Net-driving: An Alternative to Autonomous Driving

Fawad Ahmad∗
USC

Weiwu Pang∗
USC

Christina Shin∗
USC

Jacob Cashman
USC

Brandon Leong
USC

Pradipta Ghosh
USC

Ramesh Govindan
USC

ABSTRACT

Autonomous driving will become pervasive in the coming decades. Net-driving is a different point in the design space for autonomous driving, motivated by increasing deployments of 5G and edge computing, and the drop in prices of advanced sensors like LiDAR. In Net-driving, the network remotely "drives" vehicles by offloading perception and planning from the vehicle to roadside infrastructure. In this paper, we show that Net-driving can meet the performance needs for autonomous driving by processing voluminous sensor data at full frame rate with a tail latency of less than 100 ms, without sacrificing accuracy. It achieves this using an accurate and lightweight perception component that reasons on composite views derived from overlapping sensors, and a planner that jointly plans trajectories for multiple vehicles. Net-driving is safer than autonomous driving, and can enable higher throughput traffic management at intersections.

1 INTRODUCTION

Autonomous driving has made rapid strides in the past few years. Level-2 autonomous driving technologies such as adaptive cruise control and auto-steering are starting to be widely available in newer vehicles. By some estimates [47], full autonomy may be available at a price premium by the 2030s, and will get progressively cheaper in the subsequent decades.

A brief introduction to autonomous driving. Autonomous vehicles consist of three logical components [56, 57]: perception, planning, and control (Fig. 1(a)).

Perception. To extract knowledge of its surroundings, an autonomous vehicle uses one or more 3D depth-sensors (e.g., radars, stereo cameras, and LiDARs). A LiDAR, for example, sends millions of light pulses 10 or more times a second in all directions, whose reflections enable the LiDAR to estimate the position of objects or surfaces in the environment. The output of a LiDAR is a 3D point cloud that contains points with associated 3D positions (the same is true of stereo cameras). An autonomous vehicle’s perception module extracts an abstract scene description consisting of static and dynamic objects, from these point clouds. Static objects include road surfaces, lane markers [38], and safe drivable space [27]. The module also identifies dynamic objects (vehicles, pedestrians, bicyclists) and tracks them.

Planning. The planning module uses the abstract scene description from the perception component to plan and execute a safe path for the vehicle, while satisfying performance and comfort objectives [56] (e.g., smooth driving, minimal travel time etc.). Planning1 executes at the timescale of 100s of milliseconds and its output is a trajectory — a sequence of way-points, together with the precise times at which the vehicle must arrive at those way-points.

Control. This component takes a trajectory and converts it, at millisecond timescales, to low-level actuator (e.g., throttle, steering, brake) signals.

Net-driving: Motivating trends. In this paper, we explore a different point in the design space of autonomous driving, an approach we call Net-driving. In Net-driving, components in the network remotely “drive” vehicles (Fig. 1(b)), hence the name. Three technology trends motivate Net-driving. 5G, the fifth generation of cellular network technology [2, 4, 5], promises gigabit speeds and latency as low as 1 ms. Vehicle manufacturers are standardizing 5G technologies like CV2X [61] and LTE Direct [16] for cooperative perception, path-planning, and control. At the same time, edge-computing is moving computational resources (both CPU and GPU [6]) closer to data sources to support latency-critical applications [8]. By some estimates, edge-servers will handle 75% of the processing workloads handled by the cloud today [7]. Finally, LiDAR prices are falling dramatically due to competition and improved technology and manufacturing. A typical 64-beam LiDAR [45, 55] today costs a few thousand dollars. In the next few years LiDARs will likely cost a few hundred dollars [3, 77].

How Net-driving works. These trends motivate Net-driving’s design, in which LiDARs are cheap enough to mount ubiquitously on roadside light poles; edge-compute

∗Equal contributions

1This is a simplified description; [56] has a more detailed explanation.
Perception
Planning
Control
Perception
Planning
Control
(a) (b)
Roadside sensors
Figure 1: (a) Autonomous vehicles run perception, planning and control on-board. (b) Net-driving decouples perception and planning from control, placing these on edge-compute infrastructure.

Roadside LiDARs transmit data over a wired network to an edge cluster. Net-driving’s perception extracts vehicles and their motion information, and its planning component determines that the blue vehicle can safely pass the stalled vehicle. Net-driving wirelessly delivers a trajectory designed for the blue vehicle, whose control component executes the trajectory.

Net-driving generates vehicle trajectories based on more complete information than in autonomous driving. For example, at a busy intersection with lower wait times (or higher overall throughput).

Net-driving departs significantly from existing autonomous driving designs, so its full deployment will take time. Net-driving departs significantly from existing autonomous driving designs, so its full deployment will take time. Net-driving can, however, be incrementally deployed, with initial deployments in areas with high traffic congestion: streets downtown, intersections, ports etc. In these areas, Net-driving can improve traffic flow without compromising safety. For example, at a busy intersection, a Net-driving deployment might consist of four LiDARs mounted at light poles. As cars approach, they switch to following trajectories generated for them by Net-driving (as for the blue car in Fig. 2). In such a setting, Net-driving is safer than autonomous driving and can effect safe passage of vehicles without traffic lights (as we show in §4.5). Even in these deployments, much work is necessary; cost/benefit economic analyses, public-private partnerships to install infrastructure, regulatory approvals, buy-in from vehicle owners, etc. We have left these to future work; we focus on understanding technical feasibility and potential benefits.

Net-driving Challenge. Today, autonomous vehicles must process sensor data at frame rates (e.g., 10 fps for LiDAR) and produce trajectories with a tail latency of less than 100 ms [46] (faster than the time it takes a human to process a complex image [76]). If Net-driving cannot meet these requirements, it simply cannot be a viable alternative, since violations of these performance requirements can compromise safety. It is

Much work in transportation research focuses on traffic management at intersections, since these are potential choke points in traffic flow (the same reason that much networking research focuses on router designs).

3 We assume that because LiDARs are statically deployed on infrastructure, they have wired connectivity. Wirelessly streaming compressed LiDAR data can ease deployment, but we have left this to future work.

4 However, if when Net-driving is fully deployed, it can simplify autonomous driving: vehicles will no longer need expensive sensors thereby reducing cost [62, 65], nor power-hungry on-board compute [46], nor expensive high-definition maps [73].
not immediately obvious that Net-driving can meet these requirements, since the processing requirements for LiDAR or stereo camera data can be significant. For instance, the most accurate 2D and 3D object detectors on the KITTI leaderboard [34] require 60-300 ms; Net-driving must be able to not just detect objects, but determine their location and track them, using inputs from multiple sensors concurrently and generate trajectories for tens of vehicles or more at a time, all within 100 ms. Thus, to demonstrate viability of Net-driving, the central challenge is the design of perception and planning components (the Net-driving stack) that can meet these performance requirements on commodity hardware deployed on an edge cluster.

How Net-driving Addresses the Challenge. Net-driving’s stack is qualitatively different from those of open-source autonomous driving stacks like Autoware [51] and Baidu’s Apollo [10] (Tbl. 1), and exploits domain-specific properties. It does not require pre-built HD maps for positioning, but estimates positions of vehicles and traffic participants using composite views obtained from outputs of individual static LiDARs. Instead of operating on every LiDAR separately, Net-driving combines point clouds from multiple LiDARs early on in its processing pipeline. This early fusion enables robust and cheap 3D object detection (unlike other stacks that detect obstacles or use 2D camera-based object detection). Net-driving’s object detector outputs tight bounding boxes for participants; this abstraction is used extensively to accurately, yet cheaply, track and estimate the motion of vehicles (§2). Net-driving also uses a novel, GPU-accelerated, heading detection algorithm; simpler approaches have large errors in the tail.

Its planner (§3) jointly plans trajectories for multiple vehicles using perception outputs. It adapts prior work in distributed robotic motion planning [58] to support vehicles of different sizes as well as Net-driving-oblivious participants (e.g., pedestrians or legacy vehicles), to be robust to packet loss, and adds vehicle kinematic constraints (like stopping distances) and driving experience constraints (like safe driving distances). Other stacks use comparable planners, but only plan for the ego vehicle (Tbl. 1).

Summary of results. Evaluations on a testbed as well as on a photorealistic autonomous driving simulator [30] show that: Net-driving achieves its performance objectives end-to-end, being able to process frames from 3 LiDARs at 10 fps with a 99-th percentile tail latency under 100 ms; its perception component can track vehicles to centimeter-level accuracy (comparable to SLAM [53]-based positioning); and its planner results in fewer safety violations than autonomous driving and even alternative designs that rely on infrastructure-based perception, and can demonstrate safe traffic-light free intersections.

Contributions. We make the following contributions:

- A new architecture for autonomous driving vehicles which offloads perception and planning to the network edge.
- A novel, fast, pipeline for perception using composite views obtained from multiple roadside sensors.
- A fast planner for jointly planning trajectories of multiple vehicles while taking practical constraints into account.
- An end-to-end implementation that satisfies tail latency, throughput, and accuracy requirements.

Net-driving is inspired by recent and planned roadside deployments of advanced sensors, and a large body of work on augmenting connected autonomous driving vehicle perception using infrastructure-based sensors (§5). To our knowledge, no prior work has even considered combining infrastructure-based perception and planning — Net-driving demonstrates the end-to-end viability of such a capability.

2 NET-DRIVING PERCEPTION

Net-driving consists of two components (Tbl. 1): perception and planning. In this section, we describe the former.

2.1 Perception Overview

Inputs and Outputs. The input to perception is a continuous sequence of LiDAR frames from each LiDAR in a set of overlapping LiDARs deployed roadside. To ground the discussion, consider four LiDARs deployed at an intersection. Fig. 3(a) shows a view of this intersection and LiDARs are mounted on traffic light poles at the four corners.

The output of perception is a compact abstract scene description: a list of bounding boxes of moving objects in the environment together with their motion vectors (Fig. 3(e)). This forms the input to the planner (§3), which also requires other inputs that we describe later.

Goal and Challenge. Perception must obtain the outputs from each frame of the input (a) in under 100 ms (perception should leave some of the latency budget for planning and for wireless transmission, we discuss this in §4), (b) while achieving localization and tracking accuracy comparable to the state-of-the-art. This is challenging because a single frame of the 64-beam Ouster LiDAR we use is over 3 MB. At 10 fps, this corresponds to 240 Mbps, uncompressed. Extracting precise semantic information from multiple LiDARs, in our example 4 LiDARs, at 1 Gbps within a tight latency

3In this and subsequent sections, we focus on traffic management at intersections, a significant challenge in transportation research [43]. Net-driving generalizes to other settings (e.g., a busy thoroughfare), since it assumes nothing more than static deployment of overlapping LiDARs.

4Practical deployments of roadside LiDARs will need to consider coverage redundancy and other placement geometry issues, which are beyond the scope of this paper.
Net-driving uses a qualitatively different stack to meet throughput and latency requirements. Both perception and planning use different algorithms (highlighted in bold) than existing open-source stacks.

<table>
<thead>
<tr>
<th>Perception</th>
<th>Net-driving</th>
<th>Baidu Apollo [10]</th>
<th>Autoware [51]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapping</td>
<td>None</td>
<td>Pre-built HD-Map</td>
<td>Pre-built HD-Map</td>
</tr>
<tr>
<td>Localization</td>
<td>Live Fused Point Cloud ($\S 2.2$)</td>
<td>LiDAR scan matching fused with GPS-RTK and IMU</td>
<td>LiDAR scan matching</td>
</tr>
<tr>
<td>Object detection</td>
<td>3D object detection ($\S 2.3$)</td>
<td>Obstacle detection in 2D BEV projection</td>
<td>2D object detection in camera, projected to 3D</td>
</tr>
<tr>
<td>Tracking, Motion Estimation</td>
<td>Kalman filter with matching, Heading Estimation ($\S 2.4$)</td>
<td>Kalman filter with matching</td>
<td>Kalman filter with matching</td>
</tr>
</tbody>
</table>

**Planning**

| Centralized interval planner ($\S 3$) | Ego-planner using hybrid A* search | Ego-planner using conformal lattices |

**Table 1**: Net-driving uses a qualitatively different stack to meet throughput and latency requirements. Both perception and planning use different algorithms (highlighted in bold) than existing open-source stacks.

**Figure 3**: (a) A bird’s eye view of an intersection. LiDARs are mounted on the traffic light poles at each of the four corners of the intersection. (b) An individual frame from each of the 4 LiDARs. (c) A fused frame. (d) Point clouds of traffic participants at the intersection. (e) Bounding boxes and motion vectors for traffic participants, calculated over successive frames.

budget requires (a) careful algorithmic and implementation choices, (b) exploiting properties of the setting.

**Approach.** To achieve its goals, Net-driving’s perception uses a three stage pipeline, each with three sub-stages (Fig. 4). **Fusion** combines multiple LiDAR views into fused frames, and then subtracts the static background to reduce data. **Participant extraction** identifies traffic participants by clustering and estimates tight 3D bounding box around each object. **Tracking** associates objects across frames, and estimates heading and motion vectors of these objects.

This design achieves Net-driving’s goals using three ideas:

1. Net-driving exploits the fact that LiDARs are static to cheaply fuse point clouds from multiple overlapping LiDARs into a single **fused frame**. Such a frame may have more complete representations of objects than individual LiDAR frames. Fig. 3(b) shows the view from each of the 4 LiDARs; in many of them, only parts of a vehicle are visible. Fig. 3(c) shows a **fused frame** which combines the 4 LiDAR frames into one; in this, all vehicles are completely visible.\(^7\)

2. Net-driving builds most algorithms around a single abstraction, the 3D bounding box of a participant. Its tracking, speed estimation, and motion vector estimation rely on the observation that the centroid of the bounding box is a convenient consistent point on the object for estimating these quantities, especially when fused LiDAR frames provide comprehensive views of an object.

3. Net-driving uses, when possible, cheap algorithms rather than expensive deep neural networks (DNNs). Only when accuracy is important does Net-driving resort to expensive algorithms, but employs hardware acceleration to meet the latency budget; its use of a specialized heading vector estimation algorithm is an example.

**Qualitative design comparison.** Net-driving’s perception design is qualitatively different (Tbl. 1) from that of other popular open-source autonomous driving stacks, Baidu’s Apollo [10] and Autoware [51], in three different ways. (a) Those approaches rely on a pre-built HD-map, and localize the **ego-vehicle** (the one on which the autonomous driving stack runs) by matching its LiDAR scans against the map (Apollo additionally improves position estimates using multi-sensor fusion). In contrast, Net-driving does not require a map for positioning: all vehicle positions can be directly estimated from the fused frame. (b) Autonomous driving stacks use different ways to identify traffic participants. Apollo extracts obstacles on a 2-D occupancy grid and infers participants

\(^7\)This may not be immediately obvious from the figure, but a LiDAR frame, or a fused frame is a 3D object, of which Fig. 3(c) is a 2D projection.
Point Cloud Alignment. Each frame of a LiDAR contains a point cloud, a collection of points with 3D coordinates. These coordinates are in the LiDAR’s own frame of reference. Fusing frames from two different LiDARs is the process of converting all 3D coordinates of both point clouds into a common frame of reference. Alignment computes the transformation matrix for this conversion.

If the precise 3D positions and 3D orientations (or pose) of the LiDARs are known, the transformation matrix can be trivially computed from these. This is hard to do in practice. A strawman approach might use GPS to obtain the pose of the LiDAR. Even if GPS were perfect, this is not sufficient: we also need to know the transformation between the GPS (geoidal) coordinate system, and the local coordinate system chosen by the LiDAR hardware to position the points. LiDAR hardware positions points with the LiDAR at the origin, but at an unknown orientation. We need to align the orientation with the geoidal coordinate system, and we know of no way of doing this using GPS alone.

ICP and the importance of good initial guesses. Prior work has developed Iterative Closest Point (ICP) [17, 26] techniques that search for the lowest error alignment. The effectiveness of these approaches depends upon the initial guess for LiDARs’ poses. Poor initial guesses can result in local minima. Most uses of ICP attempt to align successive point clouds obtained from a LiDAR on a moving vehicle; in these cases, a good initial guess for a frame can be obtained

from the previous frame. Our setting is different, so we have developed a novel technique to obtain initial guesses.

**Initial guesses using minimal information.** Net-driving only needs the distances on the ground (using, for example, an off-the-shelf laser rangefinder) between a reference LiDAR and all others to get good initial guesses. We now describe the algorithm for two LiDARs $L_1$ and $L_2$; the technique generalizes to multiple LiDARs, but we omit that description for brevity.

**Inputs and outputs.** The inputs are two point clouds $C_1$ and $C_2$ captured from the corresponding LiDARs at the same instant, and the distance $d$ on the ground between the LiDARs. The output is an initial guess for the pose of each LiDAR. We feed these guesses into ICP to obtain good alignments.

**Fixing the base coordinates.** Without loss of generality, set LiDAR $L_1$’s $x$ and $y$ coordinates to be $(0, 0)$ (i.e., the base is at the origin). Then, assume that $L_2$’s base is at $(d, 0)$.

**Estimating height, roll and pitch.** In this step, we determine: the height of each LiDAR $z_i$, the roll (angle around the $x$ axis), and pitch (angle around the $y$ axis). For these, Net-driving relies on fast plane-finding algorithms [29] that extract planes from a collection of points. These techniques output the equations of the planes. Assuming that the largest plane is the ground-plane (a reasonable assumption for roadside LiDARs), Net-driving aligns the $z$ axis of two LiDARs with the normal to the ground plane. In this way, it implicitly fixes the roll and pitch of the LiDAR. Moreover, after alignment, the height of the LiDAR $z_i$ is also known (because the $z$ axis is perpendicular to the ground plane).

**Estimating yaw.** Finally, to determine yaw (angle around the $z$ axis), we use a technique illustrated in Fig. 5. Consider a point cloud visible in the two LiDARs. Net-driving searches for the combination of yaw settings for the two LiDARs (Fig. 5(a) and (b)) until it finds a combination that results in the smallest angle between the two point cloud’s orientations (Fig. 5(c)). We have found that ICP is robust to initial guesses for yaw that are within about 15-20$\degree$ of the actual yaw, so Net-driving discretizes the search space by this amount.
Net-driving repeats this procedure for every other LiDAR \( L_i \) with respect to \( L_1 \), to obtain initial guesses for the poses of every LiDAR. It feeds these into ICP to obtain an alignment.

**Re-calibrating alignment.** This algorithm is run only once, when installing the LiDARs. Re-alignment may be necessary if a LiDAR is replaced or re-positioned.

**Using alignment: Stitching and Background Subtraction.** LiDARs generate frames at 10 fps (or more). In Fig. 3, when each LiDAR generates a frame, the fusion stage performs stitching and background subtraction. Stitching applies the coordinate transformation for each LiDAR generated by alignment, resulting in a fused frame (Fig. 3(c)).

Background subtraction, which removes points belonging to static parts of the scene, is (a) especially crucial for voluminous LiDAR data and (b) feasible in our setting because the LiDARs are static. It requires a calibration step to extract a background point cloud from each LiDAR [41], then creates a background fused frame using the results from alignment; taking the intersection of a few successive point clouds and the aggregating intersections taken at a few different time intervals, works well to generate the background point cloud.

**Optimization: background subtraction before stitching.** Net-driving subtracts the background fused frame from each fused frame generated by stitching. We have found that a simple optimization can significantly reduce processing latency: first removing the background from each LiDAR frame (using its background point cloud), and then stitching points in the residual point clouds. Stitching scales with the number of points, which this optimization reduces significantly.

**Optimization: exploiting LiDAR characteristics.** Many LiDAR devices only output returns from reflected laser beams. Generic background subtraction algorithm requires a nearest-neighbor search to match a return with the corresponding return on the background point cloud. Some LiDARs (like the Ouster), however, indicate non-returns as well, so that the point cloud contains the output of every beam of the LiDAR. For these, it is possible to achieve fast background subtraction by comparing corresponding beam outputs in a point cloud and the background point cloud.

At the end of these two steps, the fused frame only contains points belonging to objects that are not part of the scene’s background (Fig. 3(d)).

### 2.3 Reusing 3D Bounding Boxes

Net-driving’s efficiency results from re-using the 3D bounding box of a participant (Fig. 3(e)) in many processing steps. Net-driving uses a standard, fast, clustering algorithm (DBSCAN) [32] to extract multiple clusters of points, where each cluster represents one traffic participant. Then, Net-driving uses an off-the-shelf algorithm [1], which determines a minimum oriented bounding box (Fig. 3(d)) of a point cloud using principal component analysis (PCA). From these, we can extract the three surface normals of the object (e.g., vehicle): the vertical axis, the axis in the direction of motion, and the lateral axis.

**Using the 3D bounding box.** Net-driving uses the bounding box for many of its algorithms. This results in low latency perception without compromising accuracy.

- To associate objects across frames (tracking), it uses a Kalman filter to predict the position of the centroid of the 3D bounding box, the finds the best match (in a least-squares sense) between predicted positions and the actual positions of the centroids in the frame. Although tracking in point clouds is a challenging problem [78] for which research is exploring deep learning, this approach works exceedingly well in our setting. The biggest challenge in tracking is occlusions: when one object occludes another in a frame, it may be mistaken for the other in subsequent frames (an ID-switch). Because our fused frame includes perspectives from multiple LiDARs, ID-switches occur rarely (in our evaluations, §4).

- To estimate speed of a vehicle, Net-driving measures the distance between the centroid of the bounding box in one frame, and the centroid \( w \) frames in the past \( (w \) is a configurable window size parameter), then estimates speed by dividing the distance by the time taken to generate \( w \) frames.

- Net-driving needs to associate an object seen in the LiDAR with a cyber endpoint (e.g., an IP address) so its planning component can send a customized trajectory to each vehicle. For this cyberphysical association (Fig. 4), Net-driving uses a calibration step performed once. In this step, given a vehicle for which we know the cyberphysical association (e.g., a LiDAR installer’s vehicle), we estimate the transformation between the trajectory of the vehicle seen in the LiDAR view with the GPS trajectory (we omit the details for brevity). When Net-driving runs, it uses this to transform a vehicle’s GPS trajectory to its expected trajectory in the scene, then matches actual scene trajectory to expected trajectory in a least-squares sense. To define the scene trajectory, we use the centroid of the bounding box of the vehicle.

In all of these tasks, the centroid of the bounding box is an easily computed, consistent, point within the vehicle that aids these tasks. Moreover, because we have multiple LiDARs that capture a vehicle from multiple directions, the centroid of the bounding box is generally a good estimate of the actual centroid of the vehicle.

Besides these, Net-driving (a) estimates heading direction from the axes of the bounding box (discussed in §2.4) and (b) uses the dimensions of the box to represent spatial constraints for planning (discussed in §3).
2.4 Fast, Accurate Heading Vectors

Planning also needs the motion vector to predict trajectories for vehicles. To compute this, Net-driving first determines, for each object, its instantaneous heading (direction of motion), which is one of the three surface normals of the bounding box of the vehicle. It estimates the motion vector as the average of the heading vectors in a sliding window of w frames. Most autonomous driving stacks can obtain heading from SLAM or visual odometry (§5), so little prior work has explored extracting heading from infrastructure LiDAR frames.

A strawman approach. Consider an object A at time t and time t + 1. Fig. 6(a) shows the points belonging to that object. Since those points are already in the same frame of reference, a strawman algorithm is: (a) find the vector between the centroid of A at time t and centroid of A at time t + 1, (b) the heading direction is the surface normal (from the bounding box) that is most closely aligned with this vector (Fig. 6(a)). We have found that the error distribution of this approach can have a long tail (although average error is reasonable). If A has fewer points in t + 1 than in t (Fig. 6(b), upper), the computed centroid will be different from the true centroid, which can induce significant error.

Net-driving’s approach. To overcome this, we (a) use ICP to find the transformation matrix between A’s point cloud in t and in t + 1, (b) “place” A’s point cloud from t in frame t + 1 (Fig. 6(b), lower), (c) then compute the vector between the centroids of these two (so that the centroid calculations are based on the same set of points). As before, the heading direction is the surface normal (from the bounding box) that is most closely aligned with this vector.

GPU acceleration. ICP is compute-intensive, even for small object point clouds. If there are multiple objects in the frame, Net-driving must run ICP for each of them. We have experimentally found this to be the bottleneck, so we developed a fast GPU-based implementation of heading vector estimation, which reduces the overhead of this stage (§4.3).

Moreover, this step scales linearly with the number of vehicles but parallelizes easily to multiple GPUs; at intersections with many vehicles, Net-driving can use edge computing resources with multiple GPUs.

3 NET-DRIVING PLANNING

Inputs, Outputs, and Goals. From perception, Net-driving’s planner receives bounding boxes for non-background objects, and their motion vectors. Some of these objects will be Net-driving-capable vehicles; these follow trajectories designed by Net-driving. For these, Net-driving needs each vehicle’s navigation goal, e.g., where it is headed: we assume the planning component can get this by communicating with the vehicle. Other objects will be Net-driving-oblivious: these include pedestrians, or vehicles driven by humans or on-board autonomous driving agents. The output of planning is a trajectory — a sequence of way-points, and times at which those way-points should be reached — for each Net-driving-capable vehicle.

Planning executes at every frame and must satisfy two goals: (a) it must fit within the tail latency budget of 100 ms, (b) and it must generate collision-free trajectories at every instant even in the presence of Net-driving-oblivious objects.

Prior Work in Planning. Autonomous driving uses decentralized motion planning algorithms that search for a path for the ego-vehicle in graphs, based on variants of A* [21] search. For example, Autoware uses a lattice-based discretization of the state space [50] for the search, while Baidu Apollo [10] uses a modified A* search with heuristics to guide the search. In contrast, Net-driving requires a centralized motion planner (one that plans for all vehicles, not just the ego-vehicle), the best known of which is Conflict Based Search [70], but this algorithm has high tail latency [14, 58].

SIPP. Net-driving adapts a fast, decentralized, planner for a single robot for environments with dynamic obstacles, SIPP [58]. A dynamic obstacle is one whose position changes over time. SIPP discretizes space into a 2D occupancy grid (Fig. 7(a)) and plans a path for a given robot from a one
location (source) to another (destination). Planning in dynamic environments adds an extra dimension i.e., time, to the search space which can increase computational complexity. SIPP’s efficiency relies on the following intuition. Consider an arbitrary grid element (e.g., a square on the road surface). The time steps \((t_1, t_2, \ldots, t_n)\) that this element is safe for robot occupancy (meaning no other robot occupies it at that time) is unbounded and infinitely large. However, the number of time intervals \(([t_1, t_n])\) at which that location is safe is bounded and generally small. A (maximally) safe time interval \(([t_1, t_n])\) in one in which the location is safe for occupancy throughout the interval, but not safe one step before \(t_0\) and one step after \(t_{n+1}\) the interval.

The input to SIPP is a goal for a robot and the positions over time of the dynamic obstacles in the environment. Using the latter, SIPP first generates a sequence, for each grid element, of safe and unsafe intervals. Then, using the former, SIPP uses \(A^*\) search to find a provably collision-free shortest path for the robot [58].

Net-driving’s Planner. At each frame, Net-driving must plan trajectories for every Net-driving-capable vehicle. Without loss of generality, assume that vehicles are sorted in some order. Net-driving’s centralized planner runs SIPP for each vehicle in order; when running SIPP on the \(i\)-th vehicle, Net-driving represents all \(i-1\) previously planned vehicles as dynamic obstacles in SIPP (Fig. 7(b)).

This approach may sacrifice optimality for speed: vehicles later in the planning order may see longer transit times. However, the approach may allow Net-driving to achieve novel traffic policies (e.g., those that prioritize emergency vehicles). Future work can explore such traffic policies.

Because it was designed for robots, SIPP makes some idealized assumptions. Net-driving adapts SIPP to relax these.

Robustness to Net-driving-oblivious participants. SIPP plans the entire trajectory for a robot once, given all the dynamic obstacles in the environment. Net-driving, to deal with Net-driving-oblivious vehicles whose trajectories can change dynamically, re-plans trajectories for every Net-driving-capable vehicle at every frame. This also relaxes assumption in SIPP that all objects move at a fixed speed. For each Net-driving-oblivious object, Net-driving uses its bounding box, and motion and heading vector.

Robustness to perception errors. SIPP assumes that each object has the same size, occupying one grid element. To relax this, Net-driving uses the bounding boxes generated by perception to determine grid occupancy. Because these bounding boxes may be incorrect, Net-driving extends the bounding box by a small buffer on all sides. This extended buffer is motion-adaptive: the faster the vehicle travels the larger is the buffer around it (Fig. 7(c)).

Robustness to packet losses. The trajectory generated by Net-driving is transmitted over a wireless network to the vehicle. To be robust to packet losses, Net-driving generates waypoints over a longer horizon (10 s) than the inter-frame time (100 ms in our case). This way, it transmits trajectories every 100 ms, but if a transmission is lost, the vehicle can use the previously-received trajectory.

Robustness to vehicle kinematics limitations. SIPP assumes that a vehicle can start and stop instantaneously. In practice, a vehicle’s kinematic characteristics dictate how quickly vehicles can start and stop. The longer planning horizon helps with the vehicle’s controller time to adapt to vehicle kinematics. The controller can determine, from the received trajectory, if it has to stop at some future point in time; it can then decide when to apply the brakes, and with what intensity, to effect the stop.

Net-driving also extends on-board control to respect vehicle kinematic constraints; we omit these for brevity.

4 EVALUATION

Our evaluations show that Net-driving: achieves frame rate processing and less than 100 ms tail latency both in real-world experiments (§4.2) and in simulation (§4.3); achieves accuracy comparable to prior work (§4.4); can be safer than autonomous driving (§4.5); and permits high-throughput traffic management (§4.6).

4.1 Methodology

Implementation. We implemented Net-driving’s perception and planning on the Robot Operating System (ROS [63]). ROS provides inter-node (ROS modules are called nodes) communication using publish-subscribe, and natively supports points clouds and other data types used in perception-based systems. Net-driving’s perception component runs as a ROS node that subscribes to points clouds, processes them as described in §2, and publishes the results. Net-driving’s planner also runs as a ROS node, subscribes to the perception results, and publishes trajectories for each vehicle. Perception requires 6909 lines of C++ code, and planning 3800.

Real-World Testbed. Our testbed consists of three 64-beam Ouster [55] LiDARs (two with 90° and one with a 45° field of view). The testbed also includes an edge computing unit with an Intel i9-10900KF CPU (20 core, 3.7GHz) and a GeForce RTX 3080 GPU; Net-driving’s planning and perception components run on this. The LiDARs and the edge computing unit connect through Ethernet cables and an ethernet switch (using an off-the-shelf Wi-Fi access point). Finally, Net-driving

If there is a catastrophic failure, Net-driving-capable vehicles must have a fallback-to-human strategy, but autonomous driving will also need this.
transmits trajectories to an in-vehicle Raspberry Pi 3 over Wi-Fi (as a proxy for 5G).

Simulation. In our testbed, Net-driving-derived trajectories cannot be used to control vehicles, since vehicular control systems are closed. So, we complement our evaluations with a simulator, which also helps us explore scaling of perception and planning. CarLA [30] is an industry-standard photorealistic simulator for autonomous driving perception and planning. It contains descriptions of virtual urban and suburban streets, and, using a game engine, can (a) simulate the control of vehicles in these virtual worlds, and (b) produce LiDAR point clouds of time-varying scenes within these virtual worlds. Unless otherwise noted, our simulation based evaluations focus on intersections; several challenge scenarios for autonomous driving focus on intersections [22, 54].

Metrics. We quantify end-to-end performance in terms of the 99th percentile of the latency (p99 latency) between when a LiDAR generates a frame, and when the vehicle receives the trajectory corresponding to that frame. To quantify accuracy of individual perception components, we use metrics described in prior work (defined later). We quantify planning efficacy by the rate of safety violations, or of undesirable outcomes such as deadlock.

Comparison. Relative to autonomous driving, Net-driving’s perception has a more comprehensive scene understanding and can jointly plan for multiple vehicles at a time. To quantify this, we compare it against autonomous driving.

4.2 Real-world Experiment

We deployed our testbed along a thoroughfare near our campus, and collected data for nearly 23 minutes.

Results. Fig. 8 shows the end-to-end latency for each frame, for over 14000 frames (23 min), broken down by component. The y-axis is in log scale. The p99 latency is 49 ms; in our experiment, a small number of frames exceeded 100 ms. Moreover, Net-driving processed LiDAR input at full frame rate. This is encouraging, and suggests that it is possible to achieve end-to-end performance with Net-driving comparable to those in today’s autonomous driving.

From this graph, we also observe that: network latency is small, as is planning latency (together they are under 7-8 ms); however, planning scales as the number of vehicles, so in more realistic deployments, we expect its share of latency to be higher; and, perception dominates, which motivates the careful algorithmic and implementation choices in §2.

Perception latencies were higher (p99 of 37 ms) than in our CarLA-based evaluations (§4.3). In our deployment, perception latency was affected by object size. Our LiDARs were mounted at lower heights (2 m) than one might expect in a roadside deployment (to mimic which, our LiDARs in CarLA were mounted at 11 m). As a result, the point clouds corresponding to individual objects are larger, and several components that scale with the number of points of an object (like clustering, and heading vector estimation) take more time in our experiments. In practical deployments, point cloud size may not affect performance; because LiDARs scans are radial, the number of points representing an object drops off quickly with distance from the sensor.

4.3 Latency Breakdown

To explore the total latency with more vehicles, and to understand the breakdown of latency by component, we designed several scenarios in CarLA with increasing numbers of vehicles concurrently traversing an intersection. In these scenarios, CarLA (a) generates LiDAR data and (b) controls vehicles based on received trajectories; the results use the same Net-driving code used in the real-world experiment.

Breakdown for Perception. Tbl. 2 depicts the breakdown of 99-th percentile (p99) latency by component for perception, as well as the total p99 latency, as a function of the number of vehicles in the scene. In all our experiments, Net-driving processed frames at the full frame rate (10 fps).

Table 2: p99 per-frame latency for perception.
Table 3: Impact of optimizations on p99 latency

<table>
<thead>
<tr>
<th>Order of stitching background subtraction</th>
<th>Before</th>
<th>After</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploiting LiDAR characteristics in background subtraction</td>
<td>9.95</td>
<td>1.5</td>
<td>6.63</td>
</tr>
<tr>
<td>GPU acceleration of heading vector estimation</td>
<td>1057.72</td>
<td>28.86</td>
<td>36.65</td>
</tr>
</tbody>
</table>

The total p99 latency for perception increases steadily up to 82 ms for 14 vehicles from 20 ms for 2 vehicles. This highlights the dependency of perception on scene complexity (§2); many components scale with the number of participants. At 14 vehicles, Net-driving exceeds the latency budget, since planning requires over 20 ms for 14 vehicles (see below). These numbers suggest that modest off-the-shelf compute hardware that we have used in our experiments might be sufficient, at least for traffic management at moderately busy intersections. This data dependency also suggests that deployments of Net-driving will need to carefully provision their infrastructures based on historical traffic (similar to network planning and provisioning).

The three most expensive components are background subtraction, clustering, and heading vector estimation. Background subtraction accounts for about 10 ms, but depends slightly on the number of vehicles; to be robust, it uses a filter (details omitted in §2.2) that is sensitive to the number of points (or vehicles). Clustering accounts for about 20 ms with 14 vehicles and is strongly dependent on the number of vehicles, since each vehicle corresponds to a cluster.

Heading vector estimation accounts for nearly 65% of perception latency, even after GPU acceleration (§2.4). Not shown in these results is the fact that heading vector estimation is not only dependent on the number of vehicles, but their dynamics as well. When we ran perception on 16 vehicles, p99 latency actually dropped; in this setting, 16 vehicles congested the intersection, so each vehicle moved very slowly. Heading vector estimation uses ICP between successive object point clouds; if a vehicle hasn’t moved much, ICP converges faster, accounting for the drop.

Other components are negligible. Stitching is fast because of the optimization described in §2.2. Bounding box estimation is inherently fast. Track association is cheap because it tracks a single point per vehicle (the centroid of the bounding box). Motion estimation relies on positions computed during stitching. Thus, careful design choices that leverage abstractions and quantities computed earlier in the pipeline can be crucial for meeting latency targets (§2.4).

Benefits of optimizations. Tbl. 3 quantifies the benefits of our optimizations. Stitching before background subtraction requires nearly 70 ms in total; reversing the order reduces this time by 6.7×. By exploiting LiDAR characteristics (§2.2), Net-driving can perform background subtraction in 1.5 ms per frame. A CPU-based heading vector estimation requires nearly 1 s; which would have rendered Net-driving infeasible; GPU acceleration (§2.4) reduces latency by 35×.

Calibration Steps. Finally, alignment (§2.2) of 4 LiDARs takes about 4 minutes. This includes not just the time to guess initial positions, but to run the ICP (on a CPU). Because it is invoked infrequently, we have not optimized it.

Planning. Tbl. 4 depicts the p99 planning latency. As expected, there is a dependency on the number of vehicles, since Net-driving individually plans for each vehicle. Planning latencies can be slightly non-monotonic (the planning cost for 12 vehicles is more than that for 14) because the planner’s graph search can depend upon the actual trajectories of the vehicles, not just their numbers.

4.4 Accuracy

Using both from real-world data and simulation traces, we show that Net-driving’s positioning, heading, velocity, and tracking accuracies are comparable to that reported in other work. (For our real-world traces, we manually labeled ground-truth positions).

Metrics. We report positioning error (in m), which chiefly depends upon the accuracy of alignment. Heading accuracy is measured using the average deviation of the heading vector, in degrees, from ground truth. For estimating accuracy of velocity estimation, we report both absolute and relative errors. Finally, we use two measures to capture tracking performance [18]: multi-object tracking accuracy (MOTA) and precision (MOTP). The former measures false positives and negatives as well as ID switches (§2); the latter measures average distance error from the ground-truth track.

Results. Tbl. 5 summarizes our findings. Positioning error is about 8-10 cm in Net-driving, both in simulation and in the real world; the state of the art LiDAR SLAM [85] reports about 15 cm error. Our heading estimates are comparable to prior work that uses a neural network to estimate heading. Speed estimates are highly accurate, both in an absolute sense (error of a few cm/s) and in a relative sense (over 97%).

For many of our experiments, including this one, we have generated videos to complement our textual descriptions. These are available at an anonymous YouTube channel: https://www.youtube.com/channel/UCpb947nEBAv_oikE7KoC-A.

Table 4: p99 per-frame planning latency.

<table>
<thead>
<tr>
<th># of Vehicles</th>
<th>2</th>
<th>4</th>
<th>7</th>
<th>10</th>
<th>12</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>p99 Latency</td>
<td>2.39</td>
<td>5.09</td>
<td>11.89</td>
<td>17.12</td>
<td>28.71</td>
<td>24.25</td>
</tr>
</tbody>
</table>
Finally, tracking is also highly accurate. In the real-world experiment, tracking was perfect. In simulation, with 10 vehicles concurrently visible, MOTA is still very high (over 99%), and significantly outperforms the state-of-the-art neural network in 3D tracking [40], for two reasons. The neural network solves a harder problem, tracking from a moving LiDAR. Our fused LiDAR views increase tracking accuracy; when using a single LiDAR to track, MOTA falls to 93%. Finally, MOTP is largely a function of positioning error, so it is comparable to that value.

### 4.5 Safety

Net-driving’s perception has a more comprehensive view of the scene (e.g., at an intersection), so Net-driving can lead to increased safety. To demonstrate this, we implemented two scenarios in CarLA from the US National Highway Transportation Safety Administration (NHTSA) precrash typology [54]; these are challenging scenarios for autonomous driving [22].

**Red-light violation.** A red truck and the ego-vehicle (yellow box) approach an intersection (Fig. 9). An oncoming vehicle (red box) on the other road violates the red traffic light. The red truck can see the violator and hence avoid collision, but the ego-vehicle cannot.

**Unprotected left-turn.** The ego-vehicle (yellow box) heads towards the intersection (Fig. 10). A vehicle (red bounding box) on the opposite side of the intersection wants to make an unprotected left-turn. The ego-vehicle’s view is blocked by the trucks on the left lane.

**Methodology and metrics.** In each scenario, the ego-vehicle is Net-driving-capable. When comparing against autonomous driving, to ensure a more-than-fair comparison, we: (a) equip autonomous driving with ground-truth, so its perception is perfect; and (b) use Net-driving’s planner and controller for autonomous driving in single-vehicle mode (it plans only for the ego-vehicle). The alternative would have been to use an open-source autonomous driving stack like Autoware [51] which has its own perception and planning modules. However, in these experiments we are trying to understand the impact of architectural differences (autonomous driving vs. Net-driving), we chose a simpler approach that equalizes implementations.

For both scenarios, we vary speeds and positions of the ego-vehicle and oncoming vehicle to generate 16 different experiments. We then compare for what fraction of experiments each approach can guarantee safe passage.

**Results.** In both scenarios (Tbl. 6), autonomous driving ensures safe passage in fewer than 20-40% of the cases. Net-driving achieves safe passage in all cases because it senses the oncoming traffic even though it is occluded from the vehicle’s on-board sensors\(^ {12}\). This gives the planner enough time to react and plan a collision avoidance maneuver (in this case, stop the vehicle). Of the two cases, the unprotected left-turn was the more difficult one for Net-driving (as it is for autonomous driving, which fails more often in this case) yet it is still able to guarantee safe passage in all 16 cases. In the red-light violation scenario, Net-driving senses the oncoming traffic early on and has enough time to react. However, in the unprotected left-turn, the ego-vehicle is traveling relatively fast and the oncoming traffic takes the left-turn at the last moment. Though Net-driving has a smaller time to react, its motion-adaptive bounding box and stopping distance estimation ensure that the vehicle stops on time.

**Robustness to Packet Loss.** Net-driving’s planner generates trajectories over a longer time horizon (§3) to be robust to packet loss. To quantify its robustness, we simulated packet losses ranging from 0 to 100% for in the red-light violation scenario (Fig. 9). For each loss rate, we measured the stopping distance between the ego-vehicle and the oncoming traffic which violates the red-light. Higher stopping distances are good (Fig. 12). We repeated the experiment multiple times for each loss rate. As we increased the packet-loss, the stopping distance decreased because the ego-vehicle was operating on increasingly stale information. Even so, Net-driving ensures collision-free passage for the ego-vehicle through the intersection till 70% loss, with minimal degradation in stopping distance till about 40% loss.

### 4.6 High-Throughput Traffic Management

Intersections contribute significantly to traffic congestion [43]; traffic-light free intersections can reduce congestion. We verified that Net-driving, because it centrally plans trajectories for all Net-driving-capable vehicles, is able to provide safe passage to vehicles in both scenarios.

<table>
<thead>
<tr>
<th>Positioning Error (m)</th>
<th>Real</th>
<th>Sim</th>
<th>Prior</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOTA (%)</td>
<td>99.54</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>MOTP (m)</td>
<td>0.04</td>
<td>0.06</td>
<td>0.04</td>
</tr>
</tbody>
</table>

**Table 5: Perception accuracies from real-world data and simulation. The last column provides accuracies reported for these by prior work, for calibration, where available.**

<table>
<thead>
<tr>
<th>Scenario</th>
<th>NHTSA Safe Passage (%)</th>
<th>Net-driving Safe Passage (%)</th>
<th>Autonomous Driving Safe Passage (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red-light Violation</td>
<td>100</td>
<td>37</td>
<td>18</td>
</tr>
<tr>
<td>Unprotected Left-turn</td>
<td>100</td>
<td>18</td>
<td>18</td>
</tr>
</tbody>
</table>

**Table 6: With more comprehensive perception, Net-driving can provide safe passage to vehicles in both scenarios.**

\(^ {12}\)Please see YouTube channel for videos.
to plan collision-free trajectories for up to 10 vehicles at a time at an intersection without traffic lights.\footnote{Please see YouTube channel for videos.}

This can significantly reduce wait times at intersections, thereby enabling higher throughput. To demonstrate this, we compared the average wait times for all vehicles for four approaches: a) conventional/static traffic lights, b) intelligent traffic light control which prioritizes longer queues, c) Net-driving, and d) autonomous-vehicles with centralized perception but on-board (decentralized) planning. For the first two, we obtained policies from published best practices \cite{9, 11, 12}.

**Results.** Compared to static and intelligent traffic lights, Net-driving reduces average wait times for vehicles by up to 5× (Fig. 11). Net-driving performs better than even intelligent traffic lights because it can minimize the stop and start maneuvers at the intersection (by preemptively slowing down some vehicles), and hence can increase overall throughput, leading to lower wait times.\footnote{Please see YouTube channel for videos.}

Finally, although one might expect that a decentralized planner might have comparable throughput to Net-driving, we found that it led to a deadlock with as few as two vehicles (each vehicle waited indefinitely for the other to make progress, resulting in zero throughput).\footnote{Please see YouTube channel for videos.} It might be possible to augment the decentralized planner with tie-break rules to avoid deadlock, but we have left this to future work.

## 5 RELATED WORK

**Connected Autonomous Vehicles.** Network connectivity in vehicles has opened up large avenues for research; we cover these briefly. A large body of work \cite{71} has explored wireless technologies and standards (such as DSRC) for vehicle-to-vehicle, and vehicle-to-infrastructure communication. Connected autonomous vehicles have also inspired proposals for cooperative perception \cite{60}, collaborative map updates \cite{13}, and cooperative driving \cite{16, 28, 48} in which autonomous vehicles share information with each other to improve safety and utilization. Some have proposed approaches to offload route planning (but not trajectory planning) to the cloud \cite{37}. Others explore is platooning \cite{42, 68} in which vehicles collaboratively and dynamically form platoons to enable smooth traffic flows. Beyond inter-vehicle collaboration, several proposals have explored infrastructure support for connected autonomous vehicles, with infrastructure augmenting perception \cite{36, 64}, or delivering traffic light status \cite{68}. Other work focuses on infrastructure-assisted traffic management at intersections \cite{43}. Net-driving goes beyond this body of work in proposing, and demonstrating the feasibility of, decoupling both perception and planning from vehicular control.

**Infrastructure LiDAR based Perception.** Prior work has explored using infrastructure LiDAR to detect pedestrians \cite{86}, and other road features such as lanes and drivable surfaces \cite{38, 81}, and to warn vehicles of impending collisions \cite{15, 75}. One work \cite{84} proposes a genetic algorithm based LiDAR alignment, but unlike Net-driving, has not explored the efficacy of an entire perception pipeline built on top of LiDAR fusion.

**Point Cloud Alignment.** Net-driving’s alignment builds upon point cloud registration techniques \cite{19, 25, 69}. Prior work has tried to match features \cite{39, 67} to align point clouds; these don’t work well for Net-driving, where LiDARs capture the scene from very different perspectives.

**Deep Neural Nets for 3D Detection and Tracking.** Some work has developed neural nets for point-cloud based detection \cite{24, 44, 79, 83, 87}, and tracking \cite{20, 35, 48, 49, 59, 72, 82}. These are too heavyweight for Net-driving, and have been developed for vehicle-mounted LiDAR (detection and tracking are easier for static LiDARs, as our evaluations show).

**Motion Estimation.** Heading and velocity can be estimated using deep neural networks \cite{23, 49}, SLAM \cite{31, 52, 53}, or visual odometry \cite{33, 66, 80}. Net-driving uses a lightweight technique since it relies on static LiDARs.
6 CONCLUSIONS

Net-driving decouples autonomous vehicle perception and planning from low-level control. By doing this, it can perceive more than a single vehicle can, and jointly plan for multiple vehicles. To demonstrate that it can do this, while meeting the performance requirements of existing autonomous vehicles, and without sacrificing accuracy, we develop a novel stack containing new perception algorithms and a fast planner. Jointly, these achieve the desired performance targets on commodity hardware, and have accuracy comparable to state-of-the-art algorithms. Net-driving enables higher safety than autonomous driving, and higher throughput traffic management at intersections compared to using traffic lights.

REFERENCES