussian Process (GP) to introduce local correlations in the map where the local correlations are described by a kernel function $k : \mathcal{X} \times \mathcal{X} \to [0, 1]$. The kernel describes the similarity between measured points x_i and the points in the map x_j. Using a GP, the measurements in the dataset can be used to update the map in the following fashion.

$$\alpha_j^k := \alpha_0^k + \sum_{i=1}^{M} k(x_j, x_i) y_i^k \tag{1}$$

To extend this approach to mapping of dynamic obstacles, we set the concentration parameters as a function of time so that $\alpha_j^k := \alpha_j^k(t)$ with $\alpha_j^k(0) = \alpha_0^k$. Suppose measurements are taken at times $(t_1, \dots, t_N)$ producing corresponding datasets $(D_1, \dots, D_N)$ with $D_n = \{(x_{i,n}, y_{i,n}))\}_{i=1}^{M_n}$. Let $t_0 = 0$. Then, the corresponding *sequential* update rule is written as follows for $n > 0$:

$$\alpha_j^k(t_n) := \alpha_j^k(t_{n-1}) + \sum_{i=1}^{M_n} k(x_j, x_{i,n}) y_{i,n}^k \tag{2}$$

Note that equations (1) and (2) produce identical results when $D = \bigcup_{n=1}^{N} D_n$. Our goal is to modify the sequential update rule so that the corresponding map is more accurate with respect to the ground truth without adding significant computation time.

Evaluating the mean and variance of the class of a particular grid point can be evaluated in the same way as in [4]. The only difference is that the mean and variance are now also a function of time.

$$\mathbb{E}[\theta_j^{\bar{k}}](t) = \frac{\alpha_j^{\bar{k}}(t)}{\sum_{k=1}^{K} \alpha_j^k(t)} \tag{3}$$

$$\mathbb{V}[\theta_j^{\bar{k}}](t) = \frac{\frac{\alpha_j^{\bar{k}}(t)}{\sum_{k=1}^{K} \alpha_j^k(t)} \left(1 - \frac{\alpha_j^{\bar{k}}(t)}{\sum_{k=1}^{K} \alpha_j^k(t)}\right)}{\sum_{k=1}^{K} \alpha_j^k(t) + 1} \tag{4}$$

Additionally we provide the following shorthand notation. $\alpha_{j,\mathcal{D}}$ are the set of concentration parameters corresponding to dynamic classes.

IV. METHODOLOGY

We employ three changes to the nominal method of the continuous semantic counting sensor model [4] in order to more accurately account for dynamic changes in the map. The first is a *local update* which uses the previous laser scan concurrently with the current scan in order to update the map based on a simple assumption of the motion of dynamic obstacles in the scene. The second is a *class-based* refinement of the local update that infers parameters describing the motion of dynamic obstacles using data in the scene. The third change is introducing *decay* for dynamic obstacles that are no longer observed.

A. Local Update

The local update method intends to account for the fact that dynamic obstacles can move between subsequent laser scans. The standard mapping algorithm will consider measurements of dynamic classes at new regions in the map as novel measurements, completed unrelated to the dynamic obstacle that is nearby in the previous scan. However, since we know the object is moving, one reasonable assumption is that we should be similarly confident that the new region of space is occupied by the same dynamic obstacle. In the model, one way this can be realized is by copying the concentration parameters α from points in the previous scan to points where the new scan measures a dynamic obstacle.

A number of methods could be employed to predict the motion of dynamic obstacles from one scan to the next which dictates which concentration parameters should be copied. One method is to use the Iterative Closest Point algorithm [8] to align the point clouds for a dynamic object between adjacent scans. However, this will only work well for the case of one dynamic object. Another method is to employ multi-object tracking (via time-series clustering or some other method). Recent methods employ neural networks to predict the motion of pixels or points in 3D space from RGB images which will be described further in Section IV-D. In practice, we use an uninformed motion prediction, i.e., points in space do not move at all. With the motion prediction, grid points could be propagated and then the concentration parameters of the nearest propagated point can be copied to the grid points updated at the current scan. Any of these methods can be employed as long as the result is the propagation of a grid point between subsequent scans.

In general, we can propagate all points in the map but we found that, in practice, it is sufficient to propagate only the subset of all points in the map where a dynamic measurement is taken in the previous scan. We say a measurement is taken at a grid point x_j for class k in a scan n if $\sum_{i=1}^{M_n} k(x_j, x_{i,n}) y_{i,n}^k > 0$. Then, let $\mathcal{M}_n^{\mathcal{D}} \subset \mathcal{M}$ be the set of all points in the map where a dynamic measurement is taken in scan n. We then propagate these points according to a propagation function $F : \mathcal{M} \to \mathcal{X}$ found by one of the methods described in the previous paragraph. Let $F(\mathcal{M}_n^{\mathcal{D}})$ be the set of all points propagated through the propagation function. Then, for each grid point where a dynamic measurement is taken in the current scan, the concentration parameters of the nearest neighboring point in the propagated set of the previous scan are copied to the current grid point. This procedure is illustrated in Figure 1. If uncertainty in the propagation is also considered then the nearest neighbor search can be replaced with a maximum likelihood estimate.

A number of additional tweaks are needed before implementing this approach. First, new dynamic objects will enter the scene as the map is being built. In this case, concentration parameters of points far away from these new dynamic objects should not be copied. We address this issue by introducing a maximum distance, computed as the product

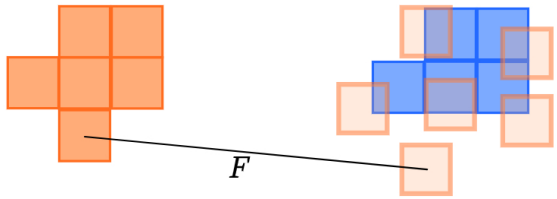

Fig. 1: Representation of the local update procedure. In a previous scan, dynamic classes are measured at the orange squares. The orange squares are then propagated with the function F to their predicted locations at the next scan shown in light orange. Novel dynamic measurements are taken at the blue squares. We copy the concentration parameters from the nearest propagated point in the previous scan, and then reset the concentration parameters of the points in the previous scan to the prior.

of a fixed maximum velocity and the time difference between the current and previous scans. If the nearest neighbor is further than the maximum distance, then the corresponding concentration parameters are not copied. This notion of a maximum distance is expanded on in the class-based refinement presented in the next section. Finally, due to our assumption that a dynamic obstacle is moving, if a point is measured to be dynamic in a previous scan but not in the subsequent scan then we reset the concentration parameter to the prior. However, we only reset the concentration parameter if the point is still in the perception field P i.e. there exists $k \in \mathcal{K}$ s.t. $\sum_{i=1}^{M_n} k(x_j, x_{i,n}) y_{i,n}^k > 0$.

The local update algorithm is summarized in Algorithm 1. The function `NearestNeighbor` finds the nearest point and the corresponding alphas of the grid point.

B. Class-Based Refinement

The class-based refinement of the local update provides a way to infer the maximum velocities v that should be used as a cutoff. The selection of this hyperparameter on a per-class basis is motivated by the intuition that different dynamic obstacles move at different speeds. For instance, a car can move faster than a bicycle which in turn can move faster than a pedestrian. To infer this distance on a per-class basis, we will use data collected from previous laser scans. We assume that the distribution of the distance of the nearest neighbor results from two sources: 1) uncertain motion of dynamic objects and 2) new dynamic objects entering the perception field. The first source will result in measurements less than the maximum speed of an obstacle multiplied by the time difference between scans. We assume the second source contributes to a uniform distribution of measured distances. This assumption is not entirely accurate as the perception field depends on the environment and is not necessarily uniform with respect to distance. After visualizing the data, we select maximum velocities corresponding to each dynamic class. This is incorporated into the local update algorithm by simply comparing maximum velocity to distance traveled with respect to each class.

Algorithm 1: Local Update

Parameters: max velocity v
Input: map $\mathcal{M}$, dataset D_n, propagation $F(D_{n-1})$, perception field P
Output: map $\mathcal{M}$

foreach x_j *in* $\mathcal{M}$ **do**
  $\alpha_{NN}, x_{NN} =$ `NearestNeighbor`$(x_j, F(D_{n-1}))$
  foreach $k \in \mathcal{D}$ **do**
    if $\sum_{i=1}^{M_n} k(x_j, x_{i,n}) y_{i,n}^k > 0$ *and* $|x_j - x_{NN}| < v(t_n - t_{n-1})$ **then**
      $\alpha_j^k(t_n) = \alpha_{NN}^k(t_{n-1}) + \sum_{i=1}^{M_n} k(x_j, x_{i,n}) y_{i,n}^k$
    else
      $\alpha_j^k(t_n) = \alpha_j^k(t_{n-1}) + \sum_{i=1}^{M_n} k(x_j, x_{i,n}) y_{i,n}^k$
foreach $x_j \in \mathcal{M}_n^{\mathcal{D}}$ **do**
  if $x_j \in P$ **then**
    foreach $k \in \mathcal{D}$ **do**
      if $\sum_{i=1}^{M_n} k(x_j, x_{i,n}) y_{i,n}^k > 0$ **then**
        $\alpha_j^k(t_n) = \alpha_0^k$

C. Decay of Concentration Parameters on Dynamic Classes

Due to the nature of the current mapping algorithm[4] that updates the concentration parameters α by evidence from sensors, the dynamic objects would remain on the map if they have been out of the perception field and no further evidence on that object received. This is not true in most cases of dynamic objects, and it will even harm the quality of the map if the object is moving in the mapping process, which causes an elongated object created on the map like a panorama glitch, as shown in Figure 2. To solve this issue, two algorithms are proposed, which are naive decay and decay by motion estimation.

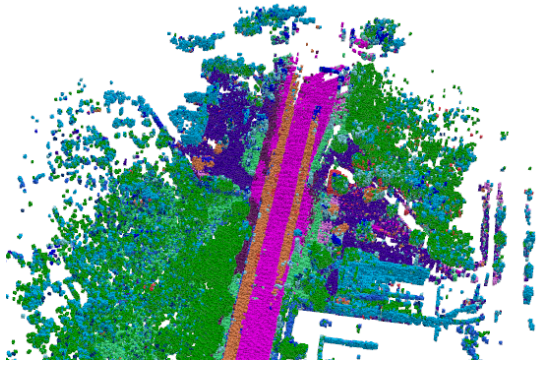

Fig. 2: Mapping glitch that two moving cars were captured by the sensors while mapping. Two elongated orange lines represent the trajectory of two moving cars, which is a problem on a real-time map.

1) Naive decay: Naive decay method assumes that all dynamic objects in the map will eventually move, and it neglects the real-time speed of the object. As illustrated in Algorithm 2, all the concentration parameters of dynamic classes $\alpha_{j,\mathcal{D}}$ will decay over time by the decay rate η. Decay rate η is a predefined tunable parameter. It is recommended to determine the decay rate η based on the prior belief of the speed for a particular dynamic class. For example, if the decay rate of the car class is 0.5, the decay rate of the bicycle should be around 0.8, because the percent change between the speed of a bicycle and that of a car on a city road is normally 60%. Once the concentration parameters of dynamic classes $\alpha_{j,\mathcal{D}}$ decays below a threshold τ, it is set to be zero and will not be tracked unless it enters the perception field again in order to accelerate the decay process and reduce the computation burden.

The drawback of this method can be easily inferred from its assumption: it will remove the object in dynamic class with zero speed on the map, e.g. a parking car. In the long term, it is a reasonable decision, and the assumption is valid. However, it is dangerous in the real-time application, like in a SLAM problem, because the robot may hit a zero speed object that is in the dynamic class when planning the path based on the decayed map. Therefore, this method is only safe to use in the environment where there is no zero speed dynamic object, e.g. expressway, or the application in which the path-planning does not purely rely on the real-time mapping process.

Algorithm 2: Naive Decay

Input: map $\mathcal{M}$, perception field $\mathcal{P}$
Parameters: threshold τ, decay rate η, time since last update Δt

foreach $x_j \in \mathcal{M}$ **do**
  if $x_j \notin \mathcal{P}$ **then**
    foreach $k \in \mathcal{D}$ **do**
      if $\alpha_{j,k} > \tau$ **then**
        $\alpha_{j,k} = \eta_k \, \Delta t \, \alpha_{j,k}$
      else
        $\alpha_{j,k} = 0$

2) Decay by motion estimation: Unlike the naive decay method, decay by motion estimation method considers the real-time speed of an object. Therefore, the issue of zero speed dynamic objects can be resolved using this method. However, it brings another challenge, motion estimation problem, into the mapping process as discussed in Section IV-D. If the motion estimation problem is solved, two potential methods can be utilized as discussed below.

a) Improved naive decay: The first method is similar to the naive decay method in Algorithm 2. The only difference is that the decay rate η will depend on the real-time speed that is estimated before the object is out of the perception field instead of the prior belief of the speed in the naive decay

method. This method is better than the naive decay method, but it still assumes that the object moves in a constant velocity after it is out of the perception field.

b) Decay with motion uncertainty: The second method, as shown in Algorithm 3, considers the uncertainty of the object's velocity after it is out of the perception field. The function `Neighbors` returns all points in the map within the given distance centered at the given point. The motion uncertainty is modeled as the Gaussian distribution. It first predicts the position in the next step based on the object's velocity, and then it distributes the concentration parameters of dynamic classes $\alpha_{j,\mathcal{D}}$ of the original position to the neighborhood of the predicted position by the Gaussian distribution. Therefore, the moving object will keep moving based on its last estimated velocity by Gaussian uncertainty and fade out of the map over time while the zero speed dynamic object remains at the original position. The advantage of this method is obvious in that it is able to create a reliable real-time map in most situations and also removed the mapping glitch shown in Figure 2.

Algorithm 3: Decay by motion estimation and Gaussian uncertainty

Input: map $\mathcal{M}$, perception field $\mathcal{P}$, motion V
Parameters: threshold τ, uncertain window size D, time since last update Δt, Gaussian distribution PDF $f_N \sim \mathcal{N}(0, \Sigma)$

foreach $x_j \in \mathcal{M}$ **do**
  if $x_j \notin P$ *and* $V_j > 0$ **then**
    $x_{pred} = x_j + V_j \Delta t$
    $X_{nbr} = \texttt{Neighbors}(x_{pred}, D)$
    foreach $x_{j'} \in X_{nbr}$ **do**
      $d = |x_{j'} - x_{pred}|$
      foreach $k \in \mathcal{D}$ **do**
        $\alpha_{j',k} = \alpha_{j',k} f_N(d)$
        if $\alpha_{j',k} < \tau$ **then**
          $\alpha_{j',k} = 0$

D. Motion Estimation

The motion estimation problem is defined as an optical flow estimation problem in this project using the RGB images from the scene. Two main approaches, energy minimization approach [20] [21] [22] and deep learning approach[23] [24], are found and tested to solve this problem. Theoretically, the energy minimization approach is able to estimate the flow precisely while it is more computation expensive and takes longer, which means it is not suitable for the real-time application but it can be used if the online estimation is not necessary. The deep learning approach has shorter solving time, but it needs time to train the neural network model. If a reliable neural network model is obtained, then it can be applied to the local update and decay method by applying the camera-world transformation using the intrinsic and extrinsic matrix of the camera and other information.

V. EXPERIMENTAL RESULTS

We run our experiments using the Semantic KITTI[25] [26] dataset which provides labeled sequential pointcloud scans for 20 driving sequences. In addition to the scans, it also provides the lidar pose for each scan. The lidar runs at a frequency of 10Hz. In addition to labeled pointcloud scans, a ground truth dataset is also provided. The ground truth dataset is also in the form of labeled point clouds.

The experimental platform used for processing the dataset is ThinkPad X1 Extreme equipped with an INTEL® CORE™ i7-9750H Processor(2.60 GHz, up to 4.50 GHz with Turbo Boost, 6 Cores, 12 Threads, 12 MB Cache) and 16 GB Memory(DDR4 2666MHz). Also, the project runs on ROS Melodic, Ubuntu 18.04 and the results are displayed in Rviz.

A video of the results is available here: `https://youtu.be/dcQJYqtgdHA`. Code is available on GitHub here: `https://github.com/ChengyangHuang/DynamicSemanticMapping`.

A. Inferring Class Velocities

To infer maximum velocities for each class, we display the inferred velocity and number of observations for two dynamic classes in the 4th Semantic KITTI sequence. The number of measurements here means the number of times the distance between the current point and the nearest neighbor is observed. While there is certainly a peak near zero as expected, the cutoff point for the maximum velocity is not completely clear. Some expertise in the problem is needed to select an appropriate maximum velocity and this could potentially be improved through some automatic selection of this hyperparameter. Nonetheless, we see that the cutoff point for a car is larger than for a bicyclist as expected.

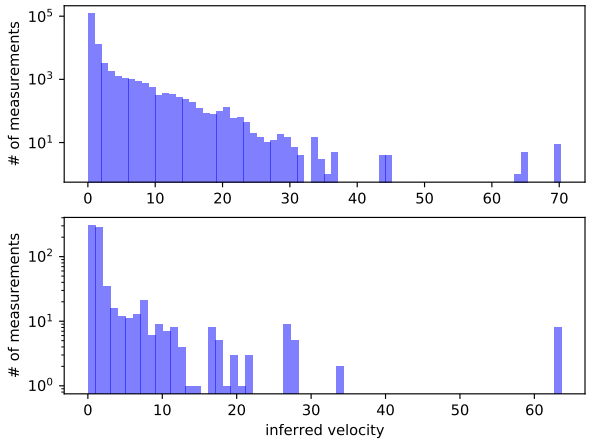

Fig. 4: Inferred velocity versus the number of measurements for cars (top) and bicyclists (bottom) plotted in log scale. From this data, we selected a maximum velocity of 20 m/s for cars and 3 m/s for bicycles.

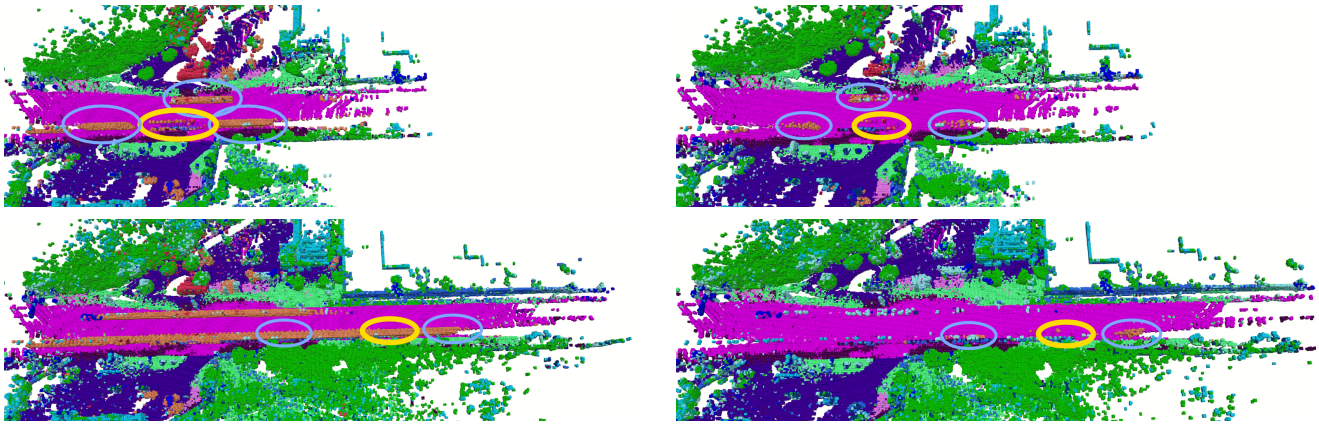

Fig. 3: Results of running the baseline mapping algorithm (left) and our method (right) at $t = 1.5$ seconds (top) and $t = 4.5$ seconds (bottom). The ego car is outlined in yellow in each image, and other cars on the road are highlighted with light blue ellipses. Three important classes of voxels to note here are the road (pink), moving cars (orange), and parked cars (light blue).

B. Mapping Dynamic Obstacles

In our experiments, we implemented the local update with a propagation function that is simply the identity, i.e. all dynamic obstacles stay where they are. We found that in practice a more sophisticated propagation function was not needed. In terms of decay, we implemented the naive decay algorithm with decay rate $\eta = \frac{2}{3}$ and threshold $\tau = 1$, the decay by motion estimation approach is not successfully implemented due to the failure of the transformation of motion estimation described in Section V-D.

A key issue that we noticed in the baseline method presented by Gan et al. [4] is the "streaking" of dynamic obstacles. As dynamic obstacles are measured, the grid point is updated as occupied but the method never recovers. A primary cause of this issue is the lack of density of measurements in free space. Despite laser scans passing through occupied voxels on the map, free space measurements are not sampled in those voxels. This results in locations were dynamic obstacles are never cleared.

Rather than sampling dense free space measurements, our method takes advantage of the knowledge of which classes are dynamic in order to quickly update the map. In Figure 3, we see that our method is capable of removing the streaking of cars in the environment whereas the baseline produces significant streaking leading to an obviously wrong map.

As shown in Figure 3, the ego car (where the LiDAR is mounted) is outlined in yellow in each image, and other cars on the road are highlighted with light blue ellipses. Two important classes to note here are the road (pink) and cars (orange). For the baseline method, the cars streak across the map, predicting that the region above the road is occupied. Additionally, this makes it impossible to identify where other cars on the road are at $t = 4.5$ seconds. However, using our method, cars are tracked as part of the map rather than streaked. This is especially clear in the top right figure where two cars are mapped in the bottom lane and one is mapped in the top lane. It is notable that the moving cars are occasionally mistaken as parked vehicles (light blue). Since this is a static class, our method does not update these values which results in sparsely occupied voxels above the road.

The quantitative results are shown in Table I. This comparison is performed in the following way. The ground truth scan of the environment is provided which maps visible voxels to its ground truth label. Our method then computes the most likely label for the same points. The Jaccard index is computed over the resulting ground truth and predicted label set. Our method performs comparably for all classes, both static and dynamic, to the baseline. A significant improvement is not realized in part due to the nature of the ground truth data set. The ground truth dataset is produced by hand labeling the endpoints of laser scans. Therefore, most of the free space voxels are not included in the ground truth which are the parts of space where our method provides an improvement by a reduction in streaking of dynamic obstacles (see Figure 3). Unfortunately, ground truth data that includes dense free space labels are not available so we are not able to provide that comparison.

C. Time Consumption

We run a time-consumption test on both the baseline method BKI-CSM [4] and our improved version for the first 50 scans in the Semantic KITTI [25] [26] dataset. The baseline method takes up 675 seconds, with an average of 13.50s/scan; while in our case it takes 692 seconds, with an average of 13.84s/scan, as shown in Table II. The longer time indicates that we introduce some computational burden since we maintain a couple of lists to track the dynamic nodes and use additional iterating loops to perform the local update and the decay for dynamic obstacles.

D. Motion Estimation

After implementing simplified energy minimization approaches, e.g. Gunner Farneback's algorithm by OpenCV and dense Lucas-Kanade algorithm, and deep learning approach, e.g. PWC-Net, the optical flow estimation example is shown

TABLE I: Jaccard index for each class for our method and the baseline. A Jaccard index of 1 is achieved when the ground truth is perfectly classified (no false positives or false negatives) and 0 is achieved when no point is classified correctly (no true positives). The ground truth set is dense, so the Jaccard index is quite small fro all classes. However, this analysis excludes free space since the ground truth data set is comprised of the endpoints of all laser scans. Free space is where our method provides the most improvement as demonstrated in Figure 3.

Class	Car	Road	Sidewalk	Other Ground	Building	Fence	Vegetation	Trunk	Terrain
Ours	3.126e-05	**3.144e-03**	**5.238e-06**	**8.483e-03**	7.229e-06	7.310e-05	**9.319e-04**	4.404e-05	6.317e-03
BKI-CSM	**1.292e-02**	3.130e-03	0	8.284e-03	**7.283e-06**	7.311e-05	8.183e-04	**4.461e-05**	6.317e-03

TABLE II: Time comparison between our method and the baseline.

	Ours	BKI-CSM	Change
Avg. time(s)	13.84	13.50	2.52 %
Total time(s)	692	672	2.98 %

in Figure 4. It is found that the deep learning approach has more accurate motion estimation in a shorter solving time than the simplified energy minimization approaches. Therefore, in a real-time application, it is optimal to use the PWC-Net model or other similar neural network models to estimate the motion. However, like the semantic segmentation process in this experiment, we pre-processing the motion estimation in order to save the testing time.

When transforming the 2D velocity vector into a 3D world, we found that incorporating 2D motion estimation did not improve results. This is due to an inherent ambiguity in translating 2D motion into 3D motion. Since the 2D motion is estimated in the plane of the camera, motion along the view-line of the camera is not registered. Therefore, this is not incorporated into the local update approach and decay with the motion estimation approach.

VI. DISCUSSION AND FUTURE WORKS

Our framework for processing dynamic obstacles in environment mapping shows that considerable benefit can be gained by using dynamic information about the scene. Through the local update strategy, we can track vehicles in the scene without densely sampling free space. Furthermore, voxels that are no longer visible can also be updated without any measurement in a reasonable fashion such as the voxels behind the car following the ego car (lower left blue ellipse of the lower right image in Figure 3). Verifying the increase in performance will require a different evaluation as compared to the ground truth, as we will need to explicitly consider the accuracy of our built map in free space regions.

The code in our method is implemented naively, and additional performance benefits can be gained via a couple of strategies. First, using a KD-Tree will likely provide some performance benefit in the nearest neighbor search as compared to brute force. We initially implemented brute force in the case where uncertainty would be included in the motion model of the dynamic obstacles, and did not have time to switch to KD-Trees once we decided to not include uncertainty. For decay, it is possible to implement the decay method in a lazy fashion by additionally tracking the last time a node is updated. This would eliminate the need to update any node that is not in the field of view while mapping. If such a map is used for planning or localization purposes, all nodes can be updated according to the decay algorithm once at the time of planning or localization.

At the moment, the most pressing future work is incorporating a motion model for dynamic obstacles in the scene. As discussed previously, we were unable to use the results from the motion prediction using solely images due to the inherent ambiguity of motion along the view-line of the camera. In order to incorporate a motion model, a motion prediction in 3D space is necessary, with or without uncertainty.

Beyond incorporating the motion model, another improvement could be accomplished by considering the interaction between dynamic and static obstacles. First, a significant source of error in our model comes from mistaking a dynamic object for a static object. Without densely sampling free space, this could be addressed by considering the motion of dynamic objects in regions believed to be occupied by a static obstacle. For instance, as a car travels through voxels that have previously measured to be static, it may be clear that the region is not occupied by a static obstacle depending on the concentration parameters for the static and dynamic obstacles.

VII. CONCLUSION

In this paper, we provide a couple of adjustments to the continuous semantic counting sensor model that accounts for dynamic obstacles in the scene. Our first change is the introduction of a local update which updates concentration parameters by inferring the motion of dynamic obstacles. This is improved via a class-based refinement that uses data to separates the case where a new dynamic obstacle enters the scene from the case where a dynamic obstacle moves within the scene. Our second key change is the introduction of a decay term that processes dynamic obstacles outside of the field of perception of the robot. These changes together give rise to a dynamic continuous semantic counting sensor model that is capable of both mapping static obstacles in the scene while tracking dynamic occupancy.

We verify the success of our algorithm by qualitatively demonstrating its effectiveness in eliminating dynamic obstacle streaking. Additionally, we show that this method does not degrade in performance as compared to the baseline while only slightly increasing the computational load.

(a) First frame (b) Second frame

(c) Opencv-GF (d) Pyramid-LK (e) PWC-Net

Fig. 4: Visual results of optical flow estimation of first frame by Opencv-GF, Dense-LK, and PWC-Net using KITTI 2015 data set

REFERENCES

[1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[2] A. Elfes, "Sonar-based real-world mapping and navigation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 3, pp. 249–265, 1987.

[3] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[4] L. Gan, R. Zhang, J. W. Grizzle, R. M. Eustice, and M. Ghaffari, "Bayesian spatial kernel smoothing for scalable dense semantic mapping," *IEEE Robotics and Automation Letters*, vol. 5, pp. 790–797, April 2020.

[5] J. Stückler, N. Biresev, and S. Behnke, "Semantic mapping using object-class segmentation of rgb-d images," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3005–3010, IEEE, 2012.

[6] H. He and B. Upcroft, "Nonparametric semantic segmentation for 3d street scenes," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3697–3703, IEEE, 2013.

[7] S. Sengupta, E. Greveson, A. Shahrokni, and P. H. Torr, "Urban 3d semantic modelling using stereo vision," in *2013 IEEE International Conference on robotics and Automation*, pp. 580–585, IEEE, 2013.

[8] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Sensor Fusion IV: Control Paradigms and Data Structures* (P. S. Schenker, ed.), vol. 1611, pp. 586 – 606, International Society for Optics and Photonics, SPIE, 1992.

[9] S. Yang, Y. Huang, and S. Scherer, "Semantic 3d occupancy mapping through efficient high order crfs," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 590–597, IEEE, 2017.

[10] S. T. O'Callaghan and F. T. Ramos, "Gaussian process occupancy maps," *The International Journal of Robotics Research*, vol. 31, no. 1, pp. 42–62, 2012.

[11] S. Kim and J. Kim, "Gpmap: A unified framework for robotic mapping based on sparse gaussian processes," in *Field and service robotics*, pp. 319–332, Springer, 2015.

[12] M. G. Jadidi, J. V. Miró, R. Valencia, and J. Andrade-Cetto, "Exploration on continuous gaussian process frontier maps," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6077–6082, IEEE, 2014.

[13] J. Wang and B. Englot, "Fast, accurate gaussian process occupancy maps via test-data octrees and nested bayesian fusion," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1003–1010, IEEE, 2016.

[14] F. Ramos and L. Ott, "Hilbert maps: scalable continuous occupancy mapping with stochastic gradient descent," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1717–1730, 2016.

[15] M. G. Jadidi, J. V. Miro, and G. Dissanayake, "Gaussian processes autonomous mapping and exploration for range-sensing mobile robots," *Autonomous Robots*, vol. 42, no. 2, pp. 273–290, 2018.

[16] K. Doherty, J. Wang, and B. Englot, "Bayesian generalized kernel inference for occupancy map prediction," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3118–3124, IEEE, 2017.

[17] K. Doherty, T. Shan, J. Wang, and B. Englot, "Learning-aided 3-d occupancy mapping with bayesian generalized kernel inference," *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 953–966, 2019.

[18] M. G. Jadidi, L. Gan, S. A. Parkison, J. Li, and R. M. Eustice, "Gaussian processes semantic map representation," *arXiv preprint arXiv:1707.01532*, 2017.

[19] L. Gan, M. G. Jadidi, S. A. Parkison, and R. M. Eustice, "Sparse bayesian inference for dense semantic mapping," *arXiv preprint arXiv:1709.07973*, 2017.

[20] B. G. S. Berthold K.P. Horn, "Determining optical flow," *Artificial Intelligence*, 1981.

[21] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis* (J. Bigun and T. Gustavsson, eds.), (Berlin, Heidelberg), pp. 363–370, Springer Berlin Heidelberg, 2003.

[22] J.-Y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker," 1999.

[23] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume," *arXiv preprint arXiv:1709.02371*, 2017.

[24] W.-C. Ma, S. Wang, R. Hu, Y. Xiong, and R. Urtasun, "Deep rigid instance scene flow," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[25] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences," in *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[26] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pp. 3354–3361, 2012.