Development and Testing of a Software Stack for an Autonomous Racing Vehicle

Andrew Saba

CMU-RI-TR-23-85

December 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

*Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Robotics.*

To my mom, who pushed me to be the best I can be.

Abstract

Autonomous racing aims to replicate the human racecar driver with software and sensors. As in traditional motorsports, Autonomous Racing Vehicles (ARVs) are pushed to their dynamic limits in multi-agent scenarios at high speeds ($\geq 100mph$). This Operational Design Domain (ODD) presents unique challenges across the autonomy stack. The Indy Autonomous Challenge (IAC) is an international competition aiming to advance autonomous vehicle development through ARV competitions. To compete in the IAC, we developed a full autonomy stack from scratch.

We present our design philosophy and strategy when developing the stack from scratch while under the constraints of the competition, including the ODD, rules, timeline, and limited field testing time. In particular, we present our contributions to the design, integration, and testing of the Perception and State Estimation systems and lessons learned from testing and deploying these systems on a real, full-sized system (the Dallara AV-21 platform). Finally, we demonstrate how our design process enabled rapid adaptation and development of the stack, which was shown capable of overtaking an opponent ARV at speeds exceeding 150mph.

Acknowledgments

I would like to first thank my committee members, Dr. John Dolan, Dr. Sebastian Scherer, Dr. Deva Ramanan, and Brady Moon. I am very thankful to you for all of your support in not only producing this thesis, but in your kindness, encouragement, and backing all of these years. Without Deva, I would not have been able to continuing working on the IAC. I also am thankful for everything you have taught me about how to present my ideas in a clear, concise way and capture your audience. I am also grateful for Basti, who supported a naive undergraduate student who was excited about robots but not sure where he wanted to do. Basti and Deva also supported me when things were rough, unwavering in your belief in what I was capable of, despite my own misgivings.

I would also like to thank all of my CMU friends. Being a part of the CMU robotics community and getting to grow and become the roboticist I am today has been the most rewarding experience of my life. I am especially thankful to my lab mates in the AirLab and Deva's Lab. I am especially grateful to Rachel Burcin, who never gave up on me, even when I wanted to give up on myself.

I would also like to thank all of my Pitt friends and advisors. IARC was the project that introduced me to real robotics and it is where I learned many of the skills that got me through my masters. Without Levi Burner, Aaron Miller, Ritesh Misra, Liam Berti, Andrew Lobos, Kaylene Stocking, Jay Maier, and many, many others, I would not be where I am today. I also am especially thankful to Dr. Samuel Dickerson, who pushed us to pursue the IAC in the first place and who was always my biggest cheerleader.

The IAC is not an easy competition to hold and run. Without the IAC organizers and the other teams, we would have been able to advance autonomous racing. I would like to thank Paul Mitchell, Gina O'Connell, Kris Kozak, Tyler Crane, Kevin, Andy Keats, and Janam Sanghavi from the IAC organization for all of your hardwork in hosting and supporting the competition. I also want to thank Lauren and Craig, who were our original "Racing Mom and Dad" and believed in everything our team and the competition stood for and supported us with all of your being. I also want to thank all of the other teams and the countless friends I made along the way, too many to list here, but no less significant.

I also want to thank our team sponsors. Without our great sponsors, our rag-tag group of students could not pursue this amazing project. I especially would like to thank Lynne Canavan and Neil Puthuff and the entire RTI company, who not only provided financial and technical support to our team, but have become friends and mentors that I can always talk to for help and support.

The last three years could not have happened without the entire MIT-Pitt-RW team, especially Josh Spisak, Nayana Survana, and Prem Bharatia, who co-founded the team with me, and Cindy Heredia, Andrew Tresansky, Dan Reilly, and Chris Chang, who brought the MIT and Pitt teams together, and Andy Keats and Ben Zhang and Brian Mao, who rounded out our MPRW family. I also want to thank Adam Johnson, Matthew Sivaprakasam, Arjun Chauhan, Austin Brown, and Noah Gasparro for all of the work we have done together to make our incredible perception stack. I also would like to thank the OG Monza testing squad, Charlie King, Connor Kirby, Jatin Mehta, Ritesh Misra, and Miriam Dukaye. It was a joy field testing in Italy with you. I also want to thank Micah Nye, who has been the best protege I could have ever asked for, constantly eager to learn and grow, always giving his everything for the team. Finally, I want to thank Rotimi Adetunji, for being the best co-captain she can absolutely be, putting in 1000% all the time, and never giving up, no matter what. It has been the highest honor and privilege to work alongside you and seeing how this team has flourished and grown under your guidance.

Finally, I want to thank my family, my dad Shehab, my brother Anthony and his husband, Ed, and my sister Nicole and her husband, Matt. Without your support, I could not have accomplished what I have done. I am especially grateful to Nicole and Matt and everything they have done to support me over my masters and getting to this point today. Finally, I want to thank my mom, Nellie, who taught me to always work my hardest, putting out my best work and never doing anything halfway. She also taught me the importance of kindness and compassion, making sure I always gave back to others. I know she is looking down on me, proud of everything I have accomplished.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

Historically, motorsports have been a venue for advancing automotive technology in the name of competition and brand recognition. Racing teams develop increasingly sophisticated technologies to shave off seconds from lap times. Over time, technology and lessons learned from car racing have been commercialized and adopted in standard passenger vehicles. With the advent of Autonomous Vehicle (AV) technology, motorsports are poised to play a similar role in the development of autonomous systems. Autonomous Racing Vehicle (ARV) leagues, such as the Indy Autonomous Challenge (IAC) and others, are challenging software, not drivers, to operate a vehicle at the performance limit. Autonomous racing provides an operational design domain (ODD) that challenges conventional algorithms and systems.

In this work, we examine these challenges and present a design methodology for developing an autonomy stack to address them. We then apply this methodology specifically to the ego state estimation (i.e. knowing where the agent is in the world) and opponent state estimation (i.e. detecting and estimating the state of other agents) software modules. Finally, we present results and lessons learned from deploying these modules, within a larger software stack, for the Indy Autonomous Challenge (IAC) competition. All data and results are taken from competition runs during the 2022-2023 season of the IAC, where our ARV was capable of passing another ARC at speeds in excess of 150 mph, achieving fourth place out of nine teams.

1.1 Autonomous Racing

Formally, autonomous racing is when an autonomous agent's (ground vehicles, drones, etc.) primary purpose is to compete directly against other agents (human[19][57] or other autonomous agents) in racing events, with the goal being to replicate human racing ability. An autonomous racing vehicle (ARV) is a ground-based racing vehicle, modified with sensors and drive-by-wire systems, and driven by software. The Dallara AV21[38] is one such ARV, as it is a full-sized racecar specially built to race at high speeds, autonomously. Smaller ARVs exist, such as in F1/10[13].

ARVs have a different objective than conventional autonomous vehicles (AV). An AV company may want to prioritize rider comfort, safety, and "human-esque" driving ability. Formally, this means that the AV does not act in a manner that is strange or unusual to what human drivers would consider normal. Put another way, an AV motion-control stack may have an objective like the following, presented as an optimal control problem, where we are minimizing a cost function J, subject to a dynamics function $x_{n+1} = f(x_n, u_n)$ and constraints C:

$$\begin{aligned} \min_{x,u} J = J_{comfort} + J_{smooth} + J_{destination} + J_{humanness} \\ \text{subject to } x_{n+1} = f(x_n, u_n), \\ C_{safety}, C_{rules} \end{aligned} \tag{1.1}$$

The optimization problem in equation 1.1 is minimizing the sum of four cost functions J that penalize uncomfortable actions (i.e. sudden braking), encourage smooth actions (i.e smooth steering inputs), getting to the desired destination, and penalizes behavior that is not human-like. Finally, we want to constrain the optimization against unsafe actions C_{safety} or actions that violate the rules of the road C_{rules}.

For autonomous racing, the objective looks very differently. Firstly, in racing, safety and avoiding a DNF (did not finish) by not crashing are two very different things. While driving on the road, drivers are usually careful to not risk the safety of others by engaging in potentially dangerous maneuvers, such as passing in front of someone very suddenly, driving very close, and more. However, in racing, as long as someone is being sportsman-like and is obeying the rules of the race, these behaviors are acceptable, as long as a crash does not occur. As such, cost objectives for racing may look like the following:

$$\begin{aligned} \min_{x,u} J = J_{time} + J_{penalties} + J_{passing} + J_{risk} \\ \text{subject to } x_{n+1} = f(x_n, u_n), \\ C_{collisions}, C_{rules} \end{aligned}$$

Where J_{time} penalizes slower driving, $J_{penalties}$ penalizes committing infractions that result in time penalties, $J_{passing}$ encourages passing, and J_{risk} encourages the vehicle to take some measured risks. This objective function would encourage an autonomous agent to avoid collisions, adhere to rules, but try to pass other agents and go as fast as possible and explore the state and action space to get closer to the true limits of the vehicle.

1.2 Period vs Latency

Two concepts, period and latency, are often misconstrued for one another, so it is important to first define and clarify what each means and why they are important concepts. Period is defined as the time between outputs of a pipeline. In the context of estimation, for example, this is the time between measurements. Latency refers to time between an input being received and when an output is produced. For example, this can be considered processing, in addition to the time to transport the data from one process to another. Latency and period are depicted pictorially in Figure 1.1.

In the figure is a toy timeline, which depicts the LiDAR scan to action timeline, with time snapshots that are not necessarily indicative of the real system. The timeline starts with the very first LiDAR point being scanned. It takes approximately

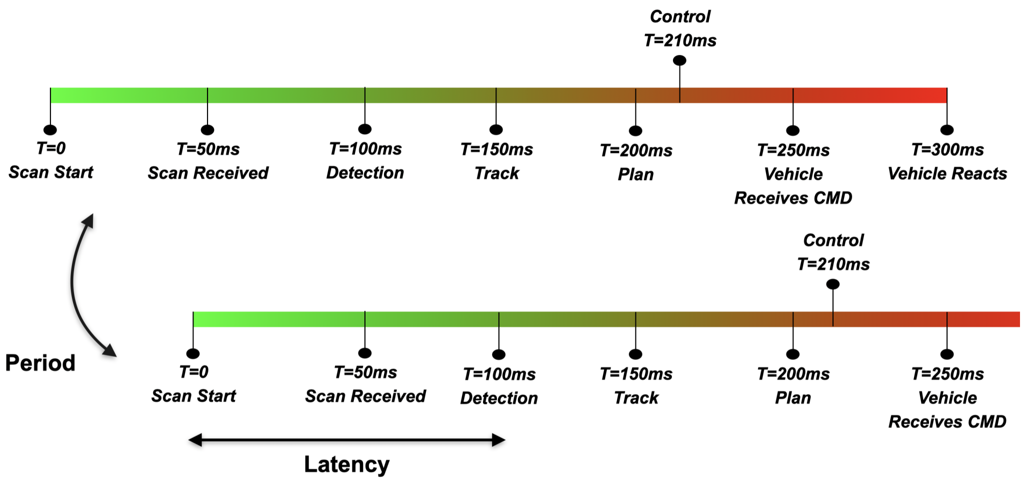


Figure 1.1: Visual depiction of latency vs period. The timeline depicts a toy LiDAR scan to action timeline. The pipeline starts with the very first LiDAR point being scanned. It takes approximately $50ms$ for the full scan to be completed, processed, and passed along to the computer. Finally, a detection is produced another $50ms$ later. As a result, it can be said that detections have a latency of up to $100ms$, since it takes about that long to take a measurement and produce a detection. While the first scan is being processed, a second scan begins. This scan is also processed after about $50ms$ and a detection is produced another $50ms$ later. This means that the time between detections, or the period, is about $100ms$.

$50ms$ for the full scan to be completed, processed, and passed along to the computer. Finally, a detection is produced another $50ms$ later. As a result, it can be said that detections have a latency of up to $100ms$, since it takes about that long to take a measurement and produce a detection. While the first scan is being processed, a second scan begins. This scan is also processed after about $50ms$ and a detection is produced another $50ms$ later. This means that the time between detections, or the period, is about $100ms$.

For autonomous racing, both latency and period are important. Lower latency improves the reaction time of the system, which is critical due to the high accelerations and speeds that can be achieved by ARVs. Lower period means that more frequent updates of the world is provided, which reduces the need on extrapolation or prediction, thereby allowing for more accurate results. This is especially important because ARVs are capable of accelerating rapidly and producing hard to predict, adversarial maneuvers.

1.3 Indy Autonomous Challenge

The Indy Autonomous Challenge (IAC) was started to foster autonomous vehicle (AV) development through racing. From the competition website [37], the goal of the IAC is to "advance technology that can speed the commercialization of fully autonomous vehicles and deployments of advanced driver-assistance systems (ADAS)". Additionally, the organizers[37] are seeking to address three main barriers to AV commercialization:

1. Solving edge cases, which are difficult to handle or challenging scenarios that are not typically seen. For example, an agent in the world behaving in unexpected, uncommon ways.
2. Fostering new technologies and the next generation innovators
3. Public engagement and increase wide-spread acceptance of AV technology

The IAC addresses these barriers by hosting competitions for nine international university teams, who develop software for a standardized ARV platform named the Dallara AV-21. These competitions have involved both single-agent, fastest lap style races, and events where two ARVs race against one another in a head to head competition. Table 1.1 presents the previous events and the competition formats.

1.3.1 Competition Rule-Sets

There have been primarily two race formats: fastest laps and the passing competition. The first event in Indianapolis took the average speed across two fast laps and had a third lap with static obstacles to avoid. The most recent event in Monza was based on the fastest time from three 15 minute runs. Finally, the passing competition was created to encourage high-speed overtaking and avoidance between two ARVs. Events in seasons one and two were held on oval super-speedways and season three has so far been held on Monza, a road circuit. The track type plays a critical part in dictating the strategies for stack development. Before the in-person installments, there were multiple simulation practice events and a simulation competition, where the teams

Indy Autonomous Challenge Events	
Season One (2021-22)	
Track	**Event Format**
Indianapolis Motor Speedway (IMS)	Fastest Lap with static avoidance
Las Vegas Motor Speedway (LVMS)	Passing Competition
Season Two (2022-23)	
Texas Motor Speedway (TMS)	Passing Competition, relaxed racing lines
Las Vegas Motor Speedway (LVMS)	Passing Competition, relaxed racing lines
Season Three (2023-24)	
Autodromo Nazionale Monza (Monza)	Fastest Lap

Table 1.1: The Indy Autonomous Challenge (IAC) has held five events thus far, at four different tracks, with three formats. Season Two saw a continuation of the passing competition introduced at the AC@CES2022 installment in Las Vegas. For Season Two, the defender is given more freedom on the racing line they may take, which makes passing more challenging.

verified their software in single and multi-agent scenarios. All instalments of the competition have been supervised and executed by ”race control”, managed by the IAC. During competition, the race flags and team roles are remotely controlled by race control, allowing for minimal human intervention during the race.

1.3.2 The Passing Competition

The multi-agent passing competition [36] assigns one competitor the ”attacker” role and the other the ”defender” role. During each lap, the defender is remotely assigned a speed to hold while the attacker must pass the defender within two laps. If the pass is successful, the roles are exchanged and the defender's speed is incrementally increased ($125mph$, $135mph$, etc.). A pass is complete once the attacker gains its position in front of the defender with a longitudinal gap of at least $30m$. If an attacker fails to pass at a certain speed, the roles are exchanged. The winner of the round is determined once one of the attackers cannot complete a pass. If both teams cannot complete the pass, the round ends in a draw. Figure 1.2 shows a breakdown of the track and the possible paths to take into consideration.

There are two factors the attacker must consider while deciding to make a pass: safety and dynamic limitations. The attacker must maintain safe lateral and

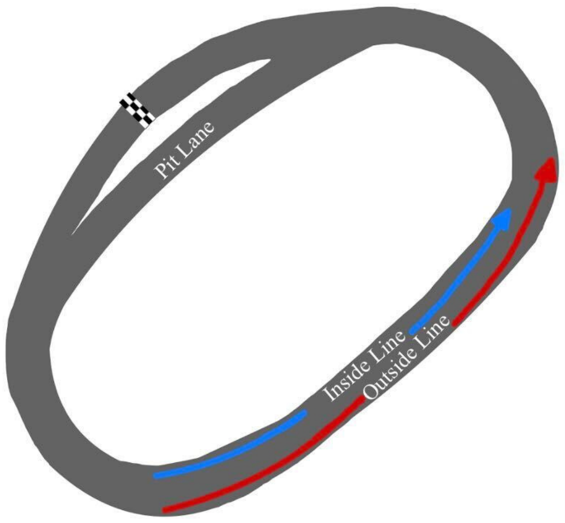


Figure 1.2: Track lanes for the Passing Competition, held at the Las Vegas Motor Speedway in January 2022 and 2023.

longitudinal separation from the defender at all times. Additionally, when considering a pass, the attacker must ensure its trajectory will keep the car within the dynamic limitations of the vehicle and not result in loss of control. Combining the two, the attacker must ensure that the accelerations and decelerations are timed appropriately to stay within the dynamic limits of the vehicle. As the passing competition progresses, it becomes more difficult for the attacker to exceed the defender's speed, particularly in corners of the track.

In addition to these considerations for the attacker, the defender can make passing more difficult for the attacker by adjusting their position within their lane, as long as they maintain rule compliance. For example, if the defender moves outwards, the attacker has to travel more distance to complete the pass. A winning strategy for an attacker is to maintain the minimum allowed distance to the defender and to initiate the pass whenever the attacker can maneuver it safely. Completing a pass is also further complicated if the attacker starts the pass too late; they may get trapped too far out on the outer lane into the corners, thereby increasing the distance they need to cover. Prediction and motion forecasting of the opponent agent is imperative to make intelligent strategic decisions.

1.3.3 AV-21 platform

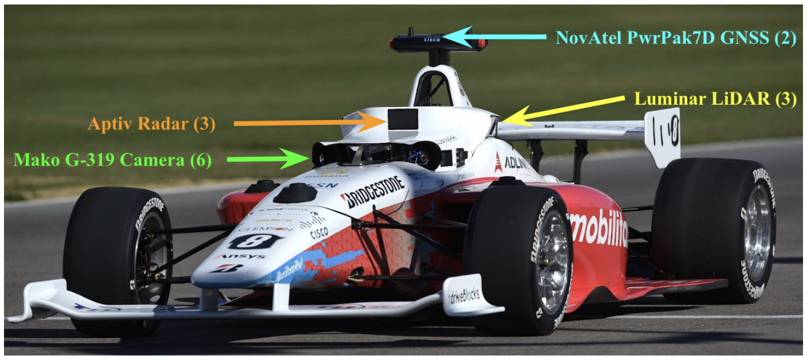
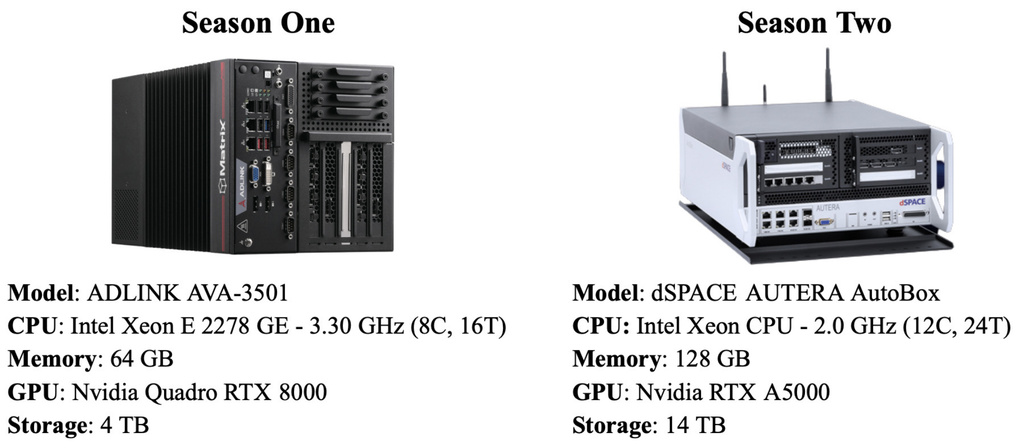


Figure 1.3: Sensors on the AV-21. Six cameras and three LiDARs provide redundant 360° coverage and over 200m of sensing range.

The Dallara AV-21 is the official vehicle of the Indy Autonomous Challenge (IAC). Every competitor must use the same hardware, including vehicle setup, autonomy sensors, and compute. The vehicle is a modified version of the Indy Lights IL-15 chassis, retrofitted with a package of automated vehicle sensors, drive by wire, and compute. The engine is a 4 Piston Racing-built Honda K20C. Sensors onboard the AV-21 include 3 Luminar Hydra LiDARs, 3 Aptiv Medium Range Radars, 2 NovAtel PwrPak7D-E1 GNSS, and 6 Mako G-319 Cameras. In total, these sensors provide redundant 360° coverage and over 200m of sensing range. Figure 1.3 shows the AV-21 platform and sensor locations.

Between Seasons One and Two of the IAC, the AV-21 underwent a hardware refresh that included the addition of a VN-310 Vectornav GNSS system and an update to the main compute platform. In Season One, the main compute was an ADLINK AVA-3501 with an 8 core, 16 thread Intel Xeon CPU and an NVIDIA Quadro RTX 8000 GPU. In Season Two and Three, a dSPACE AUTERA AutoBox with a 12 core, 24 thread Intel Xeon CPU and an RTX A5000 NVIDIA GPU served as the main compute platform. The AutoBox provided many advantages over the ADLINK, including automotive-grade ruggedness, higher available networking bandwidth, and CAN channels built into the computer. While the AutoBox provides automotive-level reliability and integration, its slower per core clock speeds resulted in worse single threaded performance. Since many critical core algorithms are fundamentally



Season One

Model: ADLINK AVA-3501
CPU: Intel Xeon E 2278 GE - 3.30 GHz (8C, 16T)
Memory: 64 GB
GPU: Nvidia Quadro RTX 8000
Storage: 4 TB

Season Two

Model: dSPACE AUTERA AutoBox
CPU: Intel Xeon CPU - 2.0 GHz (12C, 24T)
Memory: 128 GB
GPU: Nvidia RTX A5000
Storage: 14 TB

Figure 1.4: Main compute platforms. Specifications and images taken from respective product websites[2][10].

single threaded (i.e. controller calculations, state estimation updates, etc.), careful considerations were made into what ultimately ran on vehicle to ensure enough capacity for the entire stack. A full breakdown of the compute platform differences can be seen in Figure 1.4.

1.4 Our Approach

Our software architecture follows a typical, standard autonomy software design, with localization, perception, tracking, prediction and motion planning, and controls. The Robot Operating System (ROS), specifically ROS 2 Galactic, is used for communication between each process, or node. Various libraries and frameworks are utilized from ROS for visualization, math utilities, communication, and more. Figure 1.5 shows the data flow of the whole stack. All modules run asynchronously with one another, usually on a preset frequency, except for perception and portions of localization, which are driven by sensor data arrival.

The algorithms chosen are all standard and well-proven. For example, the lateral controller is an LQR solver with pure-pursuit. This is a function of the limited tested time, the competition ODD, and compute limitations. Through the process of designing and integrating this stack, a design methodology naturally emerged, that guided our process. In this work, we will present this methodology (see Ch. 2) and

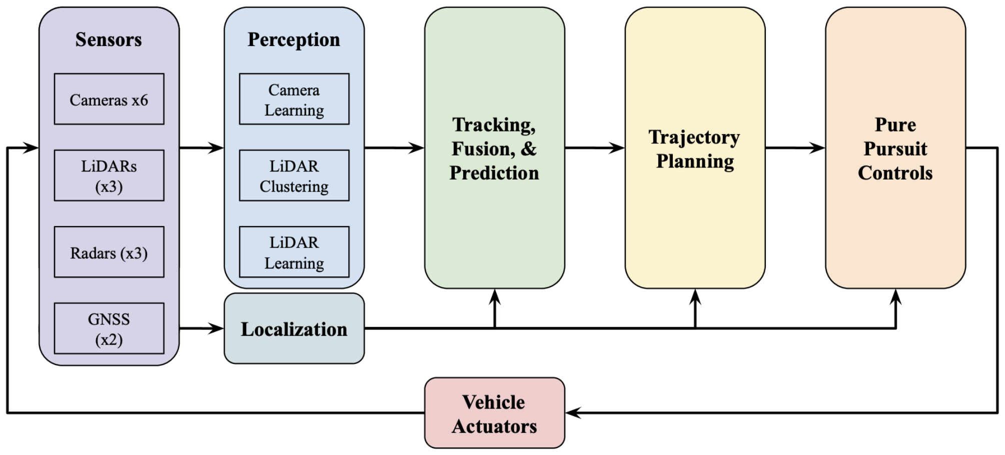


Figure 1.5: Overview of the full software stack.

show its application to the opponent agent estimation modules (i.e. Perception and Tracking and Fusion in Figure 1.5). When our team did not follow a methodology, the result was an incoherent set of software modules that were haphazardly developed and that did not meet our needs. By using the methodology, we focused our development on what was absolutely necessary and ended up with a system capable of meeting the competition demands and able to pass another ARV while driving at over 150mph.

1.5 Related work

The call for advancing autonomous vehicle (AV) technology has been present since the early twenty-first century when the Defense Advanced Research Projects Agency (DARPA) launched the 2004 and 2005 DARPA Grand Challenges [4], [6]. These challenges, shown in Figure 1.6, demonstrated some of the capabilities of AVs. The teams in the Grand Challenge autonomously navigated across southern Nevada on a 132-mile course of rugged desert terrain. Succeeding the Grand Challenges was the 2007 DARPA Urban Challenge [7], which introduced a time-based competition focused on city driving. This competition maintained the competitive nature of completing a course, but focused on navigating an urban environment. Each team needed to stop at stop signs, yield for oncoming traffic, complete U-turns, and obey all other traffic laws. These challenges were the first full-scale autonomous racing competitions and laid the groundwork for future AV research and development.

Figure 1.6: DARPA Grand and Urban Challenges, two prize competitions for autonomous vehicles. From [1][39].

Since then, there have been several autonomous racing competitions, such as Formula Student [17], [52], Roborace [44], [5], and now the Indy Autonomous Challenge. Moreover, companies such as Argo AI, Motional, Waymo, and many more have been publicizing and realizing AV development around the globe. These companies have tasked themselves with challenges to motion plan in dynamic, unpredictable environments and perceive in inclement conditions. The challenges ARVs face differ, focusing on detecting vehicles, planning motion, and actuation while driving at speeds over 150mph. Unlike AVs that operate in an open world, ARVs do not need to worry about cyclists on the road [3] or pedestrians crossing a street; however, the issue of making quick and accurate detections and actions remains a nontrivial problem still under research.

1.5.1 Perception for Autonomous Racing Vehicles

Although there is substantial work demonstrating perception in conventional AVs, less work has been published discussing the deployment of perception algorithms for autonomous racing. In Formula Student Driverless, the ARV drives on the track alone and is solely tasked with detecting white and blue cones [17], [52]. Currently, there are few works that present full perception stacks for detecting other agents for autonomous racing. One such work is [55], which presents a full perception stack that utilizes camera, radar, and LiDAR sensors. Improving perception efficiency is an extensively researched topic in AV development. The point-cloud clustering-based detection system in [51] is fast and efficient at detecting other actors. Works such as [23] and [59] focus on Streaming Perception, which emphasizes combining latency and accuracy when developing benchmarks for computer vision algorithms. Multi-modal perception is also a very well-studied area of research. The work in [25] fuses LiDAR and camera features with a learned cross-modal attention alignment. In [27], a combined LiDAR-camera birds-eye-view (BEV) projection is generated efficiently and can be used for downstream tasks such as object detection. In both works, multiple sensor modalities are fused early in the detection pipeline.

1.5.2 Motion Planning for Autonomous Racing Vehicles

Motion planning involves determining the best sequence of actions to be taken and generating a trajectory to execute those actions. For oval racing, it is possible to distill the problem in a series of action primitives, including maintaining the current trajectory behind an opponent, e.g., ”trailing” or passing. One way to approach the passing problem is to treat it as a sequence of lane changes, where the ego vehicle merges between several lanes of travel on the track. Lane merging is a well-studied area, with previous work utilizing polynomials [26], splines [47], [14], or Bézier curves to parameterize paths [62]. In [50], 5^{th} order polynomials are generated within a Darboux frame using a convex combination of the origin and target paths. Heading and curvature continuity is guaranteed for any lane change maneuver without numerical differentiation. In addition, compliance with track boundaries is also guaranteed a *priori*.

1.5.3 Motion Control for Autonomous Racing Vehicles

While AVs cannot yet legally drive faster than 80 to $130mph$, depending on locale and road conditions [9], the operating domain for ARVs is typically 100 to $200 + mph$. This sizeable difference dictates the differences in vehicle architectures, modeling, and controller techniques. Additionally, at higher speeds, assumptions made in vehicle dynamics models may not apply, necessitating better vehicle modeling. Some works have explored addressing this problem by combining a model predictive controller (MPC) with a deep-learning-based model [21], [60]. Other works, such as [15], look to estimate tire friction parameters online for use in an MPC controller. Finally, due to model limitations at higher speeds, the controller must be robust and capable of reasoning about potential bounds on actual dynamics. Works such as [56] address this with a Tube-MPC controller that can reason about uncertainty in the dynamics. Additional work, such as [42], layout a whole navigation stack for use in an ARV. Finally, for our approach, we looked to robust and fast controllers, such as linear-quadratic regulators (LQR) and iterative LQR control [29], [8].

Chapter 2

Design Methodology

Designing a whole software stack from scratch for a new competition, on a new vehicle capable of reaching speeds of over 180mph, in less than six months is a daunting task. However, the pressures of the competition and our operational design domain did not just challenge our software stack, but also the way we approached designing and devloping the autonomy system. A design methodology emerged that enabled our eventual success, after some initial failures.

2.1 Motivation

In the process of developing our software stack over two years, we recognized a marked shift in our success once we started following to our design methodology. For example, our the development of our perception stack (i.e. understanding the state of the world around us, including where other agents are) was haphazard and struggling. Algorithms were implemented and tested that were unfeasible due to computational constraints, the lack of data, or were not absolutely necessary for the competition.

By January 2022, when qualifying for the first IAC@CES event (see Table 1.1 for all events), the perception stack was barely functioning offline and was not capable of working well on vehicle. Figure 2.1 depicts the period of the object detection stack in January 2022 and January 2023. The time between each detection was on average over 200ms, often reaching over a second. It is impossible to have a functioning autonomy stack when detections of other vehicles are only produced every second

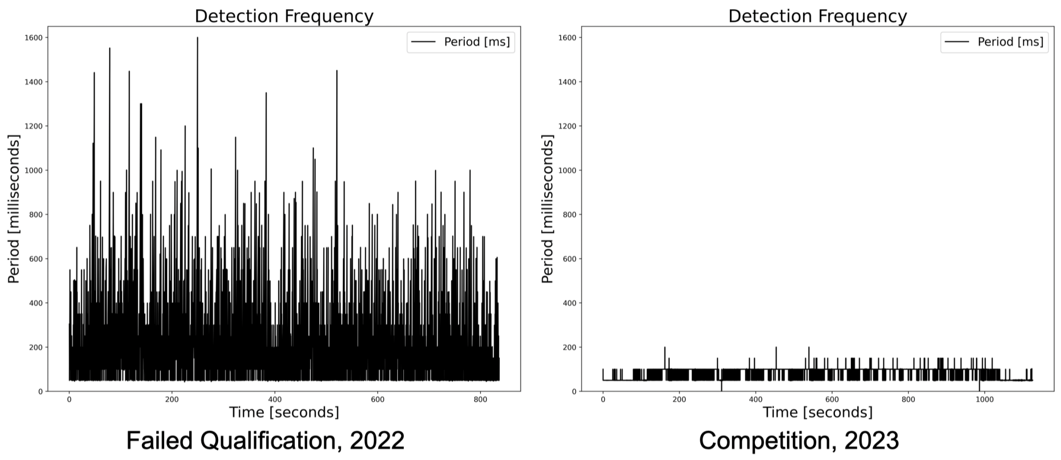


Figure 2.1: Period of the object detection stack in January 2022 (left) and January 2023 (right). Period was calculated by taking the time difference between subsequent detections. Part of the reason we failed to qualify in 2022 for the first passing competition event was due to a breakdown in the perception and tracking stacks. Fixing these issues over the ensuing year yielded invaluable experiences and learning's for system design, integration, and testing that motivated this thesis.

or only at most four times a second. As a result, we did not meet qualification requirements to compete in the passing competition. One year later, the stack was improved and able to reliably produce outputs 10-20 times per second, allowing the ARV to overtake an opponent ARV traveling over 125mph, reaching over 150mph in the process.

These successes are attributable to following a more disciplined design methodology when developing the software stack. This methodology came out of the failures, refined over time with experience.

2.2 The Four-Step Methodology

Our design methodology focuses on the following four steps:

1. Formalize the operational design domain (ODD)
2. Define the hardware constraints
3. Enumerate algorithmic and system approaches

4. Construct a test plan

We found that by following this process, in a continuous fashion, we were able to better define system requirements, narrow the space of feasible approaches, and be more proactive in creating a system that can be effectively tested. The next few sections will detail these steps in detail. Finally, subsequent chapters will present the application of this methodology to the perception subsystem and show how better development outcomes were realized.

2.3 The Operational Design Domain

An operational design domain (ODD) is the set of conditions under which the system is expected to operate. For example, one potential, high level ODD for the Indy Autonomous Challenge (IAC), passing competition, can be defined as:

- Drive no slower 80mph on an oval super-speedway
- Pass another ARV that is driving no slower than 80mph
- Maintain a safe distance to the opponent ARV when not passing
- Maintain a raceline and set speed when acting as Defender

It is important to define the ODD because it explicitly lays out the requirements of the system. This is useful for preventing “feature creep,” which is when a system has too many unnecessary features, and for making critical decisions about the space of feasible approaches. For example, in the passing competition, there is only one other vehicle at a time, so this allows simplifications in the approach that would not otherwise be possible. Sometimes, this can hurt long-term development, especially if the solutions are too over-fit to the short-term requirements. As a result, important considerations and thought should be put in balancing the current ODD and how the ODD is expected to evolve over time and prioritize effort accordingly. In our case, we recognized that someday we will be expected to race against more than one other ARV, so the approaches taken are easily extended to handling more than one opponent at a time.

Additionally, having an ODD also defines what the system must be capable of achieving, which is useful for test planning. For example, in the passing competition, you must drive at least 80mph, so a test can be constructed to verify that this requirement is met.

2.4 Define the Hardware Constraints

Defining the hardware constraints involves understanding the target hardware platform and how to best exploit it. For example, in the IAC, teams are provided redundant 360° sensor coverage, with multiple GNSS units, LiDARs, cameras, and radars (see Section 1.3.3 for more details). Teams are allowed to utilize how ever many or few of these sensors as they wish. For example, TII Unimore Racing uses all of the available sensors in their perception and localization pipeline, including utilizing a GPS-denied localization pipeline for redundancy [43]. Ultimately, the space of possible solutions are constrained by, but not limited to:

- Time synchronization (clock sync, synchronized triggering)
- Software driver and firmware versions and compatibility
- Sensor quality (noise), update rate, range, resolution, etc.
- CPU, GPU, memory, etc. features and limitations
- Memory, storage, communication bus (CAN, USB, wired and wireless networking, etc.), etc. bandwidth
- Actuators and power unit characteristics
- Mechanical constraints
- Power consumption

Time synchronization, in particular, impacts the performance of perception algorithms. For example, early-fusion works such as [25] develop a means to align sensor features to improve object detection. However, it, and any method using sensor fusion, rely on having synchronized sensor triggers, or measurements that can be aligned through interpolation (i.e. motion compensation and deskewing with LiDAR pointclouds). However, if the sensors are not on the same clocks or triggered in sync, then temporal alignment is impossible. In our experience, time synchronization is by far the most common issue seen with robotic systems, with sensor driver issues typically coming second.

Heterogeneous systems, or systems with a CPU and a dedicated accelerator, including GPUs, FPGAs, etc., are increasingly popular and popular robotics frameworks (i.e. the Robot Operating System (ROS) 2) are creating tools to better exploit such systems [32]. The basic idea is that some workloads are better suited to dedicated hardware or GPUs versus CPUs. Recent work [49], however, has shown that vector instructions, when used properly, can also speed up computation on a CPU, negating the need for a GPU. In [49] case, the overhead of copying back and forth to a GPU could easily negate the parallelization gains, but, by utilizing vector instructions on a CPU, no device to device copying had to occur and the parallelization gains could be made, which was particularly useful for operations happening many thousands or millions of times a second. Regardless, the trend has been to offload many tasks, from perception to even controls and planning [54], to the GPU and other dedicated hardware.

Ultimately, it is important to go back to the hardware constraints when making decisions on potential approaches. Will these collision checks our planner is conducting be able to run fast enough on a single CPU core or should they be parallelized? Is there enough data bandwidth to save all the camera images at full rate to disk? How much telemetry can be sent back to the remote operator? How fast does our controller need to run to be able to react fast enough to disturbances? Oftentimes, the biggest advancements come from trying to overcome these limitations.

2.5 Enumerate Approaches

Historically, roboticists have come to think of possible autonomy approaches on a spectrum- how much learning is incorporated in a system. “Traditional” approaches have come to mean ones that have separate modules for planning, controls, perception, localization, etc. and “modern” approaches rely on machine learning and large datasets to replace rules-driven algorithms. However, in practice, we have found that it is not helpful nor possible to think of approaches on a single axis scale. Rather, we have found it most helpful to classify the state of approaches on a two axis spectrum: how coupled the modules within the architecture are and how data driven the algorithms used are.

2.5.1 Modularity

Modularity refers to the dependence of one module on another module's outputs, assumptions, and intricacies. For example, a typical perception pipeline might have an object detector and a tracker. The detector outputs candidate objects with some class and three-dimensional pose. The particular track chosen might be setup in a way where it can accept objects with this interface, but also needs velocity information. The system designer either needs to estimate velocity from another source, change the tracker, or change the object detector. There may be more subtle dependencies. For example, in our stack, a bug occurred because the planner was expecting tracks to be ordered by relative distance, but were actually ordered by age, so the planner made assumptions that later broke.

Modularity can be thought of from an information theory perspective. Sensors provide high resolution, noisy, and high dimensional temporal measurements of the state of the world and the ego agent state. Turning this information into concrete actions and decisions requires digesting and transforming it through multiple modules, or a single module for end-to-end approaches. The assumptions, dependencies, and information shared between modules ultimately makes up how modular an approach is.

It is important to consider Modularity when designing a system because it impacts performance, ease of testing, and code reuse. For example, when modules are decoupled, it makes it easier to test them individually with integration tests. However, in the tracker example, having velocity information may make the tracker more accurate, but it requires a more sophisticated object detection module.

2.5.2 Data Driven vs Rules Based

Modern robotics development has been dominated by machine learning, particularly deep neural networks that allow for learning from enormous amounts of data. Rules based refers to methods where engineers endow the algorithm with some prior set of rules or heuristics to guide decision making. For example, a reinforcement learning algorithm will learn how to make a safe merge onto a highway, trained on thousands or even millions of merges. An optimal control algorithm uses human-engineered cost functions, constraints, and heuristics to optimize for the merging behavior. However, it is much harder to certify that the learned algorithm is safe because it is not clear how it resulted in its decisions or actions. This problem does not exist with the rules-based approach.

The problem, however, is that rules-based approaches have their limitations. It is impossible to tune for every possible case an autonomous agent may experience, but, with enough data, the odds are higher that it has seen a similar scenario before. The caveat is having enough data to teach the agent the necessary skills. When developing a system, data is not always immediately available or easy to collect, so rules-based approaches provide a proven solution to get working.

2.5.3 Summary

To summarize, important questions to ask include:

- How easily can the ODD be addressed with hand-engineered rules?
- How readily available is quality data?
- Does the code need to be reused for other ODDs?
- How important is interpretability and the ability to isolate modules?
- Can you trade-off robustness for peak performance? Or vice versa?

The answers to these questions will clarify the feasibility of an approach and help decide on if a data driven or rules based approach is better and if the system needs to be modular or if assumptions can be made.

2.6 Test Planning

Real-world testing is expensive. For example, in the case of the IAC, to rent a single day on the Las Vegas Motor Speedway cost over $50k. Additionally, crashes can cost just as much, or more. Logistics for moving, maintaining, and operating autonomous racing vehicles can be in the multiple tens or even hundreds of thousands of dollars over a year. While the field of autonomous racing has this unique challenge, there are logistical and other practical considerations that go into real world testing for other systems.

It is important to develop a plan for testing a systems because these challenges will emerge and, if there is no way around it, then a different approach may need to be taken. For example, when we were designing our initial object detection pipeline, real-world data did not exist and the AV21s were not yet capable of sharing a track together to collect data. As a result, we developed a clustering algorithm that was easy to test in the garage. At the same time, we developed a simulation environment that allowed us to collect ground truth labels very easily and train our initial data-driven approach. However, with the clustering algorithm, we were able to safely run on track, albeit with an inferior solution, and collect more data to refine our model.

Test planning should be continuous, meaning that, as approaches develop and change, the tests need to evolve alongside it. It is also not always clear what should be tested and whether that testing should occur offline (i.e. in development versus on robot). Offline tests can include using simulation, prior data sets, and unit and integration tests. If it is possible to replicate the real system conditions as closely as possible in simulation testing, then that should be used. However, this is not always possible, so careful consideration must be applied to ensure that when code is tested on the real system, the results are close to what is expected.

It is also important to define what is considered a successful test and what outcomes are expected. Defining the metrics, in concert with examining the ODD, allows a clearer picture of where compromises on performance may be made. For example, our LiDAR object detection pipeline does not do motion compensation, as it did not meaningfully affect the accuracy in a way that negated the processing time speedup. Additional discussions on this, and other points, are presented in Chapter 3.

Chapter 3

Opponent Agent State Estimation

In this chapter, we will apply the design methodology presented in Chapter 2 to the problem of opponent agent state estimation. This problem entails detecting, tracking, and estimating the state of opponent autonomous racing vehicles (ARVs). Having applied the design methodology, we will then detail our approach for object detection and tracking and estimation of agent states.

3.1 Applying the Design Methodology

3.1.1 The Operational Design Domain

The passing competition (see Section 1.3.2) involves only two ARVs on track at a time, passing one another at successively higher speeds. Initially, the track can be assumed to be an oval super-speedway, an assumption that will not hold in future events. Additionally, there are no other vehicles or objects on the track besides the two ARVs that are to be detected. All interactions between the two vehicles will take place on the track surface, not in the pit lane or any other places. Finally, the track boundaries can be mapped and known a-priori.

The competition's minimum speed is 80mph and can reach over 180mph[35]. The AV21 is capable of very high accelerations (greater than $20m/s^2$) and speeds (greater than $180mph$), meaning LiDAR or camera frame-to-frame movement can be significant. Additionally, when the perception pipelines were first developed, there were no existing data sets for detecting AV21s and little data showing the performance of sensors, such as LiDARs and cameras, at our target speeds. Finally, all ARVs participating in the competition have identical hardware.

3.1.2 Hardware Constraints

More details on the AV21 hardware can be found in Section 1.3.3. Some details are copied below for convenience. Onboard the AV21 are six cameras, three LiDARs, and three radars, in addition to three GNSS systems, each connected to two antennas and receiving RTK corrections.

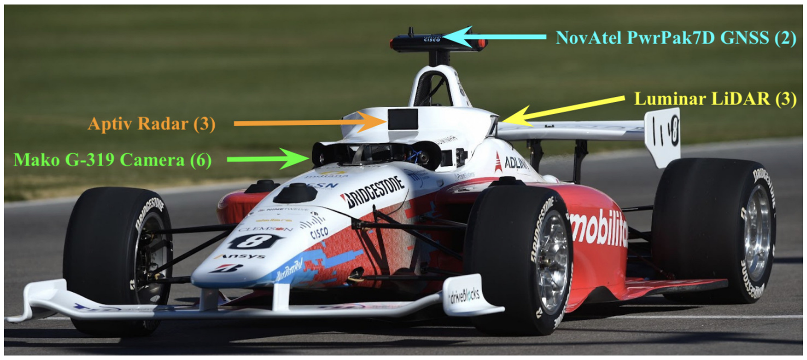


Figure 3.1: Sensors on the AV-21. Six cameras and three LiDARs provide redundant 360° coverage and over $200m$ of sensing range.

The onboard compute is limited to a single computer, a dSPACE AUTERA AutoBox with a 12 core, 24 thread Intel Xeon CPU and an RTX A5000 NVIDIA GPU. All sensors, except for the RADARs, are connected via standard networking. A Cisco IE5000 industrial switch acts as the main hub, connecting the AutoBox to all of the sensors via two 10gigabit SFP+ connections.

Unfortunately, the onboard CPU is very slow when compared to more recent hardware. Benchmarks put the CPU on par with a fourth generation Intel i7 desktop processor, which is orders of magnitude slower than the latest twelfth or thirteenth generation processors available today. The GPU, however, performs very well, being from the latest generation of NVIDIA GPUs and having a lot of potential for processing large amounts of data in parallel. However, the motherboard and CPU are limited to PCIE Gen 3, which can limit the maximum memory bandwidth available.

Season One

Model: ADLINK AVA-3501
CPU: Intel Xeon E 2278 GE - 3.30 GHz (8C, 16T)
Memory: 64 GB
GPU: Nvidia Quadro RTX 8000
Storage: 4 TB

Season Two

Model: dSPACE AUTERA AutoBox
CPU: Intel Xeon CPU - 2.0 GHz (12C, 24T)
Memory: 128 GB
GPU: Nvidia RTX A5000
Storage: 14 TB

Figure 3.2: Main compute platforms. Specifications and images taken from respective product websites[2][10].

Additionally, the slower single core CPU performance has also shown itself when copying data to and from the GPU. For example, a bug in our perception pipeline caused an up to 10-20% slowdown due to extra data copying.

3.1.3 Enumerate Potential Approaches

There are numerous potential approaches that could have been taken. With three sensor modalities, there are early fusion (i.e. combining the raw sensor data) and late fusion approaches (i.e. fusing the localized detections). Additionally, an approach could have been taken that does not utilize a tracker at all and does object detection and tracking in one. Finally, rules based approaches could be used for all three modalities over a data driven approach. Ultimately, we chose to go with a late-stage fusion with data driven methods, utilizing the LiDARs and cameras.

Due to the lack of training data, our initial LiDAR perception approach for Season One utilized an unsupervised clustering algorithm. A Cloth Simulation Filter (CSF) [61] was used to remove ground points and a density-based clustering technique, DBSCAN [11], was used to identify obstacles of specific dimensions within the bounds of the track. While seemingly viable, this approach proved to have many failure cases, such as identifying dust above the track as an obstacle, shown in Figure 3.3. By observing these failures and the potential to break down at higher vehicle speeds,

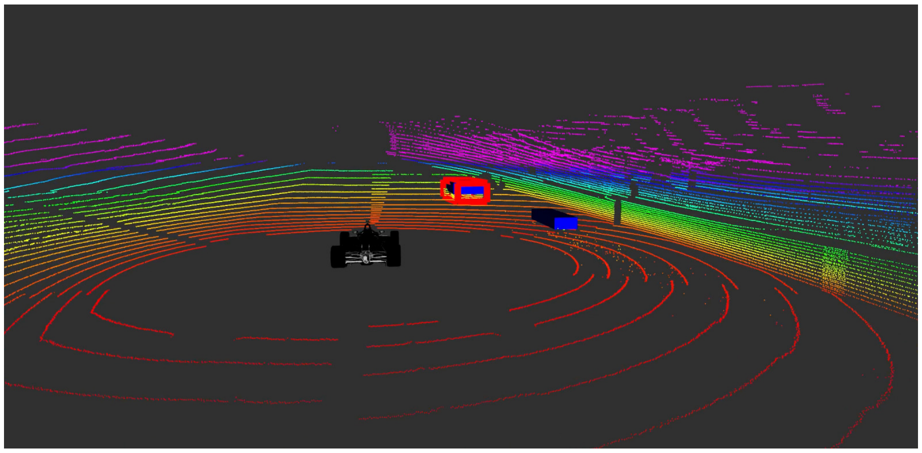


Figure 3.3: Red bounding box is from PointPillars and blue is from clustering. Clustering is susceptible to detecting dust and noise as other agents. Clustering is unsupervised, but, as a result, is unable to differentiate between other agents and dust and debris.

the need for a robust, efficient, learning-based perception approach was evident. No rules-based approach was taken for camera object detection, as we did not feel it was viable.

A late-stage fusion approach was taken because there was concern that the sensors were not reliable and, while the clocks were synchronized between the sensors, the shutters of the sensors are triggered at different moments. As a result, alignment temporally is necessary to ensure the best result, which was potentially expensive, prone to errors, and not guaranteed to be an improvement for our use case. As our ODD highlighted, there is only one other car on track at a time, which makes detection a lot simpler.

3.1.4 Test Planning

As previously stated, since there was a lack of prior training data, all of our early work on clustering was tested on data-sets collected from working in the garage, with other cars around us. We sat down and manually tuned the thresholds and the algorithm's heuristics and hyper-parameters until performance was acceptable. However, we knew this was not viable long-term.

We developed a simulation environment, with sensors configured as closely as we could get to the real vehicle with the simulation model toolbox we had available. In this environment, we could collect ground truth labels that then could be used to train a model. By developing this model, we could generate nearly infinite amounts of data to validate that our model was producing something close to what we expected. However, this was not enough, as we needed to collect real-world data to test with.

Ultimately, our test plans for object detection looked like the following:

- Using simulation, generate a dataset to train a model and validate its performance, in real time
- Using off-track real-world data, test and validate a rules-based clustering algorithm.
- Validate performance and efficiency offline before deploying to vehicle
- While off-track, test and validate that both the rules-based clustering algorithm and learned model are performing as expected on the vehicle.
- On track, collect more data to train and validate the learning based approach.

Finally, the unique aspect of this approach is that it resulted in an initial model and clustering algorithm that can both be used to generate auto-labels on real world data that can then be refined by a human labeler. In our experience, having an initial guess speed up labeling significantly. This exercise was also useful for understanding the existing model's, and data-set's, weaknesses. However, as we will cover later, timelines make it difficult to always complete all test plan steps, which can hurt later when the items that are tested are not necessarily the correct things to have tested.

3.2 Object Detection

In this section, we will detail our approach to the detection of other opponent ARVs, which was derived and refined over time by applying the design methodology and reasoning presented earlier. The input to this system is the raw sensor data (LiDARs, cameras, etc.) and the output is a list of candidate detections of other agents, scored by some confidence.

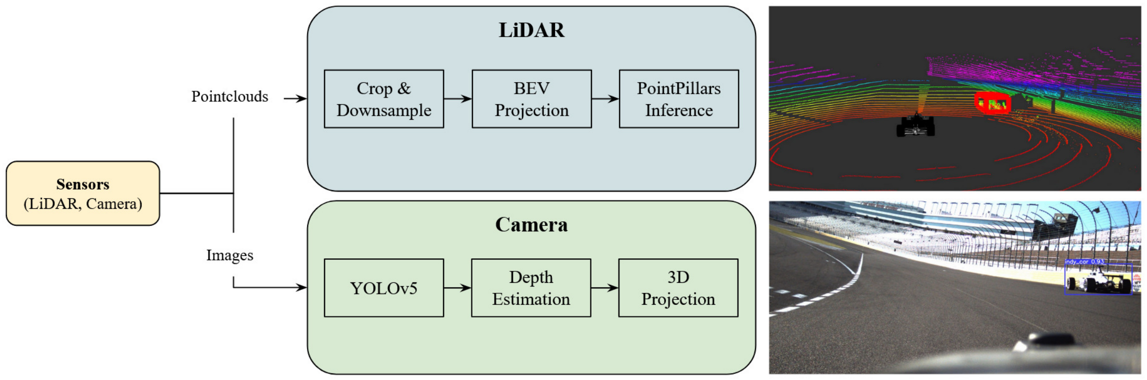


Figure 3.4: Object Detection Stack Overview. Each sensor modality is processed in its own independent pipeline and the resulting detections are associated and fused by the Tracking stack, presented in Section 3.3.

3.2.1 Overview of Approach

Figure 3.4 shows our object detection stack's. As inputs are the camera images and LiDAR data and outputs are 3D object candidates. Each sensor has unique strengths and weaknesses. For example, cameras alone do not give an accurate depth estimation but can operate at a much higher frame rate (up to $75Hz$) and resolution (2064×960) than the LiDARs. A camera-based detection pipeline can provide a higher frequency update on our belief of the world and has the potential to see other agents from further away. Figure 3.5 shows some additional challenges faced with perception. To best exploit the sensors' strengths and for robustness and redundancy, our perception stack uses each sensor independently and in parallel and feeds all localized detections to our tracking stack.

3.2.2 Camera

For full coverage and maximum range, the two front-facing cameras utilize a narrow lens to improve far-field resolution. The four remaining cameras use a wider field of view (FOV) to provide 360° coverage around the vehicle. Due to the lower effort required to label 2D bounding boxes, YOLO v5 [16] was chosen for our initial approach. The model was trained on a custom, hand-labeled data set of other AV-21 vehicles, with images taken from onboard our vehicle. Because the model outputs 2D bounding boxes, other assumptions and processing is required to provide a 3D pose of the other

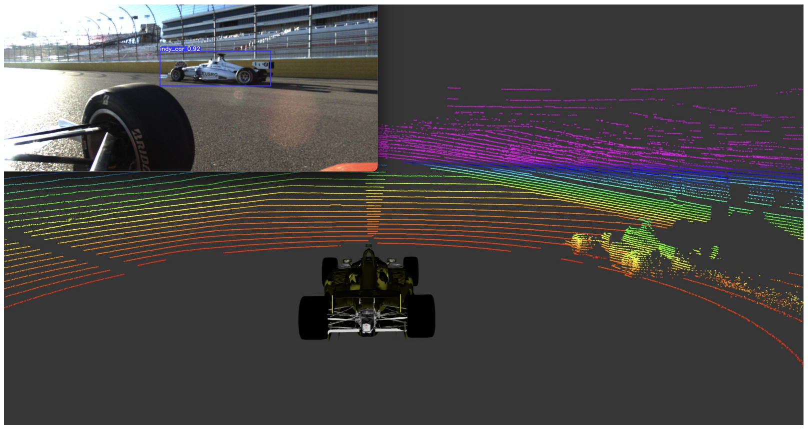


Figure 3.5: Perception Stack Challenges. Lens flare, vibrations and motion blur, glare, and other challenging light and environmental conditions make camera detection challenging. Additionally, dust challenges simpler, unsupervised LiDAR detectors, such as clustering, but requires robust data engineering to ensure a robust deep neural network model. Note: Point cloud is colorized on the z-axis.

agents. By exploiting the fact that the size and shape of the vehicles are known, we can estimate a depth from the 2D bounding boxes from the model by using a standard pinhole optics model [12]:

$$Depth = \frac{(Height_{known} * f)}{Height_{pixels}} \tag{3.1}$$

where f is the calibrated focal length of the camera, $Height_{known}$ is the known height of the vehicle in meters, and $Height_{pixels}$ is the detected height of the detected vehicle in pixels. This monocular algorithm yields accurate results for mid/far-field detections; however, the error increases proportionally with the real-world distance between the camera and the other agent. While far-field detections ($> 100m$) tend to be less accurate, the additional sensor modalities, including LiDAR, cannot see as far as the camera with nearly the exact resolution and fidelity, so some measurement is better than none. As the other agent gets into the LiDAR operating range, we refine the estimates using these detections, and our confidence in the agent's position increases. The unique long-range capability of the camera perception pipeline can provide motion planning more time to respond to agents in our path. Figure 3.6 shows the result of the camera detection pipeline.

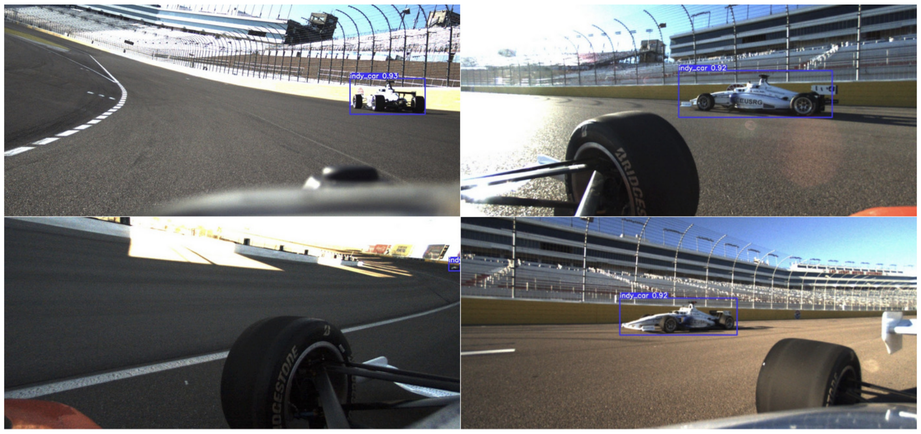


Figure 3.6: Camera Detection Results

3.2.3 PointPillars

The AV-21 platform has three Luminar Hydra LiDARs[28] positioned in a triangular fashion. Each LiDAR has a field of view (FOV) of 120°, together allowing for 360° coverage around the vehicle. Each LiDAR is capable of excellent coverage at over $100m$, thereby providing an over $200m$ radius circle of coverage around the track. Since the track is only so wide, this cloud is cropped further to being $200m \times 40m$. An example cloud can be seen in Figure 3.7.

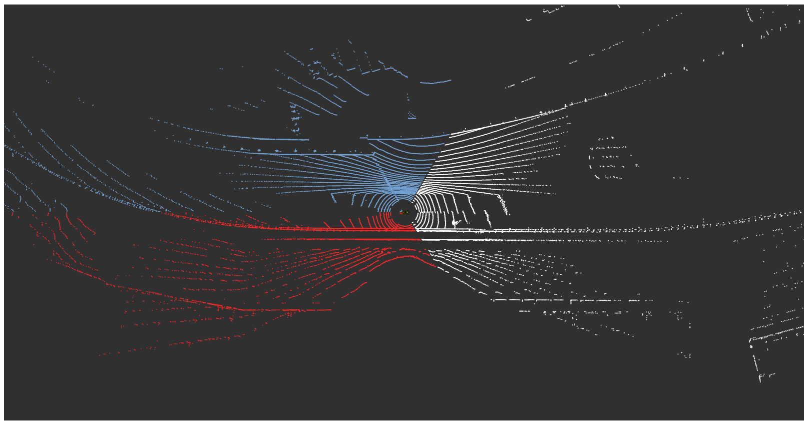


Figure 3.7: Pointcloud from all three Luminar LiDARs. Red points are from the left LiDAR, blue are from the right, and the white are from the front. The LiDARs each can see over 100 meters. The clouds are combined and cropped to provide coverage of $200m \times 40m$.

Numerous Deep Learning methods of object detection using LiDARs have shown promising results, such as VoxelNet [63], PointRCNN [46], SECOND [58], and others. Low-latency inference and accurate detections are of the utmost importance for our use case of high-speed autonomous racing. For this reason, PointPillars [22] serves as our primary detection method, capable of reliably detecting vehicles at ranges up to 100m away. The birds-eye-view projection and 2D convolutions used within PointPillars allow for the removal of computationally expensive and time-consuming sparse 3D convolutions performed by other LiDAR networks.

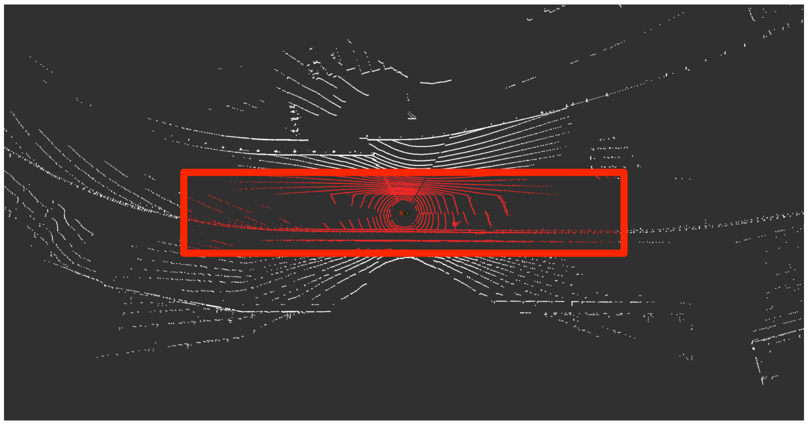


Figure 3.8: The point cloud is cropped, within the LiDAR driver, to the region in red, 200m $\times$ 40m in size.

To reduce processing time, our PointPillars implementation is single-sweep, meaning we do not accumulate scans over time before running inference. Additionally, to further simplify the pipeline, inference is done directly on the raw scans, after down-sampling and applying a crop (Figure 3.8). We explicitly chose to not compensate for distortion caused by the ego vehicle's motion. Based on the data observed and practical considerations within the larger stack, motion compensation was deemed not worth the additionally complexity and processing time required. The scanning rate ($\sim$ 50ms from top to bottom) is faster than other LiDARs, which results in less distortion. Additionally, the LiDAR data is only used for detection and the relative speed between agents is low enough that the error due to motion distortion can be ignored. The potential gains do not outweigh the additional latency. Finally, work had been done to implement distortion correction, but was removed due to integration and performance issues, which will be discussed further in Section 5.1.

3.2.4 ROS 2 and DDS Transport

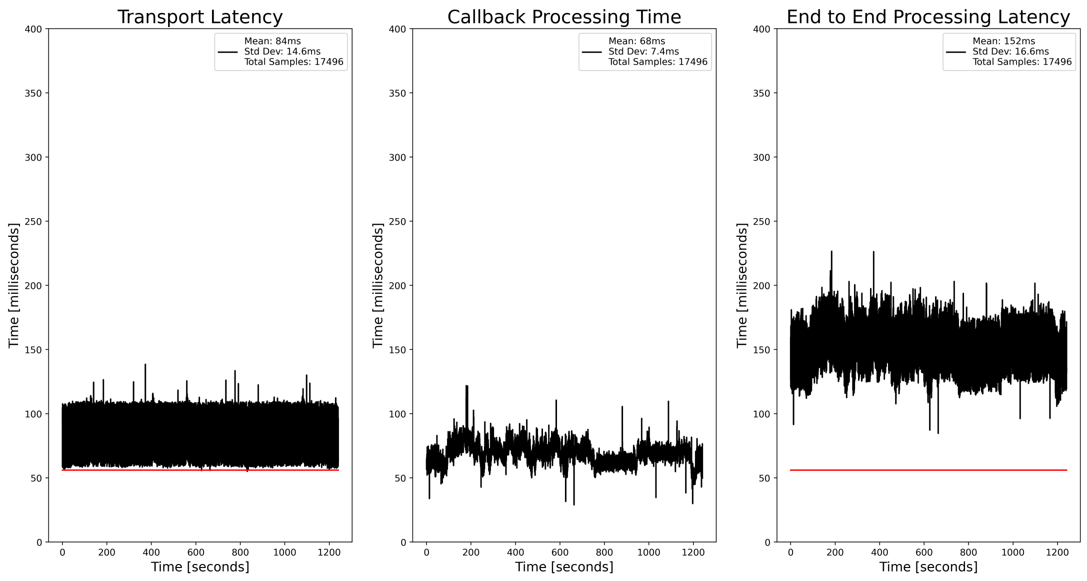


Figure 3.9: Breakdown of the processing time for the PointPillars pipeline in September, 2022. The red line depicts the time it takes from the very first LiDAR point to be scanned to when the final cloud is completed, processed, and transported to the computer via Ethernet, or about $56ms$. Assuming no additional processing, this becomes the ideal latency measurement. When taking the total pipeline latency (right) and removing the contribution from running inference (middle), the transport latency (left) can be seen. This transport latency is the time between the first LiDAR point and the ROS message containing the cloud being received by the PointPillars processing node. As you can see, this time is on average over $84ms$, which means about $28ms$ on top of the ideal latency, and is highly variable, with a standard deviation of $14.6ms$.

ROS 2 utilizes the Data Distribution Service (DDS) standard for communicating between each ROS node[30]. ROS 2 utilizes what is called the ROS Middleware (RMW) abstraction to separate the ROS API and the DDS implementations [30]. This allows for changing DDS implementations easily between different vendors. However, in our own testing, the RMW and ROS 2 client libraries provide significant overhead. For example, the following figure shows the latency for the perception pipeline in September 2022, when using ROS 2 for communication (Figure 3.9).

In Figure 3.9, the red line depicts the time it takes from the very first LiDAR point to be scanned to when the final cloud is completed, processed, and transported

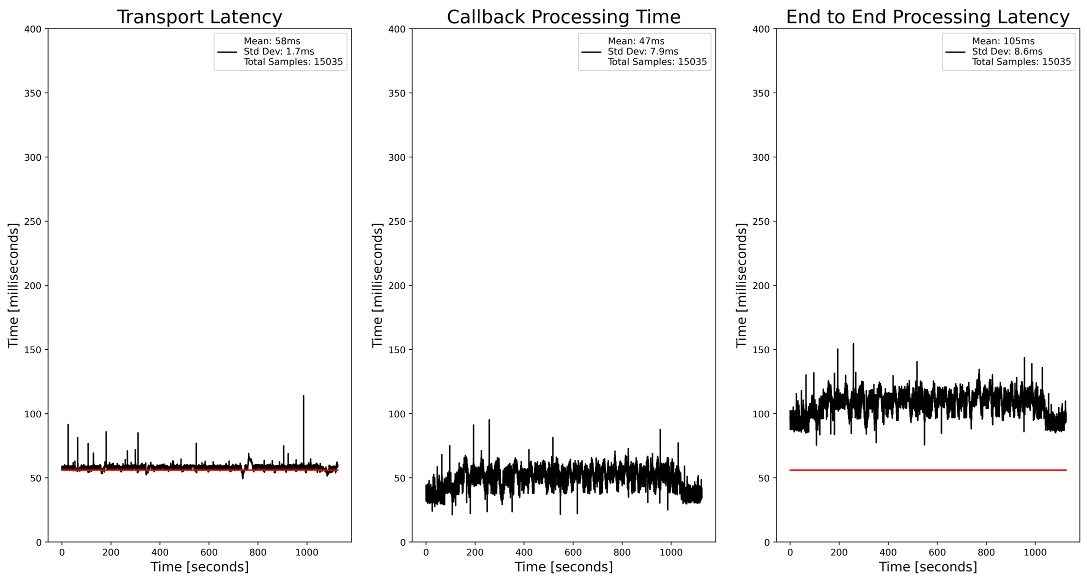


Figure 3.10: Breakdown of the processing time for the PointPillars pipeline in January, 2023. The red line depicts the time it takes from the very first LiDAR point to be scanned to when the final cloud is completed, processed, and transported to the computer via Ethernet, or about $56ms$. Assuming no additional processing, this becomes the ideal latency measurement. When taking the total pipeline latency (right) and removing the contribution from running inference (middle), the transport latency (left) can be seen. This transport latency is the time between the first LiDAR point and the ROS message containing the cloud being received by the PointPillars processing node. As you can see, this time is on average $58ms$, which means about $2ms$ on top of the ideal latency, and is very consistent, with a standard deviation of $1.7ms$.

to the computer via Ethernet, or about $56ms$. Assuming no additional processing, this becomes the ideal latency measurement. When taking the total pipeline latency (right) and removing the contribution from running inference (middle), the transport latency (left) can be seen. This transport latency is the time between the first LiDAR point and the ROS message containing the cloud being received by the PointPillars processing node. As you can see, this time is on average over $84ms$, which means about $28ms$ on top of the ideal latency, and is highly variable, with a standard deviation of $14.6ms$.

To combat this issue, we made two big changes: 1) crop the point cloud in the driver and 2) use DDS directly for communication. Inside the LiDAR driver, every point is checked to see if they are within the vehicle polygon or outside some max range. Since each point is already being iterated over, it adds negligible overhead to also crop the cloud to a fixed region around the vehicle, in our case $200m \times 40m$ (see Figure 3.8). This made the messages slightly smaller and reduced the amount of work the python PointPillars node needed to do.

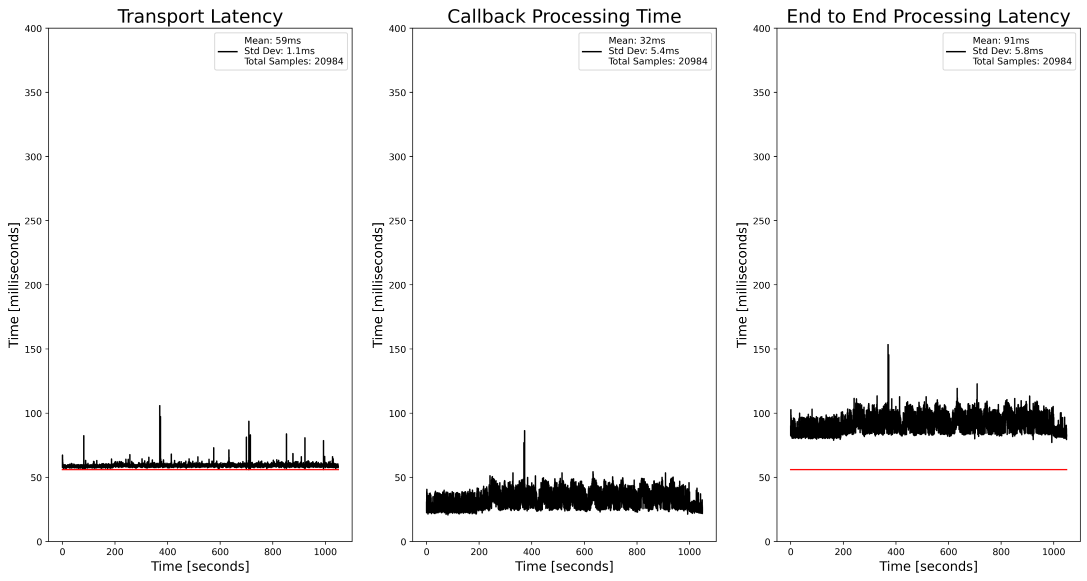


Figure 3.11: Breakdown of the processing time for the PointPillars pipeline in June, 2023, in Monza. The red line depicts the time it takes from the very first LiDAR point to be scanned to when the final cloud is completed, processed, and transported to the computer via Ethernet, or about $56ms$. Assuming no additional processing, this becomes the ideal latency measurement. When taking the total pipeline latency (right) and removing the contribution from running inference (middle), the transport latency (left) can be seen. This transport latency is the time between the first LiDAR point and the ROS message containing the cloud being received by the PointPillars processing node. Despite the point cloud being roughly twice as large, the transport time is on average $59ms$, which means about $3ms$ on top of the ideal latency, and is very consistent, with a standard deviation of $1.1ms$.

Bypassing ROS 2 involved using DDS libraries directly to access the underlying DDS participant and topics. Besides bypassing the overhead of ROS 2 client libraries, this approach also allowed for finer-grained control of the DDS quality of service (QOS) settings, allowing for tuning of the underlying data buffers and transport. The result is a significant reducing in the transport overhead and allowing for much more consistent message delivery:

By Monza (Figure 3.11), additional bugs in the inference node were refined and, despite having roughly twice the number of points per point cloud, the transport latency did not drastically increase and the total latency dropped to about $91ms$, or about $35ms$ of transport and processing, which is an almost three times speedup when compared to the $96ms$ of transport and processing (see Figure 3.9).

3.2.5 Data Collection, Labeling, & Training

At the time of development, no data-sets existed that contained AV21s racing head-to-head. Adequately training PointPillars required developing a large and robust dataset. Initially, data was collected in simulation, which helped developed an initial model. The first model dataset is broken down in Table 3.1. The simulation environment did not match the vehicle setup perfectly. In particular, while the range and coverage were similar, the point cloud was less dense than in real life. Interestingly, we found that the initial model trained off of this data transferred to detecting AV21s on real data, especially at longer ranges, where the cloud is less dense. Figure 3.12 shows a comparison of PointPillars detections against the measured trajectory of an opponent ARV.

Source	**Number of Labels**	**Percent**
Simulation	6744	92.2%
Real Vehicle	570	7.8%
Total	**7314**	100%

Table 3.1: Breakdown of label sources for training the initial PointPillars model. Later iterations incorporated many more labels from the real vehicle. Having a better model produces better auto-labels, thereby speeding up the data collection and training process.

With an initial model, it was now possible to do “auto-labeling”, where the model is used to generate new labels that are then hand-verified by a human annotator. Because the model often provides a detection that is close to ground truth, the workload on the human annotator is reduced. Additionally, by using the existing model to label more data, labels can be focused on the areas where the model performed most poorly.

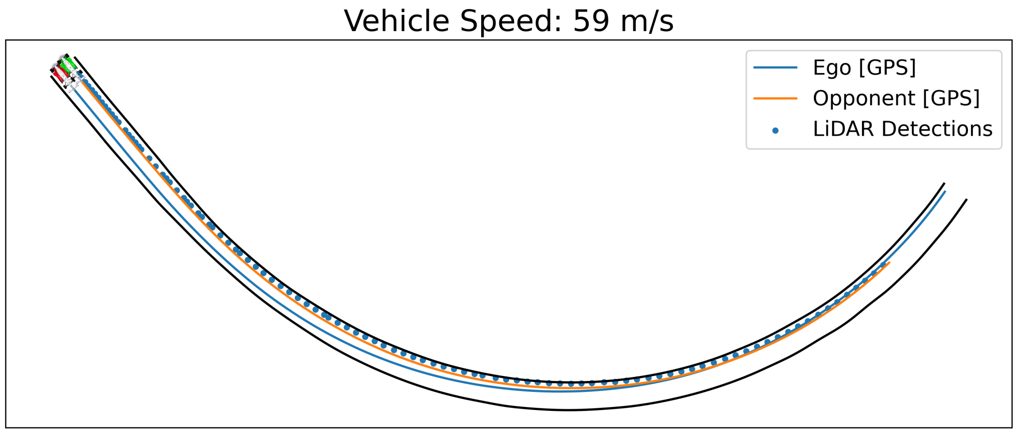


Figure 3.12: Comparison of PointPillars detections against the measured trajectory of the opponent ARV in Season Two at Las Vegas. The ego and opponent ARV positions are obtained using GPS with RTK corrections applied, which provides centimeter-level accuracy. Also shown are the track bounds (black) and the ego vehicle trajectory. Finally, this snapshot of the run was when the ego vehicle was traveling at over $59m/s$, completing a pass of the opponent vehicle.

3.2.6 Discussion: Strengths, Limitations, and Future Work

Given the prototypical nature of the AV-21 platform, the sensor plate must be disassembled every time the autonomy components need servicing. As a result, the extrinsic calibration between the cameras and the LiDARs changes frequently. This is less of an issue with the LiDARs, as they are all firmly fastened to the same aluminum plate. Additionally, the extreme operating conditions of the AV-21 platform (i.e. high speeds and accelerations) also necessitate re-calibrating the sensors regularly, even if the sensor plate has not been removed. Small extrinsic calibration errors can lead to very large projection errors, especially for distant objects. This problem is not exclusive to ARVs and is an active area of research [48][24]. Future work will center on streamlining the calibration process and developing systems that are less brittle to small errors.

Finally, due to a severe crash less than 72 hours before the competition in Season Two at Las Vegas, the camera detection pipeline was disabled for the competition events. An image from the footage of the crash, recorded by an onboard GoPro camera, can be seen in Figure 3.13. With the focus being repairing the AV21 vehicle, no time

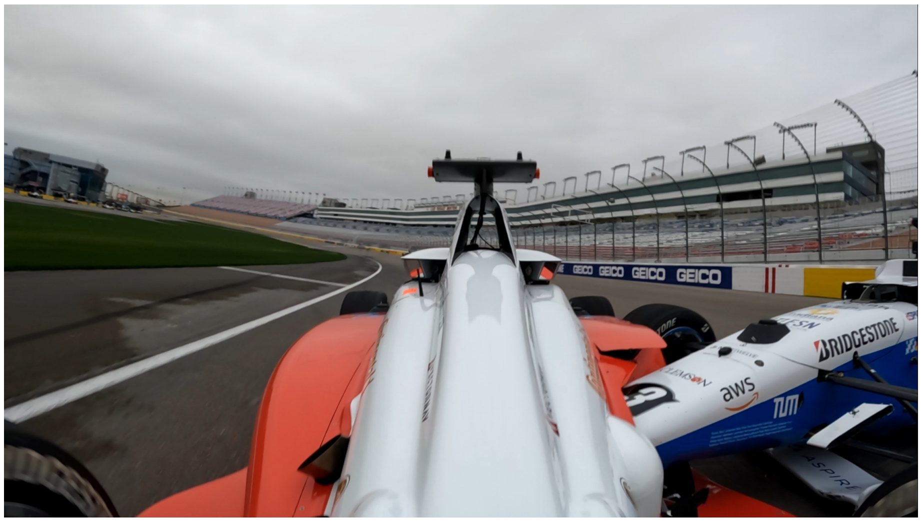


Figure 3.13: Onboard image from the first known major autonomous, multi-agent, head-to-head collision.

.

was available to properly calibrate the sensors and the potential for projection errors outweighed the benefits of running the camera pipeline. Because of the late-stage fusion and decoupled design choices, it was trivial to make such a drastic change. In a fast-paced competition environment, this modularity and flexibility proved paramount in allowing the vehicle to operate during the competition. While the redundancy and peak performance of the stack was compromised, as shown later in Section 5.1, the vehicle was still able to autonomously compete in three rounds, winning the first two, and losing the third after running out of fuel after attempting an overtake at over 150mph.

3.3 Tracking

Tracking within an ARV software stack serves to provide downstream tasks with a single belief of the states of other agents within the world. Different perception modalities capture different portions of a given agent's state space. For example, the monocular camera perception provides a noisy estimate of an agent's position, but cannot accurately predict its orientation. Our LiDAR perception produces full pose estimates of other agents, but currently does not infer the agent's velocity. While using only one of these detection methods will yield a belief that is severely limited by the outlined weaknesses, the effective fusion of both can result in each modality compensating for the drawbacks of the other.

3.3.1 Challenges and Requirements

Our implementation allows for the fusion of multiple sensing modalities in a straightforward manner, and serves to provide downstream planning tasks with the state of all perceived agents. Our decoupled approach to perception requires our tracking stack to meet the following requirements:

1. Incorporate all modalities from perception, including LiDAR and monocular camera detections
2. Estimate positions, velocities, and orientations in the world of all tracked agents
3. Provide a precise and accurate state estimate of the opponent agents
4. Provide a consistent measure of the uncertainty of the agents' state estimates
5. Be robust to false positives, missed detections, and drop-outs from one or more sensor modalities

Finally, tracking must perform all of the above while ensuring as little additional latency as possible, handling measurements from perception asynchronously and out of order, and compensating for any delay between sensor measurement and tracking.

3.3.2 Overview of Approach

Our approach consists of three main components: Filtering, Association, and Fusion. Filtering removes outliers. Association determines whether or not a detection is of a previously seen agent. Fusion is incorporating new measurements of agents' states. In order to minimize processing latency within the tracking stack, well-researched, efficient algorithms are leveraged for each module. Figure 3.14 presents the Tracking pipeline architecture.

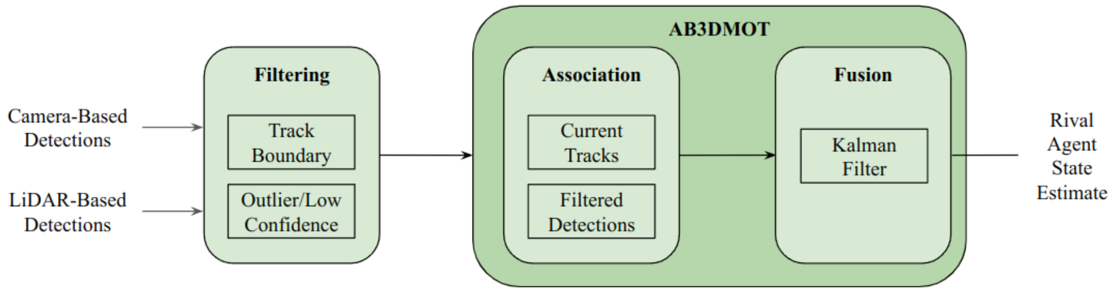


Figure 3.14: The Tracking pipeline architecture, from detection inputs to state estimate for a single opponent ARV.

3.3.3 Filtering

Detections filtered by a confidence threshold, a hyper-parameter within our tracking stack, tuned empirically by analyzing the false positives and associated confidence produced by perception. Additionally, any detection that falls outside of the track bounds is ignored. The combination of these two filtering steps helps to ensure that only valid detections are processed and used to generate tracked agents.

3.3.4 Association

AB3DMOT [53] provides the data association module utilized by tracking stack. By employing two computationally efficient algorithms, the Hungarian algorithm for data matching [20] and the Kalman filter [18] for fusion and prediction, the authors demonstrate strong results on multiple open-source data sets while also providing high-frequency predictions. In practice, we observed that the Hungarian algorithm with Euclidean distance often resulted in poor data association, especially during temporary sensor drop-out. Therefore, our implementation uses simple greedy matching with the Mahalanobis distance [31], which performed better in testing.

Track births and deaths: To reduce the probability of false positives becoming valid tracks, a new potential track is instantiated (born) only after two detections (from successive sweeps) are associated with it. This hyper-parameter provides a means to balance between the quality and confidence of tracks and end to end latency in reacting to other agents. Finally, any tracks that have not had a detection associated with it within the last five seconds are also removed (killed) to prevent stale tracks from influencing future associations.

3.3.5 Fusion

Once detections have been associated with an existing tracked agent, or have been repeatedly observed and classified as a new agent, we begin tracking the agent using fused multi-modal perception outputs. Again, we utilize a modified version of [53] as the Kalman filter for performing sensor fusion. Since incoming detections from the camera perception pipeline have already been projected into a 3-dimensional position and transformed into a common frame, both LiDAR and camera measurements can be used to update the internal Kalman filter for a given tracked agent. In this way, sensor fusion becomes a simple task that can be asynchronous across the two modalities, and the states of tracked agents can be published at the receipt of each incoming detection.

3.3.6 Discussion, Limitations, and Future Work

The tracking pipeline meets all requirements and is sufficiently accurately and performant to handle the IAC Passing Competition. Our modular design was especially important when the camera perception was disabled on race day, outlined in detail in Section 3.2.6. Given these successes, however, our system has not been robustly tested against multiple agents, specifically agents that are close together (i.e. $\leq 5m$). In traditional motorsports, humans drive aggressively in close proximity to one another. While the competition format is far from this style of racing, future works will need to handle such operating domains in order to challenge professional drivers. Multiple agents in close proximity are more difficult to track, due to higher association ambiguity and occlusions.

Finally, a Kalman filter will be replaced by an Extended Kalman filter (EKF) to enable a non-linear motion model. With an EKF and a better motion model (i.e. constant curvature), predictions of agent tracks will be more accurate, which is especially important during periods of infrequent detections (i.e. the other agent is in the blind-spot produced by the rear wing of the vehicle).

Chapter 4

Ego State Estimation and Localization

In this chapter, we will apply the design methodology presented in Chapter 2 to the problem of ego vehicle state estimation and localization. This problem entails estimating the state (i.e. position, heading, velocity, etc.) and localizing the ego vehicle in the world and onto the racetrack. All other modules in the autonomy stack consumes the odometry this module outputs, so it is imperative that estimation is robust and accurate. After having applied the design methodology, we will then detail our approach to solving these problems.

4.1 Applying the Design Methodology

4.1.1 The Operational Design Domain

In this section, the ODD is primarily focused on the IAC@MIMO event held in Monza in Season Three (see Table 1.1). The competition had the following format:

- Each team gets three run attempts to post a fastest lap
- The winning team is whomever posts the fastest single lap out of three runs
- All laps start and end at the start-finish line
- The competition is held on the Autodromo Nazionale Monza (Figure 4.1).

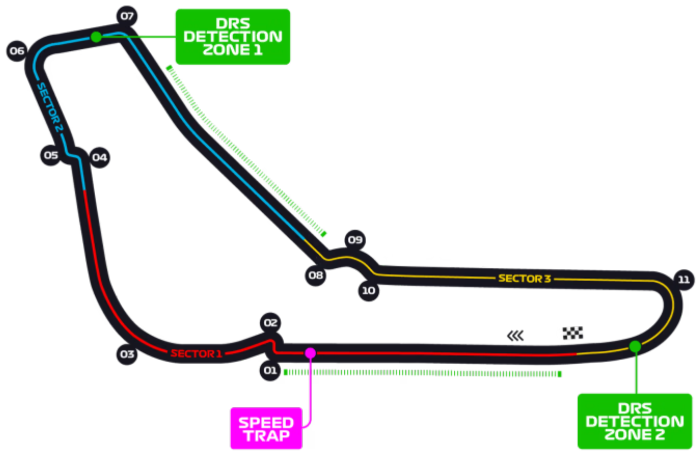


Figure 4.1: Autodromo Nazionale Monza, a famous Formula-One circuit in Monza, Italy. The circuit is one of the fastest in Formula One, with its long front straight and 11 turns, including the famous Parabolica (turn 11) and Ascari chicane (turns 9 and 10). Image from [40].

To summarize, the ODD is to drive as fast as possible, with no need for any perception or even online motion planning. This necessitates having a reliable, robust controller and localization and an optimal racing line. However, Monza has tight turns and chicanes that are more challenging to navigate than an oval super speedway. Additionally, Monza also has dense tree coverage, several walking bridges, and an overpass (see Figure 4.2), all of which obstruct direct view of the sky and negatively impacting GPS performance. GPS-based localization is much more challenging on Monza than the open sky, oval super speedways.

4.1.2 Hardware Constraints

Onboard the AV21 are six cameras, three LiDARs, and three radars, in addition to three GNSS systems, each connected to two antennas and receiving RTK corrections. The output of the GNSS systems can be seen in Table 4.1. The onboard compute is limited to a single computer, a dSPACE AUTERA AutoBox with a 12 core, 24 thread Intel Xeon CPU and an RTX A5000 NVIDIA GPU. All sensors, except for the RADARs, are connected via standard networking. A Cisco IE5000 industrial switch acts as the main hub, connecting the AutoBox to all of the sensors via two 10gigabit SFP+ connections. For more details on the compute and its limitations,

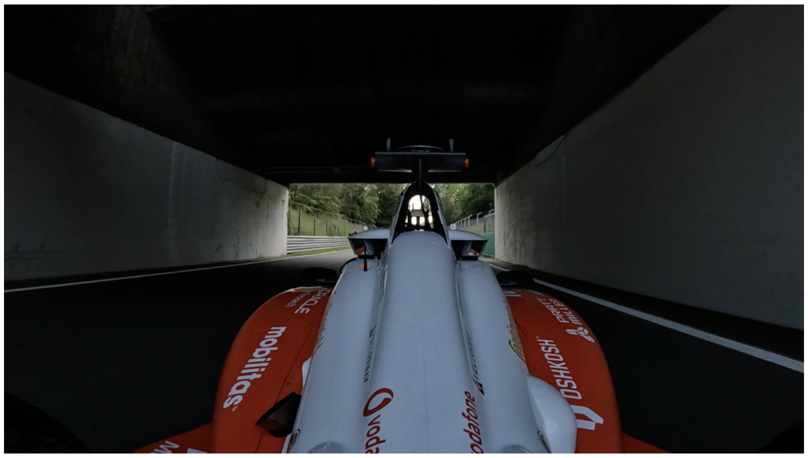


Figure 4.2: One section of the modern Monza track runs under the old track. GPS reception is completely obstructed while in the tunnel. Additionally, the track leading up to the tunnel has dense tree coverage, which creates multi-path interference and obstructs reception. The result is a challenging environment for GPS-based localization.

please see Sections 3.1.2 and 1.3.3.

Measurement Source	**State**	**Update Rate**
Novatel BESTGNSSPOS	$< x, y, z >$	$20Hz$
Novatel BESTGNSSVEL	$< \dot{x}, \dot{y}, \dot{z}, \psi >$	$20Hz$
Novatel HEADING2	$< \theta >$	$1Hz$
Vectornav GPS Pose	$< x, y, z >$	$5Hz$
IMU & Gyroscope	$< \ddot{x}, \ddot{y}, \ddot{z}, \dot{\theta} >$	$125Hz$
Wheel Speed (x4)	$< \dot{x}, \dot{y} >$	$100Hz$

Table 4.1: Localization Measurement Sources. Each GPS unit provides GPS-based solutions for pose, heading, and speed. Each vendor also provides their own inertial-based fusion solutions; however, these solutions have been found to be sub-par to our own fusion approach and are not well tuned for the AV21 platform and the high vibrations.

4.1.3 Enumerate Potential Approaches

Localization is an open area in robotics. Conventional approaches rely on hand-tuned EKF or other Kalman-filter based approaches [34]. Additionally, numerous GNSS

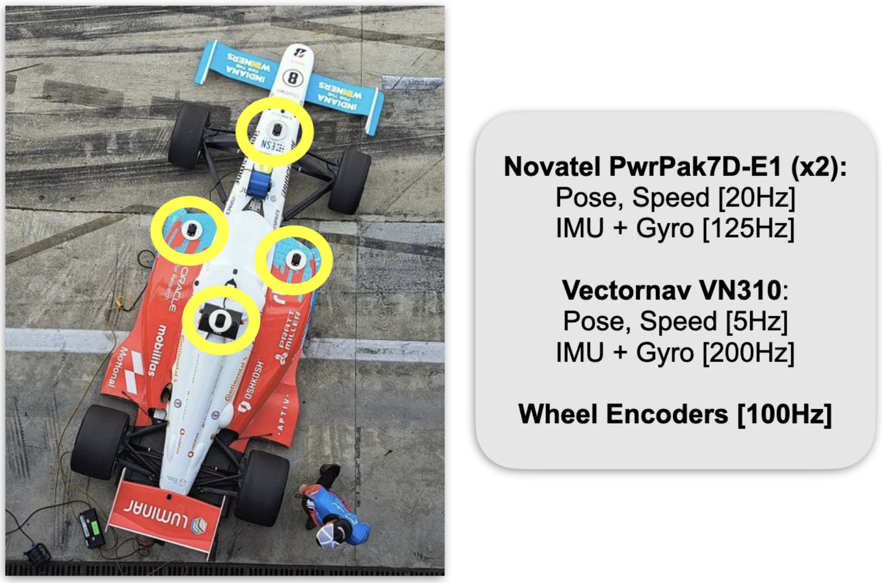


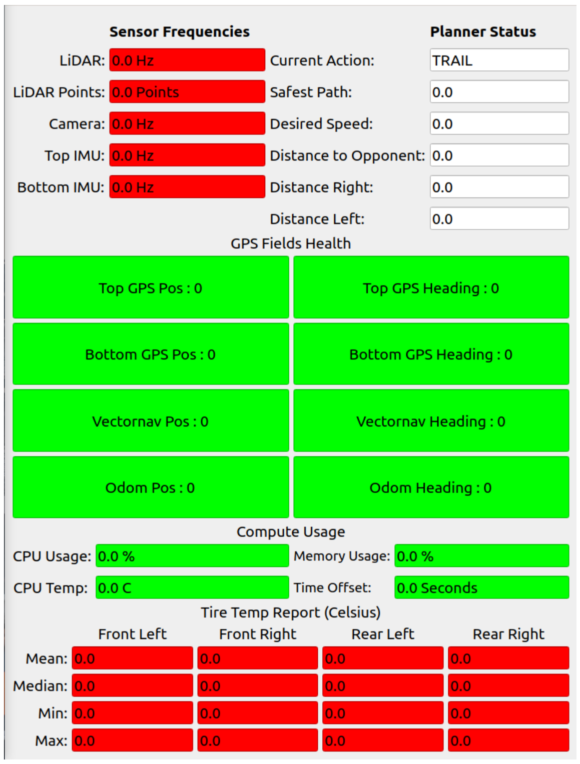
Figure 4.3: GNSS sensors on the AV-21. Each unit (2x Novatel PwrPak7D-E1 and 1x Vectornav VN-310) is connected to two antennas. One of the Novatels is connected to the side pod pair of antennas and the other Novatel and Vectornav are connected to the top antenna pair. Each sensor provides raw GPS positioning, as well as INS fused solutions. However, INS fusion is disabled because the vendor-provided fusion does not work well for the ODD and causes other issues.

hardware vendors all have their own fusion available; however, in practice, it was found that the provided Inertial Navigation System (INS) fusion provided by the AV21 vendors was not reliable and could not be used in practice, due to high vibrations from the chasis and engine.

Other localization approaches focus on GNSS-denied localization, relying heavily on LiDARs and Cameras [41]. These approaches also include localizing in a prior map or building one online (i.e. simulataneous localization and mapping (SLAM)). Since the track is know beforehand, it is possible to create a prior map to localize within. Finally, many of these approaches[45] also utilize inertial, and sometimes GPS, data, along with LiDAR and/or camera data.

However, many GPS-denied approaches struggle in high speed, rapid motion environments and do not provide good estimates of uncertainty or how good the fused result is. For safety purposes, the localization result must be robust and it must be known when there is a failure so that the vehicle can safely stop. Finally, due to the volume of data and the size of the environment, most approaches that utilize LiDARs and cameras are not computationally efficient enough, especially when running on top of a full autonomy stack, to fit within the constraints of the AV21 compute platform. As a result, it was ultimately chosen to stick to a rules-based fusion of the three GNSS systems using an EKF.

4.1.4 Test Planning



Sensor Frequencies

LiDAR:	0.0 Hz
LiDAR Points:	0.0 Points
Camera:	0.0 Hz
Top IMU:	0.0 Hz
Bottom IMU:	0.0 Hz

Planner Status

Current Action:	TRAIL
Safest Path:	0.0
Desired Speed:	0.0
Distance to Opponent:	0.0
Distance Right:	0.0
Distance Left:	0.0

GPS Fields Health

Top GPS Pos : 0	Top GPS Heading : 0
Bottom GPS Pos : 0	Bottom GPS Heading : 0
Vectornav Pos : 0	Vectornav Heading : 0
Odom Pos : 0	Odom Heading : 0

Compute Usage

CPU Usage:	0.0 %	Memory Usage:	0.0 %
CPU Temp:	0.0 C	Time Offset:	0.0 Seconds

Tire Temp Report (Celsius)

	Front Left	Front Right	Rear Left	Rear Right
Mean:	0.0	0.0	0.0	0.0
Median:	0.0	0.0	0.0	0.0
Min:	0.0	0.0	0.0	0.0
Max:	0.0	0.0	0.0	0.0

Figure 4.4: Base station user interface. The AV21 has a cellular modem that allows for the wireless transmission of telemetry from the autonomy stack and vehicle to be transmitted to the base station operator. The operator can also send commands to the vehicle, telling it to stop in the event of an issue or providing a speed limit. In the interface, the operator can see at a glance if each of the GPS units is provided a healthy measurement for position and heading and if the fused odometry is considered healthy.

Before testing on Monza, one of the other teams was able to collect a dataset from their own road vehicle driving on track. Initial integration and testing could be done with this dataset; however, it was not a perfect substitute. With the first track days in Monza, data-sets could be collected that can then be used to test changes to the localization pipeline. Additionally, due to extensive safety checks and telemetry, it was possible to debug and make changes and tune the whole pipeline live, while the vehicle was traveling around the track. All of this was monitored in real-time from a passenger vehicle following the AV21 and from the race control room, seen in Figure 4.4. Finally, between testing days, localization could be tested by manually pushing the car in a parking lot with dense tree coverage.

4.2 Overview of Approach

The design process led us to pursue a rules-based filter, utilizing an EKF to fuse GPS measurements from all three GNSS units. The resulting pipeline must be able to handle degraded solutions from one or more, even all, GNSS units at once and produce an accurate position, velocity, heading, and heading rate, all awhile running onboard the AUTERA compute platform at $100Hz$.

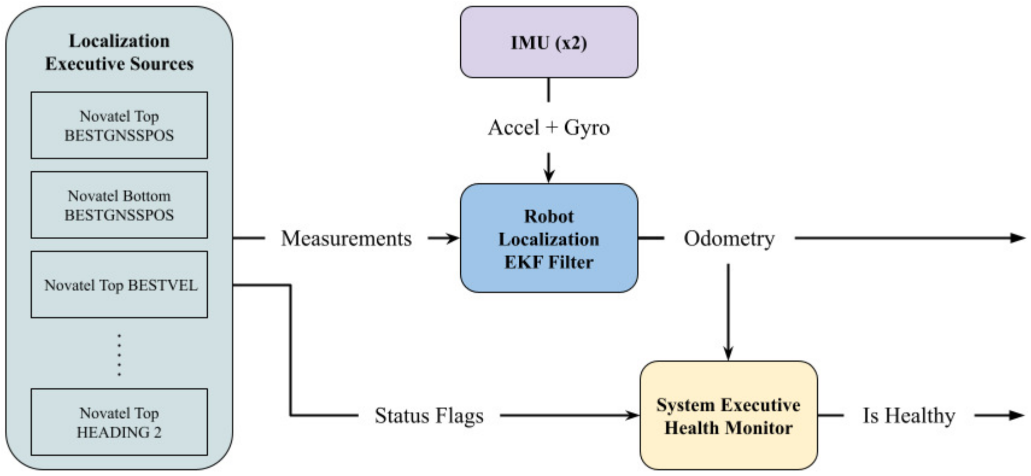


Figure 4.5: Overview of localization pipeline. All raw measurements are first preprocessed by the Localization Executive. The transformed and filtered measurements are then fused by an open-source Extended-Kalman Filter (EKF) package. Finally, health status flags are communicated to the System Executive, which triggers a safety response if needed.

The Localization module is split into two main components: the Localization

Executive and the Robot Localization [33] EKF Filter node. Figure 4.5 shows an overview of the localization stack. The GNSS units are configured to provide the best position, velocity, and heading solutions, utilizing GNSS only, meaning any INS fusion is disabled. These measurements, from the varying sources, undergo the following transformations:

- Pose measurements are transformed from latitude-longitude coordinates into a Local Tangent Plane (LTP) coordinate frame
- Solution status flags, the number of satellites in the solution, covariance measurements, solution status, and other health indicators are tested against a set of heuristics to verify that the measurement is good and worth fusing
- Finally, if the measurement meets quality checks, the measurement is converted to a standard ROS message type and published to the filter for fusion

4.2.1 Localization Executive

The node responsible for all of the above is called the Localization Executive. When building the Localization Executive, it was important to have a flexible, generic, and modular framework for defining sources, safety thresholds, and defining safety checks and heuristics. The Localization Executive's modularity allows us to prioritize high quality measurements and filter out bad measurements on a per-gps and per-source (i.e. GPS N pose, heading, velocity) basis. In general, we found it better to remove potentially erroneous measurements, or measurements with worse quality, from ever being considered by the EKF filter. The consequence of this is that the filter would often "dead-reckon" off of wheel speed and IMU measurements when position and heading measurements are not provided for extended period of times. Figure 4.6 shows the filtered pose measurements from the Novatel and Vectornav GNSS systems and the resulting fused odometry. As you can see in the combined plot, there are extended outages, but the filter is able to extrapolate and produce a reasonable estimate.

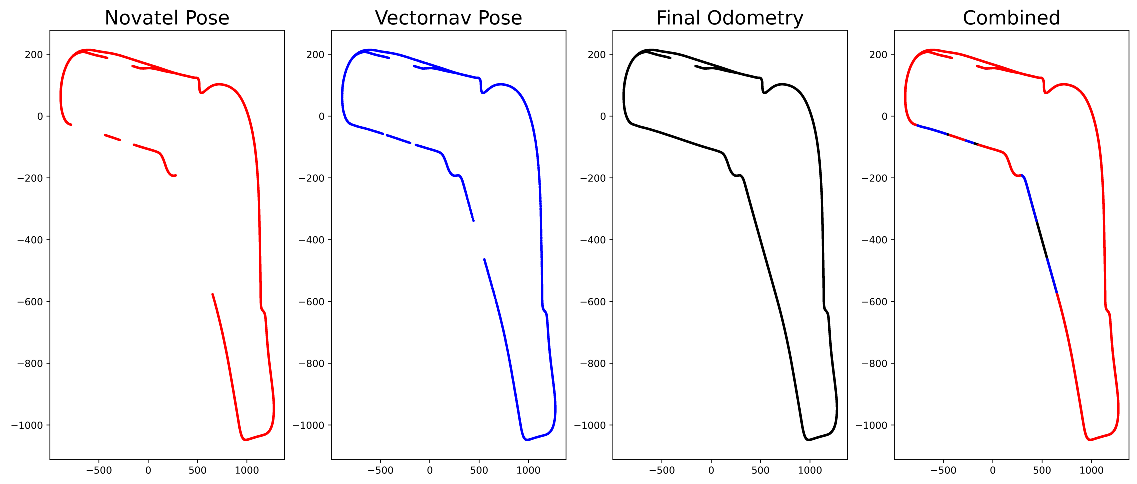


Figure 4.6: The filtered pose measurements from the Novatel and Vectornav GNSS systems and the resulting fused odometry. As you can see in the combined plot, there are extended outages, but the filter is able to extrapolate and produce a reasonable estimate.

4.2.2 Fusion: Extended Kalman Filter (EKF)

The EKF filter used is Robot Localization [33], an open-source filtering and fusion package that provides a flexible and modular interface for fusing an arbitrary number of sensor sources. Robot Localization was chosen because 1) it is very well tested and maintained in the robotics and unmanned vehicles community, 2) its modular design provides extreme flexibility during development and testing, and 3) it works well and exposes several tuning parameters and control, while able to update its state at $100Hz$ and fuse all incoming measurements asynchronously, including with the ability to handle out of order measurements.

Finally, health status flags for the various sources and final odometry results are communicated to the System Executive, the high-level arbiter of the stack. If any of the following scenarios occur, the vehicle is brought to an immediate controlled stop:

- Total connection loss from all GPS units
- IMU measurement update rate drops
- Loss of a quality solution from all GPS units for an extended period of time
- Fused odometry covariance exceeds the safety threshold

If one unit is healthy, or if a combination of pose, heading, and velocity can be constructed from measurements from both units, then the stack will continue operating normally. Additionally, all health status flags are also communicated to the base station and presented in the interface (Figure 4.4). If partial, but not disastrous, failures occur, it is the responsibility of the base station operator decide whether to terminate the run or not.

4.2.3 Discussion: Strengths, Limitations, and Future Work

The localization pipeline was able to work successfully for the IAC@Mimo event in 2023. Moving forward, this approach may struggle if future competitions are held on more challenging tracks, i.e. tracks with even longer GPS outages. So, while this approach has worked well, significant future work will center on incorporating a GPS-denied localization source that can be used in tandem with GPS. This new approach will need to overcome the challenges presented earlier, namely be robust, handle high speeds and dynamic movements, be efficient, and be safe and robust, especially in the presence of other AV21s for multi-agent competitions.

Chapter 5

Experiments and Results

5.1 Evaluation and Results

The aforementioned stack has been developed for and fielded on four different oval raceways across two competition seasons and three years. Over that time, the AV21 ARV has seen hundreds of autonomous miles, in both single and multi-agent scenarios. In this section, we present evaluations of several real-world runs that demonstrate the stackś effectiveness and shortcomings.

Additionally, between Seasons One and Two, while the overall approach and design of the stack had not changed materially, the execution and implementation details have been improved based on lessons learned from the first season. These insights are valuable for designing and fielding complex autonomous systems.

First, we present results and analysis from our failed qualification for the final competition event for the IAC@CES Event on January 7^{th}, 2022. We lay out what failed and the key takeaways. Next, we present a similar analysis for runs from Season Two, highlighting the key changes in the actualization of the stack that enabled single-agent driving and head-to-head passing in speeds above $150 mph$.

5.1.1 Failed Qualification Run Evaluation

In this evaluation, we discuss the compounding issues that prevented the vehicle from detecting and tracking an opponent vehicle, which ultimately led to the team being disqualified from participating in the multi-agent passing competition. Ultimately, the issues were narrowed down to perception and tracking failures, which stemmed from a multitude of causes.

Perception & Tracking Failures

In the run-up to competition day, several last-minute decisions and changes were made to the software, vehicle and sensor configuration, and computer hardware. All of these changes compounded and resulted in a total failure of our perception and tracking stack on the last day to qualify for the head-to-head competition event. While disappointing, the lessons learned reached far beyond software design and development.

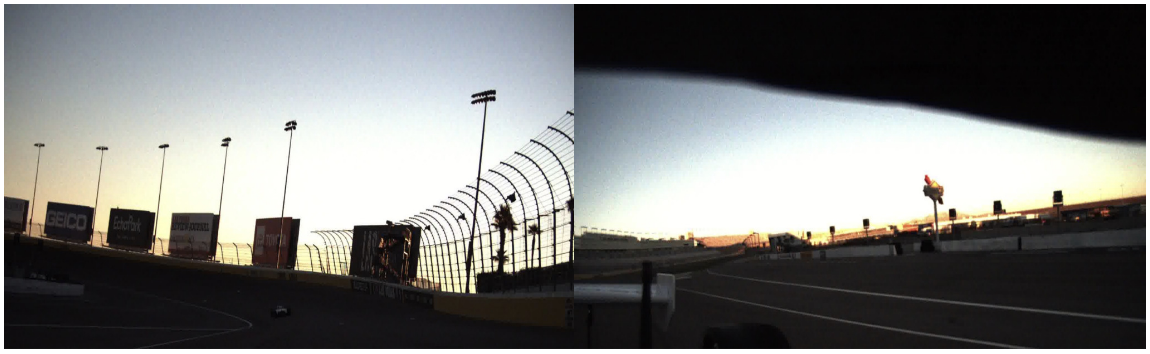


Figure 5.1: Examples camera images from the failed qualification run.

Camera Hardware & Driver Changes

Due to issues with the network connections on the vehicle, the cameras had to be reconfigured to reduce their network bandwidth. However, due to the way the driver configured the cameras, the network settings were not always properly set. Additionally, since the network settings were wrong, all six cameras were trying to use a single network link's bandwidth, which was not nearly enough for the full bandwidth. This meant that any attempts to reset the picture settings, such as crops or exposure parameters, were not set properly, resulting in the streaming of wrongly-cropped or poor quality images. These images, examples shown in Figure 5.1, on top of a lack of training data from the LVMS track, resulted in a higher-than-expected false positive rate from the few images that were actually received over the congested network link. Since we did not have adequate checks at the time, this issue was not caught until after the failed run.

Failed Motion Compensation Changes

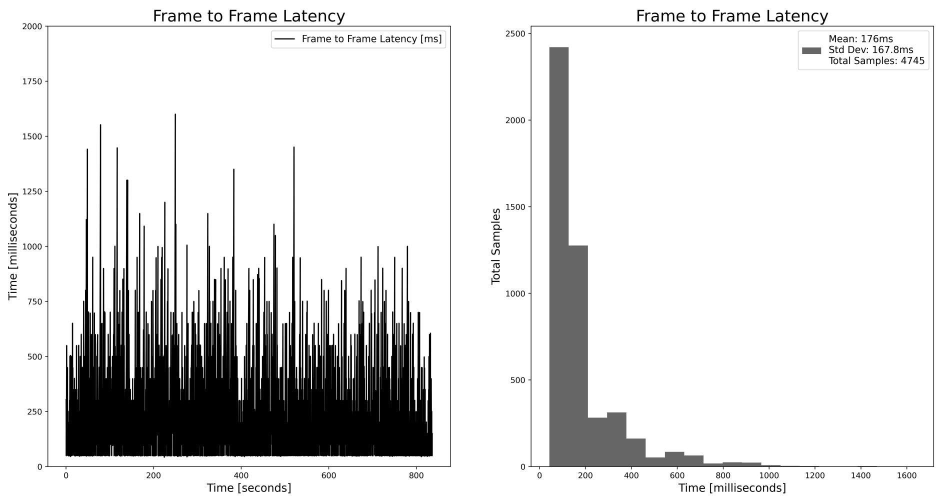


Figure 5.2: Frequency of LiDAR detections during the failed qualification run in January, 2022. The time period between detections was calculated by taking the time delta between sequential LiDAR detections. With the LiDAR streaming at $20Hz$, the expected frame-to-frame period is $50ms$, assuming processing every frame sequentially.

Since the LiDAR is not scanning every point instantaneously, it can be important to account for the ego vehicle's motion during the LiDAR scan for the highest-accuracy result. However, leading up to the competition, the motion compensation solution developed was not thoroughly validated on the vehicle while it was under full system load. On the AV21, there are three LiDARs, each scanning at $20Hz$. The motion distortion node needed to combine all three clouds, project each point into the world to correct the distortion, and, optionally, project the cloud back into a local frame of reference. ROS 2 provides a message synchronization library that gives a convenient interface to subscribe to multiple topics and have all three topics delivered at the same time, in the same callback.

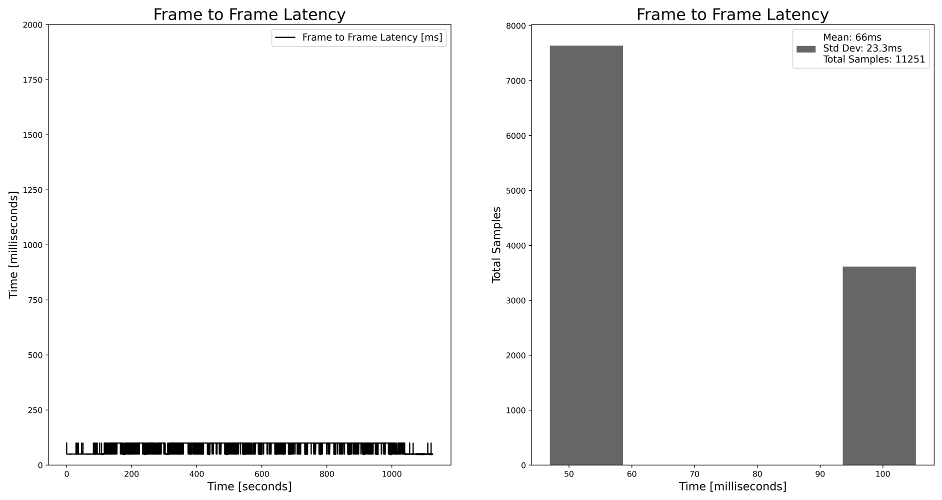


Figure 5.3: Frequency of LiDAR detections during the semi-finals match in January, 2023. The time period between detections was calculated by taking the time delta between sequential LiDAR detections. With the LiDAR streaming at $20Hz$, the expected frame-to-frame period is $50ms$, assuming processing every frame sequentially.

When under full system load, the DDS middleware and synchronization library was not delivering every LiDAR measurement from all three LiDARs. Messages were frequently being dropped. However, because the issue only showed up when the system was under load, it was not discovered until deployed and run on the vehicle during the qualification run.

Due to the dropped messages, the effective LiDAR frame rate received by PointPillars dropped to below $5Hz$. This corresponds to an average period of roughly $200ms$. Assuming sensor processing is keeping up with the sensor frame rates, the expected period is $50ms$, as the LiDAR is scanning at $20Hz$. In actuality, the period between LiDAR measurements was very inconsistent, often spiking over half a second (see Figure 5.2). This inconsistent and slow update rate compounded with other issues with the tracker that resulted in very poor perception performance overall.

Poorly Tuned & Flawed Outlier-Rejection

With the camera pipeline producing false positives, albeit at a low rate due to network bandwidth issues, and LiDAR running at a much lower frequency than what was expected, the tracker's fusion module relied primarily on the Radar for tracks. It must be noted that the tracker in use at the time is very different than the approach discussed in Chapter 3. The issues experienced with this tracker is what necessitated a complete redevelopment and redesign.

Ultimately, last-minute code changes and tuning resulted in poor outlier rejection, which is especially important due to the high noise in radar tracks. While some good tracks surfaced and were followed, tracking failed to prune too many false positives. Additionally, the radar can only see in front of the vehicle, which is insufficient on its own for passing.

Lessons Learned

All three issues quickly compounded and it was clear to the base station operator that there were too many false positives from the perception and tracking stack. As a result, a strategic decision was made to terminate the run and have the vehicle return safely home. While a disappointing result, the run provided invaluable data and insights into testing and deploying a full, complicated system onto real hardware. Most importantly, this failure highlighted the importance of proper integration testing to identify unexpected issues sooner. Additionally, the next section will address the solutions that were developed and tested over Season Two and the resulting performance on track in competition.

5.1.2 Season Two: IAC@CES 2023 Competition Performance

Six out of nine total teams qualified to participate in the passing competition at CES in Las Vegas. In total, MIT-Pitt-RW (MPRW) had two single-agent runs, three successful multi-agent events, and one multi-agent run where the primary communication radio on the vehicle failed, rendering the car unable to compete. A full breakdown of every run, peak speed, and the result is presented in Table 5.1. The following evaluations will focus on Runs 2 and 6 from the table above.

Run	Format	Peak Speed [mph]	Competing Team	Result
1	Single-Agent	145 mph	——	Time Trial, Run One
2	Single-Agent	150 mph	——	Time Trial, Run Two
3	Multi-Agent	115 mph	KAIST	Quarter-Final, Win
4	Multi-Agent	146 mph	KAIST	Re-Run Run 3, Win
5	Multi-Agent	——	Polimove	Radio Died, Disqualified
6	Multi-Agent	153 mph	AI Racing	Fuel Depleted, Loss

Table 5.1: Competition Day Runs. Formats consisted of single-agent time trials and the Passing Competition multi-agent events. The peak speed reached by the vehicle during the run is presented.

Ranking	Team Name	Average Speed [mph]
1	PoliMove	168.2
2	TUM	164.9
3	TII Euroracing	144.4
4	MIT-Pitt-RW	143.8
5	KAIST	138.2
6	AI Racing Tech	65.9

Table 5.2: Final Time Trial rankings and the average of the fastest laps from two runs. To determine the starting brackets and run order, a time trial was held on the morning of the event. Each team had two single-agent runs, each consisting of up to ten laps. The fastest lap time from each run was averaged to determine the team's overall score and ranking.

5.1.3 Season Two: Time Trial, Single-Agent Performance

To determine the starting brackets and run order, a time trial was held on the morning of the event. Each team received two, single-agent runs, each consisting of up to ten laps. Then, the speed from fastest lap time from each run was averaged to determine the team's overall score and ranking. MPRW finished fourth, coming within $0.6mph$ of the third-ranking team, TII Euroracing. The full rankings are presented in Table 5.2.

Overall, the controller was stable and able to navigate at high speeds. As shown in Figure 5.4, the LQR controller balanced performance (maintaining $\leq 1.5meter$ cross-track error (CTE)) while navigating turns with over $24m/s^2$ of acceleration. The speed controller exhibited low-frequency oscillations in tracking the desired speed,

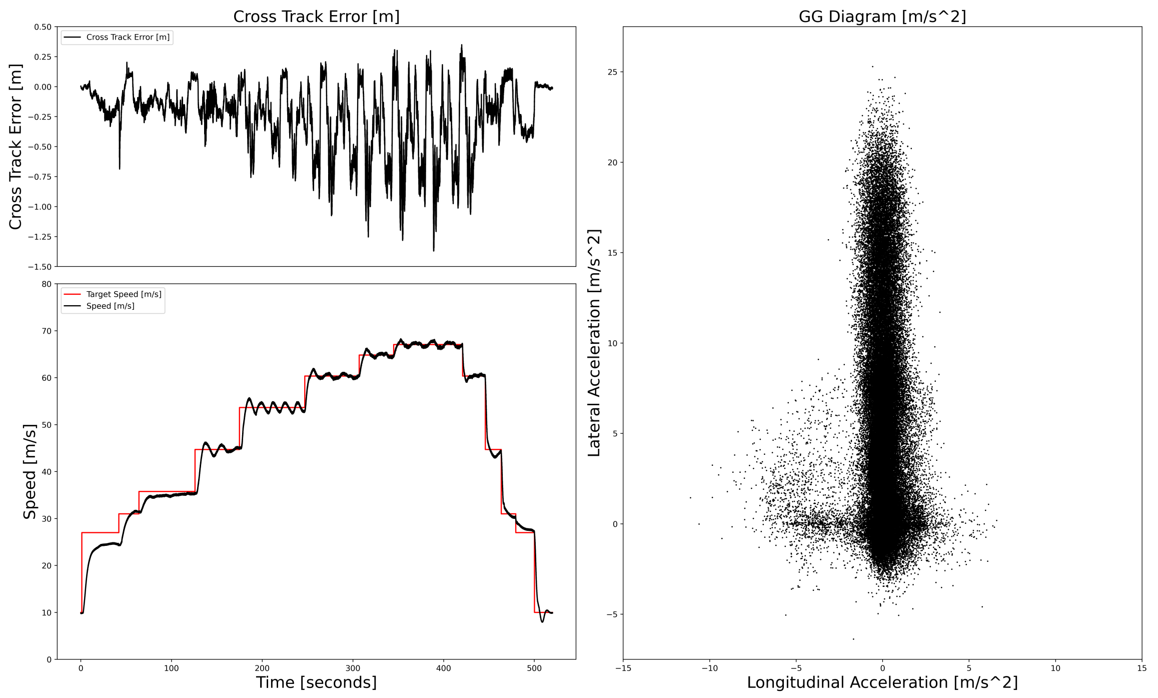


Figure 5.4: **(Top Left)** Cross-track error. **(Bottom Left)** Commanded and actual vehicle speed. **(Right)** G-G diagram showing the vehicle acceleration, in m/s^2. The vehicle reached a peak speed of over $67m/s$, which equates to $150mph$. The cross-track error peaked at approximately $1.4meters$. Lateral acceleration reached over $24m/s^2$, or almost $2.5g's$ of acceleration.

centered around the desired speed. Part of this failure is believed to be a result of changed power train dynamics after the engine was repaired and the vehicle was reassembled. Future work includes better power train modeling and improving the tuning of the throttle controller. The data from this run in particular is instrumental for that future work.

Discussion and Lessons Learned

The first time trial run (at $145mph$) broke the team's speed record from the previous year ($141mph$). The second run quickly broke the team's record again, finally pushing through the $150mph$ barrier after more than three years of development. These milestones boosted team morale, in preparation for the passing competition. It also set the team up for the quarterfinal match-up against the KAIST team.

This run also validated that simple, feedback-based controllers could navigate an ARV at high speeds on an oval race track. It is still unknown what issues will happen on more complicated circuits, i.e. ones with sharp left and right turns. In particular, LQR on its own does not reason about control limits, outside of a naive clamp. Additionally, because it is only ever trying to drive the current state error to zero, LQR can be myopic. On an oval track, this behavior does not cause issues, as the vehicle rarely needs full control bandwidth. However, on more complicated circuits, it is not uncommon to hit the steering limits. Future work is two-pronged: adding a feed forward component to the control produced by LQR and developing a model predictive controller that explicitly considers the maximum control constraints.

5.1.4 Season Two: Passing Competition, Multi-Agent Performance

After the radio failure in Run 55.1, MPRW was disqualified and was not able to compete in the finals. AI Racing Tech also experienced hardware issues earlier in the day, which prevented them from a second time trial run, thus impacting their overall score seen in Table 5.2. As a consolation for the hardware failures, the competition organizers had the two teams compete in a match for 3^{rd} place.

In the match, both teams passed back and forth at the 80, 100, 115, and 125 *mph* speed brackets. The majority of the event was flawless, with only two minor issues, neither of which were the fault of either team's autonomy software. First, early in the run, communication was delayed with the vehicle, so a warning was sent to Race Control as the behavior was similar to what was experienced when the radio had died in the previous match. Race Control preemptively slowed down the AI Racing Tech vehicle, but that was not needed as communication restored itself without any further issues. The second issue was after the 115*mph* bracket was successfully passed. Instead of maintaining the round speed at 115*mph* for AI Racing Tech, Race Control set the speed to 125*mph* prematurely. This was quickly communicated over the radio and Race Control remedied the issue by setting the correct speed and allowing AI Racing Tech an extra lap before considering the round started.

In Figure 5.5, it is possible to see when the vehicle is "trailing" the opponent versus Defending. Periods of a flat, constant commanded speed are when the vehicle

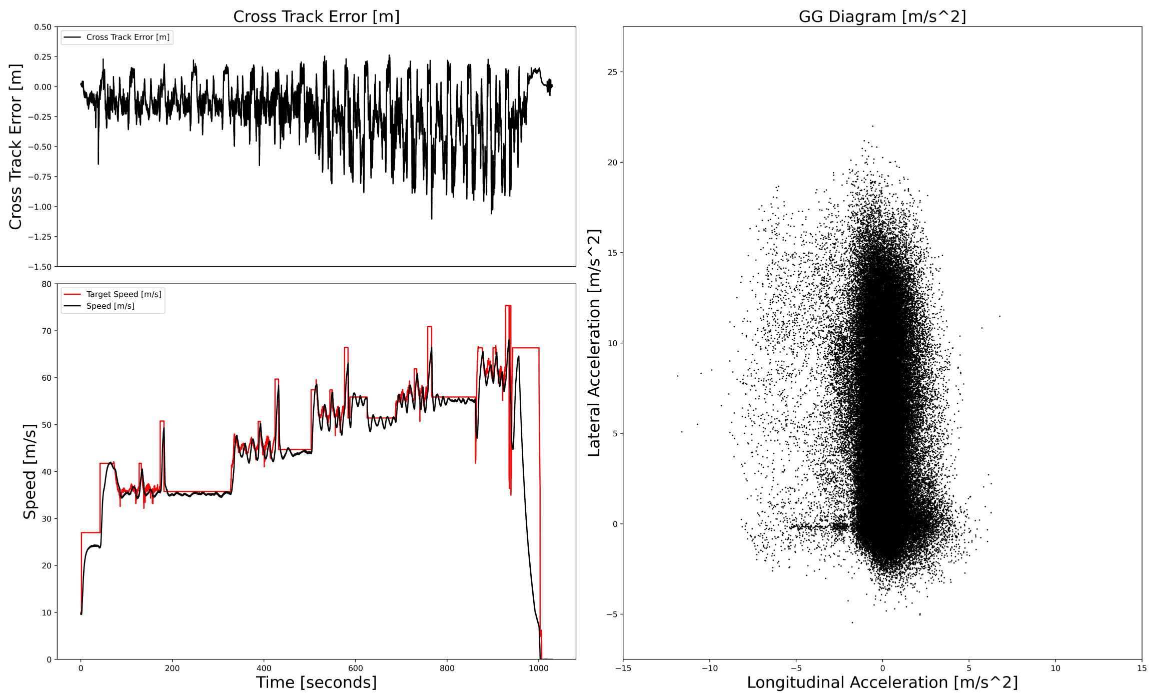


Figure 5.5: **(Top Left)** Cross-track error. **(Bottom Left)** Commanded and actual vehicle speed. **(Right)** G-G diagram showing the vehicle acceleration, in m/s^2. The vehicle reached a peak speed of over $68m/s$, which equates to $153mph$. The cross-track error peaked at approximately $1.4meters$. Lateral acceleration reached over $20m/s^2$, or over $2g's$ of acceleration.

is Defending. Periods of varying commanded speed, followed by large spikes, are when the vehicle is trailing and then passing the other vehicle. In total, four passes were completed, and a fifth was initiated but not completed. When attempting the final pass, a bug with the planner caused a drop in the requested speed. The planner would oscillate between identifying the agent as being in front or behind the ego vehicle, which resulted in an oscillating speed command. At the same time, the vehicle was also running out of fuel. Immediately after falling back behind AI Racing Tech, the planner commanded a higher speed to catch up; however, the vehicle ran out of fuel and could not maintain speed. It is unknown exactly when the engine started experiencing a drop in fuel, as it could have been during the initiation of the pass or after. Additionally, there is a possibility that if more fuel had been in the vehicle, the vehicle would have been slower (due to increased mass), thereby changing the entire dynamics up to this moment. Finally, this occurred in lap one of two, so

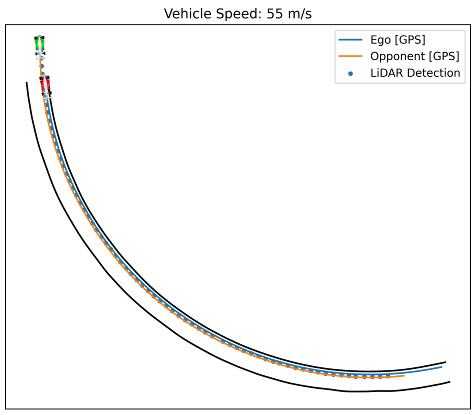


Figure 5.6: Sequence 1 from Defending at 125mph. Each sequence is 7 seconds long. The opponent AV-21 starts from behind the vehicle, moves to the outside lane, and accelerates and maintains a clear inside lane for our vehicle while completing the pass. Our AV-21 is able to detect and track the opponent during the entire sequence, even when they accelerate to pass while our vehicle is maintaining a speed of 125mph.

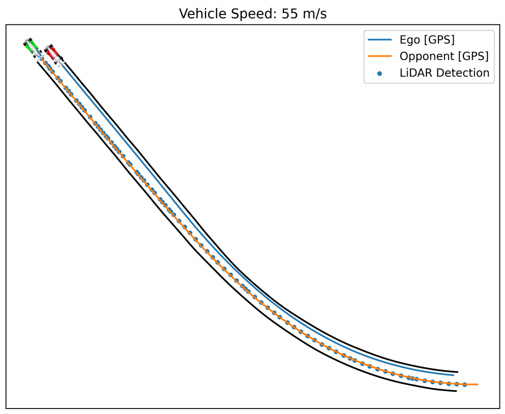


Figure 5.7: Sequence 2 from Defending at 125mph. Each sequence is 7 seconds. The opponent AV-21 starts from behind the vehicle, moves to the outside lane, and accelerates and maintains a clear inside lane for our vehicle while completing the pass. Our AV-21 is able to detect and track the opponent during the entire sequence, even when they accelerate to pass while our vehicle is maintaining a speed of 125mph.

the vehicle would have had a second chance at completing the pass, but did not due to running out of fuel.

Figures 5.8, 5.10, and 5.12 show 7 second long sequences from each of the speed rounds during the match. The RTK GPS measurements from both ego and opponent

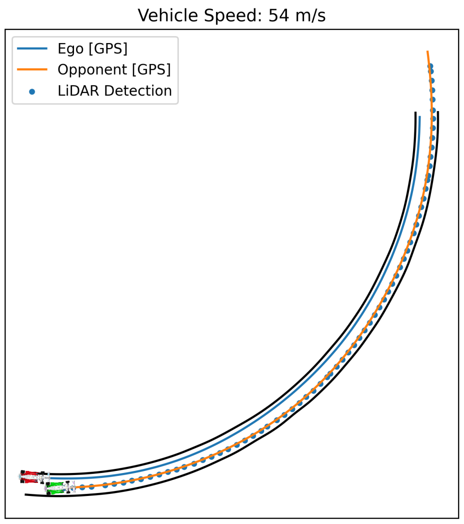


Figure 5.8: Sequence 3 from Defending at 125mph. Each sequence is 7 seconds. The opponent AV-21 is completing the pass, all awhile maintaining a clear inside lane for the ego vehicle. Our AV-21 is able to detect and track the opponent during the entire sequence, even when they accelerate to pass while our vehicle is maintaining a speed of 125mph.

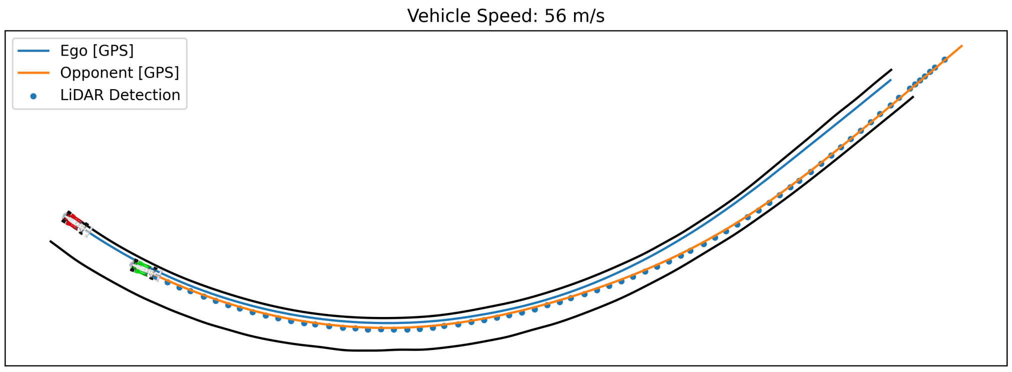


Figure 5.9: Sequence 1 from Attacking at the 125mph speed. Each sequence is 7 seconds long. During the pass, the vehicle reached a peak speed of almost 150mph. The vehicle is preparing for the pass by choosing to maintain the inside lane. During the entire sequence, the stack is able to detect and track the other agent, providing the planner a reliable belief of where the other agent is, allowing for a safe pass, even at such high speeds.

vehicles are plotted, with the LiDAR detections overlaid. The starts of the sequences are denoted by the vehicle graphics. The ego vehicle is red and the opponent is green.

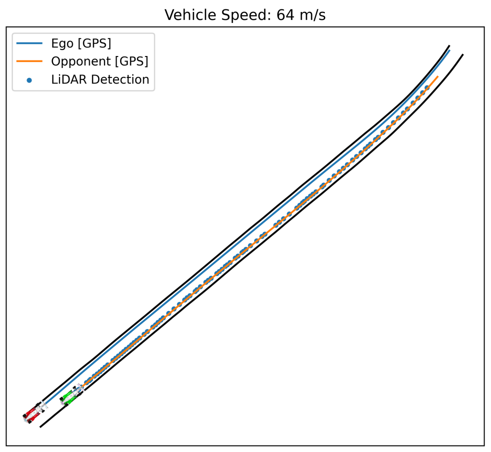


Figure 5.10: Sequence 2 from Attacking at the 125*mph* speed. Each sequence is 7 seconds long. During the pass, the vehicle reached a peak speed of almost 150*mph*. By the time the vehicles enter the turn, our vehicle has completed the pass and begins to drop back to the round speed. During the entire sequence, the stack is able to detect and track the other agent, providing the planner a reliable belief of where the other agent is, allowing for a safe pass, even at such high speeds.

Figure 5.8 shows the performance of the vehicle while maintaining the Defender role. As Defender, the vehicle must maintain a set speed and follow right of way, which depends on roles, relative distances between cars, and where the cars are on the track. While in this role, our planner meets these requirements. Additionally, the perception stack is able to reliably detect and track the other agent, even during a pass. Figure 5.10 shows the vehicle passing another AV21, reaching a top speed of over 150*mph*. During the sequence, the stack is again able to detect and track the opponent and complete the pass safely. Finally, Figure 5.12 shows another pass sequence, but, in this attempt, a combination of a planner bug and the vehicle running out of fuel, the pass is not completed. During all of the sequences, the full stack is working well, able to complete all requirements of the competition and pass another vehicle while traveling at very high speeds.

Discussion and Lessons Learned

In this run, the vehicle achieved a new personal record for its highest peak speed, and it was also the only time the vehicle passed another car going over 125*mph*. In

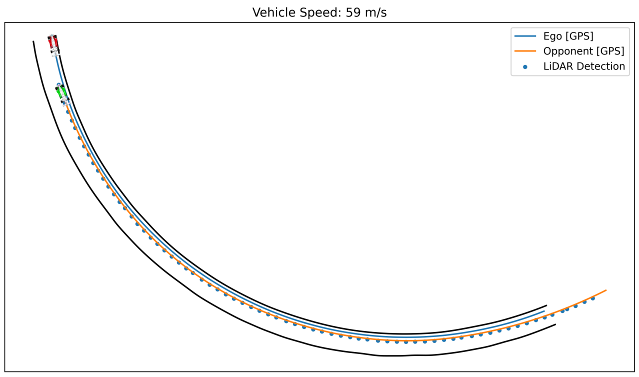


Figure 5.11: Sequence 1 from Attacking at the 135mph speed. Each sequence is 7 seconds long. Even at over 60m/s, the vehicle continued to track the other agent very well. Not long after these sequences, the final pass failed and the vehicle ran out of fuel, rendering the match over.

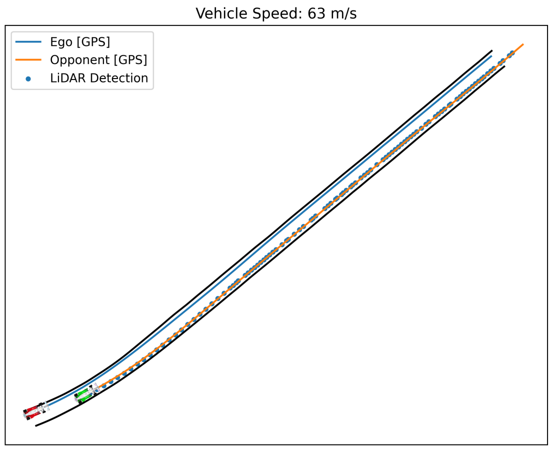


Figure 5.12: Sequence 2 from Attacking at the 135mph speed. Each sequence is 7 seconds long. Even at over 60m/s, the vehicle continued to track the other agent very well. Not long after these sequences, the final pass failed and the vehicle ran out of fuel, rendering the match over.

the testing leading up to race day, the fastest defender ever passed was maintaining 80mph. Additionally, before race day, only five passes were ever achieved, due in part to lost testing time from the crash a few days earlier. On race day, we achieved the following:

1. Three passes at 80mph bracket
2. Two passes at 100mph bracket

3. Two passes at 115*mph* bracket
4. One pass at 125*mph* bracket

In the end, an operational mishap was the deciding factor in finishing in fourth place. While it is likely that the planner's indecisiveness may have surfaced again, we will never know how the vehicle would have performed if it had more fuel. However, in the end, the software stack demonstrated strong performance in executing the IAC Passing Competition.

In the future, fuel consumption will be more carefully monitored and accounted for. Additionally, further testing and focus will be on path planning, to help determine the root cause of the behavior seen in the final pass attempt. As the IAC evolves and tackles more complicated operational design domains (OODs), it is also important for the software stack to evolve as well. Future work will dismantle the assumptions and simplifications to unlock more general, robust performance.

Chapter 6

Conclusions and Future Work

6.1 Lessons Learned

Keep it Simple At many critical points in the project, a decision had to be made on what direction to pursue for specific portions of the stack. What kind of controller do we implement? How should we do camera detection? What methods should we pursue for localization? The tendency was to pursue a complicated solution; however, we recognized that identifying a functional solution was more critical than identifying the best possible solution from the outset. In doing so, we chose to focus on simple solutions and build out complexity when necessary.

One example of this struggle was LiDAR object detection. This proved to be challenging, as we recognized that clustering could be applied as a temporary solution, but was not intended to be a long-term solution. We also recognized that by implementing clustering while simultaneously building a parallel solution, we had a solution to fall back to. Eventually, with enough maturity, a more complicated solution (PointPillars) was produced that met our requirements and was orders of magnitude better than clustering. Additionally, this more complicated solution will scale better as the competition travels to new circuits and incorporates more agents.

The Smallest Issues have the Largest Consequences Issues with time synchronization, balancing Ethernet network load across multiple links, cable and connector integrity, and more can severely diminish system performance. For example, the first week of testing with our recently assembled vehicle was progressing as planned, with quick integration and with the absence of major issues. However, the very next week, when testing on the Indianapolis Motor Speedway for the first time, the RTK GPS did not function properly. An entire test day, less than three weeks from the first IAC event, was lost. Eventually, the root cause was traced to our RTK login being used by a competitor's unit, causing a conflict with the RTK service. This issue was not caught the week prior due to our teams' testing schedules being on alternating days.

In Field Robotics, deploying a system requires understanding how each component interacts and what failures may occur. When many complicated pieces are combined into one package, it is easy to overlook the tiniest details. How does the RTK service handle two units trying to communicate to the server with the same license at the same time? How does DDS handle message delivery if a ROS 2 node cannot keep up with the message rate? How does that library being called decide how many parallel threads to use for processing and what are the downstream effects on other software components running on the same system? All of these issues can destroy the performance of a whole system, which is why testing and validation are so critical.

Know What to Test and Actually Do It Due to the complexity of the systems being built, it is important to have a rigorous and principled testing regime. The size, speed, and operating costs associated with testing full-sized ARVs, such as the AV-21, make testing arduous, expensive, and rare. Offline testing, such as in simulation or off of collected data-sets, is critical to catching issues. While the time pressures of a high-paced competition and the need to develop an entire ARV stack can make thorough validation difficult, our experience has shown that it is imperative to successful deployments. Balancing development and testing is non-trivial when working with limited resources.

Between Seasons One and Two, the team focused on identifying the software failures, and also determining why our development practices failed. In particular, the testing of the perception stack was completely revamped. Data-sets were created by merging our log collections with logs from other teams so that the perception detections could be compared against the RTK GPS of both vehicles at any instant. Any perception code change was validated on this dataset and run on hardware that matched the performance of the ADLINK from Season One (see Figure 1.4). A simulator was developed to simulate the object detection pipelines, with customizable levels of noise, false positive rate, and output “detection” rate, to aid the development and testing of the tracker pipeline. These practices, and more, allowed the team to more thoroughly evaluate performance and prepare for the Season Two events.

Autonomous Racing Demands Strong Algorithms and Systems With this work, we are beginning to address some of the largest challenges in Autonomous Racing. We recognize that there are many failure modes with our approach and assumptions that it makes that only apply under our Operational Design Domain (ODD). What has been achieved is a full, baseline stack that is capable of participating in the Indy Autonomous Challenge head-to-head Passing Competition.

However, in designing and building this software stack, it is clear that no one algorithm can meet every requirement. For example, Model Predictive Control (MPC) is a state-of-the-art control approach; but, a complex numerical optimization approach carries the risk of ill-conditioning or high computation time. A potential solution is to use LQR as a fallback for MPC. For both LQR and MPC, there still exists the potential for model mismatch, such as when driving at very high speeds, or at the traction limits of the tires. Solving these problems requires exploring multiple solutions, including designing better algorithms (i.e. robust MPC) and building better systems (i.e. safety monitoring and response).

Designing the software stack requires a holistic approach. Individual components depend on a set of assumptions about their inputs, the problem, and what their output should be. A misalignment in assumptions between two successive components can lead to degraded performance. For example, a planner may assume an upper bound on the quality of the incoming agent beliefs and a certain level of performance from the trajectory tracking controller. If the planner is too optimistic, it may guide the vehicle too close to other agents. A better algorithm at one level (i.e. using PointPillars over clustering) allows dependent tasks to be more optimistic. Finding a good balance of performance and understanding how to set assumptions is a nontrivial task in systems-level engineering.

6.2 Conclusion and Future Work

We have presented a modular and fast software stack for an Autonomous Racing Vehicle (ARV) capable of navigating at high speeds with minimal lateral deviations, reliably detecting vehicles and tracking an opponent ARV at over $100m$ away, even at high speeds, and safely trailing and passing opponent ARVs. Modularity, speed, and efficiency permeate the entire stack through our choices of algorithms and the systems built around them.

With our approach, we have competed in the Indy Autonomous Challenge events in Indianapolis, Las Vegas, and Texas, which will serve as the base for our entry into future competitions. As MIT-Pitt-RW's approach has evolved, the team's performance has become increasingly competitive. The results for the competition are as follows: Did Not Finish (DNF), Did Not Qualify (DNQ), Quarter-Finalist, and Semi-Finalist. The testing, lessons learned, and data gleaned over this series of events, especially in Las Vegas, are informing future developments of the stack.

Moving forward, we intend to continue to validate our tracking and fusion stack and improve its performance. Additionally, we will build new models with data collected from tooling for auto-labeling; and evaluation metrics built from opponent GPS data will increase our stack's ability to detect competitor ARVs. With a more robust data set, we can explore alternative approaches to improve performance, particularly for long-range detection and velocity estimation. Finally, with a more intelligent controller and the introduction of online vehicle model estimation, we can improve our ability to navigate highly dynamic scenarios at even higher speeds.

Our current software stack addresses many of the challenges laid out previously but notably does not address adversarial agents. Several research directions stem from interactions between ARVs; including motion prediction and forecasting in highly dynamic scenarios, planning under uncertainty in racing scenarios, planning to maximize the reward to the agent while minimizing the risk of collision or instability at high speeds, and more. We hope to explore several of these directions in the future to meet the challenges needed to solve full head-to-head autonomous racing.

Bibliography

[1] Evan Ackerman. Carnegie mellon solves 12-year-old darpa grand challenge mystery, 2017. (document), 1.6

[2] ADLINK. title — adlinktech.com, 2022. [Accessed 23-Apr-2023]. (document), 1.4, 3.2

[3] Sarfraz Ahmed, M. Nazmul Huda, Sujan Rajbhandari, Chitta Saha, Mark Elshaw, and Stratis Kanarachos. Pedestrian and cyclist detection and intent estimation for autonomous vehicles: A survey. *Applied Sciences*, 9(11), 2019. 1.5

[4] R. Behringer, S. Sundareswaran, B. Gregory, R. Elsley, B. Addison, W. Guthmiller, R. Daily, and D. Bevly. The darpa grand challenge - development of an autonomous vehicle. In *IEEE Intelligent Vehicles Symposium, 2004*, pages 226–231, 2004. 1.5

[5] Johannes Betz, Alexander Wischnewski, Alexander Heilmeier, Felix Nobis, Tim Stahl, Leonhard Hermansdorfer, Boris Lohmann, and Markus Lienkamp. What can we learn from autonomous level-5 motorsport? In Peter Pfeffer, editor, *9th International Munich Chassis Symposium 2018*, pages 123–146. Springer Fachmedien Wiesbaden, 2019. 1.5

[6] M. Buehler, K. Iagnemma, and S. Sanjiv. The 2005 darpa grand challenge. In *Springer Tracts in Advanced Robotics*, volume 36, 2007. 1.5

[7] M. Buehler, K. Iagnemma, and S. Sanjiv. The darpa urban challenge. In *Springer Tracts in Advanced Robotics*, volume 56, 2009. 1.5

[8] Jianyu Chen, Wei Zhan, and Masayoshi Tomizuka. Autonomous driving motion planning with constrained iterative lqr. *IEEE Transactions on Intelligent Vehicles*, 4(2):244–254, 2019. 1.5.3

[9] Doreen De Leonardis, Richard Huey, and James Green. National traffic speeds survey iii: 2015, 2018. 1.5.3

[10] dSpace. Autera — dspace.com, 2023. [Accessed 23-Apr-2023]. (document), 1.4, 3.2

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-

based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996. 3.1.3

[12] David Forsyth and Jean Ponce. *Computer vision: A modern approach.* Prentice hall, 2011. 3.2.2

[13] F1TENTH Foundation. F1tenth foundation- about us, 2023. 1.1

[14] Joseph Funke and J Christian Gerdes. Simple clothoid lane change trajectories for automated vehicles incorporating friction constraints. *Journal of dynamic systems, measurement, and control*, 138(2), 2016. 1.5.2

[15] Leonhard Hermansdorfer, Johannes Betz, and Markus Lienkamp. A concept for estimation and prediction of the tire-road friction potential for an autonomous racecar. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 1490–1495, 2019. 1.5.3

[16] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, Adam Hogan, lorenzomammana, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Francisco Ingham, Frederik, Guilhen, Hatovix, Jake Poznanski, Jiacong Fang, Lijun Yu, changyu98, Mingyu Wang, Naman Gupta, Osama Akhtar, PetrDvoracek, and Prashant Rai. ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements, 2020. 3.2.2

[17] Juraj Kabzan, Miguel de la Iglesia Valls, Victor Reijgwart, Hubertus Franciscus Cornelis Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Gosala, Mehak Gupta, Ramya Sivanesan, Ankit Dhall, Eugenio Chisari, Napat Karnchanachari, Sonja Brits, Manuel Dangel, Inkyu Sa, Renaud Dubé, Abel Gawel, Mark Pfeiffer, Alexander Liniger, John Lygeros, and Roland Siegwart. Amz driverless: The full autonomous racing system, 2019. 1.5, 1.5.1

[18] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 1960. 3.3.4

[19] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023. 1.1

[20] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. 3.3.4

[21] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):712–733, 2021. 1.5.3

[22] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and

Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds, 2018. 3.2.3

[23] Mengtian Li, Yuxiong Wang, and Deva Ramanan. Towards streaming perception. In *ECCV*, 2020. 1.5.1

[24] Xingchen Li, Yuxuan Xiao, Beibei Wang, Haojie Ren, Yanyong Zhang, and Jianmin Ji. Automatic targetless lidar–camera calibration: a survey. *Artificial Intelligence Review*, 2022. 3.2.6

[25] Yingwei Li, Adams Wei Yu, Tianjian Meng, Ben Caine, Jiquan Ngiam, Daiyi Peng, Junyang Shen, Bo Wu, Yifeng Lu, Denny Zhou, Quoc V. Le, Alan Yuille, and Mingxing Tan. Deepfusion: Lidar-camera deep fusion for multi-modal 3d object detection, 2022. 1.5.1, 2.4

[26] Yonggang Liu, Bobo Zhou, Xiao Wang, Liang Li, Shuo Cheng, Zheng Chen, Guang Li, and Lu Zhang. Dynamic lane-changing trajectory planning for autonomous vehicles based on discrete global trajectory. *IEEE transactions on intelligent transportation systems*, 23(7):8513–8527, 2022. 1.5.2

[27] Zhijian Liu, Haotian Tang, Alexander Amini, Xinyu Yang, Huizi Mao, Daniela Rus, and Song Han. Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation, 2022. 1.5.1

[28] Luminar. Hydra - luminar (nasdaq: Lazr), 2022. 3.2.3

[29] Jun Ma, Zilong Cheng, Xiaoxue Zhang, Masayoshi Tomizuka, and Tong Heng Lee. Alternating direction method of multipliers for constrained iterative lqr in autonomous driving, 2020. 1.5.3

[30] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. 3.2.4

[31] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936. 3.3.4

[32] Víctor Mayoral-Vilches, Sabrina M. Neuman, Brian Plancher, and Vijay Janapa Reddi. Robotcore: An open architecture for hardware acceleration in ros 2, 2023. 2.4

[33] T. Moore and D. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, 2014. 4.2, 4.2.2

[34] T. Moore and D. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, 2014. 4.1.3

[35] Energy Systems Network. Polimove wins the autonomous challenge at ces®, mak-

ing history as the first head-to-head autonomous racecar competition champion, 2022. 3.1.1

[36] Energy Systems Network. Indy autonomous challenge rule set, 2022. 1.3.2

[37] Energy Systems Network. The indy autonomous challenge, 2023. 1.3

[38] Energy Systems Network. Racecar - av-21 dallara, 2023. 1.1

[39] NREC. Urban challenge, 2007. (document), 1.6

[40] Formula One. Italian Grand Prix 2023 - F1 Race — formula1.com, 2023. [Accessed 14-12-2023]. (document), 4.1

[41] Tong Qin, Jie Pan, Shaozu Cao, and Shaojie Shen. A general optimization-based framework for local odometry estimation with multiple sensors, 2019. 4.1.3

[42] Ayoub Raji, Alexander Liniger, Andrea Giove, Alessandro Toschi, Nicola Musiu, Daniele Morra, Micaela Verucchi, Danilo Caporale, and Marko Bertogna. Motion planning and control for multi vehicle autonomous racing at high speeds, 2022. 1.5.3

[43] Ayoub Raji, Danilo Caporale, Francesco Gatti, Andrea Giove, Micaela Verucchi, Davide Malatesta, Nicola Musiu, Alessandro Toschi, Silviu Roberto Popitanu, Fabio Bagni, Massimiliano Bosi, Alexander Liniger, Marko Bertogna, Daniele Morra, Francesco Amerotti, Luca Bartoli, Federico Martello, and Riccardo Porta. er.autopilot 1.0: The full autonomous stack for oval racing at high speeds, 2023. 2.4

[44] J.M. Rieber, H. Wehlan, and F. Allgower. The roborace contest. *IEEE Control Systems Magazine*, 24(5):57–60, 2004. 1.5

[45] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Rus Daniela. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5135–5142. IEEE, 2020. 4.1.3

[46] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud, 2018. 3.2.3

[47] Tim Stahl, Alexander Wischnewski, Johannes Betz, and Markus Lienkamp. Multilayer graph-based trajectory planning for race vehicles in dynamic scenarios. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3149–3154, Ithaca, 2019. IEEE. 1.5.2

[48] the Long-Range Perception Team. Continuous real-time sensor recalibration: A long-range perception game-changer, 2023. 3.2.6

[49] Wil Thomason, Zachary Kingston, and Lydia E. Kavraki. Motions in microseconds via vectorized sampling-based planning, 2023. 2.4

[50] Andrea Ticozzi, Giulio Panzani, Matteo Corno, and Sergio M. Savaresi. Fast and efficient smooth polynomial lane change generation for high-speed autonomous driving. *OCAR-ICRA 2022*, 2022. 1.5.2

[51] Micaela Verucchi, Luca Bartoli, Fabio Bagni, Francesco Gatti, Paolo Burgio, and Marko Bertogna. Real-time clustering and lidar-camera fusion on embedded platforms for self-driving cars. In *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, pages 398–405, 2020. 1.5.1

[52] José L. Vázquez, Marius Brühlmeier, Alexander Liniger, Alisa Rupenyan, and John Lygeros. Optimization-based hierarchical motion planning for autonomous racing, 2020. 1.5, 1.5.1

[53] Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani. AB3DMOT: A baseline for 3d multi-object tracking and new evaluation metrics. *CoRR*, abs/2008.08063, 2020. 3.3.4, 3.3.5

[54] Grady Williams, Andrew Aldrich, and Evangelos A. Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017. 2.4

[55] Alexander Wischnewski, Maximilian Geisslinger, Johannes Betz, Tobias Betz, Felix Fent, Alexander Heilmeier, Leonhard Hermansdorfer, Thomas Herrmann, Sebastian Huch, Phillip Karle, Felix Nobis, Levent Ögretmen, Matthias Rowold, Florian Sauerbeck, Tim Stahl, Rainer Trauth, Markus Lienkamp, and Boris Lohmann. Indy autonomous challenge – autonomous race cars at the handling limits, 2022. 1.5.1

[56] Alexander Wischnewski, Thomas Herrmann, Frederik Werner, and Boris Lohmann. A tube-mpc approach to autonomous multi-vehicle racing on high-speed ovals. *IEEE Transactions on Intelligent Vehicles*, 2022. 1.5.3

[57] Peter R. Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J. Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael D. Thomure, Houmehr Aghabozorgi, Leon Barrett, Rory Douglas, Dion Whitehead, Peter Dürr, Peter Stone, Michael Spranger, and Hiroaki Kitano. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022. 1.1

[58] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors (Basel, Switzerland)*, 18, 2018. 3.2.3

[59] Jinrong Yang, Songtao Liu, Zeming Li, Xiaoping Li, and Jian Sun. Real-time object detection for streaming perception, 2022. 1.5.1

[60] Mario Zanon, Janick V. Frasch, Milan Vukov, Sebastian Sager, and Moritz

Diehl. *Model Predictive Control of Autonomous Vehicles*, pages 41–57. Springer International Publishing, Cham, 2014. 1.5.3

[61] Wuming Zhang, Jianbo Qi, Peng Wan, Hongtao Wang, Donghui Xie, Xiaoyan Wang, and Guangjian Yan. An easy-to-use airborne lidar data filtering method based on cloth simulation. *Remote Sensing*, 8(6):501, 2016. 3.1.3

[62] Ling Zheng, Pengyun Zeng, Wei Yang, Yinong Li, and Zhenfei Zhan. Bézier curve-based trajectory planning for autonomous vehicles with collision avoidance. *IET intelligent transport systems*, 14(13):1882–1891, 2020. 1.5.2

[63] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection, 2017. 3.2.3