AVR: Augmented Vehicular Reality

Hang Qiu
University of Southern California
hangqiu@usc.edu

Fawad Ahmad
University of Southern California
fawadahm@usc.edu

Fan Bai
General Motors Research
fan.bai@gm.com

Marco Gruteser
Rutgers University
gruteser@winlab.rutgers.edu

Ramesh Govindan
University of Southern California
ramesh@usc.edu

ABSTRACT

Autonomous vehicle prototypes today come with line-of-sight depth perception sensors like 3D cameras. These 3D sensors are used for improving vehicular safety in autonomous driving, but have fundamentally limited visibility due to occlusions, sensing range, and extreme weather and lighting conditions. To improve visibility and performance, not just for autonomous vehicles but for other Advanced Driving Assistance Systems (ADAS), we explore a capability called Augmented Vehicular Reality (AVR). AVR broadens the vehicle's visual horizon by enabling it to wirelessly share visual information with other nearby vehicles, but requires the design of novel relative positioning techniques, new perspective transformation methods, approaches to isolate and predict the motion of dynamic objects in order to hide latency, and adaptive transmission strategies to cope with wireless bandwidth variability. We show that AVR is feasible using off-the-shelf wireless technologies, and it can qualitatively change the decisions made by autonomous vehicle path planning algorithms. Our AVR prototype achieves positioning accuracies that are within a few percent of car lengths and lane widths, and is optimized to process frames at 30fps.

CCS CONCEPTS

• **Networks** → **Cyber-physical networks**;

KEYWORDS

Autonomous Cars, Collaborative Sensing, Extended Vision

ACM Reference Format:
Hang Qiu, Fawad Ahmad, Fan Bai, Marco Gruteser, and Ramesh Govindan. 2018. AVR: Augmented Vehicular Reality. In *MobiSys '18: The 16th Annual International Conference on Mobile Systems, Applications, and Services, June 10–15, 2018, Munich, Germany.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3210240.3210319

MobiSys '18, June 10–15, 2018, Munich, Germany

ACM ISBN 978-1-4503-5720-3/18/06…$15.00
https://doi.org/10.1145/3210240.3210319

1 INTRODUCTION

Autonomous cars are becoming a reality, but have to demonstrate reliability in the face of environmental uncertainty. A human driver in the United States can achieve, on average, nearly 100 million miles in between fatalities [40], and users will expect autonomous cars, and other advanced driving assistance systems (ADAS), to significantly outperform humans in terms of reliability. To achieve high reliability, these technologies use advanced sensors for depth perception, such as radar, lidar and stereo cameras. These *3D sensors* permit vehicles to precisely position themselves with respect to the surrounding environment, and to recognize objects and other hazards in the environment.

These 3D sensors periodically capture, at rates of several 3D frames per second, representations of the environment called *point clouds* (§2). A point cloud captures the 3D view, from the perspective of the vehicle, of static (*e.g.,* buildings) and dynamic (*e.g.,* other vehicles, pedestrians) objects in the environment. Because points in a point cloud are associated with three dimensional position information, vehicles can use pre-computed 3D maps of the environment, together with the outputs of these 3D sensors, to precisely position themselves.

However, all of these 3D sensors suffer from two significant limitations. They have limited range which can further be impaired by extreme weather conditions, lighting conditions, sensor failures, *etc.* They primarily[1] have line-of-sight visibility, so cannot perceive occluded objects. These limitations can impact the reliability of autonomous driving or ADAS systems in several situations: a car waiting to turn left at an unprotected left turn has limited visibility due to a truck waiting to turn left in the opposite direction; or, a car is forced to react quickly when the car ahead brakes suddenly due to an obstruction on the road that is not visible to vehicles behind it.

In these situations, vehicles can benefit from *wirelessly sharing visual information* with each other, effectively extending the visual horizon of the vehicle and circumventing line-of-sight limitations. This would augment vehicular visibility into hazards, and enable autonomous vehicles to improve perception under challenging scenarios (Figure 1), or ADAS technologies to guide human users in making proper driving decisions. This capability, which we call Augmented Vehicular Reality (AVR), aims to combine emerging vision technologies

[1]Some radar sensors can shoot signals under a vehicle to detect one more car ahead, but with limited accuracy.

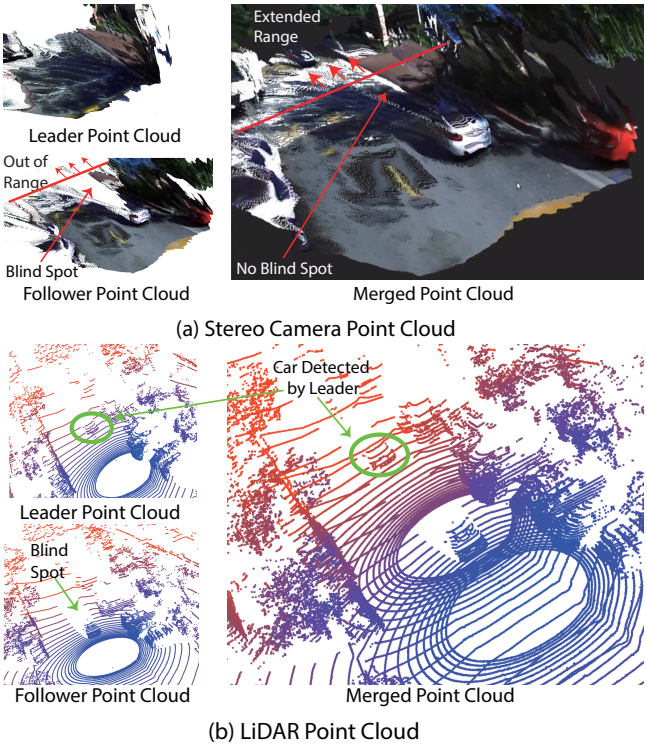

Figure 1—AVR allows a follower vehicle to see objects that it cannot otherwise see because it is obstructed by a leader vehicle. The pictures show a bird-eye view of a leader, a follower and a merged point cloud with different sensing modalities. The top (a) is generated by stereo cameras, while the one on the bottom (b) is obtained from LiDARs.

that are being developed specifically for vehicles [41], together with off-the-shelf communication technologies.

In AVR[2], vehicles exchange raw 3D sensor outputs (point clouds). To achieve this, however, AVR must solve several challenging problems: how to find a common coordinate frame of reference between two cars; how to resolve perspective differences between the communicating cars; and how to minimize the communication bandwidth and latency for transferring 3D views.

Contributions. To address these challenges, this paper makes three contributions (§3). First, AVR devises a novel relative positioning technique where the recipient of a point cloud can position itself with respect to the sender. This positioning technique adapts a recent feature-based SLAM technique developed for stereo vision, which extracts sparse features of the environment. Using a sparse 3D map containing only these features, a receiver can position itself with respect to the sender by learning the sender's position relative to the shared sparse 3D map. Then, a receiver can re-orient received point clouds with respect to its own position using a perspective transformation.

Second, because transferring full 3D point clouds of the environment can exceed the capacity of even future wireless technologies, AVR incorporates techniques to reduce bandwidth requirements. It isolates dynamic objects in the environment using a homography transformation, so it need only transmit point clouds of these objects. It uses off-the-shelf lossless compression on these point clouds, and also extracts motion vectors that enable reconstruction by dead-reckoning at the receiver. It uses an adaptive transmission strategy that sends motion vectors instead of point clouds to cope with channel variability. Finally, it permits *cooperative* sharing: two cars with overlapping perspectives can cooperate to eliminate redundancy in the set of objects shared with a third car.

Third, AVR's algorithms can incur significant processing latency, which can impact the throughput of the system (the rate at which 3D frames are processed) and the freshness of information at the receiver. Our AVR prototype incorporates careful pipelining to increase the frame rate, and leverages motion prediction to hide latency.

Our AVR prototype (§5), when used with a 60GHz radio, can transmit visual information within 150-200 ms. This delay is significantly lower than the maximum permissible delay for AVR to help improve safety while overtaking or stopping due to obstructions. It also adapts gracefully to channel variability. AVR's extended vision, when used as input to path planning algorithms, can avoid dangerous overtake attempts resulting from limited visibility. Using extensive traces of stereo camera data, we show that AVR can achieve 20-30 cm positioning accuracy, which corresponds to 2-10% of lane widths or vehicle lengths. AVR's processing pipeline is optimized to process frames at 30fps. Its major source of error, speed estimation, can be dramatically improved by the use of LiDAR; we are currently working on incorporating LiDAR into AVR.

2 BACKGROUND AND MOTIVATION

3D Sensors in Cars. Today, a few high-end vehicles have a 3D sensing capability that provides depth perception of the car's surroundings. This capability is likely to become pervasive in future vehicles and can be achieved using *3D sensors* such as advanced multi-beam LiDAR, RADAR, long-range ultrasonic and forward-facing or surrounding-view camera sensors. These 3D sensors can be used to detect and track moving objects, and to produce a high-definition (HD) map for localization [55, 59]. This HD map makes the car aware of its surroundings: *i.e.,* where is the curb, what is the height of the traffic light, *etc.*, and is able to provide meter-level mapping and absolute positioning accuracy. Recent research in autonomous driving [19, 53, 24] has leveraged some of these advanced sensors to improve perception.

All of these 3D sensors have one common feature: they generate periodic 3D frames, where each frame represents the instantaneous 3D view of the environment. A 2D image frame is represented by an array of pixels, but a 3D frame is represented by a *point cloud*. Each *point* in the point cloud frame contains the *three-dimensional position* (which enables depth perception) and (optionally) the color of the point. These 3D sensors can differ in the rate at which they generate point clouds, and their field of view. For example, the 64-beam Velodyne LiDAR [64] can collect point clouds

[2]AVR Video Demo: https://youtu.be/9rOtH3hDcw8

Figure 2—Mockup of a heads-up display with AVR's extended vision.

at 10 Hz containing a total of 2.2 million points each second encompassing a 360° view with an effective sensing range of 120 m. In this paper, we use stereo cameras, which can collect point clouds at 60 Hz with over 55 million points per second, but with a limited field of view (110°) and an effective sensing range of 20 m [68].

The Problem. These 3D sensors only provide line-of-sight perception and obstacles can often block a vehicle's sensing range. Moreover, the effective sensing range of these sensors is often limited by weather conditions (*e.g.,* fog, rain, snow, *etc.*) or lighting conditions [51, 16]. These limitations can impact the efficacy of autonomous driving or advanced driver assistance systems (ADAS).

Consider the following examples. A car is following a slow-moving truck on a single-lane highway. The car would like to overtake the truck, but its 3D sensor is obstructed by the truck so it cannot see oncoming cars in the opposite lane. Similarly, two cars, waiting to turn left at an unprotected intersection, can each be "blinded" by the other. As a third example, consider a platoon of two cars, a leader and a follower. The leader's driver, distracted, brakes suddenly upon noticing a pedestrian entering a crosswalk. The follower, unable to see the pedestrian, cannot brake in time to prevent rear-ending the leader. In each of these scenarios, even if the 3D sensors are not obstructed, they may not have sufficient range to view oncoming vehicles or pedestrians.

Augmented Vehicular Reality. In this paper, we explore the feasibility of a simple idea: *extending the visual range of vehicles by wirelessly communicating visual information between vehicles.* We use the term *Augmented Vehicular Reality* (AVR) to denote the capability that embodies this idea. Specifically, in the one-lane highway scenario, if the truck were to communicate visual information from its 3D sensors to the car, using some form of V2V technology, the latter's autonomous driving or ADAS software could determine the safest time and the speed at which to overtake the truck. Similarly, in the left-turn or the platoon scenarios, the transmission of visual information, can help cars turn, or stop, safely. In each of these cases, transmission of visual information can either compensate for the line-of-sight limitations of 3D sensors, or their limited range.

AVR can also extend vehicular vision over larger spatial regions, or over time. It can be used to detect and warn users of hazards such as parked cars on the shoulder, police vehicles, or objects on the road. It can also be used to augment HD maps to include transients in the environment like lane closures or construction activity.

What Visual Information to Transmit. Instead of transmitting point clouds, AVR could choose to transmit object labels and associated metadata (e.g., the label "car" together with the car's position). This is similar to the labels exchanged over DSRC except that, in current DSRC protocols, vehicles only share their own position but not the position of other vehicles observed from sensors. Labels lack contextual information present in point clouds, such as lane markers, traffic regulators, and unexpected objects not defined in the label dictionary. Such contextual information can improve the quality of driving decisions.

However, point cloud transmission is necessary in some cases. The first is when the extended view is displayed using a heads-up display. Figure 2 shows a mock-up of a heads-up display where a car can visualize a vehicle in the opposite lane. The second is when the sender and receiver have different autonomous driving algorithms and processing capabilities. Autonomous driving algorithms use images and point clouds to map, localize, detect and control. Some of those algorithms, such as object detection [46], path planning and steering control [5], are composed of deep and wide neural networks that require substantial computing resources, such as expensive GPUs. In general, the accuracy and speed of autonomous driving algorithms is proportional to the computing resources available to them. Higher-end (*e.g.,* luxury) vehicles are likely to have more computing resources, and would be able to make more accurate control decisions if they were to receive raw point clouds from other cars, than if they were to receive labels generated by cars with fewer computing resources.

3 AVR DESIGN

AVR consists of two logically distinct sets of components (Figure 3). One runs on a *sender* and contains the 3D frame processing algorithms that generate visual descriptions to be sent to one or more *receivers.* Receivers can either feed the received visual descriptions to a heads-up display, or reconstruct an extended view containing the visual descriptions received from the sender with their own 3D sensor outputs[3]. This extended view can be fed into ADAS or autonomous driving software.

AVR poses several challenges. First, for AVR, each vehicle needs to transform the received visual information into its own view. To do this, AVR must *estimate its position and orientation* with respect to the sender of the raw sensor data, and then perform a *perspective transformation* that re-orients the received point cloud.

To address this challenge, both the sender and the receiver share a *sparse 3D map* of the road and the road-side structures. This map is analogous to the 3D maps that autonomous driving systems use to position themselves with respect to the environment, but with one important difference: it is *sparse*, in that it only contains *features* (the green squares in Figure 4) extracted from the denser 3D maps used by these

[3]Each vehicle includes both sender-side and receiver-side processing capabilities. We describe them separately for ease of exposition.

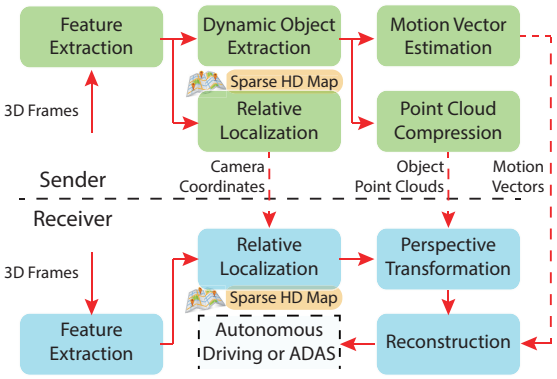

Figure 3—AVR sender and receiver side components.

systems. For AVR, such a sparse map suffices for *relative* positioning. As we describe later, the sparse 3D map can be constructed offline, and potentially crowd-sourced. With this sparse map, the sender processes 3D frames from its camera and extracts features within the 3D frames, then uses some of these features to position its own camera relative to the sparse 3D map. The sender sends its position and a compressed (see below) representation of the 3D frame to the receiver. The receiver uses the sender's camera coordinates, features extracted from its own 3D sensor, and its own copy of the sparse 3D map to estimate its position relative to the sender. After decompressing the received point clouds of dynamic objects, the receiver applies a perspective transformation to these objects to position them within its own coordinate frame of reference.

Second, if AVR were to transmit 3D frames at full frame rates, the bandwidth requirement could overwhelm current and future wireless technologies. Fortunately, successive 3D frames contain significant redundancy: static objects in the environment may, in most cases, not need to be communicated between vehicles, because they may already be available in precomputed HD maps. For this reason, an AVR sender can also, instead of sending full frames, transmit point clouds representing dynamic objects (*e.g.,* cars, pedestrians) within its field of view, and also the motion vector of these dynamic objects. The receiver uses the object's motion vectors to reconstruct the object position, and superimposes the received object's point cloud onto its own 3D frame.

Third, many of the 3D sensor processing algorithms are resource-intensive, and this impacts AVR in two ways. It can limit the rate at which frames are processed (the *throughput*), and lower frame rates can impact the accuracy of algorithms that detect and track objects or that estimate position. It can also increase the latency between when a 3D frame is captured and when the corresponding point cloud is received at another vehicle. AVR selects, where possible, lightweight sensor processing algorithms, and also optimizes the processing pipelines to permit high throughput and low end-to-end latency. Its motion vectors permit receivers to *hide* latency, as described later.

A complete realization of AVR must include mechanisms that prevent or detect sensor tampering and protect the privacy of participants who share sensor data (§7). We have left such mechanisms to future work.

Figure 4—The green dots represent static features that are used to construct the sparse HD map, while features from the moving vehicle are filtered out.

Our initial design of AVR is based on relatively inexpensive (2 orders of magnitude cheaper than high-end LiDARs) off-the-shelf stereo cameras, but we also present some evaluations with a LiDAR device.

3.1 Relative Localization

In AVR, a receiver needs to be able to estimate its position relative to the sender. Absolute positions from GPS would suffice for this, but GPS is known to exhibit high error especially in urban environments, even with positioning enhancements [28]. Specialized ranging hardware can estimate distance and relative orientation between the vehicles, but this would add to the overall cost of the vehicle. The depth perception from the 3D sensors can estimate relative position between the sender and receiver, but, in AVR, the sender and receiver need not necessarily be within line-of-sight.

Key idea. Instead, AVR uses prior work in stereo-vision based simultaneous localization and mapping (SLAM, [38]). This work generates sparse 3D *features* of the environment (called ORB features [38]), where each feature is associated with a precise position relative to a well-defined coordinate frame of reference. AVR uses this capability as follows. Suppose car A drives through a street, and computes the 3D features of the environment using its stereo vision camera. These 3D features contribute to a sparse 3D map of the environment and each feature has a position relative to the position of A's camera at the beginning of A's scan of the street (we call this A's coordinate frame).

Another car, B, if it has this static map, can use this idea in reverse: it can determine 3D features using its stereo camera, then position those features in car A's coordinate frame by matching the features, thereby enabling it to track its own camera's position. A third car, C, which also shares A's map, can position itself also in A's coordinate frame of reference. Thus, B and C can each position themselves in a consistent frame of reference, so that B can correctly determine its position relative to C, and correctly overlay C's shared view over its own. To our knowledge, this is a novel use of stereo-vision based SLAM.

Generating the sparse 3D map. As a car traverses a street, all stable features on the street, from the buildings, the traffic signs, the sidewalks, *etc.*, are recorded, together with their coordinates, as if the camera were doing a 3D scan of the

street. A feature is considered stable only when its absolute world 3D coordinates remain at the same position (within a noise threshold) across a series of consecutive frames. Figure 4 shows the features detected in an example frame. Each green dot represents a stable feature. Features from moving objects, such as the passing car on the left in Figure 4, are not matched or recorded.

Each car can then *crowd-source* its collected map. Map crowd-sourcing is a more scalable way of collecting 3D maps than today's 3D map collection mechanisms which use dedicated fleets of vehicles that require significant capital investment. We have developed a technique to stitch together crowd-sourced sparse 3D maps. Suppose car A traverses one segment of street X. If car B traverses even a small part of that same segment of X, and then traverses a perpendicular street Y, then B can upload its sparse map to a cloud service. As long as B's traversal overlaps even a little with A's sparse map, the cloud service can combine the two sparse 3D maps by translating B's map into A's coordinate frame of reference, generating a composite sparse 3D map. This can be extended to multiple participants, and we have used this technique to generate a static map of our campus. The details of map stitching are described in a companion paper [2].

In practice, we have found that AVR needs only one traversal of a road segment to collect features sufficient for a map since these features represent static objects in the environment. The amount of data needed for each road segment depends on the complexity of the environment. As an example, AVR creates a 97MB sparse HD map for a 0.1 mile stretch of a road on our campus. If necessary, this can be compressed using standard compression techniques [50].

Using the sparse 3D map for relative positioning. The sender processes each 3D frame, and extracts the ORB features, then matches these with the sparse 3D map. AVR uses the ORB-SLAM2 software [38] which matches up to 2000 features and uses these to triangulate the sender's camera position at that frame. This process runs continuously, to compute, at each frame, the sender's camera position relative to the sparse 3D map's coordinate frame. The sender continuously transmits these position estimates to the receiver.

The receiver uses the same technique to estimate the position of its own camera every 3D frame, with respect to the sparse 3D map's coordinate frame. The sender and receiver can be synchronized to within inter-frame granularity using network time synchronization methods [37, 35], and the receiver can then use the sender's position estimates to determine its position relative to the sender.

3.2 Extending Vehicular Vision

With the help of the sparse 3D map's coordinate frame, vehicles are able to precisely localize themselves (more precisely, their cameras), both in 3D position and orientation, with respect to other vehicles easily. However, vehicles can have very different perspectives of the world depending on the location and orientation of their sensors. So, if car A (the sender) wants to share its 3D frame with car B (the receiver), AVR needs to transform the frame's point cloud in the sender's local view to that of the receiver. This *perspective transformation* is performed at the receiver (Figure 3).

Key Idea. The receiver knows the pose (position and orientation) of the sender's camera, as well as its own camera, in the sparse map's coordinate frame. Using these, it can compute a *transformation matrix* T_{cw}, as shown in Equation (1), which contains a 3x3 rotation matrix and a 3-element translation vector. This transformation matrix describes how to transform a position from one camera's coordinate frame to the sparse 3D map's coordinate frame. Specifically, to transform a point V in the camera (c) domain ($V = [x, y, z, 1]^T$) to a point in the world (w) domain ($V' = [x', y', z', 1]^T$), we use $[x, y, z, 1]^T = T_{cw} * [x, y, z, 1]^T$, where

$$T_{cw} = \begin{bmatrix} RotX.x & RotY.x & RotZ.x & Translation.x \\ RotX.y & RotY.y & RotZ.y & Translation.y \\ RotX.z & RotY.z & RotZ.z & Translation.z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

The perspective transformation. Now, suppose that the sender's transformation matrix is T_{aw} and the receiver's is T_{bw}, then, the receiver can compute the perspective transformation of a point V_a in the sender's view to a point V_b in the receiver's view as follows: $V_b = T_{bw}^{-1} * T_{aw} * V_a$.

3.3 Detecting and Isolating Dynamic Objects

Transferring full 3D frames between a sender and a receiver stresses the capabilities of today's wireless technologies: VGA 3D frames at 10fps require 400 Mbps, while 1080p frames need 4 Gbps. So, AVR will require several techniques to reduce the raw sensor data to be transmitted between vehicles.

Key idea. AVR extracts and transmits point clouds, at the sender, only for *dynamic* objects (moving vehicles, pedestrians *etc.*). To identify a moving object, AVR can analyze the motion of each point in successive frames. Unfortunately, it is a non-trivial task to match two points among consecutive frames. Prior point cloud matching techniques [25, 33] involve heavy computation unsuitable for real-time applications. AVR exploits the fact that its cameras capture video, and matches pixels between successive frames instead of points. It can do this because, for stereo vision cameras, a pixel and its corresponding point share the same 2D coordinate.

The homography transformation. AVR uses the ORB features extracted in every frame to find the homography transformation matrix between two successive frames. This matrix determines the position of the current frame in the last frame. Because vehicles usually move forward, the last frame often captures more of the scene than the current frame (Figure 5). AVR matches ORB features between successive frames to compute the homography matrix H, a 2D matrix that can transform one pixel ($P = [x, y]^T$) to the same pixel ($P' = [x'y']^T$) in the previous frame with a different location ($P' = H * P$). AVR's sender computes this transformation continuously for every successive pair of frames.

Detecting dynamic objects. Once pixels can be matched between frames, AVR can match the corresponding points because of the correspondence between the two. It computes

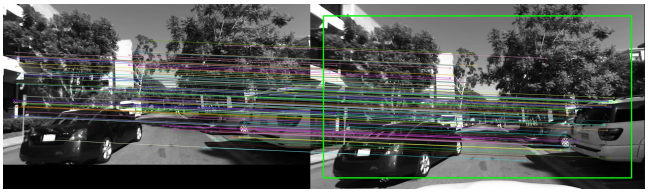

Figure 5—An illustration of a homography transformation. The frame on the right is taken immediately before the frame on the left and the green box represents the part of the left frame visible on the right. The line match features in the two frames.

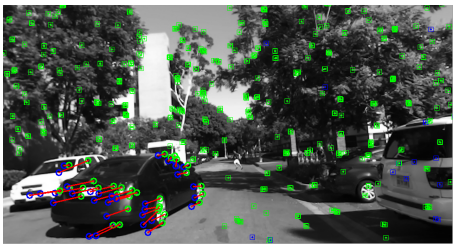

Figure 6—Estimating the motion of a vehicle. The lines show the motion of features (represented by circles) across two successive frames.

the Euclidean distance between matching points from consecutive frames. Before calculating this point displacement, AVR transforms the matching points into a common coordinate frame (*e.g.,* the current frame, using perspective transformation, discussed above). It then applies a threshold to determine the points belonging to dynamic objects. This threshold is necessary because sensor noise can result in small displacements even for points belonging to static objects: with a vehicle cruising at 20mph, and a stereo camera recording at 30fps, the average displacement of the stationary points is $< 5cm$ per frame.

Compression. To further reduce bandwidth requirements, AVR uses point cloud compression techniques [50].

3.4 Extracting Object Motion and Reconstruction

Even after extracting dynamic objects and compressing them, the bandwidth requirements for AVR can be significant.

Key idea. To further reduce bandwidth requirements, AVR estimates, at the sender-side, the motion vector of dynamic objects detected in the previous step. Then, it transmits the compressed point cloud for a dynamic object from a frame, and only the motion vector for, *e.g., k* frames following that frame (AVR adapts k dynamically, as discussed below). Since the motion vector can be compactly represented, this provides an additional bandwidth savings of nearly a factor of k.

Estimating the motion vector. AVR leverages computations performed in previous step to estimate the motion vector. It determines which ORB features belong to dynamic objects using the homography matrix (§3.3), then computes the average motion vector of those feature points as an estimate of the total motion (Figure 6). When there are multiple moving objects, AVR uses optical flow segmentation [65] to separate the objects and compute separate motion vectors. Finally, based on the frame timestamps, AVR converts motion vector into speed and direction estimates (*velocity vectors*) and transmits these to the receiver.

To get smoother velocity vector estimates, AVR tracks the same set of features over m frames, and computes the motion vector for the current frame using features from m frames in the past, rather than the immediately previous frame. Features may not persist after a few frames, and new features can appear in a frame, so AVR must use a consistent set of features while permitting feature entry and removal; we omit the details for brevity.

Reconstruction. On the receiver, AVR uses the point cloud and subsequent velocity vectors to "dead-reckon" the position of the sender and thereby reconstruct the received object at the correct position. The AVR receiver applies perspective transformation on the received object point cloud, then applies each received velocity vector to each point in the point cloud to obtain an estimated position of the point cloud. AVR can then superimpose this point cloud into its own 3D frame and feed the resulting composite frame into an ADAS or autonomous driving algorithm.

Latency hiding. In practice, as we discuss below, AVR's processing pipeline can incur some delay. Thus, an object in a frame captured at the sender at time T might arrive at the receiver at $T + \delta$, after processing and transmission delays. AVR can use this dead-reckoning to *hide this latency*: dead-reckoning can calculate the expected position of the sender at $T + \delta$ using the last known velocity vector for the object. Thus, object motion estimation serves two purposes: it can reduce bandwidth usage and hide the processing latency.

3.5 Adaptive Frame Transmission

In AVR, we consider situations where a leading car may transmit point clouds to a follower. The sender has a choice of transmitting full frames, only dynamic objects, or motion vectors. Dynamics in the environment can cause fluctuations in channel capacity, or in the number of dynamic objects in the scene. To adapt to these variations, AVR uses an adaptive strategy to decide which of these to transmit. If a previous transmission of a full frame completes before the next frame has been generated, AVR transmits the next frame in its entirety. If not, AVR transmits only the dynamic objects in the next frame. When channel capacity is insufficient event to transmit dynamic objects, AVR reverts to transmitting only velocity vectors[4]. This technique, inspired by adaptive bitrate techniques for video, naturally adapts to capacity and scene changes, and we demonstrate this experimentally (§5).

3.6 Cooperative AVR

A final technique to reduce bandwidth requirements, and one that is easy to realize in our AVR design, is *cooperative AVR*. Consider two vehicles A and B which are traveling in the same direction on adjacent lanes, followed by another car C. Cars A and B have overlapping views, and can eliminate redundancy in transmitting objects to each other and to C:

[4]Velocity vectors need to be transmitted with full frames and dynamic objects since they are needed for latency hiding.

B can avoid transmitting objects in its view that are also in A's view (which B can determine when it receives the camera coordinates and pose from A).

To do this, B would first need to perform a *reverse* perspective transformation on each dynamic object it detects, to determine if this object falls in the field of view (FOV) of A. Specifically, assume the angle of the horizontal FOV is α, and the angle of the vertical FOV β, and assume x axis is horizontal, y is vertical, z is the depth. Then, the 3D coordinates (x, y, z) of the *transformed* object should satisfy both $-z * \tan\frac{\alpha}{2} \leq x \leq z * \tan\frac{\alpha}{2}$ and $-z * \tan\frac{\beta}{2} \leq y \leq z * \tan\frac{\beta}{2}$ to be within the FOV of A. In other words, B only transmits objects that are outside the rectangular pyramid defined by A's FOV. For those objects that are in the FOV of A, B double-checks the coordinates of the dynamic objects sent by A to verify the redundancy before removing them from its own transmission queue. For the cooperation to work, B must determine that C can also receive the objects sent by A, which can be achieved by exchanging neighbor sets between neighboring nodes.

4 AVR OPTIMIZATIONS

Implementation. Our AVR implementation builds upon a publicly available implementation of ORB-SLAM2 [43] which is designed for stereo vision and is currently the highest ranked open source visual odometry algorithm on the popular vehicle vision KITTI benchmark [15]. We use ORB-SLAM2 for relative localization, the PCL library [48] for *lossless* point cloud compression, OpenCV for homography transformations and feature tracking, but have built other modules from scratch, including perspective transformation, dynamic object isolation, motion vector estimation, real-time bandwidth adaptive streaming and 3D frame reconstruction. Our AVR prototype is 11,786 lines of C++ code.

Our current implementation uses the ZED stereo vision cameras [67] which have a built-in GPU pipeline designed to synthesize 3D frames.

Optimizations. Many of our modules perform complex processing on large 3D frames, and can incur significant processing latency. Our initial, unoptimized prototype could only process 1 frame every 3 seconds on a relatively powerful desktop machine, so we implemented several optimizations to increase the frame rate and reduce the end-to-end delay.

The primary bottleneck in AVR is the underlying SLAM algorithm. From each 3D frame, SLAM computes the 3D position of each feature, and uses a non-linear pose optimization algorithm for each feature. The complexity of this step is a linear function of the number of features that ORB-SLAM2 computes. The software, by default, detects up to 2000 features, but we have been able to re-configure ORB-SLAM to detect up to 500 features without noticeable loss of accuracy while reducing processing latency by nearly 4x.

Several AVR algorithms, including optical flow, homography transformation and SLAM, use features, and feature extraction can increase processing latency. AVR computes ORB features once, and uses these for localization, dynamic object isolation, and motion vector estimation (Figure 7).

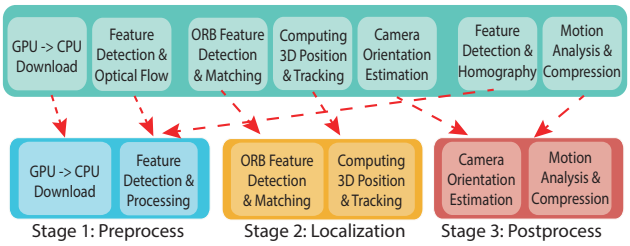

Figure 7—Pipelining the different components carefully enables components to be executed in parallel, resulting in higher frame rates.

Loading a 3D frame from GPU memory to CPU memory incurs latency. To interleave I/O and computation, and to leverage parallelism where possible, we employ pipelining. While pipelining does not affect the end-to-end latency experienced by a frame, it can significantly improve the throughput of AVR (in terms of frame rate). We carefully designed AVR's 3-stage pipeline (Figure 7) so that each stage has comparable latency. The pre-processing stage transfers the 3D frame from the GPU and performs ORB feature extraction and feature pose optimization. When the pre-processing stage processes frame $k+1$, the localization stage processes frame k, computing 3D positions and matching features. In parallel, the post-processing stage processes the $k-1$-th frame to first finish camera pose estimation, then extract dynamic objects, estimate their motion vectors, and compress the point cloud.

5 AVR EVALUATION

Methodology. Our evaluation uses end-to-end experiments as well as traces of stereo camera data collected on our campus. Our end-to-end experiments involve using two Alienware laptops each with an Intel 7th generation quad-core i7 CPU clocked at 4.4 GHz, 16 GB of DDR4 RAM and an nVidia 1080p GPU equipped with 2560 CUDA cores. We place one laptop in a leader vehicle, and the other laptop in a follower vehicle. Each laptop is connected to a ZED [67] camera and to a TP-Link Talon AD7200 802.11ad wireless router [56] on each vehicle. The routers communicate using the wireless distribution system (WDS) mode in the 60 GHz band. While 802.11ad has several drawbacks with respect to vehicle-to-vehicle communication, including the fact that it requires line-of-sight communication, we use it to demonstrate proof-of-concept. We are not aware of any off-the-shelf high bandwidth wireless radios designed specifically for vehicle-to-vehicle communication. One possibility is to incorporate AVR into future versions of DSRC technology. The US FCC has reserved 75 MHz for DSRC over seven 10 MHz channels. Some of these channels have been reserved (*e.g.,* for Basic Safety Messages that announce a vehicle's position and trajectory), but the usage of several service channels is still under discussion.

In our setup, ZED can create a real-time point cloud at a frame rate of up to 60fps with a resolution of 720p (1280 x 720), and up to 100 fps with VGA (640 x 480). Our traces record at 30fps, which is the rate AVR's pipeline is able to

achieve. We evaluate AVR on traces of stereo camera data, while driving these vehicles around campus. Our traces span 123,000 stereo frames, or about 0.5 TB of point cloud data. In addition, we have generated a sparse 3D map of our campus by driving a vehicle with a ZED camera around the campus. We use this sparse 3D map for relative localization in our trace driven evaluations.

In one of our evaluations, we use an HDL-32E LiDAR sensor with a range of 80-100m to demonstrate improvements to AVR that might be possible with LiDAR. We have left a complete integration of LiDAR (which can, in general, perform better than stereo cameras under poor visibility *e.g.*, at night) into the AVR pipeline, and the evaluation of AVR performance under those conditions for future work (§6).

Metrics. In our end-to-end experiments, we investigate end-to-end delay, bandwidth requirements, and the performance of adaptive transmission. To measure end-to-end delay, we synchronize the clocks on the laptops before the experiment. For our trace-driven studies, our primary metric is *accuracy* of the position of the reconstructed object in the receiver's view. This accuracy is a function of the bandwidth used to transmit the dynamic objects. Contributing to this accuracy is the error induced by relative localization, and by using motion vectors to predict position; we also quantify these. We also evaluate two other metrics, *throughput* and *latency* of the AVR pipeline. Lower throughput can impact accuracy, as can high latency.

5.1 The Benefits of AVR for ADAS and Autonomous Driving

Autonomous driving and ADAS systems use several building blocks including localization, object detection, drivable space detection, path planning, and so on. Many of these could benefit from AVR. We have implemented two of these algorithms, *road surface detection* and *path planning*, to demonstrate the benefits of AVR for the *overtaking* scenario, in which a follower would like to overtake a leader car, but its view is obstructed by the leader.

Our road surface detection algorithm uses the point cloud library (PCL), and applies several optimizations to extract the points on the road plane from our stereo camera data. Next, we convert the point cloud into an occupancy grid, where each grid (0.5m × 0.5m) is either drivable, occupied, or undefined. Finally, we use the A* search algorithm to find a viable path that can permit a box corresponding to the vehicle's dimensions to pass through, without hitting any occupied or undefined grid.

We then collected a trace with two vehicles, a leader and a follower, driving along a road and a third oncoming vehicle in the opposite lane. The follower would like to overtake the leader. We then fed this trace into our path planner. Figure 8 shows the result of both road detection and path planning with and without AVR, where the point cloud of the detected road is marked in blue, and the planned path is marked in connected green crosses. In the first case without AVR, the follower could only see the leader's trunk and detect

Figure 8—Path Planning and Road Detection in Action: with extended vision, the oncoming vehicle and more road surface area is detected, and the path planner decides to not attempt overtaking.

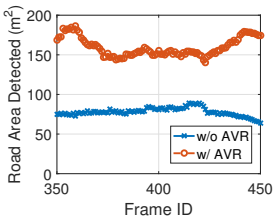

Figure 9—AVR helps the follower detect twice the road than it might have otherwise.

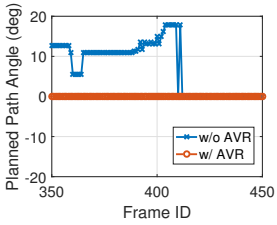

Figure 10—With AVR, the follower would avoid the overtake maneuver.

road surface upto the sensing range limit with occlusion. The planner could find a clear path to overtake the leader switching to the left lane. With AVR, the follower can not only detect more of the road surface, but also the oncoming vehicle, so the path planning algorithm does not attempt the overtaking maneuver, instead choosing to follow the leader. This example demonstrates that extended vision can help autonomous driving algorithms avoid potential hazards.

Figure 9 shows that with AVR, the follower is able to detect twice as much *visible* road surface as without AVR, thanks to the extended vision. Figure 10 shows the planned path angle over the entire trace. Without AVR, the planner switches to the left lane to overtake the leader until it can sense the oncoming vehicle, whereas with AVR, the planner decides to not switch lanes but to follow the leader.

5.2 AVR End-to-End Performance

In this experiment, we quantify the performance of our adaptive transmission strategy by running our AVR prototype live on two moving vehicles (driven within our campus) connected by a 60 GHz 802.11ad link. Specifically, we demonstrate two adaptive transmission strategies: one in which *full* frames are transmitted, else motion vectors, and another in which *dynamic* objects are transmitted, else motion vectors.

We are interested in three different aspects of performance: the average bandwidth achieved over these radios, the *end-to-end delay* between when a frame was generated at the transmitter and when they were received, the *transmission delay* between received frame and the receiver's own frame at reception, the fraction of frames for which motion vectors are transmitted, and the average run-length of motion vectors.

Figure 11 shows the mean of these quantities averaged over 3 mins for the two scenarios: full and dynamic. In the full frame case, AVR transmits on average 1.17 velocity vectors per frame to adapt to the bandwidth, achieving an effective throughput of 367.02 Mbps. For comparison the maximum throughput we have been able to achieve between these two

Mean	Throughput (Mbps)	End-to-End Delay (ms)	Transmission Delay (ms)	Velocity Vector Streak	Velocity Vector per Frame
Full	367.02	221.78	59.25	2.80	1.17
Dynamic	34.95	131.56	39.05	1	0.003

Figure 11—End-to-End results demonstrating AVR over a 60Ghz wireless link between two cars in motion.

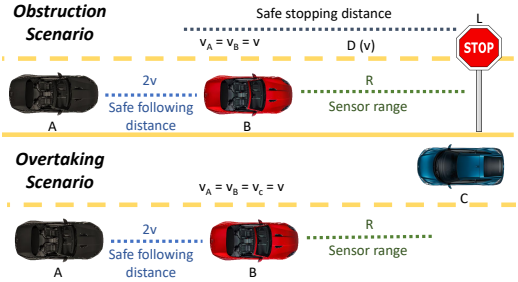

Figure 12—Two AVR use cases with different delay requirements: a stationary obstruction, and the overtaking scenario.

radios is 700-900 Mbps in the lab. The frequency of the velocity vectors indicates that the radios do not have enough capacity to sustain transmission of full point clouds every frame. Transmitting only the dynamic point clouds significantly reduces the bandwidth requirement by one order of magnitude, improving both end-to-end delay and transmission delay by 41% and 34% respectively. One interesting quantity is the velocity vector streak (the average number of consecutive velocity vectors), to which the accuracy is very sensitive. The longer velocity streak is, the longer AVR performs dead reckoning with only velocity estimates. In the full frame scenario, AVR's bandwidth adaptive scheme transmits an average of 2.8 velocity vectors continuously, whereas in the dynamic case, a velocity vector is almost always followed by a dynamic object, and velocity vectors are very rare (about 3 in 1000 frames).

How much delay is enough. Our current end-to-end delays are about 220ms for transmitting full frames, and about 130ms when dynamic objects are transmitted. To understand if these delays are sufficient, we consider simple models of two AVR use cases (Figure 12): the *obstruction* use case, and the *overtaking* use case. In both of these cases, we assume that a human is making driving decisions.

For the obstruction example (Figure 12, top), consider car A following car B, while a stationary object L (e.g., a parked car) is in the same lane ahead of B. Assume for simplicity that all cars are traveling at the same speed v. Assume also that car A follows the two second rule [57] for safe following distances between cars. Car B sees L when the latter is within sensing range of B. For this information to be useful to A, it must reach A ahead of the *safe stopping distance* for a car. Most drivers can decelerate at about $6m/s^2$, and have a reaction time of 1s [63]. Then, the maximum permissible delay for AVR to be useful in this scenario is $0.84 + \frac{R}{v}$.

In the overtaking case (Figure 12, bottom), consider a car A following car B, while car C is an oncoming car in the opposite lane. Assume for simplicity that all cars are traveling at the same speed v. Assume also that car A follows the two second rule [57] for safe following distances between cars. Now car B sees car C when the latter is within sensor (stereo camera or LiDAR) range of B. For this information to be useful to A, it must reach A before A itself can see C. Then, the maximum permissible delay for AVR to be useful in this setting is $\frac{R}{2v}$.

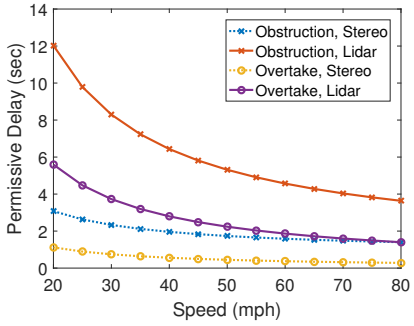

Figure 13—The maximum permissible end-to-end delay for the two use cases, for two different sensor types, as a function of speed.

Figure 13 plots the delay as a function of vehicle speed for the two scenarios using these equations, and assuming nominal ranges for stereo cameras (20m) and LiDARs (100m). From Figure 11, the end-to-end delay for AVR is about 200ms. As the figure shows, AVR can use either sensor for the obstruction scenario: even at high speeds, a stereo camera is sufficient to be useful. Interestingly, however, the overtaking scenario is much more stringent: at higher speeds, AVR latencies start to approach the maximum permissible delay, suggesting that AVR will need LiDAR for this scenario.

5.3 Accuracy Results

In this section, we evaluate the accuracy of the extended vision, namely the position of the reconstructed point cloud. We first evaluate the end-to-end reconstruction accuracy of both static objects and dynamic objects with different relative speed. Next, we conduct a detailed analysis of the tradeoff between bandwidth and reconstruction accuracy and the sensitivity to processing and transmission delay. Finally, we investigate whether using a LiDAR device could improve speed estimation accuracy.

Reconstruction. The primary measure of accuracy is the *positional* error of AVR reconstruction. That is, if p is the known position of the object in the sender's view, and p' is the derived position at the receiver, the position error is given by $|p - p'|$. More precisely, we estimate the position error for a given ORB feature corresponding to the object, and average these position errors across multiple ORB features. The positioning accuracy of ORB-SLAM2 has been benchmarked on the KITTI visual odometry dataset [30], so we do not evaluate it in this paper.

Setup. For this experiment (Figure 14) we mounted cameras A and B on two moving vehicles, one in front of the other, with a third vehicle C moving in the opposite direction. L is stationary object on the side of the road. We measure the relative positional error of L and C as viewed from A without extended vision, and their reconstructed view with extended vision at A, using visual information received from B.

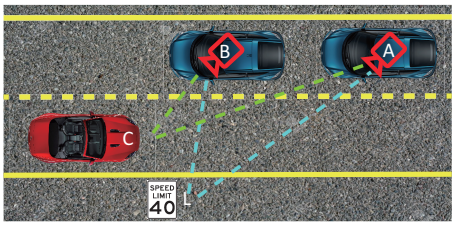

Figure 14—Experiment setup for evaluating end-to-end reconstruction accuracy.

		Moving Objects			Static Objects		
	Relative Speed (mph)	10	20	30	10	20	30
Perfect Calibration	Reconstruction Error: Median (m)	0.068	0.07	0.298	0.016	0.027	0.036
	Reconstruction Error: 90%ile (m)	0.185	0.193	0.415	0.022	0.045	0.07
Realistic Calibration	Reconstruction Error: Median (m)	0.253	0.316	0.592	0.212	0.251	0.298
	Reconstruction Error: 90%ile (m)	0.392	0.496	0.708	0.441	0.481	0.549

Figure 15—End-to-End reconstruction error, and its main components: camera pose estimation, motion compensation, and camera calibration.

For this experiment, we collect stereo camera traces from the two cameras A and B while varying their speeds with respect to the third moving vehicle C. The leading vehicle B transmits the full point cloud it observes along with the velocity estimate of C at every frame. At the following vehicle A, AVR reconstructs the received point cloud from B using perspective transformation. To evaluate the positional accuracy of L, A compares the estimated position of L in its own point cloud with the estimated position in the reconstructed point cloud it received from B. Similarly, A predicts the positional accuracy of C by compensating the delay with respect to its current frame for the reconstructed point cloud from B, and compares it with the position of C in its current point cloud. For this, A uses the velocity vectors received from B while receiving the last full object point cloud.

We collected traces of different relative speeds between objects and cameras of 10, 20, 30 mph, while keeping A, B, C at the same speed. When evaluating the accuracy, we assume an average transmission delay of 60 ms (Figure 11), and compare the reconstructed dynamic point cloud versus the receiver's own point cloud frame by frame. For static objects, since there are no motion estimates, we randomly sample frames from the two footages and compute the reconstruction error. Intuitively, the only sources of error during the reconstruction of static objects, are the camera pose estimation and camera calibration, while the major source of error for the dynamic object is motion compensation. To understand the impact of camera calibration, we also run two AVR instances using one footage from the same camera, not only for static objects, but also dynamics, which emulates perfect calibration.

Results. Figure 15 shows the reconstruction error for both the static object L and dynamic object C for different speeds of the vehicles A and B. At low speeds, with perfect calibration, static objects can be localized to within *1.6 cm* at the median. At 30 mph, the errors are still low, about *7 cm* in the 90th percentile. These errors are entirely due to camera pose estimation, and accuracy decreases at higher speeds for static objects because of the noise in camera pose due to car motion.

The more challenging case for reconstruction accuracy is when the object is also moving (Figure 15). In this case, with perfect calibration, 90th percentile reconstruction errors are within 20 cm for speeds up to 20 mph, going up to 40 cm for the 30 mph case. The difference in accuracy between static and dynamic objects is entirely attributable to errors in speed estimation: these errors add about 15 cm to the reconstruction error. Below, we explore if using LiDAR can enable more accurate speed estimation.

To put these numbers in perspective, the lane width in the US highway system is about 3.7 m, while the average car length is 4.3 m [62]. Thus, the reconstruction error for a static object is about 1% of these quantities, and for a dynamic object is 5-10%. Because many autonomous driving algorithms make decisions at the scales of cars or lane widths, we believe these reconstruction accuracies will be acceptable for future ADAS and autonomous driving systems.

The errors described are *intrinsic* to our system. There is an *extrinsic* source of error, camera calibration. Realistic calibration add 5-30 cm error to reconstruction likely due to the relatively cheap stereo camera we use. Production vehicles are likely to have more finely calibrated cameras.

The Bandwidth / Accuracy Tradeoff. Reconstruction error can depend on bandwidth, so in this set of experiments, we throttle the wireless link bandwidth to evaluate the tradeoff between bandwidth and accuracy while using adaptive transmissions. AVR sends a full point cloud, the *key cloud*, followed by a series of velocity vectors when the streaming rate is larger than the link capacity. The data size of the velocity vectors is negligible compared to the volume of the point cloud. Thus, throttling the bandwidth directly triggers AVR to send fewer full point clouds and more motion estimates, which impacts the reconstruction accuracy. In this experiment, we throttle the bandwidth with different thresholds, and evaluate the reconstruction accuracy degradation, as well as the ratio between the number of velocity estimates sent versus a full point cloud.

Results. Figure 16 shows that average reconstruction error degradation increases significantly up to nearly half a meter once bandwidth is throttled below 20 Mbps. In this regime, AVR transmits, on average, 6-8 motion vectors per full frame. Speed estimation using motion vectors obtained from stereo cameras can be error prone, and can impact reconstruction accuracy. We show below that, when we use LiDAR, speed estimation is significantly more accurate.

The impact of delays. Between when a frame is captured at a sender, and when an object is reconstructed at the receiver, there can be two sources of delay: the AVR pipeline's *processing latency* and the vehicle-to-vehicle *transmission latency*. AVR can *hide* these latencies by dead-reckoning, using motion vectors, the current position of the dynamic object. In this section, we quantify the reconstruction accuracy degradation due to the impact of dead reckoning of various delays.

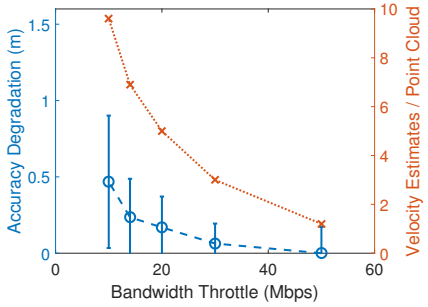

Figure 16—At lower bandwidths, reconstruction accuracy can increase by half a meter because of errors in speed estimation.

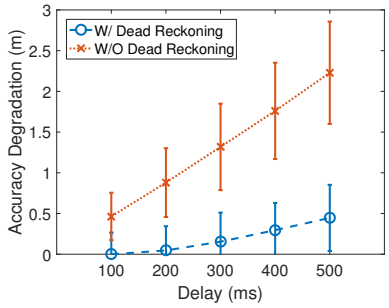

Figure 17—At higher delays, dead reckoning can help contain reconstruction errors significantly, motivating our use of velocity vectors.

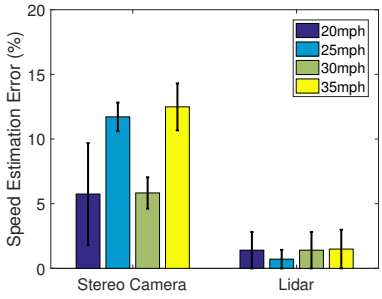

Figure 18—LiDAR can improve speed estimates by an order of magnitude.

Results. Reconstruction error increases by 30-40 cm when the delay increases from 100 ms to 500 ms (Figure 17). The figure also quantifies the accuracy degradation *without* latency hiding: if AVR simply used the last key cloud as an estimate for the current position of the vehicle, its error can be nearly 2 m at high latencies. Such a high error would significantly reduce the usefulness of AVR for safety-based driving assistance, and motivates the use of dead-reckoning using speed estimates.

Better Speed Estimation. The main source of reconstruction errors comes from motion estimation among consecutive frames. To understand whether speed estimation would be significantly better with LiDAR, we obtained a Velodyne 32-beam LiDAR[5]. We mounted both a stereo camera and the LiDAR on the side of the road, and drove a car at 4 different fixed[6] speeds (20, 25, 30, and 35 mph). For the stereo camera, we use AVR to estimate the motion vector, then derive speed estimates from the motion vector. For LiDAR, we measure the front and rear positions of the car in each frame and calculate the speed estimates. All speed estimates are averaged over a sliding time window.

As Figure 18 shows, LiDAR is several times more accurate than a stereo camera at estimating speed. While stereo cameras can estimate speeds to within 10-15% error, LiDAR speed estimates are in the 1-2% range across all the speeds, almost an order of magnitude lower.

Cooperative AVR. We collected traces from two cars, driven side by side in adjacent lanes, on city main streets with normal traffic. Over 12,000 frames, on average 61.63% (standard deviation 24.9%) of the objects appeared in both cars' views, reducing the total bandwidth of transmitting the combined view of the two vehicles by almost another 30%.

5.4 Throughput and Latency

Throughput and latency of the processing pipeline also significantly impact the performance of AVR. These are a function of (a) our algorithms, (b) our performance optimizations, and (c) the hardware platforms on which we have evaluated AVR.

[5]We have been unable to incorporate LiDAR into AVR because we have not found a suitable off-the-shelf SLAM implementation. This is left to future work.

[6]Cruise control does not work reliably at lower speeds, hence this choice of speed.

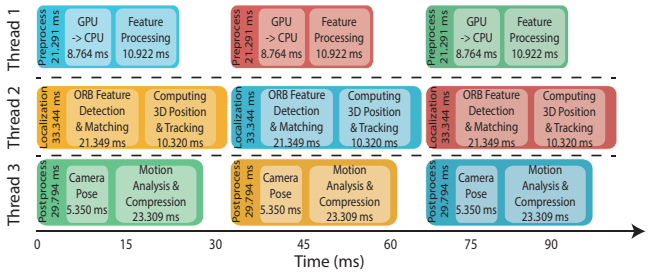

Figure 19—Pipelining enables AVR to process 30fps with an end-to-end delay of under 100ms.

The compute capabilities of on-board platforms have evolved significantly in the past couple of years: the current NVidia Drive PX 2 platform has 2 Denver 2.0 CPUs, four ARM Cortex A57 CPUs, and a 256-core Pascal GPU [17]. We do not have access to this device, so our evaluations are conducted on a desktop machine with Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz, 12GB DDR3 RAM @ 1.6GHz, 144-core GeForce GTX 635M GPU with a memory of 1GB.

Results. Figure 19 shows the average processing latency of each module in the pipeline. With all the optimizations (§4), AVR finishes the whole pipeline in 33 ms, enabling it to have a throughput of *30 frames per second* while finishing all the computations described above. The pipeline is well-balanced: the localization stage (thread 2) takes 33 ms, while the preprocessing stage requires 22 ms. Post-processing is at 30 ms. Within each stage, the boxes indicate the primary processing bottlenecks: GPU to CPU data transfer, feature detection and matching, and point cloud compression. In the future, we plan to focus on these bottlenecks.

The end-to-end processing latency for a given frame (e.g. the blue frame) through the three stages of the pipeline is below 100 ms. This is about twice the network transmission latency for dynamic objects (Figure 11). Processing latency can even be further reduced through careful engineering of the pipeline and choice of platform. Moreover, higher-end vehicles are likely to have lower processing latency since they can afford more on-board compute power than lower-end vehicles. These numbers explain why latency hiding is an important component of AVR.

Finally, AVR consumes on average 1.4GB RAM and 537 MB GPU memory, well within the capabilities of today's

platforms. Since on-board power sources are plentiful, we have not optimized for, and do not quantify, energy usage.

6 LIMITATION AND FUTURE WORK

AVR currently uses ZED [67], a short range stereo camera, which limits the system in two ways. First, we have experimentally observed that a shorter range depth sensor may be able to detect fewer 3D features in some environments, such as open roads with few roadside features (trees, buildings *etc.*). Since ORB-SLAM2's accuracy increases with the number of matched features, positioning accuracy can be low in these environments. Second, inclement weather and changing lighting conditions might result in a different set of features than those present in the 3D map. Fortunately, AVR does not need to match every feature in the map in order to localize a vehicle. It performs quite well in our experiments where our traces were gathered under different lighting conditions. Future work can improve localization accuracy when fewer features are visible. Also, these shortcomings of the stereo camera can be addressed by incorporating LiDAR into AVR and we plan to do so by incorporating a newly proposed LiDAR SLAM [34] algorithm.

One motivation for communicating point clouds in AVR is the use of a heads-up display. Heads-up displays can reduce distraction and improve safety [22]. While beyond the scope of this paper, it would be important for future work to perform user studies to determine if AVR-based displays like Figure 2 can improve driving decisions.

Cooperative AVR (§3.6) is a step towards scaling AVR out to clusters of vehicles, but future work can explore point cloud stream compression [29], as well as intermediate representations such as 3D features.

Finally, future work can explore lightweight representations of the 3D map. Currently, AVR stores all the stationary features and metadata for localization in the sparse 3D map, which is storage and communication-intensive. In future work, we plan to investigate lightweight representations and their impact on localization accuracy.

7 RELATED WORK

Connected Vehicles. Work on connected vehicles [47] explored technical challenges in inter-vehicle communications and laid out a research agenda for future efforts in this space [10]. Other research has studied content-centric [32] or cloud-mediated [31] inter-vehicle communication methods for exchanging sensor information between vehicles. Prior research has explored using connected vehicle capabilities for Collaborative Adaptive Cruise Control (CACC) [3]. This paper is an extension of our earlier work [45] and introduces methods to isolate and track dynamic objects, designs cooperative AVR, analyzes the redundancy to save bandwidth, and conducts an extensive evaluation aimed at exploring feasibility. Prior work on streaming video from leader to a follower for enhanced visibility [18, 42] does not use 3D sensors, so lacks the depth perception that is crucial for automated safety applications in autonomous driving or ADAS systems. Finally, automakers have started to deploy V2V/V2X communication in their upcoming high-end models *e.g.,* Mercedes E-class [1] and Cadillac [61]. A large connected vehicle testbed in the US aims to explore the feasibility and applications of inter-vehicle communication at scale [7, 60], while pilot projects in Europe (simTD [26] and C2X [52]) are evaluating these technologies. AVR depends upon these technologies, and is a compelling use of this technology.

Vehicle Sensor Processing. Prior work on sensor information processing has ranged from detection and tracking of vehicles using 360 degree cameras mounted on vehicles [14], to recognizing roadside landmarks [49], to using OBD sensors [28, 44] and phone sensors [6, 44] to detect dangerous driving behaviour, to systems that monitor blind spots using a stereo panoramic camera [36] and pre-crash vehicle detection systems [13]. Other work has proposed infrastructures to collect and exfiltrate vehicle sensor data to the cloud for analytics [27]. AVR is qualitatively different, focusing on sharing raw 3D sensor information between vehicles.

Localization. Localization of vehicles is crucial for autonomous driving [55, 8, 58] and this line of research has explored two avenues. GPS enhancements focus on improving the accuracy of absolute GPS-based positioning using differential GPS [20], inertial sensors on a smartphone [4, 21], onboard vehicle sensors, digital maps and crowd-sourcing [28], and WiFi and cellular signals [54]. These enhancements increase the accuracy to a meter. AVR builds upon a line of work in the robotics community on simultaneous localization and mapping [11]. Visual SLAM techniques have used monocular cameras [9], stereo camera [12, 66], and LiDAR [23]. Kinect [39] can also produce high-quality 3D scan of an indoor environment using infrared. AVR adds a relative localization capability to ORB-SLAM2 [38], using the observation that its sparse 3D feature map can be used to localize one vehicle's camera with respect to another.

8 CONCLUSIONS

In this paper, we have discussed the design and implementation of an AVR system which extends vehicular vision by enabling vehicles to communicate raw 3D sensor information. AVR can be used as input to driving assistance and autonomous driving systems. The design of AVR uses a novel relative localization technique, careful perspective transformation, dynamic object isolation, latency hiding using velocity vectors and adaptive frame transmission. Our AVR prototype is flexible enough to transmit full frames or dynamic objects, achieves 200ms end-to-end delay, can reconstruct objects to within 2-10% of car lengths and lane widths, and achieve 30 fps throughput.

Acknowledgements. The authors would like to thank our shepherd Dr. Junehwa Song and the anonymous reviewers for their valuable comments and helpful suggestions. This work is supported by the National Science Foundation under Grant No. 1330118 and a grant from General Motors.

BIBLIOGRAPHY

[1] *18 Awesome Innovations in the New Mercedes E-Class*. http://www.businessinsider.com/mercedes-e-class-2017-features-2016-6/ (Cited on page 12).

[2] F. Ahmad et al. "QuickSketch: Building 3D Representations in Unknown Environments using Crowdsourcing". In: *2018 21st International Conference on Information Fusion (Fusion)*. 2018, pp. 1–8 (Cited on page 5).

[3] Mani Amoozadeh et al. "Platoon Management with Cooperative Adaptive Cruise Control Enabled by VANET". In: *Vehicular Communications* 2.2 () (Cited on page 12).

[4] Cheng Bo et al. "SmartLoc: Push the Limit of the Inertial Sensor Based Metropolitan Localization Using Smartphone". In: *Proceedings of the 19th Annual International Conference on Mobile Computing and Networking*. MobiCom '13. 2013 (Cited on page 12).

[5] C. Chen et al. "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2722–2730. DOI: 10.1109/ICCV.2015.312 (Cited on page 3).

[6] Dongyao Chen et al. "Invisible Sensing of Vehicle Steering with Smartphones". In: *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '15. 2015 (Cited on page 12).

[7] *Connected Ann Arbor*. http://www.mtc.umich.edu/deployments/connected-ann-arbor (Cited on page 12).

[8] *DARPA Grand Challenge 2007*. http://archive.darpa.mil/grandchallenge/ (Cited on page 12).

[9] Andrew J. Davison et al. "MonoSLAM: Real-Time Single Camera SLAM". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 29.6 (June 2007) (Cited on page 12).

[10] Falko Dressler et al. "Inter-vehicle Communication: Quo Vadis". In: *IEEE Communications Magazine* 52.6 (2014), pp. 170–177 (Cited on page 12).

[11] Hugh Durrant-Whyte and Tim Bailey. "Simultaneous Localization and Mapping: Part I". In: *IEEE robotics & automation magazine* 13.2 (2006), pp. 99–110 (Cited on page 12).

[12] J. Engel, J. Stueckler, and D. Cremers. "Large-Scale Direct SLAM with Stereo Cameras". In: *iros*. 2015 (Cited on page 12).

[13] Tarak Gandhi and Mohan Trivedi. "Parametric Ego-motion Estimation for Vehicle Surround Analysis using an Omnidirectional Camera". In: *Machine Vision and Applications* 16.2 (2005), pp. 85–95 (Cited on page 12).

[14] Tarak Gandhi and Mohan M Trivedi. "Motion Based Vehicle Surround Analysis using an Omni-Directional Camera". In: *Intelligent Vehicles Symposium, 2004 IEEE*. IEEE. 2004, pp. 560–565 (Cited on page 12).

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012 (Cited on page 7).

[16] A. Gern, R. Moebus, and U. Franke. "Vision-based Lane Recognition Under Adverse Weather Conditions using Optical Flow". In: *Intelligent Vehicle Symposium, 2002. IEEE*. Vol. 2. 2002, 652–657 vol.2 (Cited on page 3).

[17] *Get Under the Hood of Parker, Our Newest SOC for Autonomous Vehicles*. https://blogs.nvidia.com/blog/2016/08/22/parker-for-self-driving-cars/ (Cited on page 11).

[18] P. Gomes, F. Vieira, and M. Ferreira. "The See-Through System: From Implementation to Test-drive". In: *2012 IEEE Vehicular Networking Conference (VNC)*. 2012, pp. 40–47. DOI: 10.1109/VNC.2012.6407443 (Cited on page 12).

[19] *Google Self-Driving Car Project Monthly Report September 2016*. https://static.googleusercontent.com/media/www.google.com/en//selfdrivingcar/files/reports/report-0916.pdf (Cited on page 2).

[20] Mahanth Gowda et al. "Tracking Drone Orientation with Multiple GPS Receivers". In: *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. MobiCom '16. 2016 (Cited on page 12).

[21] Santanu Guha et al. "AutoWitness: Locating and Tracking Stolen Property While Tolerating GPS and Radio Outages". In: *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. SenSys '10. 2010 (Cited on page 12).

[22] *Heads Up Display*. http://continental-head-up-display.com/ (Cited on page 12).

[23] Wolfgang Hess et al. "Real-Time Loop Closure in 2D LIDAR SLAM". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1271–1278 (Cited on page 12).

[24] *How Uber's First Self-Driving Car Works*. http://www.businessinsider.com/how-ubers-driverless-cars-work-2016-9 (Cited on page 2).

[25] Jing Huang and Suya You. "Point Cloud Matching based on 3D Self-similarity". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*. IEEE. 2012, pp. 41–48 (Cited on page 5).

[26] Dirk Hübner and G Riegelhuth. "A New System Architecture for Cooperative Traffic Centres-the Simtd Field Trial". In: *ITS World Congress*. 2012 (Cited on page 12).

[27] Bret Hull et al. "CarTel: a Distributed Mobile Sensor Computing System". In: *Proceedings of the 4th international conference on Embedded networked sensor systems*. SenSys '06. ACM. 2006, pp. 125–138 (Cited on page 12).

[28] Yurong Jiang et al. "CARLOC: Precise Positioning of Automobiles". In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. SenSys '15. 2015 (Cited on pages 4 and 12).

[29] J. Kammerl et al. "Real-time Compression of Point Cloud Streams". In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 778–785. DOI: 10.1109/ICRA.2012.6224647 (Cited on page 12).

[30] *KITTI Visual Odometry Benchmark*. http://www.cvlibs.net/datasets/kitti/eval_odometry.php (Cited on page 9).

[31] Swarun Kumar, Shyamnath Gollakota, and Dina Katabi. "A Cloud-assisted Design for Autonomous Driving". In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC '12. 2012 (Cited on page 12).

[32] Swarun Kumar et al. "CarSpeak: A Content-centric Network for Autonomous Driving". In: *SIGCOMM Comput. Commun. Rev.* 42.4 (Aug. 2012) (Cited on page 12).

[33] Mathieu Labbé and François Michaud. "Online Global Loop Closure Detection for Large-scale Multi-session Graph-based Slam". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 2661–2666 (Cited on page 5).

[34] *Lidar-Monocular Visual Odometry*. https://github.com/johannes-graeter/limo (Cited on page 12).

[35] Sathiya Kumaran Mani et al. "MNTP: Enhancing Time Synchronization for Mobile Devices". In: *Proceedings of the 2016 Internet Measurement Conference*. IMC '16. 2016 (Cited on page 5).

[36] Leanne Matuszyk et al. "Stereo Panoramic Vision for Monitoring Vehicle Blind-spots". In: *Intelligent Vehicles Symposium, 2004 IEEE*. IEEE. 2004, pp. 31–36 (Cited on page 12).

[37] D. L. Mills. "Internet Time Synchronization: the Network Time Protocol". In: *IEEE Transactions on Communications* 39.10 (1991), pp. 1482–1493 (Cited on page 5).

[38] Raul MurArtal, J. M. M. Montiel, and Juan D. Tardos. "ORB-SLAM: a Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* (2015) (Cited on pages 4, 5, and 12).

[39] Richard A. Newcombe et al. "KinectFusion: Real-time Dense Surface Mapping and Tracking". In: *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*. ISMAR '11. 2011 (Cited on page 12).

[40] *NHTSA Federal Accident Reporting System*. https://www-fars.nhtsa.dot.gov/Main/index.aspx (Cited on page 1).

[41] *NVidia Drive PX 2*. http://www.nvidia.com/object/drive-px.html (Cited on page 2).

[42] C. Olaverri-Monreal et al. "The See-Through System: A VANET-enabled Assistant for Overtaking Maneuvers". In: *2010 IEEE Intelligent Vehicles Symposium*. 2010, pp. 123–128. DOI: 10.1109/IVS.2010.5548020 (Cited on page 12).

[43] *ORB-SLAM Code*. http://webdiis.unizar.es/~raulmur/orbslam/ (Cited on page 7).

[44] H. Qiu et al. "Towards Robust Vehicular Context Sensing". In: *IEEE Transactions on Vehicular Technology* 67.3 (2018), pp. 1909–1922. ISSN: 0018-9545. DOI: 10.1109/TVT.2017.2771623 (Cited on page 12).

[45] Hang Qiu et al. "Augmented Vehicular Reality: Enabling Extended Vision for Future Vehicles". In: *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications*. ACM. 2017, pp. 67–72 (Cited on page 12).

[46] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). URL: http://arxiv.org/abs/1506.02640 (Cited on page 3).

[47] Dirk Reichardt et al. "CarTALK 2000: Safe and Comfortable Driving based upon Inter-vehicle-communication". In: *Intelligent Vehicle Symposium, 2002. IEEE*. Vol. 2. IEEE. 2002, pp. 545–550 (Cited on page 12).

[48] Radu Bogdan Rusu and Steve Cousins. "3D is here: Point Cloud Library (PCL)". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011 (Cited on page 7).

[49] Andrzej Ruta et al. "In-vehicle Camera Traffic Sign Detection and Recognition". In: *Machine Vision and Applications* 22.2 (2011), pp. 359–375 (Cited on page 12).

[50] Ruwen Schnabel and Reinhard Klein. "Octree-based Point-Cloud Compression." In: *Spbg*. 2006, pp. 111–120 (Cited on pages 5 and 6).

[51] Miguel Angel Sotelo et al. "A Color Vision-Based Lane Tracking System for Autonomous Driving on Unmarked Roads". In: *Auton. Robots* 16.1 (Jan. 2004), pp. 95–116. ISSN: 0929-5593 (Cited on page 3).

[52] R Stahlmann et al. "Starting European Field Tests for Car-2-X Communication: the DRIVE C2X Framework". In: *18th ITS World Congress and Exhibition*. 2011 (Cited on page 12).

[53] *Tesla Autopilot*. http://www.businessinsider.com/how-teslas-autopilot-works-2016-7 (Cited on page 2).

[54] Arvind Thiagarajan et al. "VTrack: Accurate, Energy-aware Road Traffic Delay Estimation Using Mobile Phones". In: *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. SenSys '09. 2009 (Cited on page 12).

[55] Sebastian Thrun et al. "Stanley: The Robot That Won the DARPA Grand Challenge: Research Articles". In: *J. Robot. Syst.* 23.9 (Sept. 2006), pp. 661–692. ISSN: 0741-2223 (Cited on pages 2 and 12).

[56] *TP-Link Talon AD7200 Multi-Band WiFi Router*. https://www.tp-link.com/us/products/details/cat-9_AD7200.html (Cited on page 7).

[57] *Two Second Rule*. https://en.wikipedia.org/wiki/Two-second_rule (Cited on page 9).

[58] C. Urmson and W. ". Whittaker. "Self-Driving Cars and the Urban Challenge". In: *IEEE Intelligent Systems* 23.2 (2008), pp. 66–68 (Cited on page 12).

[59] Chris Urmson et al. "Autonomous Driving in Urban Environments: Boss and the Urban Challenge". In: *J. Field Robot.* 25.8 (Aug. 2008), pp. 425–466. ISSN: 1556-4959 (Cited on page 2).

[60] *U.S. Details Plans for Car-to-car Safety Communications*. http://www.autonews.com/article/20140818/OEM11/140819888/u.s.-details-plans-for-car-to-car-safety-communications (Cited on page 12).

[61] *V2V Safety Technology Now Standard on Cadillac CTS Sedans*. http://media.cadillac.com/media/us/en/cadillac/news.detail.html/content/Pages/news/us/en/2017/mar/0309-v2v.html (Cited on page 12).

[62] *Vehicle Average Length*. https://www.reference.com/vehicles/average-length-car-2e853812726d079d (Cited on page 10).

[63] *Vehicle Stopping Distance and Time*. https://nacto.org/docs/usdg/vehicle_stopping_distance_and_time_upenn.pdf (Cited on page 9).

[64] *Velodyne LiDAR HDL-64E Datasheet*. http://velodynelidar.com/docs/datasheet/63-9194%20Rev-E_HDL-64E_S3_Spec%20Sheet_Web.pdf (Cited on page 2).

[65] Christoph Vogel, Konrad Schindler, and Stefan Roth. "3D Scene Flow Estimation with a Piecewise Rigid Scene Model". In: *Int. J. Comput. Vision* 115.1 (Oct. 2015) (Cited on page 6).

[66] Y. Xu et al. "3D Point Cloud Map Based Vehicle Localization using Stereo Camera". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 487–492. DOI: 10.1109/IVS.2017.7995765 (Cited on page 12).

[67] *ZED Stereo Camera*. https://www.stereolabs.com/ (Cited on pages 7 and 12).

[68] *ZED Stereo Camera Datasheet*. https://www.stereolabs.com/zed/specs/ (Cited on page 3).