AUTONOMOUS RALLY RACING WITH AUTORALLY AND MODEL PREDICTIVE CONTROL

A PhD Dissertation
Presented to
The Academic Faculty

By

Brian Goldfain

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

August 2019

AUTONOMOUS RALLY RACING WITH AUTORALLY AND MODEL PREDICTIVE CONTROL

Approved by:

Date Approved: June 11, 2019

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Rally racing is an especially difficult driving task that requires a tight coupling of task objectives and low level control. The goal of rally racing is to drive a vehicle as quickly as possible in pursuit of the lowest course time while navigating an unpaved road at the limits of handling. We chose to study autonomous rally racing, which has broad applications to safe autonomous driving. We use a scaled self-driving vehicle testbed driven by a model predictive controller and human driver. The testbed we created is AutoRally, a robust 1:5 scale autonomous vehicle that enables the study of autonomy in the domain of aggressive driving on the real world dirt surfaces. Model predictive path integral control, a stochastic optimal controller, is tasked with driving AutoRally around a dirt track as fast as possible. To move from driving quickly to the task of racing, a second optimization layer is added on top of the controller to optimize the cost function used by the controller. The cost function optimization layer operates on the time scale of laps and uses the lap times as the reward signal. A human analogy to this setup is when an expert human driver arrives at a new race course and is allowed practice laps to tailor their skills to the particular track before a competition. We extend previous work with the model predictive controller to enable the optimization of the task description, encoded as a cost function. We investigate a variety of representations for cost function components and compare performance of sampling based optimization methods. Finally, we compare the optimized autonomous driving system against a non-optimized baseline and human drivers as the first time trial rally race between an expert human and autonomous agent where both drivers operate identical AutoRally platforms at the Georgia Tech Autonomous Racing Facility. The analysis provides insights into how the coupling of platform design, stochastic model predictive control, and cost function optimization enables autonomous rally racing in a real world setting. The data collected for the rally race provides a real world performance benchmark for state-of-the-art model predictive control with expert human driving on an open testbed.

CHAPTER 1
INTRODUCTION

Creating self-driving vehicles that are as capable as human drivers remains an open problem, despite significant advancements from universities, car manufacturers, and technology companies. The holy grail for self driving vehicles is full autonomy, defined by the Society of Automotive Engineers (SAE) as level 5 in the SAE levels of automated driving. Level 5 autonomy is 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. It is estimated that level 5 autonomous vehicles on public roads will help eliminate more than 90 percent [1] of the 35,000 annual traffic fatalities caused by human error in the United States [2], free up commute time for other activities, reduce road congestion and pollution, and increase driving resource utilization [3].

Rally racing, where a driver operates a vehicle on unpaved surfaces as quickly as possible, offers an alternative to the typical urban driving scenario in which the many scenarios that are difficult for control and perception algorithms to handle occur relatively frequently. The existence of expert human drivers for this task offers a real world performance benchmark that can be used for comparison and analysis. However, rally racing requires robust vehicles to withstand the punishment of operating a vehicle at the limits of its capabilities on unpaved surfaces. In the pursuit of creating the most capable autonomous vehicles, this work opens the domain of autonomous rally racing to roboticists. We demonstrate a custom scaled autonomous driving platform being driven by a state-of-the-art control algorithm up to the dynamics limits of the system and present a comparison to an expert human performing the same task with the same platform.

Traditionally, the self-driving vehicle software stack is broken into many modules that independently solve sub-tasks. This architecture was partially a result of the theory and hardware limitations at the time the vehicles were fielded for the DARPA Grand and Urban Challenges [4, 5, 6]. While these methods made the current wave of self-driving vehicles possible, it is now apparent that these approaches are not sufficient to achieve full autonomy, partially due to information and performance bottlenecks created by the modular design.

Recent advances in stochastic model predictive control (MPC) with the model predictive path integral (MPPI) algorithm have been shown to work well in challenging driving situations [7, 8, 9]. A key benefit of this method is that it combines some of the traditionally separate planning and control modules by creating one optimization framework that operates on a task description and dynamics model of the system to directly generate low level actuator commands. Within MPPI, the task encoding, also referred to as the cost function, is hand-coded by an expert and the dynamics model is learned from data. Cost functions encode the best guess for what the designer believes is important to solving a task, but are prone to misspecification. A current solution to achieve robust performance with hand-coded cost functions is cost shaping, or reward hacking, in which an expert iteratively modifies weights until the exhibited behavior is acceptable. This process is time-consuming and requires an expert to achieve the best performance.

An alternative approach to hand-tuning cost functions is to optimize them from data. This data-driven solution can still take advantage of domain knowledge in the representation and initialization of model parameters, if available. One potential difficulty for cost function optimization is that interactions with the real system are required to achieve the best performance because the available models and simulation tools are not accurate. Successive interactions with a real system, especially in the domain of aggressive off-road driving, require a robust hardware platform able to withstand constantly pushing the limits of the system as well as the inevitable crashes during the optimization process.

We introduce, for the autonomous racing scenario, an optimization module that is able to adapt the behavior of an underlying model predictive controller to suit the high-level task

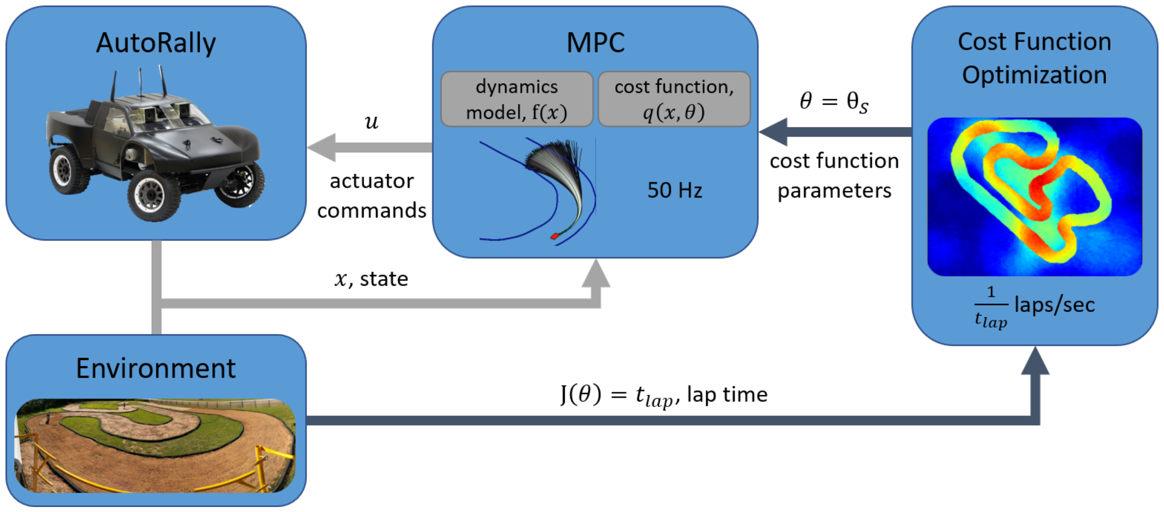


Figure 1.1: Model predictive control, cost function optimization framework for optimizing a cost function used by a model predictive controller by interacting with a system. For this work, the task of rally racing is used with an AutoRally platform driven by the MPPI algorithm at the Georgia Tech Autonomous Racing Facility.

objective through repeated interactions with the system. The platform we develop for the domain of rally racing is the AutoRally robot. Given only the stochastic model predictive controller with a pre-trained dynamics model and cost function optimized using interactions with the system to generate lap times, we achieve a level of performance than exceeds previous methods as measured by lap time. We investigate multiple representations for the cost function as well as multiple methods to optimize the cost function. Additionally, we compare the autonomous driving results with the performance of a human expert and other agents operating the same platform.

1.1 Related Work

This thesis draws on prior work from autonomous systems, stochastic optimal control, and stochastic optimization. There is a long history of custom, scaled autonomous testbeds. We attempt to summarize and discuss some of the shortcomings. We have shown in previous work involving the MPPI controller with AutoRally at the Georgia Tech Autonomous Racing Facility (GT-ARF) oval track, that the autonomous system can drive up to the mechanical limits of the system [8]. When a tube MPC formulation of the controller [9] is used at the larger GT-ARF track that includes a varied layout of turns and straights, we found empirically that no one set of hand-coded cost function parameters is sufficient for the controller to drive up to the limits of the system all the way around the track so a different approach is needed to reach the performance limit.

1.1.1 Scaled Autonomous Driving Platforms

It is vital to conduct extensive experimental testing to develop and validate technology applied to self-driving vehicles, but cost and safety considerations are major barriers that prohibit routine testing and experimentation using full-scale vehicles, especially in tasks that involve aggressive maneuvering such as racing. 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, let alone university, is prepared to conduct tests at the limits of performance where damage to the vehicle or injuries to the operator are very likely to occur. Autonomous vehicle researchers traditionally rely on custom platforms, often derived from scaled vehicles, designed for one set of experiments. Below, we briefly summarize some prior efforts in scaled autonomous vehicle research. This review of prior work provides context for the current activities. Full size platforms based on passenger vehicles are not discussed in this section, as any attempt to summarize that ecosystem would be out of date in a matter of months given the current pace of development.

Scaled platforms constructed from modified RC cars are popular in the academic and hobby communities. These platforms are typically 0.2 m to 1 m long and weigh between 1 kg 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 is 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.

Documentation for open source platforms normally includes parts lists, build instructions, and interface software for the sensors and actuators. Availability of tutorials, simulation environments, and public datasets vary by project. Common sensors include wheel speed, inertial measurement unit (IMU), cameras, depth sensors, ultrasonic, and light detection and ranging (Lidar) units. The target environment for these platforms is typically indoors on a smooth surface. The Donkey Car [10] is an easy to build 1:16 scale autonomous platform for the DIY Roborace events targeted at hobbyists. Onboard computing and sensing are a Raspberry Pi 3 with a matching wide angle camera. The Berkeley Autonomous Race Car (BARC) [11] 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 [12] and accompanying Autonomous Racing Competition allows teams to race against one another 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) [13] from 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 1.1 provides a comparison of these open source scaled platforms.

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 position system (GPS) device, which is a common sensor for outdoor vehicles. Instead of GPS, global position information

Table 1.1: 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 does not include 3D 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 are indicated.

Platform	Cost [$]	Build Time [h]	Weight [kg]	Computing	Simulation	Data Sets
Donkey Car	200	2	2	Raspberry Pi	Y	Y
BARC	500	3	3.2	Odroid XU4	Y	Y
MIT RACECAR	3,383	10	4.5	Nvidia Jetson	Y	N
F1/10	3,628	3	4.5	Nvidia Jetson	N	Y

can be provided by instrumenting the environment such as 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 sensor field of view and resolution restrictions, and 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. Off board computation introduces its own set of problems including increased latency, 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 limits 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 [14], a model predictive control (MPC) algorithm running on a stationary desktop computer with a motion capture system has been 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 [15] and [11], and to push scaled autonomous driving to the friction limits of the system in [16]. A framework with multi-fidelity simulation and accompanying hardware platform for use in reinforcement learning problems relating to autonomous driving was presented in [17]. A 1:5 scale autonomous platform was developed to investigate stability control in [18, 19]. While these platforms were successfully used for the experiments in their respective publications, there is not enough public information available to be able to build, operate, and program one without essentially starting from scratch.

Traditionally, scaled autonomous driving platforms were purpose-built for one experiment, but a new wave of open source platforms are 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. Scaled platforms therefore show great promise in the wide variety of experiments they enable, but previous attempts fall short in terms of design, fidelity, and repeatable performance.

Computer simulations offer an alternative to testing with full-scale vehicles or scaled models. Indeed, the fidelity of computer simulations has improved significantly in recent years [20, 21]. They can be used to generate an almost unlimited amount of simulated driving conditions cheaply, but 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. 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 such as difficult driving dynamics.

1.1.2 Autonomous Driving

Several different approaches have been taken to investigate the problem of aggressive autonomous driving, and autonomous driving in general. In [**laurense2017path**, 22, 23], analytic approaches are explored. The performance limits of a vehicle are pushed using a simple model-based feedback controller and extensive precomputation of a racing line to follow around a track. [14], showed the benefits of model predictive control on a 1:10 scale radio-controlled car following waypoints through a challenging obstacle course. The benefits of real-time MPC in an outdoor, dirt environment are shown in [8, 9].

All of these methods rely on a fixed description of the task which is prespecified by a human or extensive precomputations. In all cases, there is no opportunity to adapt performance during additional interactions to improve performance. Further, models used to precompute trajectories inevitably are brittle in the face of disturbances and unable to account for changes to the environment.

More recently, [24] directly learns affordances necessary for autonomous driving by a low level controller and [25] uses an end-to-end approach to learn a mapping from images to steering commands from about 20 minutes of human driving. However, these end-to-end approaches cannot be used to drive with a model predictive control framework such as [8] used for aggressive control, and the lack interpretability of the final systems precludes their usefulness for creating safer self-driving vehicles.

An analysis of expert human race car drivers was conducted in [26]. This work also includes a public dataset of sensor and control information logged during testing. While the data is valuable for training and testing a model offline, the testing platform is inaccessible so new results cannot be generated. Driving behavior for a novice and expert given the same task of driving as fast as possible in a circular pattern was compared in [27] along with a method to instrument a full size car for data capture.

Some early evaluation and examples of autonomous driving systems were developed at Carnegie Mellon University including Navlab [28, 29] and projects to understand high speed autonomous navigation [30] and the combination of multiple goals to achieve a desired task [31, 32].

1.1.3 Cost Function Optimization

A classical approach to optimization in the reinforcement learning community is policy gradient methods [33], of which finite-differencing methods are popular, also sometimes referred to as the simultaneous perturbation algorithm for stochastic optimization (SPSA) [34]. SPSA is a straightforward algorithm to implement and can handle stochastic and deterministic policies, but it is mostly limited to simulation with a deterministic world. The parameter perturbations must be performed very carefully as bad choices tend to cause instability in the system. Policy gradient methods also struggle to make optimization progress in the face of noise in the real system. Therefore, policy gradient methods are generally unsuitable for the application of aggressive off-road racing operating in the real world with significant noise in the system.

More recently, the Cross Entropy Method (CE) has been used for motion planning [35] and path integral policy improvement (PI^2) [36]. Both are appealing sampling-based optimization method because they can handle arbitrary cost functions, which matches the generality of the underlying model predictive path integral controller used to drive the AutoRally platform. A common criticism of CE is fast convergence, which can be mitigated, to some extend, through covariance matrix adaptation, which sets the sampling covariances at each iteration according to how large of a performance step was achieved. The resulting algorithm, known as Cross Entropy Covariance Matrix Adaptation (CE-CMA) [36] works well with system noise and is much less sensitive to the initial choice and parameter sampling scheme than policy gradient methods. A common criticism of the method is premature convergence, potentially to a local minimum.

CHAPTER 2
THE AUTORALLY PLATFORM

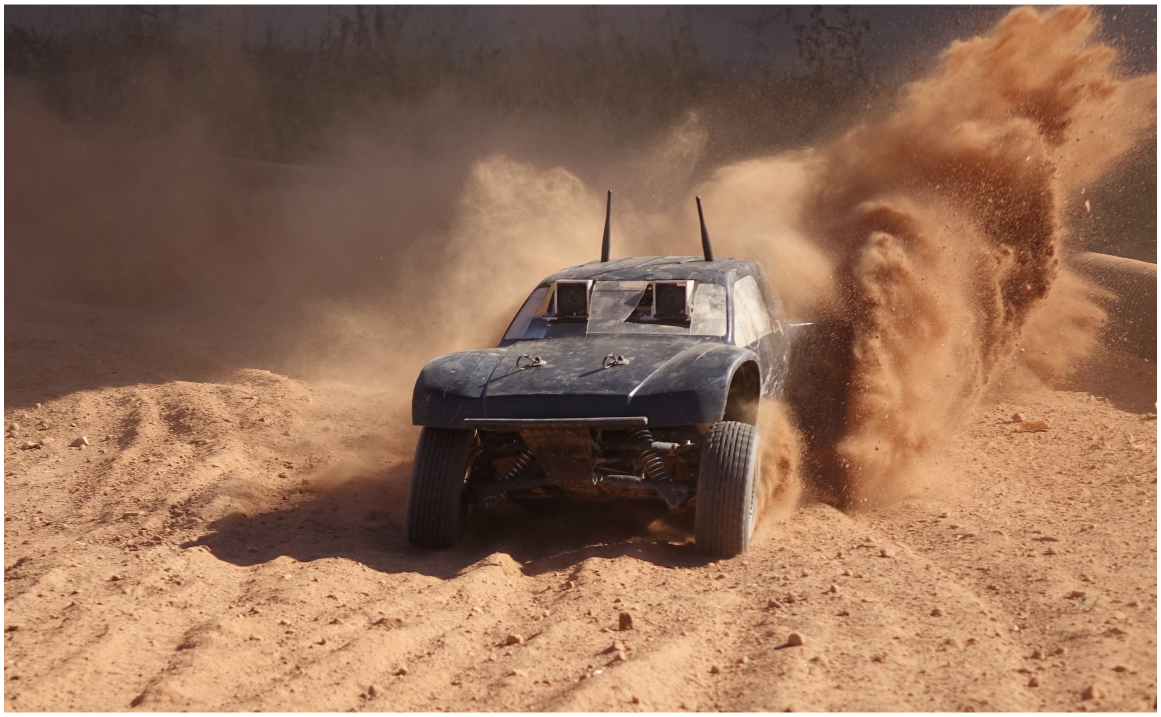


Figure 2.1: Autonomous driving with AutoRally at the Georgia Tech Autonomous Racing Facility.

This section describes AutoRally [37], shown in Figure 2.1, a 1:5 scale robotics testbed developed at Georgia Tech for autonomous vehicle research. 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 then 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 [38]. In addition, operating procedures, a simulation environment, core software and reference controllers written with the Robot Operating System (ROS) [39] in C++ and Python, along with a collection of human and autonomous driving data are publicly available [40]. To date, the fleet of six AutoRally robots at Georgia Tech have collectively driven under autonomous and manual control at the two GT-ARF tracks for hundreds of kilometers. The complete system diagram for the AutoRally robot with a remote operator control station (OCS) is shown in Figure 2.3.

Construction time for an AutoRally chassis is 40 hours, and 60 hours for a compute box. The full platform construction time of 100 hours does not include custom part fabrication time that depends on the tools available. AutoRally can take significantly less than 100 hours to construct if you already have experience with radio controlled (RC) 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 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 (STL) 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 by using the provided portable document format (PDF) files printed on US 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 (DXF) files and bend patterns included in the instruction materials.

As a research tool, AutoRally has been part of numerous publications [7, 41, 42, 8, 43, 9, 44, 45, 46, 37, 47]. The remainder of this chapter describes the main components that comprise the AutoRally platform, broken into hardware and software sections.

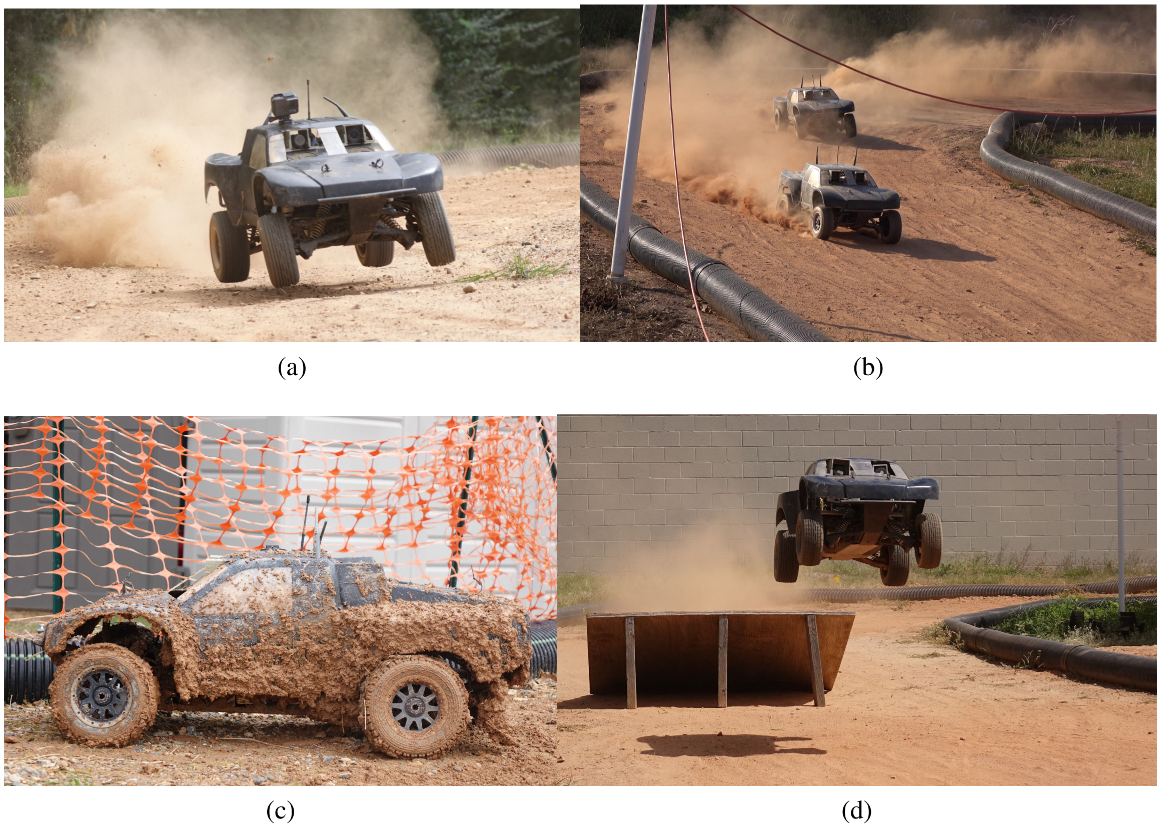


Figure 2.2: AutoRally platform during autonomous experiments at the Georgia Tech Autonomous Racing Facility tracks. (a) Airborne from bouncing over rough terrain. (b) Two platforms driving autonomously. (c) At the end of a muddy day of testing. The platform carried 12 kg of mud stuck to the inside of the body and chassis, increasing the total weight by almost 50 %. (d) AutoRally platform airborne during a jump.

2.1 Hardware

AutoRally is based on a 1:5 scale RC truck and is approximately 1 m long, 0.6 m wide, 0.4 m high, weighs almost 22 kg, and has a top speed of 90 kph. 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 while retaining a large payload capacity. The AutoRally capabilities offer a large performance improvement over traditional scaled autonomous vehicles without the need for large infrastructure investments and safety considerations of full sized autonomous vehicles.

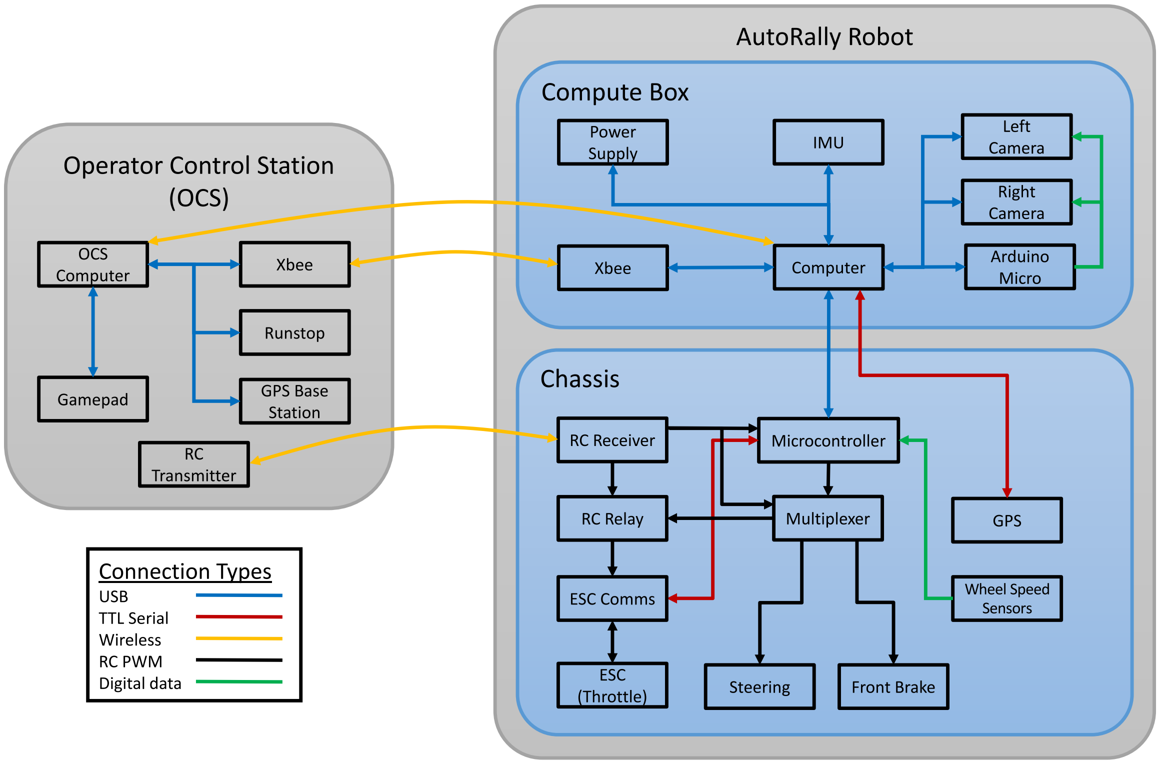


Figure 2.3: AutoRally system diagram. All major electronic system components and their connections to the rest of the system are shown. The setup includes the AutoRally robot composed of a chassis and compute box along with a remote operator control station connected by three wireless communication links.

2.1.1 Chassis

The chassis is designed as a self-contained system that can easily interface to a wide variety of computing and sensor packages. Similarly 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.

1:5 Scale Radio Controlled Truck

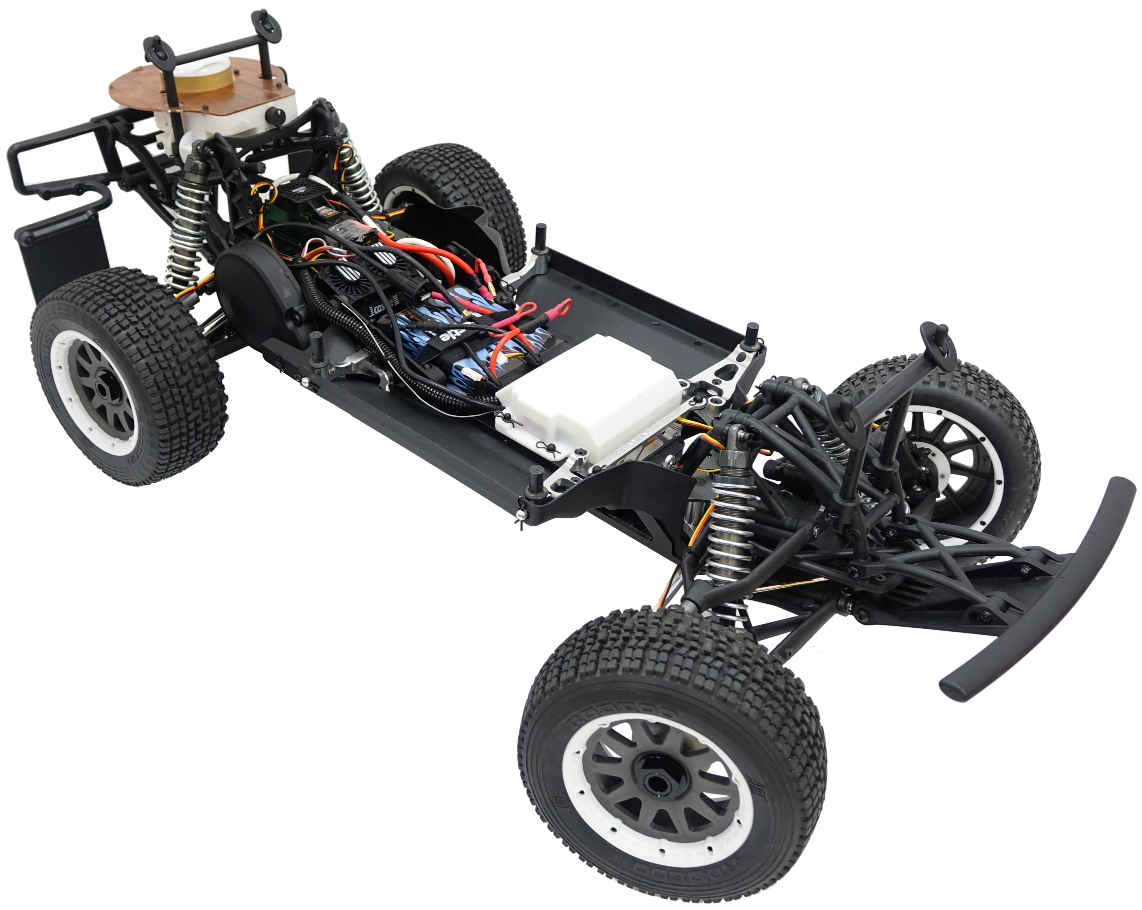


Figure 2.4: Assembled AutoRally chassis.

The chassis is based on a HPI Baja 5SC RC trophy truck. Figure 2.4 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 2-stroke gasoline engine with a 10 hp peak output electric motor and ESC from Castle Creations. Compared to the stock engine, the electric motor is much more powerful, more responsive, and more 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, 4-cell 14.8 V 6500 mAh Lithium-Polymer (LiPo) batteries connected in series. A full charge lasts between 20 and 90 minutes, 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 in order to improve lateral stability and make room to mount the front brakes and the wheel rotation sensors.

The stock suspension springs were replaced with stiffer springs 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 lb/in to 15 lb/in and the rear spring constant from 11.17 lb/in to 19.09 lb/in. The shock oil viscosity was also increased roughly 58% from 500 cSt to 850 cSt to properly damp the upgraded springs.

The stock 2-channel transmitter was replaced with a programmable 4-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 Safety System section.

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 magnets. The magnet can trigger the sensor from up to 0.58 cm away. Larger magnets could be used to increase the maximum tripping distance but 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. Inter-magnet 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 is 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 pulse width modulation (PWM) signals with a duty cycle from 1 ms to 2 ms, with a neutral value of 1.5 ms. The servos do not have position feedback.

Custom Components

The custom 3D printed 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.

2.1.2 Compute Box

Most modern control and perception algorithms are CPU- and GPU-intensive. In order 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 is designed to withstand a 10 g direct impact from any angle without damaging internal electronic components and fabricated out of 2.286 mm thick 3003 aluminum sheet. The 10 g impact is larger than impacts experienced when testing at GT-ARF according to IMU data that include collisions with fixed objects and rollovers. The box's impact tolerance was verified with finite element analysis (FEA) 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 (TIG) welded to the bottom to accurately join the large panels of relatively thick aluminum sheet without leaving gaps. Aluminum dust filters coupled with a foam membrane allow continuous airflow through the box while keeping out environmental contaminants such as dust and rocks. Combined with the all-aluminum exterior, the assembled compute box provides excellent electromagnetic interference (EMI) containment.

The cameras and lenses are mounted facing forward on the top of the box on an aluminum plate for rigidity and 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.

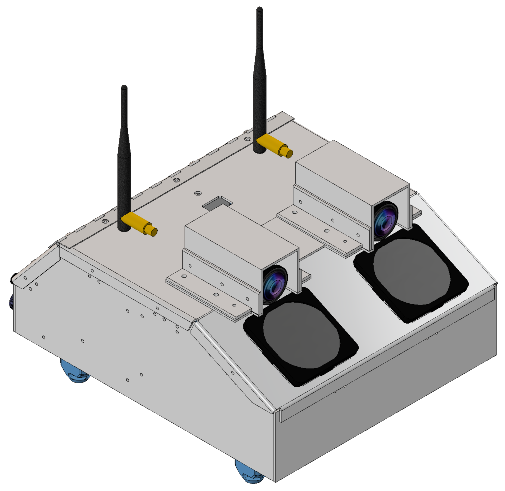
Four 3D printed components are inside the compute box: a battery holder, a SSD holder, a GPU holder, and a RAM holder. The battery holder tightly secures the compute box battery and power supply. The battery slot is slightly undersized and lined with foam so that the battery press-fits into the mount and can be removed for charging and maintenance without removing any internal screws. The SSD holder is used to securely mount a 2.5 inch

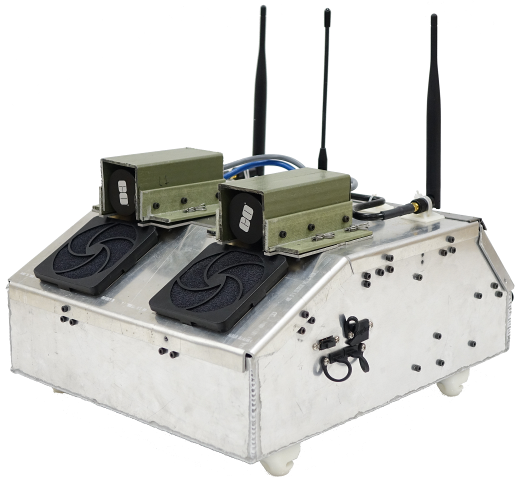
SSD to the side wall of the compute box. The GPU holder fits over the GPU and secures it to the main internal strut while still allowing adequate airflow.

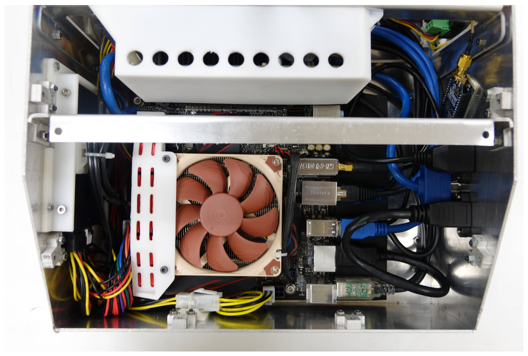
The compute box attaches to the chassis with four 3D printed mounts attached to the bottom of the compute box. The mounts fit over vertical posts on the chassis rail guards and are secured to the chassis with a cotter pin though the mount and post. Special consideration was taken to design the mounts 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 FEA 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 and leaves the internal components undamaged. All panel mount components such as the power button, LEDs, 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 (max 1 kHz) and fused orientation estimates at 200 Hz (max 500 Hz). The two machine vision cameras are mounted on top of the compute box are Point Grey Flea3 FL3-U3-13E4C-C color cameras with a global shutter that run up to 60 Hz. Lenses are 70 degree field of view (FOV), 4.5 mm fixed focal length. Each camera connects to the motherboard with a USB3.0 cable and is externally triggered by an Arduino Micro microcontroller with the general purpose input/output (GPIO) connector. Both cameras are connected to the same trigger signal which runs at a configurable rate. Internal battery voltage and computer temperature sensors are used to monitor system health.


(a)


(b)


(c)

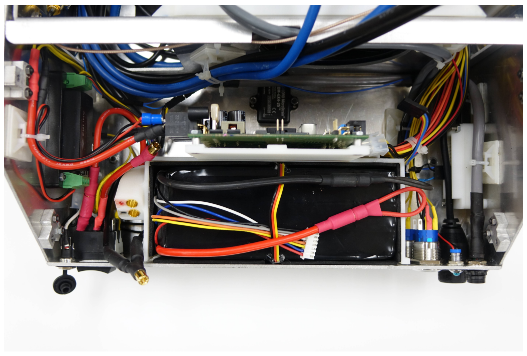

(d)

Figure 2.5: AutoRally mini-ITX compute box. (a) Assembled computer-aided design (CAD) model. (b) Fully assembled compute box ready to be mounted on a chassis. (c) Front of compute box viewed from above with motherboard and compute components visible. (d) Rear of compute box viewed from above with power system components visible.

Table 2.1: AutoRally compute box components. Major computing and power parts are listed along with their specifications.

Component	**Detail**
Motherboard	Asus Z170I Pro Gaming, Mini-ITX
CPU	Intel i7-6700, 3.4 GHz quad-core 65 W
RAM	32 GB DDR4, 2133 MHz
GPU	Nvidia GTX-750ti SC, 640 cores, 2 GB, 1176 MHz
SSD storage	512 GB M.2 and 1 TB SATA3
Wireless	802.11ac WiFi, 900 MHz XBee, and 2.4 GHz RC
Power supply	Mini-Box M4-ATX, 250 W
Battery	22.2 V, 11 Ah LiPo, 244 Wh

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 enables the reconfiguration of sensing and computing payloads without mechanical modifications as requirements evolve. Table 2.1 lists the details of the compute box components. WiFi 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 about 2 Hz to improve GPS performance. The XBee radio onboard 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 message and RTK correction messages to all AutoRally robots in 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.

2.2 Software Interface

The AutoRally software was designed to leverage existing tools wherever possible. All computers in the system run the latest long term support (LTS) 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) [39]. ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify 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, that enable a safe and robust autonomous system. They are a combination of electronics and software. The software interface, OCS graphical user interface (GUI), and simulation environment for the robot are also presented.

2.2.1 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 2.6 shows how timing information is propagated for time synchronization. The time source for the entire system is the GPS board on the chassis which emits National Marine Electronics Association (NMEA) 0183 messages and a pulse-per-second (PPS) signal. The PPS signal provides a marker that is 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 supported by

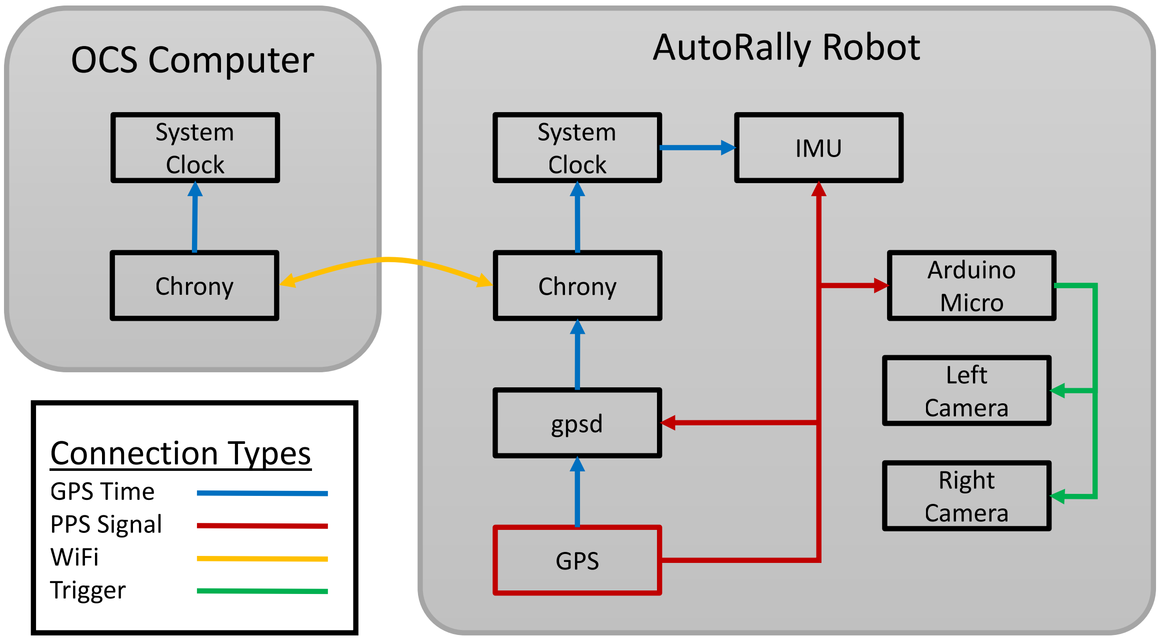


Figure 2.6: AutoRally time synchronization diagram. Clocks on computers and sensors that support clock control are synchronized to global positioning system (GPS) time with a combination of the pulse-per-second (PPS) signal that marks the beginning of a second, and timing messages that identify which second the PPS signal represents. On the AutoRally robot and operator control station (OCS) computers, the system utilities gpsd and Chrony are used for clock synchronization. The two cameras rely on an external tigger signal to capture frames at the same time. The Arduino Micro microcontroller generates the camera trigger signal.

devices that require time synchronization. The PPS signal is routed into GPSD running on the motherboard, the IMU, and the 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 (NTP) 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 WiFi 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 Jan 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 less than one millisecond to system time.

The cameras provide an external trigger interface to control when each image is captured. The Arduino Micro is used to provide the cameras with the triggering pulse at a specified frame rate. Each time a PPS pulse comes 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.

2.2.2 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 2.7, are a wireless deadman relay located in the electronics box 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 which circumvent the Wifi, XBee, and software control on the compute box by using the additional RF link between the RC transmitter and receiver located in the electronics box of the chassis. The deadman 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 deadman 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 2.8, using incoming runstop ROS messages to enable and disable software control

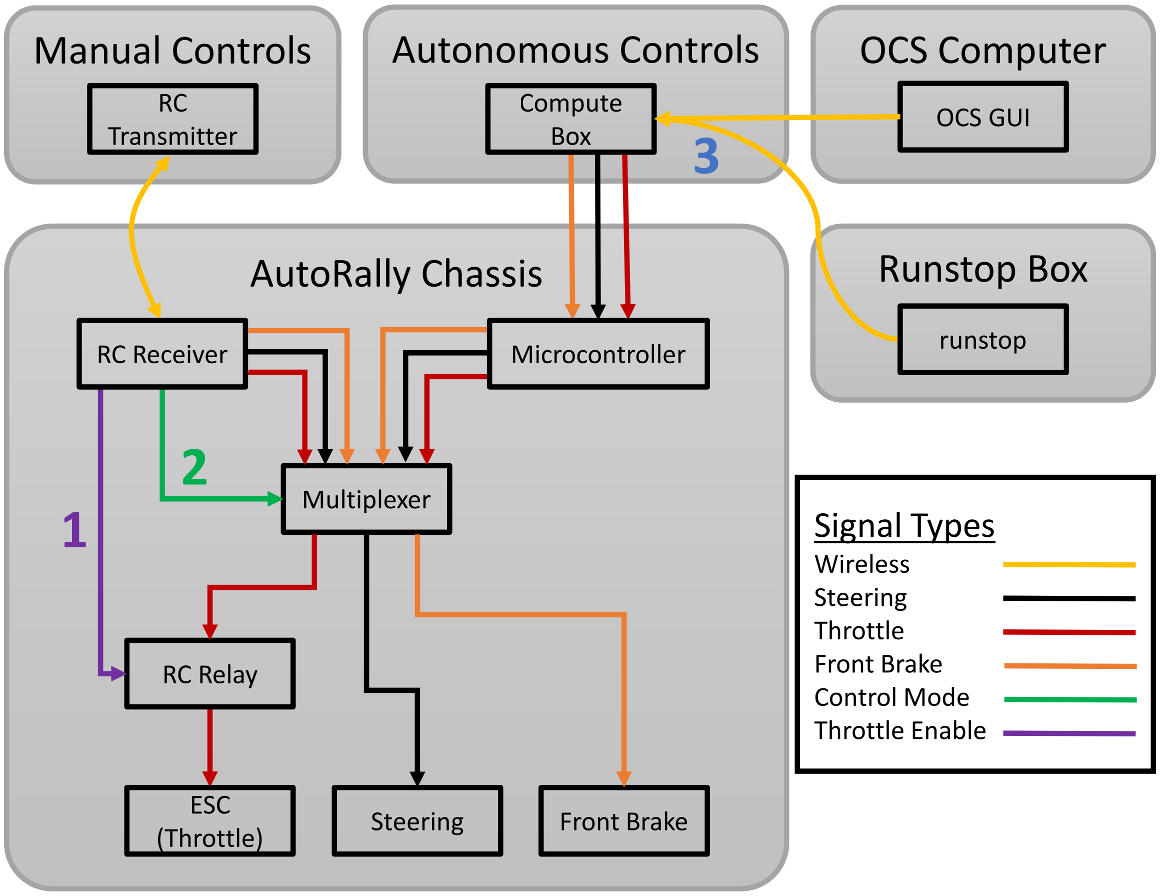


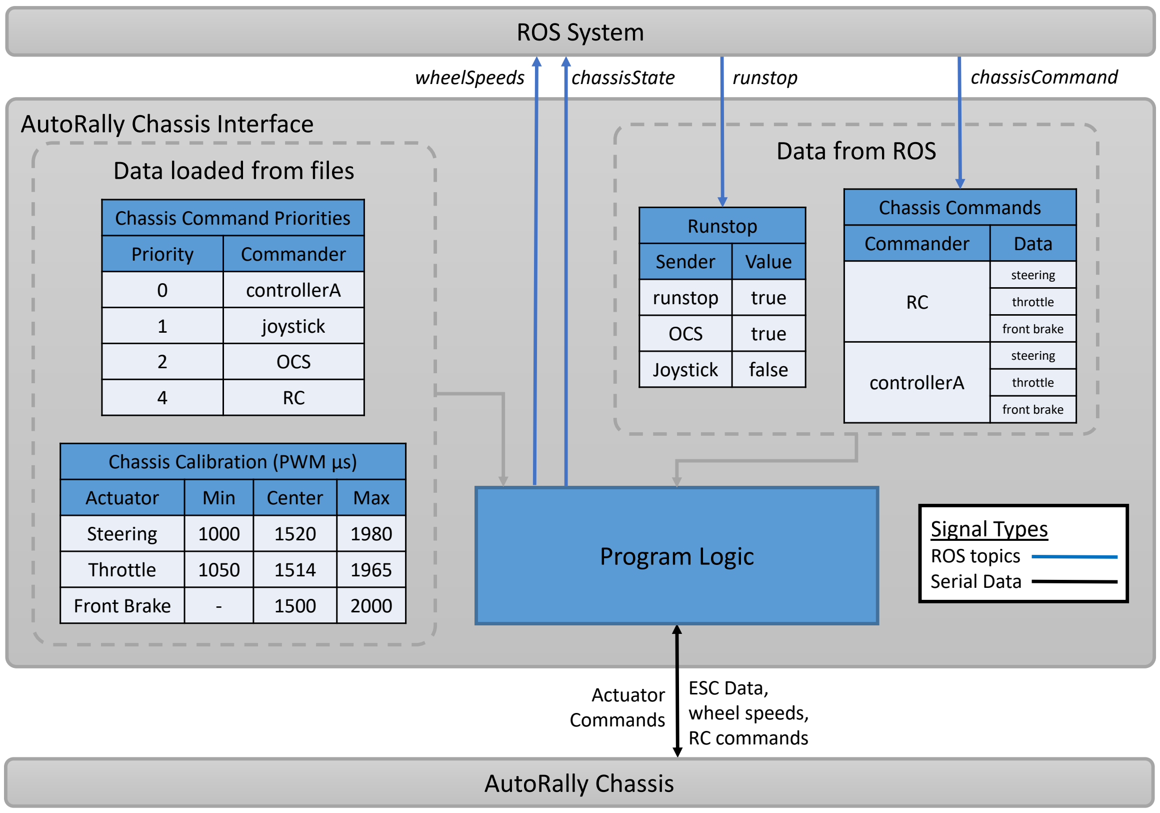
Figure 2.7: AutoRally safety system. The human operated radio control (RC) transmitter sends signals to the RC receiver in the AutoRally chassis. The RC receiver provides actuator signals from the human driver, controls a safety relay to enable and disable the throttle signal into the electronic speed controller (ESC), and switch 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.

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 is transmitted over WiFi 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 is 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 controls autonomous motion for all of the robots simultaneously. The AutoRally 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.

2.2.3 AutoRally Chassis Interface

The AutoRally chassis interface software is implemented as a ROS nodelet and communicates with the micro-controller 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



Chassis Command Priorities

Priority	Commander
0	controllerA
1	joystick
2	OCS
4	RC

Chassis Calibration (PWM µs)

Actuator	Min	Center	Max
Steering	1000	1520	1980
Throttle	1050	1514	1965
Front Brake	-	1500	2000

Runstop

Sender	Value
runstop	true
OCS	true
Joystick	false

Chassis Commands

Commander	Data
RC	steering
	throttle
	front brake
controllerA	steering
	throttle
	front brake

Figure 2.8: AutoRally chassis interface program information flow. The program, which runs on the compute box onboard 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 (USB) 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.

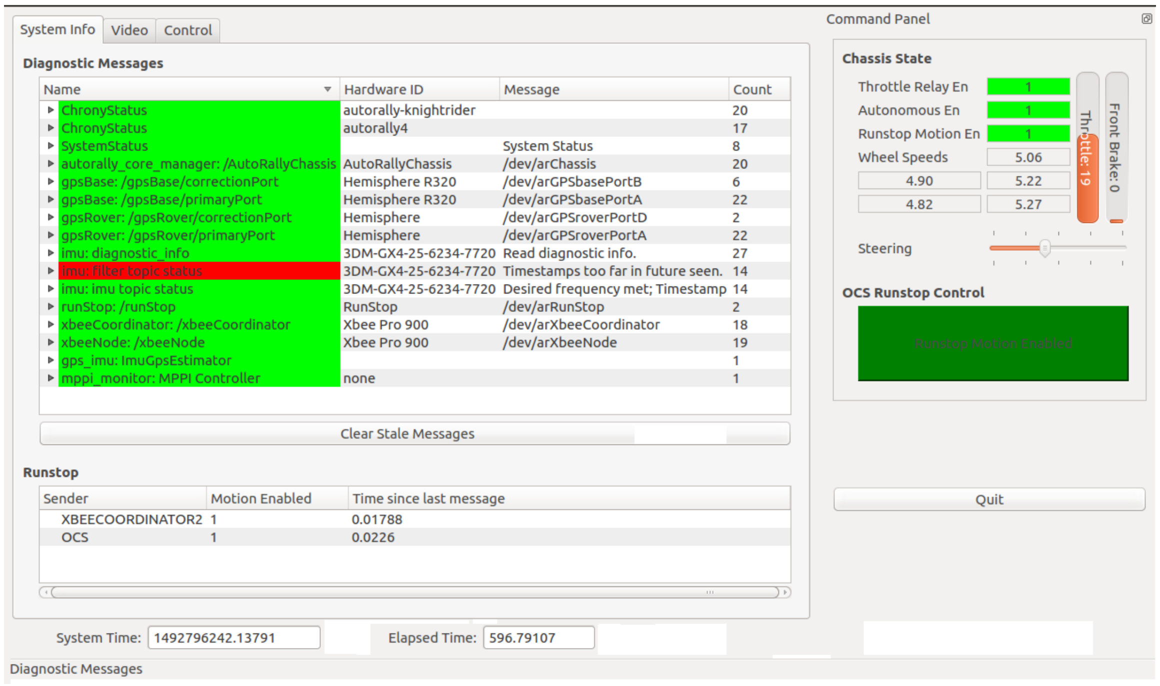
interface software. Specified in the file is the minimum, center, and maximum pulse width for each actuator in μs. When properly calibrated, a */chassisCommand* ROS message on any AutoRally platform will elicit the same behavior. For example, commanding a steering value of zero will make any calibrated 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 1 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 0 for no brake to 1 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 between controllers and use multiple controllers 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.

2.2.4 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 which 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 travels over a local WiFi network.

The OCS GUI also provides an interface for direct control of the robot's actuators



Diagnostic Messages

Name	Hardware ID	Message	Count
ChronyStatus	autorally-knightrider		20
ChronyStatus	autorally4		17
SystemStatus		System Status	8
autorally_core_manager: /AutoRallyChassis	AutoRallyChassis	/dev/arChassis	20
gpsBase: /gpsBase/correctionPort	Hemisphere R320	/dev/arGPSbasePortB	6
gpsBase: /gpsBase/primaryPort	Hemisphere R320	/dev/arGPSbasePortA	22
gpsRover: /gpsRover/correctionPort	Hemisphere	/dev/arGPSroverPortD	2
gpsRover: /gpsRover/primaryPort	Hemisphere	/dev/arGPSroverPortA	22
imu: diagnostic_info	3DM-GX4-25-6234-7720	Read diagnostic info.	27
imu: filter topic status	3DM-GX4-25-6234-7720	Timestamps too far in future seen.	14
imu: imu topic status	3DM-GX4-25-6234-7720	Desired frequency met; Timestamp	14
runStop: /runStop	RunStop	/dev/arRunStop	2
xbeeCoordinator: /xbeeCoordinator	Xbee Pro 900	/dev/arXbeeCoordinator	18
xbeeNode: /xbeeNode	Xbee Pro 900	/dev/arXbeeNode	19
gps_imu: ImuGpsEstimator			1
mppi_monitor: MPPI Controller	none		1

Clear Stale Messages

Runstop

Sender	Motion Enabled	Time since last message
XBEECOORDINATOR2	1	0.01788
OCS	1	0.0226

System Time: 1492796242.13791

Elapsed Time: 596.79107

Diagnostic Messages

Chassis State

Throttle Relay En	1
Autonomous En	1
Runstop Motion En	1
Wheel Speeds	5.06
4.90	5.22
4.82	5.27

Throttle: 19

Front Brake: 0

Steering

OCS Runstop Control

Runstop Motion Enabled

Quit

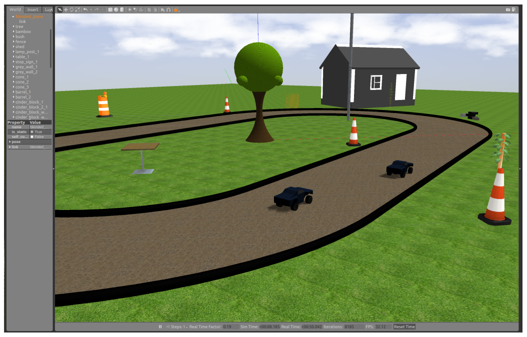
Figure 2.9: Operator Control Station (OCS) graphical user interface (GUI) that display diagnostic, sensor, and actuator information in real time to a remote operator, captured during autonomous testing.

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.

2.2.5 Simulation

Despite the robust AutoRally hardware platform, there are still high-risk maneuvers and software testing that are best run in a simulation environment before executing them on the physical platform. A 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 environments, shown in Figure 2.10 are based on Gazebo. Gazebo is a robot simulator with tight ROS integration that includes graphical interfaces and multiple physics engines to choose from. The AutoRally Gazebo simulation environments and robot model match their real-world counterparts and support the same software interface through ROS messages as the real hardware. The simulated track environments are the


(a)

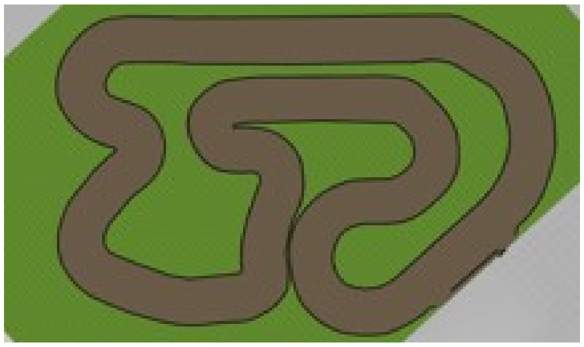

(b)

Figure 2.10: Simulation environments for AutoRally built using Gazebo and modelled after the Georgia Tech Autonomous Racing Facility tracks. Multiple AutoRally robots can be simulated together and each simulated platform has the same Robot Operating System (ROS) 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. (a) Small oval track. (b) Large track.

same size and configuration as the GT-ARF small oval and large tracks. For the simulated AutoRally platform, the steering servo and Ackermann linkage of the 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 simulation are measured from the steering servo specification provided by the manufacturer and by measuring 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 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 (PID) controller on a linear actuator with a target set-point 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 come from the hector_gazebo_plugins ROS package from TU Darmstadt [48] and are configured according to the specifications of their physical analogs. We developed our own wheel speed sensor node, as 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 as part of the hardware and software infrastructure that allows for a smooth testing of software with ROS and the AutoRally platform, and to easily debug the control, perception, and communication software. The fidelity of computer simulations has improved significantly in recent years, but they still cannot completely replicate the complex interactions of an autonomous vehicle with the real world. Specifically for the task of racing on dirt roads, the tire-ground interactions are not handled well by the physucs engine. Therefore, simulations are a helpful, but not final, step on the path to proving self-driving technologies in the real world.

2.3 Georgia Tech Autonomous Racing Facility

To enable testing with AutoRally platforms, Georgia Tech Autonomous Racing Facility (GT-ARF) has been created with two tracks on Georgia Tech propetry. Both tracks have the same dirt surface and boundaries made of six inch diameter black corrugated pipe and infrastructure on site such as power, water, and storage for equipment. An oval track is located on main campus and a large track with much more complex features, is located at Gerogia Tech's Cobb County Research Facility, which is a 20 minute drive North from main campus. The oval track covering an area of 30 m by 15 m, with a track surface 3 m wide and centerline length of 68 m. The second track, which is much larger and the location for the experimental results with the AutoRally platform in this work, is designed to have a variety of road features, and covers an area of 42 m by 30 m, and has a track surface 4.5 m wide and a centerline length of 183 m.

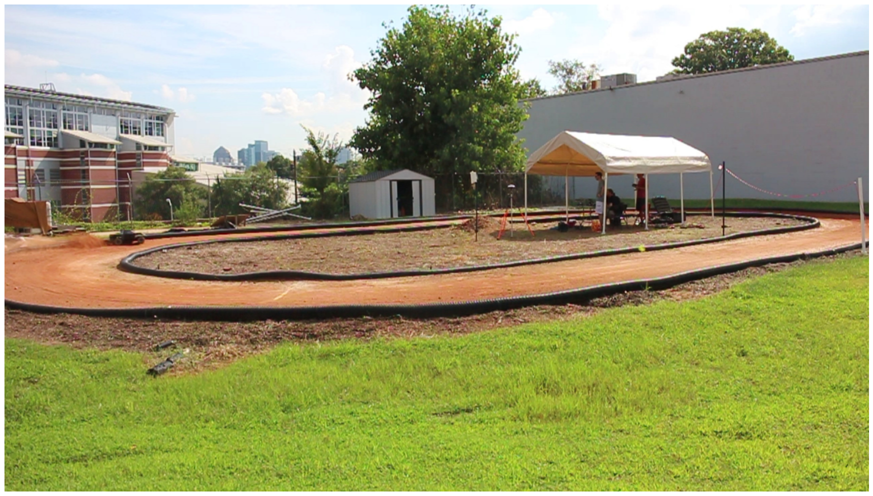


Figure 2.11: Georgia Tech Autonomous Racing Facility oval track located on main campus.

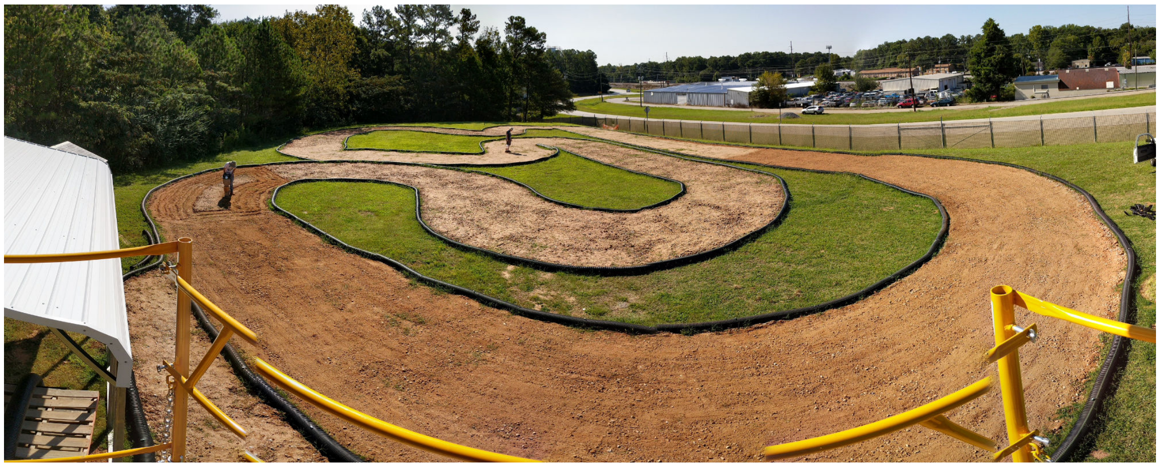


Figure 2.12: Georgia Tech Autonomous Racing Facility large track located at Cobb County Research Facility.

CHAPTER 3
STOCHASTIC MODEL PREDICTIVE CONTROL

In order for AutoRally to be driven autonomously, control method is required that can handle the nonlinear dynamics, general cost criteria, and operate in real time onboard the AutoRally platform. Model predictive control (MPC) is an increasingly active area of research in control theory for nonlinear systems [49]. In traditional MPC problems, the optimization is tasked with a stabilization or tracking task, but in the case of autonomous rally racing the task is much more complicated. This objective complexity increases the computational cost of the optimization, a major problem since optimization must occur in real time. A tractable approach is receding-horizon differential dynamic programming (DDP) [50], which is capable of controlling complex animated characters in realistic physics simulators and has been shown to work well with AutoRally [44]. However, DDP still requires a differentiable cost function, which restricts the ability to encode hard constraints such as enforcing the constraint of staying within track boundaries for the racing case. A more flexible MPC method is model predictive path integral (MPPI) control, a sampling-based algorithm which can optimize for general cost criteria, convex or not, and has been shown to work well on a variety of tasks in simulation including the canonical cart-pole swing-up task, quadcopter navigation, helicopter landing, and in the real world driving aggressively in an off-road setting with AutoRally [7, 8, 46].

The remainder of this chapter presents MPPI with an emphasis on the task description, encoded as a cost function, used during online optimization of control plans in the next chapter. MPPI is the control method we have selected to operate AutoRally in the autonomous rally racing domain and we have implemented and tuned the MPPI controller on the AutoRally platform at GT-ARF.

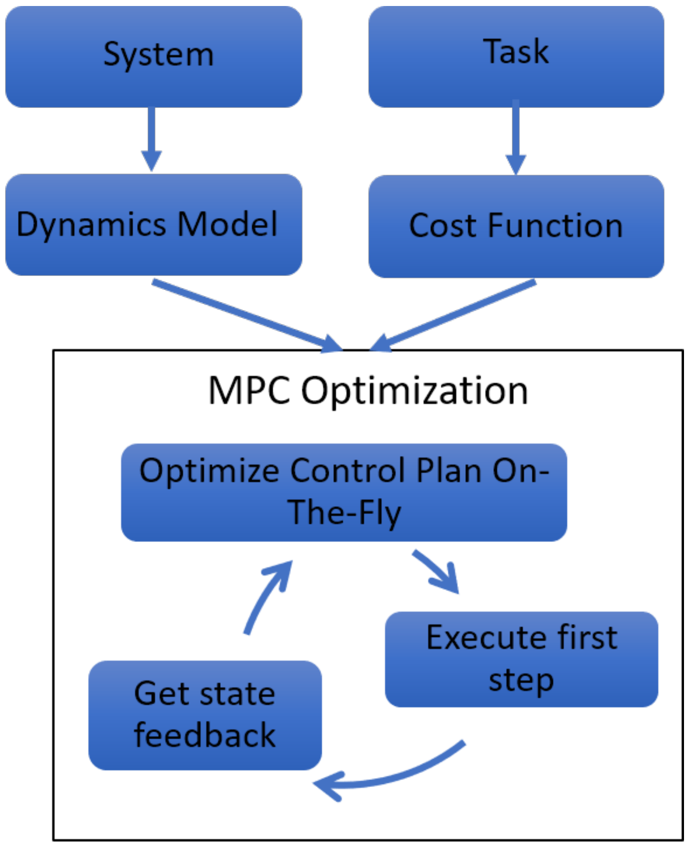


Figure 3.1: Model predictive path integral control. MPPI is a stochastic nonlinear model predictive control method that optimizes control plans on-the-fly and can handle general dynamic models and cost criteria. The algorithm executes the first step of its control plan, receives state feedback from the system, then re-optimizes the control plan using a dynamics model of the system and task description, encoded as a cost function.

3.1 Model Predictive Path Integral Control

In MPC, optimization and execution take place simultaneously: a control sequence is computed, and then the first element of the sequence is executed. This process is repeated using the un-executed portion of the previous control sequence as the importance sampling trajectory for the next iteration. The key requirement for sampling-based MPC is to produce a large number of samples in real time. Sampling is performed in parallel on a graphics processing unit (GPU) using Nvidia's CUDA architecture. Additionally, MPPI does not rely on derivatives, and does not impose any restrictions on the structure of the cost function or dynamics model.

3.1.1 Dynamics Model

Prior work used a dynamics model inspired by a a bicycle model augmented with slip terms. This model was fit from data collected on the real system, and was used, in part, because the original MPPI derivation required the dynamics model to be control-affine. While physics-based model proved useful, it was not very accurate on dirt surfaces and struggled to capture the wide range of dynamics exhibited by AutoRally across all regimes in the high-speed dirt driving task including sliding. The more general information theoretic derivation of MPPI [8], eliminated the control-affine restriction and demonstrated that a neural network dynamics model can provide a comparable level of performance to the physics-based approach. The current version of MPPI uses a vehicle dynamics model represented as two-layer neural network, learned from data, to optimize a series of control actions under a given cost function at 50 Hz on a low power desktop GPU.

To train the model, we collected a system identification dataset of approximately 30 minutes of human-controlled driving at speeds varying between 4 and 10 m/s at the large GT-ARF track. 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), (3) linear acceleration maneuvers which consist of accelerating the vehicle as much as possible in a straight line, and then braking or coasting to a stop, (4) drifting maneuvers, where the pilot attempts to slide the vehicle as much as possible, and (5) high speed driving where the pilot simply tries to drive the vehicle around the track as fast as possible. Each one of these maneuvers was performed by a skilled, but non-expert driver, for three minutes while moving around the track clockwise and for another three minutes moving counter-clockwise.

3.1.2 Cost Function

The cost function is of particular interest to this work. The cost function for the task of driving continuously around a dirt track consists of several parts:

$$q(x) = \omega_1 C_T(p_x, p_y) + \omega_2(v_x - v_x^d)^2 + \omega_3 0.9^t I + \omega_4 \left(\frac{v_y}{v_x}\right)^2. \tag{3.1}$$

where term (1), $C_T(p_x, p_y)$, is the positional cost of being at the body frame position (p_x, p_y). This term is currently determined using the estimated vehicle state and a pre-surveyed map. Term (2), $(v_x - v_x^d)^2$, is a cost for achieving a desired forward speed, in the body frame, v_x^d, set by the operator and fixed for the duration of a test. Term (3), I, is an indicator variable which is turned on if the track cost, roll angle, or heading velocity become too high, and (4), $\left(\frac{v_y}{v_x}\right)^2$, is a penalty on the slip angle of the vehicle. The coefficient vector used in our experiments was $w = (200, 4.25, 10000, 100)$. Note that the first two terms encode the objective of staying on the track surface and traveling at a desired speed. The last two terms penalize undesirable behavior such as driving on to non-driving surface, rolling over, or sliding out, and are trivial to compute given the vehicle's state estimate.

During the cost function optimization work in the next chapter, we look to optimize the speed cost component in the cost function. The other terms in the cost function are not optimized because the task constraints do not change: the robot is still not allowed to crash or slide out, which would change the direction of travel around the track, for autonomous rally racing. All other parameters that influence the behavior of the controller remain fixed for all tests, including the sampling variances for throttle and steering within the rollouts, and the terminal cost of each rollout. This is due to our exploration of cost function optimization as a black box optimization tool for an MPC. If components of the MPC are changing during optimization, that would create a non-stationary distribution we are trying to optimize for, which is a potential area for future investigation, but not the focus of the work contained in this thesis.

3.2 Cost Function Representation for Optimization

An update to the form of the cost function used in prior work is required to use MPPI in the cost function optimization framework. The task constraints encoded in the cost function, the crash and slip penalties, are kept the same because, for the task of racing, the vehicle still cannot crash or change direction of travel on the track. Prior work represented the speed cost as $\omega_2(v_x - v_x^d)^2$, penalizing deviation from a desired speed single value set by the human operator, that is fixed for the duration of an experiment. In practice, the value was used as a desired upper speed limit for the vehicle during a testing. This worked well for the simple oval track in simulation and the real world because the track has a relatively narrow speed range from 5.5 m/s on the turns to 10 m/s on the straight. However, the single speed target value limits performance when testing at the large CCRF track that includes a variety of track features. On the large track, the possible velocity range is much wider, from approximately 3 m/s around a hairpin turn, to at least 13 m/s on the longest straight. Currently, the upper speed limit for the straight is not known. The single speed target set around around 8 m/s for the large track results in MPPI successfully navigating the hairpin, but generates slow (large) lap times by traveling at speeds well below the limits of the system on the straights. Setting the speed target close to the expected maximum achievable speed on the straight causes the speed cost term to overwhelm the other cost terms within MPPI, which results in the vehicle crashing off the track when attempting to navigate turns at speeds that exceed what the algorithm predicts are possible.

An alternative speed cost function could be to use a L1 norm instead of L2, similar to what was used in Tube-MPPI /citewilliams2018robust, which took the form $\omega_2 \left| v_x - v_x^d \right|$, and used a target speed, $v_x^d = 25$ m/s for experiments. This reformulation allows for a constant gradient relative to the desired speed, so that setting a high target speed does not cause the optimization to just focus on achieving a high speed to the detriment of the other task objectives and constraints in the cost function. In the case of an MPC where the dynamics model is very accurate, the optimization is then left to choose the actual speed to drive. In practice this works well, as measured by lap times, but results in brittle performance that requires careful tuning of cost function component weights and results in the MPC being prone to getting stuck in local optima due to a lack of meaningful gradient information. The L1 representation for speed cost would also not be desirable for cost function optimization because, by design, changing the value does not provide meaningfully different information to the MPC. The purpose of an L1 representation serves the opposite purpose of our goal in cost function optimization: determine precisely what speed the vehicle should be traveling at each point on the track. Instead, it provides a weak signal to generally travel at a higher speed, while passing off the determination of the exact speed to travel to other parts of the MPC.

One possibility for cost function optimization is to use a L1 speed cost component would be to optimize the component weight, ω_2, instead of the desired speed, which would be set to a speed higher than what is known to be possible. This would be changing the slope of speed cost, instead of the zero crossing in the case of changing the target speed. While experiments in simulation indicate that it is possible to perform this type of cost function optimization with MPPI, it tends to push the dynamics model into unstable driving regimes that result in degenerate behavior instead of accomplishing the task such as stopping on the track or spinning in tight circles indefinitely. The relationship between the cost function and unstable modes of the neural network dynamics model within MPPI was not investigated here as it is outside of the scope of this work to use MPPI as a MPC for the purposes of cost function optimization.

The cost function representation that will be used keeps the L2 form and replaces the desired speed value with a speed cost function, C_S, that parameterizes the speed cost with the vehicle x and y position in a local coordinate frame like the track cost function and additional set of model parameters θ_S

$$q(x, \theta) = \omega_1 C_T(p_x, p_y) + \omega_2 (v_x - C_S(p_x, p_y, \theta_S))^2 + \omega_3 0.9^t I + \omega_4 \left(\frac{v_y}{v_x} \right)^2 . \tag{3.2}$$

Using this updated cost function, the cost function optimization problem can be posed as finding the parameters $\theta^* = \theta_T^*, \theta_S^*$ that results in the minimum lap time, t_{lap}^*, when used by the MPPI algorithm to drive AutoRally around a track

$$J(\theta^*) = \arg\max_{\theta} \left(\text{MPPI}(\theta_T, \theta_S) \right) . \tag{3.3}$$

A diagram of this optimization is shown in Figure 1.1. Note that there are two different optimization loops in this setup. The first is MPPI optimizing control plans at 50 Hz using the cost function and dynamics model to drive AutoRally. The second, that the next chapter focuses on, is the cost function optimization, that occurs on the time scale of lap times, and can be viewed as a meta-optimization for the MPC. This hierarchical optimization is in contrast to a more traditional reinforcement learning approach that could replace all of MPPI with a neural network and then attempt to directly optimize the parameters of the neural network. This approach would not be tractable on a real system because of the number of interactions (laps) that would be required, and it is not apparent how to create a parameterization of the lap time that we could optimize over.

CHAPTER 4
COST FUNCTION OPTIMIZATION

In this chapter, we present a methodology to automatically tune a cost function used by a MPC through interactions with the system. This work specifically investigates replacing a hand-coded task description, or cost function, that the MPPI MPC uses with one optimized automatically through interactions with a system. Another view of this methodology is as a hierarchical cost function optimization where the inner loop is the MPC optimizing control plans in real time, which is on the order of 50 Hz, to drive AutoRally around an environment. The second level of optimization works on the timescale of laps, which range from around 10 to 30 seconds to complete in the available environments, and optimizes components of the cost function using lap times as the reward signal.

An analogy to expert drivers is that of a racer preparing and then competing in an event. When a professional driver arrives at a track, they bring with themselves years of training and practice and racing knowledge for how best to approach the track. In the model predictive control, cost function optimization (MPC-CFO) framework, this can be thought of as the MPC. Before a race begins, each driver performs a series of practice laps to specialize their skills to the particular track, driving conditions, and vehicle for the race. This specialization by the human driver for the purposes of attaining the highest level of performance on a track is analogous to the cost function optimization portion of the MPC-CFO framework. In essence, the goal is the opposite of generalization: take all prior knowledge of a capable driver and perform a limited number of additional interactions with the system in the deployment environment to tailor performance as much as possible.

In the remainder of this chapter, we explore various ways to represent the speed cost component of the cost function for the task of rally racing, the stochastic optimization methods used to perform the cost function optimization, and experimental results on the

AutoRally platform in simulations and the real world. For all representations presented in this work, the vehicle position p_x, p_y is at the body frame of the AutoRally platform and is in a fixed, local coordinate frame with the east-north-up convention. The velocities, v_x, v_y, used in the cost function are in the body frame.

4.1 Cost Function Representations

A key question we seek to investigate in this work is what type of cost function component representation enables task performance improvement for the task of racing. Specifically, we will be dealing with the speed cost component, C_S, of the MPPI cost function, which can loosely be interpreted as a desired speed for the vehicle to drive. In all previous work the desired speed was a single value determined by a human operator. In practice, this speed cost acted as a loose upper speed bound when operating on a track. The single value target speed will be used as the baseline comparison method when evaluating representation. Two other representations based on radial basis functions RBFs will be explored that allow for increased representational power to capture how the desired speed varies around a track while retaining relatively low model complexity.

All previous versions of MPPI assume static cost function components that are loaded at run time and did not support cost function component modification while the controller runs. A new interface was added to MPPI to enable dynamic cost function loading for this work. The interface allows ROS Image messages to be sent to the controller using the standard ROS publisher-subscriber paradigm and standard image format where each pixel is represented by one byte. This paradigm is consistent with how the AutoRally system is designed and allows for easy integration into the existing software stack. The implementation also allows for a clean separation in execution of the cost function optimization and controller driving the robot as depicted in the MPC-CFO diagram in Figure 1.1. The remainder of this section discusses the pixel-based cost function form required by MPPI in order to run and the RBF-based cost function representations used in the cost function optimization framework.

4.1.1 Image

Because the MPPI controller is implemented in CUDA and runs on a GPU, we must keep in mind the form that the cost function must take for use by the MPC running on the robot. The cost function for MPPI is stored in GPU texture memory, and can be efficiently indexed when it is in a 2-dimensional array that can be thought of as a top-down view of the environment, where the value of each pixel represents the cost function component at that point. This means that in practice, a parameterized model can be used to represent cost function components during cost function optimization, but then for use by MPPI on the GPU these cost components must be sampled out to a 2-dimensional array and passed down to the controller for use. This array also allows for efficient look-ups on the GPU, which is a requirements because the cost function is evaluated over a million times per second by the controller. The cost function is evaluated at each of the 100 time increments for each of the 1920 rollouts MPPI computes per iteration, where runs at 50 Hz. All together, the cost function is evaluated almost 10 million times per second.

All cost function component images for use by MPPI at simulated and real versions of the CCRF track are 1,000 by 900 pixels, at a pixel density of 20 pixels per meter. Each pixel is represented as a one-byte unsigned integer with valid integer values of 0 to 255. A scaling factor parameter is used in the representation to scale the pixel values to meters per second that the representation expects. Using a scaling factor of 10 allows for one decimal point of precision and speeds in the range of 0.0 to 25.5 m/s. This range is sufficient to represent roughly double the range of possible speeds in our testing environments, which have maximum speeds of roughly of 14 m/s. From previous hand tuned experiments with MPPI and AutoRally, one decimal place is sufficient precision for the purpose of setting a target speed. The image size allows MPPI to optimize for tracks that cover a 50 m by 45 m area. In practice, the discretization and overall size of the environments are selected by

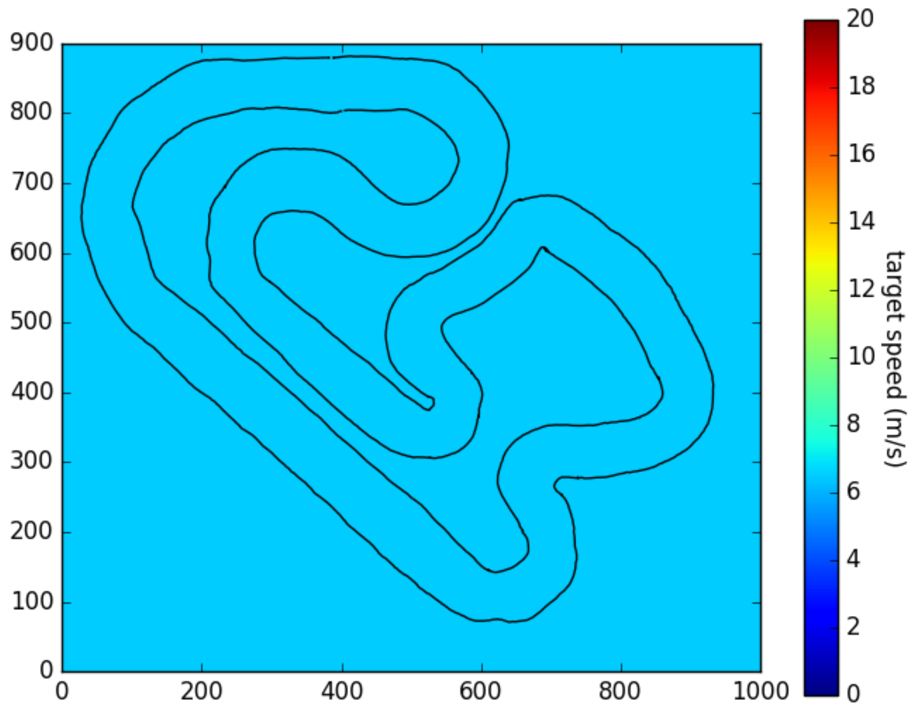


Figure 4.1: Example fixed speed cost component visualized as an image with track boundaries overlaid. The value of each pixel is the cost at that location in the local fixed coordinate frame. Large GT-ARF cost map.

the MPPI designers. While the image representation allows for intuitive understanding of the components by humans, it would not be practical for direct optimization for the task of racing. Working with the image representation for the CCRF track, the optimization would have to operate on 900,000 parameters per cost function component image. That parameter space is hopelessly large give that we are limited to around 100 samples per condition in the real world for our task, where each sample represents a complete lap around a track. These limitations are due to the time it takes to collect data, longevity of the hardware, and environmental degradation (ruts forming) caused by repeated hard accelerations on dirt surfaces. Further, many of the pixels represent regions outside of the driveable surface in an environment, so no direct interactions with large portions of an image cost function representation are possible.

An example speed cost image for the GT-ARF large track track for a constant target speed of 6.5 m/s is shown in Figure 4.1 with the track boundaries overlaid. The track boundaries are automatically generated from GPS survey data collected at the real track. Cost function component images are interchangeable between all environments because the simulation world share the same local coordinate frame origin as the real world.

4.1.2 Radial Basis Function

An alternative representation to the single speed target and image representations would be one with more representational power offered by the single speed target value, but require orders of magnitude fewer parameters than the number of pixels in the image representation. Gaussian radial basis functions (RBFs) are a representation that we investigate here to model the speed cost component. A single Gaussian RBF has the form:

$$K(p, h, \mu, \sigma) = he^{-\frac{\|\mu - p\|^2}{2\sigma_n}}, \tag{4.1}$$

where p is the evaluation point, h is the weight, μ is the mean, and σ is the standard deviation, or size, of the RBF. N RBFs can be combined with a background value similar to the single speed value, to create a cost function speed model. The resulting model has the form

$$C_S(p) = B(p) + \sum_{n=1}^{N} h_n e^{-\frac{\|\mu_n - p\|^2}{2\sigma_n}}, \tag{4.2}$$

where $p = (p_x, p_y)$ is the feature vector representing the evaluation point, h is the weight of the RBF, and $\|\mu_n - p\|^2$ is the squared Euclidean distance between the current RBF and the sample point. The term $B(p)$ is a background term to provide support for the model. If $B(p)$ is not used, areas within the parameter space that do not have any support from a RBF evaluate to 0, which for the speed cost will cause the robot to stop indefinitely during optimization. For initialization, the background is initialized to a constant value. During optimization, the RBFs then represent the deviation from the background portion of the model. An example RBF grid speed cost function for the GT-ARF large track track is shown in Figure 4.2. The coordinate system used for the location of the RBF in this model is the local coordinate frame of the track, which can be transformed into pixel location with a simple change of basis using the pixels per meter and origin location infromation for a track. The model size is $1+4N$, where N is the number of RBFs. The number of RBFs can be be chosen either automatically through a search, or determined by the operator based roughly on the number of driving features such as turns and straights that make up the environment. No prior knowledge of track geometry is explicitly encoded in the model, so the RBF locations are distributed uniformly around the space for initialization. In practice, the number of RBFs chosen are 16, 25, or 36, which creates models with 65, 101, and 145 parameters, respectively.

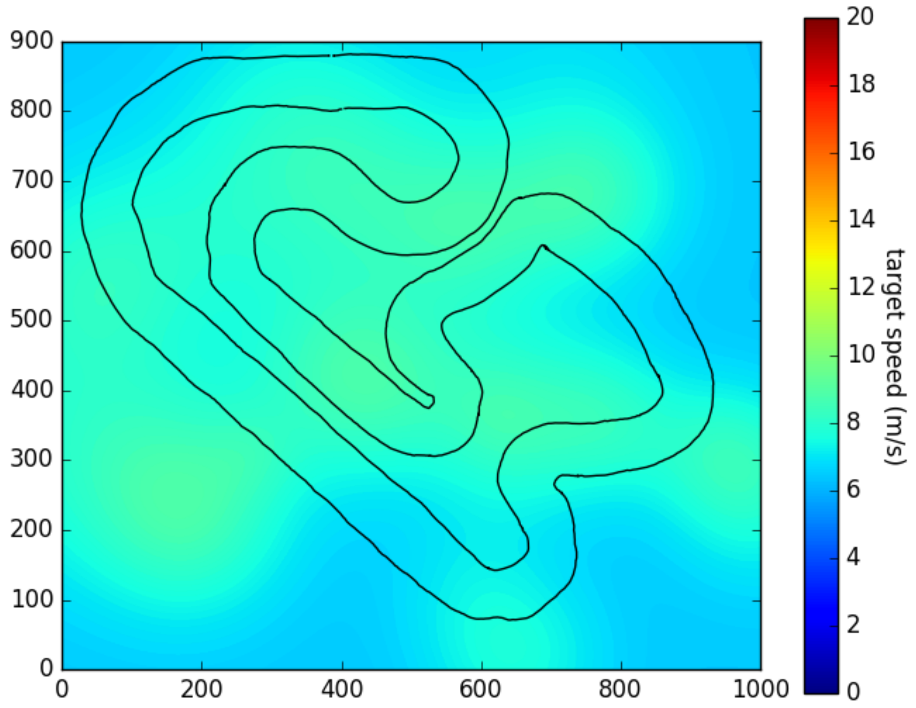


Figure 4.2: Grid radial basis function speed map represented as images for the large CCRF track. The value of each pixel is the cost at that location in the local fixed coordinate frame.

While the coordinate system of the grid RBF representation allows for efficient integration with the MPPI algorithm, and is intuitive for the cost function designer, there are two drawbacks that we encountered during experiments. First, many of the parameters represent regions of the space that are not driveable area, so perturbations to those parameters during optimization do little to effect task performance. One possibility would be to clip the representation to only the track surface using the known track boundaries, but that would require us to include prior knowledge of track geometry into the representation, which we specifically want to avoid, since this would be duplicating the track boundary information already present in the track cost component. Second, the grid representation allows track sections that are close to each other in space may be far away from each other when driving around a track, but their close proximity in the representation's coordinate space causes them to interfere with each other during the optimization, which leads to premature convergence and incorrect optimization results. An example is a hairpin turn that butts up along a very long straight. The target speed along the straight will be very high relative to the tight turn, but because they are close together in coordinate system used, the tight turn can cause a portion of the long straight to have a slow section.

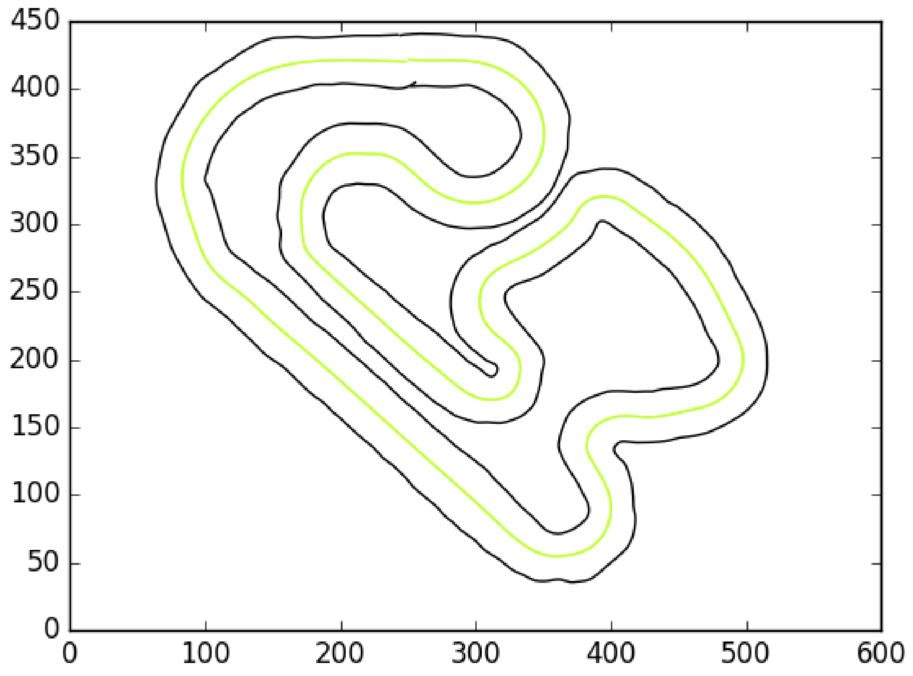


Figure 4.3: Track centerline spline, shown in green, along with inner and outer track boundaries. The track boundaries are computed from a GPS survey collected at the CCRF large track. The centerline spline is computed as the middle of the two boundaries.

To address these shortcomings of the grid RBF representation, we propose to use a new coordinate system, based on the centerline of the track, for the speed cost representation.

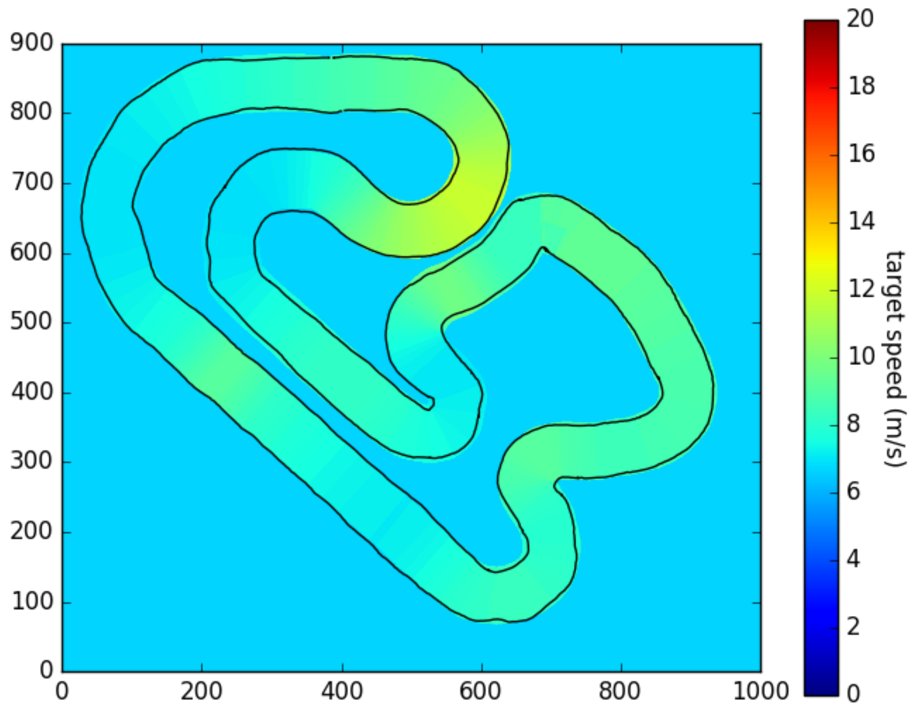


Figure 4.4: Track centerline radial basis function speed map represented as images for the large CCRF track. The value of each pixel is the cost at that location in the local fixed coordinate frame.

This centerline RBF representation reduces the model dimensionality by one per RBF by only having one position parameter, which represents the distance along the spline that the RBF is located. This centerline distance ranges from zero to one, where zero represents the starting line of the track. As the position value increases, the RBF location moves along the centerline of the track in a counter clockwise direction. A centerline representation allows us to implicitly take into consideration the geometry of the track through the spline fitting process, without having to enforce any track geometry in the representation. In other words, our representation maintains generality by rolling the specific track geometry into the change of coordinates that takes place when creating the speed cost image for use by MPPI from the speed cost model.

The centerline spline can be fit from the surveyed track boundaries by sampling the boundary splines at regular intervals, interpolating the position of the middle point, and then spline the computed middle points. The centerline spline representation also acts as the coordinate transformation method to move between the spline coordinate system that is single dimensional from 0 to 1, and the image coordinate system. A spline of degree 10 has been found empirically to work well for all of the tracks that have been evaluated to date, and an example is shown in Figure 4.3. The centerline RBF representation eliminates the possibility for unrelated track features that are close in our final coordinate space to interfere with each other over the course of cost function optimization. The centerline model size is $1 + 3N$, where the first parameter is the background speed and N is the number of RBFs. In practice, the number of RBFs chosen should be a function of the number of distinct driving features on a track, but values in the range of 12 to 25 have been found to work well. Even though no samples are collected where the vehicle drive on non-track regions, the rollouts that MPPI computes often results in the vehicle departing the track. While these rollouts incur a crash penalty, it was found that a non-zero target speed in non-track regions improves performance, especially in difficult sections of a track where many of the rollouts computed by MPPI end up off of the track.

In summary, there are three speed cost representations that are explored: the single speed target from prior work will act as a baseline, the grid RBF method, and the track centerline RBF representation.

4.2 Cost Function Optimization Methods

This section describes the cost function optimization methods investigated as part of the MPC-CFO framework. Building on top of the possible representations for cost function components, the goal is to use optimization methods that do not restrict the form of cost function representations, can optimize models that have 10s to low 100s of parameters in real time, and can function in the face large amounts of system noise and with relatively few samples. For the purpose of the task of racing, real time means that all required computations must be completed in less than one lap, which can range from around 10 to 30 seconds. This is a practical limitation for data collection so that the robot can run continuously without having to drive a lap and then wait some time for computations to complete before continuing again. Gradient-based methods are not suitable for cost function optimization for all of those reasons, the most prominent is the inability to deal with relatively high-dimensional models and handle significant system noise. Therefore, we will explore using two different sampling-based optimization methods in the MPC-CFO framework. The general procedure for sampling-based optimization methods can be summarized in two steps given a parameterized model:

1. generate samples from a distribution and compute their costs,

2. update the distribution using a ranking scheme based on the sample costs,

which is repeated to convergence.

For the task of racing where the goal is to achieve the minimum lap time, generating samples means providing a sampled component to MPPI as it drives around a track generating lap times. One sample is completed when a lap time is generated from MPPI driving for that entire lap. The ranking portion of the optimization is based purely on the lap times generated.

Cross entropy (CE) is a stochastic optimization method that has been shown to work well for motion planning and optimization problems in robotics[35, 36], and in aggressive off road driving [41]. It is related to simulated annealing and genetic algorithms and suitable for non-linear and high-dimensional systems. The update step for CE involves computing an average of elite sample parameters. Elite samples are the samples that generated the lowest costs. In practice it has been found that a good value for the eliteness threshold, which ranges from 0 to 1, is $K_e = 0.2$, which means that the top 20% of samples are used to compute the output from each optimization iteration. Another stochastic optimization optimization method related to CE is pi squared (PI^2), which maintains all of the benefits of stochastic optimization, but uses a different update rule for the samples. Instead of an average of the elite samples, PI^2 uses a weighting scheme that is proportional to the normalized relative performance difference among all of the samples in each optimization iteration. Instead of the eliteness threshold required by CE, the one parameter required to be set for PI^2 is a different eliteness parameter h , which controls how quickly the weight for a sample drops off as a sample cost is further from the best sample of the epoch. In practice, $h = 10$ was found to work well been found to work well in [36], and is used in this work.

As a vehicle approaches the limits of performance, even small changes in strategy can results in task failure or significant departures from a desired racing line such as an unintended skid or over correcting through a turn. However, when first starting an optimization it is not known exactly how far the solution is from the initial conditions, so the sampling variances must be high enough to make significant progress if the number of samples is limited. Because of these two competing goals: fast progress early in the optimization, then slow refinement approaching a solution, the covariance matrix adaptation (CMA) variant of each optimization algorithm is employed. CMA provides an update scheme for the parameter covariances that is a function of the magnitude of progress made each iteration which can address the limitations of choosing fixed parameters that results in unstable optimization or no meaningful progress. In practice, the convergence rates are still related to the magnitude of the initial parameters, but not to the same extent as without CMA.

For the application of cost function optimization to task of autonomous rally racing, candidate cost maps are generated by sampling model parameters, then MPPI is tasked with driving around a test track for a specified number of laps, normally set to $N = 1$, with each sample. After laps for each sample are complete, the samples are ranked by the lap times they produced, and then the model parameters are updated according to the optimization method. Algorithmically, the process is as follows:

Note that the for PI^2-CMA, temporal averaging is not included as it is not required when samples have only one time step (each cost of each sample is a single lap time as opposed to an evaluated trajectory in the case of applications in control and motion plan-

Given: MPC method: MPPI;
C_S: cost function;
$\hat{\theta}_{S,0}, \hat{\Sigma}_{S,0}$: initial cost function model parameters;
K: number of samples;
N: laps per sample;
δ: cross entropy eliteness threshold (< 1.0);
$K_e \leftarrow \mathrm{round}(K\delta)$: elite samples;
σ: parameter variance decay rate;

```
1  while $tasknotcomplete$ do
2      $\alpha \leftarrow epochnumber$;
3      sample $\theta_{S,k=1\ldots K}$ from $\mathcal{N}(\hat{\theta}_S, \hat{\Sigma}_S)$
4      for $k \leftarrow 1$ to $K$ do
5          for $n \leftarrow 1$ to $N$ do
6              $t{+} = \mathrm{MPPI}\left(C_S(\theta_{S,k})\right)$
           end
7          $\bar{t}_{lap,k} = \frac{1}{N} t$
       end
8      $\theta_{S,k=1\ldots K} \leftarrow \mathrm{sort}(\theta_{S,k=1\ldots K}, \mathrm{key} = \bar{t}_{lap,1\ldots K})$
9      $\hat{\theta}_S = \sum_{k=1}^{K_e} \frac{1}{K_e} \theta_{S,k}$
10     $\hat{\Sigma}_S = \sum_{k=1}^{K_e} \frac{1}{K_e} (\theta_{S,k} - \hat{\theta}_S)(\theta_{S,k} - \hat{\theta}_S)^T$
11     $\hat{\Sigma}_S = \min\{\hat{\Sigma}_S, \hat{\Sigma}_{S,0} e^{-\sigma\alpha}\}$
   end
12 return $\{\hat{\theta}_S, \hat{\Sigma}_S\}$
```

Algorithm 1: Model Predictive Control - Cost Function Optimization with the Cross Entropy method.

Given: MPC method: MPPI;
C_S: cost function;
$\hat{\theta}_{S,0}, \hat{\Sigma}_{S,0}$: initial cost function model parameters;
K: number of samples;
N: laps per sample;
h: PI squared eliteness parameter;
σ: parameter variance decay rate;

```
1  while $tasknotcomplete$ do
2      $\alpha \leftarrow epochnumber$;
3      sample $\theta_{S,k=1...K}$ from $\mathcal{N}(\hat{\theta}_S, \hat{\Sigma}_S)$
4      for $k \leftarrow 1$ to $K$ do
5          for $n \leftarrow 1$ to $N$ do
6              $t+ = \text{MPPI}\,(C_S(\theta_{S,k}))$
           end
7          $\bar{t}_{lap,k} = \frac{1}{N}t$
       end
8      for $k \leftarrow 1$ to $K$ do
9          $-\frac{1}{\lambda}S_k = \frac{-h(\bar{t}_{lap,k} - \min(\bar{t}_{lap,k}))}{\max(\bar{t}_{lap,k}) - \min(\bar{t}_{lapk})}$
10         $P_k = \frac{e^{-\frac{1}{\lambda}S_k}}{\sum_{k=1}^{K}[e^{-\frac{1}{\lambda}S_k}]}$
       end
       $\hat{\theta}_S = \sum_{k=1}^{K} P_k \theta_{S,k}$
11     $\hat{\Sigma}_S = \sum_{k=1}^{K} P_k(\theta_{S,k} - \hat{\theta}_S)(\theta_{S,k} - \hat{\theta}_S)^T$
12     $\hat{\Sigma}_S = \min\{\hat{\Sigma}_S, \hat{\Sigma}_{S,0}e^{-\sigma\alpha}\}$
   end
13 return $\{\hat{\theta}_S, \hat{\Sigma}_S\}$
```

Algorithm 2: Model Predictive Control - Cost Function Optimization with the PI Squared optimization method. Note that optimal baselining is used for $-\frac{1}{\lambda}S_k$.

ning). A common criticism of CE and PI^2 is that of premature convergence. We experience this issue, especially in the real world experimental results that have many environmental conditions out of our control and significant system noise, but we address it by adding an exponentially decaying lower bound in the CMA update step. This bound is initialized to the initial sampling variances, and decays as a function of the optimization epoch, α at a rate controlled by σ. For all of the experiments that use MPC-CFO, a value of 0.4 for σ was used, which is based on the observation that performance is often converges between epochs five to ten of the cost function optimization when using $K = 10$ samples per epoch with all of the representations.

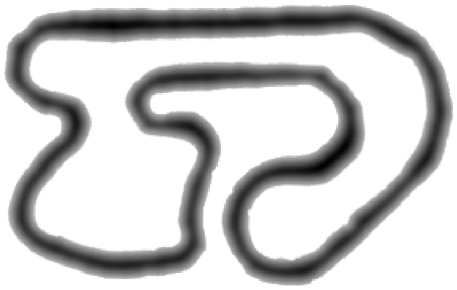
4.3 Experimental Results

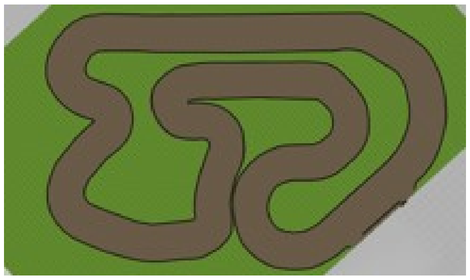
For experimental results, we work through multiple environments of increasing fidelity with all representations and optimization methods from a simple simulation world, to a physics-based simulation world with a simulated robot and visual rendering, and finally complete tests at a real track running hardware. As part of the real world experiments, we also discuss the level of performance obtained versus prior results at the track with the MPPI controller.

4.3.1 Testing Environments

To study MPC-CFO with MPPI and our chosen cost function representations and cost function optimization methods, three testing environments shown in Figure 4.5 were selected, all of them with an identical track layout.

The first is ground truth simulation, which is simple simulation environment where the dynamics model used by MPPI is used as physics model of the world. The key feature of this environment is that the controller has perfect knowledge of the dynamics. In this environment the vehicle will never crash, though it can occasionally encounter unstable modes in the dynamics model and gets stuck spinning in tight circles. That behavior has


(a)


(b)

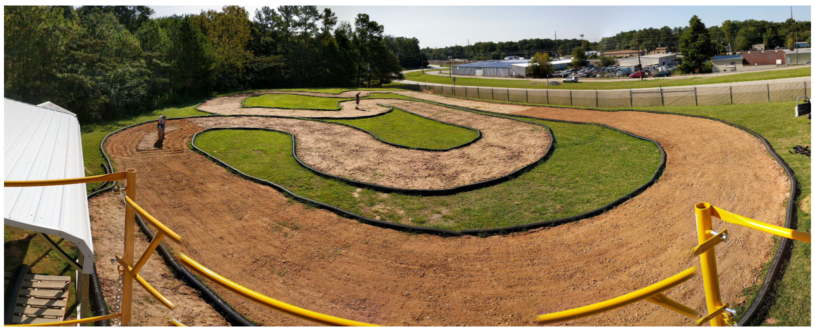
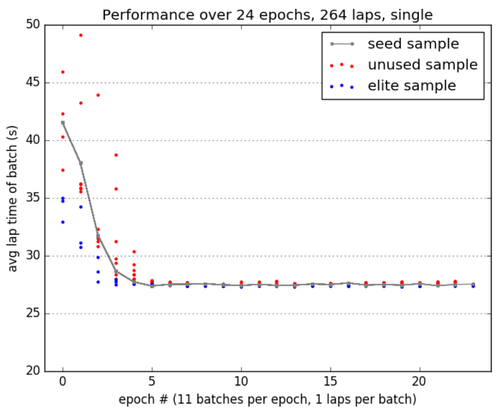

(c)

Figure 4.5: Testing environments for MPC-CFO. All tracks have identical layouts and coorindate frames. (a) Ground truth simulation where the MPC has perfect knowedge of dynamics. (b) Gazebo-based simulation with simulated AutoRally platform, simulated sensors, and an identical software interface as the physical robot. (c) Georgia Tech Autonomous Racing Facility large track located at Georgia Tech's Cobb County Research Facility.

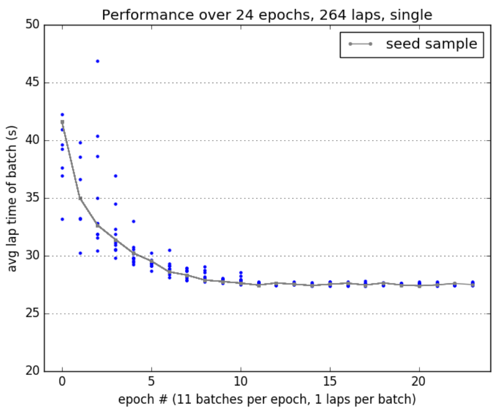
been occasionally observed if trying to drive faster then what is possible in the real world, and was not encountered during any experiments presented here.

The second testing environment was built using Gazebo, and includes a simulated AutoRally platform with sensors and actuators and is part of the publicly available AutoRally software. A significant limitation of the Gazebo simulation is that the physics engine in Gazebo becomes numerically unstable when the simulated AutoRally vehicle approaches the friction limits of the system. This is due to the cone contact model employed by the physics engine that is designed primarily for rigid body dynamics with manipulators, slow moving ground vehicles. The result of these instabilities is that large spikes in the simulated sensors are seen, which also propagates through the objects in the simulation environment causing sudden changes in direction. Another limit of the cone friction model for wheeled ground vehicles moving quickly in Gazebo is that instead of smoothly transitioning between static and sliding friction regimes as would happen when tires lose grip in the real world, the vehicle oscillates between small slides and no sliding, which causes significant shaking in the vehicle that typically ends in the vehicle rolling over at speeds well below what is capable on the same turn at the real track. Both simulated tracks were developed from the track boundary survey conducted at the real GT-ARF large track and share identical coordinate frames.

The third testing environment is the real world GT-ARF large track with AutoRally vehicles. A feature of the real world that will be discussed in some detail at the end of the experimental results is that, no matter how much one attempts to control external environmental factors in an experiment, the use of hardware on a dirt track for extended experiments results in changing driving conditions hour to hour and day to day. Many of these variables could be more easily controlled if the chosen task was on road driving with an automotive grade platform, but that is beyond the scope of this thesis.

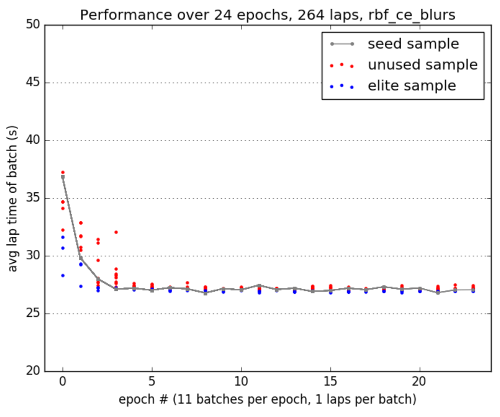


(a)

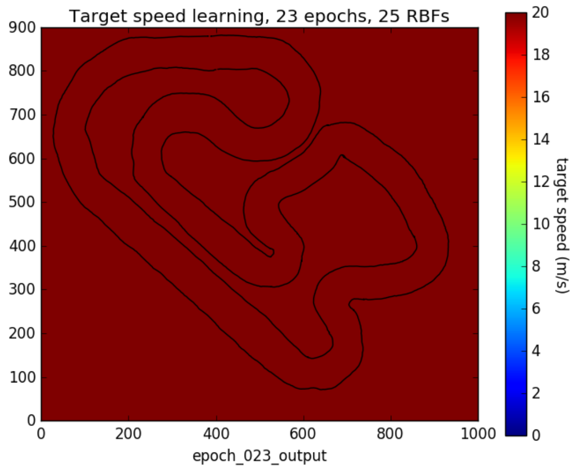


(b)

Figure 4.6: Single speed target cost function optimization in the ground truth simulation environment. Lap times are shown over 24 epochs of optimization, 264 total laps around the track. (a) Performance with the cross entropy optimization method, optimized target speed is 16.6 m/s. (b) Performance with the PI squared optimization method, optimized target speed is 16.4 m/s.



(a)

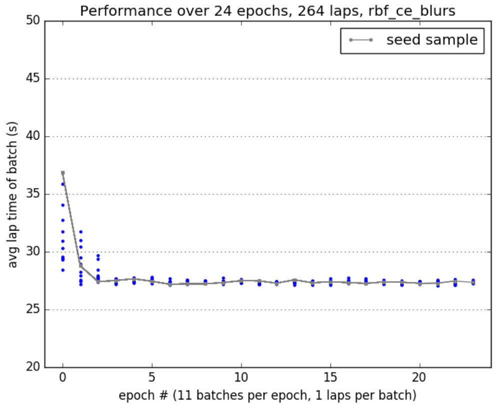


(b)

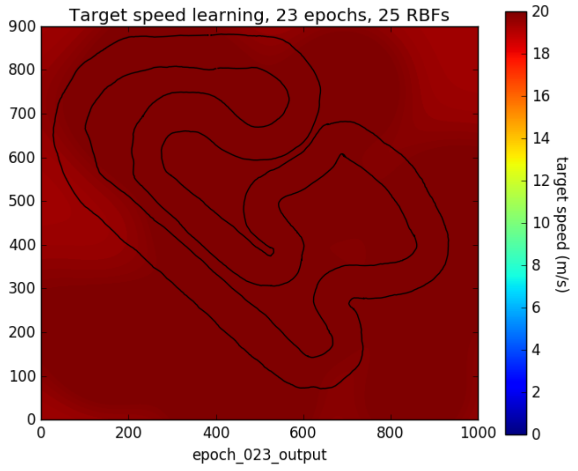
Figure 4.7: Grid RBF cost function representation with CE optimization in the ground truth simulation environment. (a) Lap time performance over 24 epochs of optimization, 264 total laps around the track.(b) Visualization of the speed cost at the end of optimization, direction of motion is counter clockwise.

Table 4.1: Minimum lap times achieves across all cost function optimization conditions in ground truth simulation.

		Cost Function Representation		
		Single Speed Target	Grid RBF	Centerline RBF
Opt. Method	Cross Entropy	27.32	**26.78**	26.97
	PI Squared	27.35	27.06	26.93

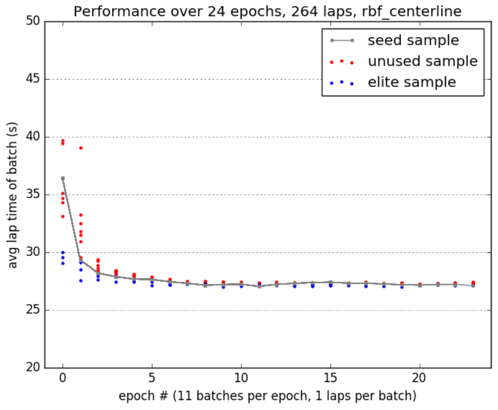


(a)

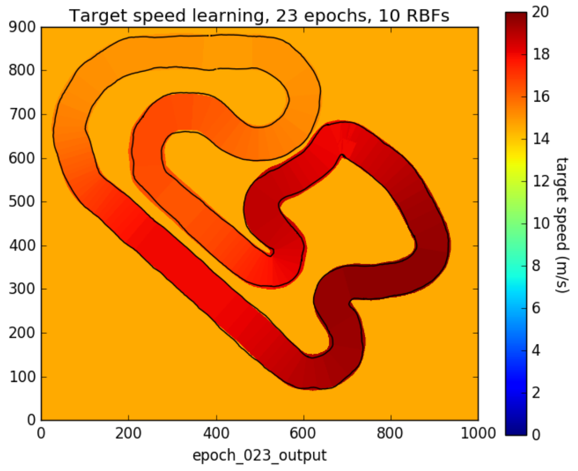


(b)

Figure 4.8: Grid RBF cost function representation with PI^2 optimization in the ground truth simulation environment. (a) Lap time performance over 24 epochs of optimization, 264 total laps around the track.(b) Visualization of the speed cost at the end of optimization, direction of motion is counter clockwise.

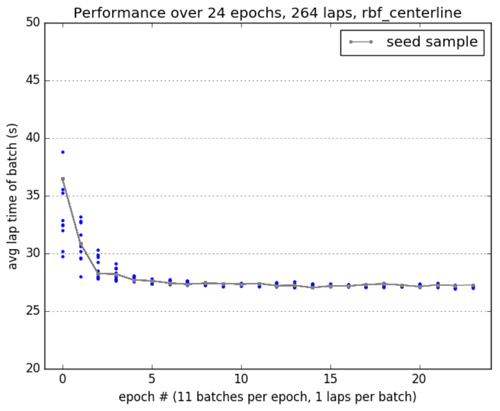


(a)

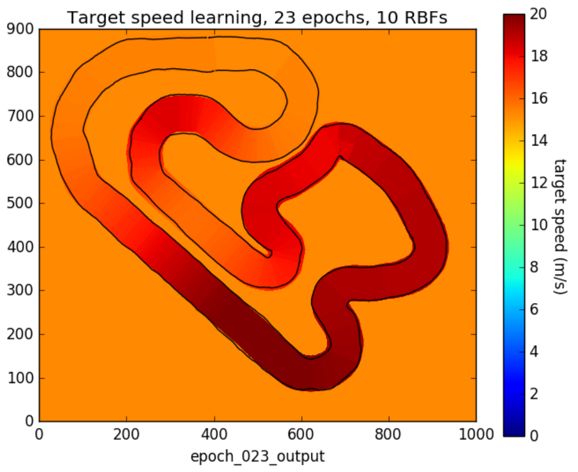


(b)

Figure 4.9: Centerline RBF cost function representation with CE optimization in the ground truth simulation environment. (a) Lap time performance over 24 epochs of optimization, 264 total laps around the track. (b) Visualization of the speed cost at the end of optimization, direction of motion is counter clockwise.



(a)



(b)

Figure 4.10: Centerline RBF cost function representation with PI^2 optimization in the ground truth simulation environment. (a) Speed map optimization with image model and RBF noise at GR-ARF large track. (b) Visualization of the speed cost at the end of optimization, direction of motion is counter clockwise.

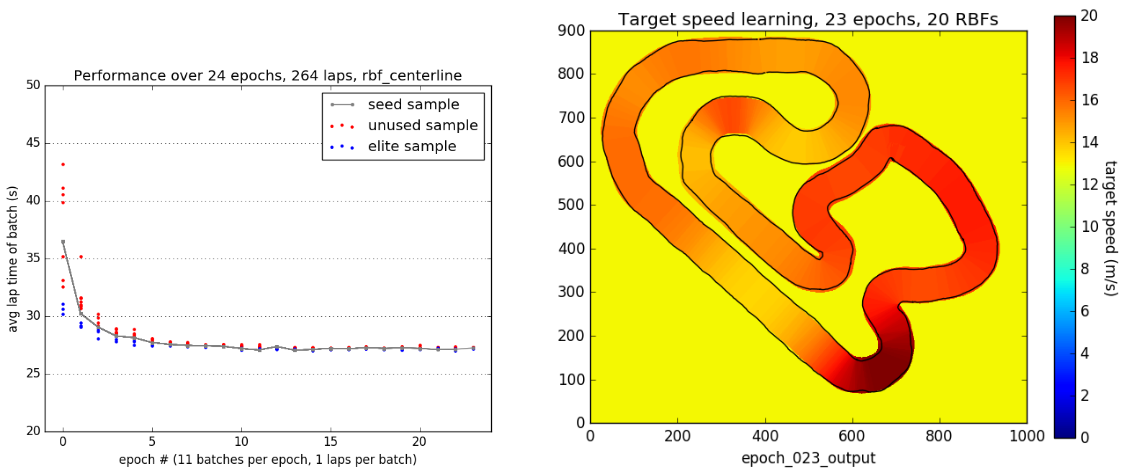
4.3.2 Ground Truth Simulation

The first set of cost function optimization experiments was conducting using the ground truth simulation where the the MPC has perfect knowledge of the world dynamics. For every combination of speed cost representation and optimization method, MPC-CFO was run for 24 epochs which required 264 laps. Performance converged after 5 or 10 epochs of optimization for all experiments, which took approximately 2 hours to run on a computer with equivalent computing to an AutoRally Compute box. The simulations cannot be run faster than real time because MPPI is configured to run in real time on the GPU available for computational limits. The dynamics model used for all ground truth simulation scenarios is one trained from a system identification dataset collected on an AutoRally platform and publicly available on the AutoRally Github repository. The model is contained in the file autorally_nnet_09_12_2018.npz. This is the same dynamics model used by MPPI for experiments collected at GT-ARF.

Overall, the performance difference, as measure by fastest lap time generated, across all ground truth simulation experiments, is about 0.57 s with the best minimum lap time was 26.78 s generated by grid RBF with CE and the worst minimum lap time was 27.35 s generated by single speed target representation and PI^2 optimization. The minimum lap time results are presented in Table 4.1.

Results for optimizing the single speed target are shown in Figure 4.6, where the speed targets that the produced the best performance are 16.6 m/s with CE and 16.4 m/s with PI^2. The seed value for the optimization was a target speed of 6.5 m/s and a sampling variance of 1.0. These optimized speed targets are much higher than what we know to be possible in the real world, but with perfect knowledge of the world dynamics, the controller is able to push the target speed beyond what would cause a crash in the real world. In simulations, the vehicle never achieves a speed faster than 12.4 m/s. In essence, a speed target above what is possible on a track is a signal to the MPC to always try to drive faster, regardless of the current speed or position on the track. Implicit in these results is that for the ground truth simulation, regardless of what speed MPPI is instructed to drive at, the vehicle will not crash. Further discussion on this topic is presented in 4.3.5.

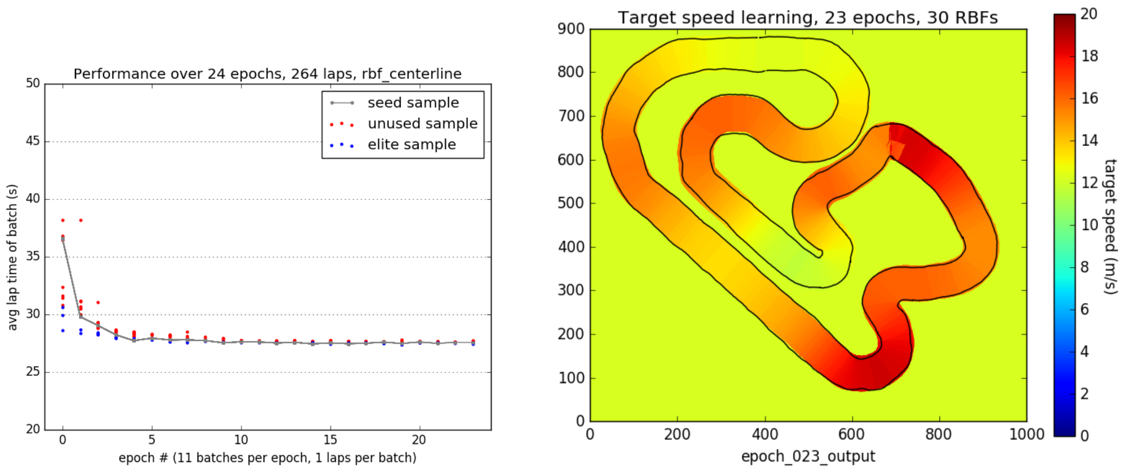
The performance for the grid RBF representations appear to converge at approximately the same rate as for single speed target. Results with CE are shown in Figure 4.14 and with PI^2 in Figure 4.15. The grid RBF model used in both experiments has 25 RBFs for a total of 101 parameters, and is initialized with a background speed of 7.0 m/s, 2.5 m/s variance. The RBFs are spread uniformly around the space in a 5 by 5 grid, each with an initial height of 1.0 m/s, height variance of 0.5 m/s, size of 90 pixels with size variance of 15 pixels, and x and y position variances of 100 pixels. At the end of both CE and PI^2 optimizations the speed maps are almost uniformly 20 m/s, which is higher than either of the optimized single target speeds, although task performance is almost identical. This suggests that, as was the case with the single speed results, in the case where the MPC method has perfect knowledge of the world dynamics, the cost function is optimized to constantly ask the controller to drive faster, regardless of the track geometry and actual speed driven by the vehicle.



(a) (b)

Figure 4.11: Model size effect on convergence, RBF centerline representation with 20 RBFs, model size of 61 parameters, CE optimization. (a) Lap time performance over 24 epochs of optimization, 264 total laps around the track. (b) Visualization of the speed cost at the end of optimization, direction of motion is counter clockwise.

Optimization experiments with the centerline RBF representation produce more visually interesting speed maps, shown in Figures 4.16 and 4.17, although the task performance does not vary significantly from the other representations. The initial background speed and background speed variance was the same for the grid RBF representation, but instead of 25 RBFs, only 10 were used that were evenly distributed along the centerline of the track with initial heights of 1.0 m/s, height variance of 0.5 m/s, size of 0.06, which is percentage of the total track length that ranges from zero to one, with variance of 0.015. The number of RBFs was selected in part because the centerline track representation confines RBFs to the driveable surfaces of the environment which are less than half of the total area in the environment. The minimum speed targets range from about 14 m/s to 20 m/s, and show similar speed patterns on track features. In particular, the target speed is high at the end of the long straight through the next three turns, and is at the lowest value just before the main straight. Similar to the other optimized speed maps, the target speed at every point on the track is higher than the realized speed by a significant amount. The total model size is 31 parameters.

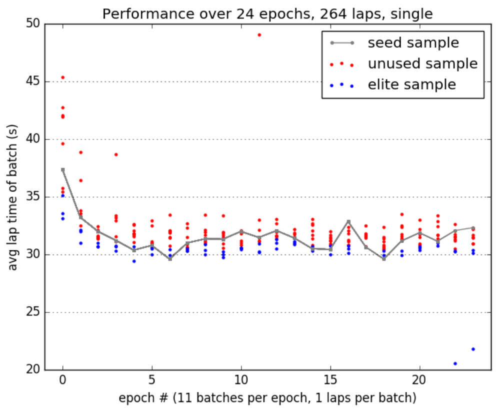


(a) (b)

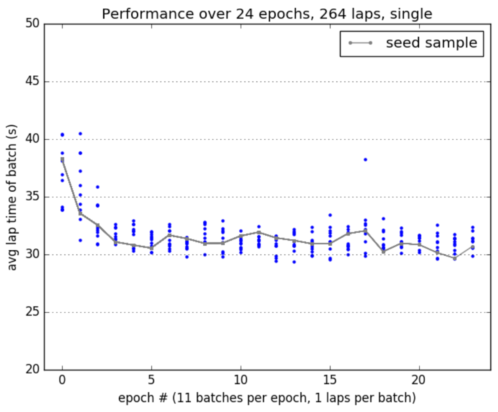
Figure 4.12: Model size effect on convergence, RBF centerline representation with 20 RBFs, model size of 91 parameters, CE optimization. (a) Speed map optimization with image model and RBF noise at GR-ARF large track. (b) Visualization of the speed cost at the end of optimization, direction of motion is counter clockwise.

The rate of convergence and level of performance attained for the cost function optimization in the ground truth simulation does not appear to be effected by the number of parameters in the model as the grid RBF model has 101 parameters, the centerline RBF model has 31 parameters, and the single speed value has one parameter. Figures 4.11 and 4.12 show two experiments with the RBF centerline representation and CE optimization where the number of RBFs was set to 20 and 30, respectively. The fast lap time during optimization with the 20 RBF model was 27.02 s, and for the 30 RBF model the fast lap was 27.37. Both experiments converge at indistinguishable rates compared to the other experiments in the debug simulation, including the single speed target that has a model size of one, and result in similar lap times. Instead, the choice of initial sampling variance has more of an effect on the convergence rate. If the set too low, performance plateaus somewhere above 27 s lap.

Given that the MPC has perfect knowledge of the world dynamics for these experiments, it appears that the optimization is using the speed costs as a general rule to push the MPC to drive faster as opposed to determining the exact speed to drive at each point.



(a)



(b)

Figure 4.13: Single speed target cost function optimization in the Gazebo simulation environment. Lap times are shown over 24 epochs of optimization, 264 total laps around the track. (a) Performance with the cross entropy optimization method, optimized target speed is 9.2 m/s. (b) Performance with the PI squared optimization method, optimized target speed is 8.8 m/s.

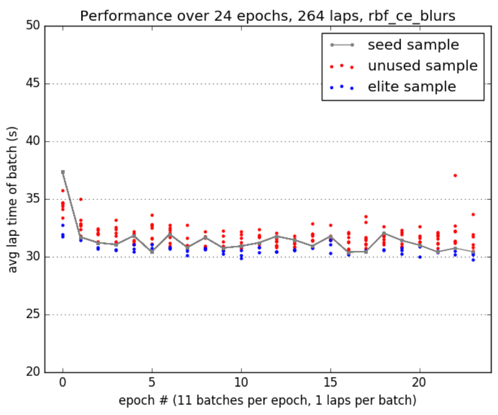
Table 4.2: Minimum lap times achieves across all cost function optimization conditions in the Gazebo simulation.

		Cost Function Representation		
		Single Speed Target	Grid RBF	Centerline RBF
Opt. Method	Cross Entropy	29.45	29.73	29.84
	PI Squared	**29.34**	29.86	30.11

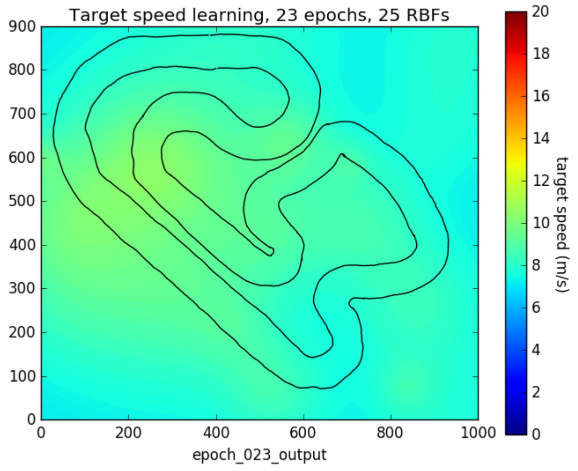
Another way to think of this is that referring to the parameters we are optimizing as target speeds is useful for intuitive understanding, but not completely correct. A second order behavior of this parameter optimization is a tuning of the sensitivity of the optimization to the speed target, with respect to the other components in the cost function. In the passed, all cost function component weights were tuned by experts, and are likely specified incorrectly for the task we are considering. Further analysis on this topic is presented later in the chapter.

4.3.3 Gazebo Simulation

The Gazebo test environment offers a different type of optimization challenge for the MPC-CFO framework than the ground truth simulation where the MPC has perfect knowledge of the world dynamics. In the Gazebo simulation environment, the dynamics model of the

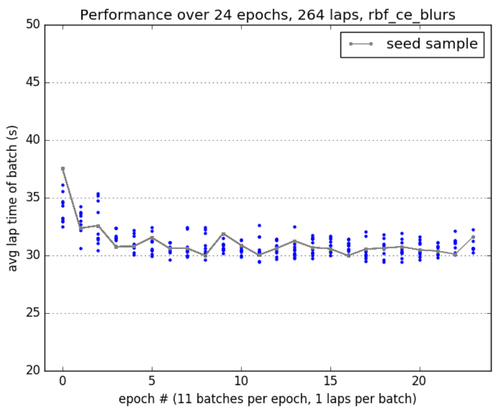


(a)

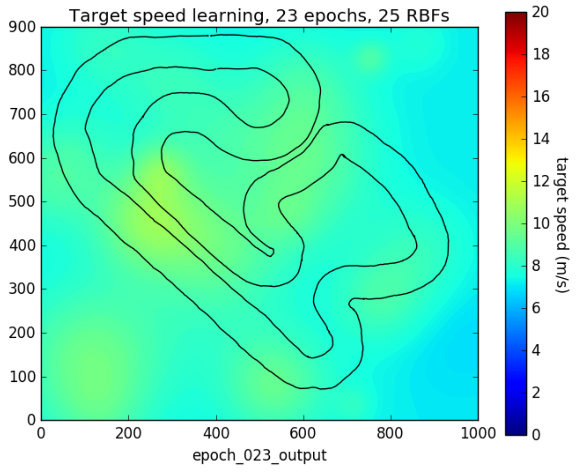


(b)

Figure 4.14: Grid RBF cost function representaiton with CE optimization in the Gazebo simulation environment. (a) Lap time performance over 24 epochs of optimization, 264 total laps around the track.(b) Visualization of the speed cost at the end of optimization, direction of motion is counter clockwise.

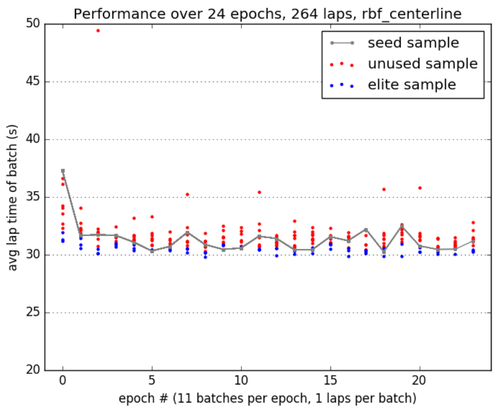


(a)

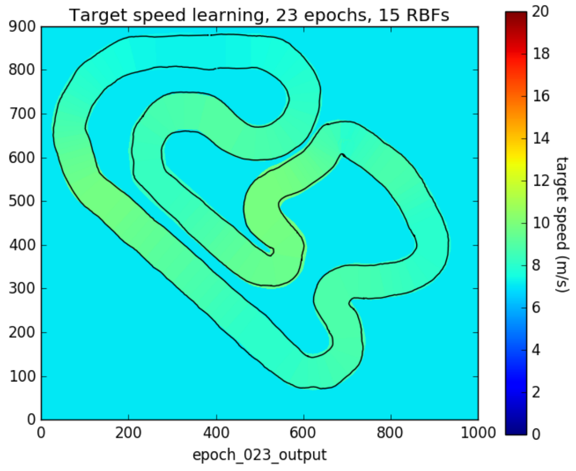


(b)

Figure 4.15: Grid RBF cost function representaiton with PI^2 optimization in the Gazebo simulation environment. (a) Lap time performance over 24 epochs of optimization, 264 total laps around the track.(b) Visualization of the speed cost at the end of optimization, direction of motion is counter clockwise.

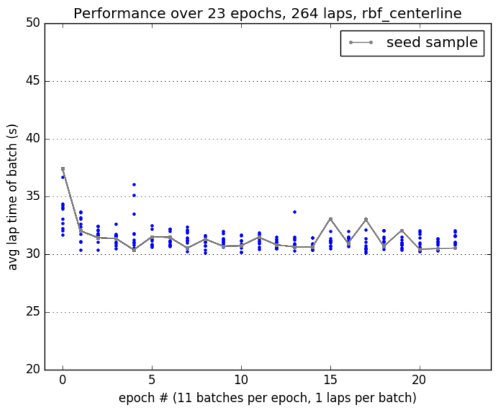


(a)

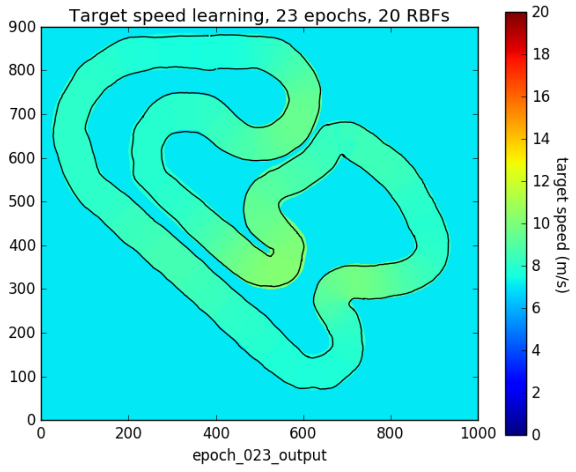


(b)

Figure 4.16: Centerline RBF cost function representaiton with CE optimization in the Gazebo simulation environment. (a) Lap time performance over 24 epochs of optimization, 264 total laps around the track. (b) Visualization of the speed cost at the end of optimization, direction of motion is counter clockwise.



(a)



(b)

Figure 4.17: Centerline RBF cost function representaiton with PI^2 optimization in the ground truth simulation environment. (a) Speed map optimization with image model and RBF noise at GR-ARF large track. (b) Visualization of the speed cost at the end of optimization, direction of motion is counter clockwise.

MPC differs greatly from that of the environment but because of numerical instabilities in the simulation environment and use of a dynamics model trained in a different environment. The MPC is given the same dynamics model trained from data collected on a real AutoRally platform that is used in the other experiments. A good dynamics model tailored to Gazebo was able to be trained because the system identification dataset was unreliable because human drivers cannot drive the simulated robot in a wide variety of driving conditions. Even if a high quality system identification dataset were to be captured in Gazebo, the model would likely not be able to capture the highly oscillatory behavior when the vehicle drive close to the friction limits that is the hallmark of the Gazebo simulation. Because of these differences, the optimized speed functions are qualitatively very different than those produced in the ground truth simulation environment.

Overall, the performance difference, as measure by fastest lap time generated, across all Gazebo simulation experiments, is about 0.77 s with the best minimum lap time was 29.34 s generated by single speed target with PI^2 and the worst minimum lap time was 30.11 s generated by centerline RBF representation and PI^2 optimization. The minimum lap time results are presented in Table 4.1. The fact that the smoothest speed function resulted in the best task performance in the Gazebo simulation is reasonable because it eliminates one potential source of variability when the MPC is already tasked with handling significant noise from the simulation. This behavior is shown in the difference of lap times of about 3 seconds within later optimization epochs when the speed functions change very little compared to lap time differences well below 1 second at the later epochs of optimization in the ground truth simulation.

Across the grid RBF and centerline RBF representations, the optimized speed maps look qualitatively similar, but much different than those from the ground truth simulation. For the grid RBFs, the same 25 RBFs arranged in a 5 by 5 grid were used, and with the centerline RBF representation the number of RBF was 15 for CE and 20 for PI^2. Despite the varying number of RBFs, there was no significant difference in convergence rate nor level of performance during cost function optimization, supporting the conclusion from the ground truth experiments that the dominant factor to determine rate of convergence is the initial sampling variances. The target speeds in the optimized speed functions for Gazebo range in value from around 8 m/s to about 12 m/s. These speeds are consistently much closer to the speed that the simulated platform drives than results from the ground truth simulation. In Gazebo, the dynamics model that the MPC is using is significantly different from the actual physics of the environment. Because of this, over the course of cost function optimization, the MPC is not free to just drive as fast as it wants, but must rely on the target speed component of the cost function to make up for a lack predictive capability of the dynamics model. We see minimal second order optimization effects of tuning the sensitivity of the cost function to the speed component as the MPC relies on the function in order to decide what speed to drive without crashing.

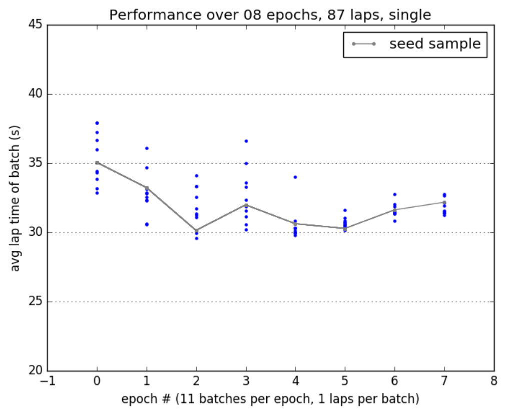
4.3.4 Real World Experiments with AutoRally

Cost function optimization experiments were performed with an AutoRally platform on the large GT-ARF. All experiments were performed with the same chassis for consistency. Due to the time required for the cost function optimization experiments, we selected three experiments to perform instead of the full six across all representations and optimization methods. The first is single speed target optimization with cross entropy, and other two are centerline RBF using CE and PI^2. The single speed target optimization is a baseline and serves as a direct comparison to previous MPPI results that used the same form of speed cost component, but was selected by hand tuning instead of automatically. The centerline RBF representation was chosen for the other experiments because we believe that it offers a better representation for our task of minimizing lap times than grid RBFs.

All three experiments were run on a damp track from rain a few days prior, but without mud and standing water. Depending on the time of year in Atlanta, the track may never completely dry out for more than a month, and other times it is bone dry for months. The conditions were a result of testing in the Spring in between regular rain storms. From previous work, we know that these are among the best track conditions for testing, and further discuss how track conditions effect performance and generalization later in this chapter. The single speed target optimization ran for 87 laps over 8 epochs, and resulted in a minimum lap time of 29.6 s generated by a target speed of 9.9 m/s. Optimization performance is shown in Figure 4.18. The initial parameter values were the same for CE and PI^2 optimization tests with 15 RBFs of size 0.04, size variance of 0.01, height of 1.5 m/s, height variance of 1.5 m/s, and location variance of 0.04.

Examining the speed cost functions that produced the fastest lap times for each optimization method shown in Figures 4.16 and 4.17, they appear to resemble the maps generated in the Gazebo simulation more closely then those from the ground truth simulation. From our previous analysis, this indicates that the primary focus of the cost function optimization in the real world is to determine the target speed at each point on the task with little regard to the adapting sensitivity of the optimization to the speed cost. As a result, it appears that, although the dynamics model MPPI used for the experiments was trained from data collected on a real AutoRally platform, that dynamics model differs from the system dynamics on the day of testing. From previous MPPI analysis, it was also shown that the dynamics model is only capable of accurately predicting motions about 0.5 seconds into the future. Despite this fast, a high level of performance is achieved in the experiments due to both the fast feedback and optimization of the MPPI algorithm as well as the optimized speed map.

The speed map that produced the fastest lap in the CE optimization one that has higher speeds after turns which then slowly decrease as approaching the next turn. Keep in mind the overall direction of travel in all of the experiments is counter clockwise. The speed map from PI^2 optimization indicates some sensitivity optimization as a high speed target region just before the tightest turn on the track would be relatively far from the actual speed driven through that portion of the track. Across all experiments, both optimization methods



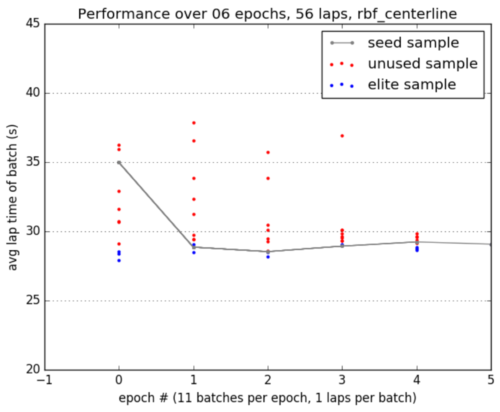
(a)

Figure 4.18: Single speed target cost function optimization at the GT-ARF large track. Lap times are shown over 8 epochs of optimization, 87 total laps around the track. (a) Performance with the cross entropy optimization method, optimized target speed is 9.9 m/s.

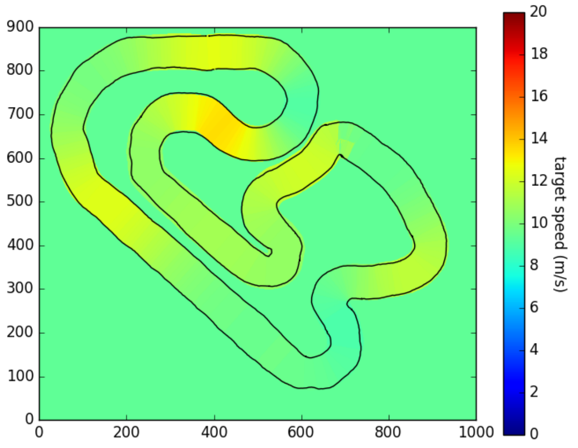
Table 4.3: Minimum lap times achieves across all cost function optimization conditions in the Gazebo simulation. An ‘X’ indicates a condition where no experimental data was collected.

		Cost Function Representation		
		Single Speed Target	Grid RBF	Centerline RBF
Opt. Method	Cross Entropy	29.6	X	27.91
	PI Squared	X	X	**27.68**

produce similar results. An example of how the two methods differ in computing the output from an epoch of optimization from the real world experiments is shown in Figure 4.21. The lap times and speed cost functions are shown for the three fastest speed functions. With the eliteness threshold of 0.2 used in CE, the top three samples are averaged together to create the output. For PI^2, the weight of each sample is computed according to the time differences between the samples. For the example in the figure, PI^2 is better able to focus on the one sample that outperforms the others, and one sample in the next epoch generates an even faster lap time, whereas the CE experiment does not produce a faster lap time. Note that both methods incorporate information from multiple speed maps, so the lap time generated from the output from an epoch are usually slower than the fastest map of the previous epoch. A primary reason for that is because samples can achieve a good lap time many different ways in the early epochs of optimization, but the differing samples can interfere with each other when combined.

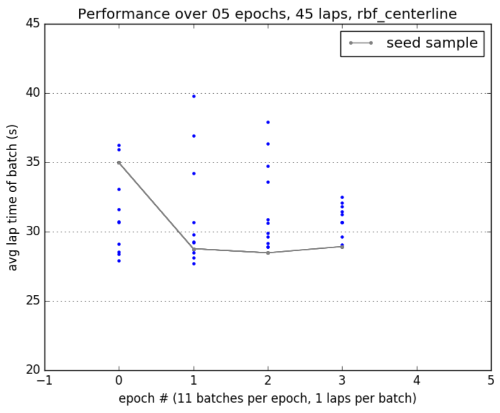


(a)

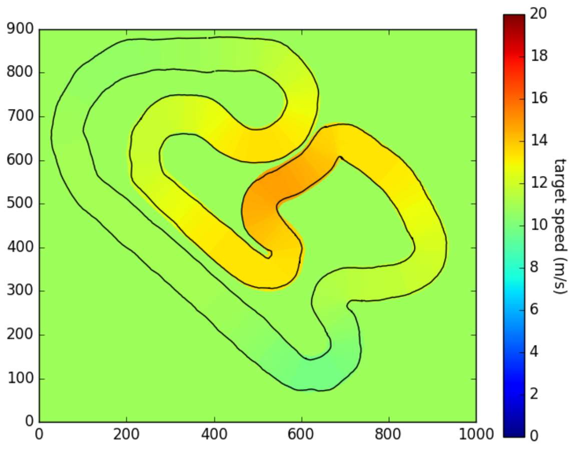


(b)

Figure 4.19: Centerline RBF cost function representation with CE optimization at the GT-ARF large track. (a) Lap time performance over 6 epochs of optimization, 56 total laps around the track. (b) Visualization of the speed cost that produced the fastest lap time, direction of motion is counter clockwise.



(a)



(b)

Figure 4.20: Centerline RBF cost function representation with PI^2 optimization in the ground truth simulation environment. (a) Lap time performance over 5 epochs of optimization, 45 total laps around the track. (b) Visualization of the speed cost that produced the fastest lap time, direction of motion is counter clockwise.

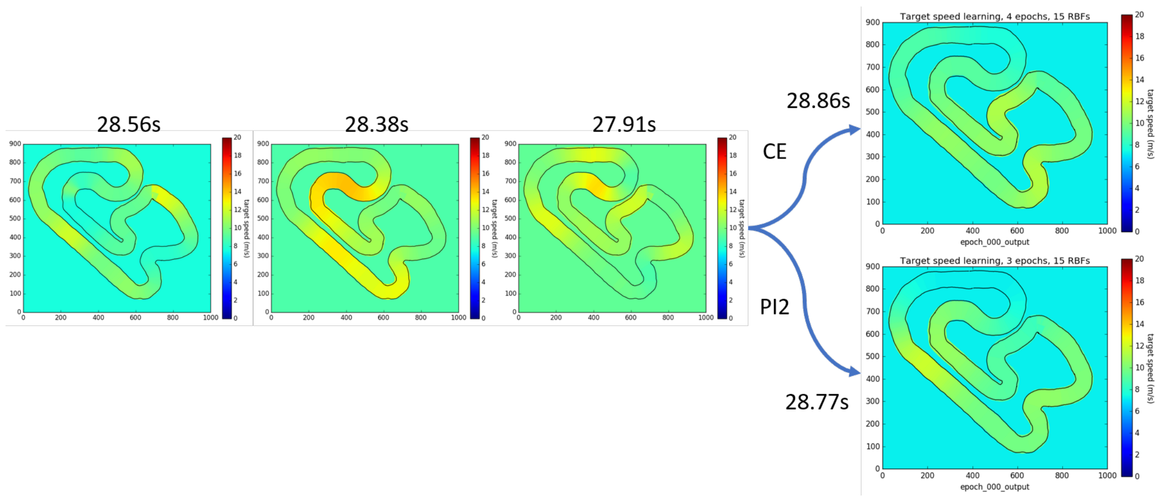


Figure 4.21: Comparison of cross entropy and PI^2 optimization methods with RBF centerline representation during testing at GT-ARF large track. The best three sampled speed cost functions from one epoch along with their lap times are shown left, followed by the computed speed cost functions and the associated lap times for each optimization method.

A common factor in all of the real world cost function optimization results are that the fastest laps came during the first few optimization epochs, which were then followed by a slow increase in lap times as optimization continued. We believe there are a variety of reasons for this behavior, mainly due to environmental factor such as track surface wear, battery levels, and hardware health.

Many fewer laps were completed for these two experiments than the simulations because real world experiments are much more difficult and time consuming. Over the course of the cost function optimization, the AutoRally vehicles crashed several times with a variety of severity and causes. Some crashes involved the platform sliding wide on a turn into the track barriers while others had the robot roll over at high speed when attempting to drift. If the platform was still operational after a crash, we would restart the optimization after assigning the map that caused the crash a high lap time as a penalty. Over the course of the experiments at GR-ARF, 186 laps were completed which represents about 20 miles of autonomous driving, and 9 rollover crashes which were the only unplanned times that the human safety driver had to assume manual control. An image sequence taken from an off board camera shows an example rollover crash due to excessive side slip at high speed in

Figure 4.23. The scheduled human takeovers occurred every two epochs of optimization to bring the platform back to the pit lane to swap chassis batteries and then continue running.

At the end of each lap during cost function optimization, a new speed map is fed to MPPI. This process takes some non-zero amount of time, so one option is to discard the lap time generated from the lap when the speed map occurred. That is not desirable to have to discard half of your driving, especially in real world testing dealing with hardware. Instead, we implemented a series of timing gates around the track that allow adequate space after a lap is completed for a new speed map to be passed to MPPI before starting timing on a new lap. The eight timing gates are shown in Figure 4.22. The lap timing works by incrementing the start/stop line by one gate in the direction of travel around the track (counter clockwise), so that the switching time is not counted into driving performance, and only 1/8 of a lap of driving is discarded instead of 1 full lap if the start/stop line were to remain stationary.

Prior to this work, the best performance of the MPPI controller at the GT-ARF large track was from [47] that reported a fast lap time of 27.9 s during a five lap batch of driving as fast as possible. This dataset was collected after multiple days of a controls theoretician tuning the algorithm and adjusting the setup of the AutoRally platform to ensure the most favorable testing conditions possible. The centerline RBF representation with CE optimization recorded a fast lap time of 27.91 s to match previous performance through cost function optimization without the MPC designer on site for tuning after just a few minutes of driving on the track. Further, the centerline RBF representation with PI^2 optimization recorded a lap time of 27.68 s, which outperforms all previous results with the MPPI controller driving an AutoRally platform at the GT-ARF large track.

Robots in the Real World

The MPC-CFO experiments have been the longest set of experiments run on the platforms to date. All together, in preparation for the experiments and running intermediate versions

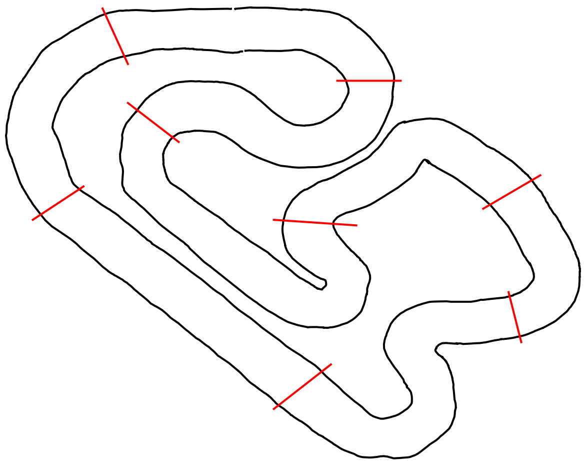


Figure 4.22: Timing gates for cost function optimization. After one complete lap is completed, the start/finish line for the next lap is moved one position counter clockwise around the track to take into account the time it takes to update the cost function MPPI is using while the robots is moving.

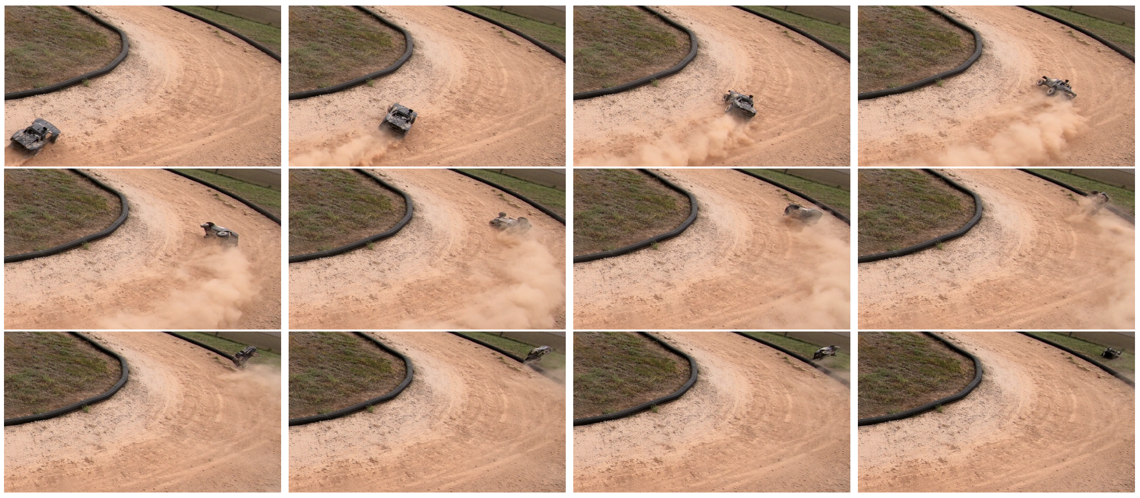
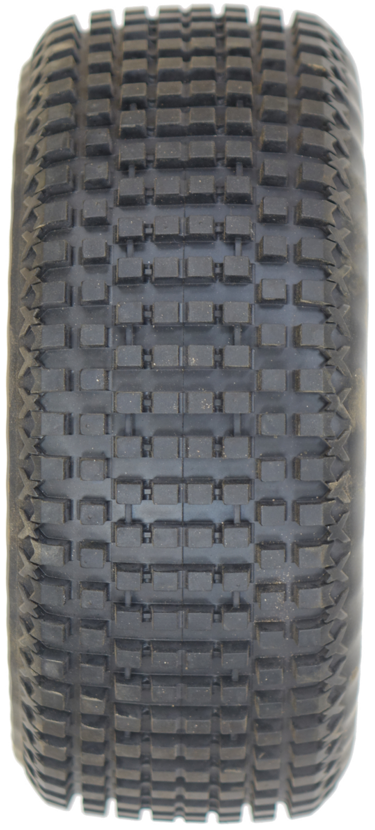


Figure 4.23: Sequence of images captured from offboard video of a multiple rollover crash during cost function optimization at GT-ARF large track. The crash was a result of excessive speed and side slip going through a turn.

of the cost function algorithm resulted in more than 500 laps of autonomous driving, which represents more than 50 miles. Over the course of the experiments at the GT-ARF track, many parts wore out and were replaced, but it became apparent that a few critical components on the platform that had an effect on the overall performance level of the vehicle wore down regularly during experiments. One example, which we have not yet been able to quantify the extend to which it effects performance, is tire wear, shown in Figure 4.24. The wear occurs over the same time scales as the cost function optimization experiments, so in order to minimize the amount that tire wear effected performance, the rear tires were swapped for a set with new tread every 80 laps, or sooner if the tread showed signs of deterioration. One reason that the tires wear down so fast during these experiments is that every lap is pushing the vehicle to the performance limits with hard accelerations several times per lap and significant side slip angles at a variety of speed. That aggressive driving, coupled with an overall platform that weighs about 50% more than the stock platform, contributes to tires that wear out much faster than in previous testing. Additionally, testing in high traction conditions when there is still moisture in the track results in the highest level of performance, but also the fastest tire wear because of the relatively high grip driving surface.

Anecdotally, running with relative new tires on the AutoRally platform rarely results in the spin-outs, but when the tires are worn, spin-outs and significant fish-tailing when attempting hard accelerations out of sharp turns become common. Breaking from high speed in preparation for a slow turn with worn tires can also cause the robot to slide through a turn into a barrier due to an inability to slow down as fast as expected. Combine worn tires with a dry and dusty track and the problem is just made worse. One potential solution to vehicle dynamics that change on the time scale of high 10s of laps would be model adaptation. Model adaptation has been explored on AutoRally, but was explicitly excluded from use with cost function optimization as it would mean that MPC-CFO is optimizing over a non-stationary distribution. This could be a future direction of work focused on


(a)

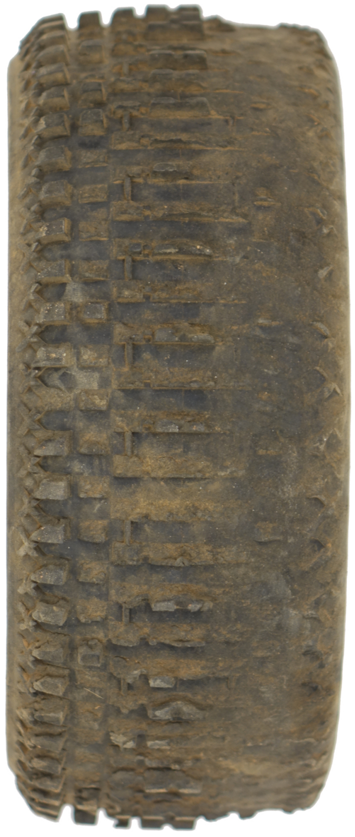

(b)

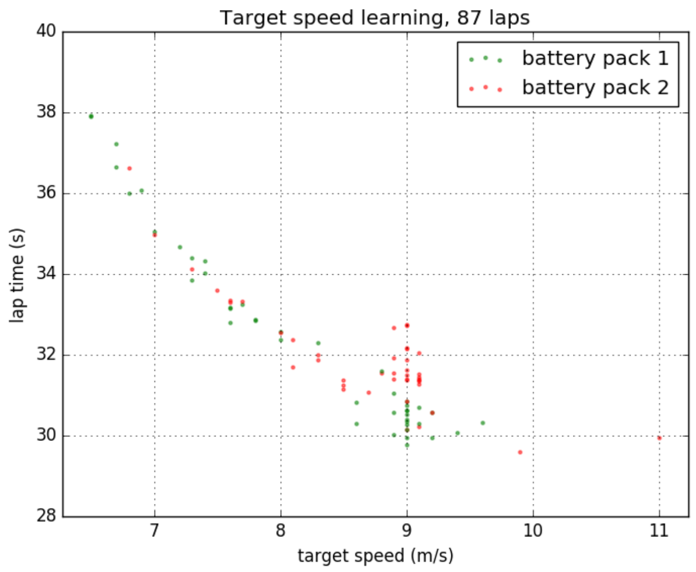
Figure 4.24: Comparison of new tires and worn tires after aggressive driving tests with the AutoRally platform at GT-ARF. (a) New tires with zero laps of wear. (b) Worn tires after 120 laps (about 12 miles) of aggressive driving.

generalization of MPC-CFO.

Over the course of testing in the real world, the chassis battery packs have to be changed regularly. The standard packs will last for for 20 to 25 laps of aggressive driving at the GT-ARF large track. With six AutoRally platforms, there are plenty of spare battery packs so standard practice when testing is to use two or three packs of chassis batteries at each test so that the robot can run as much as possible as the batteries are cycled between draining on the robot and charging.

During the MPC-CFO experiment for single speed target optimization, oscillations in lap times were noted toward the end of the optimization that did not appear to correlate with the target speed that was being tested. Upon further inspection of the hardware there were no apparent mechanical or electrical failures or signs of wear that could cause that behavior. When combing through the data from the experiment shown in Figure 4.18, we decided to plot the lap times as a function of the target speed and color code the lap times according to which of the two battery packs were install in the chassis at that time (we swapped between the two battery packs every 2 epochs that day, for the duration of the test). What this analysis shows, presented in Figure 4.25, is that the selected battery pack can have a significant effect on performance when attempting to drive at the limits of performance. Under casual driving behaviors, we see that the performance of each battery pack is indistinguishable from the other. However, under high load conditions one battery pack is able to keep up with the power demands of the robot while the other pack lags. The result is two distinct clusters of lap times about 1.5 seconds apart when operating with target speeds that push the limits of performance of the platform on the GT-ARF large track. This difference is likely due to the natural wear and tear that battery packs endure over their life cycle, and is another reminder for how much seemingly insignificant variations in hardware configuration can have large effects on real world performance.

Consider the following scenario: the strong battery pack is run for two epochs of cost function optimization. At the end of those epochs the vehicle is driving close to the perfor-



(a)

Figure 4.25: Lap time as a function of target speed during single speed target optimization experiment. Lap times are colored according to which of the two chassis battery packs used the in the test was powering the chassis.

mance limits, requiring consistently high power draw form the batteries. Then, the weaker pack is swapped in under standard testing conditions and the result is a performance plateau or degradation in subsequent epochs. One would normally conclude that the optimization has converged, but in fact that perceived converges could have just been a result of swapping in a battery pack that is not capable of driving at the level of the previous pack.

The difference in battery pack performance was a also noted by the human expert driver, which will be discussed in the next chapter, with the comment that one of the battery packs felt "squishy." When asked to clarify, the driver indicated that one of the battery packs provided much more responsive throttle behavior than other other pack.

4.3.5 Cost Function Optimization Effect on MPPI

The question this chapter seeks to answer is whether MPC-CFO is a methodology that can automatically tune a cost function as a vehicle drives around a track over the course of a few minutes of driving in order to achieve a high level of performance. It views the particular

MPC method used as a black box that is being optimized given lap times. This section departs from that goal to open that box and analyze what the cost function optimization is doing with the MPPI controller. It was hinted at in previous sections to describe the qualitative differences between optimized speed maps across the testing environments, but this section will present a more in-depth analysis of the claim that MPC-CFO is doing essentially to things for MPPI. At the first level, the cost function optimization performs as intended to determine what speed the vehicle should be traveling at each point along a track. However, MPPI uses more than the cost function to determine how to drive. Specifically, the other major component in MPPI is the dynamics model, and the difference between that dynamics model and the dynamics of the environment MPPI is driving in has a secondary effect on the cost function optimization. This secondary effect is for the cost function optimization operating on speed maps used by MPPI adjusts the sensitivity of MPPI to speed component of the cost function.

To dig deeper into what is happening we first need a metric to quantify what if happening with the cost functions. Our metric is the average difference between the speed map and the actual speed when a vehicle is driven by MPPI using that speed map. To collect the data for this metric, MPPI is run for about 20 laps with the speed map to be studied. Then, the trajectory MPPI driven is overlaid on top of the speed map to generate our metric and the graphs in this section. We chose a subset of the total experiments to conduct this analysis on: single speed target and centerline RBF representations from the debug simulation, and a centerline RBF representation from the Gazebo simulation environment. As a baseline for the comparison, we see that in the case of MPPI operating in the ground truth simulation environment with a single speed target map, shown in Figure 4.26, the speed target acts as an upper bound on speed. At the fastest point on the track, which a little further than half way down the longest straight, the vehicle reaches speeds close to the speed target. Everywhere else around the track, the actual speed is far below the speed target. The speed difference results are presented in Table 4.4.

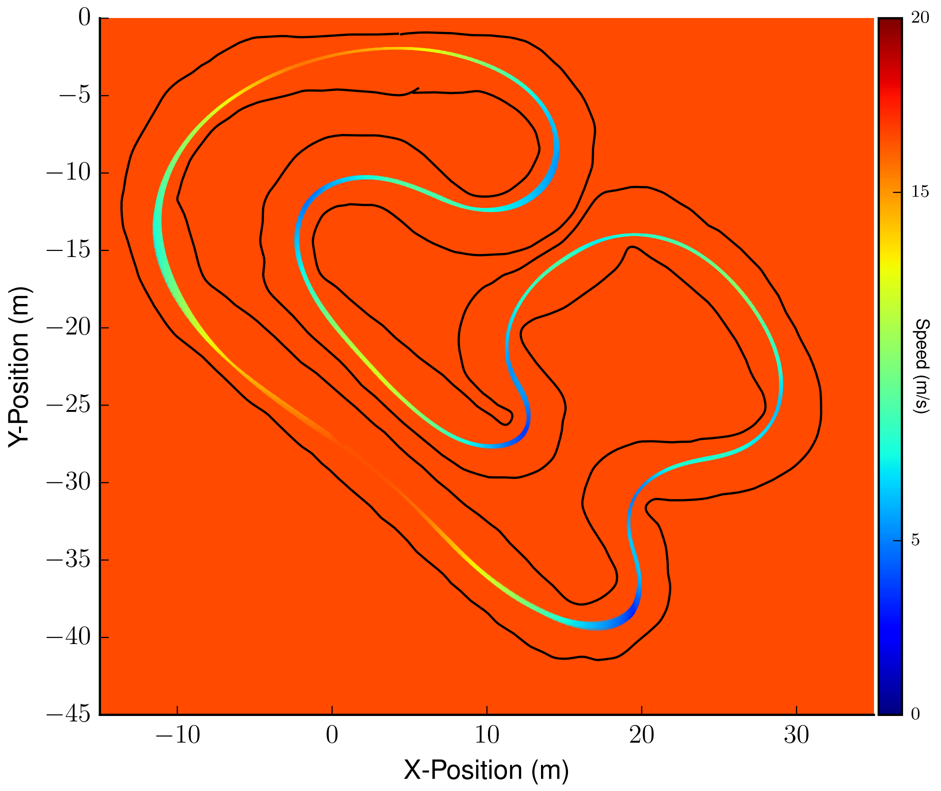


Figure 4.26: Trajectory traces colored by speed for MPPI driving overlaid on the constant speed map provided in the ground truth simulation environment.

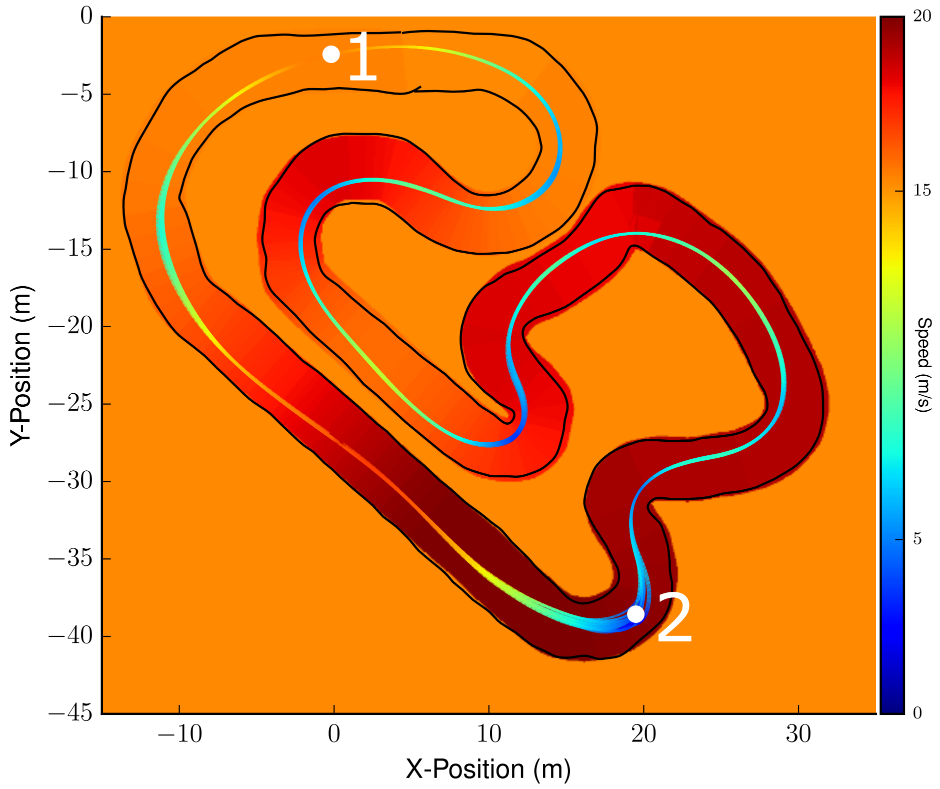


Figure 4.27: Trajectory traces colored by speeed for MPPI driving overlaid on the centerline RBF speed map provided in the ground truth simulation environment. Example points selected to evaluate cost function behavior at two parts of the track.

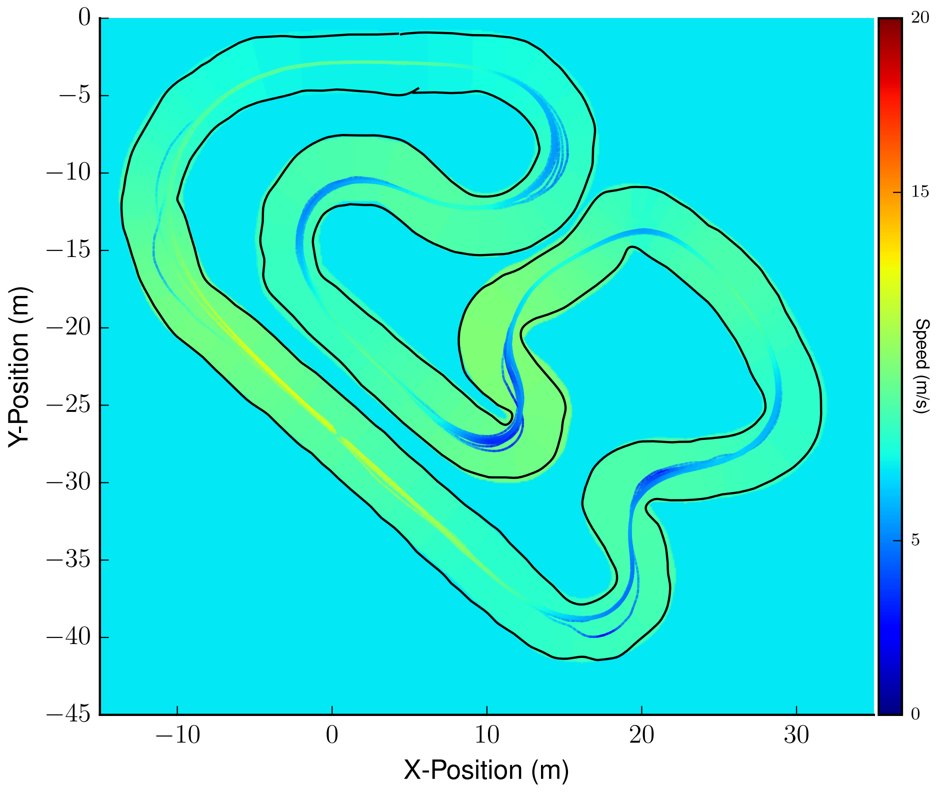


Figure 4.28: Trajectory traces colored by speed for MPPI driving overlaid on the centerline RBF speed map provided in the Gazebo simulation environment.

To understand more about why this secondary effect of sensitivity optimization is happening, we examined the cost function component values at a few chosen points with the RBF centerline speed map in the ground truth simulation environment. The weight vector for all of the test was $\omega_1 = 200, \omega_2 = 4.25, \omega_3 = 10,000, \omega_4 = 100$. The two examination points shown in Figure 4.28 were selected as "easy" and "difficult" areas of the track for MPPI to determine a high quality control plan. The individual cost function component values for the minimum cost rollout are shown as well as the minimum, average, and maximum costs for one iteration of MPPI in Table 4.5. At point 1, the "easy" location, the vehicle is in the middle of the track, travels close to the desired speed, few rollouts end up off the track, and slip is minimal, so the min cost trajectory has a low cost. The difficulty for MPPI to find a solution is also reflected in the rollout cost statistics in addition to the composition of the minimum cost rollout. Relative to each other, the track cost and slip penalty are the dominant terms indicating that MPPI is primarily focused on positioning on the track and how aggressive to drive around the upcoming turn as opposed to worried about crashing or driving at a different speed. Examining the cost function components at point 2, their relative weights have shifted dramatically with the speed cost and crash penalty acting as the dominant terms, and the magnitude of each component has increased significantly, indicating a much more difficult situation for MPPI than point 1. In regards to how the specific cost map is able to produce fast lap times compared to one that aligned well with the actual speed of the vehicle, we can see that at difficult points in the track, which are defined as locations where MPPI can only find high cost rollouts, the high desired speed acts to balance the high crash penalty in the cost function to keep the vehicle moving forward faster than it would have if the target speed was low. Another interpretation, keeping in mind that MPPI was operating with perfect knowledge of the dynamics in this scenario, is that cost function optimization places an out-sized emphasis on low probability trajectories in difficult situations by tuning the sensitivity of the cost function to speed. It is able to do that in this environment, whereas in Gazebo and the real world it is not, due to the fact that

Table 4.4: Difference between desired speed specified in optimized speed cost and actual speed driven by MPPI in simulation environments. A negative value indicates that the actual speed was less than the desired speed.

Environment	Speed Cost Representation	Avg Speed Difference (m/s)	Speed Difference %
Ground Truth	Single Speed Target	-10.29	-1.63
Ground Truth	Centerline RBF	-8.94	1.41
Gazebo	Centerline RBF	-1.67	-0.31

Table 4.5: Example cost function evaluation at two points in the debug simulation environment. Selected points are shown in Figure 4.27. The individual component values for the minimum cost rollout are broken out and statistics for all 1920 rollouts are presented.

Min Cost Rollout Components					Rollout Cost Statistics		
Point	Track Cost	Speed Cost	Crash Penalty	Slip Penalty	Min	Avg	Max
1	72	5	39	77	171	1,280	9,048
2	40	352	324	130	846	4,268	17,288

when it is operating with perfect dynamics knowledge, the algorithm can determine exactly if a crash will or will not occur.

4.3.6 Cost Function Optimization Generalization

Although the goal of cost function optimization and the CFO-MPC framework as whole, is to explore what level of performance is attainable by specializing performance of an MPC to a specific track, it can insightful to understand where, and to what extent, MPC-CFO performance can or cannot generalize.

From our experiments we have three separate versions of the same track in two simulated and one real environment. The first simulation environment is one in which the MPC method has perfect knowledge of the dynamics. The second simulation environment is one built using Gazebo and offer the full software interface to the AutoRally platform including simulated sensors and actuators, but the physics of the simulation world cannot simulate tire-ground interactions well. As a result, the simulation is numerically unstable as the vehicle approached sliding regimes and the dynamics model used by the MPC does not do a good job capturing the world dynamics. With these two environments in mind, we can explore if a cost function we optimize in on simulation can be used by MPPI in the other, and what level of performance does it enable. Providing MPPI with a cost function optimized in the ground truth simulation when driving AutoRally in the Gazebo simulation results in immediate crashes. In no experiment was the vehicle able to even complete one turn, often times the vehicle immediately drove out of control and rolled when attempting to accelerate on a straight. For the direction of ground truth simulation to Gazebo, it is not possible to transfer cost functions for use by MPPI. This result in unsurprising considering the analysis of the difference between optimized cost functions that showed for the ground truth environment the target speed was, on average, about 9 m/s or more faster than the realized speed, whereas for the Gazebo simulation, the target speed was only 1.67 m/s faster than the realized speeds. Testing the transfer from Gazebo to ground truth, we would expect the vehicle to successfully navigate around the track, but the level of performance would be lower than for speed cost functions optimized in that map. For this test, we selected the fastest speed cost function from the centerline RBF representation in Gazebo that achieved a lap time of 29.84 s, shown in Figure 4.29. As expected, MPPI is successfully able to navigate around the track without crashing, and produces an average lap time of 32.85 ± 0.04 s over 26 laps of driving. There was no reason to expect that the test would fail given the previous understanding of performance in the ground truth simulation environment, but it is interesting to see that there is a performance loss of about 3 s in lap time, or 10%.

Considering that one of the primary differences in simulation environments is how closely the MPC dynamics model captures the environment dynamics, we elected to not attempt to test transferability of any speed maps optimized in simulation to the real world because we would expect a high likelihood of crashes. The results from the cost function optimization at the real world GT-ARF track indicate the dynamics model is closer in difference to the Gazebo simulation than the ground truth simulation. One generalization test that we unexpectedly conducted was attempting to collect MPC-CFO driving with an optimized speed map on a different day than the optimization was performed on. In an attempt to characterize the driving performance at GT-ARF, we tasked MPPI with driving

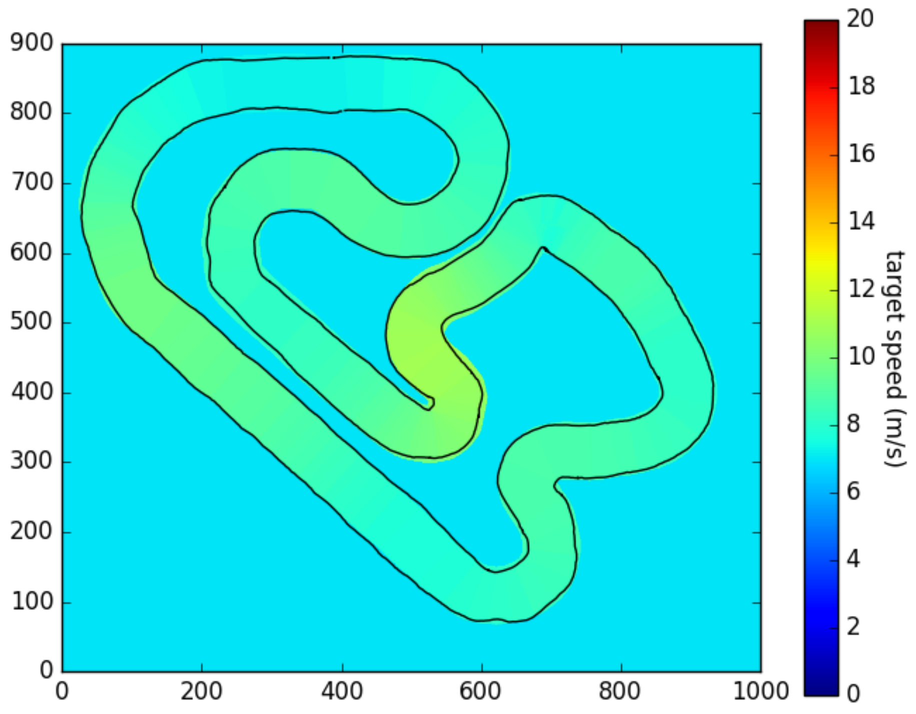


Figure 4.29: Target speed function used to test generalization from Gazebo to ground truth simulation. Optimized in Gazebo with CE and centerline RBF representation.

using the speed map that generated the 27.68 s lap time. That lap time was generated in the late morning on the track that was damp in some parts. The next day we returned to the track, now completely dry and dusty from testing, and started the experiment with the fixed speed map. On the first turn of test the AutoRally robot initiated its turn too late and slid off the track. After confirming that the state estimator was converged and accurate, we found that the path MPPI expected to follow and the one the platform actually followed were very different due to the low traction environment compared to the day prior. An example of dry and wet track conditions is shown in Figure 4.30. What we learned from this experiment is that not only is MPC-CFO specializing performance to a specific track, it is also specializing to the testing conditions of the day. Also, MPPI has been shown to be able to drive AutoRally aggressively across track conditions, but that is no longer the case for MPC-CFO when the goal si to squeeze every last bit of performance out of the track on a particular day. For this reason, cost function optimization performed on one day

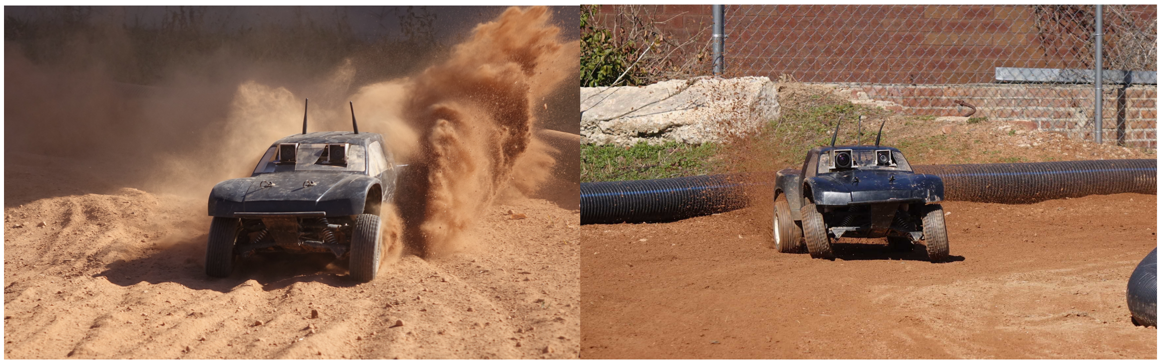

(a) (b)

Figure 4.30: Examples of track surface variability during testing. (a) A dry and dusty track where traction is low. (b) A wet but not muddy track that offers a high traction environment surface.

is only useful if the same testing conditions are encountered again, otherwise cost function optimization must be repeated at the beginning of every test if the world has changed in any way. A question of generalization within the same world is an interesting question, such as to other track geometries, but not one that we have explored in this work.

CHAPTER 5
RALLY RACING

In previous chapters, we presented a system capable of autonomously performing the task of rally racing using the AutoRally platform, the MPPI algorithm, and cost function optimization at the GT-ARF track. Using cost function optimization we demonstrated the capability to increase performance of the MPPI algorithm beyond what was possible with expert tuning. In this chapter, we present a comparison of autonomous and human agent performance driving an AutoRally platform in a time trial style rally race at the GT-ARF large track. To the best of our knowledge, this work will constitute the first dirt track racing competition of an autonomous system and human expert. Further, to control as many factors as possible, all drivers will use the same AutoRally platform. This competition is an effort to quantify the level of performance of our autonomous system and the MPC-CFO framework with respect to what we believe is is the highest level of performance possible, and provide a quantitative, implementation agnostic benchmark based on lap times. This benchmark allows comparison of past and future work with the AutoRally vehicles. To the best of our knowledge, this comparison the first of its kind to bring together an expert human and state-of-the-art AI research in a real world task. Further, the data will be publicly released as part of the AutoRally dataset.

5.1 Gaming Competitions

Competitions have long been a way to benchmark and compare AI agents with each other and against humans. One classic example of an AI system able to achieve super-human performance is Deep Blue, which defeated Garry Kasparov, the reigning world chess champion, in a six-game match [51]. Recent advancements in deep learning have shown super-human performance in many classic arcade-style video games [52] from the Atari platform.

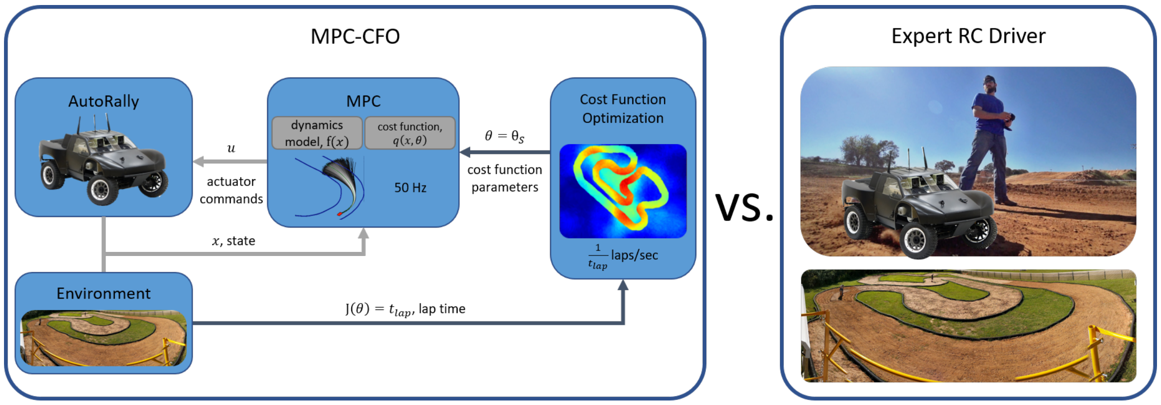


Figure 5.1: Rally racing competition between MPC-CFO and an expert human driver using AutoRally at the Georgia Tech Autonomous Racing Facility.

AlphaGo and AlphaGo Zero bested the best humans in the world in the game of Go [53], which is a difficult problem for an AI to solve because of the large branching factor. Direct competitions between AI and a single human or teams of humans were conducted with the real time strategy games StarCraft by DeepMind [54] and Dota 2 by OpenAI. The OpenAI Dota 1v1 bot was able to defeat expert humans at the game utilizes 60,000 CPU cores and 256 K80 GPUs and collects 300 years of game play experience per day during training. Despite demonstrating super-human performance in many games, these competitions all leverage simulations where the interactions with the environment can be modeled, whereas in the real world racing domain, there is no way to model all of the interaction inherent .

5.2 Vehicle Racing

For the task of racing in the real world, the number of interactions are severely limited as experiments need constant human supervision and hardware has limited durability. A dataset of two expert human racers driving five vintage race cars outfitted with sensors at Laguna Seca was collected, analyzed, and publicly released in [26]. This dataset represents the closest related work to our goal of comparing human to autonomous performance in the real world setting of vehicle racing. However, the vehicles in the study are not available for further testing so no direct comparison is possible and the vehicles were not equipped for autonomous driving. A vehicle outfitted for data logging human driving behaviors was used in [27] to record a human performing circular drifts, but also lacks the capability for autonomous driving. In [23, 55], an Audi TT outfitted for autonomous drove pre-planned paths generated from first principles vehicle models and racing lines at the Pikes Peak Hill Climb the Thunderhill Raceway. This work was the first to show an autonomous vehicle operating at a high level of performance in a racing scenario, but no direct performance comparison to human performance exists, there is no outside access to the platform for future work, and the reliance on tracking a pre-planned trajectory assumes a nearly perfect model of the environment and vehicle dynamics means that the vehicle cannot recover from large disturbances or variations in conditions during test. A follow-on to that work, [**laurense2017path**] quantified the acceptable error between computed and measured tire friction components of 2% is required for stability of the method.

The Roborace event [56], is an announced racing event that is an offshoot of the electric car racing competition Formula E, which is itself an offshoot of Formula 1 racing. Roborace promises to pit autonomous electric vehicles against one another in a head-to-head race on the same track as Formula 1 and Formula E events currently race on. To date, no competition dates or list of competitors have been announced and available information is limited to press releases and videos of two development platforms following GPS waypoints at moderate speeds through Formula 1 race courses. Current performance places the autonomous driver about 10% behind an expert human where the expert human is Roborace's CEO and current Formula E driver Lucas di Grassi. The RACECAR project at MIT [13] and F1/10 Autonomous Vehicle [12] have been used for autonomous driving competitions as part of undergraduate courses on self-driving vehicles where the goal is to navigate indoor hallways as fast as possible. The limited computational and battery capacity, lack of protective housing, and design for indoor conditions limits the platform to a teaching tool, and it cannot run state-of-the-art self-driving vehicle algorithms, operate outdoors on a dirt surface up to the mechanical limits of the system, and no performance to humans has been presented. The CARLA Autonomous Driving Challenge [20] is a simulation-based autonomous vehicle challenge where teams can choose from different competition tracks and compete in driving competency in urban driving scenarios. This competition focuses on following road rules and driving safely at all times in urban environments. The competition will take place Summer 2019, and while it is a competition evaluating self-driving vehicle performance, it is not directly comparable as urban driving requires a very different set of capabilities than closed track racing.

5.3 Rally Race at Georgia Tech Autonomous Racing Facility

To conduct the competition, multiple human and autonomous driver were tasked with driving AutoRally around the GT-ARF large track as fast as possible while all actuator and sensor data was logged. Altogether, four drivers participated in the competition, two humans and two autonomous. The human drivers were allowed to walk around and inspect the track and then drive AutoRally however they wanted for the duration of a battery pack, about 20 minutes of driving, to warm up. Human drivers stood on top of a one story tall scaffolding tower at the side of the track while they drove to provide an improved vantage point from a small ladder or ground level perspective.

For each of the human drivers, the data was broken down into individual laps for analysis. The most important metric for the competition is the fastest recorded lap time over the course of testing as an overall rank of driving performance. Lap time includes the ability of a driver to navigate turns, control positioning on the track, and smoothly accelerate and brake all in one number that is independent of any particular approach to tackling the task. Many other metrics can help quantify and understand performance differences, and inspection of the trajectories driven can provide insights into strengths and weaknesses. Figure 5.2 shows the turn numbering around the GT-ARF large track, starting with turn one at the end of the main straight, and counting up to turn nine as the last turn each lap. For each human driver, an average trajectory was computed from all laps of data. Even

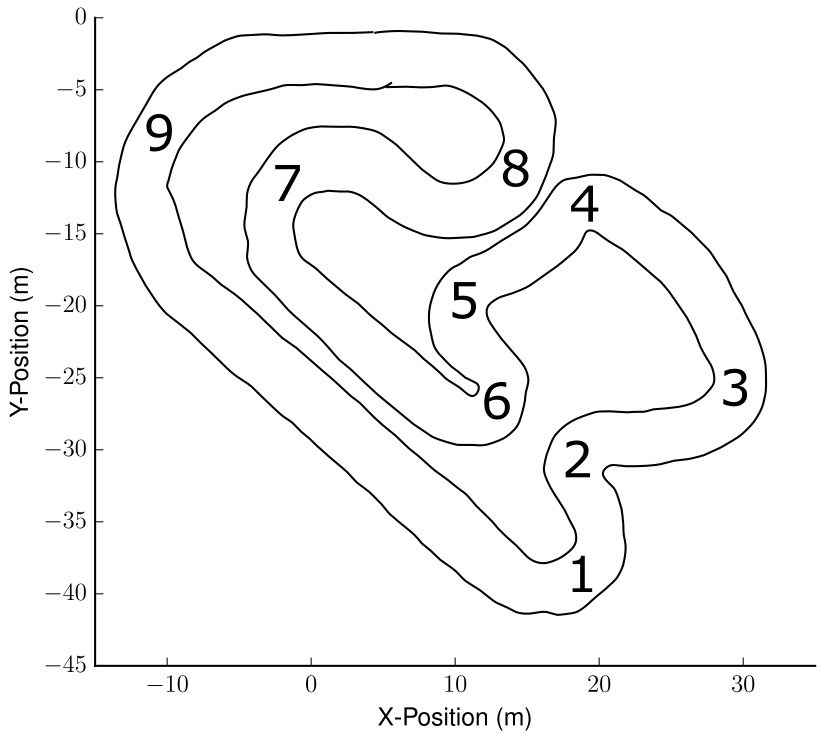


Figure 5.2: Turn numbers at the GT-ARF large track. Direction of travel is counter clockwise in all experiments with turn numbering beginning at the end of the main straight.

though each lap begins at the same location on the track, one cannot just average all of the laps together to generate the average lap for each driver since each lap is a different length of time. Instead, the raw positions and velocities from the state estimator must were splined and resampled at constant intervals to align the trajectories. The computed average trajectories for each driver are shown in addition to the raw laps.

Tracing through the turns, the first sequence at the end of the straight is one-two. To successfully navigate these first two turns it is critical for a driver to slow sufficiently from the straight to allow for an early turn entry and not slide wide. These two turns should be cut as straight as possible so that AutoRally can accelerate for the next urn sequence. Turns three-four-five can all be driven as one large left turn if a driver is able to use the full width of the turns for positioning. In reality, the track surface between three and four has a significant off-camber slope that causes vehicles to slide wider than expected, frequently off the track if too much speed is carried. As a result, most drivers navigate those turn cautiously by hugging the inner barrier and treating turns three to five as separate features.

Turn six is the slowest turn of the track. The best lines through this track are to enter the turn early and accelerate hard onto the small straight before turn seven. Turns seven-eight are the same diameter, and can be linked together if the driver is able to quickly shift steering from side to side. The exit from turn eight is critical to carry speed into turn nine and then onto the main straight. Turn nine is the largest diameter and therefore fastest turn on the track, but a driver must be careful to not enter the turn with excessive speed and give up track position in order to avoid sliding out right before the straight. Driver have a tendency to try and take turn nine faster than their skills allow in anticipation of the main straight. The most difficult part of the turn nine exit is keeping AutoRally pointed down the straight under high acceleration. Across the entire track, the highest speed of the lap is normally seen just after halfway down the straight before braking must start, and the slowest speed is seen on the turn one or six.

The rest of this section presents the data from each driver, and then compares their performance.

5.3.1 Skilled Human Driver

The skilled human driver is a project member who has acted as the primary safety driver during test runs over the course of five years, was the person driving AutoRally for all of the system identification data, and raced RC cars as a hobby growing up. This human driver was, by far, the most skilled driver working on the AutoRally project.

Examining the average trajectory of the skilled human driver, they are able to reliably link multiple maneuvers together as they navigate turn sequences of tighter turns such as turns one-two and five-six as demonstrated by the trajectory cutting a low curvature path through these turn sequences as opposed to closely following the centerline of the track. On the larger turns, sequences, three-four, and seven-eight, and nine, the driving line taken by the human tends to stay to the inside of the track instead of using the full width for positioning and accelerations. This indicates a much much more cautious approach to large turn sequences that require much higher speeds. Turns three-four are especially challenging because the track has an off-camber slope. This results in very different driving behavior on that turn sequence where AutoRally easily slides off the outside of the track if there is excess speed. Turns eight and nine prove difficult for a different reason: they must be taken at high speed while accelerating in preparation for the large straight to generate good lap times. Acceleration through turns requires precise throttle and steering control to keep AutoRally pointed down the track while minimizing fishtailing due to rear wheel slip.

Number of laps	44
Fast lap time (s)	27.09
Average lap time (s)	28.43 ± 0.73
Maximum speed (m/s)	13.02 ± 0.42
Minimum speed (m/s)	2.46 ± 0.44
Average speed (m/s)	6.11 ± 0.16
Average distance (m)	179.66 ± 3.04

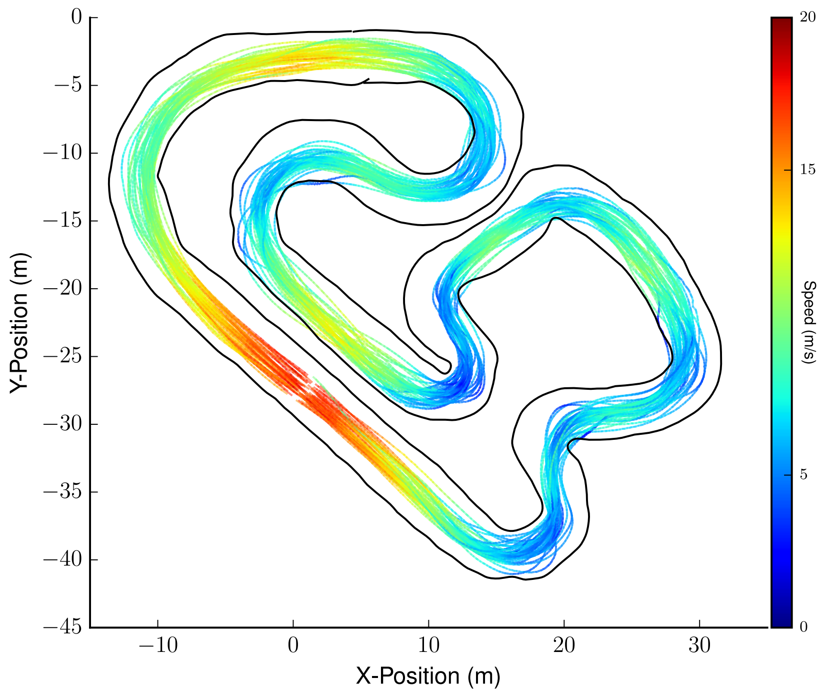
Table 5.1: Driving statistics for skilled human driver.

5.3.2 Expert Human Driver

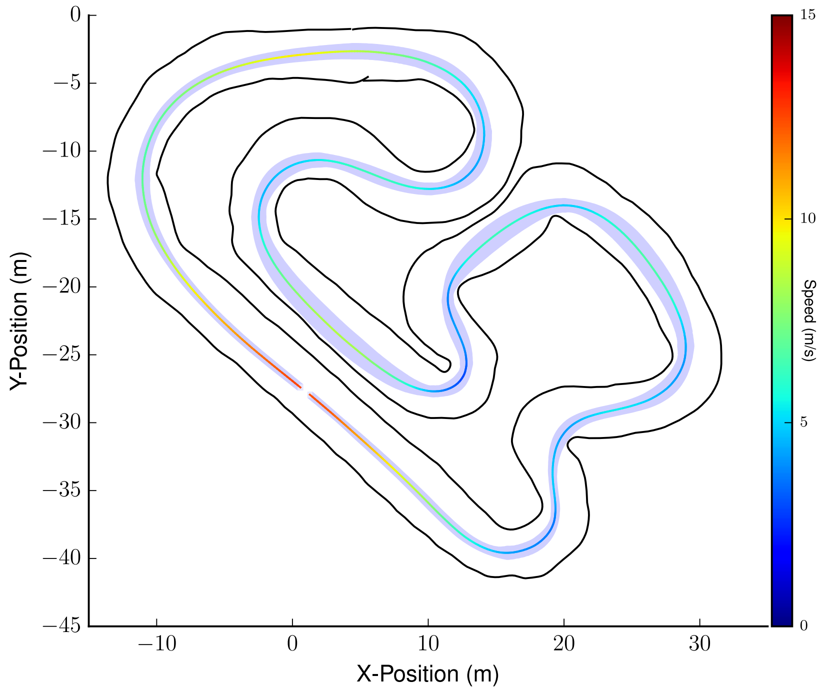
An expert for the task of racing a scaled vehicle in an off road setting is someone who has successfully raced at the competition level in events sanctioned by Remotely Operated Auto Racers (ROAR), the North American representative to the International Federation of Model Car Racing (IFMAR), which is the sanctioning body for RC car races around the world and has been active since 1985. In addition to competing in sanctioned competitions, many professional drivers also consult for RC car companies about the handling characteristics and overall impressions during development of new products.

The expert human we recruited for the study raced professionally for a decade with a carer that included several podium finishes at national and international competitions racing a variety of vehicle sizes on dirt and paved surfaces. He has since retired as a racer, but regularly participates in local competitions.

The track conditions during the human expert experiment proved to be the limiting



(a)



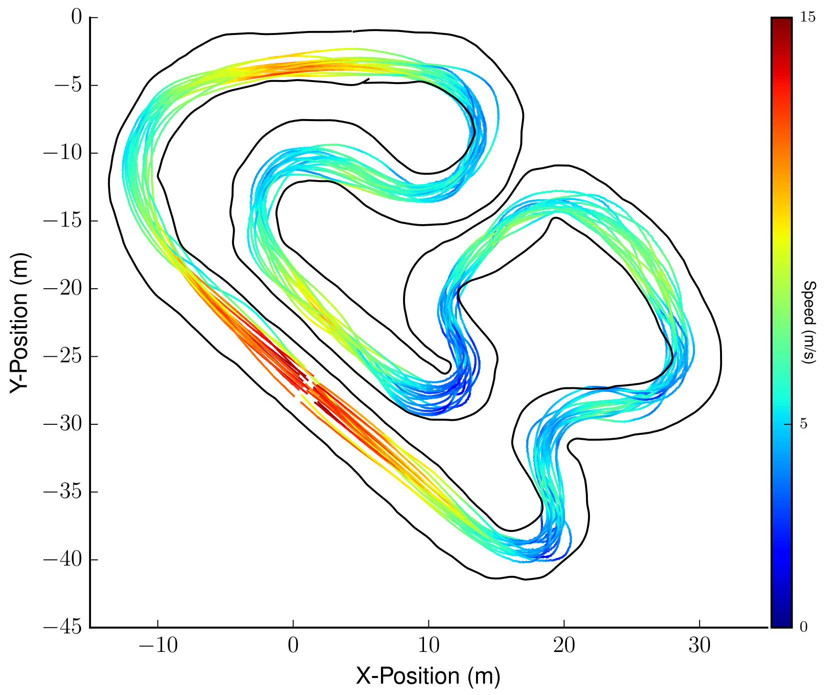
(b)

Figure 5.3: Skilled human driver data. (a) All laps colored according to speed. Breaks in trajectories around the start line on the straight are an artifact of breaking the dataset into individual laps for analysis and visualization. (b) Average trajectory along with 1-σ bound.

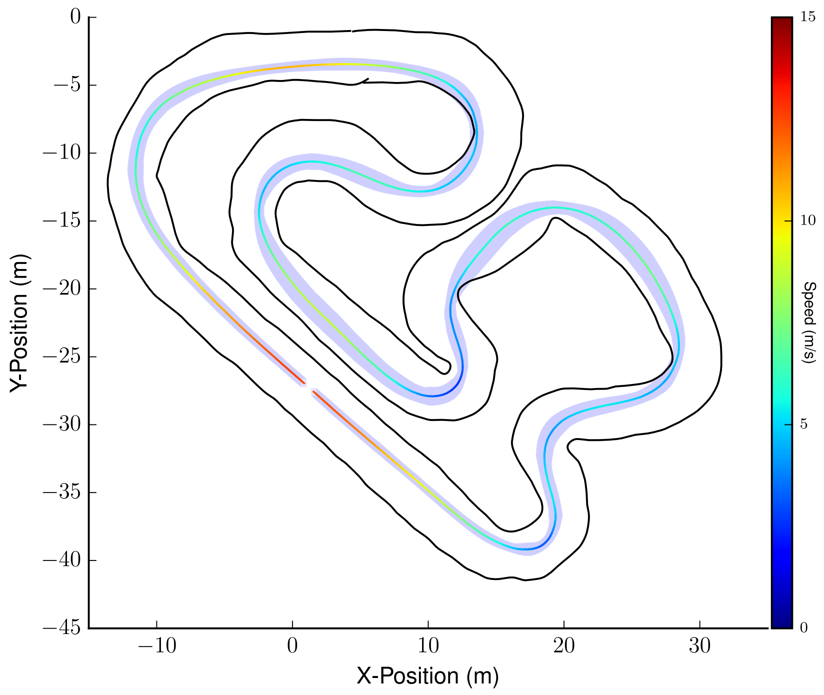
factor for data collection. Over the two days the expert was on site, the first day was cold, and the track was muddy from recent rain. By the afternoon of the second day, the track was sufficiently dry to collect one battery pack of fast running with the remaining testing time. The full speed run laps driven by the human expert are shown in Figure 5.4. Overall, the human expert fast lap of 25.56 s bests the skilled human fast lap time of 27.09 s by a margin of 1.53 seconds. The expert average lap time is also almost a half second faster than the skilled human. The expert human is able to better position the vehicle on the track to maintain speed through turns which results in a faster average lap time and higher top speeds but longer average path length. In particular, the human cuts every turn except nine very tightly, skillfully links multiple turn sequences in a row, and allows AutoRally to use more of the track width in search of faster lap times. The way in which the human expert navigates turns eight and nine is of particular interest. Instead of linking turns eight and nine to carry as much speed as possible, the human expert turns sharply around both turns, resulting in slower speeds than the other drivers during the turns, in order to focus on straight line acceleration in between turns. The sharper turning allows for AutoRally to be pointed straight for longer in between turn. This strategy sacrifices the quality of the racing line and speed in turns in order to have more time accelerating hard in a straight line, which overall appears to give a lap time advantage compared to other drivers. It allows the driver to mostly separate maneuvering around turns and hard accelerations.

Number of laps	18
Fast lap time (s)	25.56
Average lap time (s)	28.00 ± 1.05
Maximum speed (m/s)	13.34 ± 1.10
Minimum speed (m/s)	2.24 ± 0.47
Average speed (m/s)	6.24 ± 0.19
Average distance (m)	180.60 ± 2.51

Table 5.2: Driving statistics for expert human driver.



(a)



(b)

Figure 5.4: Expert human driver dataset. (a) All laps colored according to speed. Breaks in trajectories around the start line on the straight are an artifact of breaking the dataset into individual laps for analysis and visualization. (b) Average trajectory along with 1-σ bound.

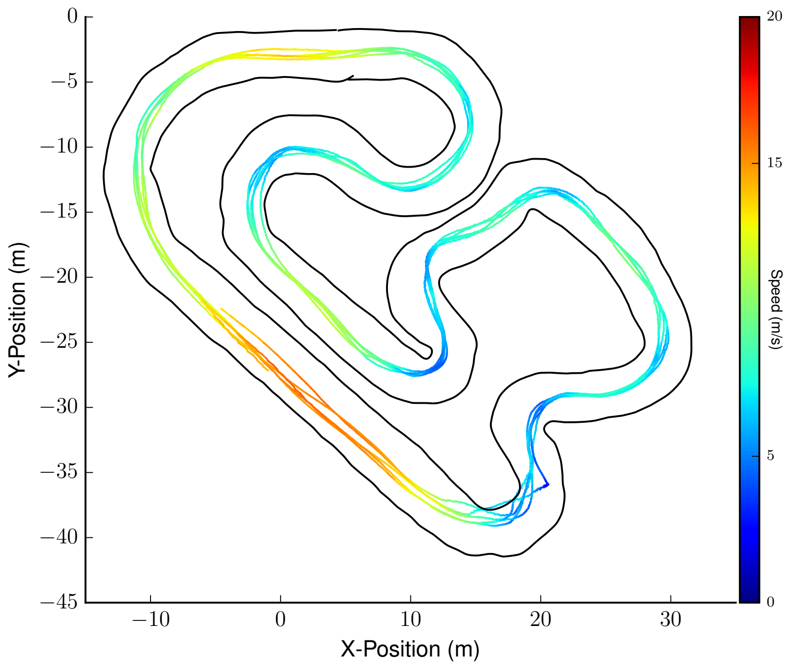


Figure 5.5: Baseline autonomous driving performance from hand tuned MPPI, five laps.

5.3.3 Baseline Autonomous Driver

The baseline autonomous driving system is the the MPPI controller that was extensively tuned by the algorithm designer over the course of a few days of testing. It is the same controller as the MPC-CFO experiments, but without any cost function adaptation. The dataset was collected and presented as the main experiments in [47]. The focus of this work was GPS-free navigation through the use of a vision-based localization system. The dataset for this paper only presented five laps of autonomous driving, but was able to achieve the fastest lap time for the MPPI controller prior to cost function optimization. In the analysis of the data, the track centerline computation was omitted due to the limited number of laps available and the consistency of the path taken by the MPPI controller from lap to lap.

Examining the path followed by the controller in Figure 5.5, the driving line is more centered through the track instead of using as much of the track width for positioning. The

Number of laps	5
Fast lap time (s)	27.96
Average lap time (s)	28.72 ± 0.72
Maximum speed (m/s)	12.15 ± 0.29
Minimum speed (m/s)	1.84
Average speed (m/s)	6.24 ± 0.19
Average distance (m)	177.18

Table 5.3: Driving statistics for hand tuned MPPI.

exceptions appear to be turns one-two and five-six, where the paths cut the turn. In one lap of the data a significant mistake was made when the entrance to turn one off the straight was initiated too early, which required corrective action to avoid hitting the inner barrier that results in a hard slide to keep driving. Overall, the autonomous driver was more consistent than the human drivers from lap to lap, but proved overall slower around the track. An interesting point is that the average path distance for MPPI is shorter than either human driver, but the slower speeds still result in longer lap times. We can see that, on average, the driver that takes the longer path around the track wins as they are able to maintain higher overall speeds that more than make up for the extra distance traveled each lap.

5.3.4 MPC-CFO Driver

The generalization capabilities of MPC-CFO were discussed previously, but because the framework specializes performance to the track conditions, vehicle configuration, and many of the other factors that effect performance on the day of testing, a dataset of CFO-MPC driving with the speed map that generated the fast lap time was not able to be collected. As a result, the driving performance analysis in this section will rely on the fast lap generated during cost function optimization. From the previous chapter, we know that the MPC-CFO driver is able to outperform the previous best hand tuned results, but there is still a performance gap to reach the best lap times of the human drivers. The two most interesting metrics to compare with the other drivers are the average and maximum speeds. MPC-CFO has a top speed lower than the skilled and expert human driver, but a higher average speed.

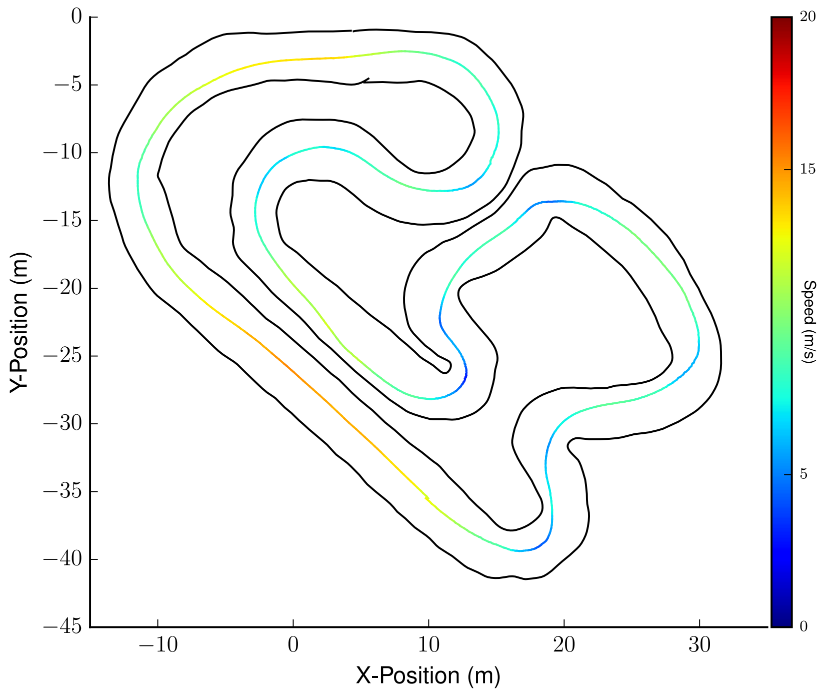


Figure 5.6: MPC-CFO fast lap.

This suggests that on the slower portions of the track, MPC-CFO is consistently driving faster than the human drivers, but then the human drivers make up for that difference in their ability to reach higher speeds on the straights. The MPC-CFO driver also has late entries into turn three, four, and five, which result require fairly aggressive steering maneuvers to get into position for the subsequent turns. The result of the late turning is slow speeds and lost time at these points on the track. Despite a few abrupt maneuvers, more of the overall track width is used than the baseline MPPI. In particular, the lines taken on turns three-four and the exit of six show significant deviations from the track centerline in pursuit of a good racing line.

Number of laps	1
Fast lap time (s)	27.68
Maximum speed (m/s)	11.45
Minimum speed (m/s)	2.27
Average speed (m/s)	6.52
Distance (m)	179.89

Table 5.4: Driving statistics for expert human driver.

5.3.5 Driver Comparison

After the performance of each driver was analyzed in isolation, we can bring the average and fast laps of each driver together for a direct comparison of their driving data. That comparison is shown in Figure 5.7. Note that for both autonomous drivers, the fast lap from each driver was used for the average trajectory figure.

The human expert fast lap took an especially short path through the track, one that was 9 m shorter than their average lap, all while having the highest top speed and average speed compared to all of the other fast laps. Interestingly, the expert human also had the lowest minimum speed of all of the fast laps, which occurs at the end of the straight in turn one where they almost overshot the track. All together, this performance resulted in a margin of 1.53 s over the next closest lap time. To date, the human expert lap time is the fastest lap that any driver has accomplished at the GT-ARF large track. In discussion after the test was over, the driver suggested that if he were able to drive at the track every weekend for a few months and adjust the configuration of the AutoRally chassis, a lap time another 0.5 s faster may be possible, but only with favorable track conditions.

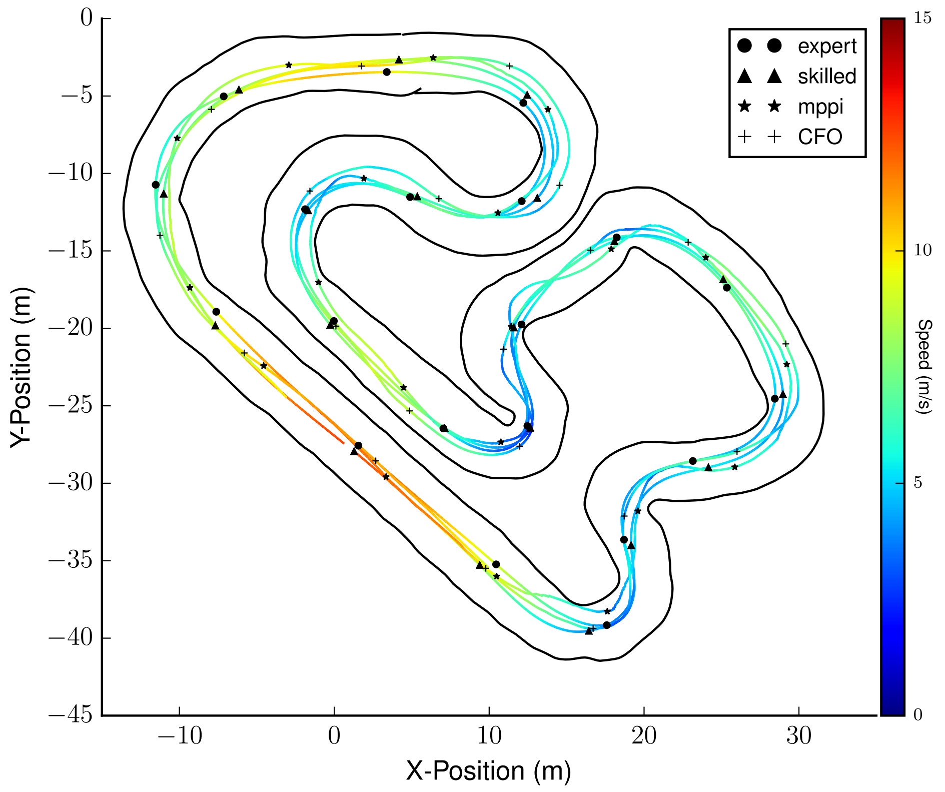
In the end, the performance gap between the human and baseline MPPI was too much for the current versions of cost function optimization to make up for in the competition. It was also shown that the track conditions on the day of testing effect the potential level of performance more than previously realized. The damp and packed conditions for the dataset collected with the human expert provided the ideal conditions the achieve fast lap times. If the amount of time it took to collect data could be reduced for the autonomous systems, it could be foreseeable to conduct another competition where driving data for all drivers is collected on the same day to manage as many environment related variables that effect driving performance.

While one expert driver provided an example for what the best level of performance may look like, an continuation of this work could be to bring more expert humans to GT-ARF for data collection. A prime candidate would be a professional racer who is currently active in the sport. A difficulty with finding drivers who have experience with driving 1/5 scale dirt track RC platforms is that almost all of the tracks and competitions in the US are on the West coast. Indeed, the expert who participated in our study currently lives in the Southwest.

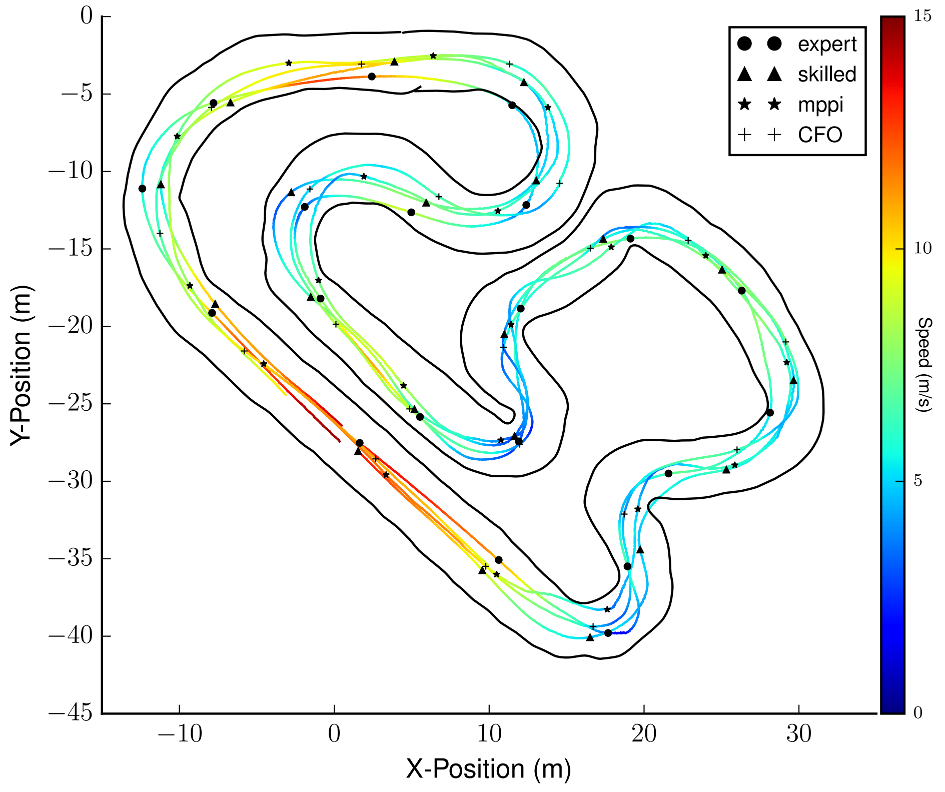
With all of the project infrastructure in place for AutoRally at Georgia Tech including the testing tracks and fleet of platforms, newer autonomous controllers may also best the performance of the current version of MPC-CFO. As designed, the framework can offer improved performance for the task of racing for any other promising MPC methods that use a cost function or other representation within their optimization that can be updated from lap to lap.

	Skilled Human	Expert Human	Baeline MPC	MPC-CFO
Lap time (s)	27.09	25.56	27.96	27.68
Maximum speed (m/s)	13.88	14.52	12.15	11.45
Minimum speed (m/s)	3.01	1.66	1.84	2.27
Average speed (m/s)	6.41	6.68	6.17	6.52
Distance (m)	173.55	171.53	177.18	179.89

Table 5.5: Fast lap comparison for all drivers.



(a)



(b)

Figure 5.7: Comparison of all drivers. (a) Average trajectory for each driver. (b) Fast lap trajectories for each driver.

CHAPTER 6
CONCLUSION

The ability to conduct experiments in the real world is a critical step for roboticists working to create autonomous systems that achieve human-level task performance. Self-driving vehicles have received significant interest in recent years, in part because of their potential societal benefit, but have not demonstrated human-level performance. The task of off-road rally racing is an especially difficult driving task where the goal is to drive through a course as quickly as possible. Many of the open challenges for self-driving vehicles occur frequently in this domain because of the need to operate the vehicle up to the dynamical limits to achieve the best performance. We chose to study autonomous rally racing as a new domain for self-driving vehicle technologies on the path to safe autonomous vehicles.

To race in the real world, a robust hardware platform is required. One potential approach is to use a scaled vehicle platform that offers a robust, cheaper alternative to full size cars. In this work, we created a scaled self-driving vehicle platform that enables us to work in the space of autonomous rally racing without the cost, safety, and space concerns that a full sized autonomous rally car would create. From the first AutoRally platform, the design has been refined and updated to withstand all of the harsh driving conditions encountered in high speed driving on dirt roads. The fleet has been expanded to six vehicles at Georgia Tech. For the first of its kind fast autonomous driving on dirt roads, a stochastic model predictive controller was brought to the AutoRally platform and tasked with driving as fast as possible with the help of extensive hand tuning from experts. In pursuit of the highest level of performance, a new framework was proposed and tested that layered an optimization scheme on top of the model predictive path integral (MPPI) controller to optimize part of the cost function through successive interactions with the system. The reward in this optimization was lap times, which allows optimize for the task, instead of the task of driving as fast as possible for the time horizon of the MPC. We demonstrated the effectiveness of this approach, named model predictive control cost function optimization (MPC-CFO), with the MPPI controller driving the AutoRally platform in simulations and in the real world at the Georgia Tech Autonomous Racing Facility. Within the MPC-CFO framework, cost function representations and stochastic optimization methods were explored in search of the fastest lap times in three different environments. An analysis of the how CFO influenced the performance of the MPPI control was performed with experimental results in a simulation environment in which the controller had perfect knowledge of the world dynamics and one in which the controller dynamics and world dynamics were dissimilar. The MPC-CFO framework was shown to outperform the previous best level of performance with a given controller with only a few interactions with a system in the real world. Two humans participated in a rally race where their time trial driving data was compared to MPC-CFO and a hand tuned version of MPPI. While the humans still outperform the autonomous drivers, the data offers a performance benchmark for future research where human performance can easily be compared to autonomous systems.

With the open source design of the AutoRally hardware and software and the initial rally competition between humans and autonomous agents, our hope is that this thesis opens the area of off road racing to study and provides an approach agnostic performance benchmark by which we can evaluate future work against.

REFERENCES

[1] D. Schaper. Human Errors Drive Growing Death Toll in Auto Crashes. Available online at `http://www.npr.org/2016/10/20/498406570/tech-human-errors-drive-growing-death-toll-in-auto-crashes` (last accessed August, 2018).

[2] K. Henry. Traffic fatalities up sharply in 2015. Available online at `https://www.nhtsa.gov/press-releases/traffic-fatalities-sharply-2015` (last accessed August, 2018).

[3] M. Bertoncello and D. Wee, "Ten ways autonomous driving could redefine the automotive world," *Retrieved from McKinsey & Company website: http://www. mckinsey. com/insights/automotive_and_assembly/ten_ways_autonomous_ driving_could_redefine_the_automot* 2015.

[4] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[5] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[6] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: Autonomous vehicles in city traffic*. Springer, 2009, vol. 56.

[7] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 1433–1440.

[8] 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," 2017.

[9] G. Williams, B. Goldfain, P. Drews, K. Saigol, J. M. Rehg, and E. A. Theodorou, "Robust sampling based model predictive control with sparse objective information," 2018.

[10] Donkey Car. Available online at `http://www.donkeycar.com/` (last accessed August, 2018).

[11] J Gonzales, F Zhang, K Li, and F Borrelli, “Autonomous drifting with onboard sensors,” in *Advanced Vehicle Control: Proceedings of the 13th International Symposium on Advanced Vehicle Control (AVEC’16), September 13-16, 2016, Munich, Germany*, CRC Press, 2016, p. 133.

[12] F1/10 Autonomous Racing Competition. Available online at `http://f1tenth.org/competition` (last accessed August, 2018).

[13] Rapid Autonomous Complex-Environment Competing Ackermann-steering Robot (RACECAR). Available online at `https://mit-racecar.github.io/` (last accessed August, 2018).

[14] N. Keivan and G. Sibley, “Realtime simulation-in-the-loop control for agile ground vehicles,” in *Conference Towards Autonomous Robotic Systems*, Springer, 2013, pp. 276–287.

[15] J. L. Jakobsen, *Autonomous drifting of a 1:5 scale model car*, 2011.

[16] S. Song, “Towards autonomous driving at the limit of friction,” 2015.

[17] M. Cutler, T. J. Walsh, and J. P. How, “Reinforcement learning with multi-fidelity simulators,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, 2014, pp. 3888–3895.

[18] D. I. Katzourakis, I. Papaefstathiou, and M. G. Lagoudakis, “An open-source scaled automobile platform for fault-tolerant electronic stability control,” *Instrumentation and Measurement, IEEE Transactions on*, vol. 59, no. 9, pp. 2303–2314, 2010.

[19] 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 *American Control Conference, 2004.*, IEEE, vol. 4, 2004, pp. 3381–3386.

[20] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[21] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, IEEE, vol. 3, pp. 2149–2154.

[22] P. A. Theodosis and J. C. Gerdes, “Generating a racing line for an autonomous racecar using professional driving techniques,” in *ASME 2011 Dynamic Systems and Control Conference and Bath/ASME Symposium on Fluid Power and Motion Control*, American Society of Mechanical Engineers, 2011, pp. 853–860.

[23] J. Funke, P. Theodosis, R. Hindiyeh, G. Stanek, K. Kritatakirana, C. Gerdes, D. Langer, M. Hernandez, B. Muller-Bessler, and B. Huhnke, "Up to the limits: Autonomous audi tts," in *Intelligent Vehicles Symposium (IV)*, IEEE, 2012, pp. 541–547.

[24] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730.

[25] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[26] J. C. Kegelman, L. K. Harbott, and J. C. Gerdes, "Insights into vehicle trajectories at the handling limits: Analysing open data from race car drivers," *Vehicle system dynamics*, vol. 55, no. 2, pp. 191–207, 2017.

[27] D. I. Katzourakis, E. Velenis, D. Abbink, R. Happee, and E. Holweg, "Race-car instrumentation for driving behavior studies," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 2, pp. 462–474, 2012.

[28] C. Thorpe and T. Kanade, "Vision and navigation for the cmu navlab," in *Mobile Robots I*, International Society for Optics and Photonics, vol. 727, 1987, pp. 261–267.

[29] C. Thorpe, M. Herbert, T. Kanade, and S. Shafer, "Toward autonomous driving: The cmu navlab," *IEEE expert*, vol. 6, no. 4, pp. 31–42, 1991.

[30] A. Kelly, "A partial analysis of the high speed autonomous navigation problem," CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, Tech. Rep., 1994.

[31] A. Stentz and M. Hebert, "A complete navigation system for goal acquisition in unknown environments," *Autonomous Robots*, vol. 2, no. 2, pp. 127–145, 1995.

[32] J. K. Rosenblatt and C. E. Thorpe, "Combining multiple goals in a behavior-based architecture," in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, IEEE, vol. 1, 1995, pp. 136–141.

[33] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, IEEE, 2006, pp. 2219–2225.

[34] J. C. Spall, "Implementation of the simultaneous perturbation algorithm for stochastic optimization," *IEEE Transactions on aerospace and electronic systems*, vol. 34, no. 3, pp. 817–823, 1998.

[35] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[36] F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, Omnipress, 2012, pp. 1547–1554.

[37] B. Goldfain, P. Drews, C. You, M. Barulic, O. Velev, P. Tsiotras, and J. M. Rehg, "Autorally: An open platform for aggressive autonomous driving," *IEEE Control Systems Magazine*, vol. 39, no. 1, pp. 26–55, 2019.

[38] AutoRally Platform Instructions. Available online at `https://github.com/AutoRally/autorally_platform_instructions` (last accessed August, 2018).

[39] 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," in *ICRA workshop on open source software*, vol. 3, 2009.

[40] AutoRally Software. Available online at `https://github.com/AutoRally/autorally` (last accessed August, 2018).

[41] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Information theoretic model predictive control: Theory and applications to autonomous driving," *arXiv preprint arXiv:1707.02342*, 2017.

[42] 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 *Proceedings of the 1st Annual Conference on Robot Learning*, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., ser. Proceedings of Machine Learning Research, vol. 78, PMLR, 2017, pp. 133–142.

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

[44] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile off-road autonomous driving using end-to-end deep imitation learning," 2018.

[45] G. Williams, B. Goldfain, P. Drews, J. M. Rehg, and E. A. Theodorou, “Best response model predictive control for agile interactions between autonomous ground vehicles,” 2018.

[46] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, “Information-theoretic model predictive control: Theory and applications to autonomous driving,” *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1603–1622, 2018.

[47] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, “Vision-based high-speed driving with a deep dynamic observer,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1564–1571, 2019.

[48] Hector Gazebo Plugins ROS Package. Available online at `http://wiki.ros.org/hector_gazebo_plugins` (last accessed August, 2018).

[49] D. Q. Mayne, “Model predictive control: Recent developments and future promise,” *Automatica*, vol. 50, no. 12, pp. 2967–2986, 2014.

[50] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, “An integrated system for real-time model predictive control of humanoid robots,” in *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, IEEE, 2013, pp. 292–299.

[51] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, “Deep blue,” *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[52] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[53] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, p. 354, 2017.

[54] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.*, “Starcraft ii: A new challenge for reinforcement learning,” *arXiv preprint arXiv:1708.04782*, 2017.

[55] K. Kritayakirana and J. C. Gerdes, “Autonomous vehicle control at the limits of handling,” *International Journal of Vehicle Autonomous Systems*, vol. 10, no. 4, pp. 271–296, 2012.

[56] Roborace. Available online at `https://roborace.com/` (last accessed August, 2018).