

Automatic Map Update Using Dashcam Videos

Aziza Zhanabatyrova^{ID}, Clayton Frederick Souza Leite^{ID}, and Yu Xiao^{ID}

Abstract—Autonomous driving requires 3-D maps that provide accurate and up-to-date information about semantic landmarks. Since cameras present wider availability and lower cost compared with laser scanners, vision-based mapping solutions, especially, the ones using crowdsourced visual data, have attracted much attention from academia and industry. However, previous works have mainly focused on creating 3-D point clouds, leaving automatic change detection as an open issue. We propose a pipeline for initiating and updating 3-D maps with dashcam videos, with a focus on automatic change detection based on comparison of metadata (e.g., the types and locations of traffic signs). To improve the performance of metadata generation, which depends on the accuracy of 3-D object detection and localization, we introduce a novel deep learning-based pixelwise 3-D localization algorithm. The algorithm, trained directly with Structure from Motion (SfM) point cloud data, accurately locates objects in 3-D space by estimating not only depth from monocular images but also lateral and height distances. In addition, we also propose a point clustering and thresholding algorithm to improve the robustness of the system to errors. We have performed experiments with different types of cameras, lighting, and weather conditions. The changes were detected with an average accuracy above 90%. The errors in the campus area were mainly due to traffic signs seen from a far distance to the vehicle and intended for pedestrians and cyclists only. We also conducted cause analysis of the detection and localization errors to measure the impact from the performance of the background technology in use.

Index Terms—Autonomous driving, change detection, localization, mapping, Structure from Motion (SfM).

I. INTRODUCTION

HIGH-DEFINITION (HD) 3-D maps are an important component in current autonomous driving solutions as they provide essential information for safe maneuvering in complex urban environments. Several mapping companies—such as HERE and TOMTOM—have already allocated diverse efforts to build, maintain, and distribute HD maps. The creation of HD maps involves vehicles equipped with high-precision LiDAR sensors driving through different areas to collect point cloud data of the environment. Due to the high costs of LiDAR sensors, the vehicle fleet in charge of this task is limited to a few units and, therefore, considerably outnumbered by the number of roads. In addition, roads that were previously mapped need to be maintained, i.e., constantly monitored for changes and updated when necessary.

Manuscript received 3 June 2022; revised 19 December 2022; accepted 9 February 2023. Date of publication 22 February 2023; date of current version 23 June 2023. This work was supported in part by the European Union’s Horizon 2020 Research and Innovation Programme under Grant 825496; and in part by the Academy of Finland under Grant 317432 and Grant 318937. (Corresponding author: Yu Xiao.)

The authors are with the Department of Communications and Networking, Aalto University, 02660 Espoo, Finland (e-mail: yu.xiao@aalto.fi).
Digital Object Identifier 10.1109/JIOT.2023.3244693

Hence, the efficiency of building and maintaining HD maps is a bottleneck for autonomous driving. Compared with LiDAR sensors, visual sensors, such as dashcams, have a considerably lower cost; they are also widely available in the market and are easy to be utilized. Previous works [1], [2] have also shown the feasibility of creating accurate 3-D point clouds from unordered images using Structure from Motion (SfM) techniques, which allows the input to be collected through crowdsourcing. However, due to the high computational complexity of SfM-based point cloud generation, it is too costly to frequently reconstruct point clouds from crowdsourced visual data. Therefore, it becomes essential to effectively detect and localize changes in the environment, and remap only the regions where the changes occurred. Currently, this is an under-explored topic and still remains an open issue.

We propose a pipeline based on SfM techniques for initiating and updating a semantic 3-D map, with a focus on automatically detecting changes based on a comparison of the metadata (i.e., types and locations of traffic signs). Our pipeline generates a sparse point cloud that, combined with image-based semantic segmentation and object detection, enables the automatic generation of semantic map data—termed as metadata and consisted of types and 3-D locations of traffic signs—assisted with a clustering algorithm that we devise. Our system supports lightweight change detection by comparing the semantic map data with a thresholding algorithm to induce robustness to errors. With the multilayer design, the dynamic map data representing temporary changes are stored on separate layers. The contributions of our work are summarized as follows.

- 1) Our pipeline provides a novel method for utilizing SfM-based point clouds to train a deep-learning model for online pixelwise 3-D localization from monocular RGB data. This method allows localizing traffic signs online with respect to the camera poses with high accuracy. Compared with our method, previous works [3], [4] provide only depth information, neglecting the lateral and height information necessary for 3-D localization, and require LiDAR solutions to serve as annotations (ground truth). By utilizing data extracted from SfM-based point clouds to serve as ground truth, we discard the need for LiDAR annotations and additionally provide lateral and height distances.
- 2) Compared with other solutions that utilize a multitude of sensors, such as LiDAR, inertial measurement unit (IMU), real-time kinematics (RTKs) positioning, and complex sensor fusion algorithms, our pipeline offers a simple solution—i.e., only single-view RGB images and GPS coordinates—for change detection while achieving similar performance. Moreover, even though our

proposal can benefit from multiple camera views, a sole camera view is enough to obtain high change detection accuracy.

- 3) Our pipeline reduces manual effort by eliminating the need for labels of changes and their locations since it does not utilize an end-to-end neural network trained on a data set specifically designed for change detection [5], [6], [7].

We collected dashcam videos from two urban areas—campus and residential—in February and December 2019, and used the data to evaluate system performance and conduct cause analysis of detection and localization errors. The change detection results in the residential area showed that our method was able to hit 100% accuracy. In the campus area, the change detection accuracy was 85%, where the errors were mainly due to traffic signs seen from a far distance from the vehicle and intended for pedestrians and cyclists only. We also provided an analysis of the errors of each component of the pipeline and how they can be improved to enable more accurate change detection and localization. Also, our proposal has been evaluated under complex urban scenarios with different weather, lighting conditions, and camera types.

The remainder of this article is organized as follows. Section II reviews the background technology used for building our solution. In Section III, we provide an overview of the system architecture, followed by the detailed designs described in Sections IV and V. The evaluation of these designs using the data sets presented in Section VI is discussed in Sections VII and VIII. The related studies and future work are summarized in Sections IX and X before concluding this article in Section XI.

II. BACKGROUND

This section introduces the technical background and prior works on SfM, semantic segmentation, and object detection.

A. Structure From Motion

SfM has been used in our system to create 3-D point clouds from 2-D images. A typical SfM pipeline consists of three steps: 1) feature extraction; 2) feature matching; and 3) bundle adjustment. The first step extracts highly distinctive and invariant features from the images, whereas the second step tries to match these features between image pairs. The matches are input for the last step that jointly produces optimal estimates of camera poses and locations of 3-D points. Such a pipeline has been implemented in several SfM software, such as COLMAP [2] and VisualSfM [8]. We implement our system based on COLMAP since it offers improved robustness, accuracy, completeness, scalability, and has been released as open-source software. Our system allows crowdsourced visual data as input, without camera calibration and motion information.

Regarding feature matching, several methods include different options, such as exhaustive, sequential, spatial, vocabulary-tree-based [9], and custom feature matching. In the case of exhaustive feature matching, each image is matched against all others. Since it can result in an excessive processing duration, it is only indicated for small data sets of unordered images [10].

Exhaustive matching is, for this reason, not utilized in this work. When the images are ordered in a sequence (such as when they are extracted from a video), sequential matching is recommended [10]. In this matching method, the images are matched only against their closest ones. Hence, the benefit is a shorter processing duration. The spatial matching method utilizes spatial data—e.g., the GPS coordinates of all images—as additional input for faster processing. However, it can often lead to model fragmentation possibly due to inaccurate location information. Hence, it is discarded in this work. In vocabulary-tree-based matching, each image is matched against its visually nearest neighbors using a vocabulary-tree with spatial reranking, which is recommended for large image collections [9]. Finally, custom matching is a method where the user defines a list of pairs of images to be matched. It is recommended for unordered data sets and requires manual labor [10].

For the present work, we discard the use of exhaustive and spatial matching due to the aforementioned reasons. The remaining choices consist of vocabulary-tree-based, sequential, and custom matching methods. As it will be described in Section VIII-A, custom matching is, especially, useful for performing camera pose estimation due to its flexibility. The choice between vocabulary-tree-based and sequential matching for mapping is not straightforward. In Section VII-A, we test both matching methods and provide a conclusion on which method is more appropriate for our case.

The SfM pipeline outputs a 3-D point cloud with random scale and orientation. Therefore, geo-registration (or georeferencing)—which consists of a similarity transformation—is typically performed afterward to rescale and align the model with respect to the real world. The geo-registration function provided by COLMAP requires the real-world Cartesian coordinates of at least three distinct images uniformly spaced across the map. The simplest way to do this is by utilizing positioning services—e.g., GPS or RTK positioning—during the visual data collection to obtain geodetic coordinates for each image and then transform these coordinates into Cartesian ones. In case positioning services are not available during the data collection, a solution is to manually select three locations in the map and obtain their GPS coordinates with the help of tools, such as Google Maps, and transform their GPS coordinates into Cartesian ones.

COLMAP does not provide methods for change detection or automatic point cloud update. However, it provides functions for deleting images and their related 3-D information from a point cloud, as well as registering new images into an existing point cloud. When a change in the scene has been detected and localized, it is possible to utilize these functions to delete the corresponding 3-D points from the existing point cloud, and then register the images capturing the new scene into the point cloud. Therefore, the focus of this article is placed on change detection and localization rather than the implementation of the point cloud update.

B. Semantic Segmentation and Object Detection

Semantic segmentation and object detection are computer vision tasks employed to detect objects in an image and

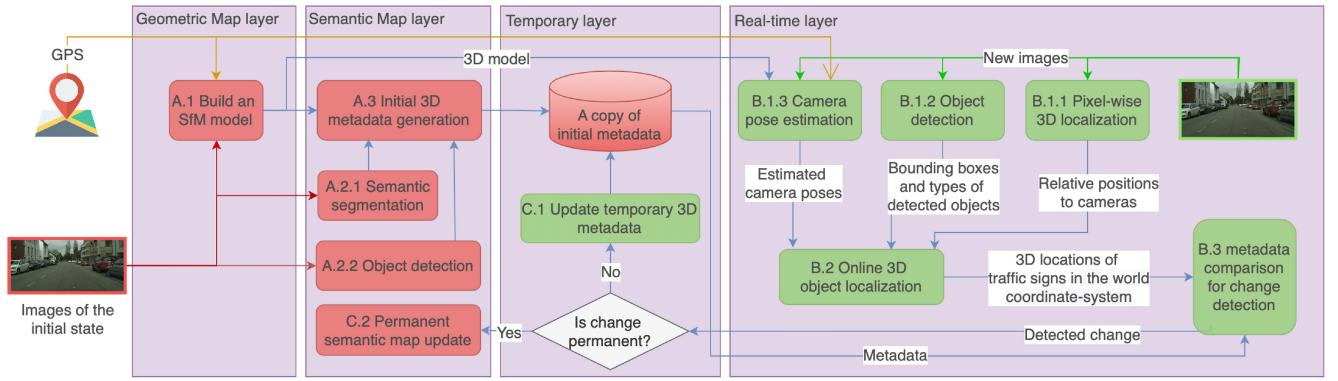


Fig. 1. Architecture of a multilayer map and the pipeline of creating and updating the map based on crowdsourced visual data. Blocks in red demonstrate the generation of the initial map, whereas those in green show the update of the map using newly collected data.

assign to them an appropriate class label. In the case of semantic segmentation, a class label is assigned to each pixel in an image. However, multiple objects of the same class are not recognized as separate objects unless a more complex form of semantic segmentation—i.e., instance semantic segmentation—is performed. Different from semantic segmentation, object detection provides an individual bounding box around each detected object. In case the boundaries of objects are not precisely defined, it may reduce the accuracy of the following object localization step. For example, if a segment that represents a traffic sign by accident covers part of a building from the background, the location of the traffic sign may be set to the location of the building, which is away from the ground truth.

Our system combines both approaches as object detection spots different instances of the same object individually while semantic segmentation gives a more precise boundary around the detected object. The algorithms are later used to detect traffic signs in images on a pixel-wised level and localize them in the 3-D map with the other methods discussed in Section III. By projecting the points of the point cloud back to the images that generated the SfM model—thus, transforming the points back into pixels—and running the objection detection and semantic segmentation neural networks on these images, we can classify each individual pixel and consequently its corresponding 3-D point from the point cloud. In this work, we use the deep learning-based semantic segmentation solution, Seamseg, proposed by Porzi et al. [11]. As for the object detection, we use the method proposed by Lu et al. [12], (SSDResNet), available in the TensorFlow Object Detection API. The details of both methods are given in Sections IV and VII.

III. SYSTEM OVERVIEW

HD maps commonly have multiple layers, where each layer serves a specific purpose and has a distinct structure [13]. We propose a pipeline for creating and updating a four-layer 3-D map from crowdsourced dashcam videos—as illustrated in Fig. 1. The four map layers consist of: 1) a geometric map layer which stores the raw point cloud generated by the SfM; 2) a semantic map layer which stores the metadata with

object semantics; 3) a real-time layer for recurrent change detection; and 4) a temporary layer that processes temporary changes. The layers are described in more detail in the following sections.

The pipeline consists of two stages, the generation of the initial map and the update of the map using newly collected data. The initial stage creates static information for the geometric and semantic map layers, while the update stage extracts dynamic map information from visual data in real time in order to detect changes in the environment by comparing its current state to the state stored in the semantic map layer.

A. Geometric Map Layer

The pipeline starts from step A.1 which reconstructs a 3-D point cloud from 2-D images using SfM. The output of the SfM pipeline, as described in Section II, includes the 3-D points, the camera pose of each image registered into the point cloud, as well as its camera extrinsic matrix \mathbf{T} and camera intrinsic matrix (also called calibration matrix) \mathbf{K} . Assuming an ideal pinhole camera model, the camera projection matrix \mathbf{P} is given as $\mathbf{P} = \mathbf{K}\mathbf{T}$. All this information is stored in the Geometric Map Layer as simple text files. The geometric map data remains unchanged unless there is a significant change in the road infrastructure. The creation of the 3-D point cloud with the SfM pipeline is part of the process of understanding the initial state of the environment. This process is detailed in Algorithm 1, defined in Section IV.

B. Semantic Map Layer

The images used for creating the point cloud are also utilized for detecting objects in the environment by the use of deep learning-based semantic segmentation (at step A.2.1) and object detection (at step A.2.2) neural networks. The output of step A.2.1 includes pixelwise semantic segmentation predictions, while that of step A.2.2 includes bounding boxes of detected objects (e.g., traffic signs) along with their corresponding classes. These outputs are used as input for step A.3 by segmenting the point cloud, where each segment of the point cloud represents a class of objects. The semantic information of static objects is stored as metadata of the point cloud at the Semantic Map Layer in the following text-based

Algorithm 1: Our Algorithm for Initial Metadata Generation. The Functions *GeoRegistration*, *GetImagesWherePointIsSeen*, *GetPixelsFromPoint*, and *GetGPSOfPoint* Are Available in COLMAP. *GetImagesWherePointIsSeen* Finds All the Images Where a Given Point Is Seen. *GetPixelsFromPoint* Finds the Pixel Coordinates in a Given Image Corresponding to a Given Point. *GetGPSOfPoint* Returns the GPS Location of a Given Point

Input: Images of the region I , GPS_coordinates, distance threshold for clustering traffic signs T_D

```

metadata = List() # create an empty list of items
point_cloud = COLMAP(images) # (Step A.1)
point_cloud = GeoRegistration(GPS_coordinates) #
(Step A.1)
for each image in  $I$  do
     $SS_{image} = \text{SemanticSegmentation}(\text{image})$  #
(Step A.2.1)
     $OD_{image} = \text{ObjectDetection}(\text{image})$  # (Step A.2.2)
for each point  $p$  in point_cloud do
    p_images = GetImagesWherePointIsSeen( $p$ )
    probabilities = zeros
    for each image in p_images do
        px, py = GetPixelsFromPoint( $p$ , image)
        probabilities +=  $SS_{image}[px, py]$ 
    point_class = argmax(probabilities)
    if point_class is traffic sign
        tf_probabilities = zeros
        for each image in p_images do
            px, py = GetPixelsFromPoint( $p$ , image)
            tf_probabilities +=  $OD_{image}[px, py]$ 
        tf_class = argmax(tf_probabilities)
        GPS_p = GetGPSOfPoint( $p$ )
        metadata.append([GPS_p, tf_class])
    metadata = K-Means(metadata, num_clusters) #
    num_clusters to satisfy Eq. 1
    Save metadata
Return True # if code was run successfully

```

format: (latitude, longitude, class name, object color, and date detected). The GPS coordinates of each object are obtained by geo-registering the point cloud. In our case, since we are only focusing on traffic signs, the class name of the static object includes information concerning the type of traffic sign. A copy of the 3-D metadata generated at step A.3 is saved also at the Temporary Layer. Algorithm 1 details this task of further understanding the initial state of the environment.

C. Temporary Layer

Initially, the Temporary Layer stores an exact copy of the metadata generated at step A.3. When changes in the environment are detected, they are stored in the Temporary Layer by modifying the metadata to represent the current state of the environment. The metadata of the Semantic Map Layer is only updated (refer to step C.2) when a change becomes permanent. We deem that a change is permanent if it has been observed

by a certain number of vehicles of the crowdsourcing application throughout a certain number of days. There does not exist standard criteria for identifying permanent changes. In practice, our pipeline could allow users/applications to define customized rules for classifying changes into permanent ones. Such rules can employ a simple threshold of time for which the change has to be observed to be deemed as permanent or a minimum amount of observations by different vehicles. In this article, you focus on detecting changes, leaving the classification of permanent changes for future work.

D. Real-Time Layer

The real-time layer takes care of change detection in three steps. After the data have been collected in real time, they are passed for the latter processing which—depending on the available computing resources—may not be performed in real-time. The first step is to run three different algorithms on input images: 1) the pixelwise 3-D localization (step B.1.1); 2) the object detection (step B.1.2); and 3) the camera pose estimation (step B.1.3). These algorithms can be run in parallel since they do not depend on each other. Step B.1.1 calculates the relative 3-D position to the camera for each pixel of the image. Step B.1.2 outputs the bounding boxes and classes of objects detected from each image. Step B.1.3 estimates the camera pose using the SfM model created in step A.1. Since the point cloud is geo-referenced, the camera poses are converted automatically into coordinates in the world coordinate system (WCS). The second step (i.e., step B.2) is to calculate the 3-D locations of the detected objects in the WCS and to generate new 3-D metadata for the point cloud accordingly. The third step (i.e., step B.3) is to compare the newly generated metadata with the latest version stored at the temporary layer. If there exists any difference, a change is detected. The copy of metadata at the temporary layer will be updated accordingly (refer to step C.1).

In our system, deep learning has been applied to implement the semantic segmentation algorithm of step A.2.1, as well as the object detection algorithm present in steps A.2.2 and B.1.2. These deep-learning models are trained independently before the initial stage. They do not need to be retrained unless the application or domain has changed. For example, if a model has been trained to classify traffic signs in one country, to work in another country—i.e., a new domain—it may need to be fine-tuned on a data set, including specific traffic signs of that country.

IV. INITIAL 3-D METADATA GENERATION

This section describes the process of the initial 3-D metadata generation (step A.2) as illustrated in Fig. 2(a). The images from the initial state of the environment used for the sparse point cloud generation (step A.1) are segmented using the semantic segmentation (step A.2.1) neural network proposed by Porzi et al. [11]—whose architecture consists of 50 layers built on the ResNet convolutional neural network (CNN) [14]. The output of the neural network represents a pixelwise semantic prediction, visualized in Fig. 3(b), with 65 urban street output classes. It consists of a text-based file that stores the

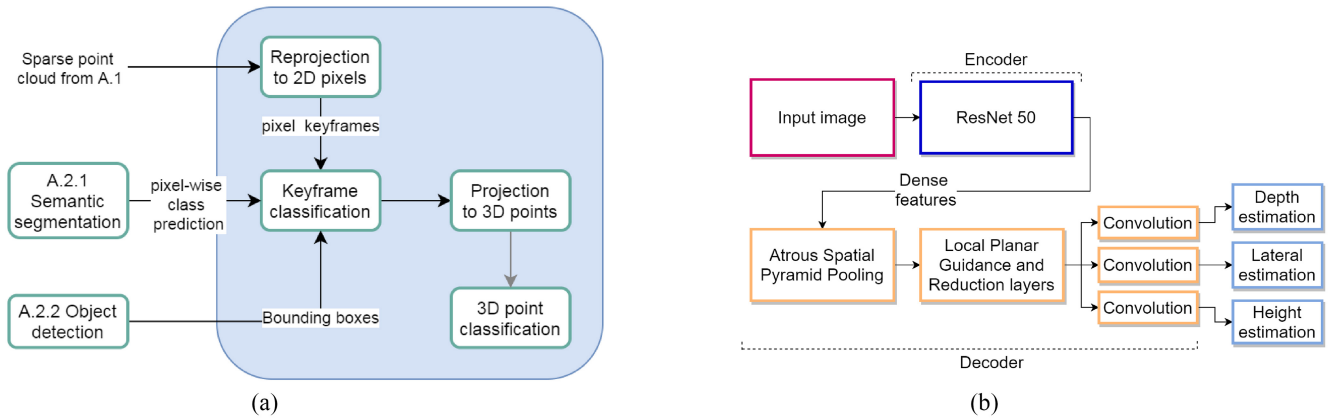


Fig. 2. (a) Workflow of the initial 3-D metadata generation (step A.3 in Fig. 1). (b) Network architecture of pixelwise 3-D localization based on BTS [4].

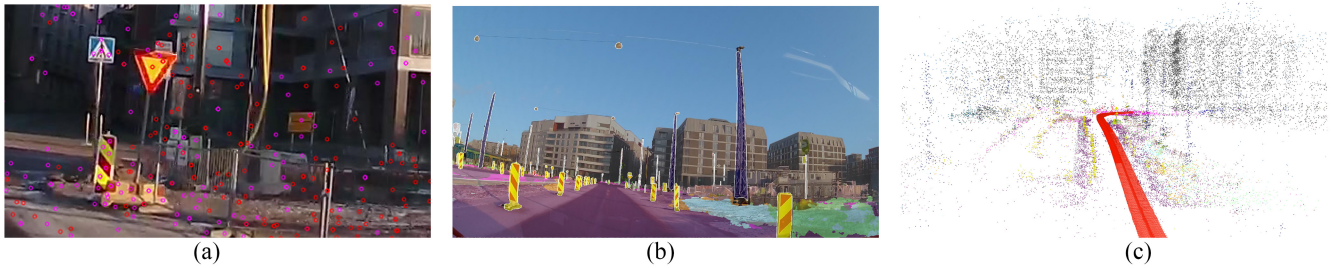


Fig. 3. (a) Example image and keypoints extracted from the image. The pink keypoints represent the ones utilized for 3-D point generation, while the red ones have not been registered in the point cloud. Note that the image has been cropped from the original one to highlight the keypoints. (b) Visualization of the results of the semantic segmentation by generating an image where each pixel takes the color that represents its class of maximum probability. The input image is also overlaid to facilitate the visualization. (c) Results of the point cloud segmentation. The camera poses predicted by the SfM pipeline are shown in red color in Fig. 3(c).

probability of the 65 classes for each pixel in the input image, thus, being similar to the input image itself with the exception that there are 65 channels instead of the three RGB channels.

For each 3-D point in the sparse point cloud, COLMAP provides a list of k images that observe it as well as the pixel coordinates—named image keypoints and shown as pink dots in Fig. 3(a)—where it is observed. Hence, given a 3-D point, we obtain k semantic segmentation predictions corresponding to the images that observe the point. The predictions consist of a probability—or confidence level—for each class considered in the semantic segmentation network. The 3-D point is then assigned the class of the highest average probability value. The files that store the point cloud are then modified to include the segmentation by colors. Fig. 3(c) shows the semantically segmented point cloud with different colors for points of distinct classes.

The results of the semantic segmentation do not include the type of traffic sign. Therefore, we utilize the object detection neural network (step A.2.2) to obtain this information and include it in the point cloud. The reasons for combining semantic segmentation with object detection are two-folded. First, combining the class predictions of the two methods improves the generalization—this is known as ensemble learning [15]. Second, the bounding boxes surrounding the detected objects often contain lots of space and may cover some objects that belong to different classes. Since semantic segmentation provides pixelwise prediction, by calculating the intersection

of each bounding box and the corresponding segment, more precise boundaries of objects can be obtained. Section VII-B2 discusses the choice of the objection detection algorithm as well as its training. Additional details on the semantic segmentation are given in Section VII-B1.

After the information on the type of each traffic sign is stored in the point cloud, all points that represent traffic signs are extracted from the point cloud into a text-based file that, for each point—stores its GPS location and its traffic sign type. Since a traffic sign can consist of a multitude of points, a clustering algorithm is executed to group all points that pertain to the same traffic sign into a single point located in the center of the cluster. The clustering is performed with the use of K -means clustering and K -means++ initialization. The sensitivity of K -means++ on the initialization of the cluster centers is in our case negligible. This is because the distance between clusters is appreciably larger than the size of the clusters and outliers are extremely rare. This also signifies that K -harmonic means clustering [16]—which is designed to be robust to the sensitivity of the cluster initialization—is unlikely to provide any advantage.

We denote as c the number of clusters given as input to the K -means clustering algorithm, D_i as the distance from the center of the cluster i to the furthest point pertaining to the cluster (note that D_i is a function of c), and T_D as the maximum allowed distance D_i . We propose to set the number of clusters equal to the minimum integer value above zero that does not

Algorithm 2: Our Change Detection Algorithm Based on the Utilization of the Camera Pose Estimation, Object Detection, and Modified BTS Neural Network

Input: image, GPS_coordinates, range of distance U from camera to traffic signs, radius R_T to search for matching traffic signs in the metadata
 pose = PoseEstimation(image) # (Step B.1.1)
 OD = ObjectDetection(image) # (Step B.1.2)
 D = ModifiedBTS(image) # (Step B.1.3)
for each detected traffic sign T_s **in** OD **do**
 px, py = GetCenterOfBoundingBox(T_s)
 if $D[px, py] \leq U$
 GPS_ T_s = GetGPSForPoint(p) # (Eq. 3)
 T'_s = closest traffic to T_s and of same type in the metadata # (Step B.3)
 if T'_s does not exist or distance from T_s to $T'_s \geq R_T$ # (Step B.3)
 Report change on the Temporary Map Layer # (Step B.3)
Return True # if code was run successfully

violate the following constraint (1). The output of the clustering algorithm is the metadata—i.e., a set of points representing each of the traffic signs, including their GPS location and type. The choice of T_D is discussed in Section VII. Algorithm 1 summarizes the process of generating the metadata

$$\max_i D_i < T_D. \quad (1)$$

V. CHANGE DETECTION

The newly collected video/images are processed in three steps (i.e., step B.1.x, step B.2, and step B.3) online to detect potential changes in the environment based on a comparison with the metadata. step B.1.1 consists of obtaining the camera pose with respect to the WCS. The same method for object detection as utilized in step A.2.2 is also used, here, in step B.1.2 to produce bounding boxes of traffic signs and to identify their type. Step B.1.3 applies monocular depth, lateral, and height distance estimation to gather the relative positions with respect to the camera of all pixels in the image. Since these steps are independent of each other, they can be executed in parallel.

Step B.2 utilizes the estimated camera poses, the bounding boxes with the types of detected traffic signs, and the relative pixel positions to the camera. These three inputs are processed to obtain the 3-D locations of the detected traffic signs in the WCS. With this, for each traffic sign, we search for matching traffic signs of the same type within a specified radius in the copy of metadata stored in the temporary layer. If there is a mismatch (e.g., there did not exist such a traffic sign earlier), a change is reported and the copy of metadata at the temporary layer can be updated accordingly. Algorithm 2 summarizes the change detection procedure.

In this section, we explain in detail how to estimate camera poses (step B.1.1), how to calculate pixelwise relative location with respect to the camera (step B.1.1), and how to convert

it into 3-D object locations (step B.2) in WCS (e.g., GPS locations of traffic signs).

A. Camera Pose Estimation

In our pipeline, the camera pose estimation (i.e., estimation of camera position and orientation in the WCS) is performed in two distinct manners depending on the presence of traffic signs. In the first method, given an image I_t at a time instant t , if a traffic sign is detected in it, the position and orientation of its camera (with respect to the WCS) are obtained by registering it into the point cloud using SfM with the custom feature matching [2]. In custom feature matching, the image pairs to be matched can be defined in a custom manner. In our case, we opt to match the image I_t with the nearest image (in terms of Euclidean distance calculated with GPS coordinates) that was utilized to build the point cloud— I'_t . Using the described custom feature matching significantly reduces the possibility of matching failure in places where the amount of visual features is insignificant.

In the second method, the camera position of I_t is obtained directly from its GPS coordinates, whereas the orientation of I_t is calculated by assuming that its orientation with respect to I'_t equals that of I_{t-1} with respect to I'_{t-1} . In practice, this assumption means that the camera pose is always fixed with respect to the car,—i.e., there is no relative movement between the car and the camera—and the car follows the exact orientation of the road reconstructed in the point cloud. Since in general this assumption holds, this represents a good approximation for finding the camera pose. Mathematically, this can be expressed with the following equation:

$$R_t = R_{t-1} \cdot R_{t-1}^T \cdot R'_t \quad (2)$$

where R_t is the rotation matrix of the image I_t with respect to the absolute reference frame—which represents its camera orientation—and similarly for R_{t-1} , R_{t-1}^T , and R'_t . The subscript T denotes the transpose operation.

The first method is utilized when the orientation of I_{t-1} is not known or periodically to avoid the accumulation of errors of the second method. The second method is a significantly computationally cheaper alternative to the first method with lower but still good accuracy. Compared with deep-learning-based camera pose estimation, such as PoseNet [17], obtaining camera poses from the SfM pipeline typically requires smaller computational costs and excludes the need for an extensive training data set. The evaluation of the camera pose estimation is given in Section VIII-A.

B. 3-D Object Localization

For each image, we first apply object detection to create a bounding box that covers the traffic sign in question (step B.1.2). After that, we select the pixel at the center of the bounding box to represent the location of the object, and calculate its 3-D coordinates relative to the camera following a process called pixelwise 3-D localization (step B.1.1). The process is implemented using CNN, as described in Fig. 2(b). We use the state-of-the-art monocular depth estimation network, BTS [4] because its architecture is designed to predict depth

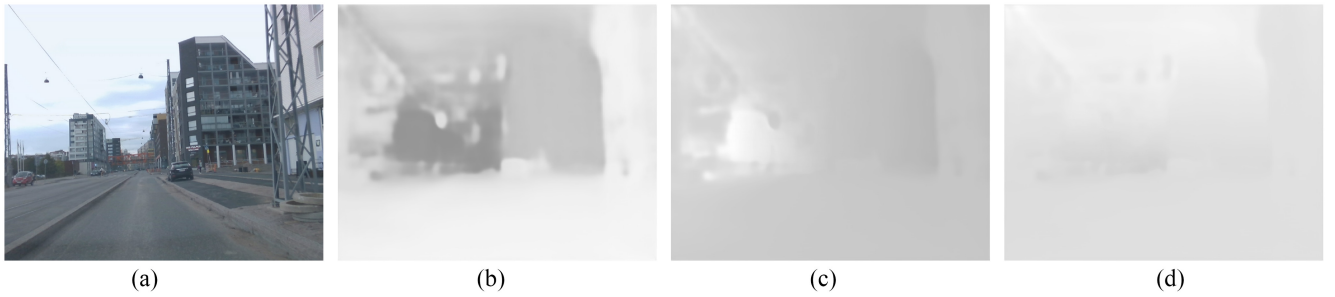


Fig. 4. Illustrative examples of the output of the BTS network. (a) RGB image. (b) Depth prediction. (c) Lateral prediction. (d) Height prediction.

from images which is a similar application to our method. BTS can be replaced in the future with a better CNN architecture designed for depth estimation, if available.

Originally, the BTS network produces a single channel, which is the pixelwise depth prediction. We have modified the output layer to produce pixelwise output with three channels representing x , y , and z coordinates in a 3-D space. To train the network, we create a labeled image set, including all the images used for creating the point cloud at step A.1. The three channels of the labels in this case represent the x , y , and z coordinates of a 3-D point (in the SfM point cloud) with respect to the camera. Note that since the point cloud is sparse, the images are also sparsely labeled—i.e., some pixels may not have annotations. During the training, these nonlabeled pixels are masked to not to influence in the minimization of the loss function. For each input image, the SfM pipeline outputs an estimated camera pose, the coordinates of the 3-D points, and the 2-D keypoints that have been used to generate those 3-D points. Also due to the nature of the sparsity in the data set, we opted to fine-tune the model instead of training it from scratch. In the fine-tuning, the encoder and the early layers of the decoder had their weights frozen.

An example is given in Fig. 4 to illustrate the outputs of the network. In Fig. 4(b)–(d), the color of each pixel represents the distance (either depth, lateral, or height) between the point depicted by the pixel and the camera. The whiter pixels indicate closeness to the camera. Note that the pixels which do not belong to any keypoint are ignored in the loss during the training. The output of the BTS network is projected into the WCS according to (3). Since the radius of the Earth in meters was utilized during geo-registration, all the calculated distances are also presented in meters

$$P = R^T \cdot B + C \quad (3)$$

where P is the 3-D position of a certain pixel in the image in the WCS, R is the rotation matrix from the image reference frame to the WCS, C is the position of the image in the WCS, and B is the vector representing the lateral, height, and depth distances of the point with respect to the image.

We may detect the same traffic sign from several images, which means we may get multiple predicted locations for a single traffic sign. To obtain a more accurate location of the traffic sign, we first filter out some noisy predictions by limiting the minimum and maximum distances from the camera to the detected traffic sign. The reason for setting the range

of distance comes from the fact that the pixelwise 3-D localization algorithm tends to perform better when the traffic sign is within a certain range of distance to the camera. Therefore, the minimum and maximum distance thresholds are decided based on the performance of the modified BTS neural network on the test set (Section VII-B3). After that, we calculate the center of the predicted locations within a specified radius and set it as the location of the detected traffic sign.

VI. DATA SETS

The data for the training and evaluation of the performance of the different system building blocks in Sections VII and VIII were collected from two different sites: 1) in a residential area and 2) around a university campus. All the collected data sets have extremely different appearances due to differences in weather or lighting conditions, camera models utilized, and camera placements. The distinction in weather and lighting conditions is a result of a data collection that took place on different days and even different seasons of the year. Notice that the purpose of having this variability across the data sets is to simulate a potential crowdsourcing use. The length of roads present in the residential area summed up to 2.4 km, whereas on the campus, this number was 4 km. In all the recordings, the vehicles were driving mostly at a speed within the range of 20–30 km/h. The main reason is due to the fact that the vehicle transited urban areas with pedestrians passing through. Also, the speed of the vehicle surpassed 40 km/h at times. Hence, all components of the system work at a higher driving speed. However, from the SfM mapping perspective, it is expected to collect visual data with a lower moving speed when the vehicle is making a sharp turn (i.e., over 90° rotation), since quick scene changes can cause reconstruction errors.

Table I summarizes the seven data sets, which in total included 23 057 images. As summarized in Table I, seven data sets, including in total 23 057 images, were generated. A Garmin 55 dashcam was utilized on Day 1, an Intel RealSense D435 on Day 2, and an iPhone 12 Pro Max on Day 3. The camera setups are visualized in Fig. 5(d)–(f). All the cameras were placed to face roughly the same direction, which results in an overlap in the views for data sets with more than one view. The video resolution was set to 1920x1080, while the frame rate to 30 FPS, which was later downsampled by decimation to 10 FPS as a higher frame rate is not required given that the vehicle speed is relatively low.

TABLE I
DATA SET DESCRIPTION INCLUDING TRAJECTORIES FOR DATA COLLECTION, CAMERA SETUP,
AND THE NUMBER OF IMAGES COLLECTED FROM EACH TRAJECTORY

Index	Date	Trajectory	# of views	# of images of each trajectory	Camera model
I	Day 1	5b (E, F) - campus	single	460, 576	Garmin 55
II	Day 1	5a (B) - residential	single	725	Garmin 55
III	Day 1	5a (A) - residential	double	1525	Garmin 55
IV	Day 2	5a (A) - residential	single	706	RealSense D435
V	Day 1	5a (C, D) - residential	triple	945, 2874	Garmin 55
VI	Day 3	5c (G, H) - campus	single (central)	656, 2000	iPhone 12 Pro Max
VII	Day 3	5c (G, H) - campus	single (left)	656, 2000	Garmin 55



Fig. 5. Data collection trajectories and camera placements. Fig. 5 (a) illustrates the trajectories A, B, C, and D in a residential area. Fig. 5 (b) and (c) depict trajectories E, F, and G on a university campus. Fig. 5 (d)–(f): the different configurations of camera placements: single view versus double view versus triple view.

The camera recordings took place on February 22 2019 (Day 1), December 2 2019 (Day 2), and February 2021 (Day 3). During data collection, the vehicle went through four trajectories in the residential area [Fig. 5(a)] and four trajectories on the campus [Fig. 5(b) and (c)]. For the residential area, only trajectory A (Data Sets III and IV) suffered changes from February 2019 to December 2019, hence, they will be utilized for evaluating the change detection. In the campus area, changes have not been captured. However, our change detection algorithm will still be utilized in trajectory G (Data Sets VI and VII) to confirm the absence of changes. Fig. 6 illustrates an example of a change in the environment. The remaining data are assigned to the training of the pixelwise 3-D localization system component (step B.1.1).

VII. PERFORMANCE OF THE INITIAL 3-D METADATA GENERATION

Our system generates the initial 3-D metadata of SfM point clouds at step A.3. The accuracy of the generated metadata depends on the accuracy of the SfM point cloud built at step A.1 as well as the accuracy of semantic segmentation (step A.2.1) and object detection (step A.2.2). In this section, we evaluate the accuracy of each building block of the Geometric Map and Semantic Map layers and analyze how it affects the overall accuracy of the generated 3-D metadata. We conducted all the experiments on a system running Ubuntu 18.04 and powered by four NVIDIA GTX 1080 Ti with 11 GB of RAM each and two Intel Xeon Gold 6134 CPUs.

A. SfM-Based 3-D Reconstruction

As discussed in Section II, we implement 3-D reconstruction based on COLMAP [2], and employ the vocabulary-tree-based method and sequential method for feature matching [9]. We create a sparse point cloud for data set III and data set IV—described in Section VI—and perform geo-registration on it. Data set III was built with vocabulary-tree-matching,



Fig. 6. Example of scenario change in the residential area. The image on the left is taken from data set IV, whereas the one on the right is from data set III. Notice that the traffic signs inside the red rectangles are only present in one state of the environment.

























whereas sequential matching was employed to data set IV. The quality of the generated point cloud can be reflected in the accuracy of the estimated camera positions (step B.1.3) and pixelwise localization (step B.1.1), which in general affects the accuracy of 3-D object localization (step B.2) in the process of change detection. In addition, point cloud quality affects the localization accuracy of the metadata in step A.3.

We take data set III as an example to evaluate the accuracy of the camera pose estimations produced at step A.1. Data set III includes video collected from two dashboard cameras at 30 fps and RTK-based positioning data at 1 Hz (i.e., the maximum sampling rate of the RTK device in use). Since RTK provides centimeter-level positioning accuracy, the positions derived from RTK samples are considered ground truth in this case. The estimated camera positions are compared with RTK measurements in Fig. 7(a). Since the sampling rate of RTK positioning data is lower than the frame rate of the video, we calculate the distance error of each estimated camera pose as the distance from its closest RTK position. The median distance error is 7.09 m, with a standard deviation of 4.96 m. Concerning the bias caused by inconsistent sampling rates, the actual errors may be lower than the ones we calculated.

As highlighted using a red box in Fig. 7(a), an accumulated drift appears at the upper end of the trajectory. Such a drift can be significantly reduced if: 1) the SfM reconstruction is

TABLE II

TRAFFIC SIGNS INSTALLED ALONG TRAJECTORY A (LEFT) AND TRAJECTORY G (RIGHT). THE TRAFFIC SIGN NAMES PRESENTED WITH AN ASTERISK * ARE THOSE INCLUDED IN OUR EXPANSION OF THE MAPILLARY DATA SET

Class	Image	Class	Image	Class	Image
information-pedestrians-crossing-g1		pass-on-either-side-g1		information-parking-g1	
shared-path-pedestrians-and-bicycles-g1*		maximum-speed-limit-40-g1*		warning-roadworks-g1*	
information-dead-end-except-bicycles-g1*		maximum-speed-limit-30-g1*		regulatory-stop-g1	
complementary-obstacle-delineator-g2		no-motor-vehicles-g6*		regulatory-keep-left-g1	
complementary-one-direction-right-g1		regulatory-no-parking-g1		warning-t-roads-g1*	
regulatory-end-of-maximum-speed-limit-40-g1*		regulatory-roundabout-g1		regulatory-keep-right-g1	
regulatory-no-pedestrians-or-bicycles-g2*		regulatory-turn-right-g1		regulatory-yield-g1	
junction-with-a-side-road-perpendicular-right-g1*		regulatory-no-stopping-g1		complementary-buses-g1	

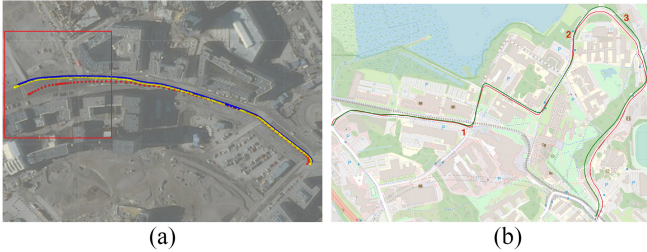


Fig. 7. (a) Estimated by SfM camera positions of each image in data set III (camera 1 in blue and camera 2 in yellow) versus the ground truth obtained with RTK technology (in red). (b) Camera poses of data set VI estimated by SfM (green) versus the path extracted from Google Maps for comparison (red solid line), the red numbers specify the locations of example images in Fig. 8.

performed with the data that has a loop closure; 2) the area of reconstruction is not too large, for example, one building block with a loop closure making up around 600 m [see region C and D in Fig. 5(a)]; and 3) the general recommendations for the SfM reconstruction are met—such as good visibility, scenery rich with features, and sufficient overlap between the camera views. Except for the drift inside the red box, the estimated camera positions align well with the ground truth. However, we report that the alignment depends on the selected GPS coordinates for the geo-registration. In this case, those were selected in order to align properly the road outside the red box in Fig. 7(a).

Fig. 7(b) compares the camera poses estimated by the SfM reconstruction (data set IV) with the GPS path extracted from Google Maps. In a similar way to reconstruction in Fig. 7(a), the geo-registration of the region has been intentionally selected to align more precisely with the upper side of the region. That side has a greater number of visual features, contrary to the rest of the path, which presents an environment with significantly fewer features due to large numbers of trees and snowy roads. Notice that the quality of reconstruction with sequential matching (data set IV) is on the same level as that with vocabulary-tree matching (data set III). However, this does not mean both methods are equally good. In fact, we were unable to generate a point cloud for data set IV with vocabulary-tree matching.

To understand the reasons that caused vocabulary-tree-based matching to completely fail, we refer to Fig. 8. Fig. 8(a) exhibits distinguishable visual features due to the presence of

buildings, which helps in a more accurate SfM reconstruction (the location of the camera pose is identified by the number 1 in Fig. 7(b)). However, Fig. 8(b) and (c) have very similar appearances characterized by densely located trees and snowy roads, even though they are from different locations [see their locations identified by the numbers 2 and 3 in Fig. 7(b)]. The similarity in visual appearance present in different regions is the cause of the failure in vocabulary-tree matching. In this type of matching, images with similar appearances are matched to each other and, when the images are from very distinct regions, the matching process is unsuccessful and fails to produce a point cloud. Sequential matching avoids this issue by matching images in a manner that preserves their spatial positioning. Therefore, we conclude that sequential matching is the most appropriate method for our case.

B. Traffic Sign Detection and Localization

1) *Semantic Segmentation on 2-D Images*: We implement the semantic segmentation component (step A.2.1), described in Section II-B, based on the Seamseg architecture proposed by Mapillary [11]. We utilize the model trained and tested by the authors on the Mapillary data set [18], which achieves 50.4% Intersection over Union (IoU) [11]. A visual example of the model performance can be seen in Fig. 3(b).

2) *Object Detection*: TensorFlow Object Detection API is utilized for the traffic sign detection component, as was mentioned in Section II-B. As for the neural network architecture, the SSD Resnet-50 FPN pretrained on COCO data set was selected [12] due to a good tradeoff between the accuracy and speed. The training set is composed of approximately 16 000 annotated images of traffic signs from the Mapillary data set [18]. However, since certain traffic signs present in our test regions differ significantly from those available in Mapillary (see Table II), we expanded the data set by including additional 4000 annotated images focused on the traffic signs exclusive to our test regions. Half of these additional annotated images consist of real images collected in different regions, but in the same country. The other half is composed of images where the traffic signs of interest were artificially overlaid on generic background images. The 24 traffic sign classes present in the test regions are illustrated in Table II. Overall, 20 000 annotated images formed the data set, where

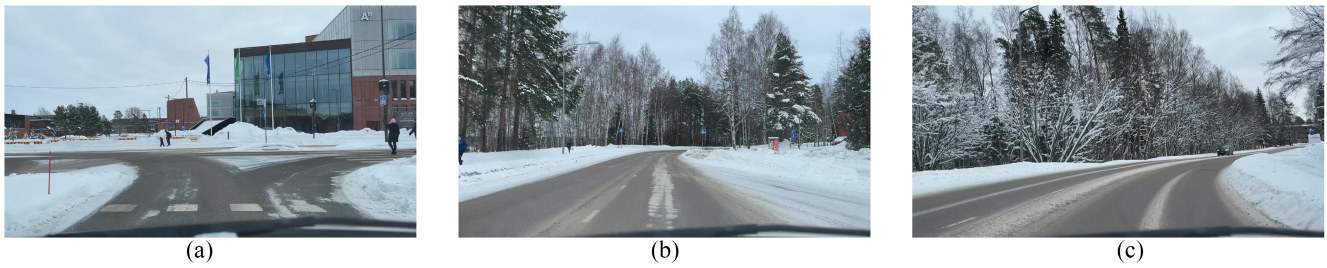


Fig. 8. Examples of images with a sufficient and insufficient number of recognizable features. While (a) has enough recognizable features due to the presence of buildings, (b) and (c) suffer from the lack of features since they are mostly covered with trees.



Fig. 9. Examples of mistakes in the objection detection algorithm. On the first row, informative traffic signs of pedestrians crossing are mistaken as being of another type. On the second row, a few more examples of misclassification including an instance where tree branches are mistaken as a traffic sign.

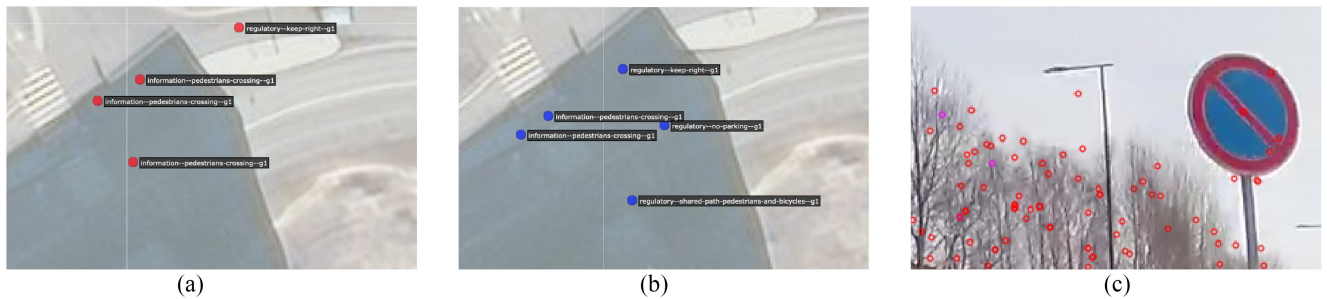


Fig. 10. In region G, the “no parking” traffic sign was not able to be localized. Notice its absence in (a) as compared with the ground truth in (b). (c) Shows the traffic sign in question with unmatched features across images in red circles on it. Due to these unmatched features, the triangulation of the traffic sign was not able to be performed, thus, resulting in its absence in the metadata. (a) Prediction. (b) Ground truth. (c) Unmatched features.

500 of them were utilized for validation with the rest being assigned to training. The training consisted of 30 epochs.

As a test set, we manually labeled the traffic signs in 97 images of region G on Day 3 (i.e., data set VI). Note that we purposely test the object detection on images recorded from a different camera than the images that formed the training set. This allows for a more realistic measurement since it is expected that crowdsourced data are taken from distinct cameras. The object detection algorithm demonstrated an mean average precision (mAP) of 0.518 at IoU threshold of 0.4, which is a fairly good result given the circumstances of different weather, lighting, and camera conditions. Fig. 9 illustrates examples of mistakes by the object detection algorithm. It is observed that when the “pedestrians crossing” traffic sign is located at a far distance from the car or in case it is at a tilted angle with respect to the direction of movement of the car, the algorithm mistakes it for another traffic sign. Other examples included in Fig. 9 indicate that the algorithm can confuse traffic signs of similar appearances—which is the case of the roadworks and perpendicular road junction traffic signs. Since the confidence score of these misclassifications happens to be below 0.4, we opted to discard any detection whose score is below this threshold value.

3) *3-D Object Localization*: To evaluate the accuracy of the object localization of our initial metadata generation method (Section IV, step A.3), we utilize 25 traffic signs (see Table II) installed along the trajectories A and G as examples. The locations of these traffic signs provided by the geo-referenced and semantically segmented SfM point cloud are compared to their ground truth. We have selected empirically $T_D = 12$ m (Algorithm 1). Lower values of T_D may result in the same traffic sign being erroneously identified as two or more distinct traffic signs, whereas higher values of T_D may cluster distinct traffic signs together.

At first, it has to be noted that the system is unable to locate the traffic signs when there are insufficient 2-D image keypoints in the intersection between a semantic segment and the corresponding bounding box of the object in question. In the example shown in Fig. 10 of region G, the “no parking” traffic sign has not been localized due to the quality of the SfM reconstruction [see the prediction in Fig. 10(a) versus the ground truth in Fig. 10(b)]. In Fig. 10(c), unmatched features are shown in red color, whereas the matched ones are illustrated in purple. Even though features for the traffic sign in question have been extracted, these are not matched with the features seen in other images. Thus, being unable to triangulate

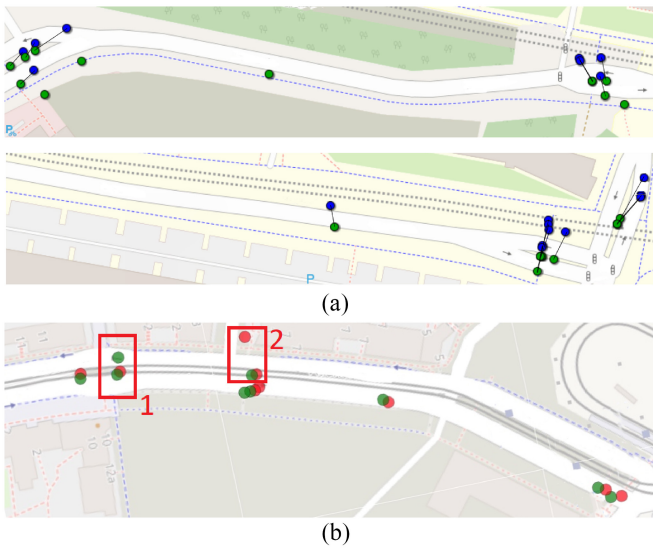


Fig. 11. Traffic sign localization using the initial metadata generation method. Due to its length, the campus area is divided into two sections. Note that the road on Section 2 is exactly the continuation of that on Section 1. (a) Estimated locations of detected traffic signs (in blue) versus the ground truth (in green) for the campus area. Top: Section 1, bottom: section 2. (b) Estimated locations of detected traffic signs (in red color) versus the ground truth (in green color) for the residential area.

the position of the traffic sign and, as a consequence, resulting in the absence of the “no parking” traffic sign in the metadata. An explanation for this is that the images utilized in the SfM reconstruction have been collected in a sequential manner in only one driving direction which might not provide a sufficient overlap between the views. Still, this demonstrates a realistic scenario.

In region A [inside rectangle 1 of Fig. 11(b)], it can be observed that the absence of one traffic sign (note the presence of two green circles compared to one sole red circle). The traffic sign that was unable to be localized is of type pedestrian walk [Fig. 12(d)]. Its absence is explained by the fact that there are two traffic signs of the same type located very near to each other. Our system mistakes these two instances as only one. It is an effect of the distance threshold for clustering T_D . To compensate for the localization errors of the SfM point cloud, this threshold imposes that instances of the same type are clustered as one sole instance. A more accurate localization by the SfM method is one solution for this issue. However, a more ingenious approach would consist of tracking each traffic sign across multiple images to identify which points in the point cloud belong to the same traffic sign and, thus, differentiate between such closely located objects of the same class.

In rectangle 2 of Fig. 11(b), one traffic sign was localized twice (notice the presence of two red circles compared to only one green circle). This traffic sign was located once in an accurate location, but also far from the ground truth due to the inaccuracy of the SfM reconstruction. One possible solution is to increase the number of viewing angles and the image quality of the data set used for the SfM reconstruction. Some traffic signs have not been localized due to object detection confidence for specific classes being lower than the threshold. An example of this is the “junction with a side road” traffic

sign as shown in Fig. 9. Reducing the threshold is not a solution as it would result in a multitude of erroneous predictions. In future work, training the objection detection model with a larger data set would help to reduce this error.

Even though the metadata may miss traffic signs, this can be corrected in the change detection stage (described in Section V). The detection of an unseen traffic sign in the initial stage will trigger the correction of the metadata by the change detection method. In the future, the work can be extended to include pedestrians or cyclists carrying smartphones and filming the environment to improve the localization of traffic signs along the pedestrian path, since sometimes these signs might be quite far from the main road or occluded by trees, street poles or other objects (see examples in Fig. 12). This specific case has been the cause of 2 errors out of a total 5.

In quantitative measures, our system locates 8 out of 9 traffic signs along the driving direction in the residential area, and 16 out of 20 traffic signs in the university area. Compared with the ground truth, in the campus area, the median distance error is 10.4 m with a standard deviation of 2.9 m. As for the residential area, the median distance error and standard deviation were measured to be 3.6 and 1.4 m, respectively. We posit that this disparity between the campus and residential area is due to the fact that the former consists of a more featureless region (i.e., devoid of objects such as a building), which affects negatively the feature extraction and feature matching processes in the reconstruction of the model. We also look into the causes of the localization errors in successful cases. These errors have been mainly caused by the errors in the SfM-based 3-D reconstruction and geo-registration. In our experiments, the data was collected from vehicles driving through the trajectories following one direction rather than two. We used up to three cameras facing the front for data collection. More cameras facing different directions would help reduce the error in the point cloud segmentation by improving the accuracy of depth prediction for the 3-D points. In addition, the GPS coordinates used for the geo-registration were selected manually by visually analyzing the image and approximating its location using Google Maps and Google Street Maps. This always introduces a certain degree of human error.

VIII. EVALUATION OF CHANGE DETECTION

A number of changes have been observed in terms of traffic sign deployment in the test areas between Day 1 and Day 2, and between Day 3 and Day 4. In this section, we evaluate the performance of the change detection method using 4 data sets. Data set III (Day 1) and data set IV (Day 2) are used to compare the changes in the residential area, while data set VI (Day 3) and data set VII (Day 4) are for evaluation of the system in the campus area. Particularly, we will measure the accuracy of step B.1.1, step B.2, and step B.3, respectively.

A. Camera Pose Estimation

Our camera pose estimation (step B.1.1.) is comprised of two distinct methods. The first method—corresponding to the registration of the image into the point cloud—presents an accuracy of 7.79 m with respect to the camera pose estimation.

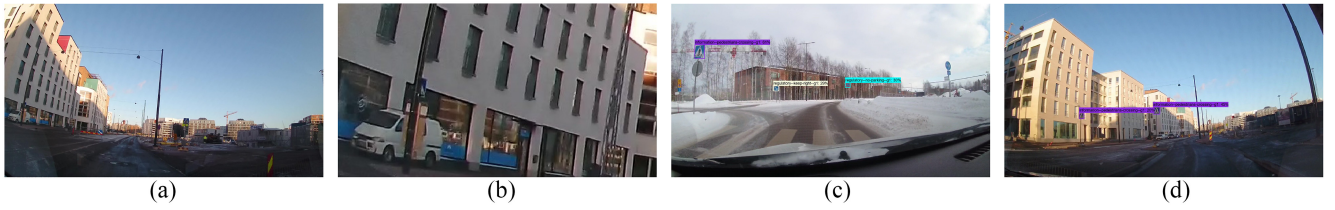


Fig. 12. Example of traffic signs which are not clearly visible due to a long distance from the camera or occlusion. Fig. 12 (a): original image of Fig. 12 (b) (data set III in Table I); Fig. 12 (b): zoomed-in cropped image of Fig. 12 (a); Fig. 12 (c): an example with Pedestrians and bicycles traffic sign (data set IV in Table I); Fig. 12 (d): an example of two traffic signs of the same type located closely to each other (data set III in Table I).

TABLE III
DEPTH, LATERAL, AND HEIGHT ERRORS OF THE PIXEL-WISE 3-D LOCALIZATION METHOD FOR DIFFERENT RANGES OF GROUND-TRUTH DISTANCES. ABBREVIATIONS: D FOR DEPTH, H FOR HEIGHT, AND L FOR LATERAL

Range	0 — 10m			10 — 25m			25 — 50m			50 — 100m		
	D.	H.	L.	D.	H.	L.	D.	H.	L.	D.	H.	L.
Abs. error	3.33m	1.01m	1.73m	3.55m	0.99m	1.54m	6.42m	1.22m	2.55m	12.58m	1.66m	4.42m
Rel. error	0.62	3.12	3.21	0.24	1.53	2.12	0.19	2.18	2.60	0.19	2.50	1.45

This result is the same as the one presented in Section VII-A since the point cloud itself is also used, here, to estimate the camera position. As the ground truth for the camera orientation is not available for any image, it is not possible to measure the accuracy of COLMAP when estimating the camera orientation of the images.

The second method of camera pose estimation presented an average error of 6.22 m with respect to the camera position. This result was obtained by comparing directly the position of the car given by the GPS device with that provided by the RTK system. The data utilized for the measurement of the camera position error comprised the entire residential area. Again, as mentioned in Section VII-A, due to the inconsistent sampling rates of the GPS (30 Hz) and the RTK system (1 Hz), the actual error may be lower. Regarding the camera orientation estimation—given by the assumption given in (2)—we have measured its validity by comparing the results given by it with those provided by COLMAP during the image registration. The results showed that the assumption—on which the second method is based—is able to estimate the camera orientation with approximately 6.18 degrees of error—which represents a fairly good result. The measurement of the camera orientation error utilized the data from the campus area.

B. Pixel-Wise 3-D Localization

As described in Section V, pixelwise 3-D localization (step B.1.3.) is built on top of the BTS [4] network. The network is initialized with the weights trained with Kitti depth data set [19]. After that, we fine-tuned the network with 13 032 samples collected from different regions of the environment [A–D and F–G in Fig. 5(a) and (b)] for 30 epochs. The validation set was comprised of 706 samples from region A collected on a different day than those samples from the same region present in the training set. Finally, the test set consisted of 576 samples collected in region E.

Table III shows the error results on the test set on two distinct metrics: 1) the absolute error and 2) the relative error. The absolute error is defined as the mean (calculated on all pixels of all images) of the absolute error (4), whereas the

relative error is the mean (calculated on all pixels) of the ratio between absolute error and the ground truth

$$\text{Absolute error} = \sum_{\text{image}} \sum_{\text{pixel}} \frac{|\text{PRED}| \text{GT}|}{n_i \cdot n_p} \quad (4)$$

$$\text{Relative error} = \sum_{\text{image}} \sum_{\text{pixel}} \frac{|\text{PRED}/\text{GT}| - 1}{n_i \cdot n_p} \quad (5)$$

where PRED and GT stand for the prediction of the neural network and its ground truth, respectively. Also, n_p and n_i refer to the number of pixels in an image and the number of images in the test set, respectively.

It is observed that the absolute errors grow as the ground truth distance increases, whereas the relative errors decrease. Compared to the original BTS [4], our relative error in depth is approximately 2.2–4.5 times larger. This is due to the following reasons. Note that we have applied data augmentation methods, such as color, gamma, and brightness changes to reduce over fitting. However, they were not effective in significantly reducing the error. The potential solutions to improve the accuracy of the SfM point cloud will be discussed in Section X.

- 1) Most of the weights in the neural network are shared between three tasks of distance estimation: a) depth; b) height; and c) lateral. This worsens the performance of the depth estimation since the learned features need to be more generic for a better estimation of the three distances together.
- 2) Our data set is comprised of samples captured with three different cameras, whereas in the original work of BTS [4] the authors trained separate neural networks for the samples of each camera available. It is known that differences in the camera directly affect the performance of computer vision algorithms.
- 3) Our data set is sparse, i.e., most of the pixels in the images are unlabeled. This is due to the fact that the point clouds are also sparse.
- 4) The ground truth is formed by estimations provided by the SfM reconstruction, thus, containing errors that

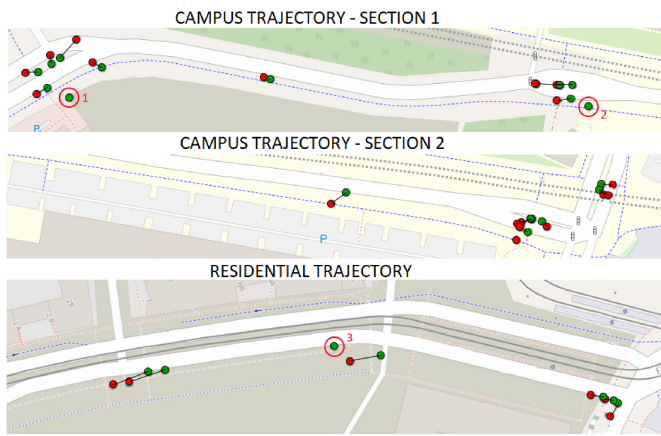


Fig. 13. Estimated traffic sign locations of the real-time layer (in red) versus the ground truth (in green). The lines linking a green circle to a red one indicate a match between the ground truth and the real-time layer. The encircled traffic signs represent unmatched cases, each of which with a number as the identifier.

affect both the training of the neural network and the accurate evaluation of the trained model on the test set.

C. Traffic Sign Localization

We perform the online traffic sign localization (step B.2.) on both campus and residential trajectories. Data sets IV and VII were utilized for the former and the latter trajectories, respectively. Since the accuracy of the 3-D object localization is dependent on the distance between the camera and the object (Table III), we propose to utilize a distance threshold (defined as U in Algorithm 1). This signifies discarding traffic signs located at a distance greater than the threshold. By carefully selecting the value of the distance threshold, it is possible to achieve better results. According to Table III, selecting higher values of U lead to larger estimations errors. On the other hand, since some traffic signs may only be seen at a specific distance range, lowering U excessively may culminate in missed traffic signs—thus, resulting in incomplete metadata. We have found empirically that setting U to 25 m provides a good equilibrium in this tradeoff.

Residential Trajectory: Fig. 13 illustrates the estimated traffic sign locations and their corresponding ground truth for the residential trajectory. It is observed that our system can locate 6 out of the 7 traffic signs. The missed traffic sign was detected with low confidence—below the considered threshold in the object detection neural network—and, thus, discarded [Fig. 14(a)]. We hypothesize that the reason for this is due to the insufficient number of labeled examples of construction work in the training data set of the object detection neural network. Reducing the confidence score threshold is not a solution, since it results in the appearance of multiple erroneous detections like the one shown in Fig. 14(a) (bottom right) where construction barricades are wrongly detected as a traffic sign of construction-work type. With respect to distance metrics, the traffic signs were located with a median distance error of 9.1 and 5.1 m. A detection accuracy of 83.3% is seen for this trajectory.

Campus Trajectory: Fig. 13 illustrates the estimated traffic sign locations and their corresponding ground truth for

the campus trajectory. Two of the traffic signs present in the ground truth and without matches from the real-time layer are of the same type: the shared path between pedestrians and bicycles (cases 1 and 2). These mismatches are due to the failure of the object detection network in detecting this specific type of traffic sign [Fig. 14(a)]. Again, we posit that a larger training set for the objection detection neural network could solve this issue. Also, we consider these minor failures since these traffic signs are not addressed to the driver. Overall, a median error of 5.27 m and a standard deviation of 2.08 m are observed for the trajectory. The considerable difference between these values for the campus trajectory compared with the residential trajectory is caused by the fact that the images of the campus trajectory are more aligned with those that composed the training set of the 3-D object localization neural network.

These are solid results considering that only monocular images were used to predict the location of the changes and the labels of the training data set utilized to train the prediction neural network were automatically generated. Although submeter-level accuracy was shown to be possible in change localization [20], it requires the utilization of a set of additional sensors, such as LiDAR, IMU, vehicle speed sensors, and highly accurate positioning solutions. We envision that, whenever a change is detected and localized, a cautionary area of a predetermined radius encompassing it is created. This signifies that, in real life, changes can be indicated as an area—instead of a point—to incorporate the inaccuracy of the change localization algorithm. In the circumstance of an autonomous vehicle entering this area, an immediate switch from automatic to manual operation mode is required. Considering this concept, highly accurate change localization results are not required.

D. Change Detection and Localization

When a traffic sign is detected and localized in the second stage (real-time layer, step B.2), the system searches around its location for matching traffic signs in the current copy of the metadata at the temporary layer (step B.3). Since there exist localization errors in the metadata and in the online traffic sign location, we define another distance threshold (denoted as R_T in Algorithm 1) for reducing the false-positive errors in change detection. The newly detected traffic sign is considered to match an existing one in the current copy of metadata if they are of the same type and the distance between them is below the threshold R_T . When a mismatch happens, a change is detected and reported. To determine this threshold, we consider that there can be an error of up to 10 m in the localization of the traffic signs in the metadata as well as in the real-time layer. Also, taking into consideration a worst case scenario where the total localization error of a traffic sign in the metadata compared to the real-time layer is doubled up, we determine the threshold R_T value to be 20 m. Higher values of R_T may result in erroneous matches, whereas lower values of R_T may result in absence of matches and indicate a nonexistent change in the environment.

Residential Trajectory: Fig. 15 illustrates the arrangement of the traffic signs (ground truth) in the residential region

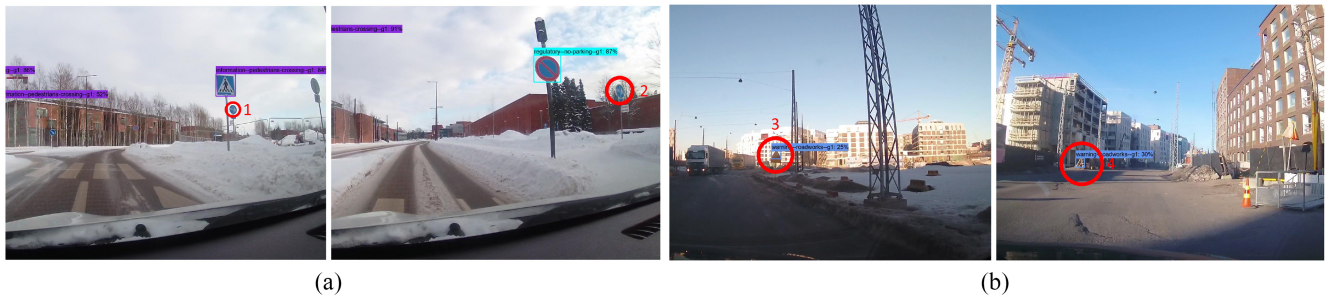


Fig. 14. Analysis of the traffic sign detection in the campus and residential areas. (a) *Left*: traffic signs of the shared path between the pedestrians and bicycles that were not detected due to their distance from the camera. (a) *Right*: The insufficient training data for this traffic sign in the object detection neural network. (b) *left*: the sole undetected traffic sign classified with a 25% confidence score—below the confidence score threshold. (b) *Right*: a case where barricades are wrongly detected and classified as a roadworks traffic sign with 30% confidence score. This explains why there should exist a threshold of confidence scores to consider. (a) Campus area—data set IV in Table I. (b) Residential area—data set VII.

before and after the environment has suffered changes. Fig. 16 illustrates the environment as seen by the real-time layer, whereas Fig. 19 shows the confusion matrix related to the detection of changes in the environment from the comparison with the metadata. It can be observed that the removal of four traffic signs was correctly detected by our algorithm. Moreover, the permanence of six traffic signs after the scene change was also accurately identified. Note that the road-works traffic sign was not detected by the real-time layer. This is also true for the metadata, hence, no change with respect to this specific traffic sign is detected—which is also the case observed in the ground truth before and after the scene change. The confusion matrix exhibits that our change detection algorithm reached the maximum possible performance.

Campus Trajectory: Fig. 17 illustrates the arrangement of the traffic signs (ground truth) in the campus region. Even though this area does not suffer any change, our change detection algorithm is required to identify the absence of changes. Similar to the residential area, Fig. 18 illustrates the environment as seen by the real-time layer, with the confusion matrix depicted in Fig. 19. The comparison with the metadata reports the appearance of two traffic signs—no-parking and T-junction—and the removal of a yield traffic sign. This represents three erroneous cases of change in the environment. However, notice that the appearance in the real-time layer of traffic signs that were erroneously not included in the metadata—which is the case of the no-parking and T-junction traffic signs—signifies that the metadata can be corrected. It is also observed that 16 out of the 20 traffic signs were correctly reported as objects of no change. Overall, the change detection method exhibits an accuracy of 85%.

E. Change Detection and Localization

To measure its latency, we executed the real-time layer on an Intel i7-11700F processor clocked at 2.50 GHz and an NVIDIA RTX 3070 8-GB GPU. Among the processes in the real-time layer, the pixelwise 3-D localization (step B.1.1) and the object detection (B.1.2) were the most computationally expensive ones by lasting approximately 0.10 and 0.06 s per image, respectively. The camera pose estimation (B.1.3) performed by the first method (discussed in Section VIII-A)

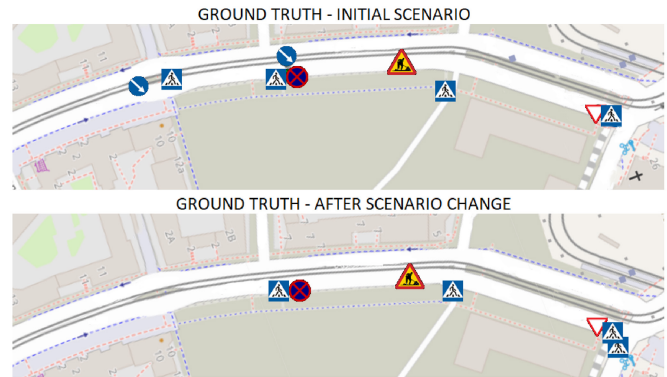


Fig. 15. Ground truth of the before (top) and after (bottom) scenario change in the residential area.



Fig. 16. Detected and localized traffic signs of the real-time layer along the residential area.

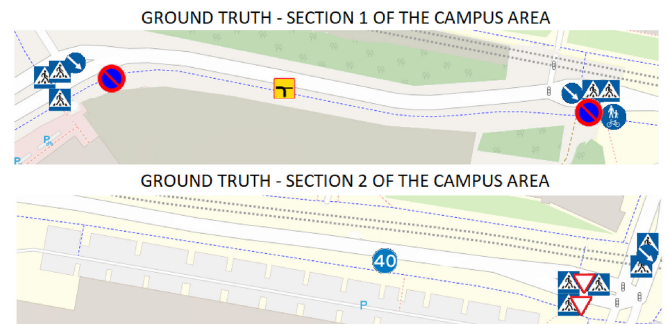


Fig. 17. Ground truth of the campus area. Note that this area does not suffer any change.

took approximately 0.05 s per image. The second method of camera pose estimation, since it involves very few operations of multiplications, was shown to be of negligible cost as well as the online 3-D object localization (B.2) and metadata

TABLE IV

COMPARISON OF OUR WORK WITH PREVIOUS STUDIES ON CHANGE DETECTION. ALTHOUGH THE STUDIES OF YEW AND LEE [23] AND PALAZZOLO AND STACHNISS [24] CAN LOCALIZE CHANGES IN A 3-D SPACE, THE AUTHORS HAVE NOT EVALUATED THE PERFORMANCE OF THIS METRIC IN QUESTION. THE CAMERA/VEHICLE SPEED IS NOT DISCUSSED OR REPORTED IN PREVIOUS STUDIES. HOWEVER, IT IS EXPECTED TO BE SIMILAR TO OURS SINCE THE IMAGES WERE COLLECTED IN URBAN AREAS WITH A REGULAR VOLUME OF TRAFFIC. IN BRIEF, OUR PIPELINE IS ABLE TO ACHIEVE HIGH-QUALITY CHANGE DETECTION WITH ONLY RGB IMAGES AND GPS DATA AS INPUT. OUR PIPELINE IS ALSO CAPABLE OF LOCALIZING 3-D CHANGES WITH AN ACCURACY LOWER THAN 7 M, WHICH IS A GOOD RESULT GIVEN THAT LiDAR, RTK, OR IMU DATA ARE NOT UTILIZED

Method	Image views requirement	Requires manual labeling	Required sensors	For changes of stationary objects	Performs 3D change localization	Accuracy of 3D change localization	Accuracy of change detection
Alcantarilla <i>et al.</i> [21]	Panoramic	Yes	RGB images, IMU, GPS	Yes	No	-	Moderate
SceneChangeDet [22]	Single	Yes	RGB images	No	No	-	Moderate
Yew and Lee [23]	Single	No	RGB images	Yes	Yes	Not Reported	Moderate
Hu <i>et al.</i> [7] Wang <i>et al.</i> [5] Santos <i>et al.</i> [6]	Single	Yes	RGB images	Yes	No	-	High
Palazzolo and Stachniss [24]	Multiple	No	RGB images	Yes	Yes	Not Reported	High
Jo <i>et al.</i> [20]	Single	No	RGB images, LiDAR, IMU, RTK, steering wheel angle, vehicle speed	Yes	Yes	Under 1m	High
Zhang <i>et al.</i> [25]	Single	No	RGB images, IMU, GPS, steering wheel angle, vehicle speed	Yes	Yes	Under 6m	High
Ours	Single	No	RGB images, GPS	Yes	Yes	Under 7m	High

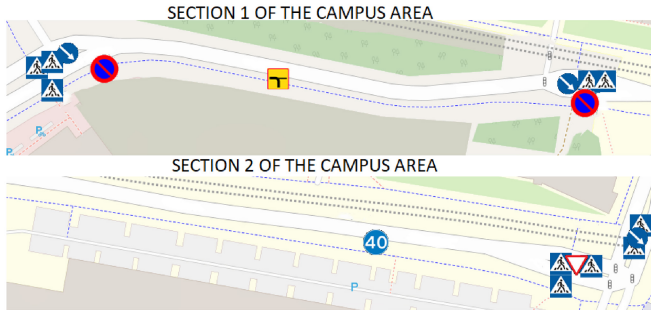


Fig. 18. Detected and localized traffic signs of the real-time layer along the campus area.

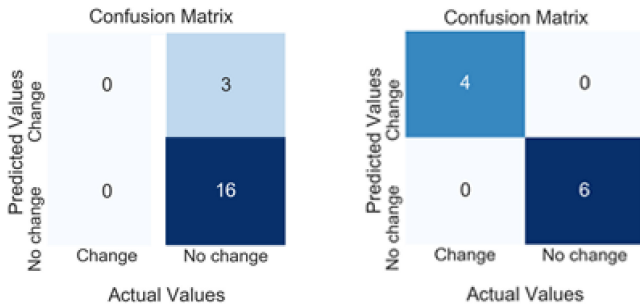


Fig. 19. Confusion matrices detailing the detection results for the campus area (left) and the residential area (right).

comparison for change detection (B.3). Overall, the real-time layer was able to process on average 5 FPS. In our tests, steps B.1.1, B.1.2, and B.1.3 were executed in a sequential manner. Since they are independent of each other, in future work, it is possible to implement them in parallel on the GPU for better resource utilization.

IX. RELATED WORK

In this section, we first review the latest works in the literature related to change detection while comparing their design

choices with our pipeline—Table IV provides an overview of the comparisons. Since semantic mapping (metadata generation and maintenance) and monocular depth estimation are also part of our pipeline, we also describe the literature on these topics.

A. Change Detection and Localization

Palazzolo and Stachniss [24] proposed an approach for detecting changes between a 3-D model representative of the initial state of a given environment and a small sequence of images of the current state of the environment. Their approach requires that the locations of all images with respect to the 3-D model are precisely known. In addition, it requires dense point clouds as input, which are challenging to generate from images captured from vehicle-carried cameras with limited viewing angles. The authors utilized images from Google Street Maps that were taken using a complex array of a multitude of cameras capturing the same scene from numerous perspectives. Such availability of different perspectives of the same scene is infeasible in crowdsourcing scenarios, where only one forward-pointing camera is used. Our pipeline differs from the aforementioned work by not requiring multiple image views, thus, being more appropriate to crowdsourcing applications.

Alcantarilla *et al.* [21] developed a change detection system for urban scenarios. First, their approach consisted in obtaining two dense 3-D reconstructions, each representing states of the environment at different times. After that, an accurate geo-registration on both point clouds was performed, which allowed the alignment of both models. This alignment was used to obtain pairs of images (each image from a different state of the environment) taken at the same location. For each pair of images, a dense convolution neural network—trained via supervision—was employed to detect changes. Their method proved effective in detecting changes under different lighting and seasonal conditions. Again, the limitation of the method is the requirement for 3-D dense reconstructions. That is, dense point clouds must be constructed for both

the initial and current states of the environment. The authors utilized panoramic image views to obtain dense point clouds. The major distinction of our pipeline is being able to detect changes with a sparse point cloud of the initial state of the environment and only a few images of the current state. Also, due to the high computational complexity, the execution of this method in real time may be infeasible for certain hardware specifications.

Rosen et al. [26] proposed a feature-based model of environmental change detection and incorporated it into graphical SLAM techniques. The method was evaluated with simulated data only. An improvement was proposed in [27] where the authors modified ORB-SLAM—a state-of-the-art visual SLAM algorithm—to enable scene change detection by incorporating a customized persistence filtering as in [26]. Instead of detecting scene changes, such as the removal or addition of objects in the environment, the method is limited to detecting changes in individual map points. Also, the authors conducted experiments only in small environments. The algorithm's performance in complex urban environments is unknown. Instead of individual map points, our pipeline detects changes at a more concrete level—i.e., objects in the environment such as traffic signs.

We have conducted tests with SceneChangeDet [22]—a monocular change detection neural network. The method failed to detect changes in stationary objects as traffic signs and only worked in cases of short-lasting changes, such as the presence of certain cars in a parking slot, which is naturally undesirable for our purposes. Compared to the previously mentioned methods on change detection [21], [24], our pipeline requires only a sparse point cloud of the initial state of the environment and excludes the need for complex arrangements of cameras as in [24]. Moreover, differently from [26] and [27], it detects changes related to concrete environmental structures—such as traffic signs—and is specifically designed for complex urban environments. Unlike [22], the detected changes consist solely of modifications of stationary objects in the environment. Furthermore, SceneChangeDet [22] only focuses on detecting changes without localizing them.

Zhang et al. [25] proposed the fusion of SLAM-based algorithms with semantic segmentation to generate a semantic point cloud. To detect changes in the environment, the authors proposed to denoise, cluster, and vectorize the point cloud before matching the semantic point clouds from the initial state with that of the current state. A recursive Bayesian depth filter combined with a camera pose estimation from motion sensors (IMU) is also utilized to obtain the 3-D positions of points in the point cloud. Since the lateral and height distances are not considered in their work, the estimations of these 3-D positions can be negatively affected. Moreover, since instance segmentation is not utilized, it is not possible to detect changes, such as the modification of the meaning of a traffic sign present in a certain region. A third distinction of this method, when compared to ours, is their use of various sensing technologies, such as steering wheel angle, vehicle speedometer, and IMU. In crowdsourcing scenarios, such sensor modalities are impractical.

He et al. [7] introduced an end-to-end deep neural network solution—named Diff-Net—for change detection from 2-D images. Their approach works by projecting HD map elements—i.e., traffic signs—onto the camera pose creating a rasterized image with such elements. The rasterized and the original images are utilized as inputs to the neural network to infer map changes. Similarly, in TransCD [5], Wang et al. proposed a transformer-based scene detection algorithm to spot changes in pairs of images. Santos et al. [6] approached the problem of detecting changes in pairs of images with a multiscale CNN architecture. Our pipeline distinguishes from the three previous methods by localizing changes in 3-D space, instead of only at a 2-D level, and by performing camera pose estimation, instead of assuming that the camera poses are known a priori.

Yew and Lee [23] proposed a method for change detection by comparing point clouds created from SfM. Since geo-registering the point clouds before comparing their points would result in large errors due to inaccurate geolocation information and possible drifts introduced by the SfM, the authors proposed a deep-learning-based nonrigid registration that allowed them to compare the point clouds more accurately. As mentioned in the Section I, in a crowdsourced visual data case, it is too costly to create point clouds for each new data collection. Therefore, the proposed method by Yew and Lee [23] is not appropriate for crowdsourcing. Our method, on the other hand, does not require the reconstruction of the environment each time it is scanned for changes.

Jo et al. [20] created a change detection and localization algorithm based on SLAM and utilizing the Dempster–Shafer evidence theory. The authors reported a detection accuracy above 90% and submeter localization accuracy. However, their system requires the utilization of burdensome additional devices, such as RTK positioning system, LiDAR, IMU, wheel speed sensor, steering angle sensor, and radar. Therefore, its use for crowdsourcing is impractical. Our method only requires the utilization of a camera and a GPS device, which can often be found included in common commercial dashboard cameras.

B. Semantic Mapping

McCormac et al. [28] fused semantic information into dense point clouds of indoor environments created with simultaneous localization and mapping (SLAM) algorithms. The authors employ a deconvolutional semantic segmentation network architecture that provides pixelwise class predictions. Similar to our work, these predictions are projected into the point cloud utilizing the tracked camera poses provided by the SLAM algorithm. Their approach requires RGB-D image sequences, whereas in our work RGB images suffice.

Rosinol et al. [29] created Kimera, an open-source C++ library for real-time SLAM with semantic information. Kimera uses mono, stereo, and inertial data to generate a semantic and metric dense reconstruction of the environment by incorporating off-the-shelf tools for 2-D semantic segmentation of images. The authors only presented results for simulated

indoor environments. Therefore, the performance of the system in large-scale outdoor environments is unclear.

Previous works [30], [31] have tried to combine LIDAR point clouds with semantic segmentation on 2-D images for detecting and locating landmarks in a 3-D environment. In [30], probabilistic methods were used to construct semantic HD multilayer maps. We apply a similar approach to reduce the manual efforts of map data generation. Instead of combining LIDAR and RGB images, our system only requires input from RGB cameras and focuses on an under-explored scenario: automatic change detection.

Nakajima et al. [32] focused on enabling real-time incremental semantic point cloud creation at the same time as providing accurate results. Their approach assigns class probabilities to entire portions of the point cloud instead of to each individual surfel, this notably reduces time complexity. In our approach, since the metadata creation is executed offline, there is not a need for real-time semantic segmentation of point clouds.

C. Monocular Depth Estimation

The Monodepth2 method [3] estimates depth from a sequence of RGB images. It is a self-supervised training method that is possible to be fine-tuned without a labeled data set. The performance of the model trained on the Kitti data set [19] was unsatisfactory since the boundaries of the estimated depth were blurred. The performance after training the model on our data sets has not demonstrated satisfactory results either.

Lee et al. [4] designed a neural network architecture based on the encoder–decoder scheme to perform depth estimation from monocular images. In their architecture, based on the local planar assumption, the authors proposed a novel layer—named local planar guidance (LPG)—located in the decoder block of the network. The experiments have shown that their method outperforms previous ones by a significant margin in diverse metrics, providing state-of-the-art results.

The limitation of such monocular depth estimation methods is that they do not support 3-D localization, including the height and lateral information. In this article, we propose an end-to-end 3-D localization network to solve this problem. Since [4] has shown satisfactory results in depth estimation, we modified it by extending the neural network architecture to support lateral and height estimations.

X. DISCUSSION

In this section, we discuss the limitations and potential improvements for future work organized into four main topics: 1) crowdsourcing and data augmentation; 2) geo-registration; 3) road topology; and 4) real-time performance.

Crowdsourcing and Data Augmentation: Our pipeline is composed of several individual components that must work in synergy for accurate change detection results. Each of the components is prone to errors that accumulate through the pipeline affecting negatively the final results. At the beginning of the pipeline, an accurate point cloud generation at step A.1 requires multiple views of the same objects with sufficient

overlap between the views and possibly at different distances to the objects. This requirement is, especially, difficult to fulfill when reconstructing large-scale environments since the vehicle is restricted to following the road, thus, generating images from limited viewpoints and distances. In environments with the presence of large buildings, the camera on the vehicle—regardless of its position—is only able to capture part of the building—most likely a plane wall—which is insufficient for an accurate visual-based feature matching. In some situations, even ultrawide cameras may not be able to capture the scene with sufficient characteristics for satisfactory feature matching. In future work, we plan to include crowdsourced data from cars, pedestrians, and cyclists to increase the number of different perspectives, thus, improving the point cloud generation. In addition, with crowdsourcing, we consider including voting from different observers of the objects in the scene that will be weighted to determine if the change at a particular timestamp is present.

The object detection, semantic segmentation, and pixelwise object localization methods are powered by deep learning, which are data-hungry algorithms. Therefore, the lack of rich data sets available for their training directly impacts their performance. Both semantic segmentation and object detection neural networks were trained on Mapillary and, even though it is undoubtedly the most complete data set for our purposes, its creators [18] point that it is still insufficient to train an end-to-end neural network and requires some extra tuning. To improve on this, large synthetic data sets can be employed together with domain adaptation [33]. It is possible to use an instance segmentation model instead of the combination of object detection and semantic segmentation. In fact, instance segmentation was devised with the purpose of combining semantic segmentation with object detection in the same neural network. In terms of performance, there is no clear advantage of instance segmentation over its counterparts. While instance segmentation allows for combining two neural networks (one for object detection and the other for semantic segmentation) and possibly provides lower resource consumption, it requires large volumes of training data whose labeling process is more laborious than semantic segmentation object detection. Training data availability is a challenging problem in instance segmentation [34] and optimizing data efficiency is still an open research question, especially, for our case scenario—i.e., traffic sign categories. In this tradeoff, we have opted for a higher resource consumption with separate neural networks for object detection and semantic segmentation; such an increase in computational resources is not significant to affect the real-time operation of our pipeline. As for the case of the pixelwise object localization, since it is trained with data directly extracted from the point cloud generated at step A.1, for a better generalization other solutions than increasing the number of views with overlapping regions include: 1) training monocular neural network observing a scene from multiple perspectives for better generalization; 2) having an accurate geo-registration since the scale of the SfM model directly affects the scale of the localization predictions in meters; and 3) having a larger amount of reconstructed point cloud data, by mapping larger area of the environment and by densifying the point cloud.

Geo-registration: It is also a part of the 3-D reconstruction and is essential for accurate change detection. It consists of two steps. First, conversion from geodetic coordinates to Cartesian ones is performed. This procedure inevitably introduces errors—especially, when the geodetic height is unspecified. Then, a similarity transformation is executed whose parameters are defined, such as to minimize the alignment error between the real world and the model’s coordinates. This alignment error can grow higher in largely reconstructed areas with limited viewpoints. To alleviate this problem, we plan to perform multiple geo-registration procedures, each of which is executed independently of the other for a segment of the model. Also, the inclusion of the topology of the region can be utilized to improve the conversion between geodetic and Cartesian coordinates.

Road Topology: It has been challenging to detect a change in case a traffic sign has been moved within a short distance to another location without changing the facing direction. This is because the current design of step B.3 sets a distance threshold and assumes that two traffic signs with the same type detected on different days but close enough to each other (i.e., below the distance threshold) are considered to be identical. In case a sign is shifted from one side of a road to the other, adding the road boundaries and markings to the attributes of metadata may help solve the problem.

Deployment at the Edge of the Internet: We plan to deploy the system in a distributed manner in that the initial point clouds are created in the cloud while the change detection and map update are conducted at the edge of the Internet, such as the computing nodes co-located with cellular base stations or roadside units. By moving computation closer to vehicles, the amount of traffic going through the core network would drop, and more importantly, the transmission latency would decrease, which could help reduce the delay of change detection.

Automatic Parameter/Threshold Selection: Our pipeline requires humans to select appropriately the parameters U , T_D , and R (presented in Algorithms 2 and 1) for best results. Future work could explore ways to eliminate or alleviate this requirement of parameter tuning, thus, increasing the level of automation of our pipeline.

Changes in the Environment: Environment changes only affect one component of our pipeline: the camera pose estimation. As described in Section V-A, this component extracts features from an image of the current state of the system and matches with those stored in the generated point cloud. As long as there exist features in common between image pairs, the camera pose can be accurately estimated. Hence, it is unlikely that changes in building facades or slight changes in road topology can affect the system’s performance. Moreover, our experiments in different seasons of the year have proved that our pipeline works well when the environment/scene changes due to weather conditions. The effect of significant changes in the road topology is left for future work. In future work, our pipeline can implement a module that detects large changes in the road topology by comparing the trajectory followed by the vehicle in real time with the road topology present in the initial state of the environment. If large

differences are seen in such a comparison, the pipeline can utilize the latest collected video data to generate an updated version of the point cloud.

XI. CONCLUSION

In this article, we presented a system for creating and updating a multilayer map for autonomous driving. Our system is partially built on top of other existing methods, e.g., SfM, semantic segmentation, and object detection. Nonetheless, the system brings new functionalities and addresses a number of challenges to enable crowdsourced-based change detection and localization in rapidly changing urban environments. Our solution is able to spot changes in the environment with accuracy above 85% by analyzing the current state of the environment with its previous one having traffic signs as the objects of interest. The results could be further improved in the future by increasing the performance of background technologies in use.

ACKNOWLEDGMENT

The authors would like to thank Kari Tammi and Risto Ojala at Aalto University for the provided equipment for data collection.

REFERENCES

- [1] J. Dong, M. Noreikis, Y. Xiao, and A. Ylä-Jääski, “ViNav: A vision-based indoor navigation system for smartphones,” *IEEE Trans. Mobile Comput.*, vol. 18, no. 6, pp. 1461–1475, Jun. 2019.
- [2] J. L. Schönberger and J.-M. Frahm, “Structure-from-motion revisited,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 4104–4113.
- [3] C. Godard, O. M. Aodha, M. Firman, and G. Brostow, “Digging into self-supervised monocular depth estimation,” in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 3827–3837.
- [4] J. H. Lee, M.-K. Han, D. W. Ko, and I. H. Suh, “From big to small: Multi-scale local planar guidance for monocular depth estimation,” 2019, *arXiv:1907.10326*.
- [5] Z. Wang, Y. Zhang, L. Luo, and N. Wang, “TransCD: Scene change detection via transformer-based architecture,” *Opt. Exp.*, vol. 29, no. 25, pp. 41409–41427, Dec. 2021. [Online]. Available: <http://www.osapublishing.org/oe/abstract.cfm?URI=oe-29-25-41409>
- [6] D. F. S. Santos, R. G. Pires, D. Colombo, and J. P. Pap, “Scene change detection using multiscale cascade residual convolutional neural networks,” in *Proc. 33rd SIBGRAPI Conf. Graph. Patterns Images (SIBGRAPI)*, 2020, pp. 108–115.
- [7] L. He, S. Jiang, X. Liang, N. Wang, and S. Song, “Diff-Net: Image feature difference based high-definition map change detection for autonomous driving,” 2021, *arXiv:2107.07030*.
- [8] C. Wu, “Towards linear-time incremental structure from motion,” in *Proc. Int. Conf. 3D Vis.*, 2013, pp. 127–134.
- [9] J. L. Schönberger, T. Price, T. Sattler, J.-M. Frahm, and M. Pollefeys, “A vote-and-verify strategy for fast spatial verification in image retrieval,” in *Proc. Asian Conf. Comput. Vis.*, 2016, pp. 321–337.
- [10] J. L. Schoenberger. “COLMAP tutorial.” 2023. [Online]. Available: <https://colmap.github.io/tutorial.html>
- [11] L. Porzi, S. R. Bulò, A. Colovic, and P. Kotschieder, “Seamless scene segmentation,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 8269–8278.
- [12] X. Lu, X. Kang, S. Nishide, and F. Ren, “Object detection based on SSD-ResNet,” in *Proc. IEEE 6th Int. Conf. Cloud Comput. Intell. Syst. (CCIS)*, 2019, pp. 89–92.
- [13] M. O. Taş, H. S. Yavuz, and A. Yazici, “Updating HD-maps for autonomous transfer vehicles in smart factories,” in *Proc. 6th Int. Conf. Control Eng. Inf. Technol. (CEIT)*, 2018, pp. 1–5.
- [14] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.

- [15] M. A. Ganaie, M. Hu, M. Tanveer, A. K. Malik, and P. N. Suganthan, "Ensemble deep learning: A review," 2021, *arXiv:2104.02395*.
- [16] B. Zhang, M. Hsu, and U. Dayal, *K-Harmonic Means—A Data Clustering Algorithm*, HP Lab., Palo Alto, CA, USA, 1999.
- [17] A. Kendall, M. Grimes, and R. Cipolla, "PoseNet: A convolutional network for real-time 6-DOF camera relocalization," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2015, pp. 2938–2946.
- [18] G. Neuhold, T. Ollmann, S. R. Bulò, and P. Kotschieder, "The mapillary vistas dataset for semantic understanding of street scenes," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2017, pp. 5000–5009.
- [19] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3354–3361.
- [20] K. Jo, C. Kim, and M. Sunwoo, "Simultaneous localization and map change update for the high definition map-based autonomous driving car," *Sensors*, vol. 18, no. 9, p. 3145, 2018. [Online]. Available: <https://www.mdpi.com/1424-8220/18/9/3145>
- [21] P. F. Alcantarilla, S. Stent, G. Ros, R. Arroyo, and R. Gherardi, "Street-view change detection with deconvolutional networks," *Auton. Robots*, vol. 42, no. 7, pp. 1301–1322, 2018.
- [22] E. Guo et al., "Learning to measure change: Fully convolutional siamese metric networks for scene change detection," 2018, *arXiv:1810.09111*.
- [23] Z. J. Yew and G. H. Lee, "City-scale scene change detection using point clouds," 2021, *arXiv:2103.14314*.
- [24] E. Palazzolo and C. Stachniss, "Fast image-based geometric change detection given a 3D model," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2018, pp. 6308–6315.
- [25] P. Zhang, M. Zhang, and J. Liu, "Real-time HD map change detection for crowdsourcing update based on mid-to-high-end sensors," *Sensors*, vol. 21, no. 7, p. 2477, 2021. [Online]. Available: <https://www.mdpi.com/1424-8220/21/7/2477>
- [26] D. M. Rosen, J. Mason, and J. J. Leonard, "Towards lifelong feature-based mapping in semi-static environments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2016, pp. 1063–1070.
- [27] Z. Hashemifar and K. Dantu, "Practical persistence reasoning in visual SLAM," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2020, pp. 7307–7313.
- [28] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "SemanticFusion: Dense 3D semantic mapping with convolutional neural networks," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 4628–4635.
- [29] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: An open-source library for real-time metric-semantic localization and mapping," 2019, *arXiv:1910.02490*.
- [30] D. Paz, H. Zhang, Q. Li, H. Xiang, and H. Christensen, "Probabilistic semantic mapping for urban autonomous driving applications," 2020, *arXiv:2006.04894*.
- [31] D. Maturana, P.-W. Chou, M. Uenoyama, and S. Scherer, "Real-time semantic mapping for autonomous off-road navigation," in *Field Service Robotics*. Cham, Switzerland: Springer, 2018, pp. 335–350.
- [32] Y. Nakajima, K. Tateno, F. Tombari, and H. Saito, "Fast and accurate semantic mapping through geometric-based incremental segmentation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Dec. 2018, pp. 385–392.
- [33] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2017, *arXiv:1703.10593*.
- [34] W. Gu, S. Bai, and L. Kong, "A review on 2D instance segmentation based on deep neural networks," *Image Vis. Comput.*, vol. 120, Apr. 2022, Art. no. 104401. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0262885622000300>

Aziza Zhanabatyrova received the B.S. degree in computer science from Eastern Mediterranean University, Famagusta, North Cyprus, in 2015, and the M.S. degree in robotics engineering from the University of Genoa, Genoa, Italy, and Jaume I University, Castell de la Plana, Spain, in 2018. She is currently pursuing the Ph.D. degree with Aalto University, Espoo, Finland.

She wrote the master's thesis with ETH Zurich, Zurich, Switzerland, in 2018. Her current research interests include deep learning, computer vision, urban scene mapping, and autonomous driving.

Clayton Frederick Souza Leite received the B.S. degree in mechanical engineering from the Federal University of Pernambuco, Recife, Brazil, in 2015, and the M.S. degree in robotics engineering from the University of Genoa, Genoa, Italy, and Warsaw University of Technology, Warsaw, Poland, in 2018, and the Ph.D. degree in computer science from the Aalto university, Espoo, Finland, in 2023.

He is currently a Postdoctoral fellow with Aalto University. His current research interests include deep-learning-based human activity recognition, deep model compression, and urban scene mapping.

Yu Xiao received the Ph.D. degree in computer science from Aalto University, Espoo, Finland, in 2012.

She is an Associate Professor with the School of Electrical Engineering, Aalto University. Her current research interests include edge computing, wearable sensing, and extended reality.