ARC: Accurate, Real-time, and Scalable Multi-vehicle Cooperative Perception

Kaleem Nawaz Khan
Rochester Institute of Technology
kk5271@rit.edu

Fawad Ahmad
Rochester Institute of Technology
fawad@cs.rit.edu

Abstract

To overcome line-of-sight limitations in 3D sensors, cooperative perception shares sensor information between vehicles in real-time. Core to cooperative perception is the alignment of sensor data in multiple coordinate systems. Existing techniques use 3D maps and GPS for alignment, but these can lead to inaccurate alignments. However, improving alignment accuracy incurs additional compute latency, which is not desirable. In this paper, we present ARC[1] , a system that navigates carefully that tradeoff between latency and accuracy for point cloud alignment. ARC uses an anchor-based alignment technique to align vehicles to a common coordinate system. It minimizes latency by using grid-based spatial reasoning to perform alignment only in overlapping regions of the point clouds. ARC reuses the same spatial reasoning to selectively share only the most relevant data among vehicles. ARC fuses point clouds from up to 40 vehicles, incurring only 20 ms latency and under 7 cm alignment error, as demonstrated on both real-world and simulated traces.

CCS Concepts

• **Computing methodologies** → **Cooperation and coordination**.

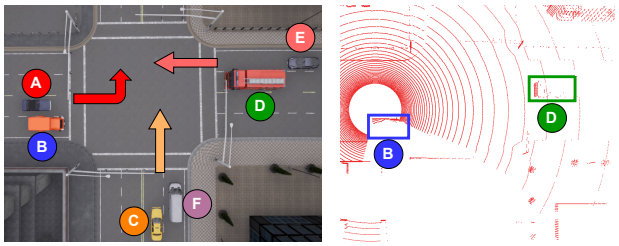
Keywords

Autonomous Cars, Multi-vehicle Cooperative Perception, Collaborative Sensing

ACM Reference Format:
Kaleem Nawaz Khan and Fawad Ahmad. 2026. ARC: Accurate, Real-time, and Scalable Multi-vehicle Cooperative Perception. In *ACM/IEEE Conference on Embedded Artificial Intelligence and Sensing Systems , May 11–14, 2026, Saint-Malo, France.* ACM, New York, NY, USA, 14 pages.

1 Introduction

Critical to the operation of autonomous vehicles is their ability to understand the environment around them. For this, they use sensors like cameras [8], Radars [43] and LiDARs [31] *etc.* LiDARs send out millions of light rays multiple times per second and, from their returns, construct 3D point clouds. These are data structures that consist of a large number of points, each point is defined by its 3D position along with other attributes like intensity and reflectivity. However, LiDARs are prone to occlusions and line-of-sight limitations. This limitation



(a) A turns left, while E and C head straight. (b) Vehicle A's LiDAR point cloud.

Figure 1: Bird's-eye view of an intersection where (a) vehicle A turns left while other vehicles head straight, (b) In A's point cloud, only B and D are visible; C and F are occluded by B, and E is occluded by D.

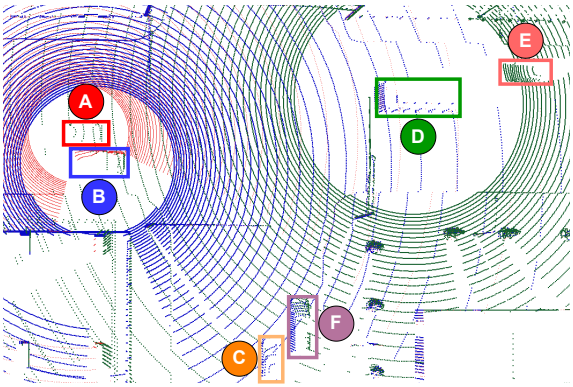


Figure 2: After fusing B's (blue) and D's (green) point clouds with A's (red) point cloud, A can clearly see all the vehicles.

is especially prevalent at traffic intersections where occlusions are common due to the denser traffic flow. Intersections account for nearly half of all fatal accidents in the U.S. [30].

Cooperative perception solves this challenge by having vehicles (and road sensors [21]) share 3D point clouds with each other [54]. For instance, in Fig. 1, vehicle A cannot perceive vehicles C, F, and E because they are occluded by vehicles B and D. If vehicles B and D share their point clouds with vehicle A, it can overcome its occlusion. After receiving the point clouds from vehicle B or vehicle D, vehicle A aligns the point cloud to its own coordinate system. Then, it appends the aligned point clouds to its own to build a fused point cloud (Fig. 2). This is called cooperative perception.

While cooperative perception is conceptually straightforward, its application to autonomous (or human) driving imposes strict latency and accuracy constraints. Autonomous vehicles must sense their surroundings, plan a path, and actuate at least 10 times per second with a tail latency under 100 ms [6, 26]. Moreover, they must localize themselves and surrounding objects with cm-level accuracy [4]. Naturally, the requirements apply to cooperative perception as well.

Accurate and fast cooperative perception introduces a number of system-level challenges. Vehicles must share and align point clouds

[1]GitHub Repository: https://github.com/nsslofficial/ARC

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
Sensys '26, Saint-Malo, France
© 2026 Copyright held by the owner/author(s).

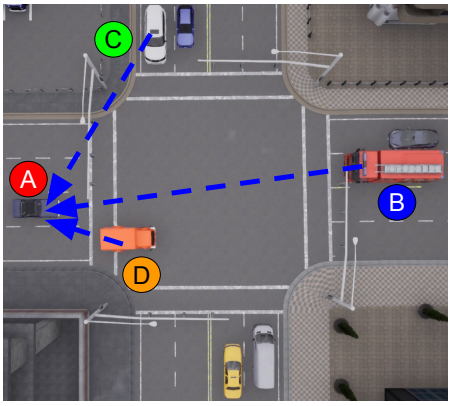


Figure 3: All vehicles align with the anchor (vehicle A).

at low latency, leaving ample time for downstream perception modules (object detection, motion forecasting, drivable space detection) to consume the fused data. The fused data should also be correct *i.e.,* the 3D point clouds must be aligned accurately. Lastly, this capability should scale to a large number of vehicles. However, traditionally, low latency and high accuracy are conflicting goals due to network and compute constraints. Point cloud alignment is compute-intensive and can be erroneous. Moreover, because 3D point clouds are voluminous [35, 45], sharing multiple point clouds over the wireless network can easily exhaust it. Although prior works [34, 54] address network bottlenecks by exchanging only the most relevant information among vehicles, *to our knowledge, achieving accurate alignment across multiple vehicles remains relatively unexplored.*

To better understand, we describe how point cloud alignment is performed. The first set of techniques, called direct alignment, match overlapping regions between point clouds (and the features in them) to compute a transformation matrix. This transformation matrix, when applied to one point cloud, aligns it to the other's coordinate system. If a vehicle has to align with multiple vehicles, it must directly align with every vehicle. For instance, if a vehicle needs data from four vehicles, it might directly align with all four vehicles. Consequently, this approach does not scale. To this end, prior works [33, 34] use indirect alignment *i.e.,* they align all vehicles to a shared 3D map[2]. Because every vehicle aligns itself to the same 3D map, they indirectly align with one another. However, 3D maps can contain noise because they are built by aligning thousands of point clouds [25]. *Noise in the 3D map can significantly degrade point cloud alignment accuracy.*

Both approaches exhibit trade-offs. Direct alignment is accurate but incurs significant latency and does not scale well. Indirect alignment, on the other hand, is fast and scalable but suffers from lower accuracy. In this paper, we ask the question: *can we design a cooperative perception system that enables high accuracy alignment at low latency and scale to a large number of vehicles?*

Our Approach. To simultaneously achieve high accuracy and low latency, in this paper, we propose an anchor-based alignment technique. Instead of directly aligning with each other, the vehicles align with a common anchor. This anchor can be a LiDAR mounted on the roadside or a moving vehicle. Once every vehicle aligns itself with the anchor, they are indirectly aligned with one another. For example, in Fig. 3, vehicle A is the anchor with which other vehicles (B, C, and D) align themselves.

Anchor-based alignment has two benefits. Although this is an indirect alignment, it achieves high accuracy. This is because vehicles align themselves to a single LiDAR frame as opposed to an accumulation of LiDAR frames (3D map) which can be prone to noise. Second, it ensures low latency because vehicles only need a single alignment operation *i.e.,* with the anchor point cloud.

Challenges. While anchor-based alignment addresses the challenges of prior techniques, it introduces challenges of its own.

- Every vehicle must align its point cloud with the anchor's point cloud at every frame. This point cloud alignment must be fast to allow the downstream perception modules to consume the fused point cloud. At the same time, the alignment must also be accurate.
- Second, once aligned, if all vehicles at the intersection share their raw point clouds, it would easily exhaust the wireless network. So, vehicles should only exchange relevant 3D data. Reasoning about which data is relevant can be computationally intensive.

Contributions. We tackle these challenges by building an end-to-end system, ARC. It makes the following contributions.

- For fast alignment, instead of aligning the entire point clouds, we compute the overlapping regions between pairs of point clouds and only align those regions. To do this, we propose a fast spatial reasoning module that uses an underlying shared 3D grid to determine the overlapping regions between the point clouds.
- Instead of using a separate module to determine what data to send to each vehicle, we reuse the 3D grid. By doing so, we enable accurate alignment without trading off computation latency. Moreover, we design the 3D grid so that operations on it can be easily offloaded to the GPU. This enables us to identify overlapping regions and blind spots almost instantly.

An end-to-end implementation of ARC aligns vehicles with up to cm-level accuracy at an end-to-end compute latency of 20 ms.

2 Background and Motivation

Limitations of 3D Sensors. Autonomous vehicles are equipped with depth perception sensors like LiDARs. A LiDAR consists of several radially positioned laser beams. Together, these laser beams emit millions of light rays per second and, from their reflections, builds a 3D point cloud of the environment. This point cloud consists of 3D points, defined by their positions and other attributes like intensity and reflectivity. However, because LiDARs build point clouds from reflected light rays, like human vision, they are prone to occlusions and line-of-sight limitations. For instance, vehicle A's point cloud does not contain reflections from vehicle C because it is occluded by vehicle B (Fig. 1). Additionally, the radial arrangement of a LiDAR's beams causes point density to decrease with distance, the density of points (or reflections) drops significantly with distance. In other words, the number of reflected points from an object of the same size placed further away from the LiDAR is smaller than that of objects nearby. Consequently, object detectors [22, 23] may not accurately recognize objects further away from the LiDAR.

Cooperative Perception. To extend a vehicle's perception range, cooperative perception shares 3D point clouds between vehicles

[2]This is a large point cloud that has a 1:1 correspondence with the physical world that vehicles use to localize themselves in the world.

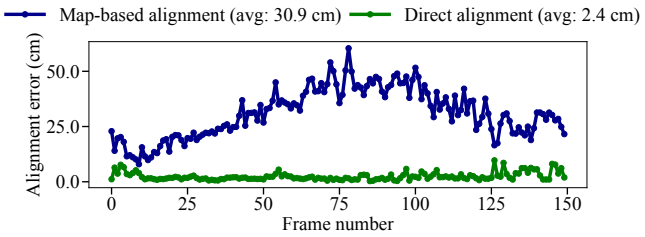


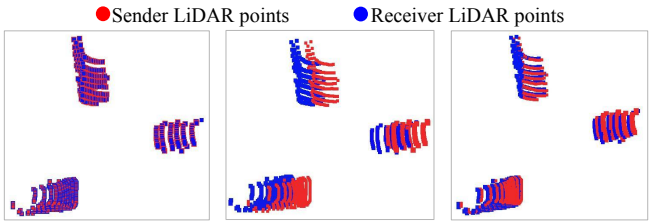
Figure 4: Indirectly aligning vehicles through a 3D map has an order of magnitude higher alignment error as compared to direct alignment.

(and roadside sensors) over the wireless network. To do this, at every frame, one vehicle (the sender) transmits its point cloud over the wireless network to the other vehicle (the receiver). Upon receiving the point cloud, the receiver aligns the point cloud in its own coordinate system. After alignment, the receiver appends the aligned point cloud with its own to build a fused point cloud. Then, downstream perception modules consume the fused point cloud.

Requirements for Cooperative Perception. An autonomous vehicle must sense, plan, and react to its surroundings at least 10 times per second with a tail latency less than 100 ms [26]. Moreover, it must position itself and the surroundings objects with up to cm-level accuracy. Consequently, cooperative perception must also be fast and accurate. If cooperative perception cannot build the fused point cloud at low latency, by the time downstream perception modules consume the data, it will be stale. Similarly, if point clouds are not aligned with cm-level accuracy, it can be catastrophic for downstream modules (like scene understanding and planning), which will operate on incorrect data. Broadly, cooperative perception approaches can use early or late fusion. Late fusion saves network bandwidth by transmitting compact, task-specific features (*e.g.,* bounding boxes). However, aligning compact features is often inaccurate and yields a processed fused representation that is not usable for all downstream applications [39]. Early fusion may consume more bandwidth because it transmits raw point clouds, but alignment is more accurate [21]. Moreover, the resultant fused point cloud can be readily processed by downstream perception and planning modules.

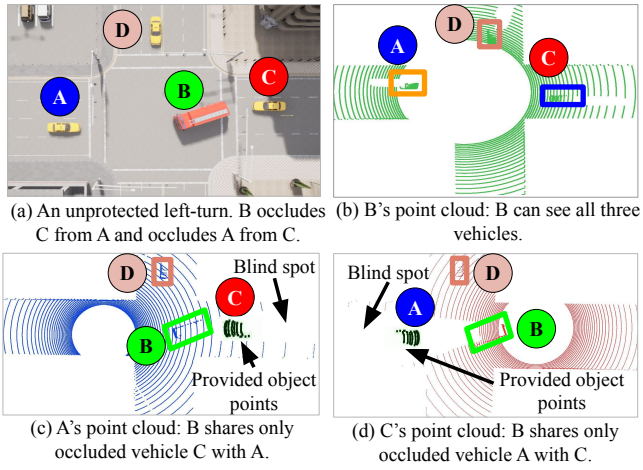
Point Cloud Fusion. Point cloud fusion is a two-step process: alignment and stitching. Given point clouds P_a and P_b captured by vehicles A and B, the goal of point cloud alignment is to find a rigid transformation T_{a-b} that aligns P_a to P_b. The transformation T_{a-b} is a 4x4 matrix, consisting of a rotation matrix R and a translation vector t. This transformation T_{a-b}, when applied to P_a, yields a transformed point cloud $P_a^{'}$ in the same coordinate system as P_b. Stitching simply appends $P_a^{'}$ to P_b to obtain a fused point cloud P_f that contains all the points from $P_a^{'}$ and P_b.

Iterative Closest Point (ICP [36]). Point cloud alignment algorithms like ICP iteratively compute a transformation matrix T_{a-b} that minimizes the 3D distance from every point in P_a to its nearest neighbor in P_b. To do this, ICP requires both a large overlap between the two point clouds and a coarse-grained alignment between them. This coarse-grained alignment is an accurate initial guess of the transformation matrix T_{a-b}. If the two vehicles are far away and/or they do not have an accurate initial alignment, ICP will not be able to accurately align the two point clouds.



(a) Ground truth alignment (b) Indirect alignment (error: 30 cm) (c) Direct alignment (error: 5 cm)

Figure 5: Indirect alignment and direct alignment errors in the fused point cloud.



(a) An unprotected left-turn. B occludes C from A and occludes A from C. (b) B's point cloud: B can see all three vehicles.

(c) A's point cloud: B shares only occluded vehicle C with A. (d) C's point cloud: B shares only occluded vehicle A with C.

Figure 6: Vehicle B shares only the relevant dynamic objects (A and C) with C and A, respectively.

Direct Vs Indirect Alignment. Direct alignment feeds the two point clouds (P_a and P_b) to an alignment algorithm, such as ICP, to compute a transformation matrix (T_{a-b}). Although direct alignment is accurate, it can incur significant latency if a vehicle has to align with 3D point clouds from multiple vehicles. For instance, in Fig. 2, to get a complete scene understanding, vehicle A needs to align with two different vehicles. This entails two direct alignments. To tackle this, prior works use indirect alignment.

Indirect alignment aligns both the point clouds to a third point cloud (*e.g.,* a 3D map). This produces two transformation matrices (T_{a-m} and T_{b-m}). Using these matrices, we can compute the alignment between P_a and P_b as $T_{a-b} = T_{a-m} * T_{b-m}^{-1}$. However, because a 3D map can contain noise, this can lead to inaccurate alignments.

To demonstrate this, we collected LiDAR traces from two vehicles and then aligned them using direct and indirect alignment. Fig. 4 shows that the error for indirect alignment (map-based alignment) can be an order of magnitude greater than direct alignment. Qualitatively, Fig. 5 illustrates the effect of alignment error on the fused point cloud. A 5 cm alignment error (for direct alignment) is hardly noticeable, whereas a 30 cm error (indirect alignment) can cause misdetections and errors in downstream perception modules. In this paper, we ask the question: *how can we achieve accurate alignment at low latency for a large number of vehicles.*

Network Bottleneck. LiDAR sensors such as the Hesai OT128 [41], a 128-beam LiDAR with a 200 m range and 360° field of view, can

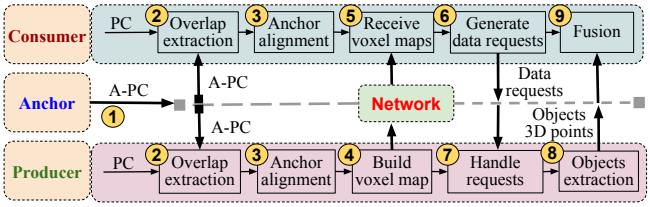


Figure 7: An overview of ARC.

produce up to 6.91 million points per second, corresponding to data rates of around 1 Gbps. Sending such high-bandwidth data over a wireless link can easily exhaust the available network capacity. To tackle this, prior works [34, 54] share only relevant information with other vehicles. They extract dynamic objects and then determine the subset of those that are relevant to every vehicle. For instance, in Fig. 6a, B occludes A and C from each other. Although B can see all vehicles at the intersection (Fig. 6b), it will only share occluded vehicle C with A (Fig. 6c), and A with C (Fig. 6d).

However, determining what information is relevant can be inaccurate and inefficient. Decentralized approaches (AutoCast [34]) rely on the sender to estimate what the receiver cannot see. Consequently, they can make inaccurate determinations [54], and since there is no coordination amongst vehicles on what to share, multiple sender vehicles might end up sharing duplicate data with the receiver vehicle (resulting in higher bandwidth consumption). Centralized approaches (RAO [54]), on the other hand, require each vehicle to compute and share visibility maps or other large data structures to inform others of their blind spots, incurring additional bandwidth and computational overhead.

3 ARC Design

Overview. Consider an ARC deployment at the intersection. There will be two types of dynamic objects: compliant and non-compliant. Non-compliant objects are vehicles, pedestrians, and cyclists that do not take part in cooperative perception. Compliant objects take part in cooperative perception and are equipped with a LiDAR, computing resources, and a wireless radio. These objects can assume one or more of the following roles: anchor, producer, and consumer. All compliant objects spatially align themselves with the anchor. The anchor can be a roadside-mounted LiDAR or a vehicle. Producers share their 3D point clouds with consumers to augment their perception. Moreover, we assume all vehicles (including the anchor) share a semantic 3D map[3], wherein every 3D point, besides its 3D position, has a semantic label associated with it (a common practice in the autonomous driving industry [2, 3, 14, 58]). This label defines the object class of the point (*e.g.,* vehicle, road, building *etc.*)

All vehicles (including the anchor) localize themselves in the 3D map. After localizing itself, as illustrated in Fig. 7, the anchor broadcasts its point cloud (A-PC) for other vehicles (step 1 in Fig. 7). Upon receiving the anchor point cloud, other vehicles align their point clouds (PC) to it. Because point cloud alignment can be compute-intensive, ARC finds and aligns only overlapping regions of the point clouds (steps 2 and 3 in Fig. 7). ARC uses a fast overlap estimation algorithm that operates on a 3D grid data

[3]Existing approaches like SuMa++ [10] build semantic 3D maps by projecting 2D image segmentation results onto LiDAR points or by directly applying 3D segmentation models like RangeNet++ [28].

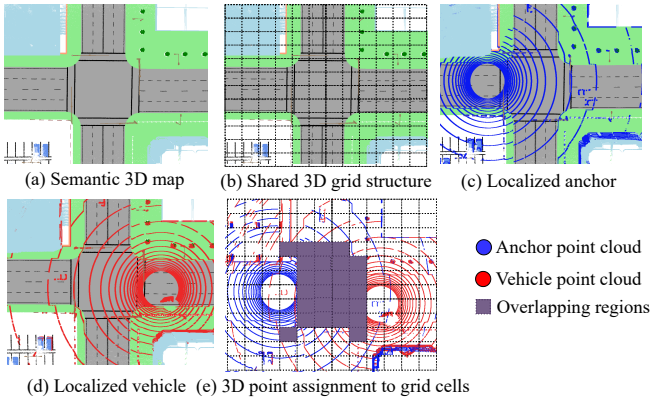


Figure 8: Computing the overlapping region between anchor and vehicle point clouds.

structure that vehicles share at the intersection. Once aligned to the anchor, vehicles can share 3D data with one another.

When sharing 3D data, ARC shares only relevant data to save network bandwidth. For this, every vehicle builds a compact scene representation on the shared 3D grid. We refer to this scene representation as the voxel map. In this representation, each vehicle declares the grid elements that are visible and occluded to it. Every vehicle broadcasts this compact representation (step 4 in Fig. 7). Upon receiving these representations from other vehicles, every vehicle tries to find other vehicles that have visibility into their occluded regions (step 5 in Fig. 7). The vehicle requesting the data is called a consumer, and the one providing the data is called the producer. The consumer requests the producer to send 3D data about those grid elements (step 6 in Fig. 7). The producer handles requests from a single or multiple consumers (step 7. in Fig. 7). It extracts the object points in the requested grid elements and then sends it to the consumers (step 8. in Fig. 7). Once the consumer receives the data, it positions and fuses it in its own point cloud and then downstream perception modules consume the data (step 9 in Fig. 7).

If the intersection has a stationary roadside unit with a LiDAR, by default, ARC will set that as the anchor. If not, then ARC uses a moving vehicle as the anchor. ARC determines the anchor vehicle based on proximity to the center of the intersection. To do this, after localizing themselves in the 3D map, all vehicles broadcast their 3D positions to others at the intersection. Then, at fixed time intervals[4], each vehicle computes the 3D distance of itself and all other vehicles from the center of the intersection. The vehicle closest to the center is selected as the anchor. In rare cases where multiple vehicles are equally close, ARC breaks the tie by first comparing the x, and then if required, the y coordinate of their 3D positions. Once selected, the anchor vehicle begins broadcasting point clouds to other vehicles. When the anchor leaves the intersection, ARC selects a new anchor.

3.1 Fast Overlap Extraction

The Problem. To reduce latency, ARC uses the anchor-based alignment. In this, all vehicles at the intersection align themselves with an anchor (a vehicle or a roadside mounted sensor). Point cloud alignment is compute-intensive and incurs latency. A single alignment operation can take up to 30 ms for point clouds generated

[4]We set this interval to half of the average time it takes vehicles to cross the intersection.

Algorithm 1: Fast overlap extraction

Input: P_{vehicle}, P_{anchor}, 3D grid G
Output: cropped P_{vehicle}, P_{anchor}

1 initialize cell assignment map M;
2 **for** $p \in P_{vehicle}, P_{anchor}$ **do**
3 find nearest cell $c \in G$;
4 append p_{vehicle} to Mc_{vehicle} and p_{anchor} to Mc_{anchor};
5 **end**
6 **for** *each cell* $c \in G$ **do**
7 $count_{P_{\text{vehicle}}} \leftarrow |Mc_{\text{vehicle}}|$, $count_{P_{\text{anchor}}} \leftarrow |Mc_{\text{anchor}}|$;
8 $\rho_{\text{vehicle}} = \frac{count_{P_{\text{vehicle}}}}{Size_{\text{cell}}}$;
9 $\rho_{\text{anchor}} = \frac{count_{P_{\text{anchor}}}}{Size_{\text{cell}}}$;
10 $\rho_{\text{int}} = \min\left(\rho_{\text{vehicle}}, \rho_{\text{anchor}}\right)$;
11 **if** $\rho_{int} > threshold$ **then**
12 mark c for cropping;
13 **end**
14 **end**
15 crop selected cells from P_{vehicle}, P_{anchor};

from a 128 beam LiDAR. This is not desirable as it does not leave ample time for downstream modules to process the fused point cloud.

Key Insight. Alignment algorithms such as ICP compute a transformation matrix that aligns two point clouds as follows. First, for every point in one point cloud, ICP finds a neighboring point in the second point cloud. Second, it computes the 3D distance between the neighboring points. Third, it computes a transformation matrix that minimizes the 3D distance calculated above. Then, it iterates over again. The accuracy and latency of ICP depend on two things: a) a coarse-grained alignment, and b) the overlapping region between the point clouds. A coarse-grained alignment is advantageous for two reasons: a) it reduces the chances of ICP converging to a local minima, and b) it reduces the number of iterations for ICP to converge to a transformation matrix. Building on prior work [21], we use vehicles' positions in a 3D map as an initial guess for alignment. These need not be accurate, since their purpose is only to reduce the search space for ICP. Lastly, finding the nearest neighbors for all 3D points can be compute-intensive. However, the overlapping regions (or common regions) between two point clouds are often relatively small. So, computing the transformation matrix does not need the entire point cloud but only the overlapping points.

Our Approach. ARC uses only the overlapping regions in two point clouds for alignment. Before aligning the two point clouds, ARC estimates the overlapping regions between the two point clouds. Then, it computes the transformation matrix for the overlapping regions and applies the transformation matrix to the entire point cloud. It is important to note though that using overlapping regions for point cloud alignment is not novel to ARC. Prior works have computed and aligned overlapping regions [21, 40]. However, computing the overlap has traditionally been compute-intensive. To this end, ARC proposes a novel and efficient algorithm (Algorithm 1) to compute the overlap between two point clouds.

As illustrated in Fig. 8, all vehicles (including the anchor) share the semantic 3D map defined earlier in §3, in which each 3D point is annotated with its object-class label (Fig. 8a). Offline, ARC overlays a 3D grid on top of the 3D map (Fig. 8b). Every vehicle (including the anchor) matches its point cloud with the 3D map to find its position in the 3D map (Fig. 8c and Fig. 8d). At every frame, the anchor vehicle broadcasts its point cloud. When vehicles receive this point cloud, using their own positions and that of the anchor in the 3D map, they project their point clouds onto the 3D grid (Fig. 8e)

After projecting both the point clouds on the grid, ARC employs Algorithm 1 to find the overlapping regions. ARC maintains a dictionary that records, for each 3D point, the index of the grid cell to which it has been assigned (Algo. 1: lines 1-5). After this assignment, for both point clouds, ARC computes the point density in each cell of the grid (Algo. 1: lines 6-9). Point density of a cell is the number of points within the cell divided by the cell's size [5]. From point densities, ARC computes interpoint density for each cell (Algo. 1: line 10). Interpoint density is the MIN of the point densities between the anchor and vehicle 3D points. Prior work [21] shows that a large number of cells with high interpoint density indicate a large overlap. This, in turn, means a higher chance of accurate alignment.

Next, ARC filters out grid cells with high interpoint density (Algo. 1: lines 11-14). From each point cloud, ARC crops 3D points belonging to the cells with high interpoint density (Algo. 1: line 15). ARC inputs only the overlapping regions to ICP and then uses the computed transformation matrix on the entire point cloud.

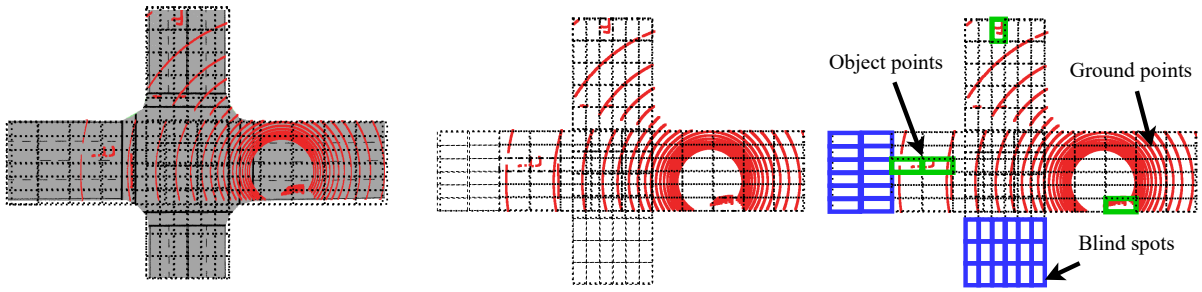
Because finding the overlapping regions is easily parallelizable, we offload this operation to the GPU. In our GPU implementation, we create a list of 3D points corresponding to the cell centers in the 3D grid. Each separate GPU thread runs a nearest neighbor search. In this search it assigns a 3D point from the point cloud to the closest center in the list. Using the same grid, we process both anchor and vehicle point clouds in parallel. After this for each cell, we calculate the point density for both anchor points and vehicle points. At the end we create a list of 3D points that belong to cells having high point density. Both these lists are the dense overlapping regions of the two-point clouds. All these operations are highly parallelizable, allowing our system to compute the overlapping regions, crop them, and align the point clouds in just a few milliseconds.

3.2 Relevant Data Extraction

The Problem. Wireless network bandwidth is limited. To this end, prior works [34, 54] extract and share only the 3D points that belong to the relevant dynamic objects in the scene. Determining what objects are relevant can be erroneous and latency-intensive. AutoCast [34], a decentralized approach, requires the sender vehicle to *estimate* what the receiver vehicle will need. This has two consequences: a) the sender vehicle can make inaccurate estimates about the receiver vehicle's blind spots [54] and b) multiple sender vehicles might transmit duplicate data to the receiver. We show and discuss this in §4.4. Centralized approaches like RAO [54], on the other hand, explicitly request the relevant 3D points from sender vehicles. To do this, they construct and transmit data structures (occupancy maps) that describes what the receiver vehicle sees. These data structures are expensive to compute and transmit over the wireless network.

Last, to achieve higher alignment accuracy, ARC adds additional computations for extracting overlapping regions and aligning them. This adds to the overall end-to-end delay. ARC must minimize the

[5] We used a grid cell size of $2 \times 3 \times 4$ meters. A larger cell size reduces latency but also lowers accuracy.



(a) Localized vehicle point cloud's drivable space

(b) 3D point assignment to grid cells

(c) Generating voxel map

Figure 9: Process of creating a voxel map representation from the vehicle's point cloud using the shared 3D grid structure.

end-to-end delay to provide ample time to downstream modules to process the fused point clouds.

Our Approach. To avoid sharing redundant data, in ARC, the receiver (consumer) vehicle explicitly requests data from one or more sender (producer) vehicles. For this, each vehicle needs to broadcast what it sees. To mitigate the compute and network overheads of sharing what the producers see and what the consumers need, ARC reuses the shared 3D grid that vehicles use to align with the anchor. Every vehicle finds out a set of indices, indicating the grid cells in which it has visibility (*i.e.,*, cells where the point density is greater than some predefined threshold).

At every frame, all vehicles (including producer, consumer, and anchor) extract drivable space from their point cloud (Fig. 9a). Drivable space is that part of the environment on which a vehicle can legally drive, *e.g.,*, the road surface. The 3D map has semantic labels for every 3D point to describe the type of object those points belong to. Because every vehicle is aligned to the 3D map, ARC can project semantic labels from the 3D map to a vehicle's point cloud and extract drivable space from it.

Next, we compute the blind spots for every vehicle. This requires spatial reasoning on the 3D point clouds, which can be expensive. Instead, we reuse computations from ARC's point cloud alignment. Recall that every vehicle has assigned 3D points from its point cloud to the shared 3D data structure (Fig. 9b). We mark grid cells where point density is below a threshold, within the vehicle LiDAR's depth range, as blind spots. Blind spots are regions of the point cloud into which the vehicle has little or no visibility.

After removing blind spots, there are two types of grid cells: with objects and without objects. Since ARC focuses only on dynamic objects, it shares only the cells containing objects. We identify between the two types of cells as follows. For the second class *i.e.,* without objects, the LiDAR point cloud contains reflections from the ground plane. While ARC could use plane-fitting algorithms (e.g., RANSAC [16]) and clustering to identify these cells, this approach is computationally expensive. Instead, ARC leverages the 3D map.

ARC subtracts the drivable space of the 3D map, at the intersection, from the vehicle's point cloud. After this subtraction, the vehicle's point cloud only has 3D points above the road surface. These points belong to dynamic objects. Because the vehicle's point cloud has already been projected in the shared 3D grid, this grid now has two types of cells: a) blind spot cells (Fig. 9c: blue boxes) and b) cells containing dynamic objects (Fig. 9c: green boxes).

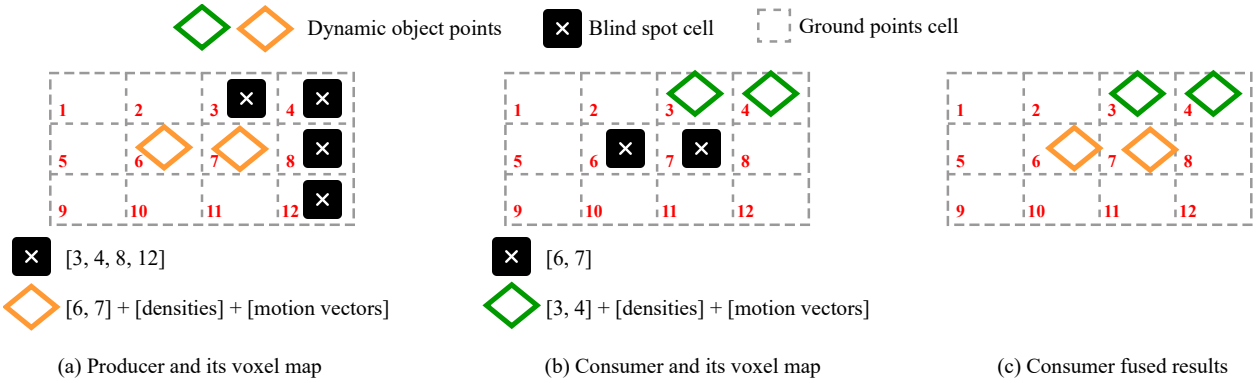
Next, we compute the motion vectors of dynamic objects in the vehicle's point cloud. For this, we follow the standard tracking approach used in previous work [54] that maintains an affinity matrix for 3D multi-object tracking. In ARC, we calculate the affinity score between occupied cells across frames using the following equation:

$$T\left(C_{t-1}, C_t\right) = w_1 \cdot L_2\left(C_{t-1}, C_t\right)^{-1} + w_2 \cdot \left|\rho_{c_{t-1}} - \rho_{c_t}\right|,$$

where C_{t-1} and C_t are the centroids of occupied cells in two consecutive frames. The first term, $L_2\left(C_{t-1}, C_t\right)^{-1}$, represents the inverse Euclidean distance between the centroids, while the second term, $\left|\rho_{c_{t-1}} - \rho_{c_t}\right|$, measures the absolute difference in point densities of the two cells. The coefficients w_1 and w_2 allow us to adjust the relative importance of these two components. For each cell in the current frame, the cell in the previous frame that maximizes $T\left(C_{t-1}, C_t\right)$ is considered to correspond to the same object. By maintaining lists of occupied cells and their point densities across consecutive frames, we construct an affinity matrix that maps cells in the current frame to those in the previous frame. Using this matrix, we then compute the motion vectors of all occupied cells.

At this stage, every vehicle has two types of cells: a) object cells (ones with dynamic objects) and b) blind spots. With every object cell, ARC associates a motion vector and point density (computed in §3.1). Next, ARC embeds this information into voxel map (Fig. 10). The voxel map is built on top of the shared 3D grid. Instead of storing the 3D location of every cell, the voxel map simply stores the index of that cell in the 3D grid. It builds two lists of indices *i.e.,* one for the blind spot cells and the other for object cells. Within the list for object cells, it embeds motion vector and point density information as well (Fig. 10a and Fig. 10b). Every vehicle broadcasts its voxel map to other vehicles at the intersection.

Every vehicle receives voxel maps from others at the intersection. Using these voxel maps, for each blind spot cell, the consumer vehicle determines which other vehicles (producers) at the intersection have that cell as an object cell with a desired point density. For instance, in Fig. 10b, the vehicle has blind spot cells 6,7. It finds another vehicle (Fig. 10a), for which these cells are object cells. The consumer requests these cells points, the producer extracts these cells and shares the points with the consumer. The consumer performs motion compensation and fuses the received points with its own objects (Fig. 10c).



(a) Producer and its voxel map (b) Consumer and its voxel map (c) Consumer fused results

Figure 10: Voxel maps and grid cells for the producer (a), the consumer (b), and the fused object points from the consumer (c).

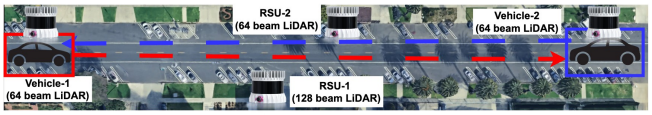


Figure 11: Real-world data collection setup with two vehicles and two RSUs.

4 Evaluation

4.1 Methodology

Implementation. We implemented ARC in C++. To build the 3D map, we used Fast-LIO2 [48]. Vehicles localize in this map using Normal Distribution Transform (NDT [5]). We used the Point Cloud Library (PCL) [38] for point cloud manipulation operations. PCL supports CUDA. ARC is 3,500 lines of C++ code. We evaluated ARC on a laptop with a 13th-generation Core i9 CPU and an NVIDIA GeForce RTX 4070 laptop.

Real-world Traces. We built a testbed to collect real-world traces using four LiDARs, two mounted on roadside units and two on moving vehicles, as shown in Fig. 11. There was other vehicular and pedestrian traffic as we collected the data. One of the roadside units was equipped with a 128-beam LiDAR, whereas the remaining roadside unit and vehicles were equipped with 64-beam LiDARs. Although we collected a large corpus of data, we manually aligned 400 point clouds for accuracy evaluations.

Synthetic Traces. We used CARLA [15], an industry-standard photorealistic simulator, to generate synthetic LiDAR traces. CARLA gives us more flexibility in simulating diverse traffic conditions and sensor configurations. Moreover, CARLA gives us ground truth alignments for point clouds, allowing us to more extensively evaluate accuracy.

Evaluation Metrics. In our evaluations, we measure the latency, accuracy, and bandwidth consumption of ARC on a per-frame basis. Latency is the time duration that starts from when a vehicle captures a LiDAR point cloud to the point in time where it fuses it with point clouds received from other vehicles. This consists of compute latency and network latency. We measure only the compute latency. This is the time to run ARC's algorithms. Network latency is the time it takes for data to travel from one vehicle to another vehicle. Network latency is more difficult to evaluate because of the scale of

Figure 12: Bird's-eye view of the intersection in CARLA used for synthetic data collection.

the experiments (*i.e.,* 40+ vehicles). For the network, we measure bandwidth consumption instead. For each vehicle, we measure the amount of data each vehicle generates to share over the network for cooperative perception.

For accuracy, we evaluate alignment accuracy. We compute relative translational error (RTE) and relative rotational error (RRE) [18]. Both metrics are the root mean square error between the estimated and ground truth transformations, with lower values indicating higher accuracy. RTE is $||t - t_g||_2$, where t is the translation vector estimated by ARC and t_g is the ground truth. RRE is $\Sigma_{i=1}^{3} |\ angle\ (i)\ |$, where the angles are extracted from $F\ (R_g^{-1} \cdot R)$. The function $F(.)$ converts a rotation matrix into three Euler angles, and R_g and R denote the ground truth and estimated rotation matrices, respectively.

Baselines. Two recent vehicle-to-vehicle cooperative perception approaches are AutoCast [34] and RAO [54]. AutoCast uses a 3D map for alignment, while RAO relies on GPS/IMU, which is known to introduce higher errors [19]. Therefore, we compare ARC only against AutoCast as a representative baseline.

Apporach	RTE (cm)			RRE (°)		
	Mean	*p95*	*p99*	*Mean*	*p95*	*p99*
AutoCast	13.1	27.4	38.14	0.2	0.38	0.47
ARC	2.0	4.87	6.4	0.1	0.14	0.16

Table 1: End-to-end accuracy comparison of ARC and AutoCast on CARLA dataset.

4.2 End-to-end Experiments: Accuracy

In this section, we evaluate ARC's ability to align point clouds accurately.

CARLA Traces. To evaluate the accuracy, we simulated vehicular traffic at an intersection in CARLA with 40 vehicles (Fig. 12). All these vehicles participate in cooperative perception. We equipped each vehicle with a 32-beam LiDAR (modeling the Velodyne VLP-32C [24]). We also mounted a roadside unit with the same LiDAR. From this experiment, we recorded 6,000 point clouds (150 point clouds for each vehicle). We then align each vehicle with every other vehicle, resulting in 50,000 unique alignments after removing duplicates. We use ARC and AutoCast [34] to estimate the transformation matrices for all vehicle pairs. Recall that AutoCast [34] uses the 3D map to align vehicles. *On average,* ARC *improves accuracy (Fig. 13 and Fig. 14) by an order of magnitude as compared to AutoCast.* The average RTE for ARC is 2 cm, whereas for AutoCast, the number is 13 cm. The average RRE for ARC is 0.1°and for AutoCast, it is 0.2°. As shown in the CDF plots (Fig. 15 and Fig. 16), ARC achieves significantly tighter alignment error distributions for both RTE and RRE, with the entire error range bounded within 7 cm and 0.15°, compared to 67 cm and 0.8° for AutoCast. Also, as reported in Tbl. 1, ARC consistently achieves substantially lower tail errors compared to AutoCast. Specifically, the 95th and 99th percentile RTE values reduce from 27 cm and 38 cm to 5 cm and 6.5 cm, while the 95th and 99th percentile RRE values improve from 0.4° and 0.5° to 0.14° and 0.16°, respectively. The improvement in alignment accuracy is because ARC utilizes an anchor point cloud for alignment, which does not accumulate noise over time. In contrast, the 3D map used by AutoCast can accumulate mapping errors due to sensor noise, inaccurate point cloud alignments, and the presence of dynamic objects. These errors, in turn, affect the accuracy of the estimated transformation matrices.

Real-world Traces. We also evaluated ARC's alignment accuracy on real-world LiDAR traces collected from our testbed, consisting of two moving vehicles and two stationary roadside units. To generate ground truth, we manually align point clouds (with a large overlap) using CloudCompare [1]. With average RTE and RRE of 6.4 cm and 0.34°, ARC demonstrates up to 2x improvement in alignment accuracy compared to AutoCast (Fig. 17 and Fig. 18). ARC achieves significantly lower tail errors compared to AutoCast. The 95th and 99th percentile RTE values decrease from 34 cm and 48 cm to 10.7 cm and 11 cm, while the 95th and 99th percentile RRE values improve from 1.22° and 1.58° to 0.53° and 0.55° (Tbl. 2), respectively. These numbers are somewhat higher than those from the synthetic traces, highlighting the inherent complexity of real-world environments.

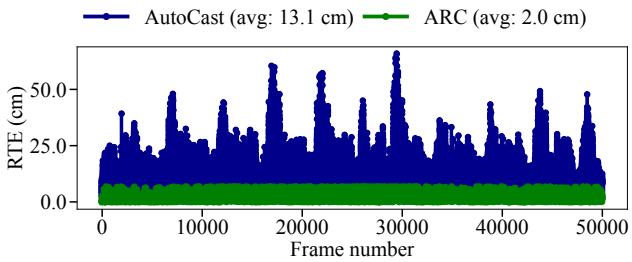


Figure 13: Comparison of ARC and AutoCast accuracy (RTE) on the CARLA dataset.

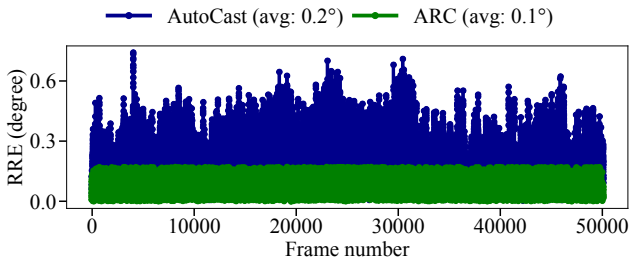


Figure 14: Comparison of ARC and AutoCast accuracy (RRE) on the CARLA dataset.

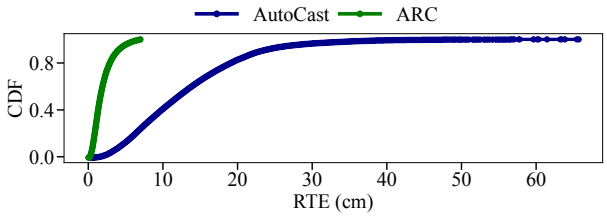


Figure 15: Comparison of CDFs illustrating ARC and AutoCast accuracy (RTE) on the CARLA dataset.

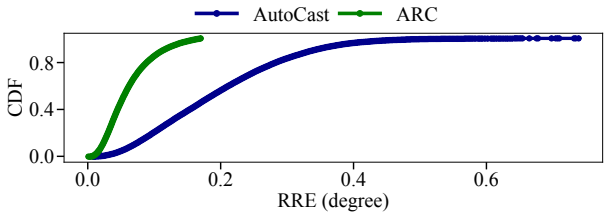


Figure 16: Comparison of CDFs illustrating ARC and AutoCast accuracy (RRE) on the CARLA dataset.

4.3 End-to-end Experiments: Latency

Next, we measure ARC's end-to-end compute latency. It is the time a vehicle takes to prepare its own data for sending, plus the time required to process incoming data from other vehicles and fuse it with its own point cloud. We measure the compute latency for AutoCast and ARC for each vehicle on the 40-vehicle CARLA dataset. We evaluate both approaches using the same compute platform as described in §4.1. For a fair comparison, we exclude the latency for path planning and collision avoidance in AutoCast. *Even so,* ARC

Apporach	RTE (cm)			RRE (°)		
	Mean	*p95*	*p99*	*Mean*	*p95*	*p99*
AutoCast	13.30	34.3	48.0	0.55	1.22	1.58
ARC	6.40	10.7	11.0	0.34	0.53	0.55

Table 2: End-to-end accuracy comparison of ARC and AutoCast on the real-world dataset (including 95th and 99th percentiles).

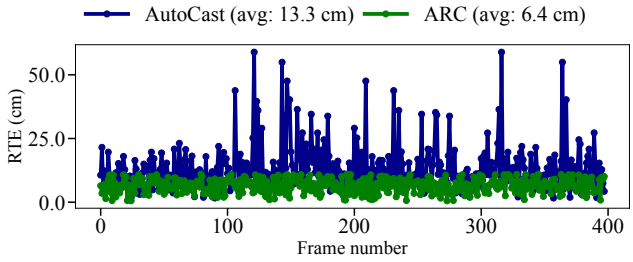


Figure 17: Comparison of ARC and AutoCast accuracy (RTE) on the realworld dataset.

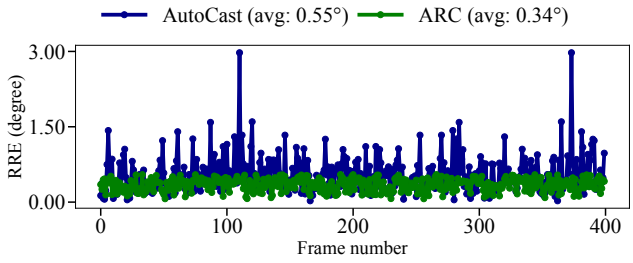


Figure 18: Comparison of ARC and AutoCast accuracy (RRE) on the realworld dataset.

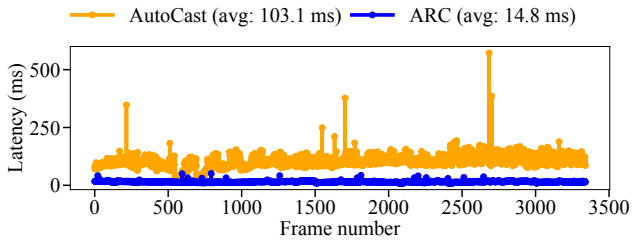


Figure 19: Comparison of ARC and AutoCast end-to-end compute latency on the CARLA dataset.

is an order of magnitude faster than AutoCast. The average latency for ARC (Fig. 19: blue) is 14.8 ms, whereas for AutoCast (Fig. 19: orange), this is 103 ms. With an approximately 15 ms end-to-end compute latency, ARC provides network and downstream modules (perception, planning, and control [32]) approximately 85 ms to process the fused point cloud. On the other hand, since AutoCast generates the fused point cloud in 103 ms, by the time downstream modules consume it, the data is already stale.

ARC ensures low latency even with additional alignment operations for accuracy, for several reasons. Alignment is efficient because it aligns only overlapping regions. Even so, ARC further reduces alignment overhead by reusing computations used in determining overlaps to calculate vehicle blind spots. Lastly, AutoCast uses CPU-intensive computations. In contrast, we designed ARC

Process (on GPU)	Latency (ms)
Localization	2.1
Overlap estimation	0.3
Anchor alignment	2.0
Producer processing	2.0
Consumer processing	8.4
Total	14.8

Table 3: ARC's compute latency breakdown for the CARLA dataset.

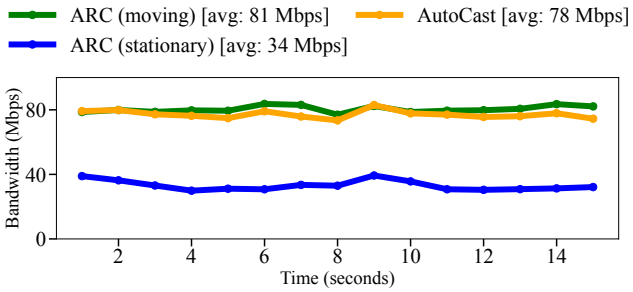


Figure 20: Cumulative bandwidth consumption for 40 vehicles using synthetic traces, comparing ARC and AutoCast.

Approach	3D Points (MB)	Control (MB)	Total (MB)
AutoCast [34]	0.87	*0.1*	*0.97*
ARC	*0.44*	0.58	1.02

Table 4: Per-frame control and 3D (LiDAR data) sharing comparison between ARC and AutoCast for CarLA dataset.

with parallelized operations so it makes use of the GPU. All the components of ARC run on GPU.

The latency breakdown for all the components of ARC is provided in Tbl. 3. Localization latency refers to the time the vehicle takes to use NDT to align with the 3D map. Overlap estimation latency is the time it takes for a vehicle to estimate an overlap between its point cloud and the anchor point cloud. Anchor alignment latency is the time required for the vehicle to align its overlapping region with the anchor. Producer processing latency involves identifying the objects requested by the consumer and sharing them with the consumer. Consumer processing is the most computationally expensive module, encompassing object extraction, blind spots detection, objects cells estimation, motion vector estimation, and producers assignment.

We also evaluated ARC's compute latency on the real-world traces. The average latency per frame was 20 ms. As expected, this is relatively higher than 15 ms for synthetic traces where we have a sparse LiDAR (*i.e.*,, a 32-beam instead of 64 and 128-beam LiDARs). We demonstrate the effect of LiDAR channels on compute latency in §4.6. In our real-world deployment, since our LiDARs are either 64-beam or 128-beam that is why we expect the compute latency to be high. For all four agents ARC takes an average of 20 ms to process each frame. In this experiment, we were unable to evaluate the compute latency of AutoCast on the real-world dataset because it assumes the presence of and requires access to the path-planning module of a vehicle. This raised a safety concern and hence we were unable to do this. Moreover, the open source implementation of AutoCast only provisions support for simulations in CARLA and not on real-world vehicles.

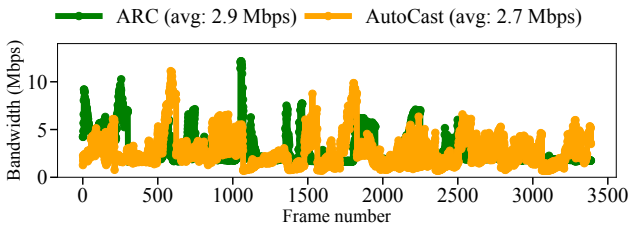


Figure 21: ARC and AutoCast comparison: bandwidth consumption per frame per vehicle for synthetic traces.

Approach	Error		Lat. (ms)	BW (Mbps)
	RTE (cm)	*RRE (°)*		
Indirect	13.1	0.20	*2.10*	78
Direct	*1.5*	*0.05*	∞	∞
ARC (moving)	2.0	0.10	14.8	81
ARC (stationary)	2.5	0.10	15.5	*34*

Table 5: Performance of various alignment strategies of ARC.

4.4 End-to-end Experiments: Network

Next, we measure the networking requirements of ARC. We quantify the amount of bandwidth needed per frame, by all vehicles combined, in both synthetic and real-world traces. On average, ARC needs 81 Mbps (Fig. 20: green) for 40 vehicles. AutoCast, on the other hand, needs 78 Mbps (Fig. 20: orange). The bandwidth requirement for ARC is higher than AutoCast, the reason is that each frame, the anchor broadcasts its point cloud. When the anchor is a stationary roadside sensor, the bandwidth requirement for ARC is much lower and only 34 Mbps (Fig. 20: blue). This is because the anchor shares its reference point cloud with the vehicles only once, not periodically. The reason for AutoCast's higher bandwidth is the redundant data sharing limitation of AutoCast, which becomes worse with a large number of vehicles.

We also provide a communication cost breakdown for both AutoCast and ARC (Tbl. 4). ARC reduces 3D point data sharing from 0.87 MB to 0.44 MB per frame by eliminating redundant transmissions, but incurs higher control overhead (0.58 MB vs. 0.1 MB) due to per-frame anchor point cloud broadcasts. Despite this, the total data produced remains comparable (1.02 MB vs. 0.97 MB).

Additionally, we also calculate the per-vehicle bandwidth consumption for each frame. Per vehicle, the required bandwidth for ARC and AutoCast is 2.9 Mbps, and 2.7 Mbps, respectively (Fig. 21). We exclude frames in which no data is shared between vehicles, as they do not contribute to cooperative perception. For real-world traces, each vehicle uses about 0.5 Mbps of bandwidth due to fewer vehicles sharing data compared to the synthetic dataset. Again, we were unable to evaluate the bandwidth consumption for AutoCast since it required access to the motion planning module of the vehicle.

4.5 Ablation Studies

In this section, we perform ablation studies to quantify ARC's design decisions for point cloud alignment and data sharing.

Alignment Strategies. In Tbl. 5, we evaluate the alignment accuracy, compute latency, and bandwidth requirement for four different strategies that align 3D point clouds in cooperative perception. The four strategies are indirectly aligning point clouds using a 3D map (Indirect), directly aligning point clouds (Direct), and using ARC to align point cloud with moving and stationary anchors. Ideally, to ensure the highest alignment accuracy, we should directly align the two point clouds. However, this approach is not feasible because of computational and network limitations. If a vehicle has to fuse its point cloud with multiple vehicles, it will need to run multiple alignment operations per frame, and this can easily exhaust compute resources. The same is true for required network bandwidth as well, because every vehicle will be broadcasting its entire point cloud.

The lowest latency is for indirect alignment because, once both vehicles are aligned through the 3D map, they are also aligned with one another. However, this approach can cause high alignment errors. ARC carefully navigates this tradeoff to simultaneously achieve low alignment error, low compute latency, and low bandwidth requirement. ARC's point cloud alignment is fast because it only aligns overlapping regions. Second, the bandwidth requirement is relatively lower because it only shares relevant information with other vehicles instead of the entire point cloud. Bandwidth requirement for ARC with a stationary anchor is the least because it does not need to broadcast the anchor point cloud at every frame and only shares relevant parts of the 3D point cloud with other vehicles. Lastly, the alignment error is lower than indirect and is comparable to direct alignment because it aligns against an anchor point cloud, as opposed to a 3D map that accumulates errors over time.

Alignment Technique	Error		Lat. (ms)
	RTE (cm)	*RRE (°)*	
FGR	448	4.0	230
SAC-IA	900	30	280
R-PointHop	733	45	600
Go-ICP	327	10	730
ICP (GPS)	536	16	185
ICP (3D map)	2.3	0.1	25
ARC	*2.0*	*0.1*	*2.7*

Table 6: Comparison of alignment accuracy and latency performance between ARC and prior scan-matching, feature-based, and learning-based alignment techniques.

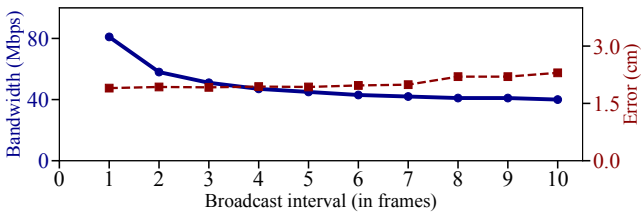


Figure 22: Impact of anchor broadcast frequency on ARC's alignment error and bandwidth requirements.

Overlap-aware Alignment. In this section, we evaluate ARC's ability to accurately and efficiently align vehicle point clouds to the anchor using overlap-aware alignment. For this, we use a CARLA dataset consisting of 150 vehicle and anchor point cloud pairs. We align each vehicle point cloud to the anchor. In Tbl. 6, we compare ARC's alignment accuracy and latency against scan matching, feature matching, and learning-based techniques. Feature-based techniques (FGR [57] and SAC-IA [37]) have high alignment errors because they are unable to match point cloud features captured from

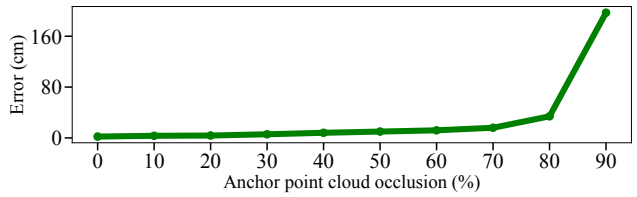


Figure 23: Impact of anchor occlusion on ARC's alignment accuracy.

diverse viewpoints. R-PointHop [20], a recent learning-based alignment technique that uses hierarchical features for correspondences, is also unable to accurately align sparser vehicle point clouds. The alignment error for R-PointHop is on the order of 700 cm.

Scan-matching techniques such as ICP [36] are sensitive to initialization. Poor initial guesses (such as those provided by GPS) result in inaccurate alignments and high latency. On the other hand, variations of ICP less sensitive to initial guesses (GO-ICP [49]) can be compute-intensive (730 ms latency). Although 3D maps provide more precise initial guesses, aligning entire point clouds is compute-intensive (latency on the order of 25 ms).

In contrast, ARC achieves fast alignment with a latency of only 3 ms and high accuracy, with a RTE of just 2 cm. Its speed and accuracy come from aligning only overlapping regions and leveraging a relatively accurate initial guess.

Anchor Broadcast Frequency. We also evaluate how reducing the anchor's broadcast frequency impacts alignment accuracy and bandwidth. Using the CARLA dataset (§4.2), we vary the broadcast interval from broadcast every frame to broadcast after every 10 frames (Fig. 22). The alignment error increases only slightly from 1.9 cm to 2.3 cm (right y-axis), while bandwidth usage drops significantly from 81 Mbps to 40 Mbps (left y-axis). The results show that our approach can adapt to lower network bandwidth by reducing broadcast frequency, with up to 50% bandwidth savings and minimal loss in alignment accuracy. However, the downside is that if a vehicle misses a broadcast, it must wait several frames before receiving the next anchor point cloud, which is not the case when broadcasting occurs every frame.

4.6 Sensitivity Analysis

We perform a sensitivity analysis to assess the impact of anchor point cloud occlusion, traffic density, vehicle speed, and sensor configurations on ARC.

Anchor Point Cloud Occlusion. ARC's alignment accuracy degrades as occlusion of the anchor's point cloud increases. Using our CARLA dataset (§4.2), we simulate occlusion levels from 10% to 90% by obscuring portions of the anchor's point cloud. As shown in Fig. 23, alignment error exceeds 10 cm beyond 50% occlusion and grows to meters past 80%. In our dataset, with 40 vehicles at the intersection, the average occlusion is 17% (corresponding to a 2 cm error), with 95th and 99th percentiles at 24% (2.3 cm) and 27% (2.7 cm), respectively. This demonstrates that even with a higher occlusion level, i.e., up to 27%, ARC is able to maintain an alignment error less than 3 cm.

Traffic Density. To evaluate the effect of traffic density, we simulated three different traffic conditions in CARLA. For low traffic, we had 10–15 vehicles, for medium 15 - 30 vehicles, and for high traffic 30 - 40 vehicles at the intersection. We summarize the results in Tbl. 7. As we increase the traffic density, the required bandwidth for ARC increases from 56 Mbps to 81 Mbps. This is because, with a greater number of vehicles, there are more blind spots and vehicles exchange more data. The compute latency, however, remains relatively constant. This is because, regardless of the number of vehicles in the surroundings, the point cloud alignment and decision of what data to share are relatively time-constant operations. Last, the alignment error increases only marginally with increased vehicles.

Traffic Density	Error		Lat. (ms)	BW (Mbps)
	RTE (cm)	*RRE (°)*		
Low (10 - 15)	1.40	0.05	14.5	56
Medium (15 - 30)	1.75	0.06	14.2	67
High (30 - 40)	2.00	0.10	14.8	81

Table 7: Effect of traffic density on ARC's performance.

Vehicle Speed	Error		Lat. (ms)	BW (Mbps)
	RTE (cm)	*RRE (°)*		
Low (25 km/hr)	1.26	0.03	13.4	56
Medium (50 km/hr)	1.30	0.03	13.6	59
High (75 km/hr)	1.40	0.05	14.5	59

Table 8: Effect of varying vehicle speed on ARC's performance.

LiDAR Channels	Error		Lat. (ms)	BW (Mbps)
	RTE (cm)	*RRE (°)*		
32	1.4	0.05	14.5	56
64	1.3	0.03	17.6	71
128	1.1	0.02	22.5	86

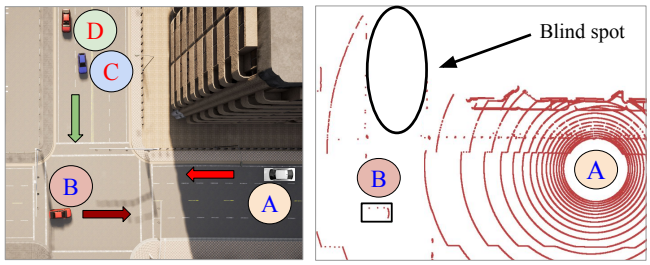
Table 9: Effect of LiDAR channels on ARC's performance.

Vehicle Speed. To quantify the impact of vehicle speed, we collected multiple traces in CARLA with vehicles driving at different speeds. We kept the traffic density constant (low). We summarize the results in Tbl. 8. Low corresponds to an average speed of 25 km/hr, medium to 50 km/hr, and high to 75 km/hr. From our results, we can conclude that ARC is agnostic of the underlying vehicle speeds. It has a minimal impact on alignment accuracy, latency and bandwidth consumption.

LiDAR Channels. In the last set of sensitivity analysis experiments, we evaluated ARC's performance in response to different LiDAR configurations. We deployed 32-beam, 64-beam, and 128-beam LiDARs at the traffic intersection in CARLA. We kept the traffic density constant (low). We summarize the results in Tbl. 9. Increasing the number of channels increases both compute latency and required network bandwidth. This is because ARC's anchor alignment, overlap estimation, and object extraction are sensitive to the number of points in the point cloud. Higher-resolution LiDARs produce denser point clouds, due to which these modules need more cycles for processing. Moreover, more dense point clouds also mean there will be denser regions to share to compensate for blind spots, leading to higher required network bandwidth. Lastly, denser point clouds lead to greater overlap, which improves alignment accuracy.

4.7 Application-Level Benefits

Improved Visibility. We demonstrate the application-level benefits of ARC over AutoCast by evaluating its ability to detect vehicles



(a) A CARLA intersection (b) Vehicle A's point cloud

Figure 24: A CARLA scenario, where vehicles D and C are occluded in vehicle A's point cloud by a building on the right side of vehicle A.

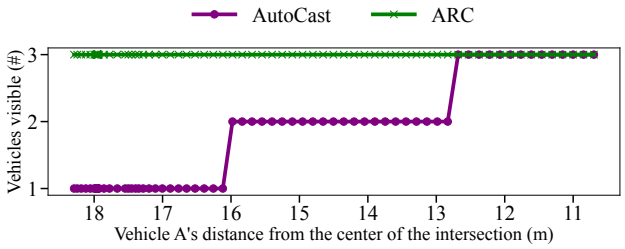
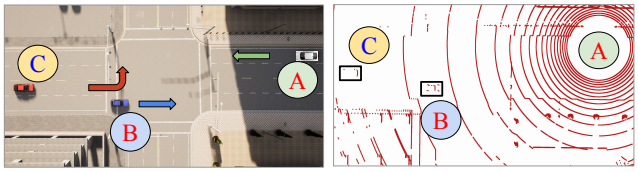


Figure 25: Comparison of AutoCast and ARC in the scenario shown in Fig. 24, showing the number of vehicles visible to vehicle A in each frame.



(a) A CARLA intersection (b) Vehicle A's point cloud

Figure 26: A CARLA scenario, where vehicles A, B, and C are approaching an intersection. In vehicle A's point cloud, both vehicle B and C are visible.

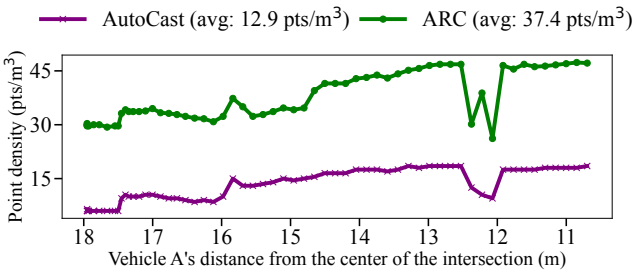


Figure 27: Comparison of AutoCast and ARC in the scenario shown in Fig. 26, showing the mean point density of objects visible to vehicle A in each frame.

in blind spots at a traffic intersection. For this, we define visibility coverage as the number of vehicles detected within the perception range of a vehicle (regardless of line-of-sight limitations).

In Fig. 24a, we show a CARLA scenario in which four vehicles are approaching an intersection. Each vehicle is equipped with a 32-beam LiDAR (120 m range). Fig. 24b shows vehicle A's LiDAR point cloud. In this view, vehicles C and D are missing. This is due to a blind spot created by the building on the right side of vehicle A.

In Fig. 25, we plot the number of vehicles detected by vehicle A as a function of its distance from the center of the intersection. Using AutoCast, vehicle A detects vehicle C only at 16 m and vehicle D at 12 m. In contrast, with ARC, vehicle A maintains visibility of all three vehicles throughout its trajectory. The reason for this is AutoCast uses ray-casting to find blind spots and shares data about objects hidden by objects (vehicles). However, it ignores occlusions caused by static objects like buildings. In contrast, ARC detects blind spots as low-density areas in the point cloud and allows vehicles to share data to fill these gaps.

Improved Point Density. In this section, we demonstrate that ARC improves the point density (number of points per unit volume) in blind spot regions as compared to AutoCast. To do this, we simulated a scenario in CARLA (Fig. 26a) with vehicles A, B, and C approaching the intersection. Each vehicle is equipped with a 32-beam LiDAR (120 m range). We compare the mean point density of vehicles visible to vehicle A (Fig. 26b) for both AutoCast and ARC. Fig. 27 plots this comparison, showing the mean point density of vehicles visible to A as a function of A's distance from the center of the intersection. ARC significantly improves, i.e., 3x, the mean point density compared to AutoCast. AutoCast activates perception data exchange only in the presence of occlusions. However, ARC focuses on regions with low point density (limited visibility), which inherently includes both blind spots and other low-density areas. This leads to more effective and comprehensive data sharing. This, in result, enables downstream perception modules, such as object detection [51] and semantic segmentation [29] to operate on point clouds with higher point density, which they require for better accuracy [13].

5 Related Work

Cooperative Perception. Cooperative perception enables vehicles to extend their sensing range beyond what their own sensors can capture. It mainly follows two approaches: vehicle-to-vehicle (V2V) [27, 33, 34, 54] and vehicle-to-infrastructure (V2I) [11, 18, 39, 50, 56]. V2V is appealing because it does not rely on roadside infrastructure, but it poses significant scalability challenges. Some V2V methods, such as AutoCast [34], and RAO [54], exchange raw point clouds, while others, including OPV2V [47], F-Cooper [9], Core [44], V2V-LLM [12], and InterCoop [46], share processed data instead. These approaches aim to balance latency, accuracy, and scalability, but often improve one at the cost of another. In contrast, ARC is a V2V cooperative perception system that achieves high accuracy and scalability without compromising latency.

Sensor Data Alignment. To fuse external sensor data, a vehicle must align it to its own coordinate system; either directly using algorithms like ICP [36] or indirectly through a shared 3D map or GPS. Direct alignment techniques are accurate [21], but not scalable. Indirect methods scale better but suffer from low accuracy. Most of the existing approaches rely on 3D maps [33, 34] or GPS [9, 47, 54] for indirect alignment, both of which provide limited alignment

accuracy. In contrast, ARC uses a single point cloud for indirect alignment to simultaneously achieve scalability and accuracy.

Sharing Relevant Information. Cellular Vehicle-to-Everything (C-V2X) [52] enables direct communication between vehicles and nearby infrastructure. It supports data rates of up to 100 Mbps [53], which can be quickly saturated when sharing raw point clouds. As a result, early methods such as AVR [33] are not scalable. EMP [55] performs spatial reasoning at the edge, but its high latency limits scalability. Recent methods [34, 54] improve scalability by sharing only relevant information, such as occluded dynamic objects. AutoCast [34] suffers from inaccuracies and redundant data sharing, while RAO [54] improves accuracy and avoids redundancy but introduces higher compute latency. In contrast, ARC performs accurate spatial reasoning. It speeds up processing by reusing computations from its alignment module. This enables scalability without added latency.

6 Discussion and Future Work

Impact of Network Latency. In practice, ARC 's end-to-end latency consists of compute latency (the time to run ARC 's algorithms) and network latency (the time to transmit data over the wireless network). Because directly measuring network latency across tens of communicating vehicles was infeasible for safety and scale reasons, our evaluation focuses on compute latency. On average, ARC 's compute latency is 20 ms per point cloud, leaving approximately 80 ms within the 100 ms reaction window required for safe autonomous operation [26].

For network evaluations, we quantified the wireless bandwidth required to run ARC. On average, ARC needs 40 Mbps of sustained network bandwidth. Existing technologies such as 5G New Radio Vehicle-to-Everything (NR V2X), which support both vehicle-to-vehicle and vehicle-to-infrastructure communication, can theoretically achieve up to 100 Mbps throughput with sub-10 ms one-way latencies [17]. Field deployments typically demonstrate 30–40 Mbps sustained throughput [7]. These figures suggest that ARC 's bandwidth and latency demands are within the operational range of current NR V2X systems. However, we note that in practice, network performance depends on several factors, including the wireless environment, technology stack, and network density. At higher traffic densities, contention and scheduling overheads can increase latency. Though we leave a more thorough evaluation to future work, below, we qualitatively describe the effects of network latency on ARC.

Anchor selection exchanges small control messages; hence network impact is minimal. In contrast, anchor broadcasts involve large point clouds that consume significant bandwidth and can delay the start of alignment if the network is congested. Reducing the frequency of broadcasts (§4.5) helps mitigate this effect but does not eliminate it, especially as fleet size grows. Future network technologies, such as mmWave [42], offering higher data rates and improved scheduling efficiency, are expected to further reduce latency, enabling more timely and scalable operation of ARC.

Adding Multiple Anchors. In §4.2, we show that ARC achieves an accuracy of 2 cm with a single anchor. Deploying multiple anchors could translate to a larger point cloud overlap between anchors and other vehicles. Moreover, it adds an additional layer of robustness to ARC. However, this comes at the cost of increased network load and complexity because all anchors must broadcast their point clouds and perform mutual alignment. We leave a thorough evaluation to future work.

7 Conclusions

In this paper, we have presented ARC, a multi-vehicle cooperative perception framework that achieves high accuracy and low latency while scaling efficiently to a large number of vehicles. We evaluated ARC against state-of-the-art methods on two datasets: one collected in the real world using 64- and 128-beam LiDARs, and another logged from CARLA. The results show that ARC achieves an order-of-magnitude improvement in both latency and accuracy. Specifically, ARC offers an alignment accuracy of less than 7 cm with an end-to-end compute latency of only 20 ms. Moreover, it provides improved point density and enhanced visibility coverage compared to existing approaches. Finally, we demonstrated the robustness of ARC through sensitivity analysis with respect to traffic density, vehicle speed, and LiDAR configuration.

Acknowledgment

We thank the anonymous reviewers and shepherd for their insightful comments. This material is based upon work supported by the U.S. National Science Foundation (award number: CNS-2348461).

References

[1] 2023. *CloudCompare*. https://www.danielgm.net/cc/
[2] 2025. Autoware: Open-source software for self-driving vehicles. https://github.com/autowarefoundation/autoware_ai.
[3] 2025. Baidu Apollo. https://www.apollo.auto/apollo-self-driving.
[4] 5G-PPP. 2015. 5G Automotive Vision. https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-White-Paper-on-Automotive-Vertical-Sectors.pdf. Accessed: Oct. 23, 2025.
[5] Naoki Akai, Luis Yoichi Morales, Eijiro Takeuchi, Yuki Yoshihara, and Yoshiki Ninomiya. 2017. Robust localization using 3D NDT scan matching with experimentally determined uncertainty and road marker matching. In *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1356–1363.
[6] An Open Source Self-Driving Car. 2023. An Open Source Self-Driving Car. www.udacity.com/self-driving-car.
[7] Waqar Anwar, Andreas Traßl, Norman Franchi, and Gerhard Fettweis. 2019. On the Reliability of NR-V2X and IEEE 802.11 bd. In *2019 IEEE 30th annual international symposium on personal, indoor and mobile radio communications (PIMRC)*. IEEE, 1–7.
[8] Automotive Research News. 2025. How Cameras are Paving the Way for Safer Autonomous Driving. https://automotiveresearchnews.com/autonomous-vehicles-av/how-cameras-are-paving-the-way-for-safer-autonomous-driving/ Accessed: 2025-07-02.
[9] Qi Chen, Xu Ma, Sihai Tang, Jingda Guo, Qing Yang, and Song Fu. 2019. F-cooper: Feature based cooperative perception for autonomous vehicle edge computing system using 3D point clouds. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. 88–100.
[10] Xieyuanli Chen, Andres Milioto, Emanuele Palazzolo, Philippe Giguere, Jens Behley, and Cyrill Stachniss. 2019. Suma++: Efficient lidar-based semantic slam. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 4530–4537.
[11] Ziming Chen, Yifeng Shi, and Jinrang Jia. 2023. TransIFF: An Instance-Level Feature Fusion Framework for Vehicle-Infrastructure Cooperative 3D Detection with Transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 18205–18214.
[12] Hsu-kuang Chiu, Ryo Hachiuma, Chien-Yi Wang, Stephen F Smith, Yu-Chiang Frank Wang, and Min-Hung Chen. 2025. V2v-llm: Vehicle-to-vehicle cooperative autonomous driving with multi-modal large language models. *arXiv preprint arXiv:2502.09980* (2025).
[13] Eduardo R Corral-Soto, Alaap Grandhi, Yannis Y He, Mrigank Rochan, and Bingbing Liu. 2023. Improving lidar 3d object detection via range-based point cloud density optimization. *arXiv preprint arXiv:2306.05663* (2023).
[14] Cruise. 2019. Cruise. https://medium.com/cruise/hd-maps-self-driving-cars-b6444720021c.

[15] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An open urban driving simulator. *arXiv preprint arXiv:1711.03938* (2017).

[16] Martin A Fischler and Robert C Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (1981), 381–395.

[17] Mario H Castañeda Garcia, Alejandro Molina-Galan, Mate Boban, Javier Gozalvez, Baldomero Coll-Perales, Taylan Şahin, and Apostolos Kousaridas. 2021. A tutorial on 5G NR V2X communications. *IEEE Communications Surveys & Tutorials* 23, 3 (2021), 1972–2026.

[18] Yuze He, Li Ma, Zhehao Jiang, Yi Tang, and Guoliang Xing. 2021. VI-Eye: Semantic-Based 3D Point Cloud Registration for Infrastructure-Assisted Autonomous Driving. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 573–586.

[19] Yurong Jiang, Hang Qiu, Matthew McCartney, Gaurav Sukhatme, Marco Gruteser, Fan Bai, Donald Grimm, and Ramesh Govindan. 2015. Carloc: Precise positioning of automobiles. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. 253–265.

[20] Pranav Kadam, Min Zhang, Shan Liu, and C-C Jay Kuo. 2022. R-pointhop: A green, accurate, and unsupervised point cloud registration method. *IEEE Transactions on Image Processing* 31 (2022), 2710–2725.

[21] Kaleem Nawaz Khan, Ali Khalid, Yash Turkar, Karthik Dantu, and Fawad Ahmad. 2024. VRF: Vehicle Road-side Point Cloud Fusion. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*. 547–560.

[22] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. 2019. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 12697–12705.

[23] Zhichao Li, Feng Wang, and Naiyan Wang. 2021. Lidar r-cnn: An efficient and universal 3d object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 7546–7555.

[24] Velodyne LiDAR. 2020. Velodyne LiDAR. https://velodynelidar.com/.

[25] Hsien-I Lin, Muhammad Ahsan Fatwaddin Shodiq, An-Kai Jeng, and Chun-Wei Chang. 2025. An Efficient Large-Scale 3D Map Stitching Algorithm Using Automatic Overlapping Area Identification. *IEEE Access* (2025).

[26] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. 2018. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the twenty-third international conference on architectural support for programming languages and operating systems*. 751–766.

[27] Yunsheng Ma, Juanwu Lu, Can Cui, Sicheng Zhao, Xu Cao, Wenqian Ye, and Ziran Wang. 2024. MACP: Efficient model adaptation for cooperative perception. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 3373–3382.

[28] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. 2019. Rangenet++: Fast and accurate lidar semantic segmentation. In *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 4213–4220.

[29] Andres Milioto, Ignacio Vizzo, Jens Behley, and Cyrill Stachniss. 2019. Rangenet++: Fast and accurate lidar semantic segmentation. In *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 4213–4220.

[30] NHTSA. 2023. NHTSA | National Highway Traffic and Safety Authority. https://www.nhtsa.gov/.

[31] Inc. Ouster. 2025. Digital Lidar Sensors for Automation, Drones & Robotics. https://ouster.com/ Accessed: 2025-07-02.

[32] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H Ang. 2017. Perception, planning, control, and coordination for autonomous vehicles. *Machines* 5, 1 (2017), 6.

[33] Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. 2018. Avr: Augmented vehicular reality. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. 81–95.

[34] Hang Qiu, Pohan Huang, Namo Asavisanu, Xiaochen Liu, Konstantinos Psounis, and Ramesh Govindan. 2022. AutoCast: Scalable Infrastructure-less Cooperative Perception for Distributed Collaborative Driving. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '22)*.

[35] Maurice Quach, Jiahao Pang, Dong Tian, Giuseppe Valenzise, and Frédéric Dufaux. 2022. Survey on deep learning-based point cloud compression. *Frontiers in Signal Processing* 2 (2022), 846972.

[36] Szymon Rusinkiewicz and Marc Levoy. 2001. Efficient variants of the ICP algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling*. IEEE, 145–152.

[37] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. 2009. Fast Point Feature Histograms (FPFH) for 3D registration. In *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*. 3212–3217. doi:10.1109/ROBOT.2009.5152473

[38] Radu Bogdan Rusu and Steve Cousins. 2011. 3D is here: Point Cloud Library (PCL). *IEEE International Conference on Robotics and Automation (ICRA)* (2011), 1–4. doi:10.1109/ICRA.2011.5980567

[39] Shuyao Shi, Jiahe Cui, Zhehao Jiang, Zhenyu Yan, Guoliang Xing, Jianwei Niu, and Zhenchao Ouyang. 2022. VIPS: Real-Time Perception Fusion for Infrastructure-Assisted Autonomous Driving. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 133–146.

[40] Christina Suyong Shin, Weiwu Pang, Chuan Li, Fan Bai, Fawad Ahmad, Jeongyeup Paek, and Ramesh Govindan. 2024. RECAP: 3D Traffic Reconstruction. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 1252–1267.

[41] Hesai Technology. 2024. OT128 Automotive-Grade 360° Long-Range Lidar. https://www.hesaitech.com/product/ot128/. https://www.hesaitech.com/product/ot128/ Accessed: 2025-10-26.

[42] Vutha Va, Takayuki Shimizu, Gaurav Bansal, Robert W Heath Jr, et al. 2016. Millimeter wave vehicular communications: A survey. *Foundations and Trends® in Networking* 10, 1 (2016), 1–113.

[43] Vehicle Empire. 2025. Understanding the Role of Radar in Autonomous Vehicles. https://vehicleempire.com/radar-in-autonomous-vehicles/ Accessed: 2025-07-02.

[44] Binglu Wang, Lei Zhang, Zhaozhong Wang, Yongqiang Zhao, and Tianfei Zhou. 2023. Core: Cooperative reconstruction for multi-agent perception. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 8710–8720.

[45] Miaohui Wang, Runnan Huang, Wuyuan Xie, Zhan Ma, and Siwei Ma. 2025. Compression Approaches for LiDAR Point Clouds and Beyond: A Survey. *ACM Transactions on Multimedia Computing, Communications and Applications* (2025).

[46] Wentao Wang, Haoran Xu, and Guang Tan. 2024. InterCoop: Spatio-Temporal Interaction Aware Cooperative Perception for Networked Vehicles. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 14443–14449.

[47] Runsheng Xu, Hao Xiang, Xin Xia, Xu Han, Jinlong Li, and Jiaqi Ma. 2022. Opv2v: An open benchmark dataset and fusion pipeline for perception with vehicle-to-vehicle communication. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2583–2589.

[48] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. 2022. Fast-lio2: Fast direct lidar-inertial odometry. *IEEE Transactions on Robotics* 38, 4 (2022), 2053–2073.

[49] Jiaolong Yang, Hongdong Li, Dylan Campbell, and Yunde Jia. 2015. Go-ICP: A globally optimal solution to 3D ICP point-set registration. *IEEE transactions on pattern analysis and machine intelligence* 38, 11 (2015), 2241–2254.

[50] Sheng Yi, Hao Zhang, Feiyu Jin, Yiyang Hu, Rongzhen Li, and Kai Liu. 2024. V2ICooper: Toward Vehicle-to-Infrastructure Cooperative Perception with Spatiotemporal Asynchronous Fusion. In *International Conference on Wireless Artificial Intelligent Computing Systems and Applications*. Springer, 52–64.

[51] Zhenxun Yuan, Xiao Song, Lei Bai, Zhe Wang, and Wanli Ouyang. 2021. Temporal-channel transformer for 3d lidar-based video object detection for autonomous driving. *IEEE Transactions on Circuits and Systems for Video Technology* 32, 4 (2021), 2068–2078.

[52] Syed Adnan Yusuf, Arshad Khan, and Riad Souissi. 2024. Vehicle-to-everything (V2X) in the autonomous vehicles domain–A technical review of communication, sensor, and AI technologies for road user safety. *Transportation Research Interdisciplinary Perspectives* 23 (2024), 100980.

[53] Eduard Zadobrischi and Ștefan Havriliuc. 2024. Enhancing scalability of C-V2X and DSRC vehicular communication protocols with lora 2.4 GHz in the scenario of urban traffic systems. *Electronics* 13, 14 (2024), 2845.

[54] Qingzhao Zhang, Xumiao Zhang, Ruiyang Zhu, Fan Bai, Mohammad Naserian, and Z Morley Mao. 2023. Robust Real-time Multi-vehicle Collaboration on Asynchronous Sensors. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–15.

[55] Xumiao Zhang, Anlan Zhang, Jiachen Sun, Xiao Zhu, Y Ethan Guo, Feng Qian, and Z Morley Mao. 2021. EMP: edge-assisted multi-vehicle perception. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*. 545–558.

[56] Jiaru Zhong, Haibao Yu, Tianyi Zhu, Jiahui Xu, Wenxian Yang, Zaiqing Nie, and Chao Sun. 2024. Leveraging temporal contexts to enhance vehicle-infrastructure cooperative perception. *arXiv preprint arXiv:2408.10531* (2024).

[57] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. 2016. Fast global registration. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*. Springer, 766–782.

[58] Zoox. 2023. Zoox. https://zoox.com/journal/putting-our-robots-on-the-map/.