AutoRally

AN OPEN PLATFORM FOR AGGRESSIVE AUTONOMOUS DRIVING

BRIAN GOLDFAIN, PAUL DREWS,
CHANGXI YOU, MATTHEW BARULIC,
ORLIN VELEV, PANAGIOTIS TSIOTRAS,
and JAMES M. REHG

The technical challenge of creating a self-driving vehicle remains an open problem despite significant advancements from universities, car manufacturers, and technology companies. Full autonomy, known as level 5 (see "Society of Automotive Engineers Levels of Driving Automation"), is defined as full-time performance by an automated driving system of all aspects of the dynamic driving task under all roadway and environmental conditions that can be managed

PHOTO COURTESY OF BRIAN GOLDFAIN

Summary

AutoRally is an open-source, 1:5-scale autonomous vehicle testbed for students, researchers, and engineers who are interested in autonomous vehicle technologies. It is designed with robustness and ease of use in mind. At 1 m in length, weighing 22 kg, and with a top speed of 90 km/h, the platform is large enough to host powerful onboard computing and sensing and run state-of-the-art algorithms. At the same time, it is simple and small enough to be maintained and operated by two people, all while providing the capability of exploring driving scenarios including drifting, jumping, high-speed driving, and multivehicle interactions.

Build instructions are publicly available, along with a parts list, computer-aided design models for fabricating custom components, and operating procedures. The platform uses the Robot Operating System (ROS) and can be programmed in Python or C++. Tutorials, reference algorithms, a Gazebo-based simulation environment, and a data set structured as ROS bag files are available from the AutoRally website. The fleet of six AutoRally platforms at Georgia Tech have been used to demonstrate control, perception, and estimation research in a high-speed, off-road driving domain. To date, the fleet has driven hundreds of kilometers autonomously at the Georgia Tech Autonomous Racing Facility.

by a human driver. It is estimated that level 5 autonomous vehicles on public roads will help eliminate more than 90% [1] of the 35,000 annual traffic fatalities caused by human error in the United States [2]; reduce commute time, road congestion, and pollution; and increase driving resource utilization [3].

A largely unexplored regime that level 5 vehicles must master before their introduction to public roads is abnormal and extreme driving conditions. These corner cases include precollision regimes, maneuvering at high speeds, and driving on surfaces other than asphalt and concrete (such as ice, gravel, and dirt), where severe understeer, oversteer, skidding, and contact loss with the road surface are common. Additionally, dynamic, unpredictable environments (such as close proximity to moving vehicles, pedestrians, and other obstacles) require short timescales for perception, planning, and control that challenge traditional methodologies.

It is vital to conduct extensive experimental testing to develop and validate solutions to the corner cases of abnormal and extreme driving. However, cost and safety considerations are major barriers that prohibit routine testing and experimentation using full-size vehicles. Large investments (often up to US$1 million per vehicle) are required for vehicle development and testing, the maintenance of infrastructure and personnel, and the implementation of safety precautions before any data can be collected. No company is prepared to perform tests at the limits of performance where vehicle damage or operator injury are very likely to occur. Moreover, even for a trial run, a closed track is necessary unless special precautions are taken or registration and insurance are obtained in one of the states that have enacted autonomous vehicle legislation [4]. Collecting data during normal driving conditions on public roads does not help much in terms of understanding these rare, but very important, driving events.

Society of Automotive Engineers Levels of Driving Automation

The Society of Automotive Engineers International's J3016 [S1], which was issued in January 2014, provides a common taxonomy and definitions for automated driving to simplify communication and facilitate collaboration within technical and policy domains. It defines more than a dozen key terms, including those italicized in Figure S1, and provides full descriptions and examples for each level.

The report's six levels of driving automation span from no automation to full automation and are described in detail in Figure S1. A key distinction is between level 2 (where the human driver performs part of the dynamic driving task) and level 3 (where the automated driving system performs the entire dynamic driving task).

These levels are descriptive rather than normative and technical rather than legal. They imply no specific order of market introduction. Elements indicate minimum rather than maximum system capabilities for each level. A particular vehicle may have multiple driving automation features, such that it operates at different levels depending upon the feature(s) that are engaged.

System refers to the driver assistance system, a combination of driver assistance systems, or an automated

SAE Level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	Monitoring of Driving Environment	Feedback Performance of Dynamic Driving Task	System Capability (Driving Modes)
Human Driver Monitors the Driving Environment						
0	No Automation	The full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems.	Human Driver	Human Driver	Human Driver	N/A
1	Driver Assistance	The *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* performs all remaining aspects of the *dynamic driving task*.	Human Driver and System	Human Driver	Human Driver	Some Driving Modes
2	Partial Automation	The *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* performs all remaining aspects of the *dynamic driving task*.	**System**	Human Driver	Human Driver	Some Driving Modes
Automated Driving System ("System") Monitors the Driving Environment						
3	Conditional Automation	The *driving mode*-specific performance by an *automated driving system* of all aspects of the *dynamic driving task* with the expectation that the human driver will respond appropriately to a *request to intervene*.	System	**System**	Human Driver	Some Driving Modes
4	High Automation	The *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*.	System	System	**System**	Some Driving Modes
5	Full Automation	The full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*.	System	System	System	**All Driving Modes**

FIGURE S1 The Society of Automotive Engineers (SAE) International J3016 levels of driving automation. (Image courtesy of SAE International.)

driving system. Excluded are warning and momentary intervention systems, which do not automate any part of the dynamic driving task on a sustained basis and, therefore, do not change the human driver's role in performing the dynamic driving task.

REFERENCES
[S1] SAE On-Road Automated Vehicle Standards Committee. (2014). Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. Society of Automotive Engineers. Warrendale, PA. [Online]. Available: https://www.sae.org/standards/content/j3016_201806/

Computer simulations offer an alternative to testing with full-scale vehicles. Indeed, the fidelity of computer simulations has improved significantly in recent years, as they can be used to generate an almost unlimited number of driving conditions. However, they cannot completely replicate the complex interactions of an autonomous vehicle with the real world. For autonomous vehicles to be safe, the failures and unforeseen circumstances encountered during real-world testing (many of which are impossible to duplicate in simulation) must be overcome. The development and evaluation of new autonomous driving technologies may not be based solely on computer simulations, which may fail to capture critical aspects of the real world.

Scaled vehicles provide a useful complement to existing methods for testing autonomous vehicle technologies in aggressive driving regimes. Scaled vehicles ranging in size from 1:16 to 1:5 the size of an actual vehicle [often based on radio-controlled (RC) vehicles] are easier and less expensive to operate than a full-sized platform. Despite recent progress and many publications detailing scaled autonomous testbeds [5]–[14], much of the available results lack reproducibility because of the one-off nature of these testbeds, restrictions imposed by the use of private data sets, and inconsistent testing methods. Inconsistency is an especially critical problem because many researchers should be able to test and compare potential algorithms under the same conditions and platforms to be able to obtain meaningful comparisons and advance the science of high-speed autonomy.

The remainder of this article describes AutoRally, shown in Figure 1, a 1:5 scale robotics testbed for autonomous vehicle research. As discussed in "Summary," this article outlines the offline and online estimation methods that were tested on AutoRally and presents experimental results collected with the fleet of six platforms at the Georgia Tech Autonomous Racing Facility (GT-ARF).

AutoRally is designed for robustness, ease of use, and reproducibility, so that a team of two people with limited knowledge of mechanical engineering, electrical engineering, and computer science can construct and operate the testbed to collect real-world autonomous driving data in whatever domain they wish to study. Complete construction and configuration instructions for the AutoRally platform are publicly available and include all required computer-aided design (CAD) files for custom part fabrication, a complete parts list, and wiring diagrams [15]. In addition, operating procedures, simulation environment, core software and reference controllers written in C++ and Python, and a collection of human and autonomous driving data are publicly available [16]. See "Build Your Own AutoRally Platform" for more information about the build process.

SCALED AUTONOMOUS DRIVING PLATFORMS

Experimental testbeds are an essential component of robotics research that enable real-world experimentation and transition to practice. Here, we summarize some prior efforts in creating scaled platforms for autonomous vehicle research. Full-size platforms based on passenger vehicles are not discussed because they are outside the scope of this article and include many commercial development activities that are not in the public domain.

Scaled platforms constructed from modified RC cars are popular in the academic and hobby communities. These platforms are typically 0.2–1-m long and weigh between 1 and 25 kg. Costs range from a few hundred to tens of thousands of dollars, largely determined by the size, sensors, and computing. Construction, maintenance, and programming are typically handled by a small team of students or researchers. Recently, several open-source projects released complete documentation and interface software, which is in contrast to the one-off nature of older work that often lacked enough information to replicate.

FIGURE 1 Autonomous driving with AutoRally at the Georgia Tech Autonomous Racing Facility.

> **It is vital to conduct extensive experimental testing to develop and validate solutions to the corner cases of abnormal and extreme driving.**

Documentation for open-source platforms normally includes parts lists, build instructions, and interface software for the sensors and actuators. The availability of tutorials, simulation environments, and public data sets vary by project. Common sensors include wheel speed, inertial measurement units (IMUs), cameras, depth sensors, ultrasonic units, and light detection and ranging (lidar) units. The target environment for these platforms is typically indoors on a smooth surface. The Donkey Car [5] is an easy-to-build 1:16-scale autonomous platform for the do-it-yourself Roborace events that target hobbyists. Onboard computing and sensing are accomplished via a Raspberry Pi 3 with a matching wide-angle camera. The Berkeley Autonomous Race Car [6] is a 1:10-scale vehicle designed as a simple and affordable research platform for self-driving vehicle technologies that has been successfully used to demonstrate various control algorithms. The onboard ODROID-XU4 is similar in computational performance to the Raspberry Pi 3, and the sensor suite includes a hobby-grade camera, IMU, four ultrasonic range finders, and Hall-effect wheel-speed sensors. The F1/10 project [7] and accompanying Autonomous Racing Competition allow teams to race using a common 1:10-scale platform developed at the University of Pennsylvania. Computing on the F1/10 platform is performed by an Nvidia Jetson. The sensor suite includes a hobby IMU, compact indoor Hokuyo 2D lidar, and optional structure and Zed depth and motion-sensing cameras. The 1:10-scale Rapid Autonomous Complex-Environment Competing Ackermann-Steering Robot (RACECAR) [8] from the Massachusetts Institute of Technology is a platform for researchers creating applications for self-driving cars. RACECAR also uses the Nvidia Jetson for computing and includes the same Hokuyo lidar and Zed stereo camera as the F1/10 platform. Table 1 provides a comparison of these open-source scaled platforms.

Build Your Own AutoRally Platform

AutoRally is an open-source platform, so complete documentation to build, configure, and run the platform is publicly available through the AutoRally platform build instructions on the GitHub repository [15]. Documentation includes build instructions for the chassis and compute box, a parts list with suggested vendors, wiring diagrams, files required for custom part fabrication, and operating procedures. Tutorials and example controllers written in C++ and Python that use the Robot Operating System, a Gazebo-based simulation environment that resembles the Georgia Tech Autonomous Racing Facility (GT-ARF) oval track, and a data set of human and autonomous driving captured at the GT-ARF tracks are available through the AutoRally software GitHub repository [16]. AutoRally is designed for robustness, ease of use, and reproducibility, so that a team of two people with limited knowledge of mechanical engineering, electrical engineering, and computer science can construct and operate the testbed to collect real-world autonomous driving data. Links to tutorials for the background skills required to construct AutoRally are included in the instructions.

Construction time is 40 h for an AutoRally chassis and 60 h for a compute box. The full platform construction time of 100 h does not include custom part fabrication time that depends on the tools available. AutoRally can take significantly fewer than 100 h to construct with prior experience with radio-controlled electronics, soldering, computer construction and wiring, or Ubuntu configuration. Conversely, the platform can take much more time to construct if individual assemblies are not thoroughly tested before integration, which can result in time-consuming rebuilds during verification.

For fabrication of custom components, access to a three-dimensional (3D) printer, laser cutter, water jet cutter, and aluminum welding are suggested. If you do not have access to a 3D printer, all custom parts provided as stereolithography files with the build documentation can be fabricated by a 3D printing service, many of which are available online. If you do not have access to a laser cutter, the custom foam and acrylic parts can be cut by hand with a blade using the provided portable document format files printed on U.S. letter paper as stencils. If you do not have access to a water jet for cutting aluminum parts or welding equipment for aluminum, most local metal fabrication shops should be able to fabricate the compute box enclosure and front brake bracket using the drawing interchange files and bend patterns included in the instruction materials.

> **The development and evaluation of new autonomous driving technologies may not be based solely on computer simulations, which may fail to capture critical aspects of the real world.**

While all of these platforms are easy to build, moderately priced, and offer some onboard sensing and compute capabilities, their design limits their use to smooth surfaces, typically indoors. All of the platforms lack a Global Positioning System (GPS) device, which is a common sensor for outdoor vehicles. Instead of a GPS, global position information can be provided by instrumenting the environment such as with a VICON external motion-capture system or beacons rigidly mounted around the environment. These systems restrict the possible operating space to a couple hundred square meters because of the sensor field-of-view (FOV) and resolution restrictions, and they are priced in the tens of thousands of dollars for out-of-the-box solutions. The chassis, mounts, and enclosures of the platforms are typically not designed for repeated crashes and collisions that are inevitable when testing novel autonomous vehicle technologies, so the delicate sensors and electronics are easily damaged when something goes wrong. Onboard computing is inadequate for much of the state-of-the-art research because of size and power limitations. This necessitates significant code optimization or offloading of computation to a remote computer. Offboard computation introduces its own set of problems, including increased latency and dependence on a reliable, high-bandwidth wireless connection (which dictates the size and configuration of testing environments). The limited payload capacity and power availability also severely limit the ability to test new sensors such as a lidar and high-frame-rate machine-vision cameras because the size, weight, and data rates quickly overwhelm the platforms.

Many one-off experimental platforms have been created for specific projects. In [9], a model predictive control (MPC) algorithm running on a stationary desktop computer with a motion capture system was used to drive a custom 1:10-scale RC platform around an indoor track with banked turns, jumps, and a loop-the-loop. Platforms were developed to test autonomous drifting

controllers in [10] and [6] and to push scaled autonomous driving to the friction limits of the system in [11]. A framework with multifidelity simulation and accompanying hardware platform for use in reinforcement-learning problems relating to autonomous driving was presented in [12]. A 1:5-scale autonomous platform was developed to investigate stability control in [13] and [14]. While these platforms were successfully used for their respective experiments, there is not enough public information available to build, operate, and program one without starting from scratch.

Traditionally, scaled autonomous driving platforms were purposely built for one experiment. However, a new wave of open-source platforms is emerging. Still, none are robust enough to survive repeatedly pushing the vehicle to its mechanical and software limits, let alone operate in outdoor environments with the payload capacity to carry a variety of popular sensors and powerful onboard computing. Therefore, scaled platforms show great promise in the wide variety of experiments they enable. However, previous attempts fall short in terms of design, fidelity, and repeatable performance.

TABLE 1 A comparison of open-source, scaled autonomous platforms. All platforms are based on 1:10-scale radio-controlled cars and include C++ and Python software interfaces that use the Robot Operating System software libraries (except for the Donkey Car). The Donkey Car is 1:16 scale and includes a Python interface. The build time and cost of each platform do not include three-dimensional-printed parts, which will vary based on the printer used. In addition to the platforms themselves, the availability of a simulation world and public data sets is indicated.

| Platform	Cost (US$)	Build Time (h)	Weight (kg)	Computing	Simulation	Data Sets
Donkey Car	200	2	2	Raspberry Pi	Y	Y
Berkeley Autonomous Race Car	500	3	3.2	Odroid XU4	Y	Y
Massachusetts Institute of Technology Rapid Autonomous Complex-Environment Competing Ackermann-Steering Robot	3,383	10	4.5	Nvidia Jetson	Y	N
F1/10	3,628	3	4.5	Nvidia Jetson	N	Y

THE AUTORALLY ROBOT

The AutoRally autonomous vehicle platform is based on a 1:5-scale RC trophy truck and is approximately 1-m long, 0.6-m wide, and 0.4-m high; weighs almost 22 kg; and has a top speed of 90 km/h. The platform is capable of autonomous driving using only onboard sensing, computing, and power. While larger than many other scaled, autonomous ground vehicles, the platform offers a cost-effective, robust, high-performance, and safe alternative to operating full-sized autonomous vehicles and retains a large payload capacity compared to other scaled platforms built from smaller RC cars. AutoRally's capabilities offer a large performance improvement over traditional scaled autonomous vehicles without the need for the large infrastructure investments and safety considerations required for full-sized autonomous vehicles. The complete system diagram for the AutoRally robot with a remote operator control station (OCS) is shown in Figure 2. The remainder of this section describes the configuration of the AutoRally chassis and compute box, including mechanical components, sensors, and computing configuration.

Chassis

The chassis is designed as a self-contained system that can easily interface to a wide variety of computing and sensor packages. Similar to a standard RC car, the chassis can be driven manually using the included transmitter. Computer control and chassis state feedback are provided by a single universal serial bus (USB) cable connected to an onboard computer. Feedback from the chassis to an attached computer includes wheel-speed data, electronic speed controller (ESC) diagnostic information, and the manually provided actuator commands read from the RC receiver.

The 1:5-Scale Radio-Controlled Truck

The chassis is based on a HPI Baja 5SC RC trophy truck. Figure 3 shows the assembled chassis with all modifications installed and the plastic protective body removed. The total weight of the assembled chassis is 13 kg. The major upgrades from the stock chassis are an electric powertrain conversion, front brake installation, and electronics box replacement. The electric conversion replaces the stock 3-hp, 26-cc, two-stroke gasoline engine with a
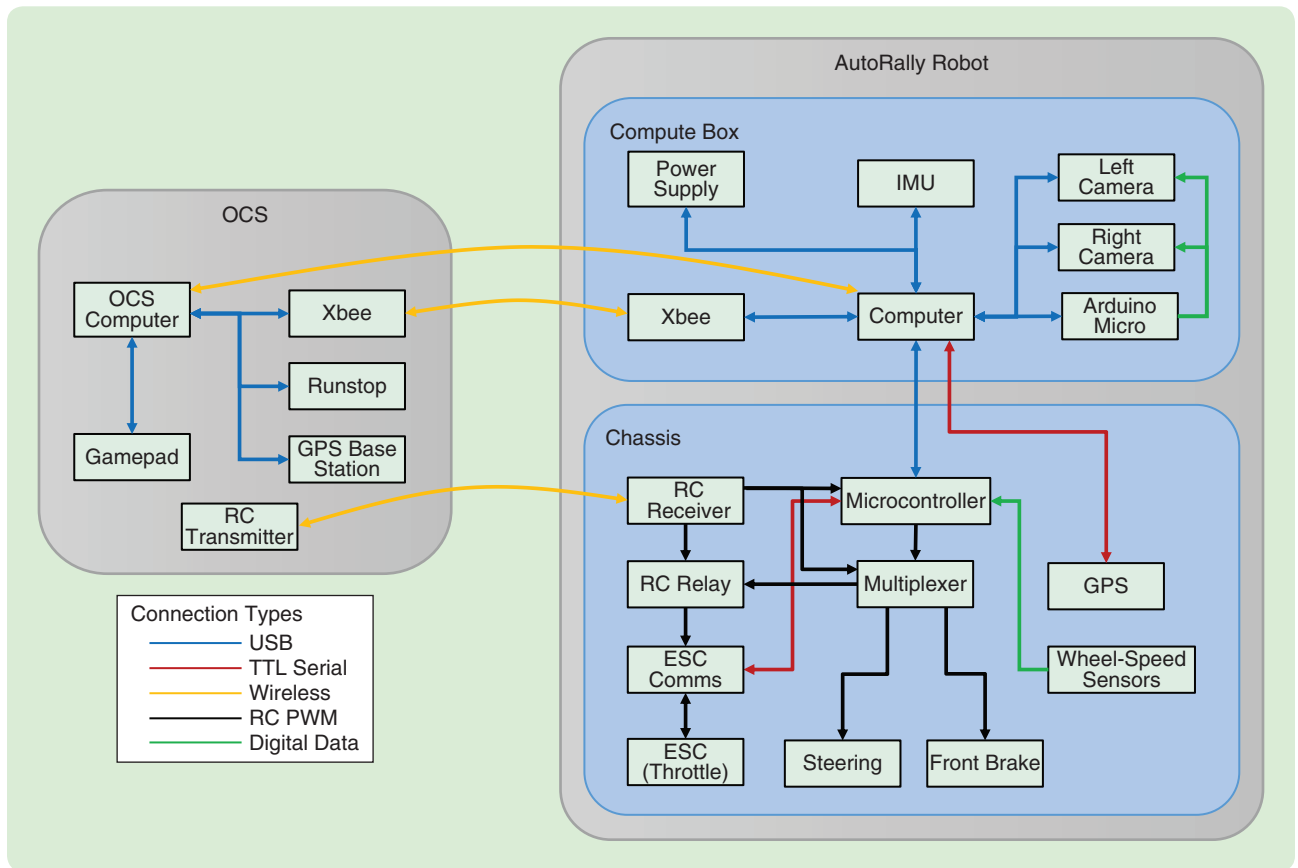
FIGURE 2 The AutoRally system diagram. All major electronic system components and their connections to the rest of the system are shown. The setup includes the AutoRally robot composed of a chassis and compute box, along with a remote operator control station (OCS). Communication between the OCS computer and robot are via hobby radio-controlled (RC) signals, Wi-Fi, and 900-MHz XBee radios. Sensors, including the inertial measurement unit (IMU) and Global Positioning System (GPS) receiver, are connected to the computer with universal serial bus (USB) cables. The actuators in the chassis include the electronic speed controller (ESC), steering, and front brake and are controlled by standard 50-Hz hobby pulsewidth modulation (PWM) signals. TTL: transistor–transistor logic.

10-hp peak output electric motor and ESC from Castle Creations. Compared to the stock engine, the electric motor is more powerful, responsive, and reliable. It also provides an integrated electronic rear brake, generates less heat and no exhaust residue, and requires minimal maintenance. The motor and chassis electronics are powered by two four-cell, 14.8-V, 6,500-mAh lithium-polymer batteries connected in series. A full charge lasts 20–90 min, depending on usage. Front hydraulic brakes are actuated by a separate brake servo.

Parts of the chassis structure were upgraded to handle the increased weight of the sensor and computing package. The stock plastic steering linkage was replaced with billet aluminum parts to withstand the increased steering torque of the upgraded steering servo and weight on the linkage. The plastic side rail guards used as mount points for the compute box were replaced with billet aluminum parts to carry the weight of the compute box without deflecting. Axle extenders were installed to increase the track of the vehicle by 3.8 cm to improve lateral stability and make room to mount the front brakes and wheel-rotation sensors.

The stock suspension springs were replaced with stiffer ones of similar overall dimensions to reduce body roll and improve driving dynamics. A full AutoRally platform weighs 58% more than the stock chassis, so the spring constants were increased by roughly the same percentage. Custom springs are prohibitively expensive, so off-the-shelf springs were sourced as close to the desired dimensions and stiffnesses as possible. The front spring constant increased from 8.48 to 15 lb/in and the rear spring constant from 11.17 to 19.09 lb/in. The shock oil viscosity was also increased approximately 58% from 500 to 850 cSt to properly damp the upgraded springs.

The stock two-channel transmitter was replaced with a programmable four-channel transmitter as part of the electronics box upgrade. The first two channels control the steering and throttle, respectively, and the remaining channels are used in the vehicle safety system discussed in the section "Safety System."

SENSORS

To sense wheel speeds, a Hall-effect sensor and magnets arranged in a circular pattern to trigger the sensor were installed on each wheel hub. The chosen sensor is a Hallogic OH090U unipolar switch, and the magnets are N52 grade, 0.3175-cm diameter, 0.1588-cm thick. The magnet can trigger the sensor from up to 0.58 cm away. Larger magnets could be used to increase the maximum tripping distance. However, the chosen setup works reliably and fits easily in the wheel-hub assemblies. Hardware timers in the Arduino Due in the electronics box are used to accurately measure the time between magnets. Intermagnet timing information is translated to rotation rates and sent to the compute box at 70 Hz.

Inside the electronics box, the RC signals from the receiver are read by the Arduino Due at 50 Hz and sent to the compute box so that, even under manual control, the control signals sent to the actuators can be recorded. This is especially useful for collecting training data where human control signals are required. The Due also receives diagnostic information from the ESC that is forwarded to the compute box.

A Hemisphere P307 GPS receiver provides absolute position at 20 Hz, accurate to approximately 2 cm under ideal conditions with real-time kinematic (RTK) corrections from a GPS base station. The GPS antenna is mounted on top of a ground plane at the back of the chassis along with the receiver. The antenna is located at the maximum distance from the compute box to reduce interference and maximize the view of the sky while still being protected during rollovers. The ground plane is an acrylic sheet coated with a copper conductive sheet and designed to break before the GPS antenna or sensitive GPS board in the event of a severe crash.

Actuators

The chassis requires one servo to operate the steering linkage and one to actuate the master cylinder for the front brakes. Both servos use the 7.4-V digital hobby servo standard, which offers more precise, higher-torque output, faster response time, and a reduced dead band compared to traditional 6.0-V analog servos. All control signals, for both servos and the ESC, are standard 50-Hz hobby pulsewidth modulation (PWM) signals with a duty cycle from 1 to 2 ms, with a neutral value of 1.5 ms. The servos do not have position feedback.

FIGURE 3 The assembled AutoRally chassis.

Custom Components

The custom three-dimensional (3D)-printed acrylonitrile butadiene styrene (ABS) plastic parts installed in the AutoRally chassis are a new electronics box, a GPS box, mounts for the back wheel-rotation sensors and magnets, and alignment guides for the front brake disks. ABS plastic is an easy and lightweight medium for quickly manufacturing complex geometries for components that do not experience significant loading. The electronics box replaces the stock one mounted in the front of the chassis superstructure, just behind the steering servo and linkage. Contained within the box are the radio receiver, Arduino Due, servo multiplexer, runstop relay, communication board for the ESC, and servo glitch capacitor. The GPS box contains the GPS board; a Cui 3.3-V, 10-W isolated power supply; a small fan; and the GPS antenna mounted to the ground plane (which is the lid). Front-brake disk aligners and mounts for the rear-wheel rotation sensors and magnets are installed on the chassis. The front-brake disk aligners are needed to keep the disks rotating smoothly because the front-wheel rotation sensor magnets unbalance the disks if left to freely rotate.

Compute Box

Most modern control and perception algorithms are central processing unit and graphics processing unit (GPU) intensive. To maximize performance and reduce hardware development and software optimization time, the compute box employs standard components instead of specialized embedded hardware typical of scaled, autonomous platforms. The compute box design provides a robust enclosure that mounts to the chassis and fits inside the stock protective body. The weight of the empty compute box is 3.3 kg (and 8.8 kg with all components installed).

Enclosure Design

The enclosure (see Figure 4) is designed to withstand a 10-g direct impact from any angle without damaging internal electronic components and is fabricated out of a 2.286-mm-thick 3003 aluminum sheet. The 10-g impact is larger than
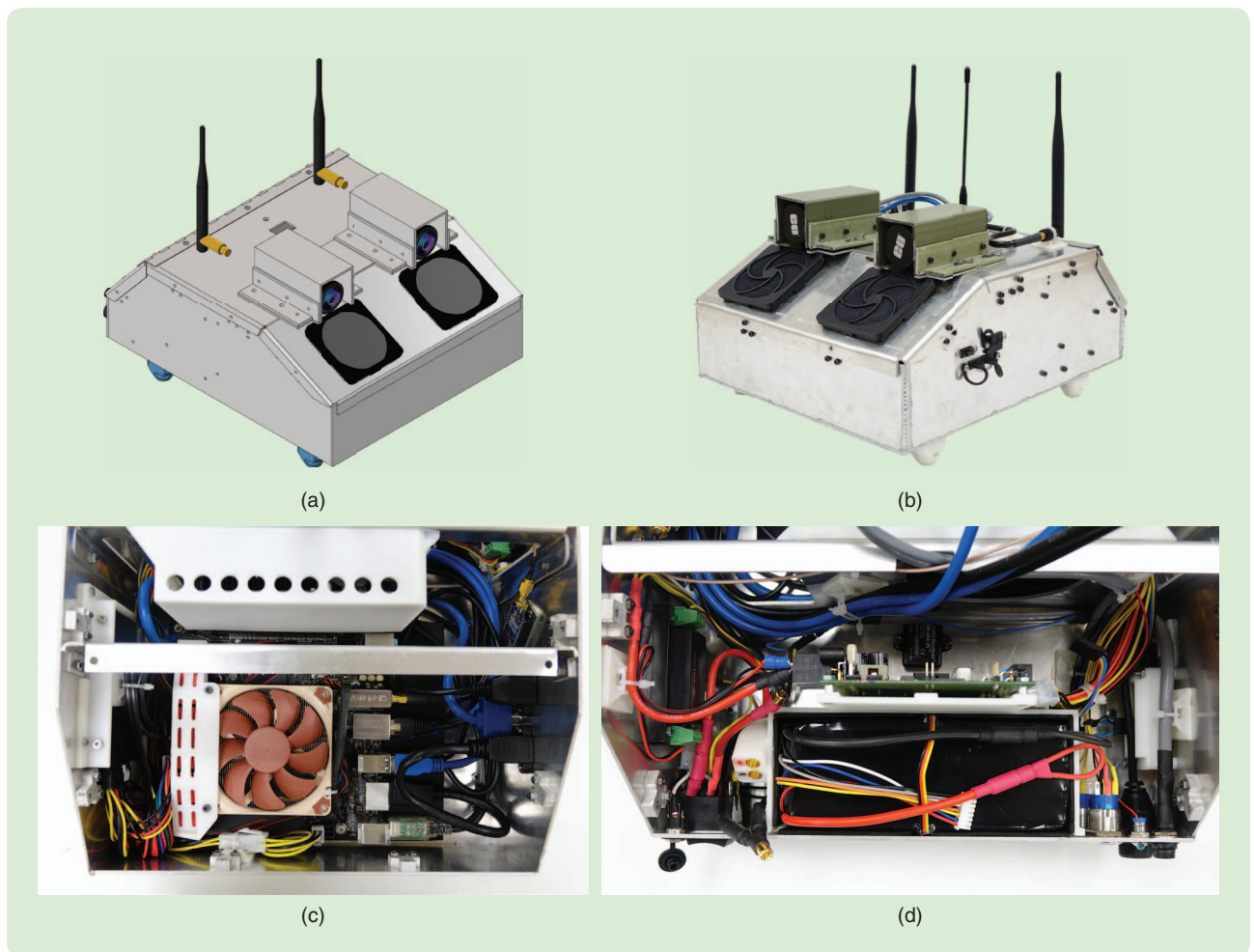
FIGURE 4 The AutoRally mini-ITX compute box. (a) The assembled computer-aided design model. (b) The fully assembled compute box ready to be mounted on a chassis. (c) The front of the compute box viewed from above with motherboard and compute components visible. (d) The rear of the compute box viewed from above with the power system components visible.

those experienced when testing at GT-ARF according to IMU data that included collisions with fixed objects and rollovers. The box's impact tolerance was verified using finite element analysis of the CAD model before fabrication. The 3003 aluminum alloy was chosen for its strength, ductility, and relatively light weight. The sides of the box are tungsten inert gas welded to the bottom to accurately join the large panels of relatively thick aluminum sheets without leaving gaps. Aluminum dust filters coupled with a foam membrane allow continuous airflow through the box while keeping environmental contaminants such as dust and rocks out. Combined with the all-aluminum exterior, the assembled compute box provides excellent electromagnetic interference containment.

The cameras and lenses are mounted facing forward on top of the box on an aluminum plate for rigidity, and they are protected by covers made from structural fiberglass that does not affect the signal quality for the antennas mounted on the top of the box. Each camera cover is secured to the compute box with four clevis and cotter pins for quick access to the lens and cameras as needed.

Four 3D-printed components are inside the compute box: a battery holder, solid-state drive (SSD) holder, GPU holder, and random access memory holder. The battery holder tightly secures the compute box battery and power supply. The battery slot is slightly undersized and lined with foam so that the battery press fits into the mount and can be removed for charging and maintenance without removing any internal screws. The SSD holder is used to securely mount a 2.5-in SSD to the sidewall of the compute box. The GPU holder fits over the GPU and secures it to the main internal strut while still allowing adequate airflow.

The compute box attaches to the chassis with four 3D-printed mounts attached to the bottom of the compute box. The mounts fit over vertical posts on the chassis rail guards and are secured to the chassis with a cotter pin though the mount and post. Special consideration was given to design the mounts as break-away points for the compute box in the event of a catastrophic crash. The mounts are easy and inexpensive to replace and break away before any of the aluminum compute box parts fail to protect the electronics within the compute box. By applying lateral forces with finite element analysis and the CAD models, the failure point of the mounts is designed to be at 8 g of force on the compute box, compared to the 10-g design load for the rest of the compute box. In practice, the compute box mounts break away during hard rollover crashes, leaving the internal components undamaged. All panel mount components such as the power button,

light-emitting diodes, and ports are dust resistant or protected with a plug to keep out debris.

Sensors

A Lord Microstrain 3DM-GX4-25 IMU provides raw acceleration and angular rate data at 200 Hz (maximum 1 kHz) and fused orientation estimates at 200 Hz (maximum 500 Hz). The two machine-vision cameras that are mounted on top of the compute box are Point Grey Flea3 FL3-U3-13E4C-C color cameras with a global shutter that operates up to 60 Hz. Lenses are 70° FOV, with a 4.5-mm fixed focal length. Each camera connects to the motherboard with a USB 3.0 cable and is externally triggered by an Arduino Micro microcontroller with a general-purpose input–output connector. Both cameras are connected to the same trigger signal that operates at a configurable rate. Internal battery voltage and computer temperature sensors are used to monitor system health.

Computing

A modular, reconfigurable, onboard computing solution was designed that uses standard consumer computer components based on the Mini-ITX form factor. Computing hardware development outpaces advancements in almost all other components, so the standard form factor, mounting method, and data connections enable the reconfiguration of sensing and computing payloads without mechanical modifications as requirements evolve. Table 2 lists the details of

TABLE 2 AutoRally compute box components. Major computing and power parts are listed with their specifications.

Component	Detail
Motherboard	Asus Z170I Pro Gaming, Mini-ITX
Central processing unit	Intel i7-6700, 3.4 GHz quad-core 65 W
Random access memory	32 GB DDR4, 2133 MHz
Graphics processing unit	Nvidia GTX-750ti SC, 640 cores, 2 GB, 1176 MHz
Solid-state drive storage	512 GB M.2 and 1 TB SATA3
Wireless	802.11ac Wi-Fi, 900 MHz XBee, and 2.4 GHz RC
Power supply	Mini-Box M4-ATX, 250 W
Battery	22.2 V, 11-Ah lithium-polymer, 244 Wh

the compute box components. Wi-Fi is used to remotely monitor high-bandwidth, non-time-critical data from the platform, such as images and diagnostic information. A 900-MHz XBee Pro provides a low-latency, low-bandwidth wireless communication channel. The GPS on the robot receives RTK corrections from the GPS base station (transmitted over the XBee radio) at approximately 2 Hz to improve GPS performance. The XBee radio on board the robot also receives a global software runstop signal at 5 Hz and the position and velocity of other AutoRally robots within communication range at up to 10 Hz.

The base station XBee, connected to the same computer as the base station GPS, transmits the software runstop and RTK correction messages to all AutoRally robots within communication range. The runstop message allows all robots within radio range (each running its own self-contained software system) to be stopped simultaneously with one button.

SOFTWARE INTERFACE

The AutoRally software was designed to leverage existing tools wherever possible. All computers in the system run the latest long-term support version of Ubuntu Desktop to take advantage of the wide availability of compiled packages and minimal configuration requirements. All AutoRally software is developed using the Robot Operating System (ROS) [17], a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that simplifies

the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Custom ROS interface programs were developed for each AutoRally component that lacked a publicly available interface. The time synchronization and safety systems presented in this section are critical components often overlooked in other scaled platforms. They are a combination of electronics and software that enables a safe and robust autonomous system. The software interface, OCS graphical user interface (GUI), and simulation environment for the robot are also presented.

Time Synchronization

Distributed system design requires robust time synchronization across all components in the system. Accurate timing is especially important as asynchronous data and control rates increase. Time synchronization is performed within the AutoRally system on all computing and sensing components with a combination of Ubuntu tools. Figure 5 shows how timing information is propagated for time synchronization. The time source for the entire system is the GPS board on the chassis that emits National Marine Electronics Association (NMEA) 0183 messages and a pulse-per-second (PPS) signal. The PPS signal provides a marker accurate to within a few nanoseconds of the start of every second according to GPS time. NMEA 0183 time messages corresponding to each PPS pulse provide timing information about that pulse. NMEA 0183 and PPS signals are widely
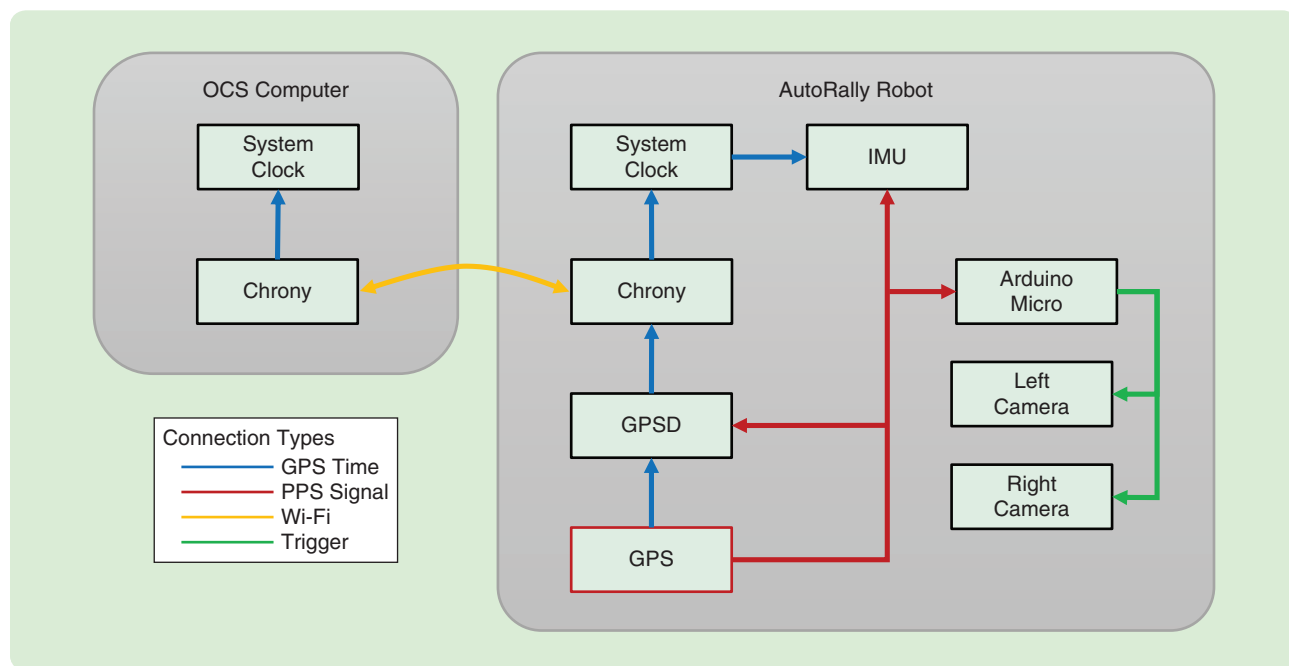
FIGURE 5 The AutoRally time synchronization diagram. The clocks on the computers and sensors that support clock control are synchronized to Global Positioning System (GPS) time with a combination of the pulse-per-second (PPS) signal that marks the beginning of a second and timing messages that identify which second the PPS signal represents. On the AutoRally robot and operator control station (OCS) computers, the system utilities GPS service daemon (GPSD) and Chrony are used for clock synchronization. The two cameras rely on an external trigger signal to capture frames simultaneously. The Arduino Micro microcontroller generates the camera trigger signal. IMU: inertial measurement unit.

supported by devices that require time synchronization. The PPS signal is routed into a GPS service daemon (GPSD) running on the motherboard, IMU, and Arduino Micro.

GPSD is a daemon used to bridge GPS time sources with traditional time servers. GPSD runs on the compute box and receives the GPS PPS signal and NMEA messages. Processed timing information is communicated through a low-latency shared memory channel to Chrony, the time server running on the computer. Compared to traditional Network Time Protocol servers, Chrony is designed to perform well in a wide range of conditions including intermittent network connections, heavily congested networks, changing temperatures, and systems that do not run continuously. Chrony's control of system time makes time synchronization transparent to programs running on the computer. The system time of the OCS computer is synchronized to the AutoRally robot by a second Chrony instance on the OCS computer that communicates over Wi-Fi with Chrony on the robot.

The IMU provides a dedicated pin for a PPS input. In addition to the PPS signal, it requires the current GPS second (GPS time is given in seconds since January 6, 1980) to resolve the time of the PPS pulse. This value can be derived from the computer's system clock. The IMU uses these two pieces of information to synchronize its own clock and time-stamp each measurement with an accuracy of significantly lower than 1 ms to system time.

The cameras provide an external trigger interface to control when each image is captured. The Arduino Micro provides the cameras with the triggering pulse at a specified frame rate. Each time a PPS pulse arrives from the GPS, a train of evenly spaced pulses at the rate specified in the ROS system is sent to the cameras. The cameras' images are time stamped with the system time when they are received by the computer.

Safety System

The three-layer AutoRally safety system is designed to remotely disable robot motion in the event of any software or electronics failure. The three layers, shown in Figure 6, are a wireless dead-man relay located in the electronics box
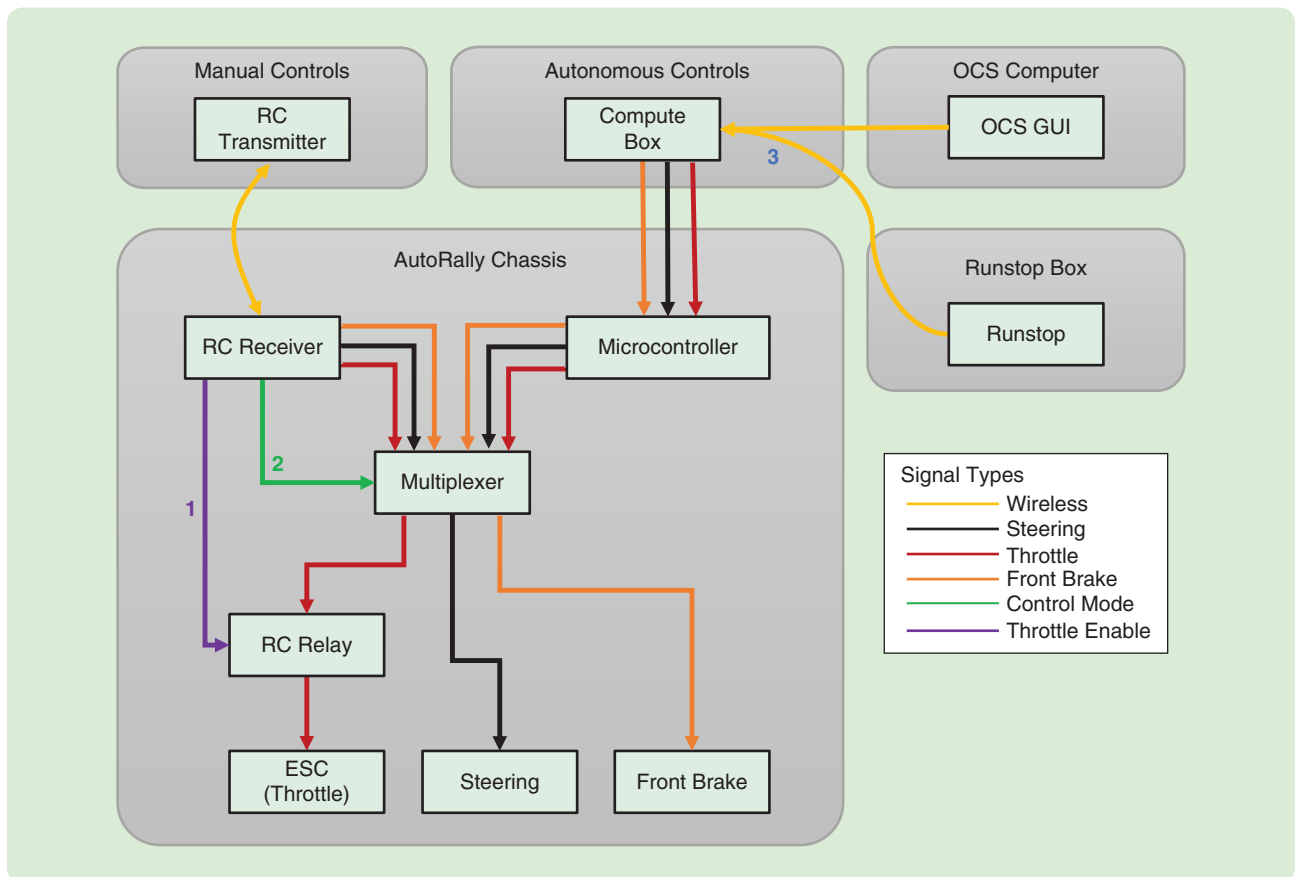

FIGURE 6 The AutoRally safety system. The human-operated radio-controlled (RC) transmitter sends signals to the RC receiver in the AutoRally chassis. The RC receiver provides actuator signals from the human driver, controls a safety relay to enable and disable the throttle signal into the electronic speed controller (ESC), and switches between human and computer control. Information relating to the state of the safety system is presented to the human operator in the operator control station (OCS) graphical user interface (GUI). Layer 1 of the safety system, shown in purple, is the throttle relay that acts as a wireless throttle live-man switch. Layer 2, shown in green, allows seamless, remote switching between autonomous and manual control modes. Layer 3, shown in blue and yellow, is the software runstop used to disable autonomous motion.

to disconnect the throttle signal, remote switching between autonomous and manual control with a PWM signal multiplexer, and a software-based runstop message. The relay and autonomous/manual modes are controlled by the state of buttons on the transmitter that circumvent the Wi-Fi, XBee, and software control on the compute box using the additional radio frequency link between the RC transmitter and receiver located in the electronics box of the chassis. The dead-man relay monitors the quality of the incoming PWM control signal so that the throttle signal is automatically disabled in the event of a signal failure between the transmitter and receiver. Additionally, the throttle signal is connected through the normally open contact of the dead-man relay, so that the throttle signal disengages in the case of a power failure on the robot.

Runstop is implemented in software by the AutoRally chassis interface program, shown in Figure 7, using incoming runstop ROS messages to enable and disable software control of the robot. Any program in the AutoRally system can publish a runstop ROS message. The chassis interface determines whether autonomous control is enabled with a bitwise OR operation of the most recently received runstop message from each message source. By default, the OCS GUI and runstop box send runstop messages. The OCS runstop message is controlled by a button in the GUI and transmitted over Wi-Fi from the OCS computer to the robot. The runstop box sends a runstop message, controlled by the button state of the runstop box, over XBee to the robot. Data transmitted over the base station XBee are delivered to every robot within communication range. This means that, even though there could be multiple AutoRally robots running at the same time (each with its own self-contained ROS system), the runstop box signal simultaneously controls autonomous motion for all of the robots. The AutoRally
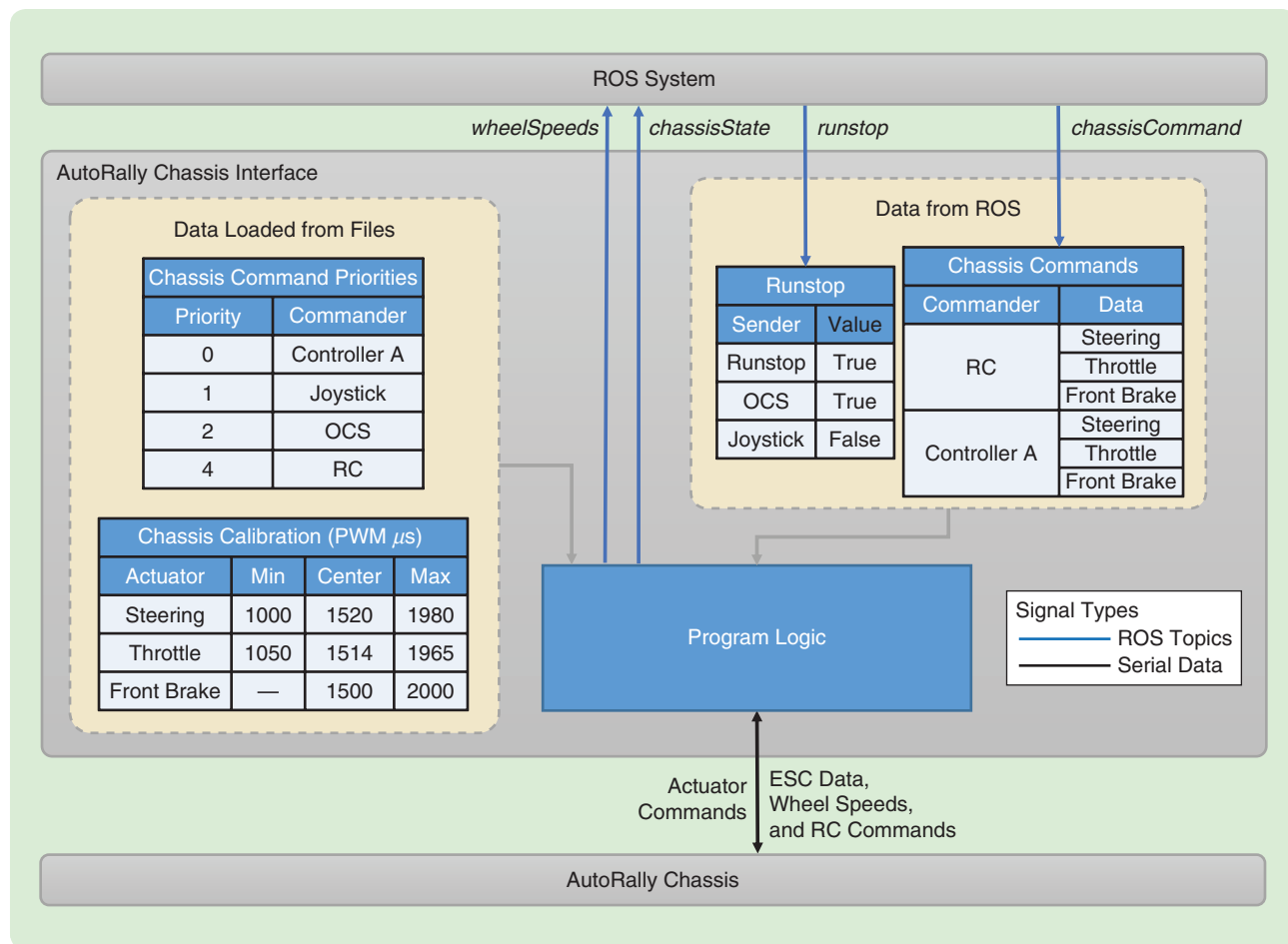
FIGURE 7 The AutoRally chassis interface program information flow. The program, which runs on the compute box on board the robot, uses a combination of configuration files loaded at runtime and messages arriving from the Robot Operating System (ROS) interface to send the highest-priority actuator commands over a universal serial bus connection to the microcontroller in the AutoRally chassis. Simultaneously, the AutoRally chassis sends state information back to the chassis interface program that includes electronic speed controller (ESC) data, wheel speeds, and human-provided actuator commands from the radio-controlled (RC) receiver. The information received from the chassis is published into the ROS system and can be viewed in the operator control station (OCS) graphical user interface. PWM: pulsewidth modulation.

robot does not have a true emergency stop that disconnects actuator power because the size, cost, and power requirements for such a system do not fit within the current package. In practice, the three-layer AutoRally safety system allows an operator to disable motion and assume manual control of the platform without delay.

AutoRally Chassis Interface

The AutoRally chassis interface software is implemented as an ROS nodelet and communicates with the microcontroller in the chassis electronics box through a USB cable. The interface sends actuator commands to the chassis and receives chassis state information including wheel speeds, the human-provided control commands read from the RC receiver, ESC diagnostic information, and safety system-state information.

The throttle, steering, and front brake of the robot are controlled by 50-Hz PWM signals standard in the hobby RC community. The AutoRally chassis software interface provides a calibration layer above the PWM signal for standardization across platforms and to prevent physical damage so commands do not exceed the mechanical limits in the steering and brake linkages. The chassis calibration is stored in a file loaded at runtime by the chassis interface software. Specified in the file is the minimum, center, and maximum pulse width for each actuator in microseconds. When properly calibrated, a /chassisCommand ROS message on any AutoRally platform will elicit the same behavior. For example, commanding a steering value of zero will make any calibrated AutoRally platform drive in a straight line. Valid actuator command values in the /chassisCommand message are between [–1, 1]. A steering value of –1 will turn the steering all the way left, and a value of one will steer all the way right. A throttle value of –1 is full (rear) brake and 1 is full throttle. The front brake value ranges from zero for no brake to one for full front brake, while negative values are undefined.

On startup, the chassis interface loads a priority list of controllers from a configuration file. The priority list is used while operating to determine which actuator commands arriving from various controllers are sent to the actuators. The priorities encode a hierarchy of controllers and define a mechanism to dynamically switch among controllers and use multiple ones simultaneously. This system allows high-priority controllers to subsume control from lower-priority controllers, as desired. Additionally, each actuator can be controlled by a separate controller, such as a waypoint following controller for the steering and a separate velocity controller for the throttle and front brake.

Operator Control Station

The OCS GUI is a tab-based program built using Qt that presents real-time diagnostic information, debugging capabilities, and a software runstop to a remote human operator for the AutoRally robot. Wheel-speed data, real-time images from the onboard cameras, and all diagnostic messages from the ROS /diagnostics topic that contain detailed information about the health of running nodes are displayed. Diagnostic messages are color coded by status and grouped by source for fast status recognition by the human operator. All of the data between the OCS GUI running on a laptop and the robot travel over a local Wi-Fi network.

The OCS GUI also provides an interface for direct control of the robot's actuators via sliders. While this interface is not appropriate for driving the car, it is used to debug software and hardware issues related to the actuators.

Simulation

Despite the robust AutoRally hardware platform, there are still high-risk maneuvers and software testing best run in a simulation environment before executing them on the physical platform. The simulation also allows the careful control of environmental parameters for gathering statistical data, which requires performing repetitive or time-consuming experiments that would take weeks or more of testing on the physical platform.

The simulation environment, shown in Figure 8(b), is based on Gazebo, a robot simulator with tight ROS integration that includes graphical interfaces and multiple physics engines to choose from. The AutoRally Gazebo simulation environment and robot model match their real-world counterparts and support the same software interface through ROS messages as the real hardware. The simulated track environment is the same size and configuration as GT-ARF. The steering servo and Ackermann linkage of the physical robot are approximated by ROS joint effort controllers that apply torque to turn each front wheel about the vertical axis. The no-load rotation speed, maximum torque, and joint limits used in the simulation are measured from the steering servo specification provided by the manufacturer and by the steering linkage angles relative to the chassis center line. The powertrain is approximated by another ROS effort controller that applies torque on the rear axle of the Gazebo model. The
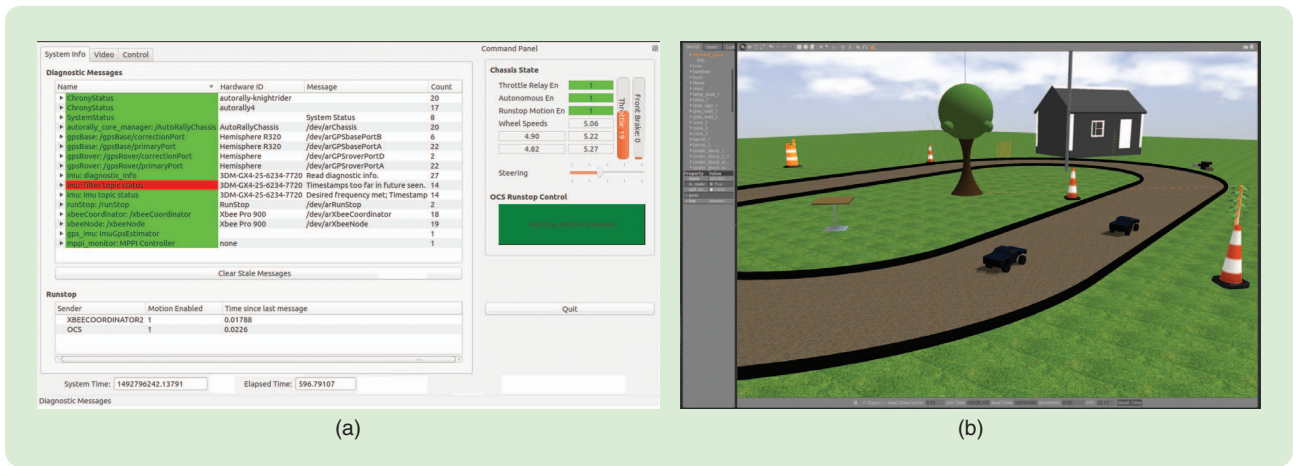
FIGURE 8 The operator control station (OCS) graphical user interface (GUI) and simulation environment. (a) The OCS GUI showing diagnostic, sensor, and actuator information captured during autonomous testing. (b) The simulation environment built in Gazebo is modeled after the Georgia Tech Autonomous Racing Facility. Multiple AutoRally robots can be simulated together, and each simulated platform has the same Robot Operating System messaging interface and simulated sensors as the physical AutoRally platform. All simulation vehicle parameters (such as mass, moments of inertia, and sensor placement and characteristics) are set according to their experimentally determined values from a physical robot.

maximum applied torque and angular velocity are calculated from the motor manufacturers' specifications. The differential in the physical platform is neglected in the simulation. The suspension for each wheel is modeled with a proportional-integral-derivative controller on a linear actuator with a target setpoint, which determines the ride height of the vehicle. The I and D terms are calculated from the dimensions and coefficients of the robot's spring configuration.

The simulation and physical platform implement identical ROS messaging interfaces to enable seamless software migration between hardware and simulation. Simulated GPS, IMU, and cameras are from the hector_gazebo_plugins ROS package from Technische Universität Darmstadt [18], and they are configured according to the specifications of their physical analogs. We developed our own wheel-speed sensor node because no similar functionality was publicly available.

Overall, Gazebo is not considered a high-fidelity simulator with respect to graphics rendering and physics realism for autonomous vehicles, especially as the vehicle approaches and surpasses the friction limits of the system. The main reason for using Gazebo as the simulator was not to produce the most accurate visuals and dynamics but, rather, as part of the hardware and software infrastructure that allows for smooth software testing with ROS and the AutoRally platform and to easily debug the control, perception, and communication software.

ESTIMATION

Parameter estimation is an essential part of controller design, especially for model-based controllers such as MPC, which rely on accurate dynamics models for motion prediction. This section details the offline and online estimation performed with the AutoRally platform. Offline, parameters were estimated to determine the platform moments of inertia (MOI) using the bifilar pendulum method. Three different vehicle models of increasing fidelity are presented. While the higher-fidelity models can be used to more accurately predict vehicle motion, the model parameters can be significantly more difficult to estimate and computationally expensive to compute. A joint-state unscented Kalman filter (JS-UKF) was used to determine the parameters of single-track and double-track vehicle models with a realistic tire forces model. An 11 degrees-of-freedom (DOF) full-vehicle model was estimated using an adaptive limited-memory unscented Kalman filter (ALM-UKF). Online, vehicle state is estimated using a factor graph-based optimization framework with GPS and IMU data, and a cost map of the terrain (similar to a traversability grid) is generated from monocular camera images for use in a stochastic MPC framework.

Moment of Inertia Estimation with the Bifilar Pendulum

The MOI of the platform is more difficult to determine than other parameters (such as the mass of the vehicle) but is important for physics-based controllers. Modern CAD software can automatically compute the MOI if an accurate model exists. No complete CAD model of AutoRally is available, so the platform MOI cannot be determined with these tools. Methods also exist to compute MOI by precomputation [19], [20] or online estimation [21] in cases where the full model is unknown or changing. An extensive survey of popular methods for experimentally determining MOI is presented in [22]. Many methods
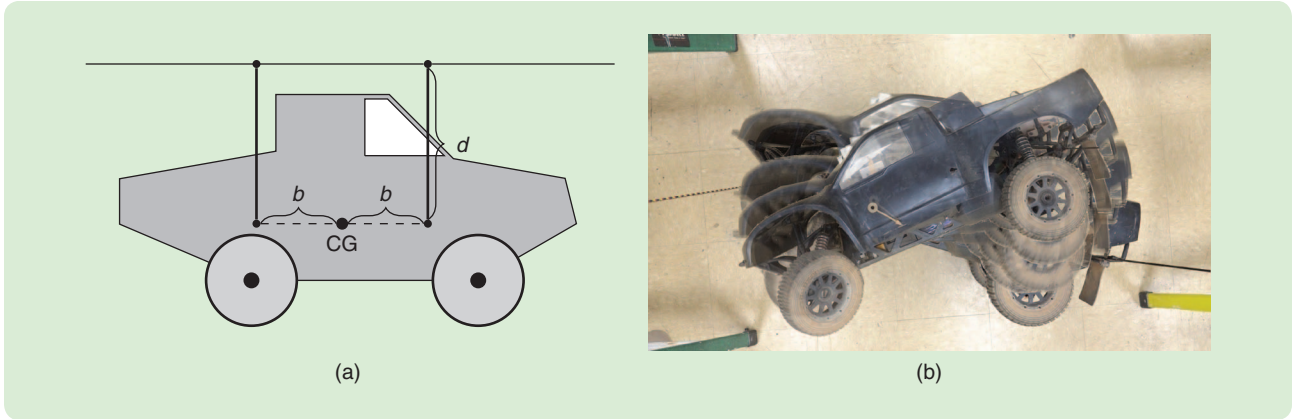
FIGURE 9 The bifilar pendulum setup for experimental determination of moments of inertia (MOI) with AutoRally. (a) An example setup with labeled support strings of length d, attached equidistant from the center of gravity (CG) for computing the yaw MOI, I_ψ. (b) A composite image of a one-half oscillation period for computing the pitch MOI I_θ.

rely on custom calibration rigs that are time-consuming, expensive, and difficult to build. In this work, we computed the necessary MOI experimentally using the bifilar pendulum method [22]. As shown in Figure 9(a), two fixed parallel cables are attached equidistant from the center of gravity of the body to isolate the desired calibration axis. The body is rotated by a small angle around the desired axis and then released to freely oscillate. Given the dimensions of the test rig and known dimensions and weights of the robot, the period of a free oscillation after an excitation determines the MOI about that axis. The equation for the MOI about a single axis using the bifilar pendulum method from [22] is

$$I = m \frac{g R_1 R_2}{4\pi^2 h} T^2, \qquad (1)$$

where m is the platform mass (including wheels), T is the oscillation period, h is the distance of the calibration object from the support (taking into account nonvertical support wires), and R_1 and R_2 are the distances from the center of gravity to the support wire attachment. The change in height from the mounting location as the vehicle is rotated is assumed zero when the angle of rotation is small. Our setup is simplified using parallel support wires, $R_1 = R_2 = b$, so that

$$h = \sqrt{d^2 - (R_1 - R_2)^2} = d, \qquad (2)$$

which makes (1) for our setup

$$I = m \frac{g b^2}{4\pi^2 d} T^2. \qquad (3)$$

Single-Track Vehicle Model
Three different vehicle models of increasing fidelity were used to test and compare the results. Here, we briefly

summarize each model. The performance of each model against experimental data is given in the section "Experimental Results." First, the single-track vehicle model [23]–[25] used for the AutoRally vehicle is described, which considers the longitudinal and lateral displacement, as well as the yaw motion of the vehicle, as shown in Figure 10.
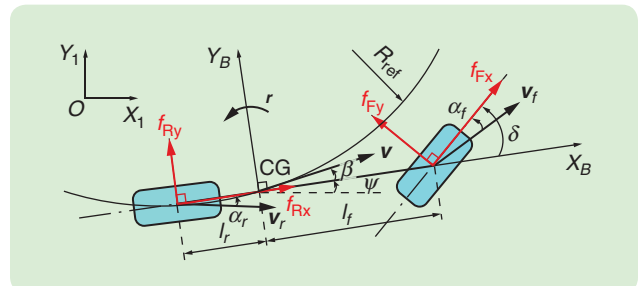
X_I-O-Y_I and X_B-CG-Y_B (where CG is the center of gravity) denote the inertial frame fixed on the ground and the body frame fixed on the vehicle, respectively. The equations of motion of the model can be expressed in a body-fixed frame with the origin at the CG from [23]

$$\dot{V}_x = (f_{Fx}\cos\delta - f_{Fy}\sin\delta + f_{Rx})/m + V_y\dot{\psi}, \qquad (4a)$$

$$\dot{V}_y = (f_{Fx}\sin\delta + f_{Fy}\cos\delta + f_{Ry})/m - V_x\dot{\psi}, \qquad (4b)$$

$$\dot{r} = [(f_{Fy}\cos\delta + f_{Fx}\sin\delta)\ell_f - f_{Ry}\ell_r]/I_z, \qquad (4c)$$

where V_x and V_y are the components of V along the X_B and Y_B directions, respectively; m is the total mass; and I_z is the moment of inertia of the vehicle about the vertical axis. f_{ij} ($i = F, R$ and $j = x, y$) denote the longitudinal and lateral friction forces at the front and rear wheels,


FIGURE 10 The single-track vehicle model. This model includes the longitudinal and lateral displacement as well as the yaw motion of the vehicle. We use X_I-o-Y_I and X_B-CG-Y_B (where CG is the center of gravity) to denote the inertial frame fixed on the ground and the body frame fixed on the vehicle, respectively.

ψ denotes the yaw angle, and δ is the steering angle of the front wheel.

Double-Track Model

The double-track model considers the longitudinal, lateral, and yaw motion of the vehicle and accounts for the load difference between the left and right wheels arising from the lateral load transfer.
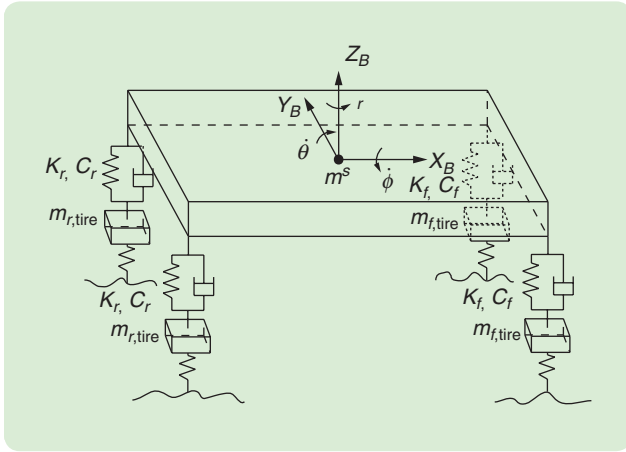
FIGURE 11 The riding model. K_i and C_i ($i = f, r$) denote the spring stiffness and damping coefficient of the suspension system related to each wheel, $m_{i,\text{tire}}$ ($i = f, r$) denotes the mass of the front and rear tire, respectively, m^s is the sprung mass, and $\dot{\varphi}$ and $\dot{\theta}$ are the rolling and pitching rate, respectively.

Let $f_{i,j,k}$ ($i = L, R, j = L, R$ and $k = x, y$) denote the longitudinal or lateral friction force for each wheel, respectively. The vehicle's equations of motion are then

$$\dot{V}_x = [(f_{LFx} + f_{RFx})\cos\delta - (f_{LFy} + f_{RFy})\sin\delta + f_{LRx} + f_{RRx}]/m + V_y\dot{\psi} \tag{5a}$$

$$\dot{V}_y = [(f_{LFx} + f_{RFx})\sin\delta + (f_{LFy} + f_{RFy})\cos\delta + f_{LRy} + f_{RRy}]/m - V_x\dot{\psi}, \tag{5b}$$

$$\dot{r} = \{[(f_{LFy} + f_{RFy})\cos\delta + (f_{LFx} + f_{RFx})\sin\delta]\ell_f - (f_{LRy} + f_{RRy})\ell_r\}/I_z. \tag{5c}$$

Full-Vehicle Model

The full-vehicle model considers the dynamics of the sprung and unsprung mass of the vehicle separately and is derived using Newton–Euler equations for the motion of rigid body systems. The equations of motion for the total mass are the same as (5a)–(5c) for the double-track model. The air resistance is also considered, and (5a) is modified such that

$$\dot{V}_x = [(f_{LFx} + f_{RFx})\cos\delta - (f_{LFy} + f_{RFy})\sin\delta + f_{LRx} + f_{RRx}]/m + V_y\dot{\psi} - C_D\rho_{air}AV_x^2/2, \tag{6}$$

where C_D is the air resistance coefficient, ρ_{air} is the air density, and A is the frontal area of the vehicle. The vertical translation is accounted for by a riding model, as shown in Figure 11. The rolling and pitching model is given in Figure 12.
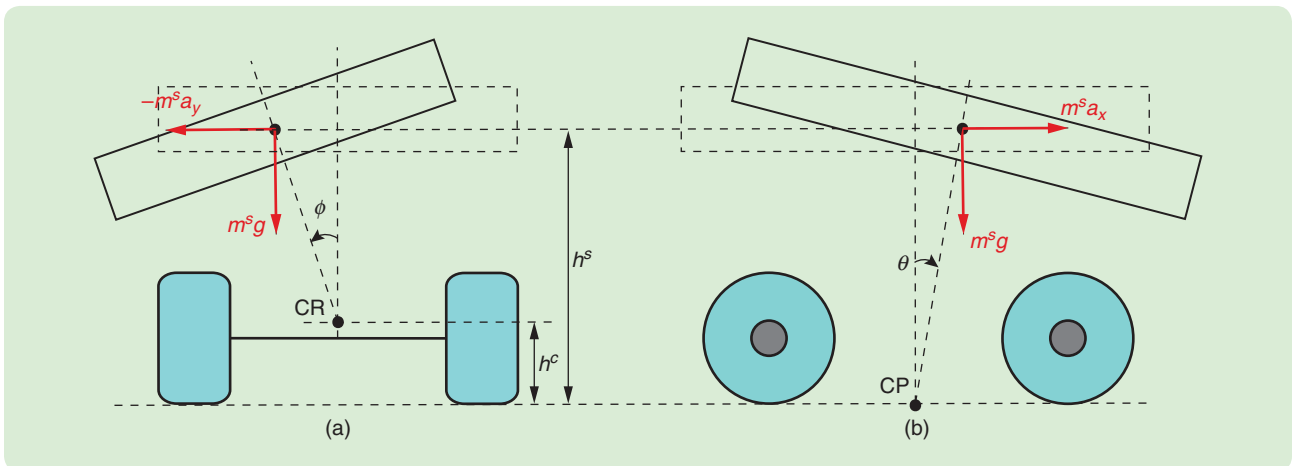
FIGURE 12 The rolling and pitching model. (a) The rolling motion arises from the lateral acceleration and the gravity center offset from the rolling center. The parameters h^s and h^c are the heights of the sprung mass center, CR is the rolling center, and φ is the roll. (b) The pitching motion arises from the longitudinal acceleration and the gravity center offset of the pitching center (CP) that is assumed to be on the ground. θ is the pitch.

In Figure 11, K_i and C_i $(i = \text{f, r})$ denote the spring stiffness and the damping coefficient of the suspension system related to each wheel, $m_{i,\text{tire}}$ $(i = \text{f, r})$ denotes the mass of the front and rear tire, respectively, m^s is the sprung mass, and $\dot\varphi$ and $\dot\theta$ are the rolling and pitching rate, respectively.

Figure 12(a) shows the rolling motion from the lateral acceleration and gravity center offset from the rolling center. The parameters h^s and h^c are the heights of the sprung mass center and rolling center, respectively. Figure 12(b) shows the pitching motion arising from the longitudinal acceleration and the gravity center offset from the pitching center assumed to be on the ground. The dynamic equations of the vertical, rolling, and pitching motion of the sprung mass are

$$\dot V_z^\text{s} = \left(-2(K_\text{f} + K_\text{r})\theta - 2(C_\text{f} + C_\text{r})V_z^\text{s} + 2(\ell_\text{f}K_\text{f} - \ell_\text{r}K_\text{r})\varphi \right.$$
$$\left. + 2(\ell_\text{f}C_\text{f} - \ell_\text{r}C_\text{r})\dot\theta\right)/m^\text{s}, \qquad (7a)$$

$$\ddot\theta = \left[2(\ell_\text{f}K_\text{f} - \ell_\text{r}K_\text{r})z^\text{s} + 2(\ell_\text{f}C_\text{f} - \ell_\text{r}C_\text{r})V_z^\text{s} - 2(\ell_\text{f}^2 K_\text{f} + \ell_\text{r}^2 K_\text{r})\theta \right.$$
$$\left. - 2(\ell_\text{f}^2 C_\text{f} + \ell_\text{r}^2 C_\text{r})\dot\theta + m^\text{s}gh^\text{s}\sin\theta + m^\text{s}a_x^\text{s}h_\text{s}\cos\theta\right]/I_y^\text{P}, \quad (7b)$$

$$\ddot\varphi = \left[-w_\text{f}^2 K_\text{f}\varphi/2 - w_\text{f}^2 C_\text{f}\dot\varphi/2 - w_\text{r}^2 K_\text{r}\varphi/2 - w_\text{r}^2 C_\text{r}\dot\varphi/2 \right.$$
$$\left. + m^\text{s}g(h^\text{s} - h^\text{c})\sin\varphi + m^\text{s}a_y^\text{s}(h^\text{s} - h^\text{c})\cos\varphi\right]/I_x^\text{R}, \qquad (7c)$$

where w_i $(i = \text{f, r})$ denotes the front and rear track, respectively; a_x^s and a_y^s are the longitudinal and lateral acceleration of the sprung mass center in the body-fixed frame; and I_x^R and I_y^P are the MOI of the sprung mass about the rolling axis and pitching axis, respectively. Note that in this derivation, the small-angle assumption for ϕ and θ is considered, which is a standard assumption.

Tire Force Model

Tire models are required to model the complex interactions of the vehicle tires with the ground. They are especially important in the driving regime of this work, where the vehicle frequently slides on a dirt surface and is almost always accelerating. A common tire model (see Figure 13) and the one used here are based on Pacejka's magic tire formula (MF). The important aspect of this model is its ability to capture the tire saturation (that is, the coupling between the lateral and longitudinal axes). According to this model, the generated tire force depends on the tire slip. Tire slip is defined by the nondimensional relative velocity instead of the angle (as is sometimes used) of each tire with respect to the road, along the longitudinal and lateral directions

$$s_{ijx} = \frac{V_{ijx} - \omega_{ijx}R_j}{\omega_{ijx}R_j}, \quad s_{ijy} = \frac{V_{ijy}}{\omega_{ijx}R_j}, \qquad (8)$$

where $i = L, R$ correspond to the left and right side, $j = F, R$ correspond to the front and rear, and V_{ijk} $(k = x, y)$ is the tire frame component of the vehicle velocity of each tire. The total slip of each tire is defined by $s_{ij} = \sqrt{s_{ijx}^2 + s_{ijy}^2}$, and the total friction coefficient related to each tire is calculated using the MF from [23]

$$\mu_{ij} = D\sin\{C \operatorname{atan}[BS_E - E(BS_E - \operatorname{atan}S_E)]\} + S_\text{v}, \qquad (9)$$

where B, C, D, and E are the stiffness, shape, peak, and curvature factors, respectively; $S_E = s_{ij} - S_\text{h}$, where S_h is the horizontal shift; and S_v is the vertical shift. The tire friction force components are

$$f_{ijk} = -\frac{s_{ijk}}{s_{ij}}\mu_{ij}f_{ijz}, \quad i = L, R; \quad j = F, R; \quad k = x, y, \qquad (10)$$

where f_{ijz} is the normal load on the corresponding tire and can be calculated from [23].

Unscented Kalman Filter

To determine the unknown vehicle parameters (including the parameters for the tire force model), a JS-UKF was used that includes the unknown parameters in the original state vector and estimates the new augmented state. In this framework, the state and noise are assumed to be Gaussian random variables. Recall that for a system

$$x_{k+1} = f(x_k, u_k) + w_k, \quad y_k = h(x_k, u_k) + v_k, \qquad (11)$$

where $w_k \sim \mathcal{N}(q, Q)$ and $v_k \sim \mathcal{N}(r, R)$ are Gaussian process and measurement noise, respectively, the extended Kalman
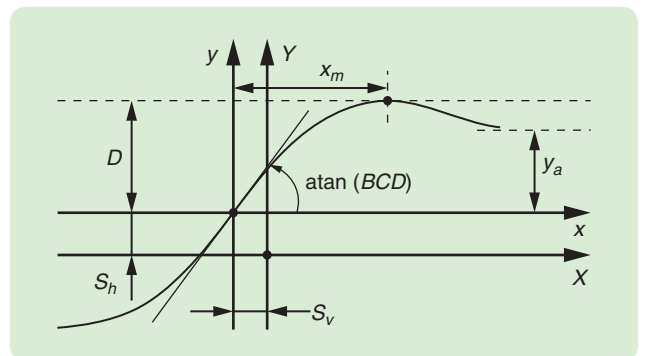
FIGURE 13 Pacejka's magic formula tire model. The graph shows slip angle (x-axis) versus normalized lateral force (y-axis). Parameters B, C, and D are the stiffness, shape, and peak factors, respectively. S_h is the horizontal shift, and S_v is the vertical shift.

filter (EKF) propagates the Gaussian random variable x_k by linearizing the nonlinear state transition (observation) function $f : \mathbb{R}^n \times \mathcal{U} \mapsto \mathbb{R}^n$ $(h : \mathbb{R}^n \times \mathcal{U} \mapsto \mathbb{R}^m)$ with the Jacobian matrix at each time step k [26]. Instead of an EKF, this work used an UKF filter since

1) the UKF propagates the Gaussian random variable through a nonlinear function more accurately than the EKF
2) the UKF avoids calculating the Jacobians that may be too cumbersome for highly nonlinear systems.

Adaptive Limited Memory Unscented Kalman Filter

A UKF is based on the unscented transformation (UT) and avoids calculating the Jacobian matrices at each time step. Assuming an L-dimensional Gaussian random variable x with mean $\hat{x}$ and covariance P_x, to calculate the statistics of $y = g(x)$, one selects $2L + 1$ discrete sample points $\{\mathcal{X}_i\}_{i=0}^{2L}$, which are propagated through the system dynamics. The UKF redefines the state vector as $x_k^a = [x_k^T, w_k^T, v_k^T]^T$, which concatenates the original state and noise variables, and then estimates x_k^a recursively [27].

The ALM-UKF, introduced in [28], is an estimation algorithm for nonlinear systems that builds on previous works on the UKF to simultaneously estimate the system state, model parameters, and Kalman filter hyperparameters related to the noise. First, recall that the adaptive Kalman filter algorithm [29] adjusts the mean and covariance of the noise online, which is expected to compensate for time-varying modeling errors. Define the set of unknown, time-varying hyperparameters for the Kalman filter corresponding to the noise statistics at the ith time step as $\mathcal{S}_i \triangleq \{q_i, Q_i, r_i, R_i\}$. $\mathcal{S}_i$ is estimated simultaneously with the system state and parameters. Since an optimal estimator for $\mathcal{S}_i$ is unavailable and many suboptimal schemes are either too restrictive for nonlinear applications or too computationally demanding [30], [31], this work adopts the adaptive limited memory algorithm in [29], with the following two extensions:

1) The algorithm is developed for a nonlinear application (that is, the UKF).
2) The unknown parameters of the system are estimated along with the state (instead of just the system state).

In the following, it is assumed that $\mathcal{S}_i$ is constant and denoted by $\mathcal{S} = \{q, Q, r, R\}$.

For the observation noise statistics r and R, consider the nonlinear observation at time k, which is $y_k = h(x_k, u_k) + v_k$. Since the true value of x_k is unknown, v_k is approximated by

$$r_k = y_k - \hat{h}(x_k, u_k), \tag{12}$$

where r_k represents a sample of the observation noise v at time k and

$$\hat{h}(x_k, u_k) = \sum_{i=0}^{2L} W_i^{(m)} h(\mathcal{X}_{i,k}^x, u_k) \triangleq \hat{h}_k. \tag{13}$$

Define a new random variable $\xi \sim \mathcal{N}(r, C_r)$, and assume that there are N samples $r_k (k = 1, \dots, N)$, such that the r_ks are N empirical measurements for ξ. An unbiased estimator for r can be the sample mean

$$\hat{r} = \frac{1}{N} \sum_{k=1}^{N} r_k, \tag{14}$$

where the term *unbiased* implies that $\mathbb{E}[\hat{r}] = \mathbb{E}[\xi] = r$. An unbiased estimator for the covariance of ξ is

$$\hat{C}_r = \frac{1}{N-1} \sum_{k=1}^{N} (r_k - \hat{r})(r_k - \hat{r})^T, \tag{15}$$

where the term *unbiased* implies $\mathbb{E}[\hat{C}_r] = \mathbb{E}[(\xi - r)(\xi - r)^T]$. Because $y_k = h(x_k, u_k) + v_k$, (12) becomes

$$r_k = h(x_k, u_k) - \hat{h}_k + v_k. \tag{16}$$

The covariance of ξ is therefore calculated as

$$
\begin{aligned}
\mathbb{E}[(\xi - r)(\xi - r)^T] &= \frac{1}{N} \sum_{k=1}^{N} \mathbb{E}[(r_k - r)(r_k - r)^T] \\
&= \frac{1}{N} \sum_{k=1}^{N} \mathbb{E}\{[h(x_k, u_k) - \hat{h}_k + v_k - r] \\
&\quad \times [h(x_k, u_k) - \hat{h}_k + v_k - r]\}^T \\
&= \frac{1}{N} \sum_{k=1}^{N} (\mathbb{E}\{[h(x_k, u_k)][h(x_k, u_k)]^T\} \\
&\quad - (\hat{h}_k)(\hat{h}_k)^T) + R,
\end{aligned}
\tag{17}
$$

where

$$\mathbb{E}\{[h(x_k, u_k)][h(x_k, u_k)]^T\} = \sum_{i=0}^{2L} W_i^{(m)} [h(\mathcal{X}_{i,k}^x, u_k)][h(\mathcal{X}_{i,k}^x, u_k)]^T. \tag{18}$$

Note that x_k and v_k are assumed to be independent in (17). An unbiased estimate of R from (15) and (17) is

$$
\begin{aligned}
\hat{R} &= \frac{1}{N-1} \sum_{k=1}^{N} [(r_k - \hat{r})(r_k - \hat{r})^T \\
&\quad - \frac{N-1}{N} (\mathbb{E}\{[h(x_k, u_k)][h(x_k, u_k)]^T\} - (\hat{h}_k)(\hat{h}_k)^T].
\end{aligned}
\tag{19}
$$

For the process noise statistics q and Q, consider the nonlinear state propagation at time k, which is $x_k = f(x_{k-1}, u_{k-1}) + w_{k-1}$. Since the true values of x_k and x_{k-1} are unknown, w_{k-1} is approximated by

$$q_k = \hat{x}_k - \hat{f}(x_{k-1}, u_{k-1}), \tag{20}$$

where q_k represents a sample of the process noise w at time step $k - 1$, and

$$\hat{f}(x_{k-1}, u_{k-1}) = \sum_{i=0}^{2L} W_i^{(m)} f(\mathcal{X}_{i,k-1}^x, \ u_{k-1}) \triangleq \hat{f}_{k-1}. \qquad (21)$$

We define a new random variable $\zeta \sim \mathcal{N}(q, C_q)$ and assume that there are M samples q_k $(k = 1, \ldots, M)$, where the q_ks are M empirical measurements for ζ. An unbiased estimator for the mean value of ζ is the sample mean

$$\hat{q} = \frac{1}{M} \sum_{k=1}^{M} q_k. \qquad (22)$$

An unbiased estimator for the covariance of ζ is

$$\hat{C}_q = \frac{1}{M-1} \sum_{k=1}^{M} (q_k - \hat{q})(q_k - \hat{q})^\top, \qquad (23)$$

such that $\mathbb{E}[\hat{C}_q] = \mathbb{E}[(\zeta - q)(\zeta - q)^T]$. The covariance Q is then calculated as

$$\begin{aligned}
&\mathbb{E}[(w_{k-1} - q)(w_{k-1} - q)^T] \\
&= \mathbb{E}[(w_{k-1} - q_k + q_k - q)(w_{k-1} - q_k + q_k - q)^T] \\
&= \frac{1}{M} \sum_{k=1}^{M} \mathbb{E}(\{(x_k - \hat{x}_k) - [f(x_{k-1}, u_{k-1}) - \hat{f}_{k-1}] + (q_k - q)\}(q_k - q)^T) \\
&= \frac{1}{M} \sum_{k=1}^{M} (\mathbb{E}\{[f(x_{k-1}, u_{k-1})][f(x_{k-1}, u_{k-1})]^T\} - (\hat{f}_{k-1})(\hat{f}_{k-1})^T - P_k) \\
&\quad + \mathbb{E}[\hat{C}_q],
\end{aligned} \qquad (24)$$

where

$$\begin{aligned}
&\mathbb{E}\{[f(x_{k-1}, u_{k-1})][f(x_{k-1}, u_{k-1})]^T\} \\
&= \sum_{i=0}^{2L} W_i^{(m)}[f(\mathcal{X}_{i,k-1}^x, u_{k-1})][f(\mathcal{X}_{i,k-1}^x, u_{k-1})]^T. \qquad (25)
\end{aligned}$$

Then Q can be estimated without bias using (23) and (24),

$$\begin{aligned}
\hat{Q} = \frac{1}{M-1} \sum_{k=1}^{M} & \Big[(q_k - \hat{q})(q_k - \hat{q})^\top + \frac{M-1}{M} \\
& \times \Big(\mathbb{E}\{[f(x_{k-1}, u_{k-1})]\}[f(x_{k-1}, u_{k-1})]^\top - (\hat{f}_{k-1})(\hat{f}_{k-1})^\top - P_k \Big) \Big]. \qquad (26)
\end{aligned}$$

Equations (14), (19), (22), and (26) provide unbiased estimates for r, R, q, and Q, which are based on N observation noise samples and M process noise samples, respectively. All samples r_k and q_k are assumed to be statistically independent and identically distributed. The summarized algorithm of the ALM-UKF based on (12)–(26) is presented in [28], where α, β, κ, and λ are the UT parameters [27].

Parameter Estimation
For the system given in (11), the following dynamics are introduced for the parameter vector p,

$$p_{k+1} = p_k + w_k^{\mathrm{p}}, \qquad (27)$$

where $w_k^{\mathrm{p}} \sim \mathcal{N}(q^{\mathrm{p}}, Q^{\mathrm{p}})$ is Gaussian process noise. The augmented state is defined as $x^{\mathrm{a}} = [x^{\mathrm{T}}, p^{\mathrm{T}}]^{\mathrm{T}}$. It then follows from (11) and (27) that

$$x_{k+1}^{\mathrm{a}} = F(x_k^{\mathrm{a}}, u_k) + w_k^{\mathrm{a}}, \quad y_k = H(x_k^{\mathrm{a}}, u_k) + v_k, \qquad (28)$$

where $w_k^{\mathrm{a}} = [w_k^{\mathrm{T}}, (w_k^{\mathrm{p}})^{\mathrm{T}}]^{\mathrm{T}}$.

The matrices $\hat{R}$ and $\hat{Q}$ in (19) and (26) may become negative-definite during the implementation (which is also mentioned in [29]). In this work, the nearest positive-definite matrices of $\hat{R}$ or $\hat{Q}$ are calculated when negative eigenvalues of $\hat{R}$ or $\hat{Q}$ are observed, such that a symmetric positive-definite matrix nearest to $\hat{R}$ or $\hat{Q}$ in terms of the Frobenius norm can be obtained [32].

Artificial Gaussian process noise w_k^{p} in (27) is used to change the parameter p when the UKF is running. However, if the value of w_k^{p} is large, then the parameter p will be changed by a large amount at each time step. This condition may further cause the filter to diverge, since the parameterized vehicle model in the section "Single-Track Vehicle Model" is sensitive to p and may therefore become unstable for unreasonable values of p. This problem was addressed by rescaling the diagonal entries of Q^{p} to be some small positive values at each time step. Other discussions on the numerical instability problems of the UKF can be found in [29] and [32].

Online State Estimation
Accurate state estimation is required for the controllers to run reliably online. Many control methods assume accurate knowledge of the system state. In general, more accurate and high-rate information is preferred. Some states, such as the wheel speeds, can be directly measured. However, position, orientation, and velocity estimates are required for the control and planning of the AutoRally platform. This information, in general, cannot be measured with a single sensor, requiring some form of sensor fusion estimation.

GPS data are inherently low rate and lack orientation information. IMU measurements are relatively high rate but do not directly provide heading or linear velocity information. By combining the time-synchronized signals from these two sensors, a very accurate and high-rate estimate of position, velocity, and orientation can be obtained. This state information is sufficient for many advanced control systems.

Previously, many systems have had success with sensor fusion using methods such the EKF or UKF, as presented in the previous section. Filtering methods are limited by the fact that all previous state and measurement information is discarded. If, instead, the system is modeled as a set of hidden states (in this case, the position and velocity of the vehicle), with sensor measurements providing probabilistic
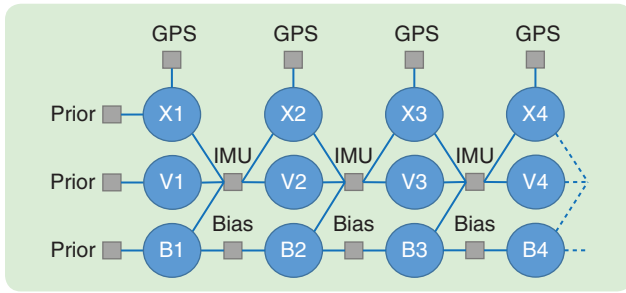
FIGURE 14 The factor graph structure used for Global Positioning System (GPS) and inertial measurement unit (IMU) sensor fusion using the Georgia Tech smoothing and mapping optimization library. The circles represent states, and the squares represent factors.

information about these states, more accurate estimates (via smoothing) can be achieved.

Factor graphs combined with advanced inference algorithms such as incremental smoothing and mapping 2 (iSAM2) [33] allow smoothing over many types of measurements, while retaining the ability to relinearize previous information. This reduces many of the problems found in the Kalman filter with states or measurements that are not approximately linear in the measurement time frame.

The factor graph representation of sensor fusion is a method of visualizing states and measurements as a bipartite graph (an example is shown in Figure 14). The factor graph has two types of nodes: *factor nodes* $f_i \in F$ and *variable nodes* $\theta_j \in \Theta$. Edges e_{ij} always connect factor and variable nodes. Variable nodes correspond to the unmeasured quantities to be estimated. Factor nodes correspond to probabilistic information gained from a measurement z_i about a set of variables (connected to the factor by edges). The factor graph as a whole represents the probability distribution generated by the probabilistic information encoded in the factors

$$p(\theta_1, \theta_2, \ldots, \theta_n \mid z_1, z_2, \ldots, z_k), \qquad (29)$$

where θ_i is a variable that is not directly observed and z_j is a measured variable. This function can then be factorized as

$$f(\Theta) = \prod_i f_i(\Theta_i), \qquad (30)$$

where Θ is the set of all variables in the graph and Θ_i is the set of variables connected to factor f_i by an edge. $f_i(\Theta_i)$ takes, for example, the following form for the first IMU factor in Figure 14, which is

$$p(X_1, V_1, B_1, X_2, V_2 \mid \omega_x, \omega_y, \omega_z, a_x, a_y, a_z), \qquad (31)$$

where X_i, V_i, and B_i are the position, velocity, and bias state variables, respectively, and w_n and a_n are the measured angular velocity and linear acceleration, respectively, from an IMU.

Independence relationships in the measurements in $f(\Theta)$ are encoded in the edges e_{ij}, where each factor f_i is a function of variables Θ_j. The goal in sensor fusion is to find the variable assignment Θ^* that minimized the function $f(\Theta)$ in (30),

$$\Theta^* = \underset{\Theta}{\arg\max} f(\Theta). \qquad (32)$$

Each $f(\Theta)$ can be written in terms of a difference between the measured value z_i and the predicted value from the measurement function $h_i(\Theta_i)$. By framing this in terms of log-likelihood, this maximization problem becomes

$$\underset{\Theta}{\arg\max} f(\Theta) = \underset{\Theta}{\arg\min}[-\log f(\Theta)]$$
$$= \underset{\Theta}{\arg\min} \frac{1}{2} \sum_i \left\| h_i(\Theta_i) - z_i \right\|_{\Sigma_i}^2, \qquad (33)$$

where $h_i(\Theta_i)$ are the measurement functions relating a set of variables Θ_i to a sensor measurement z_i. This minimization problem can be solved with several different nonlinear minimization strategies, including Gauss–Newton or Levenberg–Marquardt, which iteratively linearize and solve this problem. Using these methods, an entire graph of measurements can be created (as in Figure 14) to solve for Θ^*, which will be the maximum likelihood estimate of the variables Θ being estimated.

However, when estimating the state of the AutoRally platform, this problem is solved at each time step to produce a maximum likelihood estimate of the current state variables position X_i, velocity V_i, and accelerometer and gyroscope bias B_i. Reoptimizing the entire factor graph would be very inefficient, so the iSAM2 algorithm is used.

The iSAM2 algorithm is a part of the Georgia Tech smoothing and mapping (GTSAM) [34], [35] software package, which uses a factor graph representation to iteratively solve the smoothing and mapping problem. See "Georgia Tech Smoothing and Mapping" for a brief discussion of GTSAM. iSAM2 efficiently performs iterative updates to a factor graph and optimizes this new graph without relinearizing the full problem. To perform this optimization, a factor graph (shown in Figure 14) is constructed with successive measurements and iteratively optimized. At each smoothing time step, an additional set of states X, V, and B are added to the graph. Measurement factors for the GPS and IMU sensors also are added, along with a bias smoothness factor. To keep the computational load low while maintaining high accuracy, the factor graph contains state nodes for measurements taken at 10 Hz. The factors in the graph correspond to GPS measurements and preintegrated IMU measurements [36]. Online, the IMU measurements are integrated to interpolate the 10-Hz smoothed position to publish the state estimate at 200 Hz. Example trajectories are shown in Figure 15.

In practice, the measurements from the GPS sensor can drift slightly each day, primarily because of the RTK correction antenna position not being fixed for each test. To counteract these changes, the robot is always positioned at the same place on the track when the state estimator is started. This track position is used as the origin of a local Euclidean coordinate system, oriented tangent to the GPS reference ellipsoid. This prevents GPS drift from effecting the vehicle state estimate relative to the fixed-track boundaries.

Georgia Tech Smoothing and Mapping

The Georgia Tech Smoothing and Mapping (GTSAM) tool box [34], [35] is a C++ factor graph library released under the BSD license and developed by the Borg Lab at the Georgia Institute of Technology. It provides state-of-the-art solutions to the simultaneous localization and mapping (SLAM) and structure from motion problems and can be used to model and solve many other simple and complex estimation problems. Matlab and Python interfaces enable rapid prototyping, visualization, and user interaction.

GTSAM allows users to model a problem, such as state estimation, using a factor graph. A factor graph is a graphical model that contains variables related through factors. An example factor graph used for state estimation is shown in Figure 14. The variables are vehicle states at specific points in time, and the factors encode probabilistic information relating to sensor measurements of one or more of these variables. The absolute or relative values of several variables are the result of measurements by a sensor such as an inertial measurement unit or Global Positioning System (GPS) receiver. This factor graph representation of the state estimation problem allows GTSAM to solve the problem by finding a maximum a posteriori estimate of all of the variables in the graph.

In addition to simplifying SLAM and state estimation, it is easy to extend GTSAM to solve new problems that are naturally formulated as the set of functions mapping hidden variables to measurements. Every factor in the graph can be expressed as a measurement function relating quantities to be estimated with sensor measurements. New types of factors are implemented by defining a measurement function and its derivative for each directly related variable

$$z_k = h_k(x_{k0}, x_{k1} \ldots x_{kn}), \tag{S1}$$

where z_k is measurement k and x_{kn} is the state x_n related to measurement z_k. As a concrete example, the GPS factors in Figure 14 are direct measurements of position, with additive Gaussian noise

$$Z_i = X_i + \omega_i, \tag{S2}$$

where $Z_i = [x_{mi}, y_{mi}, z_{mi}]$ is the measured position in Euclidean space, $X_i = [x_i, y_i, z_i]$ is the state variable being estimated, and $\omega_i = [\omega_{xi}, \omega_{yi}, \omega_{zi}]$ is Gaussian noise on the measurement.

The noise in the variable w has a straightforward interpretation as the measurement uncertainty. For Gaussian noise, one must assume that the measurement is unbiased, that is, the mean is zero. The variance of the noise then becomes the uncertainty on the measurement. For example, if the GPS receiver is specified to have a position standard deviation of 1 m in the x- and y-directions and 2 m in the z-direction, then these become the parameters of the Gaussian noise $\omega_{xi}, \omega_{yi}, \omega_{zi}$ in the factor graph.

GTSAM exploits sparsity for computational efficiency. Typically, measurements only provide information on the relationship between a handful of variables, so the resulting factor graph will be sparsely connected. This is exploited by the algorithms implemented in GTSAM to reduce computational complexity. Even when graphs are too dense to be handled efficiently by direct methods, GTSAM provides iterative methods that are quite efficient.
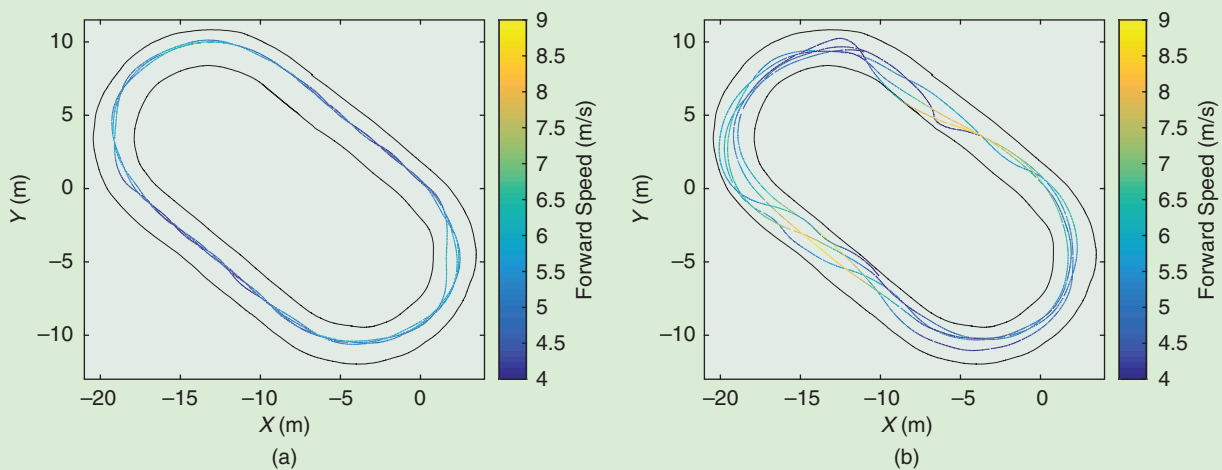

FIGURE 15 Example states generated by the state estimator built with the Georgia Tech smoothing and mapping optimization library. Inputs to the state estimator are Global Positioning System and inertial measurement unit sensor data. Each experiment is composed of four laps of data collected at the Georgia Tech Autonomous Racing facility oval track. Track boundaries are colored black, and state estimates are color coded according to the speed at which the AutoRally robot was traveling. (a) Model predictive path integral controller driving with a target speed of 6 m/s. (b) Manual driving for the system identification data set.

Semantic Segmentation
Using Convolutional Neural Networks

An important step in dynamic visual scene understanding for autonomous driving is to analyze the images captured from the onboard cameras and segment important objects [such as cars, pedestrians, or signs (using bounding boxes)] and label the key surfaces in the environment (such as streets, sidewalks, and building facades). The task of grouping pixels and labeling the surfaces they belong to is known as semantic segmentation. In the specific context of the racing domain, the primary goal is to label pixels as track or nontrack so that the vehicle knows where it can drive. In this section, a method to create and train a deep convolutional neural network (CNN) [37], [38] architecture is described that receives an image from the onboard cameras and outputs a pixel-wise labeling of track and nontrack.

Recent works have demonstrated the success of deep neural network architectures consisting of multiple CNN layers in solving challenging semantic segmentation problems [39]–[42]. The structure of the CNN takes advantage of the spatial properties of an image, namely, that nearby pixels are likely to have similar labels. For this work, we wish to perform semantic segmentation by learning a function that can map an input image $I(u, v)$ to an output image with binary pixel labels corresponding to track and nontrack and given by

$$Y(u, v) = f[I(u', v'), \Theta], \quad u' = u - r \cdots u + r, \quad v' = v - r \cdots v + r, \tag{34}$$
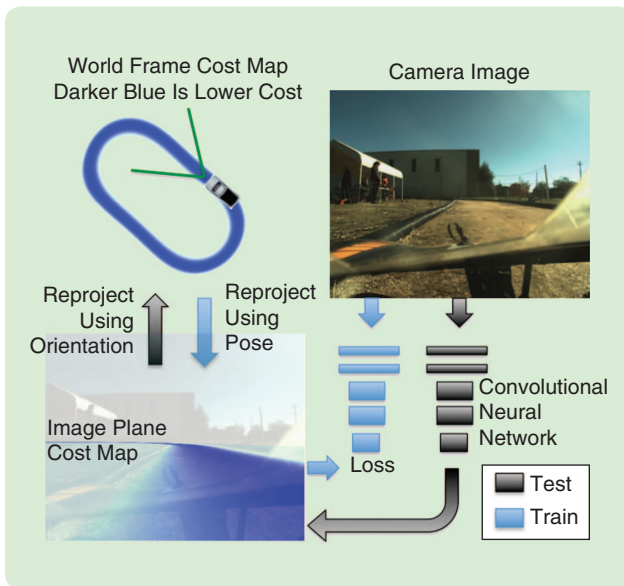
FIGURE 16 The convolutional neural network pipeline for training and testing pixel-wise image labeling of track and nontrack using images from the onboard AutoRally. For training, a survey of the track boundaries, calibrated inertial measurement unit sensor to camera transform, and state estimate is used to automatically label image pixels. For testing, the network labels each pixel of an input image as track or nontrack.

where u and v are pixel coordinates, r is the receptive field for an output pixel (which is dependent on the structure of f), and Θ is the set of parameters in $f(\cdot)$ that can be changed to create the desired function mapping.

The constructed deep neural network is composed of ten convolutional layers and two 2×2 max pooling layers (after the third and sixth layers) to reduce the 640×480-sized input image to 160×128. All layers are 3×3 convolution kernels, except the last three layers, which have 5×5 kernels. Layers before the first maxpool have 32 kernels, layers between the first and second have 64 kernels, and the final three have 96, 128, and 256 kernels, respectively.

The standard mini-batch gradient descent method was used to train a CNN function approximator to output the correct pixel labels given an input image. To use gradient descent, a loss function l must first be defined to optimize. In this work, a cross-entropy function that rewards the network for producing an output closer to the desired value of nontrack (1) or track (0). The gradient of each parameter is computed with respect to the loss function $\partial \Theta / \partial l$ to provide a direction and magnitude to move each parameter that pushes the generated output closer to the desired output. However, because computing this gradient over all images is computationally prohibitive, the derivative is computed for a random subset of ten images, the parameters are moved a small amount Θ in the direction of the gradient, and the process is repeated until convergence. A large set of training data must be created to train a neural network from scratch. Over 100,000 ground-truth-labeled images were automatically generated using data collected from the AutoRally platform. The images were labeled using a precomputed GPS survey of the test track, the output from the state estimator, and the IMU to perform camera calibration. The training and testing pipeline is shown in Figure 16. The state estimator frame is at the IMU, so a homography matrix must be computed that transforms the surveyed track map from world coordinates to image plane coordinates using the calibrated transformation between the IMU and the camera

$$H = k T_{\text{im}}^{\text{car}} T_{\text{car}}^{\text{world}}, \tag{35}$$

where $T_{\text{car}}^{\text{world}}$ is the position of the car in world coordinates, $T_{\text{im}}^{\text{car}}$ is the transformation between the IMU and camera reference frames, and k is the matrix of camera intrinsics. Given this mapping, points in the ground coordinate frame can be projected into the image

$$p_{\text{im}} = \hat{H} p_{\text{world}}, \quad \hat{H} = \begin{bmatrix} H_{11} & H_{12} & H_{14} \\ H_{21} & H_{22} & H_{24} \\ H_{31} & H_{32} & H_{34} \end{bmatrix}, \tag{36}$$

where p_{im} and p_{world} are homogeneous points. Using this method, all points on the image plane for a given image can be projected to the ground plane and given a ground truth label.

EXPERIMENTAL RESULTS

This section describes the testing facility and shows/validates the results of the parameter estimations using the bifilar pendulum method, standard UKF, ALM-UKF, and neural network cost map estimation. Data for all of the estimation results were collected with a human manually driving the AutoRally robot around GT-ARF. The same data were also used to train the

dynamics model used in the model predictive path integral (MPPI) controller described in "Model Predictive Path Integral Control." The data consist of approximately 30 min of human-controlled driving at speeds varying from 4 to 10 m/s. The driving was broken into five distinct behaviors:

1) normal driving at low speeds (4–6 m/s)
2) zig-zag maneuvers performed at low speeds (4–6 m/s)

Model Predictive Path Integral Control

All experimental data collected for training and testing track surface labeling were gathered at the Georgia Tech Autonomous Racing Facility (GT-ARF) using the model predictive path integral (MPPI) controller. MPPI is a stochastic model predictive control (MPC) method that can drive AutoRally up to, and beyond, the friction limits of the track. It has been shown to work well in practice when applied to AutoRally and some simulated systems [S2]–[S4]. A version of MPPI along with a dynamics model for AutoRally learned from human driving is available in the AutoRally GitHub repository.

MPC works by interleaving optimization and execution. First, an open-loop control sequence over a finite time horizon is optimized. Then, the first control in that sequence is executed by the vehicle. Next, state feedback is received, and the whole optimization process repeats. MPPI is a sampling-based, derivative-free approach to MPC that has been successfully applied to aggressive autonomous driving using learned nonlinear dynamics [S3].

At each iteration, MPPI begins with the estimate of the optimal control sequence from the previous time step and uses importance sampling to generate thousands of new sequences of control inputs. These control sequences are then propagated forward in the state space using the system dynamics, and each trajectory is evaluated according to a cost function. The estimate of the optimal control sequence is then updated with a cost-weighted average over the sampled trajectories. State feedback is then introduced to begin the next iteration. Real-time execution of MPPI on AutoRally is enabled by the onboard Nvidia graphics processing unit (GPU).

Mathematically, let the current planned control sequence be $(u_0, u_1, \ldots u_{T-1}) = U \in \mathbb{R}^{m \times T}$, and let $(\varepsilon_1, \varepsilon_2 \ldots \varepsilon_K)$ be a set of random control sequences, with each $\varepsilon_k = (\epsilon_k^0, \ldots \epsilon_k^{T-1})$ and each $\epsilon_k^t \sim \mathcal{N}(u_t, \Sigma)$. The control sequence update is

$$\eta = \sum_{k=1}^{K} \exp \left\{ -\frac{1}{\lambda} \left[S(\varepsilon_k) + \gamma \sum_{t=0}^{T-1} u_t^T \Sigma^{-1} \epsilon_k^t \right] \right\}, \qquad (S3)$$

$$U = \frac{1}{\eta} \sum_{k=1}^{K} \left(\exp \left\{ -\frac{1}{\lambda} \left[S(\varepsilon_k) + \gamma \sum_{t=0}^{T-1} u_t^T \Sigma^{-1} \epsilon_k^t \right] \right\} \varepsilon_k \right). \qquad (S4)$$

The parameters λ and γ determine the selectiveness of the weighted average, also known as the temperature and the importance of the control cost, respectively. The function $S(\varepsilon)$ accepts an input sequence and propagates it through the dynamics to find the resulting trajectory and then computes the state-dependent cost of that trajectory sequence, which is

denoted as $C(x_0, x_1, \ldots x_T) = \Sigma_{t=0}^{T} q(x_t)$. In the implementation of this algorithm on AutoRally, only an instantaneous running cost is used. There is no terminal cost, and we sample trajectories on a GPU using several different dynamics models. The instantaneous running cost is

$$q(x) = w \cdot \left[C_M(p_x, p_y), (v_x - v_x^d)^2, 0.9^t I, \left(\frac{v_y}{v_x} \right)^2 \right], \qquad (S5)$$

where the first term $C_M(p_x, p_y)$ is the positional cost of being at the body frame position (p_x, p_y). This positional cost is derived from a presurveyed Global Positioning System registered cost map in most of our data collection and from the output of the neural network in the section "Track Surface Labeling" and [S5]. The second term is the cost for achieving a desired speed v_x^d, and the third term in the cost is an indicator variable that is turned on if the track cost, roll angle, or heading velocity are too high. The final term in the cost is a penalty on the slip angle of the vehicle. The coefficient vector used in the experiments was $w = (100, 4.25, 10,000, 1.75)$; note that all but the first term are trivial to compute given the vehicle's state estimate, while the first term requires an analysis of the vehicle's environment. In practice, parameter weights of the cost function require some tuning. However, the tuning is relatively intuitive, as the weights correspond to easily understandable real-world values.

This algorithm was used to drive the AutoRally platform several hundred kilometers around the GT-ARF. Because it does not require explicit derivatives, it supports advanced uses such as the noisy cost maps detailed in the section "Track Surface Labeling." It has shown good performance rejecting large disturbances from potholed and muddy track conditions, while reliably running for a full battery charge at high speeds.

REFERENCES
[S2] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2016, pp. 1433–1440.
[S3] G. Williams, N. Wagener, B. Goldfain, P. Drews, B. Boots, J. M. Rehg, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, Singapore, 2017, pp. 1714–1721.
[S4] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory parallel computation," *J. Guidance Control Dyn.*, vol. 40, no. 2, pp. 1–14, 2017.
[S5] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Combining convolutional neural networks and model predictive control," in *Proc. 1st Annu. Conf. Robot Learning*, 2017, pp. 133–142.

3) linear acceleration maneuvers that consist of accelerating the vehicle as much as possible in a straight line, then braking before starting to turn
4) sliding maneuvers, where the pilot attempts to slide the vehicle as much as possible
5) high-speed driving up to 10 m/s, where the pilot tries to drive the vehicle around the track as fast as possible.

Each one of these maneuvers was performed for 3 min while moving clockwise around the track and another 3 min moving counterclockwise.

Georgia Tech Autonomous Racing Facility

All experiments were conducted at the GT-ARF (shown in Figure 17), which is a 68-m-long dirt track with required site infrastructure to support autonomous vehicle testing. The track is a 3.3-m-wide flat clay surface with two straights connected by 180° constant radius turns. The outer dimensions of the track are 27.5 and 15.5 m, and the straights are 11.5-m long. The boundaries of the track are 0.15-m-diameter corrugated drainage pipes secured in place with stakes to provide a semirigid crash barrier. The controlled track environment allows the robot to operate fully autonomously up to the limits of its mechanical, electrical, and software systems and beyond the friction limits of the track.

(a)

(b)

FIGURE 17 The Georgia Tech Autonomous Racing Facility. (a) The aggressive autonomous driving experiments. (b) An autonomous driving demo for third-grade students and parents.

A ground station is established in the center of the track that consists of an OCS laptop, wireless runstop box, and base station GPS module that provides RTK corrections. The OCS laptop and associated hardware are used to remotely communicate and monitor the status of the robot.

Bifilar Pendulum

Many vehicle parameters can be directly measured with a scale or ruler or are available through manufacturer documentation. Those parameters of the AutoRally platform are listed in Table 3. Data were collected using the setup shown in Figure 9 to estimate the MOI of the primary axis of the robot and the axis of rotation for the front and rear wheels. Table 4 details the measured parameters for each configuration and the resulting MOI. In each case, T is calculated as an average of 60 oscillation periods.

TABLE 3 AutoRally robot parameters. Values were measured with a digital scale or ruler or provided by manufacturer documentation. Rear axle offset distances are with respect to the center of gravity (CG).

Parameter	Value	Units
Total mass	21.88	kg
Front wheel mass, each	0.82	kg
Rear wheel mass, each	0.89	kg
Overall length	0.90	m
Overall width	0.46	m
Overall height	0.32	m
Wheelbase	0.57	m
Rear axle to CG, x offset	0.23	m
Rear axle to CG, z offset	0.12	m
Front track	0.395	m
Rear track	0.405	m
Wheel diameter	0.195	m

TABLE 4 The moments of inertia (MOI) about primary axes and related parameters. Values are experimentally calculated using the bilifilar pendulum method and (3) and the measured masses and lengths. The oscillation period T is calculated by averaging 60 oscillation periods using the bifilar pendulum method. $g = 9.81$ was used for all calculations. The front- and rear-wheel MOI include both wheels.

T (s)	b (m)	d (m)	Parameter	MOI (kg-m^2)
2.63	0.14	1.90	I_x	0.347
2.12	0.295	1.88	I_y	1.131
2.48	0.26	1.93	I_z	1.124
1.81	0.182	0.935	I_f	0.048
1.67	0.182	0.915	I_r	0.044
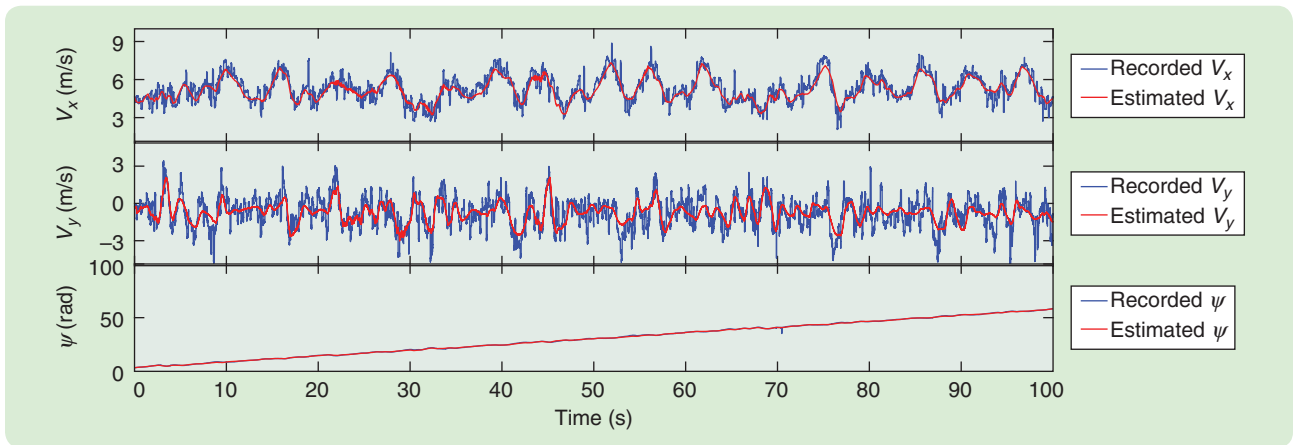
FIGURE 18 The state estimation for the single-track model using joint-state unscented Kalman filter.

Standard Unscented Kalman Filter

The standard JS-UKF was first implemented using the three different vehicle models in the section "Estimation." The hyperparameters of the filter are critical for the filter design, especially the process noise covariance Q [43]. In this section, the diagonal elements of these matrices are recursively tuned until the parameterized vehicle model shows satisfactory simulation results.

A total of 113 s of experimental data generated by a human driver using the AutoRally vehicle were selected. The first 100 s of data were used to tune the hyperparameters and estimate the vehicle parameters, and the remaining 13 s (a complete cycle around the testing track) were used to validate the results. Figure 18 shows the estimates for several selected states of the system for the single-track model. It can be seen that the estimates of the states agree well with the data. The results for the other vehicle models were similar and hence omitted.

Next, the estimated parameters in the simulation were validated to ensure that the obtained parameters were able to satisfactorily reproduce the data, hence accurately predicting the vehicle's motion in practical applications. Figure 19 shows the simulated trajectories for different vehicle models configured with the estimated parameters. The results in Figure 19 indicate that the larger the DOF of the model, the more accurate the results and the better the agreement with data.

Adaptive Limited Memory Unscented Kalman Filter

Instead of tuning the noise, the ALM-UKF was implemented to find the suboptimal estimation of the noise statistics online, during which the augmented state and noise are estimated simultaneously. Experimental data collected with the AutoRally robot were used to validate the ALM-UKF, which is presented in algorithmic form in [28]. The noise samples at each time step k are from the estimation based on the last 10 s of data. The estimation of the velocities, yaw angle, and positions states is not difficult. Thus, only the estimation results of the unknown vehicle parameters are shown. The ALM-JUKF [28] was implemented to estimate the parameters of a full-vehicle model from (5a)–(7c) using the AutoRally experimental data, which

includes the measurements for $V_x, V_y, V_z, r, \dot{\theta}, \dot{\phi}, \phi, \theta, \phi,$ and the GPS coordinates.

For stable state estimation, the joint-state system with the parameter dynamics equations in (27) must be observable. To this end, the Jacobian of the joint-state system is calculated, and the rank of the observability matrix is inspected. The result indicates that the spring stiffness and damping coefficients of the suspension system K_i and C_i ($i = \mathrm{f}, \mathrm{r}$) are unobservable (see Figure 11), which means that one cannot uniquely identify K_i and C_i ($i = \mathrm{f}, \mathrm{r}$) based on current measurements. This issue is addressed to obtain the observability by fixing the values of any two of the four parameters, specifically $K_\mathrm{f} = K_\mathrm{r} = 2000\,\mathrm{N/m}$. The values for K_f and K_r were chosen by assuming a 1–2-cm average deformation of the springs caused by the gravity of the sprung mass.

As discussed, the artificial Gaussian process noise w_k^p in (27) is used to change the parameter p when the UKF is
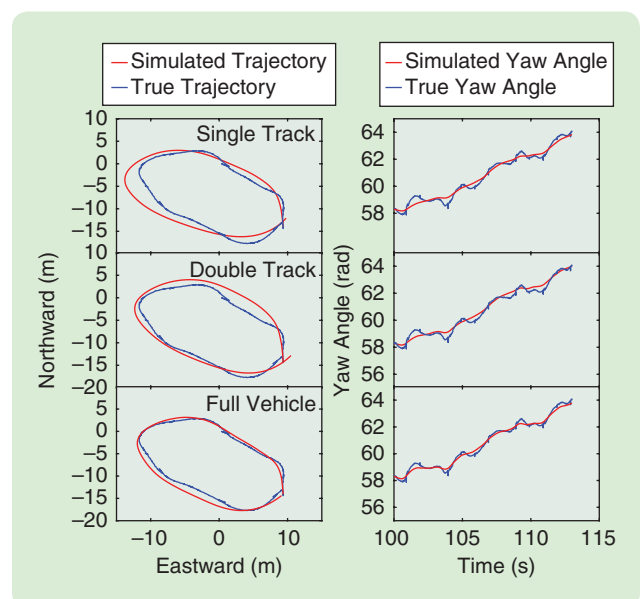
FIGURE 19 The simulation results of the estimated vehicle models using a standard unscented Kalman filter.
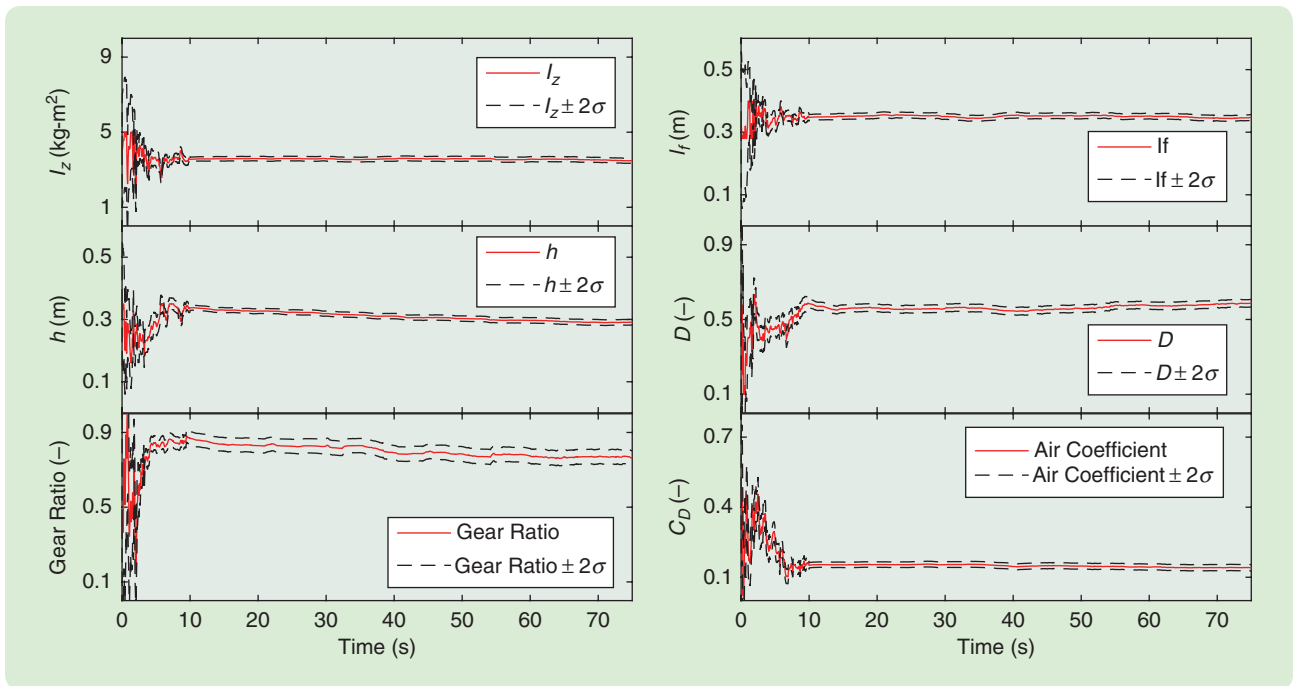
FIGURE 20 The convergence of the vehicle parameters along with the estimation process. All the parameters converge fast and stabilize after approximately 10–20 s.

TABLE 5 A parameter comparison for the adaptive limited-memory unscented Kalman filter (ALM-UKF) and bifilar pendulum methods. The parameters that do not closely match across methods result because the ALM-UKF method uses real-world driving data, where these dimensions are constantly changing due to suspension and steering articulation, and the bilfilar method estimates all parameters when the vehicle is suspended and stationary.

Parameter	ALM-UKF	Bifilar	Difference	Units
m	20.6093	21.88	5.8%	kg
I_z	1.024	1.124	8.9%	kg-m^2
I_r	0.0499	0.044	13.4%	kg-m^2
h	0.0961	0.12	19.9%	m
ℓ_f	0.4650	0.34	36.8%	m

working. However, if the value of w_k^p is large, then the parameter p will be changed by a large amount at each time step. This condition may further cause the filter to diverge, since the parameterized vehicle models are sensitive to p and may become unstable for unreasonable values of p. This problem was addressed by rescaling the diagonal entries of Q^p to be small positive values at each time step. Other discussions on the numerical instability problems of the UKF can be found in [29] and [32].

Figure 20 shows the time trajectories of several parameters listed in Table 5 during the estimation process, where all the parameters converge fast and stabilize after approximately 10–20 s. Figure 21 compares simulated outputs from
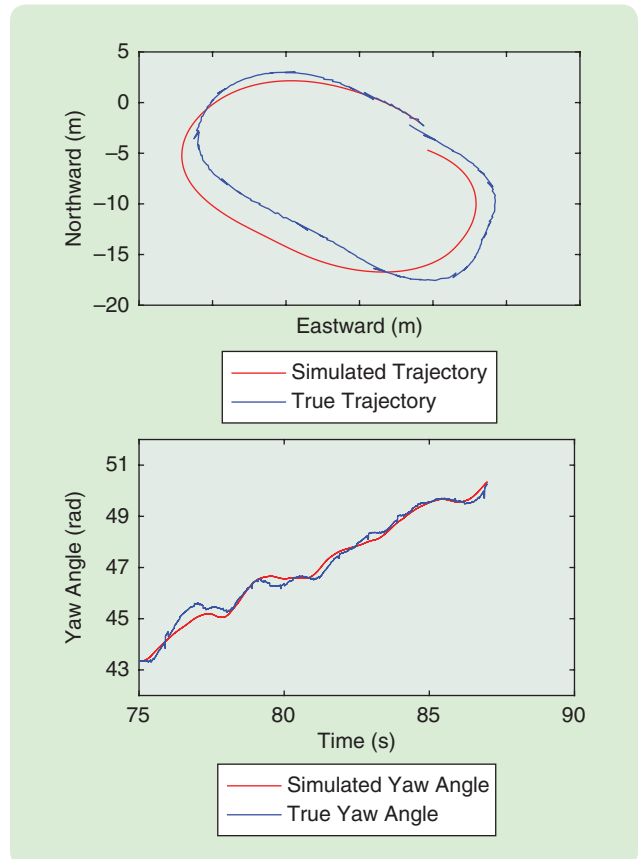
FIGURE 21 The simulation results of the estimated vehicle models using the adaptive limited-memory, joint-state unscented Kalman filter.

the full-vehicle model using the estimated parameters for the AutoRally platform and real-world data. It can be seen that, as expected, the identified vehicle model satisfactorily reproduces the data. Data collected using the AutoRally vehicle show obvious non-Gaussian noise that may have some effect on the estimation process.

Compared with the results in Figure 19, the simulated trajectories of the AutoRally vehicle in Figure 21 show larger deviation from the data. The reason may be that the estimation of the noise statistics of the standard UKF are tuned to be optimal (to some degree), but the ALM-UKF algorithm used a suboptimal estimator for the noise statistics. The advantages of the ALM-UKF are that it is more efficient and can operate online. The ALM-UKF algorithm is expected to be especially useful for time-varying parameters estimation problems, such as estimation of a linear- parameter-varying driver model [44].

Table 6 shows the MF model parameters for the single-track-vehicle model estimated using the UKF of the AutoRally platform. All of the parameters stabilized after approximately 20 s of data.

Some parameters were estimated by both the bilfilar pendulum and ALM-UKF methods, which are compared in Table 5. While the mass and MOI values estimated by both methods agree, the estimated dimensions are not as closely aligned with hand-measured values. This is because, in the bilfilar method, all parameters are measured when the vehicle is stationary, whereas the ALM-UKF method uses real-world driving data where these dimensions are constantly changing, due to suspension and steering articulation.

Track Surface Labeling

A data set of over 100,000 images, along with the corresponding position and orientation in a local coordinate frame, was recorded over the course of several days of testing with the MPPI algorithm at GT-ARF. The images were then postprocessed using the automatic labeling pipeline to produce ground-truth training images for the labeling task. The

TABLE 6 Tire model parameters estimated by the unscented Kalman filter for the single-track model of the AutoRally platform. Human driving data were collected at the Georgia Tech Autonomous Racing Facility.

| Parameter	Value	Units
B	1.1559	none
C	1.1924	none
D	0.9956	none
E	−0.8505	none
s_h	−0.0540	m
s_v	0.1444	m

neural network was trained using the Tensorflow [45] framework for 100,000 mini-batches, with each containing ten images. The results for an example training image can be seen in Figure 22, and a failure case of the network can be seen in Figure 23. The failure was a result of the dry grass adjacent to the track having a similar color and texture to the track surface and a mislabeling of the interior of the plastic protective body. Overall, the neural network was able to correctly label 97% of the pixels on the training set and 91% of the pixels on a held-out test set. Note that this approach may not generalize to variations in track conditions such as illumination changes and dynamic obstacles.

CONCLUSION

Despite decades of prior research and a renewed interest from technology companies and the research community, many gaps still remain in the capabilities of autonomous vehicles. This article introduces *AutoRally*, a high-performance robotics testbed that is 1:5 the size of a passenger car, which enables researchers to conduct experiments and collect real-world data under driving conditions that were previously untestable due to safety and cost concerns. The robust design and small
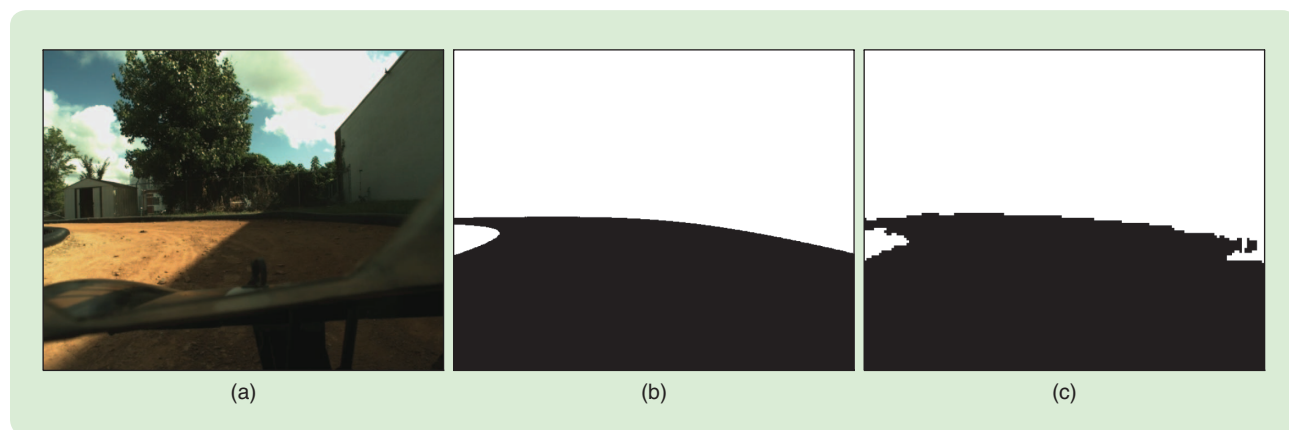
FIGURE 22 The neural network track surface labeling from camera images. (a) A training image captured by one of the cameras on AutoRally at the Georgia Tech Autonomous Racing Facility. (b) A ground-truth image generated by the labeling pipeline. (c) The pixel labeling of track and nontrack produced by the neural network.
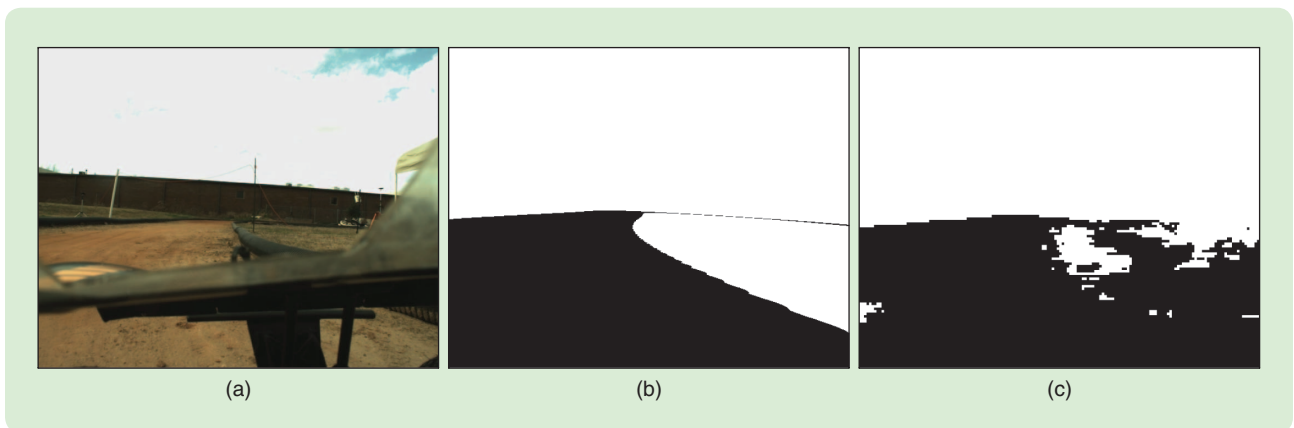
FIGURE 23 The neural network image labeling failure case. (a) The image input from the camera onboard AutoRally. (b) The ground-truth image generated by the labeling pipeline. (c) The pixel labeling of track and nontrack produced by the neural network. The failure is likely caused by the appearance similarity between the track surface and dry grass outside of the track.

size of AutoRally ameliorate the risk of high-speed crashes, allowing researchers to evaluate algorithms under conditions that would be too dangerous or expensive with a full-sized vehicle and too complex to model accurately in simulation.

A variety of online and offline estimation algorithms have been developed and applied to the platform to determine properties frequently required by control systems, including MOI and other difficult-to-measure properties. In addition, an approach based on CNNs was developed to address the task of online image segmentation trained from automatically labeled ground-truth images. To date, the fleet of six AutoRally platforms has logged over 300 km of fully autonomous driving using only onboard sensing and computing at the GT-ARF, resulting in the validation of multiple control and perception algorithms.

AutoRally is open source, so all of the documentation needed to build, configure, and run the platform is publicly available on the AutoRally GitHub repositories, including build instructions, a parts list, files required for custom fabrication, and operating procedures [15]. Tutorials and example controllers written in C++ and Python with ROS and Gazebo [16] are also available, along with a data set of human and autonomous driving. See "Build Your Own AutoRally Platform" for more information and tips on building an AutoRally platform. AutoRally opens a new frontier for the safe development and testing of autonomous vehicle technologies across a much more diverse set of operating regimes and a broader audience of investigators than previous experimental platforms have supported.

ACKNOWLEDGMENTS

AUTHOR INFORMATION

Brian Goldfain (bgoldfain3@gatech.edu) received the B.S. degree in electrical and computer engineering with a minor in robotics from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 2010 and the M.S. degree in computer science from the Georgia Institute of Technology, Atlanta, in 2013. He is currently a robotics Ph.D. student in the School of Interactive Computing at the Georgia Institute of Technology. His research interests focus on autonomous vehicle testbed development and autonomous racing. He can be contacted at the Georgia Institute of Technology, 85 5th St. NW, Office 218a, Atlanta, GA 30308 USA.

Paul Drews received the B.S. degree in electrical engineering from the Missouri University of Science and Technology, Rolla, in 2008. He is currently a robotics Ph.D. student in the School of Electrical and Computer Engineering at the Georgia Institute of Technology, Atlanta.

Changxi You received the B.S. and M.S. degrees from the Department of Automotive Engineering, Tsinghua University, Beijing, China, and the M.S. degree from the Department of Automotive Engineering, RWTH-Aachen University, Germany. He is currently a Ph.D. student under the supervision of Prof. Panagiotis Tsiotras in the School of Aerospace Engineering, Georgia Institute of Technology, Atlanta. His current research interests focus on system identification, aggressive driving, and control of (semi)autonomous vehicles.

Matthew Barulic received the B.S. degree in computer science from the Georgia Institute of Technology, Atlanta, in 2016. He is currently a software engineer at Wheego Technologies, Inc.

Orlin Velev received the B.S. degree in mechanical engineering from the Georgia Institute of Technology, Atlanta, in 2017. Currently, he is a vehicle structures engineer at Space Exploration Technologies (SpaceX).

Panagiotis Tsiotras is a professor in the School of Aerospace Engineering and the Institute for Robotics and

Intelligent Machines, Georgia Institute of Technology, Atlanta, where he is the director of the Dynamics and Control Systems Laboratory.

James M. Rehg is a professor in the School of Interactive Computing at the Georgia Institute of Technology, Atlanta, where he is director of the Center for Behavioral Imaging, codirector of the Center for Computational Health, and codirector of the Computational Perception Lab.

REFERENCES

[1] D. Schaper, "Human errors drive growing death toll in auto crashes," *NPR*. Accessed on: Mar. 2017. [Online]. Available: http://www.npr.org/2016/10/20/498406570/tech-human-errors-drive-growing-death-toll-in-auto-crashes

[2] K. Henry, "Traffic fatalities up sharply in 2015," U.S. Department of Transportation. Accessed on: Feb. 2017. [Online]. Available: https://www.nhtsa.gov/press-releases/traffic-fatalities-sharply-2015

[3] M. Bertoncello and D. Wee. (2015, June). Ten ways autonomous driving could redefine the automotive world. McKinsey & Company. [Online]. http://www.mckinsey.com/insights/automotive_and_assembly/ten_ways_autonomous_ driving_could_redefine_the_automotive_world

[4] National Conference of State Legislatures, "Autonomous vehicles: Self-driving vehicles enacted legislation." Accessed on: Mar. 2017. [Online]. Available: http://www.ncsl.org/research/transportation/autonomous-vehicles-self-driving-vehicles-enacted-legislation.aspx#enacted

[5] Donkey Car. Accessed on: Nov. 2017. [Online]. Available: http://www.donkeycar.com/

[6] J. Gonzales, F. Zhang, K. Li, and F. Borrelli, "Autonomous drifting with on-board sensors," in *Proc. 13th Int. Symp. Advanced Vehicle Control (AVEC'16)*, Munich, Germany, 2016, p. 133.

[7] F1/10 Autonomous Racing Competition. Accessed on: Nov. 2017. [Online]. Available: http://f1tenth.org/competition

[8] Rapid Autonomous Complex-Environment Competing Ackermann-steering Robot (RACECAR). Accessed on: Nov. 2017. [Online]. Available: https://mit-racecar.github.io/

[9] N. Keivan and G. Sibley, "Realtime simulation-in-the-loop control for agile ground vehicles," in *Proc. Conf. Towards Autonomous Robotic Systems*, New York, 2013, pp. 276–287.

[10] J. L. Jakobsen, "Autonomous drifting of a 1:5 scale model car," M.S. thesis, Dept. Eng. Cybernetics, Institutt for Teknisk Kybernetikk, Trondheim, Norway, 2011.

[11] S. Song, "Towards autonomous driving at the limit of friction," M.S. thesis, Dept. Mechanical and Mechatronics Eng., University of Waterloo, Ontario, Canada, 2015.

[12] M. Cutler, T. J. Walsh, and J. P. How, "Reinforcement learning with multi-fidelity simulators," in *Proc. 2014 IEEE Int. Conf. Robotics and Automation*, 2014, pp. 3888–3895.

[13] D. I. Katzourakis, I. Papaefstathiou, and M. G. Lagoudakis, "An open-source scaled automobile platform for fault-tolerant electronic stability control," *IEEE Trans. Instrum. Meas.*, vol. 59, no. 9, pp. 2303–2314, 2010.

[14] W. E. Travis, R. J. Whitehead, D. M. Bevly, and G. T. Flowers, "Using scaled vehicles to investigate the influence of various properties on rollover propensity," in *Proc. IEEE American Control Conference (ACC)*, New York, 2004, pp. 3381–3386.

[15] AutoRally Platform Instructions. Accessed on: May 2017. [Online]. Available: https://github.com/AutoRally/autorally_platform_instructions

[16] AutoRally Software. Accessed on: May 2017. [Online]. Available: https://github.com/AutoRally/autorally

[17] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: An open-source robot operating system," presented at the ICRA Workshop on Open Source Software, 2009.

[18] Hector Gazebo Plugins ROS Package. Accessed on: Mar. 2017. [Online]. Available: http://wiki.ros.org/hector_gazebo_plugins

[19] C. Doniselli, M. Gobbi, and G. Mastinu, "Measuring the inertia tensor of vehicles," *Veh. Syst. Dyn.*, vol. 37, suppl. 1, pp. 301–313, 2003.

[20] M. Gobbi, G. Mastinu, and G. Previati, "A method for measuring the inertia properties of rigid bodies," *Mech. Syst. Signal Process.*, vol. 25, no. 1, pp. 305–318, 2011.

[21] M. Rozyn and N. Zhang, "A method for estimation of vehicle inertial parameters," *Veh. Syst. Dyn.*, vol. 48, no. 5, pp. 547–565, 2010.

[22] G. Genta and C. Delprete, "Some considerations on the experimental determination of moments of inertia," *Meccanica*, vol. 29, no. 2, pp. 125–141, 1994.

[23] E. Velenis, E. Frazzoli, and P. Tsiotras, "Steady-state cornering equilibria and stabilisation for a vehicle during extreme operating conditions," *Int. J. Veh. Autonomous Syst.*, vol. 8, no. 2-4, pp. 217–241, 2010.

[24] K. Lundahl, J. Åslund, and L. Nielsen, "Investigating vehicle model detail for close to limit maneuvers aiming at optimal control," in *Proc. 22nd Int. Symp. Dynamic on Vehicle*, Manchester, U.K., 2011.

[25] M. S. Burhaumudin, P. M. Samin, H. Jamaluddin, R. A. Rahman, and S. Sulaiman, "Integration of magic formula tire model with vehicle handling model," *Int. J. Res. Eng. Technol.*, vol. 1, no. 3, pp. 139–145, 2012.

[26] S. Hong, T. Smith, F. Borrelli, and J. K. Hedrick, "Vehicle inertial parameter identification using extended and unscented Kalman filters," in *Proc. IEEE Conf. Intelligent Transportation Systems*, 2013, pp. 1436–1441.

[27] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proc. Adaptive Systems for Signal Processing, Communications, and Control Symposium*, Alberta, Canada, 2000, pp. 153–158.

[28] C. You and P. Tsiotras, "Vehicle modeling and parameter estimation using adaptive limited memory joint-state UKF," in *Proc. IEEE American Control Conference (ACC)*, 2017, pp. 322–327.

[29] K. Myers and B. Tapley, "Adaptive sequential estimation with unknown noise statistics," *IEEE Trans. Autom. Control*, vol. 21, no. 4, pp. 520–523, Aug. 1976.

[30] A. H. Jazwinski, *Stochastic Processes and Filtering Theory*. Mineola, NY: Courier Corp., 2007.

[31] J. Han, Q. Song, and Y. He, "Adaptive unscented Kalman filter and its applications in nonlinear control," in *Kalman Filter Recent Advances and Applications* V. M. Moreno and A. Pigazo, Eds. Vienna, Austria: InTech, 2009.

[32] J. Qi, K. Sun, J. Wang, and H. Liu, "Dynamic state estimation for multi-machine power system by unscented Kalman filter with enhanced numerical stability," *IEEE Trans. Smart Grid*, vol. 9, no. 2, pp. 1184–1196, 2016.

[33] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *Int. J. Robotics Res.*, vol. 31, no. 2, pp. 216–235, 2012.

[34] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," Georgia Inst. Technology, Atlanta, GA, Tech. Rep. GT-RIM-CP&R-2012-002, Sept. 2012.

[35] F. Dellaert and M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *Int. J. Robot. Res.*, vol. 25, no. 12, pp. 1181–1203, 2006.

[36] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation," presented at Robotics: Science and Systems, Rome, Italy, July 2015.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[38] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016.

[39] A. Kundu, V. Vineet, and V. Koltun, "Feature space optimization for semantic video segmentation," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 3168–3175.

[40] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, 2017, pp. 2481–2495.

[41] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[42] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFS," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, 2018, pp. 834–848.

[43] M. Saha, B. Goswami, and R. Ghosh, "Two novel costs for determining the tuning parameters of the Kalman filter," in *Proc. Advances in Control and Optimization of Dynamic Systems*, Bangalore, India, 2012, pp. 1–8.

[44] C. You and P. Tsiotras, "Optimal two-point visual driver model and controller development for driver-assist systems for semi-autonomous vehicles," in *Proc. IEEE American Control Conference (ACC)*, 2016, pp. 5976–5981.

[45] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. [Online]. Available: http://tensorflow.org/