IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 June 2023, accepted 9 July 2023, date of publication 13 July 2023, date of current version 24 July 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3295212

RESEARCH ARTICLE

Edge-Distributed Fusion of Camera-LiDAR for Robust Moving Object Localization

JOSE AMENDOLA[1], AVEEN DAYAL[2], LINGA REDDY CENKERAMADDI[2], (Senior Member, IEEE), AND AJIT JHA[1]

[1]Department of Engineering Sciences, University of Agder, 4879 Kristiansand, Norway
[2]Department of Information and Communication Technology, University of Agder, 4879 Kristiansand, Norway

Corresponding author: Ajit Jha (ajit.jha@uia.no)

This work was supported in part by the Senter for Forskningsdrevet Innovasjon (SFI) Offshore Mechatronics Project under Grant 237896; and in part by the Indo-Norwegian Collaboration in Autonomous Cyber-Physical Systems (INCAPS) Project through the International Partnerships for Excellent Education, Research and Innovation (INTPART) Program, under Grant 287918.

ABSTRACT Object localization plays a crucial role in computational perception, enabling applications ranging from surveillance to autonomous navigation. This can be leveraged by fusing data from cameras and LiDARs (Light Detection and Ranging). However, there are challenges in employing current fusion methods in edge devices, while keeping the process flexible and modular. This paper presents a method for multiple object localization that fuses LiDAR and camera data with low-latency, flexibility and scalability. Data is obtained from 360° surround view four cameras and a scanning LiDAR distributed over embedded devices. The proposed technique: 1) discriminates dynamic multiple objects in the scene from raw point clouds, clusters their respective points to obtain a compact representation in 3D space; and 2) *asynchronously* fuse the centroids with data from object detection neural networks for each camera for detection, localization, and tracking. The proposed method meets above functionalities with low-latency fusion and increased field of view for safer navigation, even with intermittent flow of labels and bounding boxes from models. That makes our system distributed, modular, scalable and agnostic to the object detection model, distinguishing it from the current state-of-art. Finally, the proposed method is implemented and validated in both indoor environment and publicly available outdoor KITTI 360 data set. The fusion occurs much faster and accurate when compared with traditional non-data driven fusion technique and the latency is competitive with other non-embedded deep learning fusion methods. The mean error is estimated to be $\approx$ 5 cm and precision of 2 cm for indoor navigation of 15 m (error percentage of 0.3%). Similarly, mean error of 30 cm and precision of 3 cm for outdoor navigation of 35 m on KITTI 360 data set (error percentage of 0.8%).

INDEX TERMS Sensor fusion, LiDAR, object localization, embedded system.

NOMENCLATURE

ϵ_{sync}	Time duration of a step for frame synchronization.
$\mathcal{C}^{XY}$	Set of 2D plane projections of all centroids.
$\mathcal{C}_i$	Set of cluster centroids at instant i.
$\mathcal{T}_i$	Set of tracks at instant i.
$\boldsymbol{a}_i^{j,l}$	Compact representation obtained from $\boldsymbol{s}_i^{j,l}$.
$\boldsymbol{b}_i^{j,k}$	Compact representation obtained from $\boldsymbol{f}_i^{j,k}$.
$\boldsymbol{f}_i^{j,k}$	Image frame captured by k-th camera connected at j-th device at a given instant i.
$\boldsymbol{p}_{i,n}^{j,l}$	n-th point from cloud frame captured from l-th LiDAR connected to j-th device at instant i.
$\boldsymbol{p}_o$	2D point representation of o-th object w.r.t LiDAR coordinate frame.
$\boldsymbol{s}_i^{j,l}$	Point cloud frame captured from l-th LiDAR connected to j-th device at instant i.
$\boldsymbol{t_k}$	Translation for k-th camera w.r.t. world frame.
$\boldsymbol{w}$	3D world point in the homogeneous co-ordinate system.

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Wei.

$\boldsymbol{x}_k$	3D world point in the k-th camera homogeneous co-ordinate system.
$\sigma_{zfilter}$	Z-axis variance theshold for cluster removal.
$assoc_o$	Boolean indicator of new centroid association with o-th object.
C	Number of image channels.
$c^{XY}_{overlay}$	2D plane projections of overlayed point.
c_g	Centroid of g-th cluster.
c_o	Class code of o-th object.
d	Object detection.
$d_{neighbor}$	Threshold distance for DBSCAN algorithm.
H	Height of image in pixels.
i	Instant at frame capture.
id_o	Unique identification of the o-th object.
j	Embedded device identification.
k	Camera identification.
l	LiDAR sensor identification.
l_{voxel}	Lateral voxel size of octree map.
M_k	Intrinsic calibration matrix for k-th camera.
N_c	Total number of cameras.
N_l	Total number of LiDAR sensors.
N_p	Total number of point in a point cloud frame.
$N_{cluster}$	Minimum number of dense points in a cluster DBSCAN algorithm.
n_{dense}	Number of neighbors of a dense point for DBSCAN algorithm.
O_i	Octree map from point cloud at instant i.
p^k_g	Transformed centroid from g-th cluster from the LiDAR coordinate frame to the coordinate frame of the k-th camera.
p^{pixel}_g	Pixel projection of centroid from g-th cluster.
P_k	k-th camera transformation matrix.
$r^k_{i,d}$	Detection Bounding Box of d-th detection from k-th camera at instant i.
R_k	Rotation matrix for k-th camera frame w.r.t. world frame.
t_o	Track of o-th object.
$u^k_{i,d}$	Detection object class code number of d-th detection from k-th camera at instant i.
W	Width of image in pixels.

I. INTRODUCTION

Localizing moving objects is a crucial task in computational perception, with applications ranging from surveillance to autonomous navigation. Accurate object localization serves as a foundation for tasks like target pursuit, behavior tracking, and obstacle avoidance. This task, among others, directly benefits from the fusion of different sensor modalities. The combination of 3D LiDARs and cameras are a very common combination since they complement each other [1], [2]. 3D LiDARs provide a depth map of environments, but fail to provide color and other semantic relevant information. In turn, RGB cameras provide offer rich information but do not provide any direct spatial information on the objects [3], [4].

Traditionally, both LiDARs and cameras are fused at early stage, by mapping of all 3D points into pixel space before further downstream tasks. This approach, however, relies on a very precise calibration and also in the alignment of frames from both sensors. Lately, there has been an increase in proposals for deep learning-based image and point cloud fusion that obtain labels and 3D location of objects [5]. Many of the fusion proposals require training and deployment of computationally expensive neural networks on high-performance hardware. These models typically demand a large amount of training data, fixed number of sensors, and inference can occur away from the input data source. That hinders their application to domains that require localization of objects to be performed in edge and computation constrained devices.

Recent advances in high-performance embedded computation hardware and deep learning algorithms allowed tasks like object classification, detection, localization, and tracking to be performed in embedded platforms (e.g. NVIDIA Jetson [6], [7], Raspberry Pi [8]). However, their computation power can not support the burden of deep learning methods fusing inputs from multi-modal sensors. Also, buffering inputs such as point clouds and multiple images is memory-intensive. The small form factor and rich hardware interfaces of edge devices, on the other hand, enable networking and distributed computation capabilities. They allow for the modular and flexible implementation of the aforementioned functionalities through the use of connected individual embedded devices.

Scaling the number of sensors and leveraging the strengths of each sensor modality can improve performance in tasks such as object localization, but pose a challenge to implementation on embedded systems. That entails breaking down complex functionalities into small chunks on different dedicated hardware components. When connected, embedded computing modules with limited capabilities can also be used; it allows easy and partial replacement of malfunctioning modules; and it also enables more dynamic and redundant sensor arrangements. Communication among sensors and computing modules in larger platforms such as self-driving cars, for example, can introduce latency and noise. Intermediate processing nodes on the edge may be able to reduce the length of raw data transmission. However, they can introduce network latency and makes it difficult to bridge the gap between modularity and time efficiency.

A system that perceives all its surroundings is needed for safer navigation, specially in cases such as collaborative robotics, where mobile platforms work in tandem with humans. While fusing sensors for surround view adds value for safety, it also adds computational cost. The necessity of having flexible and modular systems to process several sensors, summed up with the challenge of distributing the pipeline over constrained devices pose the motivation to this work. It addresses the limitations of existing approaches by proposing an edge-distributed fusion system for robust moving object localization. We aim to overcome the challenges

of processing camera and LiDAR data in real-time on distributed edge devices with constrained computational power. By leveraging the strengths of both sensor modalities, our approach aims to improve robustness and flexibility of object localization in dynamic scenarios.

To differentiate our work from the state-of-the-art, we introduce a modular and flexible implementation using two connected embedded devices. Further, four cameras and one LiDAR are employed in independent, unsynchronized computation pipelines. The system is able to detect, classify and track the position of moving objects in a 360° field of view with simplified 2D points. It is suitable to constrained computing settings in dynamic environments, where the platform should move in all four directions. Application examples could involve autonomous navigation for load transporting and inspection in industrial spaces with presence of humans, where perception of environment is required in all four directions.

The significance of our work lies in bridging the gap between LiDAR-camera fusion and modular, distributed edge computing. By optimizing the use of available computational resources, our approach enables real-time object localization, without relying on high-performance hardware or synchronization among sensors in a monolitic setting.

The summary of our contributions is listed as follows:

1) Hardware architecture: We propose an architecture combining more than one embedded device for fusing data from multiple sensors of different modality for real world applications. The architecture is designed in a way to create a distinct abstraction between sensors interface at one embedded device, and data fusion on the other one. This separation has added advantages to make system generalized, modular, and scalable. We demonstrate experimentally, backed by quantitative analysis, that this architecture has lower latency without compromising the localization task.
2) Distributed and asynchronous processing pipeline: The algorithms allow efficient network transmission and also fusion of unsynchronized data. The capacity of localizing objects becomes agnostic to the pretrained object detection model.
3) Finally, we demonstrate the concept experimentally by connecting four cameras and a LiDAR to more than one embedded device forming a network. We then implement the algorithms distributed over them for object detection, localization, and tracking multiple objects both in indoor and on publicly available data. The proposal is also validated with a hardware-in-the-loop simulation using an autonomous driving public dataset.

This paper is structured as follows: Sec. II summarizes the related work followed by problem formulation in Sec. III. Sec. IV presents the proposed method in detail; Sec. V describes the set-up adopted for the experiments followed by results in Sec. VI. Finally, Sec. VII presents conclusion and future directions.

II. RELATED WORK

Image processing algorithms have become well mature over the last years after the development of deep learning based techniques [9], [10], [11]. However, information occurs in 2D space and the important depth information required for localization tasks is not provided. Also, they do not work in dark or limited lighting conditions. Not so commonly, some attempts to estimate 3D object localization from 2D images can be found in the literature [12], but those are highly dependent on the training data distribution. To overcome the limitations, LiDAR sensors offer a dense point cloud of the environment and is considered a complement to vision sensors [13]. While cameras give a rich visual representation in 2D, point clouds give the location of objects in 3D as well as more information about shape and occupancy. In turn, deep learning methods for obtaining their location and labels solely from point clouds [14], [15], [16] are computationally expensive and require enormous training data, which might not be easy to obtain.

Sensor fusion can be categorized in three different types: early fusion, deep fusion and late fusion. In early fusion, data is combined before any individual processing or feature extraction step. In deep fusion, neural networks are used to directly process inputs, where complex relationships are learned and fusion can occur in many states with latent representations. In late fusion, each sensor's data is processed independently using sensor-specific algorithms [17], [18]. After processing and feature extraction, the outputs are combined or fused at a later stage. Also, it is possible to classify sensor fusion methods as model-driven or data-driven. In the first case a prior model is adopted, i.e.; the motion model of an object being tracked. Data-driven methods rely on training in a dataset to perform downstream task from sensors inputs, so all deep fused methods are necessarily data-driven. Some works, such as [19], can perform late fusion and combine both data-driven and model-driven approaches.

Some perception tasks also fuse data from radio frequency processing. Authors [18] proposed passive radio frequency and infrared signals (PRF-PIR) for human identification and activity recognition (HIAR). However, the classes used in their identification tasks is limited to humans, they present a limited field of view and interference from adjacent electronic devices.

In object localization, early fusion traditionally translates into overlaying the RGB image with a point cloud after coordinate transform, yielding 3D points cropped by a bounding box or semantic mask [20], [21], [22], [23]. That demands a high precision calibration among sensors and require large memory for processing data along the task.

Many of the state-of-art works in object localization use deep fusion, where neural networks sequentially produce and merge intermediate features from clouds and RGB images to perform object localization [24], [25], [26], [27], [28]. However, they present several setbacks: They require powerful computation and joint training in multi-modal datasets;

They rely on sensors synchronization and their complexity is intractable as the number of input sensors grow. Further, the annotations and their output use complete 3D bounding boxes to represent objects, adding unnecessary complexity for applications requiring only a point representation of their position.

Moving towards computing-constrained embedded applications, most of the literature focuses only on simple tasks such as object detection and recognition using only one device [7], [29], [30]. The work [31] proposed fusion of camera and radar for tracking vehicles in a road in a distributed architecture. However, they were limited to simulation only and required sensor frame synchronization.

Our work performs late fusion, having unsynchronized independent pipelines. Since it relies on a pre-trained object detection model and does not assume any motion model, out solution can be categorized as data-driven, with the benefit of requiring only image training data and not LiDAR data.

III. PROBLEM FORMULATION

The need of a large number of sensors from different modalities (both for complementary and redundancy) in embedded systems yield the need of embedded units to work together for accommodating the computation involved in perception tasks. Under this perspective, we revisit the simple perception task of tracking the position of moving objects to propose a modular and flexible system that accounts for communication among devices. Unlike other proposals [32], [33], this paper assumes that such task can be accomplished on the edge even with sensors connected to different and not synchronized embedded devices.

Referring to Fig. 1, let $\boldsymbol{s}_i^{j,l}$ denote a point cloud frame captured from l^{th} LiDAR ($l \in \{1, 2, 3 \dots N_l\}$) connected to embedded device $j \in \{1, 2, 3 \dots N_d\}$ during the experiment at a given instant i. $\boldsymbol{s}_i^{j,l}$ contains all 3D points obtained from the scene in a given frame. Mathematically, $\boldsymbol{s}_i^{j,l} = \{\boldsymbol{p}_{i,n}^{j,l}\}_{n=1}^{N_p}$, where $\boldsymbol{p}_{i,n}^{j,l} \in \mathbb{R}^3, \quad \forall i, j, l$ and N_p is the number of points contained in the frame. Similarly, $\boldsymbol{f}_i^{j,k}$ is an image frame captured by a camera $k \in \{1, 2, 3, \dots N_c\}$ connected at device $j \in \{1, 2, 3 \dots N_d\}$ at a given instant i. Each element $\boldsymbol{f}_i^{j,k} \in Z^{W \times H \times C}$ is a 3D matrix representing an RGB image with width, height and channel W, H and C respectively.

We consider real-world navigation that occurs in a 2D plane, where objects navigate in 2D plane (as they do not fly off the ground). For further efficient planning, the final perception task is simplified so as to estimate the minimum distances of objects projected in the ground plane. Thus, for every moving object of interest, we aim to obtain a set of labels $u_{o,i}$ and reference points $\boldsymbol{p}_{o,i} \in \mathbb{R}^2 \quad \forall i$, which represent the object position in the ground plane.

As illustrated in Fig. 2, sensors are connected to interface devices. The data acquired from them needs to be transmitted to the fusion device. It's desirable to obtain a compact form of important information at the interface device so it can be sent efficiently to the fusion device. Mathematically, that means

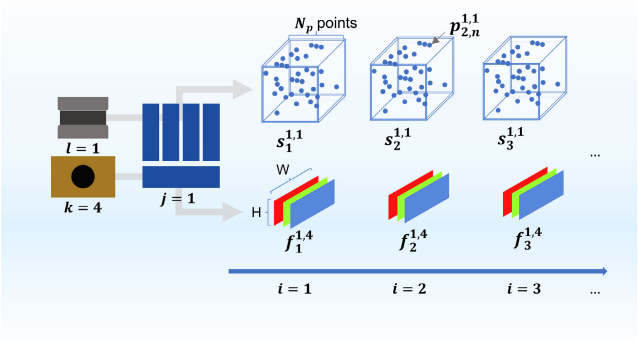


FIGURE 1. Illustration of the convention adopted. In the example, camera $k = 4$ and LiDAR $l = 1$ are connected to device $j = 1$, sending frames sequentially.

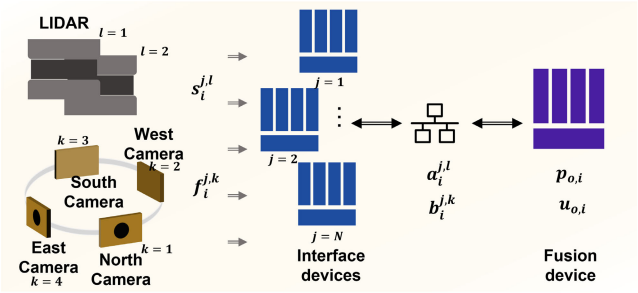


FIGURE 2. Schematic diagram of embedded devices for sensor interface and for fusion respectively as well as the data flow. $\boldsymbol{s}_i^{j,l}$ represents the point cloud frame from LiDAR l connected to device j at instant i, $\boldsymbol{f}_i^{j,k}$ the camera frame from camera k connected to device j at instant i, $\boldsymbol{a}_i^{j,l}$ the compact representation obtained from pre-processing point cloud frame from LiDAR l connected to device j at instant i, $\boldsymbol{b}_i^{j,k}$ the compact representations obtained from pre-processing image frame from camera k connected to device j at instant i. $u_{o,i}$ represents a given object label and $\boldsymbol{p}_{o,i}$ its position on the ground plane.

to acquire data from different modalities ($\boldsymbol{s}_i^{j,l}$ and $\boldsymbol{f}_i^{j,k}$) and to obtain: (a) representations from clouds $\boldsymbol{a}_i^{j,l} \in \mathbb{R}^{3 \times N_{p'}}$, with $N_{p'} < N_p \quad \forall j, l, i$ and (b) representations from images $\boldsymbol{b}_i^{j,k} \in \mathbb{R}^{W' \times B' \times C'}$, with $B' < B$, $W' < W$ and $C' < C \; \forall j, k, i$.

IV. PROPOSED METHOD

In this section we describe in detail the method proposed. First, we give an overview of the system (Subsec. IV-A); We then show the approximated camera calibration method adopted (Subsec. IV-B); The algorithms adopted in each computation node are described from Subsec. IV-C to IV-E.

A. OVERVIEW

We demonstrate the modularity and distributed computation across units by using two distinct embedded devices and five sensors – four cameras and the LiDAR. All sensors are connected to first embedded device (a.k.a interface device) and fusion occurs in the other one (a.k.a. fusion device). Even though sensors are connected to the same device, each sensor has its data processed independently, so, without any loss of generality, we are able to emulate the same conditions as if the computation was distributed among different interface devices with even more limited capacity.

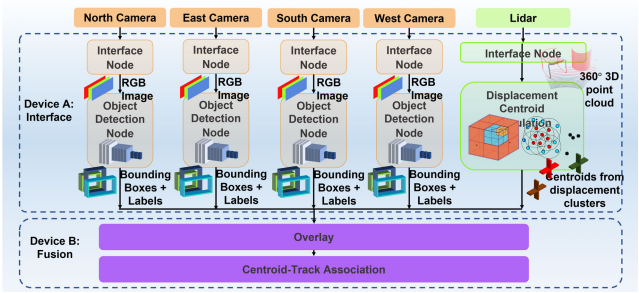


FIGURE 3. General architecture: Different pipelines with distributed nodes across two embedded devices. Data from cameras and LiDAR are preprocessed separately at a interface module and sent asynchronously to be fused at a fusion module.

The general distributed architecture with sensors integration and functionalities is illustrated in Fig. 3. All the five sensors are connected to a single device (Interface device). Data of different modalities (image and point cloud) is continuously collected. In the interface device, data coming from each sensor flows through individual pipelines, where software modules (nodes) extract the compact relevant representation of the scene from the each of four cameras (object labels and bounding boxes) and the LiDAR (displacement cluster centroids) simultaneously. Then those are exchanged through network messages in a publisher-subscriber scheme. The messages are sent to the second embedded device (fusion device) over network, where data fusion occurs and the compact representation of the scene is reconstructed. For the sake of simplicity, we omit the device j identification superscript from further variables.

Objects are identified using labels obtained in the camera pipelines, which also yield bounding boxes to be used along with the output from LiDAR pipeline for localization. To avoid dependency on multi-modal datasets and complex models at the LiDAR pipeline, we adopted a non-supervised approach based on comparison of two sequential point cloud frames, denoted as displacement centroid calculation (Subsec. IV-D). Obtaining a compact representation from images is quite straightforward due to availability of pre-trained detection models. Once the bounding boxes and labels are obtained (Subsec. IV-C), the points obtained from the cloud pipeline are overlayed by the bounding box (Subsec. IV-E). That is only possible due to the preceding calibration process described in Subsec. IV-B. The result of overlay is sent for updating tracks or creating new ones. The tracks are also updated in the absence of overlays, relying only on the output of clouds. A more detailed diagram with algorithms and data flow is depicted in Fig. 4.

B. CALIBRATION

Before fusing information from the sensors, it is crucial to calibrate them - four cameras with respect to one other and then each camera with respect to the LiDAR – to achieve a representation in the same coordinate frame. Camera k maps 3D world points to 2D image given by (1), where $\boldsymbol{x}_k \in \mathbb{R}^3$ is a 2D image in homogeneous co-ordinate system, $\boldsymbol{w} \in \mathbb{R}^4$ is a 3D world point in the homogeneous co-ordinate system, and $P_k \in \mathbb{R}^{3\times4}$ is the camera matrix given by (2), where M_k is the intrinsic parameter. R_k is the rotation and $\boldsymbol{t_k}$ is the translation of camera frame with respect to world frame respectively, together forming the extrinsic matrix.

$$\boldsymbol{x}_k = P_k \boldsymbol{w} \tag{1}$$

$$P_k = M_k[R_k|\boldsymbol{t_k}] \tag{2}$$

While extrinsic calibration ($[R_k|\boldsymbol{t_k}]$) maps the 3D world co-ordinate to camera co-ordinate, intrinsic calibration (M_k) maps the camera co-ordinate to image plane. For extrinsic calibration, the LiDAR sensor coordinate frame is adopted as reference. Thus, both the pose of LiDAR coordinate frame and the cameras is found with respect to a chosen calibration target, so the final transform matrix between each camera and the LiDAR is obtained. A board with four circular blob with known center, diameter and the position of each blob in world co-ordinate was used as the calibration target. Circular holes are detected by Circle Hough Transform algorithm; the target position with respect to the camera is then obtained using SolvePnP algorithm [34]. The transformation matrix between the LiDAR and the center of the target board was estimated by the procedure in [22], [23], and [35]. This is a well-known calibration technique because circular blobs present features that (a) are very distinguishable and can be easily identified in images by computer vision algorithms and, at the same time, (b) provide the disparity in depth that can be capture by the point clouds from LiDARs.

For intrinsic calibration, checkerboard was used as reference and the coefficients are obtained by identifying its target corner points, implementing the method based on [36]. For simplification purposes, the calibration is performed for north camera only. We also take advantage of the geometrical disposition of all cameras (as each camera is perpendicular to one other) and use the extrinsic matrix obtained for north camera. We rotate it in 90° in the plane sequentially for estimating the extrinsic matrix for other cameras.

C. OBJECT DETECTION

Referring to Fig. 3, four computing nodes in the interface module receive streams images from the 4 cameras and each of them perform object detection. This work proposes a system that is agnostic to the type of object detection, so any pre-trained detection model can be adopted, making the localization available to whichever classes the detection model can detect. We assume that the model does not create redundant bounding boxes for the same object in the image, so there will be no intersection among them.

In this work, we adopted the SSD (Multibox Single Shot Detector) architecture [37]. This neural architecture is well-known in the literature for inference speed performance. It was optimized for running on the embedded device [38] and trained in the COCO (Common objects in Context) public dataset [39]. Each camera k object detection node contains a SSD detection model. If one or more detections are present

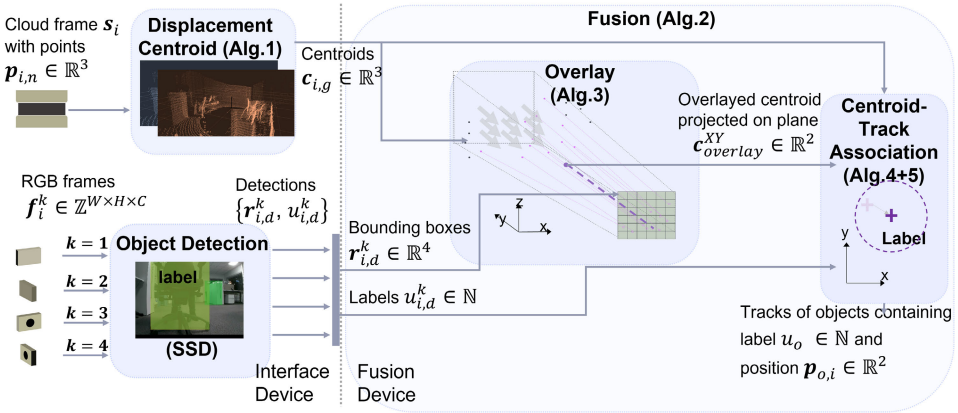


FIGURE 4. Data flow and algorithms. Cluster centroids are obtained from displacement in point clouds (Alg. 1). For each camera, bounding boxes and labels are obtained for every detection (Subsec. IV-C). After an approximate synchronization is performed (Alg. 2), the closest centroid overlayed by a bounding box is projected in the ground plane (Alg. 3) and can create or update tracks (Alg. 4). Non-overlayed centroids can also be used to update nearby existing tracks (Alg. 5).

for a given instant i, a set containing information tuples $\{r^k_{i,d}, u^k_{i,d}\}$ of each detection d from that camera is sent. For each tuple, $r^k_{i,d} \in \mathbb{R}^4$ contains the center cordinates, height and width of the bounding box and an object class code number $u^k_{i,d} \in \mathbb{N}$ (Shown in Fig. 4). In the absence of detections for a given image frame, no message is published.

SSD is fast but not so accurate, outputting intermittent detections. It was purposely chosen as object detector to prove the ability of the system to locate objects even when it fails to detect an existing object for a given frame (demonstrated in Sec. V). Although detecting object classes is essential in the proposed task, the tracking process in this work can be performed also in the absence of a detection message, hence increasing the reliability of the overall system.

D. DISPLACEMENT CENTROID CALCULATION

The overall scheme of displacement cluster formation is illustrated by Fig. 5 and Alg. 1. Similar to sending the entire image of the scene captured by four cameras, sending the complete point cloud captured by LiDAR from one platform to another is unfeasible in terms of network latency, computational cost and memory requirements. Hence, a more compact representation must be obtained from processing point clouds. The key idea behind each cloud frame processing is 3D background filtering, by subtracting points referring to static portion of the scenario. A given point cloud frame $\boldsymbol{s}_{i-1}$ is stored while processing of next point cloud frame $\boldsymbol{s}_i$. When $\boldsymbol{s}_i$ arrives at the processing node, an octree map for previous frame $\boldsymbol{s}_{i-1}$ is generated. Octrees are a quite compact volume occupancy representation based on voxels [40]. Considering a voxel as a cube, its lateral size is established by a parameter l_{voxel}. Once the octree from $\boldsymbol{s}_{i-1}$ is generated, the original points are removed from the structure and the current points from $\boldsymbol{s}_i$ are inserted in the octree structure. By design, this data structure allows a quick verification if a given voxel is occupied or not by any point. If the voxel was previously empty for $\boldsymbol{s}_{i-1}$ and became occupied by one

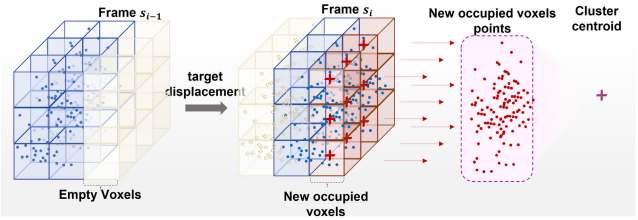


FIGURE 5. Displacement cluster scheme: Points from previous frame stored as octree(left); The points from the current frame mapped to the stored octree and displacement points selected (middle); Clustering (right).

of the current points of $\boldsymbol{s}_i$, those points are considered as displacement and selected. If there is no previous frame, the complete point cloud is considered as the displacement and that enables the system to track stationary objects at initialization.

The next step is to cluster the displacement points. The density-based clustering method DBSCAN was adopted [22]. It associates points to a cluster given the number of neighbor points within a threshold distance $d_{neighbor}$. A region around a point is considered dense and can potentially start a cluster if there are at least n_{dense} neighbors. The minimum number of grouped points $N_{cluster}$ required to form a cluster can be adjusted, so smaller associations will be considered as outliers. Due to the laser projection pattern, the ground and other planes that do not have their normal vectors pointed towards the LiDAR sensor will have a lower density of points. As they are not relevant for the task, the algorithm conveniently treat them as outliers. Other emerging points due to sensor imprecision are also discarded.

Even with noise removal, objects that were previously shadowing floor regions can allow ground emerging points as they move and those can still mistakenly be treated as a displacement cluster. To remove that artifact, a filter is applied by evaluating the variance of the cluster in the Z direction (orthogonal to the ground plane). If the value is smaller than a threshold $\sigma_{zfilter}$, the cluster is discarded. After obtaining

Algorithm 1 Displacement Centroid Calculation

Input: Previous point cloud frame $\boldsymbol{s}_{i-1}$, Current Point cloud Frame $\boldsymbol{s}_i$
Output: Displacement Centroids $\mathcal{C}_i$

$O_i \leftarrow$ Octree map from $\boldsymbol{s}_{i-1}$
$O_i \leftarrow$ Remove all points from $\boldsymbol{s}_{i-1}$ (Keep tree structure)
$\mathcal{D} \leftarrow \emptyset$ ▷ Set of displacement points
$\mathcal{C}_i \leftarrow \emptyset$ ▷ Set of displacement centroids
for each $\boldsymbol{p}_{i,n} \in \boldsymbol{s}_i$ **do**
  $v_n \leftarrow$ Find the voxel corresponding to $\boldsymbol{p}_{i,n}$ in O_i ▷ Check for each point if the voxel in the octree exists
  if $v_n = \emptyset$ **then**
    $\mathcal{D} \leftarrow \mathcal{D} \cup \boldsymbol{p}_{i,n}$ ▷ If not found, add it as a displacement point
  end if
end for
$\mathcal{G} \leftarrow$ Apply DBSCAN clustering on points in $\mathcal{D}$ and retrieve the set of clusters
for $g \in \{1, 2, \ldots, |\mathcal{G}|\}$ **do**
  $c_g \leftarrow$ Calculate centroid of cluster g
  $\sigma_{gz} \leftarrow$ Variance of cluster over the Z axis
  if $\sigma_{gz} > \sigma_{zfilter}$ **then**
    $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup c_g$ ▷ Only add centroids from clusters with variance in Z axis higher than threshold (X: front, Y: left, Z: up)
  end if
end for
Store $\boldsymbol{s}_i$
return $\mathcal{C}_i$

Algorithm 2 Fusion

Input: Centroids from current cloud frame $\mathcal{C}_i$, Tracks from previous frame $\mathcal{T}_i$
Output: Updated tracks $\mathcal{T}_{i+1}$

$\mathcal{H} \leftarrow \emptyset$ ▷ Store new created tracks to be added at the end
$\mathcal{T} \leftarrow \emptyset$ ▷ Stores temporarily the existing tracks as they get updated by associations
$\mathcal{D} \leftarrow$ Retrieve detections within time span ϵ_{sync} from all cameras ▷ Each element of $\mathcal{D}$ consist of a tuple $(r^k_{d,i}, u^k_{d,i})$ referring to a detection d coming from camera k
$\mathcal{C}^{XY} \leftarrow$ 2D plane projections of all centroids in $\mathcal{C}_i$
for $d \in \{1, 2 \ldots |\mathcal{D}|\}$ **do**
  $c^{XY}_{overlay} \leftarrow$ Overlay$(\mathcal{C}_i, r^k_{d,i})$ ▷ Alg.3
  if $c^{XY}_{overlay} \neq NULL$ **then**
    $\mathcal{H}, \mathcal{T} \leftarrow$ OverlayedAssociation$(c^{XY}_{overlay}, r^k_{d,i}, u^k_{d,i}, \mathcal{T}_i)$ ▷ Alg.4
    $\mathcal{C}^{XY} \leftarrow \mathcal{C}^{XY} - \{c^{XY}_{overlay}\}$
  end if
end for
$\mathcal{T} \leftarrow$ OnlyCloudAssociation$(\mathcal{T}, \mathcal{C}^{XY})$ ▷ Alg.5
Set $assoc_o = False \quad \forall o \in \{1, 2 \ldots |\mathcal{T}|\}$
$\mathcal{T}_{i+1} \leftarrow \mathcal{T} \cup H$
return $\mathcal{T}_{i+1}$

the clusters, their centroids c_g are calculated and sent to the fusion platform, being grouped as a set $\mathcal{C}_i$.

E. FUSION

The fusion process is described in described in Alg. 2. The messages containing detections from cameras (bounding boxes and labels) are transmitted asynchronously according to the deep learning model output. The messages containing the displacement centroids (obtained from Alg. 1) are used as a reference for trying to match detection messages to a given point cloud frame. Further, a buffer stores the maximum number of messages from each topic (detection from 4 cameras plus displacement centroids). All messages contain the timestamp of the data acquired by the corresponding sensor of the pipeline. A detection message is matched to a centroid message only if the timestamp difference between both is smaller than a threshold ϵ_{sync}. In case there is more than one timestamp match between the centroid message and detection messages from a given camera, the shortest time difference message is considered.

Once the displacement centroids and object detection are matched, they proceed to the overlay (Alg. 3), where each centroid c_g is transformed from the LiDAR coordinate frame to the coordinate frame of the camera p^k_g corresponding to the detection using the matrices obtained during calibration (Subsec. IV-B). Then, the centroid is projected on the camera plane and sub sequentially, their corresponding pixel coordinates are obtained. The centroids that lie inside of the bounding box from detection are said to be overlayed and then selected. Due to the assumption of no intersection among boxes, points are overlayed by only one bounding box each. Then, 2D ground projection of centroid $p_{overlay}$ that has minimum distance to the sensor is chosen.

F. CENTROID-TRACK ASSOCIATION

Object tracking is the task of taking an initial set of object detections and re-identifying them as they move around the scenario [41]. Tracking involves associating detections at a given time with the instance previously tracked. This association is often approached as a global minimization problem [42]. In our context, the focus is continuously localizing the objects in the field of view by providing 2D reference points and labels even in the absence of object detection from camera. Hence,we employ a simplified association scheme. In this context, tracking is also used to re assign labels to reference points $\boldsymbol{p_o}$ when they are not overlayed by any detection bounding box.

Both data overlay and tracking are explained in Alg. 4. Each track t_o is consisted by a tuple $\{id_o, c_o, \boldsymbol{p}_o, assoc_o\}$. The variable id_o is the unique identification of that object, c_o refers to its C identified by the object detection module, $\boldsymbol{p}_o$ is the two dimensional point that represents the object with respect to the LiDAR coordinate frame projected in the ground plane. The variable $assoc_o$ is a support boolean flag that indicates if the track was associated with a new incoming centroid at a given association step.

Algorithm 3 Overlay

Input: Bounding box r_d^k for a detection d obtained from camera k, Centroids $\mathcal{C}$
Output: 2D projection of closest overlayed centroid $c_{overlay}^{XY}$

$c_{overlay}^{XY} \leftarrow \emptyset$
$\mathcal{L} \leftarrow \emptyset$ ▷ Overlayed centroids projected on ground plane
${}^kT_l \leftarrow$ Retrieve extrinsic calibration matrix between LiDAR and camera k
$M \leftarrow$ Retrieve intrinsic calibration matrix for camera
for $g \in \{1, 2 \ldots |\mathcal{C}|\}$ **do**
  $p_g^k \leftarrow {}^kT_l \cdot c_g$ ▷ $p_g^k \in \mathbb{R}^3$ is the centroid in the camera k coordinate frame
  $p_g^{pixel} \leftarrow M \cdot p_g^k$ ▷ $p_g^{pixel} \in \mathbb{N}^2$ is the centroid projected in the pixel space of camera k
  if $p_g^k[2] > 0$ and p_g^{pixel} inside r_d^k ▷ Check if centroid is ahead of the camera (Z axis pointing out from lens) and if corresponding pixel lies inside bounding box **then**
    $c_g^{XY} \leftarrow$ 2D ground projection of centroid c_g
    $\mathcal{L} \leftarrow \mathcal{L} \cup c_g^{XY}$
  end if
end for
$c_{overlay}^{XY} \leftarrow c_g^{XY} \in \mathcal{L}$ with minimum distance to sensor $||c_g^{XY}||$
return $c_{overlay}^{XY}$

In case there has been a fusion of displacement centroids (previously obtained from Alg. 1) and detections, the resulting overlaid point is projected on the ground plane and considered. If it is located within a threshold distance $\rho_{overlay}$ from a current track with the same label of the detection, the fused point is matched to the track and the centroid is associated with the track point. In case any overlaid closest point is not associated with any track, a new track is created. Even though we do not assume any prior motion model for the objects, we use the assumption that the objects will not move further than this threshold value between frames.

Although tracks are only created through fusion, they can be updated by pure centroids in the absence of object detection matching a given point cloud, which improves the reliability of the system. As described in Alg. 5, the update occurs if the distance between the projected centroid and the track is lower than a threshold $\rho_{purecloud}$. The absence of detection increases the risk of mismatches, so we assign a lower value to $\rho_{purecloud}$. We assume that objects will not be closer than this value to each other.

V. EXPERIMENTS

The setup implementation[1] uses four RGB cameras, scanning Ouster LiDAR (image and point cloud), and two embedded devices (Nvida Jetson Xavier AGX) via local area network (LAN) connection. This is demonstrated in Fig. 6. The NVIDIA Jetson AGX Xavier modules contain 8 CPU cores, GPU with 512 NVIDIA CUDA cores and 64 Tensor cores and RAM memory of 32 GB. They have additionally a proprietary hardware accelerator (NVDLA) for deep learning convolution operations. The four cameras having 95° field of view each and the resolution of image 1280 × 720 are arranged to face in north, east, south and west direction. They are connected to one embedded device through a dedicated MIPI-CSI 2 interface. The images are streamed at 30 fps. The Ouster LiDAR (OS-1-64) [43] is configured with a horizontal resolution of 512 points and vertical resolution of 64 rings. Its precision varies according to the range: 0.3 - 1 m: ± 0.7 cm, 1 - 20 m: ± 1 cm, 20 - 50 m: ± 2 cm. It is connected to interface Jetson through Ethernet and stream data points from the complete surround view at 20 Hz. The LiDAR is placed on top of the cameras so its center intersects with the focus lines of the four cameras. The sensor system is mounted on a mobile platform [44] so the cameras are disposed equidistantly to cover the complete surround view.

[1]The code used in the platforms is available on https://github.com/jamendola/sur_fus_loc.git

Algorithm 4 OverlayedAssociation

Input: Centroids from current cloud frame $c_{overlay}^{XY}$, Bounding Box r^d, Detection label u_d, Tracks from previous frame $\mathcal{T}$
Output: Updated tracks $\mathcal{T}$

$\mathcal{H} \leftarrow \emptyset$ ▷ New tracks to be created
$t_{closest} \leftarrow$ Track with closest $\boldsymbol{p}_o$ to $c_{overlay}^{XY}$
if $u_{closest} = u_d$ and $||c_{overlay}^{XY} - p_{closest}|| < \rho_{overlay}$ **then**
  $p_{closest} \leftarrow c_{overlay}^{XY}$ ▷ Updates closest track position with $c_{overlay}^{XY}$ if the labels from track and detection are equal and distance is shorter than threshold
  Set track $\boldsymbol{t}_{closest}$ as associated
else
  $t_{new} = \{id_{new}, u_d, c_{overlay}^{XY}, assoc_{new} = True\}$
  $\mathcal{H} \leftarrow \mathcal{H} \cup t_{new}$
end if
return $\mathcal{H}, \mathcal{T}$

Algorithm 5 OnlyCloudAssociation

Input: Tracks $\mathcal{T}$, Left centroids projected on plane $\mathcal{C}^{XY}$
Output: Updated tracks $\mathcal{T}$

for $o \in \{1, 2 \ldots |\mathcal{T}|\}$ **do**
  if $assoc_o = False$ ▷ Not associated **then**
    $c_{closest}^{XY} \leftarrow$ plane projected centroid from $\mathcal{C}^{XY}$ with minimum distance to $\boldsymbol{p}_o$
    if $||\boldsymbol{p}_o - c_{closest}^{XY}|| < \rho_{purecloud}$ **then**
      $\boldsymbol{p}_o \leftarrow c_{closest}^{XY}$
    end if
  end if
end for

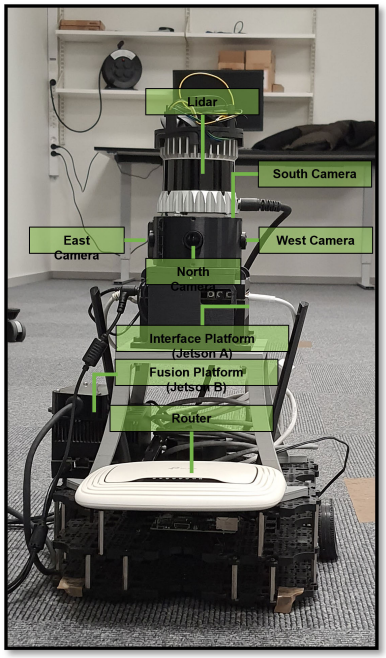


FIGURE 6. Proposed experimental setup.The mounting remains static and is placed on a suspended base for the experiments.

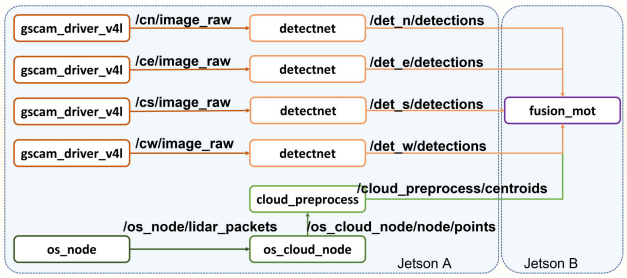


FIGURE 7. Software architecture. Arrows represent messages exchanged in a publisher-subscriber scheme.

As shown in Fig. 7, the software architecture is based on computing nodes using the open source Robot Operating System (ROS) [45]. The implementation was performed using Python, C++, frameworks NVIDIA TensorRT and Point Cloud Library (PCL). Each computing node receives and sends messages through Transmission control protocol (TCP). Messages are sorted according to a topic, so publisher nodes can communicate with subscriber nodes. The cameras are integrated with ROS through Gstreamer library (**gscam_driver_v4l** nodes). Each node instance publishes images as messages of a ROS topic. Each topic is uniquely identified according to the camera producing the images and there is a different instance of the ROS node of type **detectnet**. For each of the camera topics, the respective model receives a raw image, rescales it to 300 × 300 resolution and performs object detection. Point clouds are also pre-processed on the interface jetson on edge. The LiDAR driver is wrapped by **os_node ROS** node and packets containing measurements for small azimuth blocks are sent to

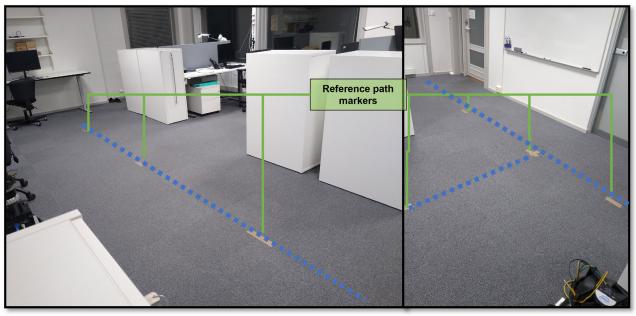


FIGURE 8. Left side and Right side of the area used for the experiments and reference path markers. The path in "T" shape to be followed by the target objects is partially shown in the picture in blue dashed lines. The sensor can be seen in bottom right with a distance of 1.5 m from each line of the path and remains static as the targe object moves through the path.

os_cloud_node node. It accumulates and organizes points, sending messages that represent a point cloud for the complete 360° view. Once a given point cloud frame is sent to node **cloud_preprocess** and processed, a much more reduced number of points is published and sent to **fusion_mot** node at fusion jetson. A more detailed specification of the ROS topics is listed below:

- **/c*/image_raw**: Header(identification and timestamp) + RGB Image Matrix with shape 1280 × 720 of 1 byte (RGB8 encoding)
- **/os_node/lidar_packets**: Header(identification and timestamp and data layout) + Buffer of 6464 bytes containing data from 16 azimuth measurement blocks.
- **/os_cloud_node/points**: Header(identification and timestamp and data layout) + 51264 elements array containing coordinates X,Y,Z, intensity, noise, range and ring
- **/cloud_preprocess/centroids**: Header(identification and timestamp) + variable buffer containing sequence xyz point coordinates (32 bit floats)
- **/det_*/detections**: Header(identification and timestamp) + array of composed data containing bounding box width,height,center x and center y (64 bit float) and class id (integer)

The complex functionalities needed for perception are divided and implemented in small chunks on the dedicated hardware platform and then integrated together (Fig. 3). This architecture adds the advantage that the embedded system with limited computation capabilities can also be used as the total work is divided among the distributed systems. Further, having a distributed architecture for sensor fusion and perception in navigation platforms allows easy and partial replacement of malfunctioning modules. It also allows more dynamic and redundant sensor arrangements. In larger platform such as self-driving cars, for example, communication among sensors and computing modules can present latency and noise. Thus, intermediate processing nodes on the edge could reduce the expanse of raw data-transmission.

FIGURE 9. Different sides of the larger area used for detecting multiple objects. The sensor setup stays static in the middle of the open area as the target objects should freely move around it.

A. INDOORS EXPERIMENTS

To evaluate the proposed methodology (Alg. 1-5) with real self-conducted experiments, we adopted the scenarios shown in Fig. 8 and 9. Even though objects can move freely around the surround view system, markers were placed on the floor of the first scenario (Fig. 8) to establish a reference path for further error calculations. The second scenario (Fig. 9) has a larger space to accommodate more than one object of interest at a larger distance from the sensors.

B. OUTDOORS PUBLIC DATASET EXPERIMENTS

To evaluate the algorithm in an outdoor environment with sensors embedded in a moving platform and also perform comparative analysis, we further performed experiments through a simulated environment using the KITTI-360 public dataset [46], where data was streamed through ROS in Hardware-in-the-loop style. For the scenario considered, the point clouds captured by a Velodyne HDL-64E LiDAR, images from cameras point forwards (1408×376 resolution) and pointing sideways (1400×100 resolution) were used. All imaged were rectified and re-scaled to 300×300 resolution when passed to detection models. To ensure that only moving objects and new objects entering the LiDAR field of view are captured, we relied on the localization information provided by the dataset and adopted the global fixed coordinate frame as reference when transforming the pixels and point clouds.

VI. RESULTS

A. INDOOR EXPERIMENTS

1) SINGLE OBJECT TRACKING

For the indoor experiments,[2] the parameters adopted are listed in Table 1.

The first experiment was conducted by having a person to walk over the reference path established by markers in the room as shown in Fig. 8. For brevity, the path was established to use three of the four cameras from the surround system for fusion. During the experiment, the person could freely rotate the body or turn his head as long as its center could keep on the path. The system was set so only objects classified as a person were tracked. The sequence of track points were then compared with the reference lines. The experiment was repeated three times, acquiring 100 frames each time, with the person walking along the lines with an average speed of 2 km/h. Fig. 10 illustrates the display of the trajectory of a track point along with the original point cloud of the scenario.

Fig. 11 shows the trajectory of the track point evolving as the person of interest is captured by different cameras. It shows the detection bounding box covering the person totally (Fig. 11A, blue shaded area), partially (Fig. 11B, blue shaded area) or even absent across the trajectory (Fig. 11C). Still, the track point (correspondingly in Fig. 11D, E, F) could be successfully updated for all the situations. That demonstrates the ability of the system to compensate for failures in pipelines from camera and detection by using only information from LiDAR.

[2]The hyper parameters were tuned for the scenarios and their impact is discussed in Sec. VII.

TABLE 1. Experiment parameters.

Parameter	Description	Value
$\rho_{purecloud}$	Association threshold for cloud only	0.3 m
$\rho_{overlay}$	Association threshold for overlay with detection	0.5 m
-	Association threshold for overlay with detection	2
ϵ_{sync}	Synchronization threshold	0.1 s
l_{voxel}	Voxel size	0.1 m
$d_{neighbor}$	Maximum neighbor distance for DBSCAN	0.05 m
n_{dense}	Minimum number of neighbors for cluster assignment	5
$N_{cluster}$	Minimum number of points in a cluster	10
$\sigma_{zfilter}$	Z axis variance threshold for cluster filtering	0.01

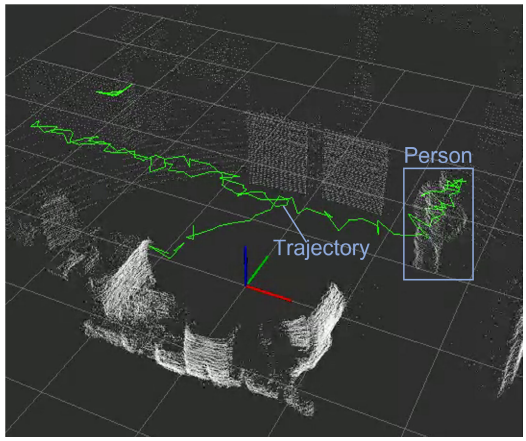


FIGURE 10. Trajectory of the reference point (geen lines) along with point cloud obtained through Alg. 1-5. The coordinate system is defined by the LiDAR on the setup, as illustrated by the red, green and blue arrows.

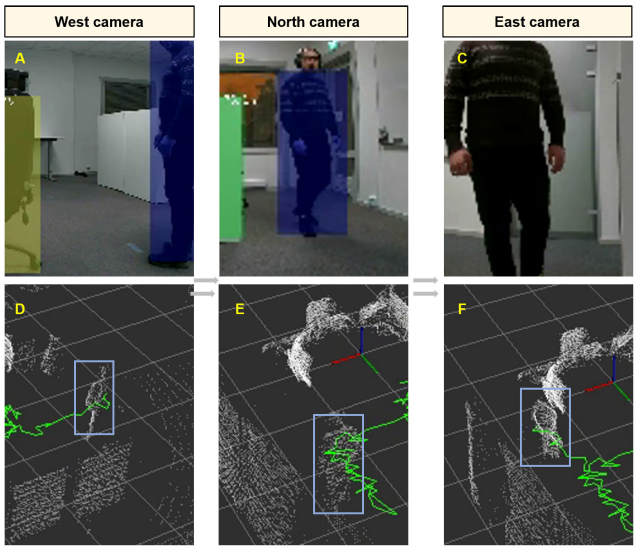


FIGURE 11. View of cameras and LiDAR point cloud for person. Camera images (A,B,C) with the respective point cloud correspondence (D,E,F) for person as object of interest. Detection occurring in both west (A) and north (B) cameras overlaying centroids for updating the track. East camera (C) without detection and only centroid is used for update.

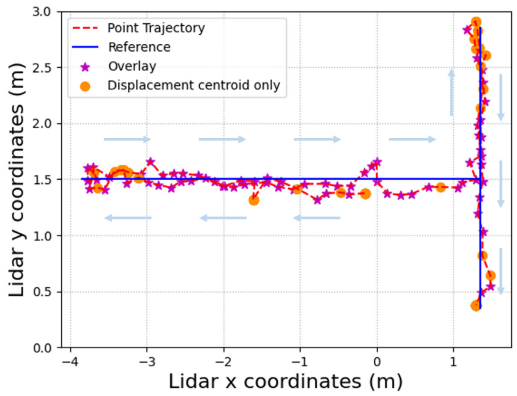


FIGURE 12. Trajectory corresponding to a person walking over the reference path. The arrows indicate the sequence of the trajectory and different symbols indicate point updates from overlayed (star) and non-overlayed centroids (circles) using for updating the track.

The trajectory points obtained are compared to the reference path and is shown in Fig. 12. The points obtained by updating the tracks are depicted by different scatter symbols to discriminate between: a) updates by centroids overlayed with detections (stars) and b) pure displacement cloud centroids in the absence of detection (circles). Points represented by circles demonstrate once more (as claimed earlier) that tracking would not be possible relying on detections coming from the SSD (or camera) at every frame. It is possible to see that the trajectory occurs in 'zig-zags'. That reveals the pattern of walking, where each step has the points of alternating legs as displacement points to be associated with the track. Head turns and body rotations also occurred on the horizontal portion of the path and cause displacement points to be captured. The mean error in distance between each track point and the path over navigation of 15.1 m is estimated to be 5.4 cm (mean error percentage = 0.3 %) and precision of ±2.2 cm.

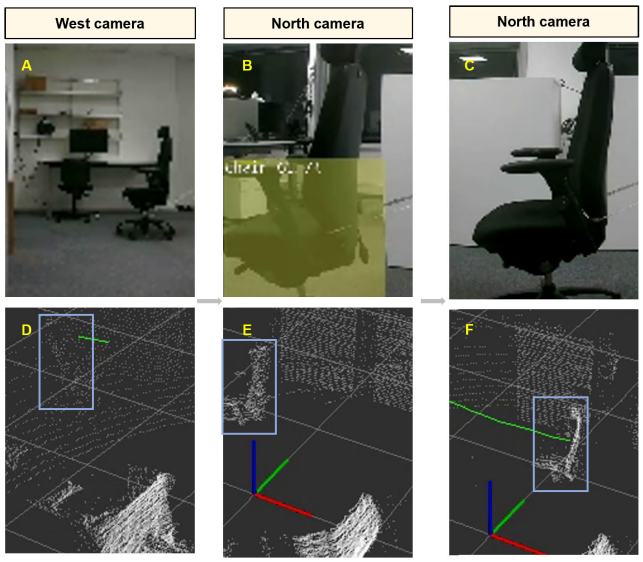


FIGURE 13. Camera images (A,B,C) with the respective point cloud correspondence (D,E,F) for a chair as object of interest. A: Even though the object can be seen from east camera, no detection occurs and the non-overlayed centroid updates the track. B: Detection occurs in north camera and centroid is overlayed. C: Detection fails again for north camera and centroid is not overlayed.

A second random object whose label is within the COCO dataset (chair) was chosen for an experiment. A rotating chair was used to make a contrast as compared to human in first experiment. As illustrated in Fig. 13, SSD fails to detect the chair correctly. Still, the system is able to keep tracking it. The trajectory and reference path are plotted in Fig. 14. Differently from the person, the trajectory does not present an evident 'zig-zag' pattern (as all the points on the chair are fixed, unlike the human where different parts move in different direction). Similar to the previous case, the experiment was repeated three times, also acquiring 100 frames each time, with the chair being moved at an average speed of 2.5 km/h. The mean error over navigation of 15 m is estimated to be 4.9 cm (mean error percentage = 0.3 %) and precision of ±3 cm respectively. The algorithm shows similar results independent upon the type of object.

2) SIMULTANEOUS MULTIPLE OBJECT TRACKING

After finding the accuracy (given by mean error) of the proposed methodology, finally, a qualitative experiment was further performed to demonstrate the functionality of surround view multiple object tracking. In the experiment (in the environment shown in Fig. 9) four individuals walk randomly around the field of view of each camera. Fig. 15 shows that the system can track all targets even at further distances. This last experiment demonstrated that the system not only tracks an object of interest across the different camera regions, but also can track more than one object simultaneously. It is important to notice that the objects started being

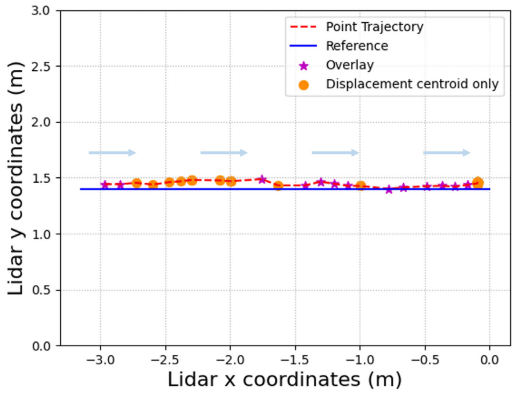


FIGURE 14. Trajectory corresponding to a chair being towed over the reference path. The arrows indicate the sequence of the trajectory and different symbols indicate point updates from overlayed and non-overlayed centroids used for updating the track.

TABLE 2. Mean values obtained for indoor experiments.

Point Cloud acquisition (s)	0.051 ± 0.001
Displacement centroids calculation (s)	0.132 ± 0.004
Centroids transmission (s)	0.003 ± 0.002
Image Streaming (s)	0.030 ± 0.001
Detection processing and transmission (s)	0.020 ± 0.002
Fusion and track update (s)	0.002 ± 0.001
Error - Chair (m)	0.049 ± 0.03
Largest Error Value - Chair (m)	0.09
Error - Person (m)	0.054 ± 0.02
Largest Error Value - Person (m)	0.09

tracked in the four different sides because of the presence of four cameras covering all the surroundings.

The larger area available also allowed the individuals to be at a further distance, revealing that the system is able to track objects as long as the displacement point clouds representing them do not become too sparse and are filtered off.

The number of centroids sent from the interface platform to the fusion platform did not exceed 15 points once the first frame was sent, which enables very low transmission latency (compared to sending the entire scene from four camera and/or LiDAR). Table 2 summarizes the average latency and errors for each relevant stage in the system pipeline. For calculating the latency values, 600 frames from both cases were used. If we consider the LiDAR and cloud processing pipeline as the bottleneck for fusion, the total time taken from data acquisition until fusion and track update takes approximately 190 ms and the computation time takes 0.14 ms. The transmission of data between the two platforms showed relatively lower latency (0.01 s), with values that are significantly lower when compared to computation time.

Further, preliminary trials revealed that running all nodes in one single Jetson computing platform, without the point cloud preprocessing and performing overlay of all points in

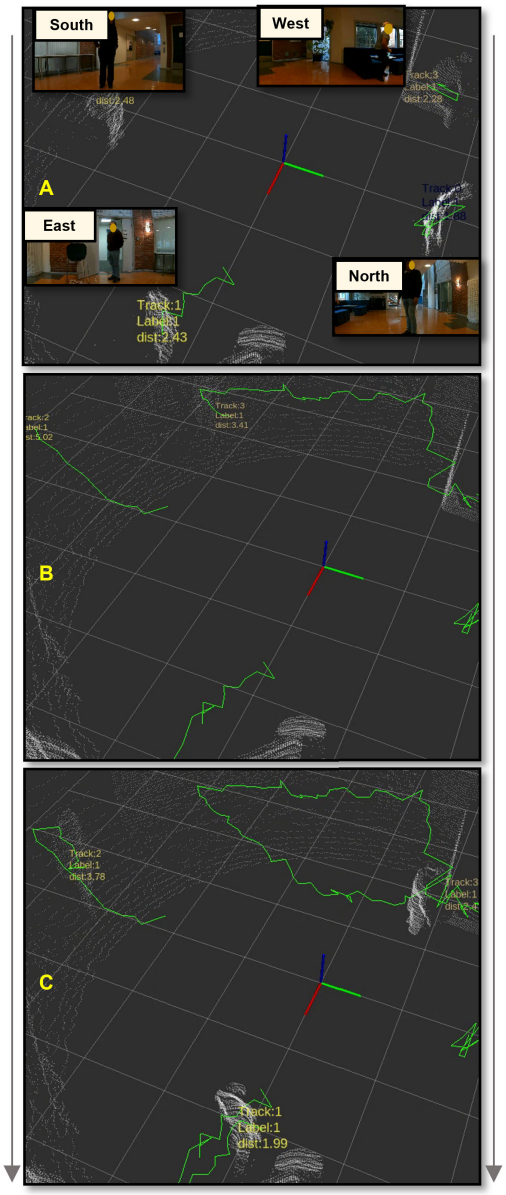


FIGURE 15. Four people walking around the system each one around a different direction. North camera is pointed in the direction of green axis. East camera is pointed in the direction of red axis. A: Camera images showing each person in each field of view. B,C: The respective trajectories of each individuals evolving over time.

the caused system crashes due to memory and processing limitations. That reassures the need of a networked architecture with processing at the edge for embedded fusion of multiple sensors.

B. OUTDOORS PUBLIC DATASET EXPERIMENTS

The proposed method was implemented on publicly available dataset KITTI 360. Fig. 16 illustrates the distance estimation of a vehicle passing by the platform in a road. Given that the scenario adopted has cars passing by the platform on its left side, we considered two situations: using images only from left-frontal camera of the car set-up (Base model), and also

TABLE 3. Use cases for ablation studies.

Case	Base	A	B	C	D	E (Proposed)
ρ	5.0	5.0	1.0	10.0	5.0	5.0
l_{voxel}	1.0	0.5	1.0	1.0	1.0	1.0
Detection model	Mobile-net	Mobile-net	Mobile-net	Mobile-net	Incep-tion	Mobile-net
Cameras	Frontal-left	Frontal-left	Frontal-left	Frontal-left	Frontal-left	All

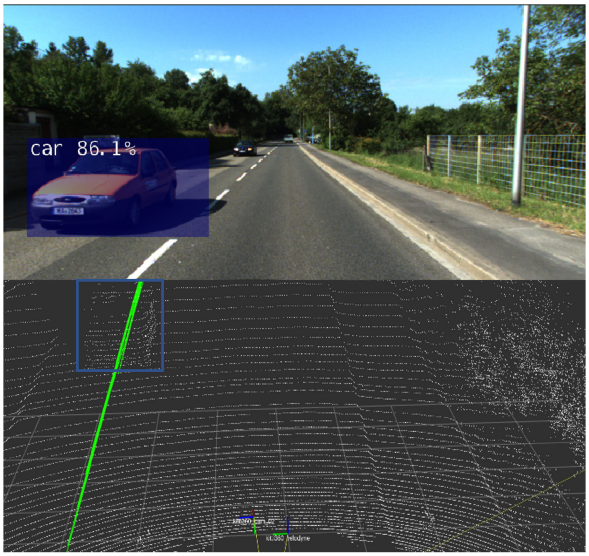


FIGURE 16. Illustration of experiments performed on KITTI-360 dataset [46]. The green line below represents the track of the car detected in the top image, displayed in the ego-vehicle coordinate frame.

using all cameras (Proposed model) for make the detailed comparison. The base, ablation and proposed cases are shown in Table 3.

For the base model (first column in Table 3) hyperparameters adopted for KITTI-360 dataset [46] were the same from Table 1, with the exception of voxel size and association thresholds which were set to $l_{voxel} = 1.0$ m, $\rho_{purecloud} = \rho_{overlay} = \rho = 5.0$ m to adapt the algorithm to a highly dynamic scenario. Other use cases (case A-D in Table 3) were defined for ablation purposes, including the change of detection backbone from Mobilenet to Inception-v2 [47]. Finally, the proposed model with hyper parameters and all four cameras and LiDAR is described in case E in Table 3.

Table 4 shows the results for the experiments with KITTI-360 dataset with total distance traversed by ego-vehicle of 35 m. In every case, the experiment was performed four times. It is evident that using the simulated environment of a road demonstrated that the algorithm is also able to perform detections when the platform is mounted in a moving vehicle. Even though the proposed algorithm together with surround view setup (case E) shows an error slight higher than the base case, the difference is not much representative in practical terms and also Case E was able to detect an additional vehicle which the rest of the cases (in Table 3) failed to detect. The implication is that the proposed surround

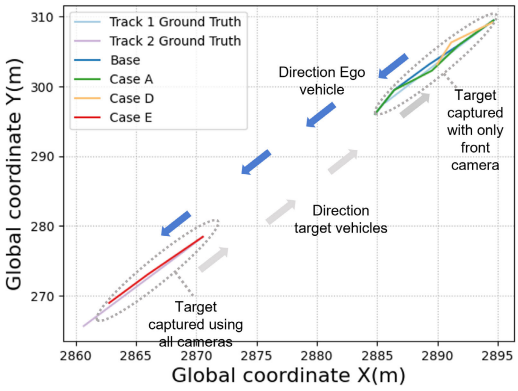


FIGURE 17. Localization of two vehicles captured using different cases and their track ground truths on KITTI-360 dataset [46] (Cases B, C ommitted for brevity). An additional car was only detected in Case E due to the extra cameras that captured it in lateral view.

view set up together with the proposed algorithm is suitable for safer navigation.

Adding up detections from a surround view using all cameras (Case E) has errors comparable to other cases with one camera (Base model, and cases A-D), but brings the advantage of the increased field of view for safer navigation. Notably, the front camera somehow failed to detect the second vehicle, possibly due to change in lightning or shadowing. In turn, the lateral camera captured it, as seen in Fig. 17.

The larger error values and their variance can be explained by the shape of the target objects. For simplification, the ground truth values considered were the center of bounding boxes. Due to the nature of the algorithm, which captures the closest displacement centroid, there will always be an error that is proportional to the distance between the center of the car and the extremities captured by the algorithm. Considering the safety aspect, the algorithm always captures the closer extremities and underestimate the distance to the objects, which is favourable from the safety point of view. The slight higher latency is caused by the larger amount of points captured by the displacement detector, once we have a moving platform in a highly dynamic scenario.

The threshold tuning was proven necessary, specially with the higher speed of target objects and the ego-vehicle. A smaller value of $\rho = 1$ (Case B) did not suffice to capture the moving centroids when detection failed and the tracks could not be properly updating, generating a higher error in localization. Using a smaller voxel size ($l_{voxel} = 0.5$, Case A) did not have a significant impact on the error, but caused smaller variance once smaller displacements began to be captured, smoothing out the track. On the other hand, it demanded a slightly computational effort and larger transmission latency due to the more generated centroids. Switching the detection model by another one with Inception v2 backbone (Case D) did not cause any significant changes in error or latency. Any differences in detection efficiency were compensated by association of centroids without detection.

TABLE 4. Results for base case, ablation cases that include change in parameters, cameras and backbone and proposed case, obtained in KITTI-360 dataset [46].

Case	Base	A	B	C	D	E (Proposed)
Displacement centroids calculation (s)	0.162± 0.007	0.174± 0.003	0.159± 0.006	0.16 ± 0.006	0.161 ± 0.006	0.159 ± 0.004
Centroids transmission (s)	0.003± 0.002	0.007± 0.004	0.003± 0.002	0.004± 0.002	0.003 ± 0.003	0.003 ± 0.003
Detection processing and transmission (s)	0.021± 0.001	0.020± 0.002	0.021± 0.001	0.021± 0.001	0.024 ± 0.003	0.022 ± 0.001
Fusion and track update (s)	0.002± 0.001	0.002± 0.001	0.002± 0.002	0.002± 0.002	0.002 ± 0.001	0.010 ± 0.004
Error (m)	0.303± 0.057	0.289± 0.031	6.471± 4.02	0.385± 0.041	0.361 ± 0.086	0.311 ± 0.038
Largest Error Value (m)	0.642	0.745	12.74	0.552	0.647	0.703

TABLE 5. Comparison with early fusion-based method for object localization.

	Other [22]	This paper (Ours)
Displacement centroids calculation (s)	n.a.	0.159± 0.004
Points/Centroids transmission (s)	0.201± 0.05	0.003± 0.003
Detection processing and transmission (s)	0.021± 0.001	0.022± 0.001
Fusion and track update (s)	1.882± 0.042	0.010± 0.004
Error (m)	1.575± 0.939	0.311± 0.038
Largest Error Value (m)	2.613	0.703

TABLE 6. Comparison with other deep learning fusion works for object localization.

Authors	MV3D [24]	AVOD [25]	Beacons [32]	This paper (Ours)
Embedded	No	No	Yes	Yes
$>$ 1 computing device	No	No	No	Yes
Joint multimodal training	Yes	Yes	No	No
Sensors	2	2	2	5
Asynch. Fusion	No	No	No	Yes
Time (ms)	360	80	200	142 (Indoor 4 person)/167 (KITTI-360 [46])

C. COMPARISON WITH OTHER METHODS

To compare our proposed method, we further implemented an early fusion algorithm that is widely used in the literature as reference. Fusing data from LiDAR and camera before performing any other downstream tasks is a very traditional method present not only for object localization, but in several other domains [13], [22], [23]. It overlays each 3D point from LiDAR complete point cloud in the image pixel space, so each 3D point is mapped to its corresponding image pixel. We implemented the reference algorithm by using the same distributed hardware and detection pipelines, but transmitting all points from point cloud to the fusion embedded device instead of extracting and sending only displacement centroids. The mapping in pixel space occurred for the complete point cloud in this case.

Using KITTI-360 dataset, we measured the fusion time for both cases (reference and our work) and further measured the error of object point positions by considering the ground truth center of 3D bounding boxes offered for the dataset. The results are shown in Table 5. It is possible to see the huge disparity in error due to the lack of centroid association in the baseline algorithm. It also demanded much higher transmission time because of the size of the unprocessed point cloud. The comparison revealed the advantages of our proposal in relation to the other algorithm.

It is worth pointing out that other state-of-art works are incompatible with distributed processing because of large monolic deep learning models and also report their results on public datasets with metrics combining average intersection over union for bounding boxes and classification accuracy (which is not the case here). Since our proposal accounts for distributed edge architecture, an independently pre-trained image detection model and outputs a simplified point representation, a suitable comparative implementation of those works becomes unfeasible. However, we provided a more global comparative table with state-of-art deep learning based models in Table 6. Even though some of those archi-

tectures provide additional information (3D bounding box of objects), our proposal is faster than state-of-art solutions. This is attained by exploiting the computing capabilities of individual devices in distributed fashion and minimal transmission latency between them.

VII. CONCLUSION AND DISCUSSION

We presented a method for multiple object localization by asynchronously fusing data of different modality and dimensions, obtained from four cameras and a scanning LiDAR distributed over embedded devices.

We addressed the challenge of flexibiliy in the LiDAR-camera fusion by imposing a clear separation between sensor interface and fusion. We eliminated the need of synchronization and accurate calibration among sensors by using only displacement centroids from point cloud and association with point tracks. Notably, our system maintains object tracking even in the presence of camera pipeline failure or intermittent streaming.

Our methodology bridged the gap between real-time fusion of the sensors and distributed computing, by allowing transmission of only detection information and compact representation from the point clouds.

The fusion time results obtained in a highly dynamic dataset (167 ms) demonstrated that our work is competitive with large monolithic fusion architectures, with the advantage of distribution over constrained edge devices. The robustness against object detection intermittency, along with location accuracy of $\approx$ 6 *cm*, reveal that the system is compliant with safety. The solution could be easily integrated with real-time planning, control and navigation for autonomous systems in open areas. Specifically, our proposal suits applications where collision must be avoided in all four directions, such as industrial plants, human populated areas or automated warehouses. The distributed nature of the system is also well-suited for in-vehicle networking.

Looking ahead, future work should focus on expanding the capabilities of our system, such as using other sensor modalities, experimenting in different weather conditions. Also, the direction orthogonal to the navigation plane can be further considered for expanding the applications in domains with depth navigation.

REFERENCES

[1] T. Kim, S. Lim, G. Shin, G. Sim, and D. Yun, "An open-source low-cost mobile robot system with an RGB-D camera and efficient real-time navigation algorithm," *IEEE Access*, vol. 10, pp. 127871–127881, 2022.

[2] F. Rovira-Más, V. Saiz-Rubio, and A. Cuenca-Cuenca, "Augmented perception for agricultural robots navigation," *IEEE Sensors J.*, vol. 21, no. 10, pp. 11712–11727, May 2021.

[3] Y. Lee and W. You, "EBAT: Enhanced bidirectional and autoregressive transformers for removing hairs in hairy dermoscopic images," *IEEE Access*, vol. 11, pp. 14225–14235, 2023.

[4] H. Wang, H. Wu, Q. Hu, J. Chi, X. Yu, and C. Wu, "Underwater image super-resolution using multi-stage information distillation networks," *J. Vis. Commun. Image Represent.*, vol. 77, pp. 1047–3203, May 2021.

[5] X. Zhao, P. Sun, Z. Xu, H. Min, and H. Yu, "Fusion of 3D LiDAR and camera data for object detection in autonomous vehicle applications," *IEEE Sensors J.*, vol. 20, no. 9, pp. 4901–4913, May 2020.

[6] H. A. Abdelhafez, H. Halawa, K. Pattabiraman, and M. Ripeanu, "Snowflakes at the edge: A study of variability among NVIDIA Jetson AGX xavier boards," in *Proc. 4th Int. Workshop Edge Syst., Anal. Netw.*, Apr. 2021, pp. 1–6.

[7] B. Yan, L. Xiao, H. Zhang, D. Xu, L. Ruan, Z. Wang, and Y. Zhang, "An adaptive template matching-based single object tracking algorithm with parallel acceleration," *J. Vis. Commun. Image Represent.*, vol. 64, Oct. 2019, Art. no. 102603.

[8] D. J. Norris, *Beginning Artificial Intelligence With the Raspberry Pi*. Barrington, NH, USA: Apress, 2017.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[10] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. 18th Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.* Cham, Switzerland: Springer, vol. 9351, 2015, pp. 234–241.

[11] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988.

[12] Z. Qin, J. Wang, and Y. Lu, "MonoGRNet: A geometric reasoning network for monocular 3D object localization," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, Jan. 2019, pp. 8851–8858.

[13] M. Jindal, A. Jha, and L. R. Cenkeramaddi, "Bollard segmentation and position estimation from LiDAR point cloud for autonomous mooring," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5700909.

[14] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3D point clouds: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 12, pp. 4338–4364, Dec. 2021.

[15] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 77–85.

[16] Y. Zhou and O. Tuzel, "VoxelNet: End-to-end learning for point cloud based 3D object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4490–4499.

[17] J. Nie, J. Yan, H. Yin, L. Ren, and Q. Meng, "A multimodality fusion deep neural network and safety test strategy for intelligent vehicles," *IEEE Trans. Intell. Vehicles*, vol. 6, no. 2, pp. 310–322, Jun. 2021.

[18] L. Yuan, J. Andrews, H. Mu, A. Vakil, R. Ewing, E. Blasch, and J. Li, "Interpretable passive multi-modal sensor fusion for human identification and activity recognition," *Sensors*, vol. 22, no. 15, p. 5787, Aug. 2022.

[19] M. P. Muresan, I. Giosan, and S. Nedevschi, "Stabilization and validation of 3D object position using multimodal sensor fusion and semantic segmentation," *Sensors*, vol. 20, no. 4, p. 1110, Feb. 2020.

[20] M. Lu, C. Hsu, and Y. Lu, "Image-based system for measuring objects on an oblique plane and its applications in 2-D localization," *IEEE Sensors J.*, vol. 12, no. 6, pp. 2249–2261, Jun. 2012.

[21] C. Zou, B. He, M. Zhu, L. Zhang, and J. Zhang, "Scene flow estimation by depth map upsampling and layer assignment for camera-LiDAR system," *J. Vis. Commun. Image Represent.*, vol. 64, Oct. 2019, Art. no. 102616.

[22] A. Jha, D. Subedi, P. Løvsland, I. Tyapin, L. R. Cenkeramaddi, B. Lozano, and G. Hovland, "Autonomous mooring towards autonomous maritime navigation and offshore operations," in *Proc. 15th IEEE Conf. Ind. Electron. Appl. (ICIEA)*, Nov. 2020, pp. 1171–1175.

[23] D. Subedi, A. Jha, I. Tyapin, and G. Hovland, "Camera-LiDAR data fusion for autonomous mooring operation," in *Proc. 15th IEEE Conf. Ind. Electron. Appl. (ICIEA)*, Nov. 2020, pp. 1176–1181.

[24] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6526–6534.

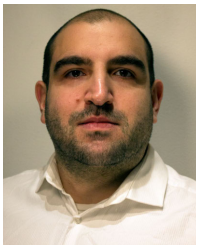
[25] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3D proposal generation and object detection from view aggregation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2018, pp. 1–8.

[26] Z. Zhang, Z. Liang, M. Zhang, X. Zhao, H. Li, M. Yang, W. Tan, and S. Pu, "RangeLVDet: Boosting 3D object detection in LiDAR with range image and RGB image," *IEEE Sensors J.*, vol. 22, no. 2, pp. 1391–1403, Jan. 2022.

[27] S. Pang, D. Morris, and H. Radha, "CLOCs: Camera-LiDAR object candidates fusion for 3D object detection," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 10386–10393.

[28] Z. Huang, C. Lv, Y. Xing, and J. Wu, "Multi-modal sensor fusion-based deep neural network for end-to-end autonomous driving with scene understanding," *IEEE Sensors J.*, vol. 21, no. 10, pp. 11781–11790, May 2021.

[29] D. S. Breland, A. Dayal, A. Jha, P. K. Yalavarthy, O. J. Pandey, and L. R. Cenkeramaddi, "Robust hand gestures recognition using a deep CNN and thermal images," *IEEE Sensors J.*, vol. 21, no. 23, pp. 26602–26614, Dec. 2021.
[30] T. Sun, W. Pan, Y. Wang, and Y. Liu, "Region of interest constrained negative obstacle detection and tracking with a stereo camera," *IEEE Sensors J.*, vol. 22, no. 4, pp. 3616–3625, Feb. 2022.
[31] Y. Fu, D. Tian, X. Duan, J. Zhou, P. Lang, C. Lin, and X. You, "A camera–radar fusion method based on edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Oct. 2020, pp. 9–14.
[32] P. Wei, L. Cagle, T. Reza, J. Ball, and J. Gafford, "LiDAR and camera detection fusion in a real-time industrial multi-sensor collision avoidance system," *Electronics*, vol. 7, no. 6, p. 84, May 2018.
[33] M. Verucchi, L. Bartoli, F. Bagni, F. Gatti, P. Burgio, and M. Bertogna, "Real-time clustering and LiDAR-camera fusion on embedded platforms for self-driving cars," in *Proc. 4th IEEE Int. Conf. Robotic Comput. (IRC)*, Nov. 2020, pp. 398–405.
[34] G. Bradski. (2010). *OpenCV Tutorial*. [Online]. Available: http://www.opencv.org
[35] O. Sorkine, "Least-squares rigid motion using SVD," *Computing*, vol. 1, no. 1, pp. 1–5, 2017.
[36] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, Apr. 2000.
[37] W. Liu, "SSD: Single shot MultiBox detector," in *Proc. ECCV*, vol. 9905. Cham, Switzerland: Springer, Dec. 2016, pp. 21–37.
[38] H. Vanholder. (2016). *Efficient Inference With Tensorrt*. [Online]. Available: https://on-demand.gputechconf.com/gtc-eu/2017/presentation/23425-han-vanholder-efficient-inference-with-tensorrt.pdf
[39] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2014, pp. 740–755.
[40] D. Meagher. (1982). *Octree Generation, Analysis and Manipulation*. [Online]. Available: https://apps.dtic.mil/sti/citations/ADA117450
[41] T. Yang, C. Cappelle, Y. Ruichek, and M. El Bagdouri, "Online multi-object tracking combining optical flow and compressive tracking in Markov decision process," *J. Vis. Commun. Image Represent.*, vol. 58, pp. 178–186, Jan. 2019.
[42] X. Weng, J. Wang, D. Held, and K. Kitani, "3D multi-object tracking: A baseline and new evaluation metrics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 10359–10366.
[43] (2021). *High-Resolution OS1 LiDAR Sensor: Robotics, Trucking, Mapping | Ouster*. [Online]. Available: https://ouster.com/products/scanning-lidar/os1-sensor/
[44] (2021). *TurtleBot*. [Online]. Available: https://www.turtlebot.com/
[45] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, vol. 3, 2009, p. 5.
[46] Y. Liao, J. Xie, and A. Geiger, "KITTI-360: A novel dataset and benchmarks for urban scene understanding in 2D and 3D," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 3292–3310, Mar. 2023.
[47] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.



JOSE AMENDOLA received the Diploma degree in electrical engineering and the master's degree in mechatronics engineering from the University of São Paulo, Brazil, in 2013 and 2020, respectively. He is currently pursuing the Ph.D. degree with the University of Agder, Norway. In 2013, he was a Researcher with the Karlsruhe Institute of Technology, Germany. He was also a Software Engineer for test automation, immersive simulators, and machine learning applications. His research interests include machine learning, sensor fusion, and perception for autonomous systems.

AVEEN DAYAL received the bachelor's degree in computer science and engineering, in 2020. He is currently pursuing the Ph.D. degree with the Department of Artificial Intelligence, Indian Institute of Technology, Hyderabad. He was a Visiting Research Student with the Department of Information and Communication Technology (ICT), University of Agder, Grimstad, Norway. His main research interest includes deep learning methods for autonomous ground and aerial vehicles.

LINGA REDDY CENKERAMADDI (Senior Member, IEEE) received the master's degree in electrical engineering from the Indian Institute of Technology Delhi (IIT Delhi), New Delhi, India, in 2004, and the Ph.D. degree in electrical engineering from the Norwegian University of Science and Technology (NTNU), Trondheim, Norway, in 2011.

He is currently the Leader of the Autonomous and Cyber-Physical Systems (ACPS) Research Group, University of Agder, Grimstad, Norway, where he is also a Professor. He is a Principal Investigator and a Co-Principal Investigator of many research grants from the Norwegian Research Council. He has coauthored more than 120 research publications that have been published in prestigious international journals and standard conferences. His research interests include the Internet of Things (IoT), cyber-physical systems, autonomous systems, robotics and automation involving advanced sensor systems, and computer vision.

Dr. Cenkeramaddi is a member of ACM and a member of the editorial boards of various international journals and the technical program committees of several IEEE conferences. Several of his master's students have received the Best Master Thesis Award in information and communication technology (ICT).

AJIT JHA received the B.Sc. degree in electronics and communication engineering, Bangladesh, in 2007, the European master's degree in photonic networks from Aston University, Birmingham, U.K., and Scuola Superiore Sant Anna, Pisa, Italy, in 2012, and the Ph.D. degree from the Technical University of Catalunya, Barcelona, Spain, and the Karlsruhe Institute of Technology, Karlsruhe, Germany, in 2016. From 2016 to 2019, he was with various industries related to autonomous vehicles involved in innovative technologies, such as automotive ethernet, ADAS, surround view systems, camera mirror systems, and blind sport warning, to name a few. He is currently an Associate Professor of mechatronics with the Department of Engineering Sciences, University of Agder, Grimstad, Norway. He has coauthored more than 20 articles and holds two patents. His research interests include sensors, sensor fusion, image/signal processing, ML, ADAS functionalities toward autonomous systems, and the IoT. He was a recipient of the Erasmus Mundus Master's Course (EMMC) and Erasmus Mundus Joint Doctorate (EMJD) funded by the European Union (EU). In addition, he has been an active reviewer and a member of the technical program committee of numerous international peer-reviewed journals and conferences.