ADAPTIVE DRIVER MODELING USING MACHINE LEARNING ALGORITHMS FOR THE ENERGY OPTIMAL PLANNING OF VELOCITY TRAJECTORIES FOR ELECTRIC VEHICLES

AND

REALIZING SIMULTANEOUS LANE KEEPING AND ADAPTIVE SPEED REGULATION ON ACCESSIBLE MOBILE ROBOT TESTBEDS

A Thesis
Presented to
The Academic Faculty

By

Thomas R. Waters

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Georgia Institute of Technology

Georgia Institute of Technology

December 2017

ADAPTIVE DRIVER MODELING USING MACHINE LEARNING ALGORITHMS FOR THE ENERGY OPTIMAL PLANNING OF VELOCITY TRAJECTORIES FOR ELECTRIC VEHICLES

AND

REALIZING SIMULTANEOUS LANE KEEPING AND ADAPTIVE SPEED REGULATION ON ACCESSIBLE MOBILE ROBOT TESTBEDS

Approved by:

Professor Aaron Ames, Advisor
School of Mechanical and Civil Engineering
California Institute of Technology

Professor Jonathon Rogers
School of Mechanical Engineering
Georgia Institute of Technology

Professor Oliver Sawodny
Institute for System Dynamics
Universität Stuttgart

Dipl. Ing. Uli Wohlhaupter
Research and Development
Daimler AG

Date Approved: September 11, 2017

It was only about a 10-minute ride.
He said, “What are you doing out here?”
I said, “I’m an actor.”
So he said, “A lot of competition in your business.”
I said, “Yeah.”
He said, “Just like mine.”
And we kicked it around a little bit, and then he said, “Just keep in mind,
there’s always room for one more good one.”
That was very helpful.
- *Excerpt from an interview with Leonard Nimoy*
on a conversation with John F. Kennedy

To my parents: who have always supported me, regardless of how outlandish my goals may have been.

To my sister: who was never afraid to challenge me, regardless of the situation.

To my friends: who may not always have done either of those things, but were great anyways.

ACKNOWLEDGEMENTS

As a second-year participant in the Joint-Master's program between the Georgia Institute of Technology and the Universität Stuttgart, I would first and foremost like to thank the program organizers Dr. Paul Neitzel and Dr. Oliver Sawodny. Their vision and hard work made my dream of working in the German automotive industry a reality.

A great deal of gratitude is owed to my advisor on the Georgia Tech side, Dr. Aaron Ames. Thank you for taking the chance on the joint-degree program and allowing me to pursue my own research interests while giving me valuable feedback and advice when I needed it. I also thank my second Georgia Tech committee member, Dr. Jonathon Rogers. Thank you to the members of AMBER Lab for their companionship and support during my time in Atlanta with special thanks due to Matthew Powell for his ability to push me in just the right direction.

On the Stuttgart side, I would like to again thank Dr. Oliver Sawodny for participating in my committee. I also thank my Universität Stuttgart advisor Florian Morlock and my Daimler advisor, and committee member, Uli Wohlhaupter. I am extremely appreciative of the patience and kindness the both of you demonstrated with my German and for the constant, invaluable feedback on my work. My year spent at Daimler AG has been unforgettable thanks to you.

A special thanks goes to my friends Marius, Ahmed, Chris, Eui, Xiaoshu, Nadine, Benni, Caroline, Mark, Katja, and the GT Climbing Club for making graduate school one of the best experiences of my life. Spending the last two years with all of you has taught me that it is just as important to push yourself in the social aspects of your life as it is in the professional ones.

Finally, I would like express my undying gratitude to my family. To my grandmother, thank you so much for your stylistic and grammatical suggestions. To my mother, father, and sister: thank you for stoking my interests, supporting my decisions, and for just being there. I will be forever be in your debt.

PART 1:

ADAPTIVE DRIVER MODELING USING MACHINE LEARNING ALGORITHMS FOR THE ENERGY OPTIMAL PLANNING OF VELOCITY TRAJECTORIES FOR ELECTRIC VEHICLES

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Driver assistance systems show the potential to increase the fuel economy and optimize the range of standard and electric vehicles. Eco-driving focused systems optimize velocity trajectories with respect to energy consumption and suggest these optimized speeds to drivers with the goal of reducing overall energy consumption. Because the systems have no direct control over vehicle behavior, the driver's inclination to follow the commands is important to their effectiveness. This can be improved by personalizing the velocity commands to suit an individual's driving behavior, requiring a model capable of accurately predicting styles of individual drivers.

Two methods for identifying, modeling, and predicting driver behavior using driving data time-series are investigated. The first, pattern recognition-based approach breaks down the data into homogeneous segments using heuristic, dynamic programming, and bottom-up methods. Segments are grouped based on acceleration behavior and used, in conjunction with function-fit regression and system identification methods, to construct models describing driving behavior. Contrary to the first approach, the second, machine learning based method constructs a model using an entire time-series by analyzing relationships between multiple variables. Finally, each method is evaluated in it's ability to accurately predict driver acceleration and velocity behavior.

CHAPTER 1
INTRODUCTION

Since the birth of the automobile over one hundred years ago, cars have become one of the most common means of transportation on the planet. The car is a modern luxury that allows the masses to travel between destinations independently, efficiently, and comfortably. Over the last 40 years, the average car ownership in developed and developing countries has been on the rise [23].

Unfortunately, this rise in the popularity has also had several negative impacts. Examples include significant contributions from motor traffic to the air concentrations of pollutants like CO, NO_x, PM_{10}, and CO_2 [13], the destruction of land and toxic pollution in extraction areas for fossil fuels consumed primarily by the automotive industry [20], and an increasing number of human fatalities every year from automotive accidents [50].

Since the 1970's, driver assistance systems have become an increasingly popular method of addressing these problems. By supporting drivers in changing road conditions, popular systems like anti-lock braking and newer, more advanced systems (like adaptive cruise control, lane keeping, and range extension systems) can improve driving safety, comfort, and sustainability. These systems are, however, limited by the amount of information they have on the intent and style of the driver operating the vehicle. For example, a novice driver would appreciate the early intervention of a safety system whereas an experienced driver might become frustrated in the same situation where they have stable control of the vehicle [68]. Knowledge about the specific driver could be used to increase general system acceptance.

In this work, we explore adaptive driver behavior modeling methods, including pattern recognition and machine learning approaches, that could be used in conjunction with a range extension driver assistance system to increase driver acceptance and trust. First, an overview of fundamental topics such as driver assistance systems and driver behavior modeling are introduced.

1.1 Driver Assistance Systems

Driver assistance systems (DAS) aid human drivers in the act of piloting an automobile. These systems can be characterized into two main categories: conventional, which work in parallel to the driver without considering them as a part of the control

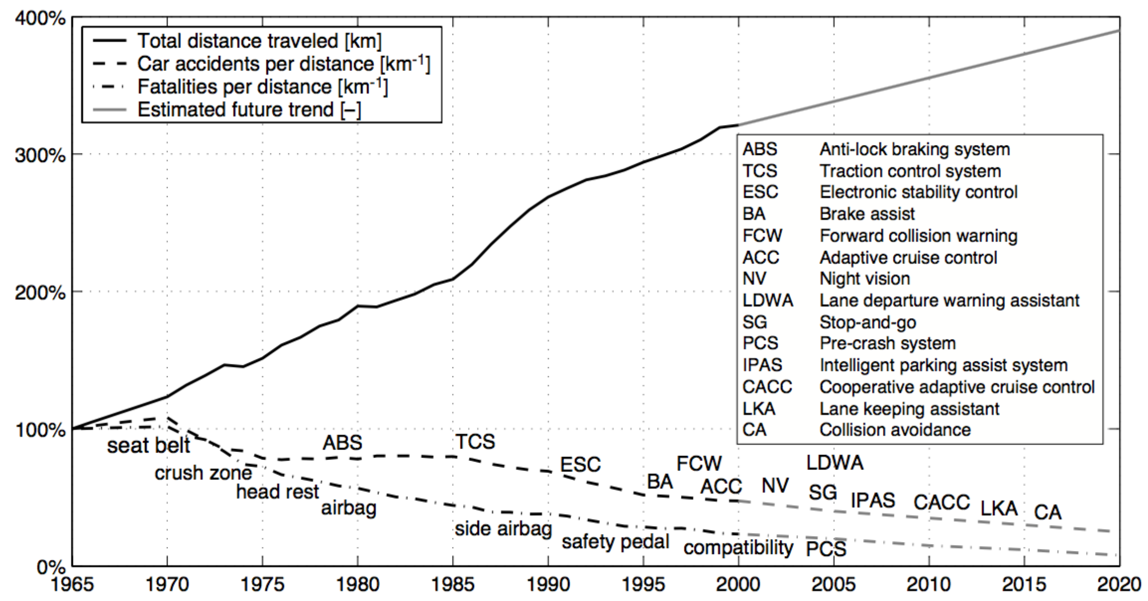


Figure 1.1: Total number of road accidents and fatalities per total distance traveled [19].

loop; and human centered adaptive systems, which directly take driver characteristics into account when making decisions [54].

In 1978, Daimler AG introduced the first production car, the Mercedes-Benz W116, to have an anti-lock braking system (ABS). This was the first conventional driver assistance system to be introduced to the market and since it's debut, other systems like electronic stability control (ESC) and traction control have followed. Their introduction, coupled with more passive safety systems like airbags and crumple zones, has led to a steady decline in automotive fatalities [10, 40]. Figure 1.1 shows the total number of road accidents per total distance traveled as a result of the introduction of passive and active safety systems [19].

Following the success of the conventional safety systems, systems focused on decreasing emissions and fuel consumption began to appear in the literature. This type of system is desirable because it uses driving style or energy management to create an optimal driving mode without the need to develop new components for the car itself. In fact, how a car is driven can have a large impact on fuel consumption and significant energy savings can be made by changing the way a driver accelerates and decelerates during a driving cycle [8, 17, 46, 58] (and references therein). Lin conducts research on the energy optimization of speed profiles for electric vehicles (EV's) using a dynamic programming approach that avoids driving at high speeds to reduce drag [43]. Karmakar took this idea a step further and optimized the energy consumption of an EV by giving speed commands to the driver [35]. Similar approaches were taken to optimize the fuel consumption of conventional cars in [49] and [44]. In a different approach, energy system management in vehicles with hybrid energy storage systems is conducted in [1].

Human centered adaptive systems are becoming more popular as well. Here, a driver model is identified and used to improve the functionality of a DAS by incorporating the style and intent of a specific driver into the system control loop. The modeling process is reviewed in Section 1.2. Within the realm of human centered systems, convenience applications are the most common. Here, the systems perform tasks for the driver like adaptive cruise control or vehicle stability control [16, 51, 63] (and references therein). Driver behavior is taken into account to confirm that system actions are necessary and make them as comfortable for the passengers as possible. Recently, driver behavior has begun to appear in eco-based driving systems, which analyze driver behavior in combination with a fuel consumption optimization algorithm [47]. Further improvements to such systems can be made by incorporating personalized driving styles into systems that seek to reduce fuel consumption by influencing driver behavior.

1.2 Driver Characteristic Recognition

As mentioned in Section 1.1, driver-specific behavior and characteristic recognition is becoming relevant in driver assistance systems. According to the survey conducted in [2], advanced driver modeling also has applications in driving training and coaching, crowd sourcing for detecting road conditions, and improving energy efficiency.

One method for driving behavior recognition is the detection of driving modes with respect to road conditions. [21, 33, 41] use a six mode classification system for detecting different road types ranging from highways to steep, rural roads. Being able to automatically detect the road type using only driving data allows the car to make adjustments to components like the suspension, brakes, and power-train that cause the vehicle to drive in a more efficient or safe manner. Driver classification based on aggressiveness is another popular modeling technique [42] (and references therein). Drivers are rated on a scale ranging from conservative to aggressive based on acceleration and braking behavior.

Personalization is another popular topic relating to the benefit of driver characteristic recognition. The idea that a consumer is more likely to buy a personalized product is a fundamental concept of business strategies like market segmentation and targeting tracing back to the 1950's [37]. This same logic can also be applied to DAS's. Systems that tailor their actions to fit an individual's driving style are more likely to be accepted than those that act against a driver's intent or judgment.

There are various methods that can be used to identify individual driving behavior, with the majority of the methods in the literature using velocity and acceleration data to classify exactly how a driver behaves. The most basic method uses a heuristic approach to identify key features of the driving data. In [48], individual sections of a velocity trajectory are defined as the points in time between stationary periods. Then, values like the maximum acceleration, acceleration time, peak velocity, average velocity, etc. are identified and used in conjunction with principle component analysis to find similarities between the different trajectories. Methods like this have the advantage of being simple to use, but are often too broad or simplistic to identify unique features of the velocity data due to their use of thresholds. If, for example, there exists an acceleration mode with a single value less than the threshold, this algorithm will fail to identify it.

Statistical methods offer a more intelligent solution. Fuzzy logic clustering methods are present in the literature [14] and use statistically optimal measures to group similar data into clusters. This type of pattern recognition is unique, however, in the way that the boundaries between the clusters are fuzzy rather than well defined, which allows the technique to be used on data that is not distinctly different [7]. In this approach, an objective function based on the mean, variance, and standard deviation is minimized to group points of multivariate data into unique segments [27, 28] (and references therein). A statistical approach has the advantage over the heuristic approach because the algorithm creates segments of data that are optimized to be as similar as the cost function allows, theoretically eliminating the threshold problem described above.

Several statistical approaches are applied specifically in this work for the identification of acceleration phases during driving. Characteristics are taken from this information to build a model using methods that will be discussed in detail in Chapter 3.

1.3 Machine Learning

Over the past three decades, the phrase "Machine Learning" has, alongside other conversation starters like "Big Data" and "The Internet of Things", become a buzzword. Machine Learning is a method for pattern recognition with two main modern applications: data classification and regression. Classification is identifying similarities in data and grouping them based on these similarities whereas regression is recognizing patterns in the relationships between variables. Achieving this pattern recognition involves two steps: training and testing. One feeds the algorithm training data with known characteristics so that the algorithm can learn the patterns and relationships between the variables. Then, once the training phase is completed, the trained algorithm can be used to recognize similar patterns or relationships in a test data set with unknown characteristics.

The early roots of Machine Learning began with the classification problem and stem from discoveries like Markov Chains and Rosenblatt's Perceptron in 1957 [56], which took an idea from neurophysiology and described it in a way that could be used by computers. Simply put, Perceptron uses a linear method to separate two types of data from one another and according to Vapnik, this, in conjunction with the proof of Perceptron Theorem by Novikoff in 1962, was the true birth of the field of machine learning [61]. Neural Networks, which are the combination of multiple Perceptrons, were the next logical step.

Today, Machine Learning is being used in a variety of ways with numerous algorithms. Decision tree learning [32], artificial neural networks [57] [38], and support vector machines [61] are examples of the more fundamental learning algorithms. Smola and Vishwanathan [59] give an excellent summary of the modern applications of machine learning, including topics like page ranking for search engines [55], collaborative filtering for sites like Netflix or Amazon [5], and also for automatic language translation [66].

In Chapter 4, machine learning will be used for the prediction of velocity trajectories. A discussion of the methods and the specific algorithms used will be presented at the beginning of that chapter.

1.4 Outline

Driver assistance systems that optimize the fuel consumption and extend the range of vehicles are presented in [35, 44, 49, 47] and references therein. These systems use road data like curvature and grade to create an energy-optimal driving trajectory that is proposed to the driver via suggested speed commands. Unlike the safety systems presented in Section 1.1, these systems have no direct control over the vehicle and their performance is entirely dependent on the driver's ability and desire to follow the commands. Therefore, this work investigates individual driver behavior identification that can be used in conjunction with these systems to personalize velocity suggestions and increase driver acceptance of the system.

This work has two approaches with respect to recognizing and modeling driver behavior. Given a test driving data set, the first uses a pattern recognition algorithm to break the data down into homogeneous segments, and classification to group segments based on acceleration behaviors. Then, both a function fit regression and a system identification approach are used to build models from the classified groups and predict future behaviors. Unlike the first method, the second, machine learning based approach, utilizes the entire driving cycle to automatically recognize behaviors and generate a function capable of predicting future ones. Because driver model accuracy is absolutely vital to the success of a potential driver assistance system, the results of both modeling approaches will be evaluated and by comparing model-predicted data with real-world driving data.

Chapter 2 presents background information necessary for the calculation and evaluation of the driving modeling methods. First, we discuss a vehicle model used to calculate the energy consumption of existing and modeled velocity trajectories as well as the fundamentals of regressions and system identification to be used in Chapter 3.

Chapter 3 presents the pattern recognition based driving behavior identification method. Section 3.1 discusses the pattern recognition and data classification approaches including one heuristic and three statistical techniques. In Sections 3.2 and 3.3, we present the use of the classified data to build two models using function fit regression and system identification approaches, respectively. Both models are used to predict driving behavior and evaluated in their ability to recreate real velocity trajectories in Section 3.4.

Chapter 4 presents velocity trajectory prediction using a machine learning approach. Section 4.1 discusses the machine learning algorithm used in this work: support vector machines. In Section 4.2 we present and compare the results of several machine learning trials against real-world test data.

In Chapter 5, the contributions and results of this work are summarized, conclusions are discussed, and possibilities for future work are presented.

CHAPTER 2
BACKGROUND

This chapter presents and develops three fundamental topics utilized in this work, leaving space in the following chapters to discuss the methods and results. First, a vehicle model, used in Chapters 3 and 4 to calculate the energy consumption of an electric vehicle over defined velocity trajectories is presented. Second, the fundamentals of a constrained, nonlinear regression, used in Section 3.2 for the identification of driving behaviors, are presented and discussed. Finally, an introduction to system identification, as used in Section 3.3, is discussed.

2.1 Vehicle Model

The passenger vehicle considered in this work is the $250e$ electric B-Class produced by Daimler AG and has a completely electric power train composed of an electric motor (EM), gearbox, and Lithium-ion battery as shown in Figure 2.1. The EM is capable of generating $132kW$ and the batteries have a maximum storage capacity of $28kWh$. Because the gearbox is single speed and engaged at all times, any power losses due to shifting or clutch slip will not be considered here.

A quasi-static longitudinal vehicle model, as first proposed in [22], will be used to model the resistance forces acting on the vehicle during driving. Any lateral dynamics or effects are not considered because the scope of this work focuses only on the energy consumption. Road and vehicle data are recorded via the on board CAN data system and has been provided for this thesis by the Daimler AG. Vehicle velocity $v(t)$, speed limit $v_{\mathrm{SL}}(t)$, and road grade $\gamma(t)$ as functions of time are given from the CAN.

2.1.1 Longitudinal Vehicle Model

As presented in [47] and shown further in Figure 2.1, the power train of an electric vehicle consists of four main components: the battery pack, auxiliary electrical components, electric motor, and gearbox. Because the electrical vehicle considered in this work has only a single speed transmission, a speed and torque dependent loss map is used to account for any losses that occur. Taking these losses into account, the

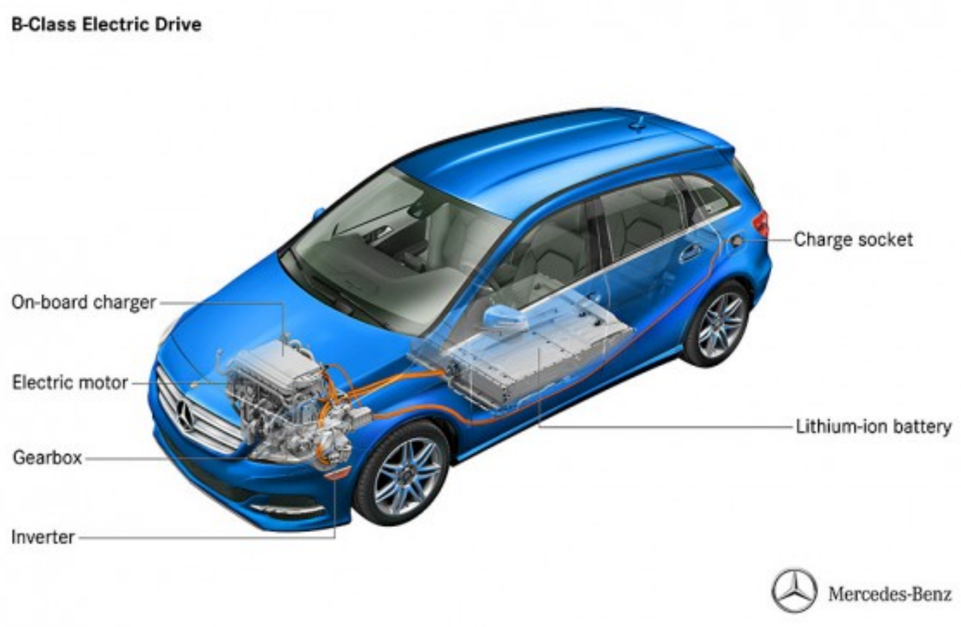


Figure 2.1: Power train diagram displaying the different components in the Mercedes 250e electric B-Class [26].

torque required to propel the electric vehicle is:

$$T_{\text{whl}} = (T_{\text{EM}} \cdot i_{\text{STU}} + T_{\text{loss,STU}}) + T_{\text{brk}}, \tag{2.1}$$

where T_{EM} is the torque generated by the electric motor, i_{STU} is the fixed transmission gear ratio, $T_{\text{loss,STU}}$ are the losses that occur within the transmission unit, and T_{brk} is any torque applied by the disk brakes present on the vehicle. It is important to note that the value of the losses can be either positive or negative depending on the value of the electric motor torque, T_{EM}. Positive values would indicate that the EM is acting as a power source and negative ones would imply that it is instead recovering energy.

In addition to the torque generated by the electric motor, there are several resistance torques acting on the vehicle during operation including rolling resistance T_{roll}, aerodynamic drag T_{air}, and the resistance due to gravitational forces on an incline T_{g}. These torques are calculated using:

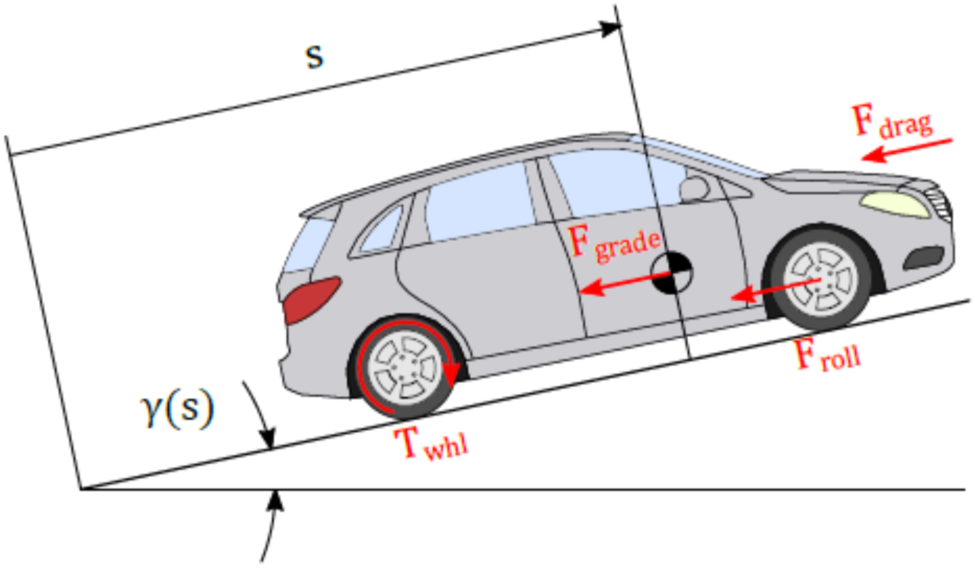


Figure 2.2: The longitudinal vehicle model showing propulsion moment and resistance forces acting on the vehicle [47].

$$T_{\text{res}} = T_{\text{roll}} + T_{\text{air}} + T_{\text{g}} + T_{\text{acc}}, \tag{2.2}$$

$$T_{\text{roll}} = -r_{\text{whl}} c_r m g \cos\gamma(t) = -r_{\text{whl}} F_{\text{roll}}, \tag{2.3}$$

$$T_{\text{air}} = -\frac{1}{2}\rho_{\text{air}} A_f c_w * v(t)^2 = -r_{\text{whl}} F_{\text{air}}, \tag{2.4}$$

$$T_{\text{g}} = -r_{\text{whl}} m g \sin\gamma(t) = -r_{\text{whl}} F_{\text{g}} \tag{2.5}$$

where r_{whl} is the vehicle wheel radius, c_{r} is the coefficient of rolling friction, m is the vehicle mass, g is the gravitational acceleration constant, ρ_{air} is the ambient air density, A_{f} is the vehicle frontal area, $\gamma(t)$ is the road grade as a function of time, c_{w} is the drag coefficient, and $v(t)$ is the velocity. These forces are shown acting on the vehicle in Figure 2.2.

This gives us the vehicle longitudinal equation of motion:

$$\frac{d}{dt}v = a = \frac{T_{\text{whl}} - T_{\text{res}}}{m_{\text{eff}} r_{\text{whl}}} \tag{2.6}$$

where m_{eff} is the effective mass of the vehicle calculated by taking the inertia of spinning drive train components, J_{eff}, into account.

$$m_{\text{eff}} = m + \frac{J_{\text{eff}}}{r_{\text{whl}}^2} \tag{2.7}$$

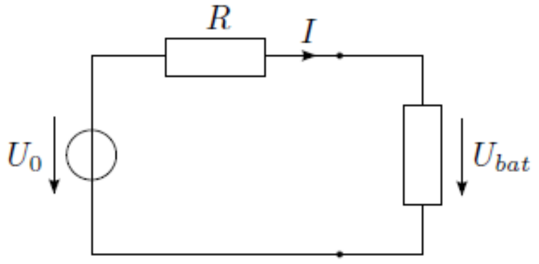


Figure 2.3: Equivalent circuit model of the lithium-ion battery [25].

From this equation of motion, we calculate the total power needed to propel the car by calculating the power required to overcome the resistive forces as a function of velocity:

$$P_{\text{res}} = v * (F_{\text{air}} + F_{\text{roll}} + F_{\text{g}}), \tag{2.8}$$

with the addition of the power needed to propel the car with a certain acceleration:

$$P_{\text{tot}} = P_{\text{res}} + \frac{1}{2}m\frac{\Delta v}{\Delta t} = P_{\text{res}} + P_{\text{acc}}, \tag{2.9}$$

where Δv is the change in velocity over time step Δt. Using this total power, in combination with a lookup table, we can calculate the torque required by the electric motor to propel the vehicle as a function of velocity and acceleration. Using the model presented in Section 2.1.2, we can then calculate the energy consumed from this torque. Equation (2.10) presents the calculation of the EM torque, where ω_{EM} is the angular velocity of the electric motor calculated using $\omega_{\text{EM}} = \frac{v i_{\text{ratio}}}{r_{\text{whl}}}$. The fixed gear ratio of the power transmission unit is represented as i_{ratio}.

$$T_{\text{EM}} = \max\left(\frac{P_{\text{tot}}}{\omega_{\text{EM}}}, T_{\text{EM,table}}(\omega_{\text{EM}})\right) \tag{2.10}$$

2.1.2 Battery and Electric Motor Model

The lithium-ion battery present in the B-Class is modeled according to [25] using the equivalent circuit model shown in Figure 2.3. Here, U_{o} is the open circuit voltage, and R_{i} is the battery resistance used to account for any losses that occur during charging and discharging. Using these terms, we can calculate the outgoing current from the battery as a function of the electric power demand P_{el}. Also note that temperature effects on the discharging efficiency of the battery are also disregarded, as they fall beyond the scope of this work.

$$I_{\text{bat}} = \frac{U_{\text{o}} - \sqrt{U_{\text{o}}^2 - 4R_i P_{\text{el}}}}{2R_{\text{i}}}, \tag{2.11}$$

$$P_{\text{bat}} = I_{\text{bat}} U_{\text{o}} \tag{2.12}$$

The total electric power demand on the battery is defined as the power required to maintain the driving torque calculated in Section 2.1.1 summed with the auxiliary vehicle power consumption. The power demand on the battery, $P_{\text{EM,el}}$, is calculated from a two-dimensional look up table as a function of motor speed and torque. The auxiliary power is used to power secondary systems on the car like battery cooling, passenger cabin heating and cooling, information display and will be considered constant.

$$P_{\text{el}} = P_{\text{EM,el}} + P_{\text{aux}} \tag{2.13}$$

Using P_{el} from Equation (2.13) to calculate the equivalent discharging current produced by the battery, we calculate the total power consumption of the battery as a function of torque demand using (2.12). Energy consumption follows naturally from power, as energy is power multiplied by the time duration in which the power was produced. Here, the time duration will be the same Δt defined in Equation (2.14).

$$E_{\text{bat}} = P_{\text{bat}} \Delta t \tag{2.14}$$

This section has defined both a vehicle and battery model that, in conjunction with vehicle data like velocity, acceleration, and road grade, can be used to calculate energy consumption as a function of time. This model is used further in Chapter 3 to quantify the energy efficiency of different driver behavior modeling methods.

2.2 Regression and Curve Fitting

In the field of mathematics, it is often desirable to obtain a inter-variable relationships framed as the dependence of one variable on the other. In some cases the dependence is obvious like, in a positive correlation example, that higher fuel consumption is directly correlated to high engine RPMs or that, in a negative one, high spending leads to lower bank account balances. One does not need to resort to mathematics in order to understand the basic trend in these examples but there are many cases where the relationship between variables is not clear. One would therefore like to find and accurately represent the correlation mathematically for prediction purposes. Examples of this include abstract relationships like weather patterns in a large city or, in this work, the relationship between velocity and acceleration behavior while driving.

Regressions have exactly this purpose: modeling and understanding the relationship between variables to predict the outcome when only one variable is known. Over the last several decades, methods have been developed to perform regressions over multi-variable relationships that display linear or nonlinear behavior. In this section, we will present the fundamentals behind, first, a simple linear regression and then a more complex, bounded, nonlinear regression between two variables that will be utilized later in this work.

2.2.1 Introduction to Least Squares Linear Regression

A regression is an equation that explains the relationship between a random variable Y, referred to here as the response, and a quantity X, the predictor, that is variable but not necessarily randomly variable [15]. In the most basic form, the regression equation takes a linear form. While it is true that many relationships exhibited in nature do not display linear correlations, linear regressions are useful for characterizing general relationships with an extremely low computational cost. A linear regression line between the variables can be written as:

$$Y = \beta_0 + \beta_1 X + \epsilon. \tag{2.15}$$

For a given predictor X, there exists a corresponding response Y with the value of $\beta_0 + \beta_1 X$ plus a small variation ϵ.

From Equation (2.15), there are three unknowns: β_0, β_1, and ϵ. Because ϵ is the error between each individual response data point and the regression line, there is a unique ϵ value for each value of Y, making it difficult to solve. Thankfully, β_0 and β_1 are constants in this equation and are therefore, straightforward to solve for by analyzing each predictor-response pair. To begin the solution process, we will define estimation variables for the response variables, $\hat{Y}$, and the regression coefficients, $\hat{\beta}_0$ and $\hat{\beta}_1$.

$$\hat{Y} = \hat{\beta}_1 X + \hat{\beta}_0 \tag{2.16}$$

To calculate our estimated values, we use the least squares function form of (2.15), where the estimated regression coefficients can be found such that (2.17) is minimized.

$$S = \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_i)^2, \tag{2.17}$$

This is done in the linear case by taking the derivative of (2.17) with respect to both of the estimated regression coefficients and solving for them algebraically, resulting in:

$$\frac{\partial S}{\partial b_0} = -2 \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_i) \tag{2.18}$$

$$\frac{\partial S}{\partial b_1} = -2 \sum_{i=1}^{n} X_i (Y_i - \beta_0 - \beta_1 X_i) \tag{2.19}$$

$$\tag{2.20}$$

By solving these two equations simultaneously, we achieve the results:

$$b_1 = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sum (X_i - \bar{X})^2} \tag{2.21}$$

$$b_0 = \bar{Y} - b_1 \bar{X} \tag{2.22}$$

There are several methods available for evaluating the accuracy of a regression equation. For the purposes of this work, we will focus on the residual sum of squares: the sum of the squared residual errors between the predicted regression line formed by the coefficients found in (2.22), $\hat{Y}_i$, and the actual predictor data Y_i. A low residual error implies that the regression line is a good fit to the training data and a good indicator that the fit will be able to accurately predict response data in the future. We now move forward to the more complex, nonlinear least squares regression utilized in Chapter 3.

2.2.2 Constrained Nonlinear Regression Problems

Similarly to the linear least squares regression problem, a nonlinear regression problem seeks to minimize the residual errors between the regression function and the response data. The two-variable, nonlinear, constrained case problem considered in this work takes the form:

$$\min_x \|f(x) - y\|_2^2 = \min_x \sum_i (f(x_i) - y_i)^2 \tag{2.23}$$

$$\text{s.t. } [x_l]_i \leq [x]_i \leq [x_u]_i \quad \text{for } i = 1...n \tag{2.24}$$

where the residual error is defined as the difference between the nonlinear fit function f evaluated at point x_i and the value of the response variable y_i, $[x_{\mathrm{l}}]_i$ is the i'th lower bound on $[x]_i$, $[x_{\mathrm{u}}]_i$ is the i'th upper bound $[x]_i$, and n is the total number of optimization variables in the vector x. Of the numerous methods that exist to solve the constrained, nonlinear least optimization problems, we will consider the "Trust Region Reflexive" algorithm used by the Optimization Toolbox and the *lsqcurvefit* function in Matlab.

In order to explain the more complex, constrained optimization problem, we will begin by discussing the unconstrained version of the basic trust region algorithm as defined in [11]. The trust region algorithm is an iterative method to solve an objective function minimization problem, in which we seek the local or even global minimum for some function $f(x)$ shown in (2.25). The algorithm starts by defining an initial point on the objective function for which the value and possibly the slope and curvature are known. We then define an arbitrary space around that point for which a model of the objective function can be calculated. The shape/radius of this space is unimportant so long as it is a "good" representation of the objective function in this region. This region is commonly referred to as the "Trust Region" because it is a window in which we can trust our model to accurately represent the objective function.

$$\min_x f(x) \tag{2.25}$$

Once the space is defined, we calculate a step away from the initial guess that minimizes the model in that region while also maintaining a position inside. Then, the objective function is evaluated at this new point and compared with the value of the model. If the model and objective function exhibit similar reductions from the initial guess, we accept the new point and iterate further using this algorithm until the local minimum of the objective function is found. If the model and objective function disagree, then we must accept that our model is not trustworthy in this region, reject the new point, reduce the radius of our "Trust Region" and start the algorithm over. A formal explanation of this algorithm follows.

- **Initialization**. Definition of initial point x_0 and the algorithm parameters, η_1,

η_2, γ_1, and γ_2, subject to the constraints: $0 \leq \eta_1 \leq \eta_2 < 1$ and $0 \leq \gamma_1 \leq \gamma_2 < 1$ Set k equal to 0.

- **Step 1: Model Definition.** Define Trust Region with radius Δ_k for iteration k such that: $\mathcal{B}_k = \{x \in \mathbb{R}^n | \|x - x_k\|_k \leq \Delta_k\}$ and define the model m_k within that region.

- **Step 2: Step Calculation and Definition.** Compute a step s_k that reduces model "sufficiently" while satisfying $x_k + s_k \in \mathcal{B}_k$.

- **Step 3: Acceptance of trial point.** Compute $f(x_k + s_k)$ and define

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}. \tag{2.26}$$

If $\rho_k \geq \eta_1$, then accept trial point and define $x_{k+1} = x_k + s_k$. Otherwise $x_{k+1} = x_k$.

- **Step 4: Update Trust Region.** Set the new radius to be

$$\Delta_{k+1} = \begin{cases} [\Delta_k, \infty) & if\ \rho_k \geq \eta_2 \\ [\gamma_2 \Delta_k, \Delta_k] & if\ \rho_k \in [\eta_1, \eta_2) \\ [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & if\ \rho_k \leq \eta_1. \end{cases} \tag{2.27}$$

Increment the value of k by 1 and repeat sequence starting at Step 1.

The selection of the step size and direction s_k is found by following the steepest descent within our Trust Region. This method is achieved by calculating the minimum of the model along the *Cauchy Arc*, whose definition can be found in [11]. In a situation where we want to solve a constrained nonlinear optimization problem, like the one discussed in this work, we solve a slightly different version of (2.25) shown in (2.29). Here the upper and lower bounds on the optimization variable x form the closed set $\mathcal{C}$.

$$\min_x f(x) \tag{2.28}$$

$$\text{s.t. } [x_\text{l}]_i \leq [x]_i \leq [x_\text{u}]_i \quad \text{for } i = 1...n \tag{2.29}$$

In order to solve the constrained optimization problem, we can no longer assume that the step calculated by the *Cauchy Arc* steepest descent method will guarantee that the solution stays within the set $\mathcal{C}$. Therefore, we adjust our "sufficient reduction" definition as can be found in Chapter 12 of [11]. Step 1 is changed to "Define a model on the objective function in the set $\mathcal{C} \cap \mathcal{B}_k$" and step 2 becomes "Compute a step s_k that reduces model "sufficiently" (according to the adjusted definition) $x_k + s_k \in \mathcal{C} \cap \mathcal{B}_k$". The remaining steps are the same as in the unconstrained case.

2.3 System Identification

System identification and modeling is a fundamental tool in scientific fields like engineering, biology, and physics. The theory is based on the idea that any system can be modeled mathematically within a certain degree of accuracy by observing the system's behavior, conducting an experiment on the system to induce a certain response, measuring that response, and devising a model that exhibits a similar one. In regard to this work, we use system identification to model driver acceleration behavior in Section 3.3. This section presents brief introduction and background on the fundamentals of system identification.

One method often used in the field of system identification is the measurement and modeling of a system's step response. As shown in Figure 2.4, a system's step response is the reaction of the output to an input of a step function. The response possibilities vary as much as dynamic systems themselves with some of the most common response types including over-damped, under-damped, and critically-damped. In the field of linear controls, a transfer function is the most common way to mathematically describe a systems dynamic response to a step input.

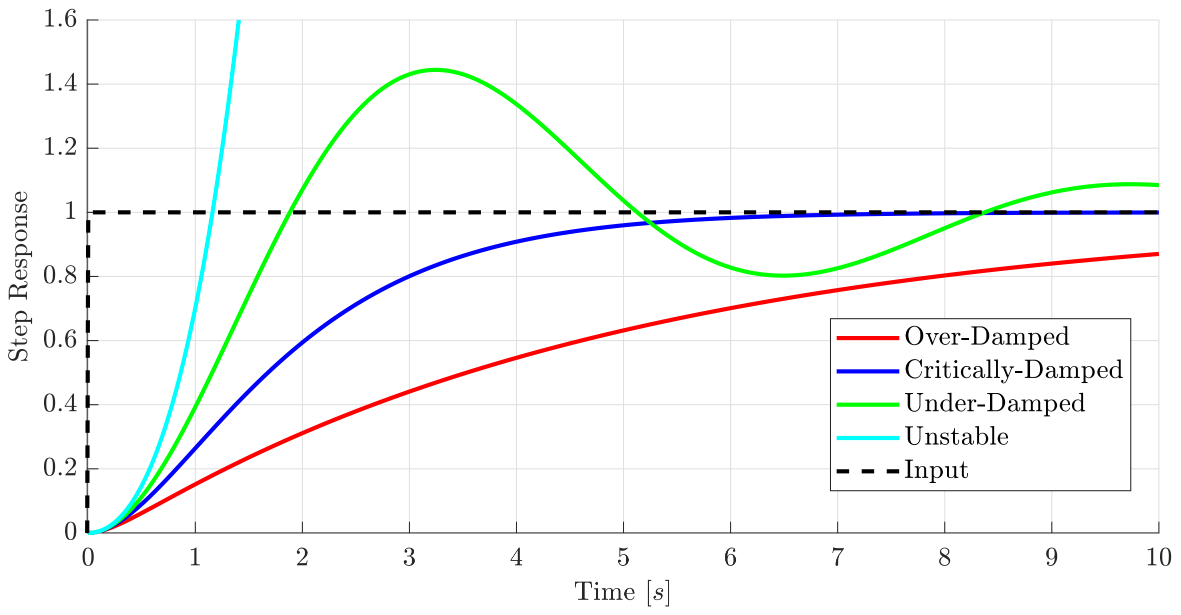


Figure 2.4: Examples of over-damped, critically-damped, under-damped, and unstable second order step responses.

Stemming from the field of classical controls, the transfer function describes the response of a system to some input $u(t)$ in the Laplace domain. The transformation to the Laplace domain, or the Laplace transform, is a method used to analyze linear, time-invariant systems and their behavior. The Laplace transformation of a real time function, $f(t)$, on the interval $[0, \infty]$ is defined as:

$$F(s) = \int_0^\infty f(t)e^{-st}dt, \tag{2.30}$$

or,

$$F(s) = \mathcal{L}[f(t)], \tag{2.31}$$

where s is the Laplace operator, $\mathcal{L}$ is the symbol for the Laplace transform, and $f(t)$ is the original, real-time function. Derived from the definition of the Laplace transformation, there also exists an inverse Laplace transformation $\mathcal{L}^{-1}[F(s)]$:

$$f(t) = \frac{1}{2\pi j}\int_{c-j\infty}^{c+j\infty} F(s)e^{st}dt, \tag{2.32}$$

where c is a large positive number, and j is the imaginary operator. As previously stated, a common equation for the system input, $u(t)$, is the step function, defined to be:

$$f(t) = \begin{cases} U & \text{for } t \geq 0 \\ 0 & \text{for } t < 0 \end{cases} \tag{2.33}$$

The step magnitude is often takes the value of 1, giving the unit step function $1(t)$. The Laplace transformation of the step function is shown in (2.34).

$$\mathcal{L}[1(t)] = \int_0^\infty 1(t)e^{-st}dt = \int_0^\infty e^{-st}dt = \frac{1}{s} \tag{2.34}$$

The Laplace transformation has the following properties:

- Linearity: $\mathcal{L}[c_1 f_t(t) = c_2 f_c(t)] = c_1\mathcal{L}[f_1(t)] + c_2\mathcal{L}[f_2(t)] = c_1F_1(s) + c_2F_2(s)$
- Differential Property: $\mathcal{L}[\frac{df(t)}{dt}] = sF(s) - f(0)$
- Final Value Theorem: $f(\infty) = \lim_{t\to\infty} f(t) = \lim s \to 0F(s)$

The primary benefit of the Laplace transformation is that differentiation and integration in the time domain become multiplication and division in the Laplace domain. Therefore, the complex differential equations representing systems can be transformed into the Laplace domain, solved easily as a polynomial, and transformed back with varying degrees of difficulty. For example, a linear dynamic system can take the form:

$$\begin{multline} y^{(n)}(t) + a_1y^{(n-1)}(t) + \cdots + a_{n-1}y^{(1)}(t) + a_ny(t) = \\ b_1u^{(n-1)}(t) + b_2u^{(n-2)}(t) + \cdots + b_{n-1}u^{(1)}(t) + b_nu(t), \end{multline} \tag{2.35}$$

where $y(t)$ and $u(t)$ are the system outputs and inputs, respectively, and $y^{(i)}(t)$ and $u^{(i)}(t)$ are the ith derivative of $y(t)$ and $u(t)$. The Laplace transformation of the system is shown in (2.36).

$$(s^n + a_1s^{n-1} + \cdots + a_{n-1}s + a_n)Y(s) = (b_1s^{n-1} + b_2s^{n-2} + \cdots + b_{n-1}s + b_n)U(s) \tag{2.36}$$

The transfer function of the system is defined as the ratio of the output variable to the input:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s^n + a_1s^{n-1} + \cdots + a_{n-1}s + a_n}{b_1s^{n-1} + b_2s^{n-2} + \cdots + b_{n-1}s + b_n}, \tag{2.37}$$

but can also be written as:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K(T_{n+1}s+1)(T_{n+2}s+1)\dots(T_{2n-1}s+1)}{(T_1s+1)(T_2s+1)\dots(T_ns+1)}, \tag{2.38}$$

where K is the ratio $\frac{a_n}{b_n}$ and the T_i's are known as time constants. In this form, we can define the stability of the system as a function of the values of the time constants. Positive values indicate that the system will exhibit a stable response to an input as shown in the over-, critically, and under-damped cases in Figure 2.4 and negative ones lead to unstable behavior. The application of transfer functions in the Laplace domain to the identification and modeling of driver acceleration behavior is presented in Chapter 3.

CHAPTER 3
MODELING THROUGH PATTERN RECOGNITION

Identifying driving behavior using a pattern recognition is a data analysis problem combined with a modeling problem. First, one must analyze a set of data by classifying the different behaviors that occur and, once this has been accomplished, use this classified data to build a model that describes the behavior. After model construction, it can be evaluated in it's ability to predict similar, future behaviors.

Data classification is achieved in this work using a single heuristic method and three statistical time-series segmentation approaches. These methods are compared and the most effective is subjected to a post processing algorithm to improve the results. Finally, models are constructed from the classified test-data set using two different methods: function fitting via a regression and system identification.

In the final section of the chapter, we test the potential effectiveness of each method to a driver assistance system by building models and testing them using real-world driving data. Training data sets are used to build each respective model and both are tested on their ability to accurately predict the driving behavior present in a final, testing data set.

3.1 Data Classification Methods

Data classification is the identification and grouping of similar patterns in a set of data. Given an unclassified data-set, one applies the data classification algorithm and receives groups containing similar segments. In this work, segments are defined as groups of adjacent data points with homogeneous characteristics. A model can be constructed using common behaviors present in the groups of segments. Figure 3.1 shows a simple classification example that identifies segments with positive, negative, and zero valued slopes. This type of classification has applications in fields ranging from data mining of medical data to human interaction classification on social network sites [3].

While classification has many applications in the field of data analysis, we focus on the classification of time-series data, as the data we use to classify driver acceleration behavior is given as a function of time. In order to achieve classification of a times-series, several different approaches can be utilized. One can, for example, use a heuristic method, where groups are first defined based on desired characteristics

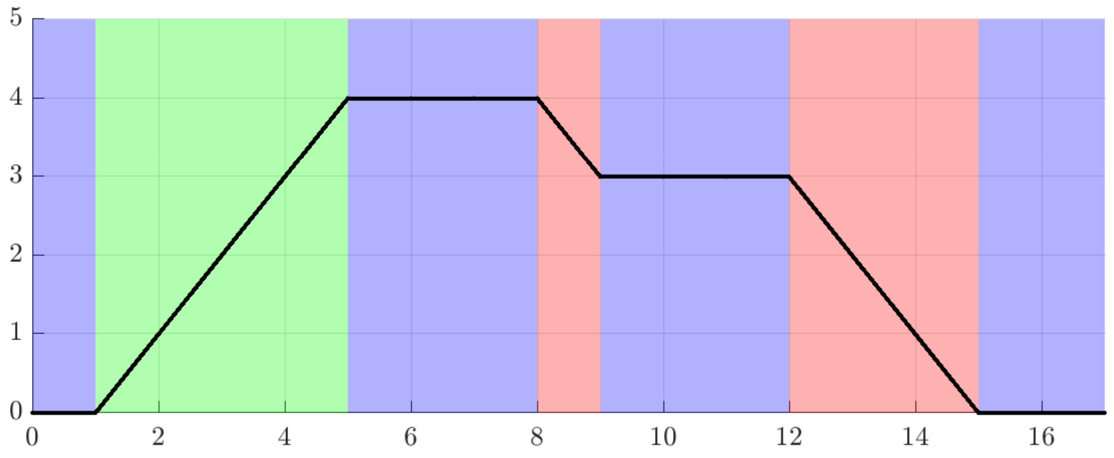


Figure 3.1: Example of a simple data classification problem.

and place data segments with corresponding properties of those groups or one can perform a segmentation on the time-series by statistically measuring the similarity of neighboring points and then forming based on these measurements.

In this section, one heuristic and three different statistical methods are discussed. A description of each algorithm and a unique set of results is presented using an example data set. Following the presentation of the different methods, we choose the most effective and discuss three potential post-processing methods for improving the classification results further.

3.1.1 Heuristic Method

The simplest data classification method, and thus also the most widely used, is the heuristic method. This algorithm begins by forming rudimentary segments from adjacent points in the time-series using a basic characteristic like, for example, positive or negative slope. Then, unique classification groups are defined based on the desired properties. Following this initial segmentation, segments are edited and placed in the predefined classification groups with corresponding characteristics. The detailed steps are shown in Algorithm 1.

Example. Here we apply the heuristic classification method to an example velocity trajectory. Because the goal of this work is the identification of driver acceleration behaviors, we form the initial segments based on any change in the sign of the derivative: acceleration. Segments are created from adjacent points with positive, negative,

```
Data: Time-Series Data, $T(t)$
Result: Heuristic Classification Method Segments
Initialize: Desired Classification Group Parameters: $p_1 ... p_n$ for groups $G_1 ... G_n$
Form: Rudimentary Segments formed based on single defining characteristic
(change in sign, slope, etc.)
for 1 to end($T(t)$) do
    Check if point belongs to a group
    if point satisfies conditions $p_1$ then
        place point in group $G_1$
    else if point satisfies conditions $p_2$ then
        place point in group $G_2$
    ⋮
    else if point satisfies conditions $p_n$ then
        place point in group $G_n$
    end
end
Reform: Edited segments from adjacent points in same groups.
```

Algorithm 1: Heuristic Classification Method

Table 3.1: Heuristic Velocity Classification Group Parameters

Characteristic	Constant v	Acceleration	Braking
1	$\lvert a\rvert < 0.3\frac{m}{s^2}$	$a > 0$	$a < 0$
2	$v > 30\frac{km}{h}$	$a \geq 0.3\frac{m}{s^2}$	$a \leq -0.3\frac{m}{s^2}$

or constant acceleration values. Figure 3.2 shows the raw velocity data and the initial segmentation.

After completing the initial segmentation, we run a loop over all points in each of the segments and check if the properties of each individual point correspond to one of the groups defined in Table 3.1. During this process, the segment lengths themselves are edited so that only points that belong to a certain group are placed in that group. The table properties check if the trajectory is in an acceleration, braking, or constant velocity phase and the constant velocity threshold, $a_{\text{const,thresh}}$, was chosen to be $0.3\frac{m}{s^2}$ to cut out negligible noise in the data that would otherwise imply the driver is accelerating.

The results of the classification loop are shown in Figure 3.3. The heuristic method has the benefit of being able to clearly define the classification groups and receive segments that fulfill all desired criteria. However, the method also has the disadvantage that because the algorithm only searches for segments that fit exactly within the

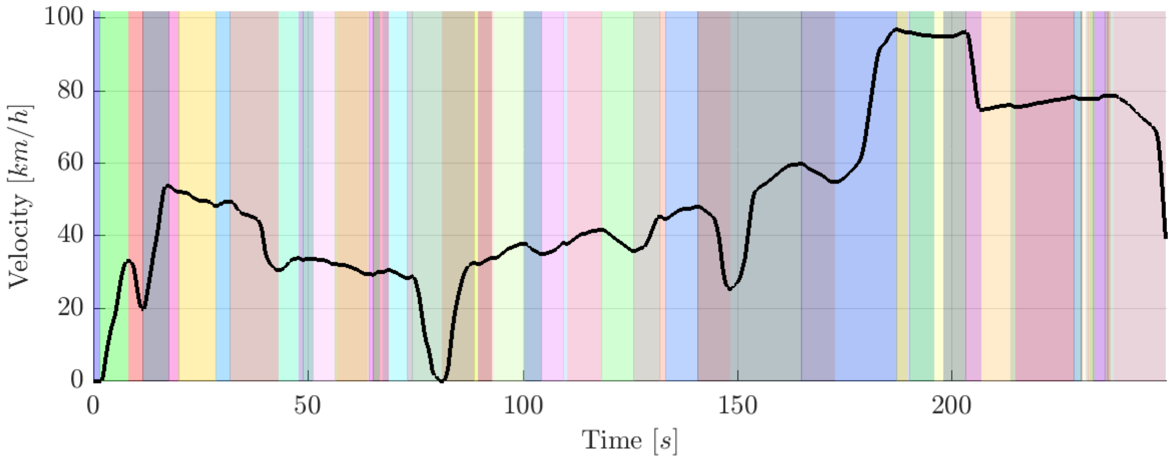


Figure 3.2: The velocity trajectory after the first, rudimentary segmentation. Segments are formed from adjacent points with acceleration values of the same sign.

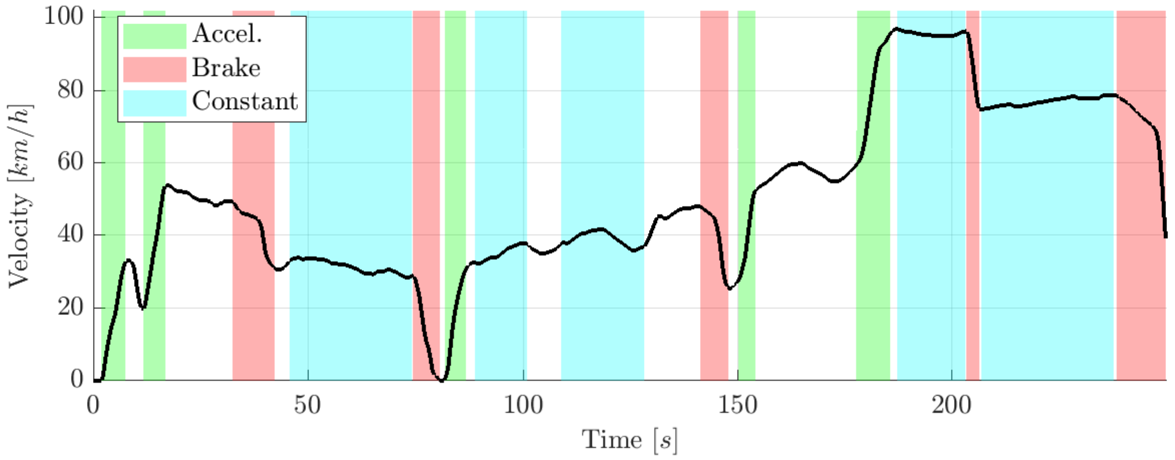


Figure 3.3: The final classification result using the heuristic method.

predefined groups, useful data is often lost.

3.1.2 Dynamic Programming Segmentation

This sections presents a background on dynamic programming as well as it's application to data classification. Then, the dynamic programming classification algorithm is applied to a small, example velocity trajectory.

Fundamentals

Dynamic Programming (DP) is a technique for solving discrete optimization problems developed by Richard Bellman in 1957 [6]. The technique is based on the principle of optimization which, directly quoted from Bellman, states:

"*An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*"

Here, a policy is defined as a sequence of decisions that are made with respect to fulfilling a goal. This could be directions for traveling from point A to point B or the display color for pixels in order to show an image. In short, the principle of optimality states that an optimal solution can be broken down into a series of optimal decisions. If some solution has a sub solution that is not optimal, the entire solution could be improved by replacing the non-optimal sub-solution with an optimal one.

Following from [39], we will now present the general theory behind using DP to solve a sequential decision process. Formally, a sequential decision process can be written to $d_{\in\Delta}H(d)$ where d is a decision chosen from an eligible set of decisions Δ and H is the objective function that has some cost associated with the decision. In almost all cases, we seek the optimal value $H^* = H(d^*)$ where d^* is the optimal decision and can be found by solving $\arg\operatorname{opt}_d\{H(d)\}$. This optimal solution can be defined as a maximum, minimum, or something else entirely related to the objective function that suits the user's needs.

To generalize, we will assume that d contains multiple decisions $\{d_1, d_2, \ldots, d_n\}$, that, when solved optimally, will yield an optimal value of H. One solution to this method is commonly referred to as the "brute force" method where every possible decision combination is calculated and the minimum is found. While this method always results in an optimal solution, it is extremely computationally expensive, requires a very long time to compute, and is therefore not practical in most applications. Thankfully, DP offers a more efficient solution.

DP assumes that the decisions in the sequential decision process must be made in a certain order, or sequentially: d_1 must be completed before d_2 can be made and so on. This decision sequence is solved such that:

$$\begin{aligned} H^* &= \operatorname{opt}_{(d_1,\ldots,d_n)\in\Delta}\{h(d_1, d_2, \ldots, d_n)\} && (3.1)\\ &= \operatorname{opt}_{d_1\in D_1}\{\operatorname{opt}_{d_2\in D_2}\{\ldots\{\operatorname{opt}_{d_n\in D_n}\{h(d_1, \ldots, d_n)\}\}\ldots\}\}, && (3.2)\end{aligned}$$

where the sequential decisions $(d_1, d_2, \ldots, d_n) \in \Delta = D_1 \times D_2 \times \cdots \times D_n$. It is important to note that the available decisions at step i are influenced by all other steps leading up to it, meaning that for $d_i \in D_i$, D_i is a function of $(d_1, d_2, \ldots, d_{i-1})$.

Plugging this fact into (3.2) results in:

$$\begin{aligned} H^* &= \text{opt}_{(d_1,\ldots,d_n)\in\Delta}\{h(d_1,d_2,\ldots,d_n)\} && (3.3)\\ &= \text{opt}_{d_1\in D_1}\{\text{opt}_{d_2\in D_2(d_1)}\{\ldots\{\text{opt}_{d_n\in D_n(d_1,\ldots,d_n)}\{h(d_1,\ldots,d_n)\}\}\ldots\}\}, && (3.4)\end{aligned}$$

To solve this decision problem, we start at the inside and move outwards. The solution to the innermost problem yields an optimal decision $d_n = d_n^*$ as a function of the other decisions $d_n^*(d1,\ldots,d_{n-1})$ and solving the outermost problem yields $d_1 = d_1^*$ as a function of d_1 and the other optimal decisions: $d_1^* = \text{opt}_{d1\in D_1}\{h(d_1,d_2^*,\ldots,d_n^*)\}$. Note, we can also reach a solution by reversing the order in which the decisions are made starting with the solution for d_1^* and ending with d_n^*. The different solution order must still result in the optimal solution, but because the method of reaching it was different the solution efficiency may vary.

Moving forward, we arbitrarily start by solving for d_n^* first and d_1^* last. This choice results in solving for each decision step sequentially, leaving the final step to solve for d_1^* as a function of d_1 and the optimal choices for each of the other decisions as shown in (3.5).

$$H^* = \text{opt}_{d_1\in D_1}\{h(d_1,d_2^*(d_1),\ldots,d_n^*(d_1))\}, \qquad (3.5)$$

The solution to this last step requires evaluating the objective function for all possible values of d_1 and choosing the optimal one. In this respect, the optimal choices for the other sequence decisions are constants and the optimization problem can be rewritten as just a function of d_1: $opt_{d_1\in D_1}\{H'(d_1)\}$. Following this logic, we suppose that the objective function is weakly separable with respect to each decision in the sequence:

$$h(d_1,\ldots,d_n) = C_1(d_1|\emptyset)\circ C_2(d_2|d_1)\circ\cdots\circ C_n(d_n|d_1) \qquad (3.6)$$

where $\circ$ is the associative binary operator (multiplication or addition). Assuming that our objective function h is weakly separable with respect to the decision steps, we can then write (3.2) as the following:

$$opts_{d_1\in D_1}\{C_1(d_1|\emptyset)\circ opt_{d_2\in D_2(d_1)}\{C_2(d_2|d_1)\circ\cdots\circ\{opt_{d_n\in D_n}\{C_n(d_n|d_1,\ldots,d_1)\}\}\ldots\}\} \qquad (3.7)$$

Using the weakly separable relationship between the sequence decisions, we can show the recursive relationship between any arbitrary step and the prior step to that. We start by defining $f(d_1,\ldots,d_n)$ as the function describing the optimal decision process where steps $(1-i-1)$ have been completed and steps $(i-n)$ have yet to be computed.

$$f(d_1,\ldots,d_n) = \text{opt}_{d_i}\{opt_{d_{i+1}}\{\ldots\{\text{opt}_{d_n}\{C_i(d_i|d_1,\ldots,d_{i-1})\circ C_{i+1}(d_{i+1}|d_1,\ldots,d_i)\circ\ldots \\ \circ\, C_n(d_n|d_1,\ldots,d_{n-1})\}\}\ldots\}\} \quad (3.8)$$

which simplifies to:

$$\begin{aligned} f(\emptyset) = \text{opt}_{d_1}\{&\text{opt}_{d_2}\{\ldots\{\text{opt}_{d_n}\{C_1(d_1|\emptyset)\circ C_2(d_2|d_1)\circ\ldots \\ &\circ C_n(d_n|d_1,\ldots,d_{n-1})\}\}\ldots\}\}, &(3.9)\\ = \text{opt}_{d_1}\{&C_1(d1|\emptyset)\circ \text{opt}_{d_2}\{C_2(d_2|d_1)\circ\ldots \\ &\circ\{\text{opt}_{d_n}\{C_n(d_n|d_1,\ldots,d_{n-1})\}\}\ldots\}\}, &(3.10)\\ = \text{opt}_{d_1}\{&C_1(d_1|\emptyset)\circ f(d_1)\}, &(3.11)\end{aligned}$$

and can be generalized to the form:

$$f(d_1,\ldots,d_{i-1}) = \text{opt}_{d_i\in D_i(d1,\ldots,d_{i-1})}\{C_i(d_i|d_1,\ldots,d_{i+1})\circ f(d_1,\ldots,d_i)\} \quad (3.12)$$

Equation (3.12) is referred to as the dynamic programming function equation (DPFE), where the unknown is the recursive function f. This equation forms the foundation of DP because solving for step i depends on the solutions for steps $(1-i-1)$. Written more generally, the equation has the form:

$$f(S) = \text{opt}_{d_i\in D_i(S)}\{C_i(d_i|S)\circ f(S')\} \quad (3.13)$$

where the steps $d_1,\ldots,d_{i-1}$ have been replaced by the set S and S' represents the next step in the solution. This equation allows us to recursively find the optimal solution to a sequential decision process by solving each step as a smaller subproblem and using the results from the previous steps to solve the next ones. In the next section we will apply this equation to the k-Segmentation problem.

Bellman k-Segmentation

The goal of the k-segmentation problem is to create k segments from a time-series consisting of sequential data points, where each segment consists of adjacent samples with homogeneous characteristics. Following from [29], we formally define the k-segmentation problem and explain how dynamic programming is applied to solve it in the context of this work.

Definition. A time-series s contains N data samples $x(1)$, $x(2)$, $\ldots$, $x(N)$, where each sample has d dimensions. A segment of the time-series on the interval from points a to b is defined here as $s(a, b)$ and contains samples $x(a)$, $x(a+1)$, $x(a+2)$, $\ldots$, $x(b)$, where $a \leq b$. For k segments, there exist $k+1$ segment boundaries $c_0 < c_1 < c_2 < \cdots < c_k$, where c_0 is the first sample in the time-series and c_k is the last. To achieve k segments that are internally uniform, we define a cost function H that measures the homogeneity of some segment:

$$cost_H(s(a, b)) = H(x; n | x \in s(a, b)), \tag{3.14}$$

where the total cost of the time-series segmentation is the sum of the individual segment costs.

$$cost_H(s_1 s_2 \ldots s_k) = \sum_{i=1}^{k} cost_H(s_i) \tag{3.15}$$

The optimal cost H^* would therefore lead to an optimal segmentation of time-series s with respect to cost function H.

Dynamic Programming is based on solving subproblems as a smaller part of a larger problem. The idea that optimal solution to smaller subproblems lead to a total solution to the larger one allows us to solve the k-segmentation problem optimally and efficiently. To apply DP to the k-segmentation problem on a time-series s with N, d-dimensional samples, we start by defining the recursive relationship between segmentation subproblems by considering solving problems with $1 \leq k' \leq k$ desired segments [65].

Let $E_\mathrm{s}[i, k']$ represent the value of the arbitrary cost function for a segmentation of samples $\{x_1, \ldots, x_i\}$ using k' segments and let $E[i, j]$ be the value of the cost function for a single segment on samples $\{x_i, \ldots, x_j\}$.

To start, we examine $k' = 1$ where a single segment is fit across all data points in the time-series. Here, $E_\mathrm{s}[N, 1]$ is simply $E[1, N]$. Increasing k' to 2 is more complicated. Now the problem is to find the sample i where the boundary between the two segments should be placed. The cost of our segmentation becomes:

$$E_\mathrm{s}[i, 1] = E[1, i], \tag{3.16}$$

$$E_\mathrm{s}[i, 2] = \min_{1 \leq j \leq i} (E_\mathrm{s}[j-1, 1] + E[j, i]) \tag{3.17}$$

Table 3.2: Table containing single segment costs for a $k = 5$ segmentation problem.

LHB \ *RHB*	1	2	3	4	5
1	$1 \to 1$	$1 \to 2$	$1 \to 3$	$1 \to 4$	$1 \to 5$
2		$2 \to 2$	$2 \to 3$	$2 \to 4$	$2 \to 5$
3			$3 \to 3$	$3 \to 4$	$3 \to 5$
4				$4 \to 4$	$4 \to 5$
5					$5 \to 5$

This can be further generalized for any value of k':

$$E_{\mathrm{s}}[i, k'] = \min_{1 \leq j \leq i} (E_{\mathrm{s}}[j-1, k'-1] + E[j, i]) \tag{3.18}$$

which has a form remarkably similar to the recursive (3.13). The solution to the k'-segmentation problem can only be solved once the solution to the $k' - 1$ segment problem has been solved.

Now that the k-segmentation problem and a solution method have been defined, we present the specific algorithm used to find the optimal solution. Equation (3.18) requires the cost function value for any segment containing samples $\{x_i, \ldots, x_j\}$ and it is therefore useful to compute the cost values for a single segment between any two points in the time-series before solving the DP problem. These values are stored in a matrix similar to that shown in Table 3.2. In this five segment example case, the left-hand segment bounds, *LHB*, are represented by rows and the right hand ones, *RHB*, by columns. The cost for a segment with the corresponding bounds is placed in the appropriate cell. Note that the diagonal contains only single sample segments and that *RHB* must be greater than *LHB*.

This table has the additional characteristic that each diagonal represents segments of equal length; i.e., the central diagonal contains segments of length one, the first upper diagonal contains segments of length two, and so on. Therefore, we can easily bound the minimum segment length by penalizing such segments with an infinitely high cost. If we wanted, for example, only segments with a length greater than three, we can manually change the cost value in each cell of the central, first, and second upper diagonals to ∞. This makes the cost for these segments non-optimal and will never be chosen by the DP solver.

With the pre-computation completed, we define two additional tables essential to the DP algorithm. In Table 3.3, we depict the recursive nature of the DP solution by showing the possible number of segments in the rows, the adjusted segment length in the columns, and the optimal segmentation in the cells. The term "adjusted length" is used because a single column has been added to the left side of the table, representing the possibility that a single segment over all samples could have a lower cost than the previously calculated segmentations. To start the algorithm, we solve for the optimal solutions to each case in the first row. Because all segmentations here contain only one segment, there is one possible solution for each cell and we simply fill all cells (starting at column two) with the first row from Table 3.2. The extra column is initialized with a value of zero and the arrow notation $i \rightarrow j$ signifies a segment between samples i and j.

After initializing the first row with the trivial values, we begin to solve the optimization problems in the second row. For example in cell $(2, 3)$, we need to find cost-optimal combination between $1 \rightarrow 2$ and $1 \rightarrow 1 + 2 \rightarrow 2$. Similar problems exist in the cells further to the right, with complexity increasing as the number of possible segment combinations increases. In the third row the recursive nature of the solution emerges and we are able to use the solutions found in our previous cells to solve current problems. In cell $(3, 4)$ we have four possible segment combinations: $1 \rightarrow 3$, $1 \rightarrow 1 \ + \ 2 \rightarrow 3$, $1 \rightarrow 2 \ + \ 3 \rightarrow 3$, and finally $1 \rightarrow 1 \ + \ 2 \rightarrow 2 \ + \ 3 \rightarrow 3$. Note that in the last two possibilities, the optimal solution between $1 \rightarrow 2$ and $1 \rightarrow 1 \ + \ 2 \rightarrow 2$ has already been found in cell $(2, 3)$ and can be applied to cell $(3, 4)$, simplifying the possible combinations. Simplifications made possible by the recursive nature of DP are shown in the table as red. The optimal solution to the k-segmentation problem over the entire time-series is solved by completing the remaining cells in this manner until we reach the lower-right most corner.

Algorithm 2 shows pseudo-code for the DP k-segmentation algorithm and begins with the initialization of the desired number of segments, k, and the cost function H. Next, we pre-compute the cost for all possible single-segment combination of points. These costs are stored in the $Cost_{\text{Matrix}}$. Following the pre-computation, we begin the dynamic programming algorithm by initializing and solving the Sol_{Matrix} which corresponds exactly to Table 3.3. Note, however, that only the costs for the optimal solutions are stored in the Sol_{Matrix}. Therefore, it is necessary to also store index corresponding to the optimal solution so that once the DP is complete, we can trace our solution back through matrix containing the solutions to each of the subproblems.

Example. An application of the dynamic programming k-segmentation algorithm on an example velocity trajectory is presented using the Z-Scale cost function found

Table 3.3: Table displaying the recursive DP solution to the k-segmentation problem.

Length -1

# of Segments	1	2	3	4	5	6	$\dots$
1		$1 \to 1$	$1 \to 2$	$1 \to 3$	$1 \to 4$	$1 \to 5$	$1 \to 6$
2			$1 \to 2$ or $1 \to 1 + 2 \to 2$	$1 \to 3$ or $1 \to 1 + 2 \to 3$ or $1 \to 2 + 3 \to 3$	$1 \to 4$ or $1 \to 1 + 2 \to 4$ or $1 \to 2 + 3 \to 4$ or $1 \to 3 + 4 \to 4$	$\dots$	$\dots$
3				$1 \to 3$ or $1 \to 1 + 2 \to 3$ or $(2,3) + 3 \to 3$	$1 \to 4$ or $1 \to 1 + 2 \to 4$ or $(2,3) + 3 \to 4$ or $(2,4) + 4 \to 4$	$\dots$	$\dots$
4					$\vdots$	$\ddots$	$\dots$
$\vdots$						$\vdots$	$\ddots$

in [28]:

$$H = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{w_i |x_{ij} - \bar{x}_i|}{S_i}, \tag{3.19}$$

$$S_i = \sqrt{\frac{\sum_{j=1}^{n} (x_{ij} - \bar{x}_i)^2}{n-1}} \tag{3.20}$$

where x_{ij} is the jth sample of variable i in a given segment, $\bar{x}_i$ is the mean of the segment with respect to variable j, S_i is the standard deviation of the segment with respect to variable i, m is the total number of variables, n is the number of samples in the segment, and w_i is a weight assigned to variable i. Our cost function is capable of handling multivariate time-series and we therefore use both velocity and acceleration of the segments to calculate the cost. Velocity data is filtered using a Hanning filter and acceleration is calculated by differentiating the velocity as shown in (3.21).

$$a(t) = \frac{dv}{dt} = \frac{v_{k+1} - v_k}{t_{k+1} - t_k} \tag{3.21}$$

We process the acceleration data one step further by setting any point with a value less than a threshold acceleration, in this case $a_{\text{thresh}} = 0.4 \frac{m}{s^2}$, to a value of zero.

```
Data: Time-Series Data, $s(t)$
Result: $k$-Segment Time-Series Segmentation
Inputs: # of desired segments $k$, cost function $H$
Define: $N = \text{length}(s(t))$ Initialize Diagonal to Zero: $\text{diag}(Cost_{\text{Matrix}}) = 0$
Pre-compute single segment costs as shown in Table 3.2
for $l = 1$ to $N$ do
    for $r = l + 1$ to $N$ do
        Pull segment data: $Segment_{\text{Data}} = s(l : r)$
        Calculate segment cost: $Segment_{\text{Cost}} = H(Segment_{\text{Data}})$
        Store cost: $Cost_{\text{Matrix}}(l, r) = Segment_{\text{Cost}}$
    end
end
Dynamic Programming Algorithm
Define first row of solution matrix: $Sol_{\text{Matrix}}(1, 2 : N) = Cost_{\text{Matrix}}(1, :)$
Pre-Allocate first row of solution path: $Sol_{\text{Path}}(1, :) = 0$
Define diagonal of solution path: $Sol_{\text{Path}}(1 : k, 1 : k) = (1 : k) - 1$
Loop through remaining sub-cases for $p = 2$ to $k$ do
    for $n = p$ to $N$ do
        Pull segmentation choices: Choices =
        $Sol_{\text{Matrix}}(p - 1, 1 : n) + Cost_{\text{Matrix}}(1 : n, n)'$
        Calculate best choice: $[best_{\text{choice}}, best_{\text{index}}] = \min(Choices)$
        Store solution: $Sol_{\text{Path}} = best_{\text{index}} - 1$
        Store solution cost: $Sol_{\text{Matrix}}(p, n + 1) = best_{\text{choice}}$
    end
end
Read Solution from $Sol_{\text{Path}}$ Matrix
```

Algorithm 2: Dynamic Programming k-Segmentation

This classification removes unwanted noise from the acceleration data, emphasizes when the acceleration is small, and allows the algorithm to better recognize constant velocity phases. An example of the velocity trajectory, acceleration data, and filtered acceleration data is shown in Figure 3.4.

After pre-processing, the data is given to the algorithm and the optimal segmentation is computed. Figures 3.5 and 3.6 shows the results of two solutions with velocity and acceleration weight values of 1.8 and 1 respectively, a minimum segment length of 5, and $k = 20$ and 40. On one hand, this method has the benefit that optimally homogeneous are calculated by taking all data into account. On the other, a long pre-computation time for the single segment cost matrix and the fact that the algorithm tends to choose shorter segments are less desirable. Because shorter segments tend to have lower costs, this normally results in many small segments and several extremely

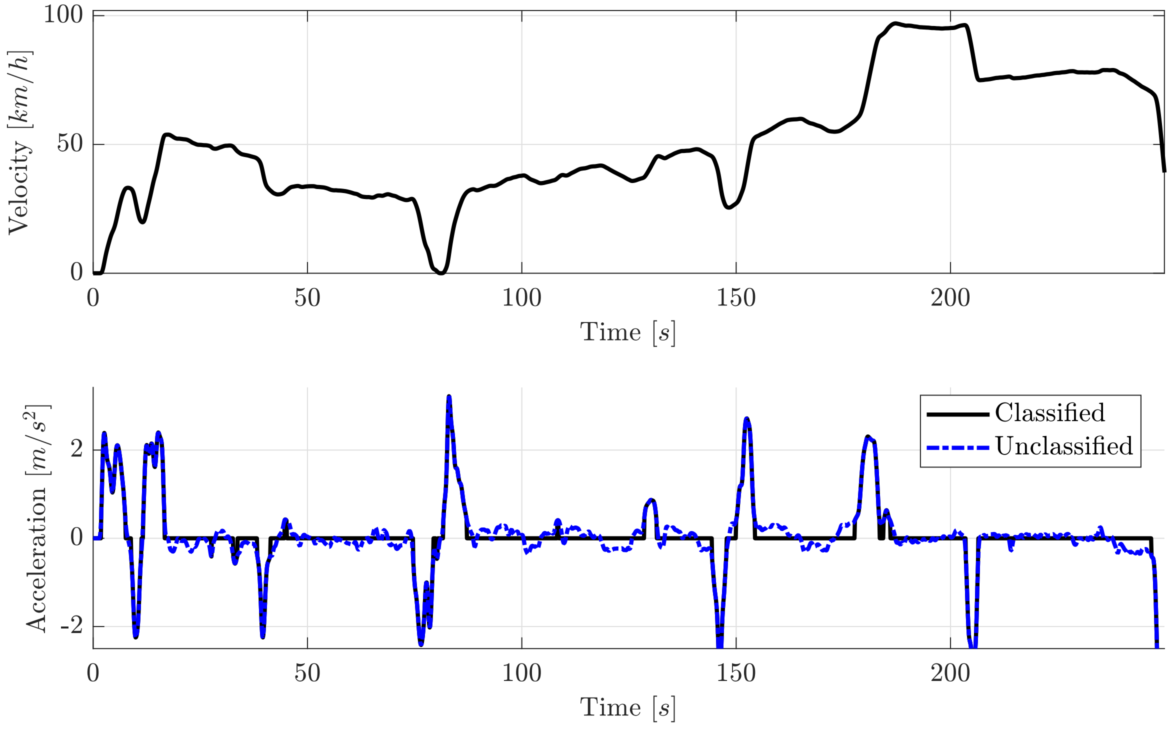


Figure 3.4: The example velocity trajectory (top) and the original and classified acceleration data (bottom).

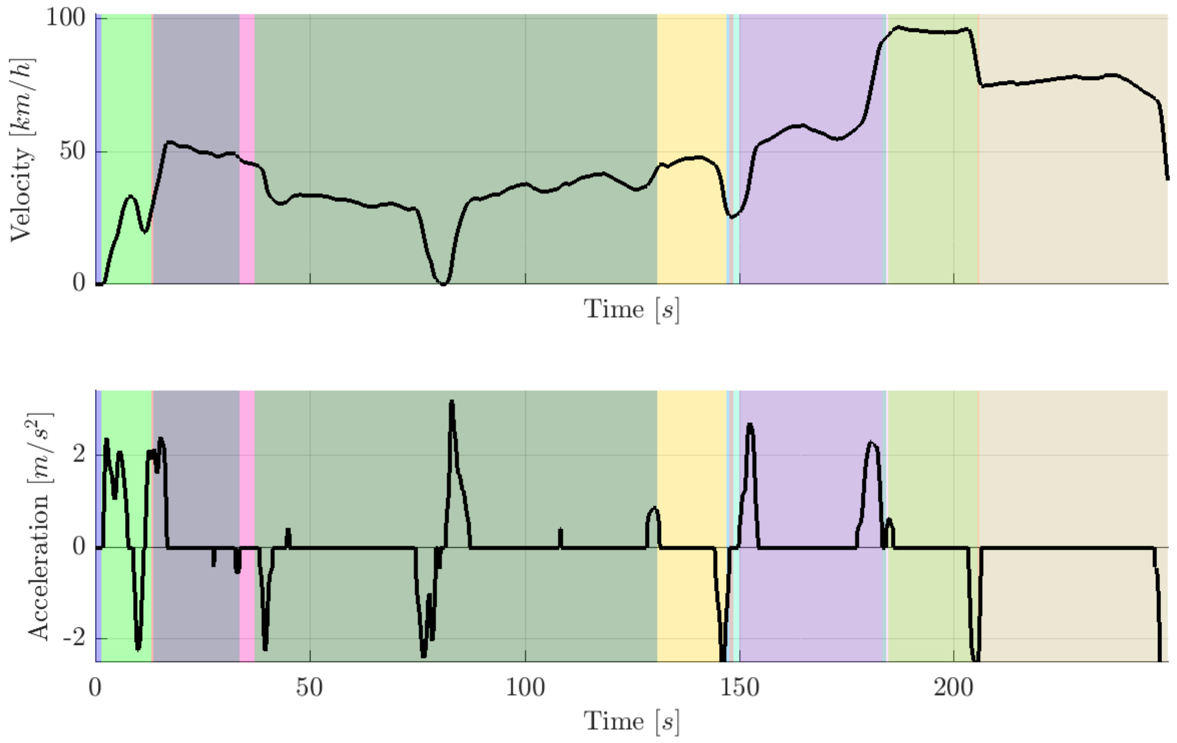


Figure 3.5: The DP k-segmentation results with $k = 20$.

long ones. The DP method is also limited by the requirement that one must choose a desired number of segments before running the algorithm. In most segmentation problems, the desired number of segments is an unknown and the user must tune the parameter k until the results look satisfactory.

3.1.3 The Bottom-Up Method

In addition to the heuristic and more formal dynamic programming approaches to data segmentation described in the previous sections, another algorithm known as the "Bottom-Up" method is utilized in this work. This method begins by creating the finest segmentation possible over the time-series, meaning that each sample is it's own segment and resulting in N segments from a time-series of length N. Then, the cost of merging each possible adjacent segment combination is calculated with respect to some cost function H. The algorithm sequentially combines the segment pairs with the lowest respective costs and continues until the stopping criteria is met. Stopping criteria examples include maximum total cost of all segments, maximum cost for a single segment, or a minimum/maximum number of segments. Note computational cost is low because after the initial step, the algorithm is only required to calculate the

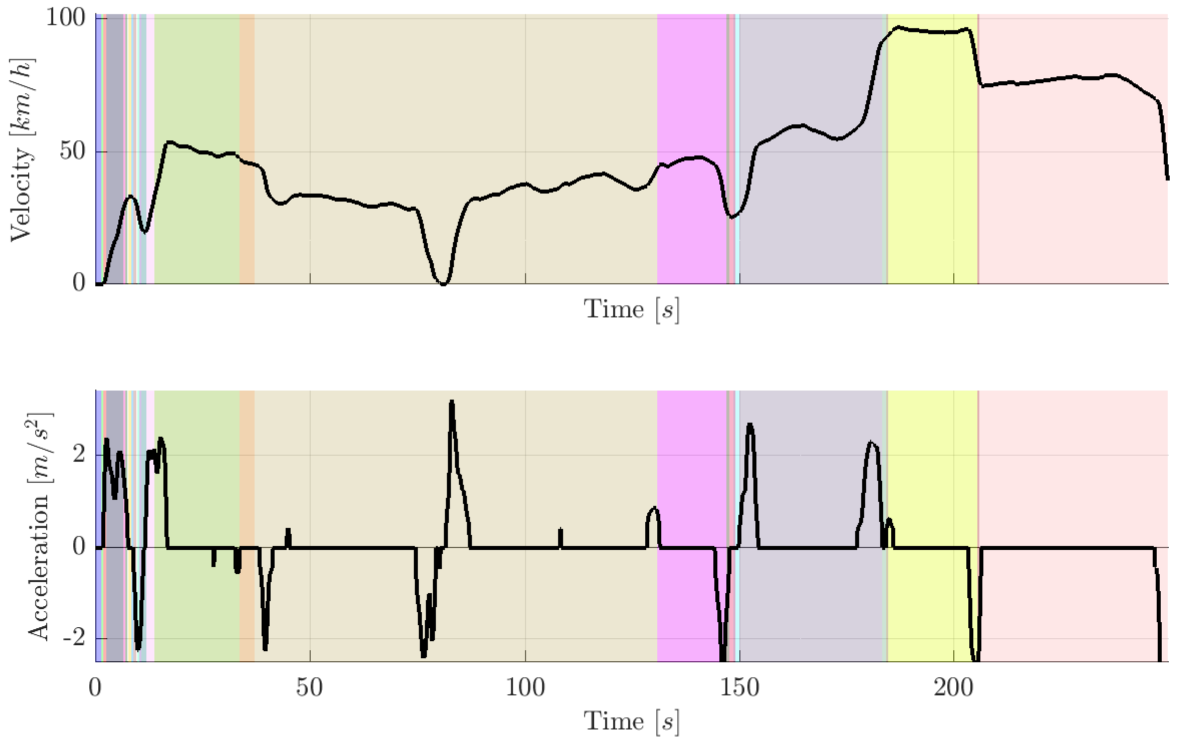


Figure 3.6: The DP k-segmentation results with $k = 40$.

costs associated with newly formed segments. Table 3.4 shows a practical example of the several iterations of the BU method. In the first row of the example table, the initial, fine segmentation of that data is shown where each sample is it's own segment. From here, the cost of each possible combination, $1 \rightarrow 2$, $2 \rightarrow 3$, etc., are computed and the first combination is found to have the lowest cost. Moving to the second line, we replace the old components of the newly formed segment and the process proceeds to the next rows.

Algorithm 3 shows the pseudo-code for the Bottom-Up segmentation method. The

Table 3.4: Bottom-Up Algorithm Demonstration with 8 samples.

	Segment Combination #							
Iteration	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8
2	$1 \rightarrow 2$	3	4	5	6	7	8	–
3	$1 \rightarrow 2$	3	$4 \rightarrow 5$	6	7	8	–	–
4	$1 \rightarrow 2$	$3 \rightarrow 5$	6	7	8	–	–	–
5	$1 \rightarrow 5$	6	7	8	–	–	–	–
6	⋮	⋮	⋮	⋮	–	–	–	–

initialization steps define the time-series $s(t)$, the error threshold e_{thresh}, and the cost function H. Next, we initialize the fine segmentation vector (as shown in Table 3.4), the possible combinations vector, and initialize the costs of these combinations in a *for* loop. After the initialization steps, we set the running cost $Cost_{\text{run}}$ to 0 and the BU method begins. The algorithm finds the index of the segment with the lowest cost, updates the segments, combinations, and cost vectors, and continues until the highest single segment cost in the segmentation is larger than the error threshold. The final segmentation can be found in the segments vector.

```
Data: Time-Series Data, $s(t)$
Result: Bottom-Up segmentation
Inputs: error threshold $e_{\text{thresh}}$, cost function $H$
Initialize first row of Fine Segmentation Vector: $Segment_{\text{Vec}}$
Initialize first row Possible Combinations Vector: $Combi_{\text{Vec}}$
Initialize Combination Cost Vector
for $i = 1$ to length($Combi_{Vec}$) do
    Pull segment data: $Segment_{\text{Data}} = s(Combi_{\text{Vec}}(i))$
    Calculate segment cost: $Segment_{\text{Cost}} = H(Segment_{\text{Data}})$
    Store cost: $Cost_{\text{Vec}}(i) = Segment_{\text{Cost}}$
end
$Cost_{\text{run}} = 0$
BU Segmentation Algorithm
while $Cost_{run} \leq e_{thresh}$ do
    Find optimal segment: $[best_{\text{choice}}, best_{\text{index}}] = \min(Cost_{\text{Vec}})$
    Update Segment, Combination, and Cost Vectors
    $Segment_{\text{Vec}} = \text{remove}(Segment_{\text{Vec}}, best_{\text{index}})$
    $Combi_{\text{Vec}} = \text{remove}(Combi_{\text{Vec}}, best_{\text{index}})$
    $Cost_{\text{Vec}} = \text{calc\_and\_remove}(Cost_{\text{Vec}}, best_{\text{index}})$
    $Cost_{\text{run}} = \max(Cost_{\text{Vec}})$
end
```

Algorithm 3: Bottom-Up Segmentation Algorithm

The Sliding Window and Bottom-Up Method

The sliding window is a simple time-series segmentation method popularized by the fact that is can be used online and a very low computational complexity. To start the sliding window algorithm, one takes the first sample in a given time-series and creates a window by concatenating adjacent points to the right of the initial one to it. When each additional point is added, the cost of the new segment window is calculated. At some sample i, the cost of the segment becomes greater than some maximum specified value and a segment is created from points $(1 : i - 1)$. Then, point i is treated as the first point in the new window and the process is repeated until the end of the time-series is reached. Unfortunately, the sliding window algorithm is limited by its ability to utilize points present only in the current segment.

Because the sliding window method alone produces segments with relatively low homogeneity, we investigate a segmentation method that combines both the Bottom-Up (BU) and Sliding Window approaches: Sliding Window and Bottom-Up (SWAB) as presented by Keogh et al. [36]. This method combines the online capability of the sliding window with the improved results of the BU method. The SWAB method maintains a window of size w that is initially chosen to be 5-6 segment lengths and begins at the first sample in the time-series. To start, the normal BU Algorithm is applied to this window. Once the stopping criteria for this BU is reached, the left-most segment in this solution is taken and the left edge of the window is shifted to left, removing the segment. In order to maintain the relative size of the window, new points are added to the right side, chosen using the normal sliding windows algorithm where new points are added to the window until the segment formed by the new points reaches the maximum allowable cost. In the offline case, this process continues until the end of the time-series is reached and the results from the final BU algorithm are accepted for the last window. Pseudo-code for the offline SWAB algorithm is provided in Algorithm 4.

Example. An example of both the Bottom-Up and SWAB methods is now presented using a sample velocity trajectory. Similarly to the DP k-segmentation, velocity and acceleration are used as the time-series variables but the cost function has been slightly modified as shown in (3.22). In this case, we divide by the segment length, l, instead of the standard deviation. This change serves to normalize the cost with respect to segment length and also encourage segments of longer lengths.

$$H = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{w_i |x_{ij} - \bar{x}_i|}{l} \tag{3.22}$$

Acceleration is calculated by differentiating filtered velocity data (as shown in (3.21)) and has been further classified using the same method shown in Figure 3.4. After the pre-processing, the data is subjected to both the generic BU and SWAB algorithms using (3.22) as a cost function. The resulting segmentations are shown in Figures 3.7 and 3.8. Here, the window length in the SWAB algorithm is $w =$

```
**Data**: Time-Series Data, $s(t)$
**Result**: SWAB segmentation
Inputs: error threshold $e_{\text{thresh}}$, cost function $H$, window size $w$
Define: $N = \text{length}(s(t))$
$Data_{\text{remain}} = N$
$Points = [1:1:w]$
while $Data_{remain}$ do
    Update remaining data: $Data_{\text{remain}} = Data_{\text{remain}} - \text{length}(Points)$
    Pull data from time-series: $Data_{\text{in}} = s(Points)$
    Calculate BU over window: $Seg_{\text{BU}} = \text{Bottom\_Up}(Data_{\text{in}}, e_{\text{thresh}})$
    Extract first segment: $S_{\text{new}} = Seg_{\text{BU}}(1)$
    Remove new segment from window: $Points = \text{remove}(Points, S_{\text{new}})$
    $d = 1$
    $error = 0$
    while $error < e_{thresh}$ do
        Add new points to window: $Seg_{\text{SWAB}} = \text{Concat}(Points, d)$
        Calculate cost of segment: $error = H(Seg_{\text{SWAB}})$
        Increment: $d = d + 1$
    end
    Add new points to window $Points = \text{Concat}(Points, Seg_{\text{SWAB}})$
end
```

Algorithm 4: SWAB Segmentation Algorithm

100 samples, the error threshold is 0.4, and the weights on v and a are 1.8 and 1 respectively.

Note that the results for the BU and SWAB methods are similar, with the exception that the segment length for the SWAB method is limited by the width of the window. This results in the SWAB segmentation containing generally shorter segment lengths and while the SWAB method does contain the same segment boundaries as the BU, there is a finer segmentation within these bounds that better captures the acceleration behavior. Both of these algorithms have the advantage that they are not computationally complex, do not require the user to define a desired number of segments, and create segments that are homogeneous with respect to the cost function.

3.1.4 Selection of a Segmentation Method

Five individual velocity and acceleration trajectories are segmented using each of the three formal possible segmentations presented in this work. These results are compared and analyzed using several different metrics to judge the effectiveness of each segmentation method. Constructing an accurate behavioral model requires training

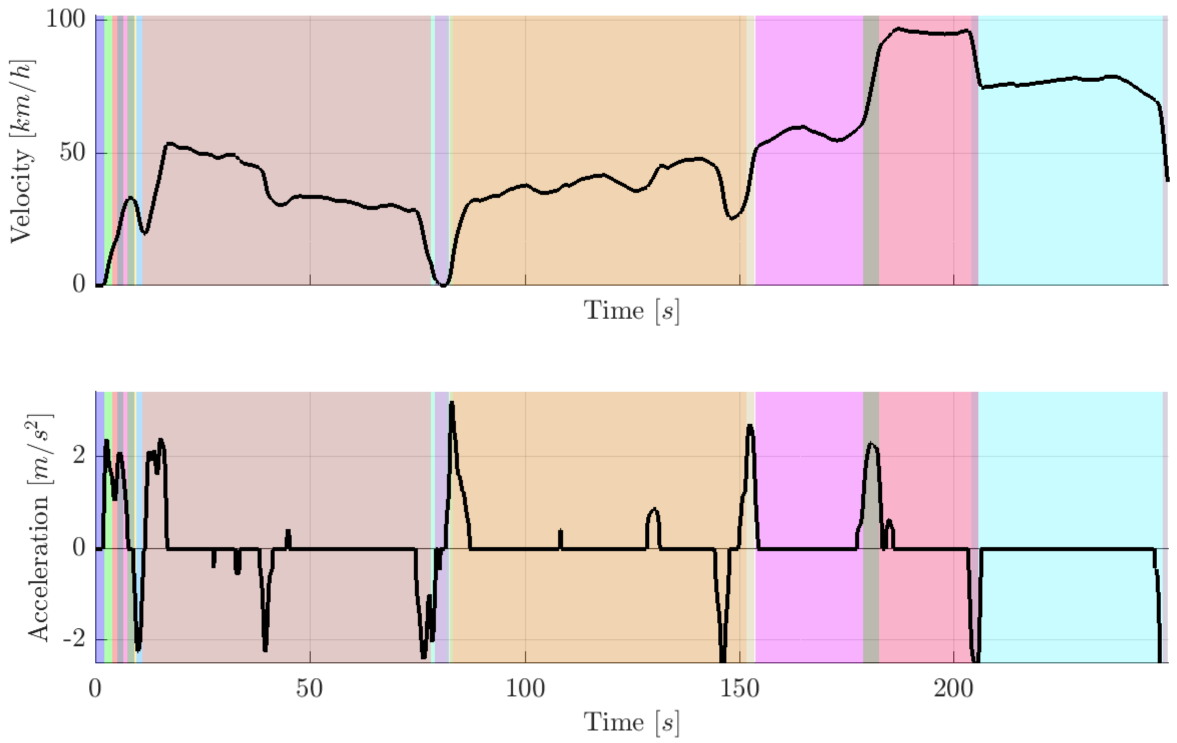


Figure 3.7: The final classification result using the BU method.

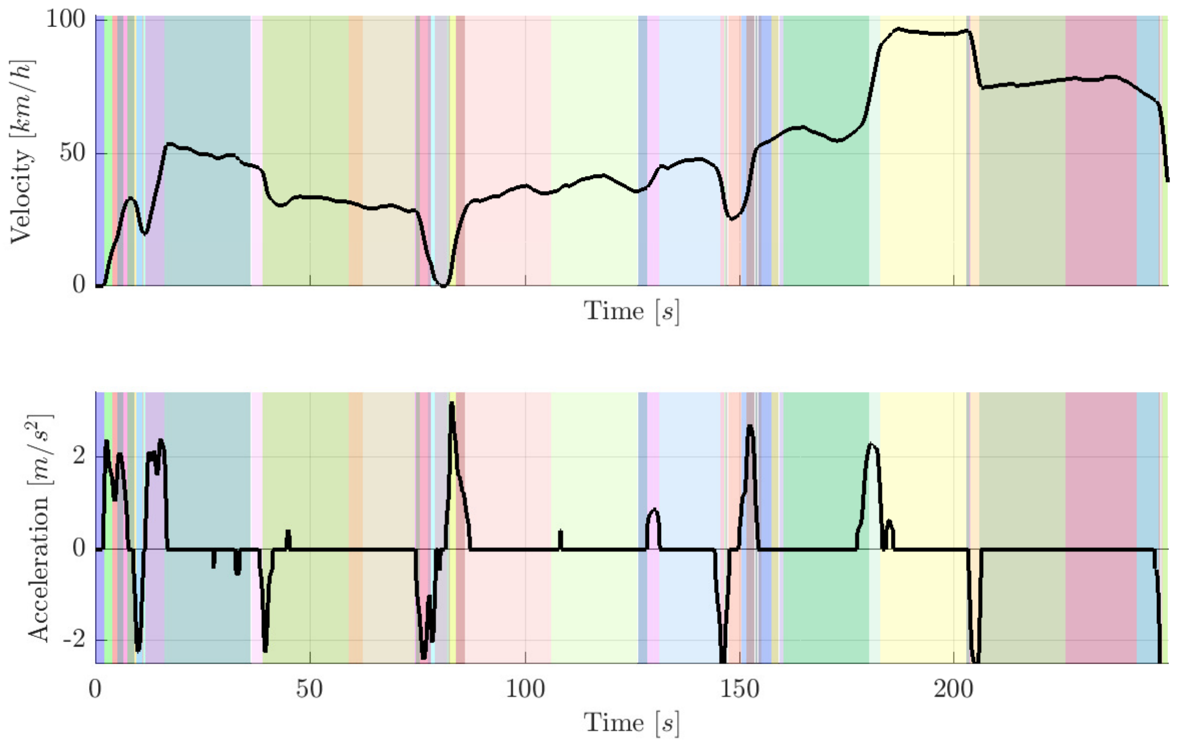


Figure 3.8: The final classification result using the SWAB method.

data that captures real-world behavior as closely as possible. Therefore, the chosen metrics serve to measure the accuracy of each segmentation method in their ability to capture homogeneous acceleration behavior in individual segments:

- The average length of the segments in the segmentation: $\bar{l}$.
- The standard deviation on the mean of the segment lengths: S_l.
- The percentage of the acceleration phases that are positive: $+\%$.
- The percentage of the braking phases that are negative: $-\%$.
- The normalized running time on a *Dell Precision T7610* desktop with an Intel Xeon 8 core processor and 64 GB of memory: t. The first 3000 samples of each trajectory are taken in order to preserve the accuracy of the timing measurement.

Acceleration phases typically last between 5-20 seconds and the length and standard deviation metrics measure the ability of the algorithm to capture this behavior. Long segments lengths imply that more than one type of behavior has been captured. In the $+\%$ and $-\%$ metrics, segments are first classified as acceleration and braking phases by analyzing the points contained in a segment. Segments consisting of 50% or more positive acceleration values are classified as acceleration phases and vice versa for segments consisting of 50% or more negative accelerations. The actual percentage of positive acceleration values in the acceleration phases (and the opposite for braking ones) is measured and used as the metric. Low values indicate that "acceleration" segments capture multiple types of behaviors whereas higher ones indicate homogeneous segments. Finally, the time-based metric serves to measure how computationally feasible each segmentation method is for a real-world application.

Table 3.5: Statistical segmentation result comparison.

	DP					*BU*					*SWAB*				
Trip	$\bar{l}$	S_l	$+\%$	$-\%$	$t\ (s)$	$\bar{l}$	S_l	$+\%$	$-\%$	$t\ (s)$	$\bar{l}$	S_l	$+\%$	$-\%$	$t\ (s)$
1	63	145	74	76.6	260.7	103	265	61.6	100	225.3	39	50	77.7	84.0	24.3
2	38	70	82.8	78.5	260.4	28	69	86.3	87.8	221.9	25	36	85.3	89.9	33.8
3	38	59	84.6	89.1	259.4	86	146	71.6	76.7	223.8	42	49	81.5	88.4	27.5
4	51	141	69.1	77.0	257.3	64	83	70.0	68.0	224.9	45	46	77.7	71.5	20.2
5	38	92	80.9	67.3	261.0	45	162	79, 2	58.8	226.9	27	42	78.1	78.0	36.8
Avg.	46	102	78.3	77.5	257.8	66	146	73.6	78.3	224.6	36	45	80.1	82.3	28.5

Table 3.5 shows the results of the comparison conducted between the three segmentation methods. Here, the error threshold for the BU and SWAB methods is $e_{\text{thresh}} = 0.2$, the weights for v and a are equal to 1.8 and 1 respectively, and the desired number of segments for the DP is equal to 80. The rows show the different velocity trajectories with the bottom most row being an average of the data presented above it. The columns show the characteristics of the results from each of the three methods: DP, BU, and SWAB. The first important difference between the methods is that the SWAB method requires significantly less run time, taking, on average, nearly a tenth of the time of the other two. Otherwise, the mean segment lengths are similar for each method, but the standard deviation on these means differs greatly. Both the DP and BU methods had standard deviations of over 100 meaning that the segments were either very long (containing 150 - 200 samples) or extremely small. The SWAB method, on the other hand, has relatively low segment lengths with an appropriately sized standard deviation. Finally, the positive and negative percentages of the acceleration and braking phases are similar for each method with the SWAB method being a handful of percentage points above than the others.

In summary, the SWAB method demonstrates a high accuracy in driving phase recognition, has appropriately sized, homogeneous segments, and requires little computation time when compared to the other two methods. Therefore, the SWAB algorithm is selected as the final method to be used in conjunction with the post processing methods and the function fit regression presented in the following sections.

3.1.5 Post Processing Methods

The three statistical methods yield segmentation results that are optimal with respect to the respective algorithms and cost functions. The dynamic programming approach yields a segmentation where the sum of the cost segments is minimally optimal and both the Bottom-Up and SWAB approaches yield segmentations that have optimal segments formed from combining other optimal segments within their respective windows. Furthermore, the SWAB method is able to produce similar results to the other two methods while requiring nearly a tenth the computation time. However, upon visual inspection of these segmentation results, the algorithms are able to identify acceleration, braking, and constant velocity phases, but with a finer degree of accuracy than the modeling processes require. For example, in an acceleration phase, acceleration is consistently variable with three typical phases: a beginning where a is initially zero and begins to increase, the constant acceleration period where the driver holds consistent behavior, and the final phase where the acceleration tapers off. As

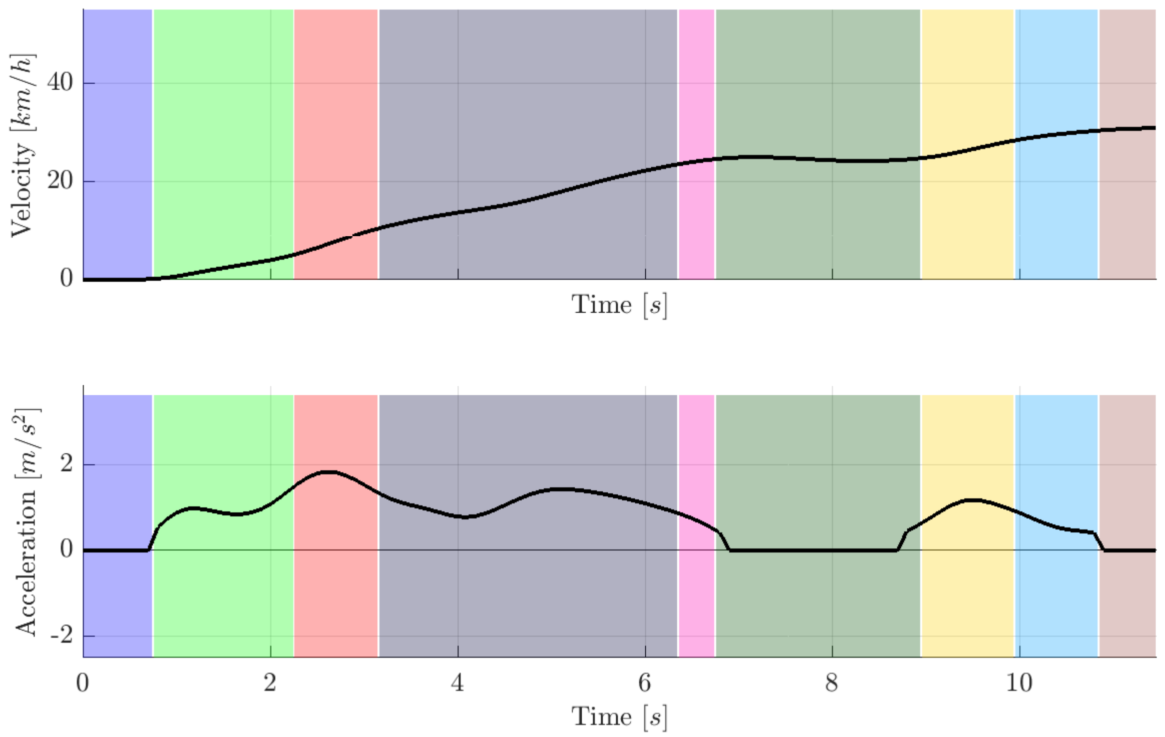


Figure 3.9: An example acceleration phase divided into multiple segments by the SWAB algorithm.

shown in Figure 3.9, the SWAB algorithm divides each phase of an acceleration as a different segment, whereas we seek to distinguish only the general acceleration behavior during such a phase. Therefore, it is necessary to perform a post-processing procedure on the segmentations with the goal of combining adjacent segments with similar acceleration behaviors. In this section, we explore three similarity measures and their application to combining adjacent segments from the SWAB segmentation results.

Mean Comparison

The first similarity measure is a generic comparison of the segment characteristics like sign, average, and magnitude. As shown in Algorithm 5, to compare the similarity of two adjacent segments, we first calculate the sign and the average. If the segments have the same sign, we compare the magnitudes by checking if the means both lie above or below a threshold acceleration. In the case that this second condition also holds, we compare the magnitudes further and check if the segment means are within a certain threshold of each other.

```
**Data**: Adjacent Segments $s_i$ and $s_j$
**Result**: Yes or no combination result.
Inputs: magnitude threshold 1 $m_{\text{thresh},1}$, magnitude threshold 2 $m_{\text{thresh},2}$
$flag_{\text{sign}} = 0$ % Initialize flags
$flag_{\text{mag},1} = 0$
$flag_{\text{mag},2} = 0$
$flag_{\text{combine}} = 0$
$\bar{s}_i = \text{mean}(s_i)$ % Calculate segment means
$\bar{s}_j = \text{mean}(s_j)$
$sign_i = \text{signum}(s_i)$ % Calculate segment signs
$sign_j = \text{signum}(s_j)$
if $sign_i = sign_j$ then
    $flag_{\text{sign}} = 1$
end
if $|\bar{s}_i| \leq m_{thresh,1}$ and $|\bar{s}_j| \leq m_{thresh,2}$ then
    $flag_{\text{mag},1} = 1$
else if $|\bar{s}_i| \geq m_{thresh,1}$ and $|\bar{s}_j| \geq m_{thresh,2}$ then
    $flag_{\text{mag},1} = 1$
end
if $|\bar{s}_i - \bar{s}_j| \leq m_{thresh,2}$ then
    $flag_{\text{mag},2} = 1$
end
if $flag_{sign}$ and $flag_{mag,1}$ and $flag_{mag,2}$ then
    $flag_{\text{combine}} = 1$
end
```

Algorithm 5: Mean-Comparison Similarity Measure

l-Association Interval Measurement

Where the first similarity measure is heuristic in nature, the second method utilizes the statistical l-association interval measurement, I_X^l, of a data set X. Following from [18], Gavilan states that according to Chebychev's inequality there is at least a $(1-1/l^2)$ proportion of samples x_i in the l-association interval and thus, we can define a similarity measure between two data sets, X and Y using the distance between their respective l-association measurements. The resulting similarity measure, $J_1(X, Y)$, is then defined as a function of the cardinalities (#), unions, intervals, and association intervals of the respective data sets as shown in (3.24).

$$I_X^l = (\bar{X} - lS_X, \bar{X} + lS_X) \tag{3.23}$$

$$J(X,Y) = \frac{\#((X \cup Y) \cap (I_X^l \cap I_Y^l))}{\#(X \cup Y)} \frac{1}{1 + d_W(I_X^l, I_Y^l)} \tag{3.24}$$

Where $d_W(I_1, I_2)$ is defined as the distance between two intervals

$$d_W(I_1, I_2) = \sqrt{(\Delta c, \Delta r) W \begin{pmatrix} \Delta c \\ \Delta r \end{pmatrix}}, \tag{3.25}$$

and c and r are defined based on the difference between the data set means and the difference between the standard deviations multiplied by the parameter l. Once the similarity of the two data sets, or segments in this case, has been calculated, the value is compared against some similarity threshold J_{thresh} and if the value is above the threshold, the segments are combined.

Percentile Measurement

Found in the same resource as the l-association similarity measurement ([18]), this method compares to data sets using their percentiles. A percentile is defined as a value in a data set, under which a certain percentage of the data points can be found. For example, the 50th percentile is the value under which 50% of values in a data set can be found. Given a vector of q percentiles of a data set, $Q_X = \{p_{1X}, p_{2X}, \ldots, p_{qX}\}$, where p_iX is the ith percentile, we calculate the similarity, J_2 between two data sets, X and Y, using the following equation:

$$J_2(X,Y) = \frac{1}{1 + \frac{1}{q-1}\sum_{i=1}^{q-1} d_W(I_{iX}, I_{iY})} \tag{3.26}$$

where $d_W(I_{iX}, I_{iY})$ is defined as

$$d_W(I_{iX}, I_{iY}) = \sqrt{\frac{1}{2}((p_{(i+1)X} - p_{(i+1)Y})^2 + (p_{iX} - p_{iY})^2)} \tag{3.27}$$

Like the l-association method, this approach also uses a similarity threshold to decide if the segments should be combined. If the similarity measure is above the threshold, the adjacent segments are combined.

Post Processing Method Selection

To choose the best similarity measurement between mean-comparison, l-association interval, and percentile approaches, we present a comparison on an example velocity trajectory. A SWAB segmentation is conducted using the same algorithm parameters presented Section 3.1.4, the three similarity-based segment combination methods are applied to the segmentation results, and the resulting segmentations are visually inspected to determine the best method.

Figure 3.10 shows the results of the mean-comparison (MC), l-association interval (LAI), and percentile approaches, respectively. The similarity thresholds are 0.005 and 0.64 for the LAI and percentile methods, respectively, and the two threshold values for the MC method are $0.4\frac{m}{s^2}$ and $0.55\frac{m}{s^2}$. Three different sections from the combination approaches have been highlighted to demonstrate the effectiveness of each method: an acceleration phase ($\sim 1-7s$), a gradual braking phase ($\sim 11-19s$), and finally a gradual acceleration phase containing many different accelerations ($\sim 62-88s$). In the first phase, we see that both the MC and the percentile methods are effective at combining the segments into a single phase but the LAI is not able to distinguish similarity between the segments. In fact, the LAI method is unable to combine a single pair of segments and will therefore be disregarded in this example. Similar to the first segment, the combination results from the MC and percentile methods are identical in the second. In the third segement however, we notice that the percentile approach is able to recognize that the segments make up one longer acceleration phase, whereas the MC method leaves them as individual segments. These and other, similar results from further case studies show that the percentile method is the most capable at combining segments with similar behavior and is be used alongside the SWAB algorithm in the function fit regression.

3.2 Function Fit Regression

In the previous sections, we defined one heuristic and three formal methods for the segmentation of a time-series, compared these three methods analytically to find the best one for application in the context of this work, and discussed three post-processing methods for combining similar segments after the segmentation process. After comparing the formal segmentation and the post-processing segment combination methods, we arrived at the conclusion that the SWAB algorithm and the percentile based similarity method are the most effective with respect to constructing an accurate driver acceleration behavior model. In this section, the process for mod-

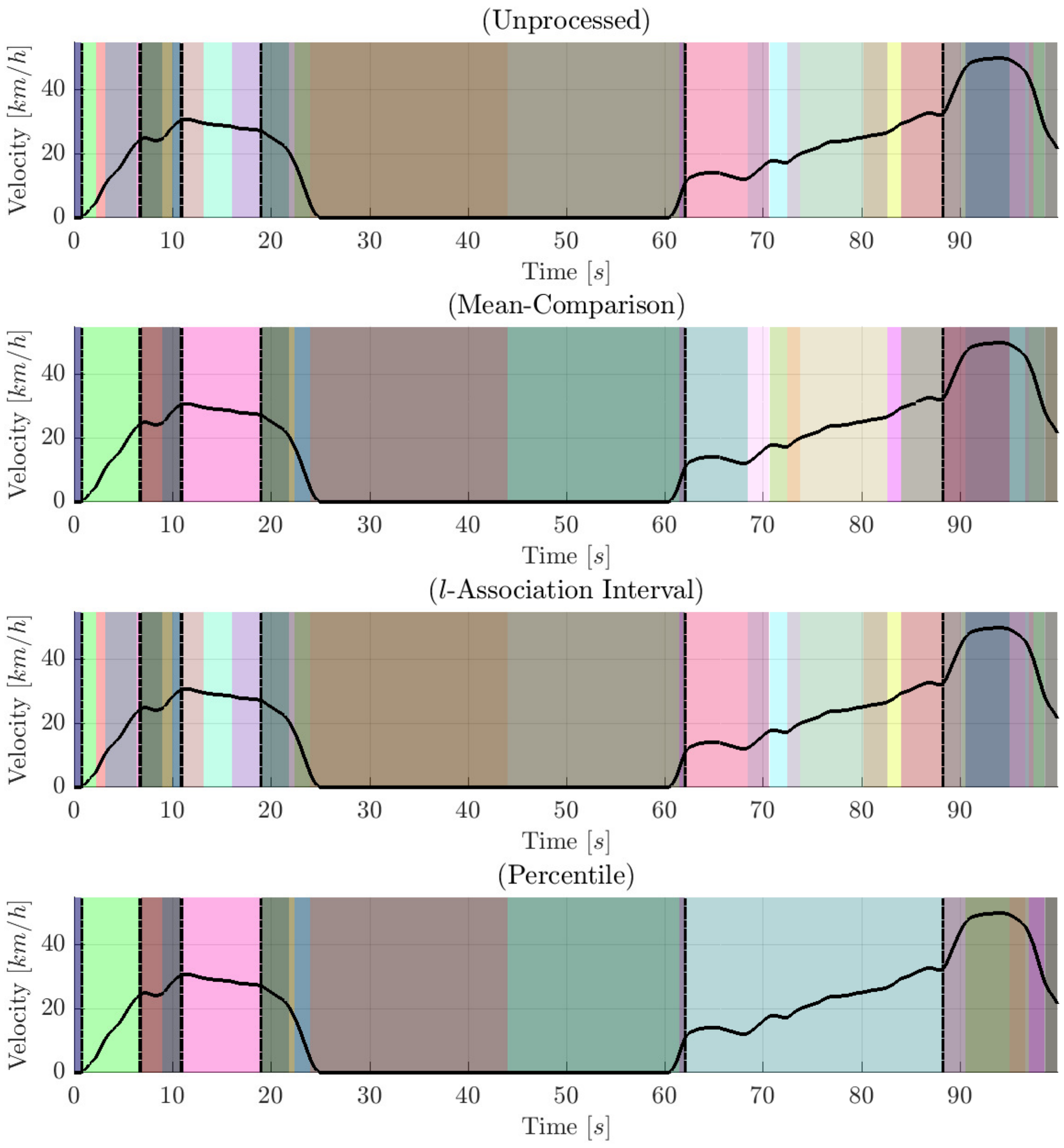


Figure 3.10: Plot comparing the segmentation post processing methods with unprocessed data (top), Mean-Comparison (top-middle), l-Association Interval (bottom-middle), and Percentile (bottom).

eling driving acceleration behavior using a function fitting technique will be presented before discussing the modeling results in Section 3.4.

The SWAB segmentation algorithm and the percentile based combination method divide a training velocity trajectory into segments that display similar behavior with respect to both acceleration and velocity. Following the segmentation, the next step in the behavior identification process is to classify the different segments into the three phases (acceleration, braking, and constant velocity) by calculating and reviewing the velocity and acceleration characteristics of each segment. The following characteristics define the three phases in our driver modeling process:

- **Acceleration Phases:** A minimum of 50% of the acceleration values in the segment must be greater than a_{thresh} (similar to the +% metric in Section 3.1.4)
- **Braking Phases:** A minimum of 50% of the acceleration values in the segment must be less than $-a_{\text{thresh}}$ (similar to the −% metric in Section 3.1.4)
- **Constant Velocity Phases:**
    1. Segment cannot consist of more than 10% of acceleration values with a magnitude greater than a_{thresh}
    2. Segment must be longer than t_{min}
    3. Segment must have an average velocity greater than v_{min}

In addition to classifying the segments, it is also necessary to ensure that the behavior of our driver is not being influenced by a leader car. This is achieved by analyzing the frontal-radar data present in the test-data and comparing it with the current velocity of our vehicle. In this work, a leader car is defined to influence the behavior of our driver when the distance between the leader and follower cars is under five times the current velocity in m/s, or $d_{\text{theoretical}}$ as shown in (3.28).

$$d_{\text{theoretical}} = 5 * v_{\text{vehicle}} \tag{3.28}$$

Figure 3.11 shows the results of the original segmentation, the classification, and finally the removal of the segments affected by a leader vehicle on a training velocity trajectory. The classification algorithm does an effective job at identifying the correct behaviors with the acceleration phases shown in green, braking in red, and constant velocity in blue. The blacked out sections in the third plot are segments where the driver was influenced by a leader car.

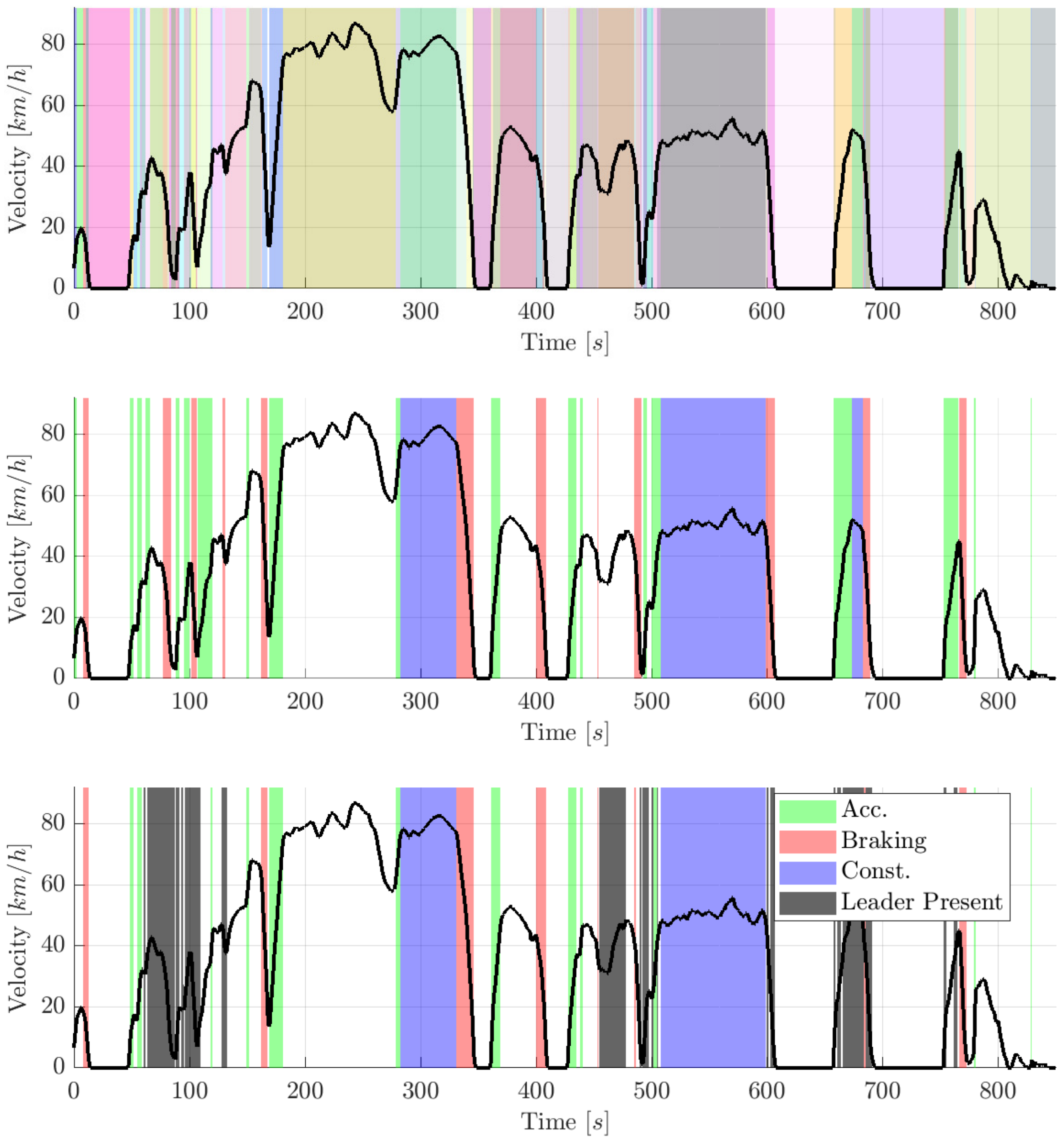


Figure 3.11: Results of SWAB segmentation (top), Results of the acceleration classification (middle), Removal of leader-affected segments (bottom).

Table 3.6: Function Fit Regression Residual Errors

		Regression	
		Inverse	Exponential
Data	Average	117.9	79.1
	All	$3.09e3$	$1.76e3$

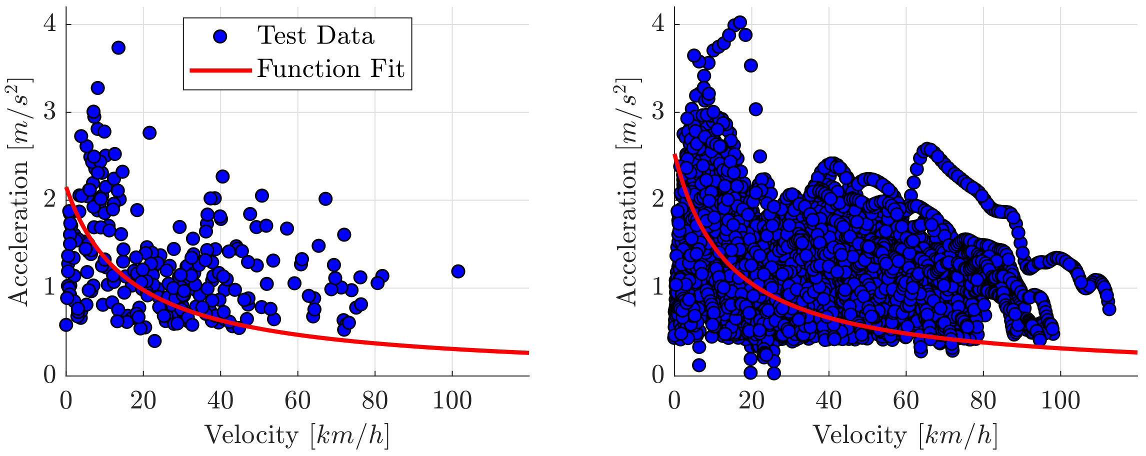
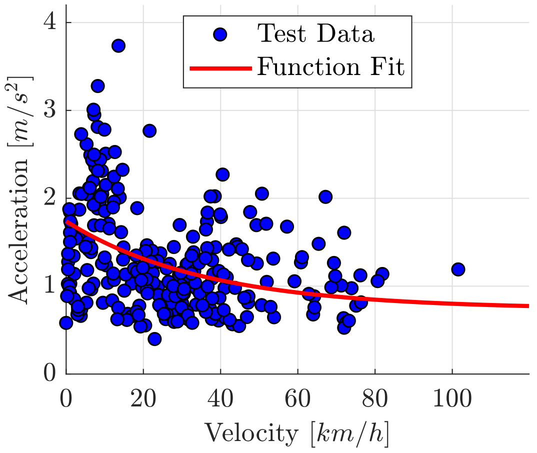


Figure 3.12: Function fit results using the inverse function. Average velocity vs. acceleration (left) and All velocity vs. acceleration data (right) are shown.

The next step in the function fit process is to calculate the mean velocity and acceleration of each segment in like groups, and plot acceleration as a function of velocity. Using a constrained nonlinear regression method, as presented in Chapter 2.2.2, we fit a curve to the data that accurately models the driver's acceleration behavior, $a(t)$, as a function of velocity, $v(t)$.

As an example, the positive acceleration phase function fit regressions using two different nonlinear regression equations over data from 265 individual training acceleration phases are shown in Figures 3.12 and 3.13. The regression equations $a(t) = 1/(p_1 v(t) + p_2)$ and $a(t) = e^{-p_1 v(t)} + p_2$ are shown in the inverse and exponential plots, respectively. The residual errors for the two regressions, shown in Table 3.6, show that the exponential function demonstrates a more accurate fit of the data than the inverse in the average case and the opposite is true in the case where all of the data is used. However, very little correlation is present in the plots using all velocity and acceleration data and will be disregarded. Therefore, because of the higher accuracy of the fit, the inverse regression function will be used in the final evaluation of the driver acceleration behavior predicted presented in Section 3.4.

The model resulting from the function fit method is acceleration, $a(t)$, as a function of velocity, $v(t)$, where $p_{1,2}$ are the model coefficients given by the regression. In order



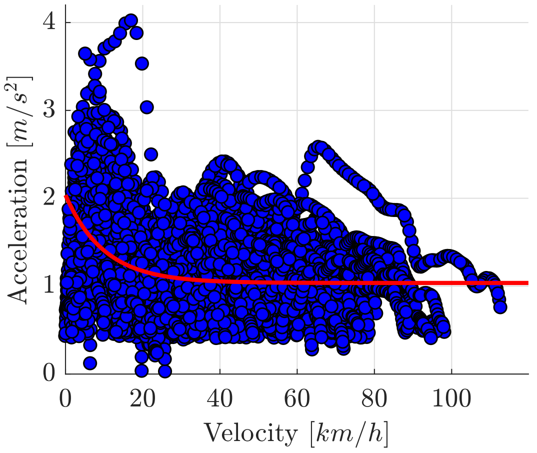


Figure 3.13: Function fit results using the exponential function. Average velocity vs. acceleration (left) and All velocity vs. acceleration data (right) are shown.

to use this model for velocity trajectory prediction as a function of time, we solve for $v(t)$. Rewriting acceleration as the time derivative of velocity, we have a first order differential equation shown in (3.29).

$$a(t) = \frac{dv(t)}{dt} = \frac{1}{p_1 v(t) + p_2} \tag{3.29}$$

Using separation of variables, we solve the ordinary differential equation resulting in a quadratic,

$$\frac{p_1}{2} v^2(t) + p_2 v(t) + c = t, \tag{3.30}$$

for which the positive root can be taken,

$$v(t) = \frac{-p_2 + \sqrt{p_2^2 - 2p_1 c + 2p_1 t}}{p_1}. \tag{3.31}$$

The integration constant, c, is found by solving the original quadratic at $t = 0$, resulting in $c = -\frac{p_1}{2} v^2(0) - p_2 v(0)$ where $v(0)$ is the initial velocity. In conclusion, the prediction function requires only a desired $v(0)$ and time duration to predict a velocity curve.

3.3 System Identification

In addition to the function fit regression method presented in Section 3.2, a system identification approach is used to construct a driver acceleration behavior model using classified data from the pattern recognition methods presented in Section 3.1. The fundamentals of system identification processes, including step responses and transfer functions in the Laplace domain, are presented in Chapter 2. With regard to this work, we notice that the typical velocity trajectory in an acceleration phase most closely resembles the critically-damped response type shown in Figure 2.4 in a positive acceleration case. The inverse is true in the negative case. Therefore, we model the driver behavior using a system with critically-damped properties.

3.3.1 Velocity Trajectory Modeling

A typical positive acceleration phase in a velocity trajectory exhibits similar behavior to a step response from an $(n > 1)$ order system. In this section, we explore two methods for identifying the time constants, T_{c1} and T_{c2} or T_c, and the gain K for the $(n > 1)$ - order systems shown in (3.33). The first method, known as the Transition-Point method, fits an $(n > 1)$-order response to the velocity curve cased on parameters like the dwell and compensation times. The second, the Rise-Time method, fits an $(n = 2)$ - order system to the curve using the rise time. Both of the methods are explained in detail and an example comparison is given at the end of the section. While the focus of this section is the modeling of positive acceleration behavior, all presented methods are applicable to the negative case by observing the behavior in reverse time.

$$Y(s) = \frac{K}{(T_c s + 1)^n} U(s) \tag{3.32}$$

$$Y(s) = \frac{K}{(T_{c1} s + 1)(T_{c2} s + 1)} U(s) \tag{3.33}$$

Note that in both of the methods, the input function $U(s)$ is taken to be the unit step function: $\frac{1}{s}$ in the Laplace domain.

The Transition-Point Method

Of the system identification methods presented in this section, the Transition-Point is the more complex of the two. The resulting system can be either second order with two different time constants or an $(n > 1)$-order system with identical time constants [4]. The identification process begins by calculating three characteristics of the curve in question: the gain or amplitude K, the effective dead-time T_U, and the build up or compensation time T_G. Calculating the gain value is a simple matter of finding the maximum value, or amplitude, of the trajectory. The two time constants are found by finding the Transition-Point, or point of maximum slope, in the response and laying

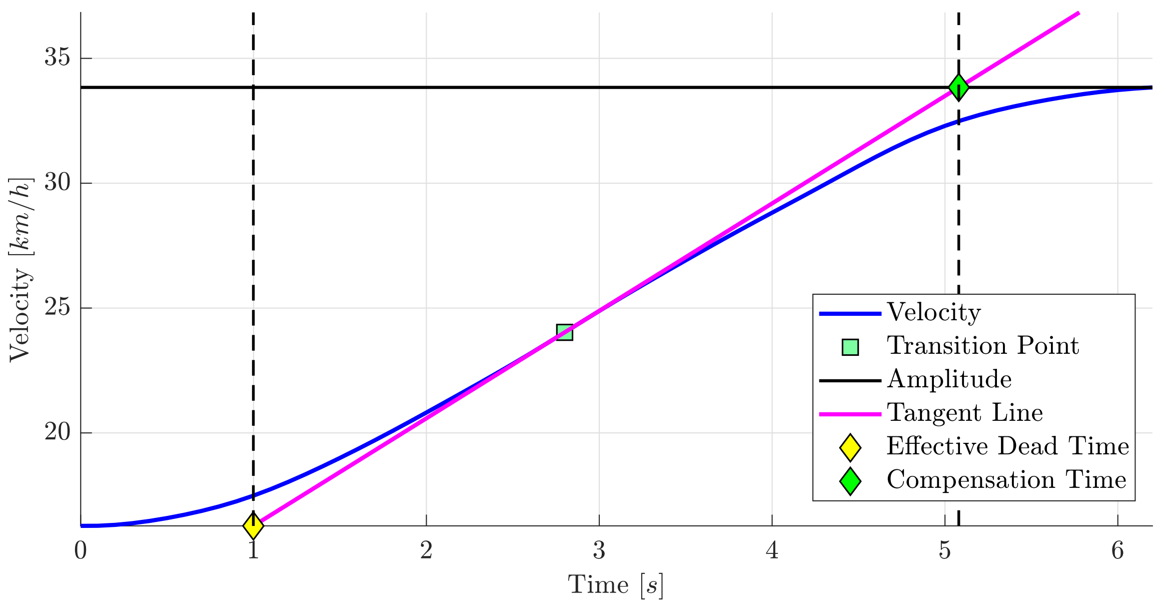


Figure 3.14: Response characteristics essential to the Transition-Point system identification method.

a tangent line on that point in the curve. The effective dead-time is found at the point where this tangent line meets the x-axis and the compensation time is found by subtracting T_U from the point in time where the tangent line meets the desired input value. These values and the Transition-Point are shown in Figure 3.14.

Once the characteristics are found, we use the ratio $\frac{T_\mathrm{U}}{T_\mathrm{G}}$ to define the type of system and the time constant(s) associated with it. In the case that $\frac{T_\mathrm{U}}{T_\mathrm{G}} < 0.1036$, then our system will be second order with two unique time constants, as shown in (3.34).

$$Y(s) = \frac{K}{(T_{\mathrm{c}1}s+1)(T_{\mathrm{c}2}s+1)} \tag{3.34}$$

These time constants, $T_{\mathrm{c}1}$ and $T_{\mathrm{c}2}$, are found using the ratio between them, $\alpha = \frac{T_{\mathrm{c}2}}{T_{\mathrm{c}1}}$, and the following equation:

$$\frac{T_\mathrm{U}}{T_\mathrm{G}} = \left(\frac{\alpha^{\frac{\alpha}{1-\alpha}}(\alpha\ln(\alpha)+\alpha^2-1)}{\alpha-1} - 1 \right) \tag{3.35}$$

The variable α is found by solving (3.35), then $T_{\mathrm{c}1}$ is found using (3.36), leaving a simple calculation for finding $T_{\mathrm{c}2}$.

$$T_\mathrm{G} = T_{\mathrm{c}1}\alpha^{\frac{\alpha}{\alpha-1}} \tag{3.36}$$

In the case that the ratio between the effective dead and compensation times is greater than 0.1036, then the system order is $n > 1$ with identical time constants. The order is calculated using Table 3.7 and the time constant value is found using (3.37).

Table 3.7: Table used for selecting system order using the relationship between T_U and T_G using the Transition-Point Method.

n	2	3	4	5	6	7	8
$\frac{T_\mathrm{U}}{T_\mathrm{G}}$	0.1036	0.2180	0.3194	0.4103	0.4933	0.5700	0.6417
$\frac{T_\mathrm{G}}{T_\mathrm{U}}$	9.6489	4.5868	3.1313	2.4372	2.0272	1.7543	1.5583

$$T_\mathrm{c} = \frac{T_\mathrm{G}}{\frac{(n-2)!}{(n-1)^{n-2}} e^{n-1}} \tag{3.37}$$

After identifying the system order, amplitude, and time constant(s), it is still necessary to transform the system back to the time domain so that the system can be simulated and validated against test data. In the second order case with two unique time constants shown in (3.34), the inverse Laplace transform is:

$$\mathcal{L}^{-1}\left[\frac{K}{(T_\mathrm{c1}s+1)(T_\mathrm{c2}s+1)}\right] = K\left(1 + \frac{T_\mathrm{c1}}{T_\mathrm{c2}-T_\mathrm{c1}} e^{-\frac{t}{T_\mathrm{c1}}} + \frac{T_\mathrm{c2}}{T_\mathrm{c1}-T_\mathrm{c2}} e^{-\frac{t}{T_\mathrm{c2}}}\right), \tag{3.38}$$

and in the $(n > 1)$-order case with identical time constants, the result is:

$$\mathcal{L}^{-1}\left[\frac{K}{(T_\mathrm{c}s+1)^n}\right] = K\left(1 - \sum_{i=1}^{n} \frac{t^{i-1}}{(i-1)!T_\mathrm{c}^{i-1}} e^{-\frac{t}{T_\mathrm{c}}}\right) \tag{3.39}$$

Both (3.38) and (3.39) can be used to predict velocity trajectories given an initial velocity, $v(0)$, and a time duration, resulting in:

$$v(t) = K\left(1 + \frac{T_\mathrm{c1}}{T_\mathrm{c2}-T_\mathrm{c1}} e^{-\frac{t}{T_\mathrm{c1}}} + \frac{T_\mathrm{c2}}{T_\mathrm{c1}-T_\mathrm{c2}} e^{-\frac{t}{T_\mathrm{c2}}}\right) + v(0), \tag{3.40}$$

$$v(t) = K\left(1 - \sum_{i=1}^{n} \frac{t^{i-1}}{(i-1)!T_\mathrm{c}^{i-1}} e^{-\frac{t}{T_\mathrm{c}}}\right) + v(0) \tag{3.41}$$

The next section presents the less complex Rise-Time system identification method followed by a comparison and evaluation.

The Rise-Time Method

The Rise-Time system identification method fits a second order transfer function of the form shown in equation (3.42) using the rise time. Traditionally, the rise time of a response equals the time required by the system to reach 95% of the desired input value; however, due to the shape of the velocity trajectories used in this identification process, we assume that 95% value is the final value in the velocity curve, making the rise time, t_{r}, the total time duration of the curve itself.

$$Y(s) = \frac{K}{(T_{\mathrm{c}}s+1)^2} \tag{3.42}$$

The second order transfer function has two unknowns: the gain or amplitude value K and the time constant T_{c}. K is defined as the maximum value in the velocity curve and the time constant T_c is calculated using:

$$T_{\mathrm{c}} = \frac{1}{5}t_{\mathrm{r}} \tag{3.43}$$

Finally, the second order transfer function is transformed back to the time domain, where it can be used to predict velocity trajectories given a desired initial velocity and time duration.

$$v(t) = \mathcal{L}^{-1}\left[\frac{K}{(T_{\mathrm{c}}s+1)^2}\right] + v(0) = K\left(1 - e^{-\frac{t}{T_{\mathrm{c}}}} - \frac{t}{\mathbf{T_c}}e^{-\frac{t}{T_{\mathrm{c}}}}\right) + v(0) \tag{3.44}$$

Selection of a System Identification Method

We now present a comparison between the Transition-Point and Rise-Time system identification approaches. The modeling methods have been presented for use with a single velocity curve, but we are interested in modeling generalized behaviors. Therefore, it is necessary to take the single curve system identification results and combine them into a unified model. Because the data offers a broad variety of velocity curves with different starting and ending velocities, we classify each individual curve into groups for which unique models can be built. Classification groups are defined based on starting and ending velocities using the values that lie between either $0-50$, $50-100$, or $100-300\frac{km}{h}$ resulting in a total of nine groups as shown in Table 3.8.

Table 3.8: System identification classification groups.

v_{final} $(\frac{km}{h})$ \ $v_{initial}$ $(\frac{km}{h})$	$0-50$	$50-100$	$100-300$
$0-50$	-	-	-
$50-100$	-	-	-
$100-300$	-	-	-

Using a sufficiently large training data set, we build models for each specific group using both identification methods. The Rise-Time method involves calculating the time constant for each individual curve, taking it's average, and using the result in the final model. The Transition-Point, however, introduces another degree of freedom: a non-constant system order. To compensate for this, we find the most common order in the classification group, set this as the final model order, and use only the time constants associated with that order to calculate the final group constants. In the case that the most common order is 2, we calculate if there are more second order systems with unique or identical time constants and use only the corresponding results to calculate the constant(s) for the final model. To compare the system identification approaches, acceleration phases are taken from a test data set, classified in the groups shown in Table 3.8, and predicted using the corresponding model. Figure 3.15 shows a case where both approaches show similar velocity curve predictions. Here, the Transition-Point method matches the test data almost perfectly for the first half of the curve and then falls below towards the end. The Rise-Time method shows opposite behavior, being above the test data until the very end where the curves align. Disregarding the small error, both approaches simulate the test data relatively well and demonstrate similar overall curvature. Figure 3.16 shows a different case where due to the erratic driving behavior, neither simulated curve captures the general shape of the test data.

The strengths of the system identification approaches are that the initial and final velocities are often simulated very well by the models, but at the risk of possibly losing the mid-phase behavior. In order to test the effectiveness of the system identification methods, the same modeling and simulation process was performed on 89 individual velocity curves selected from the data using the heuristic data classification method detailed in Section 3.1.1 and the sum-squared error between the test data and the two system identification methods was calculated. The results, as shown in Figure 3.17, show that, on average, the Transition-Point and Rise-Time methods show very similar results with the error between the actual and simulated curves being low in most cases.

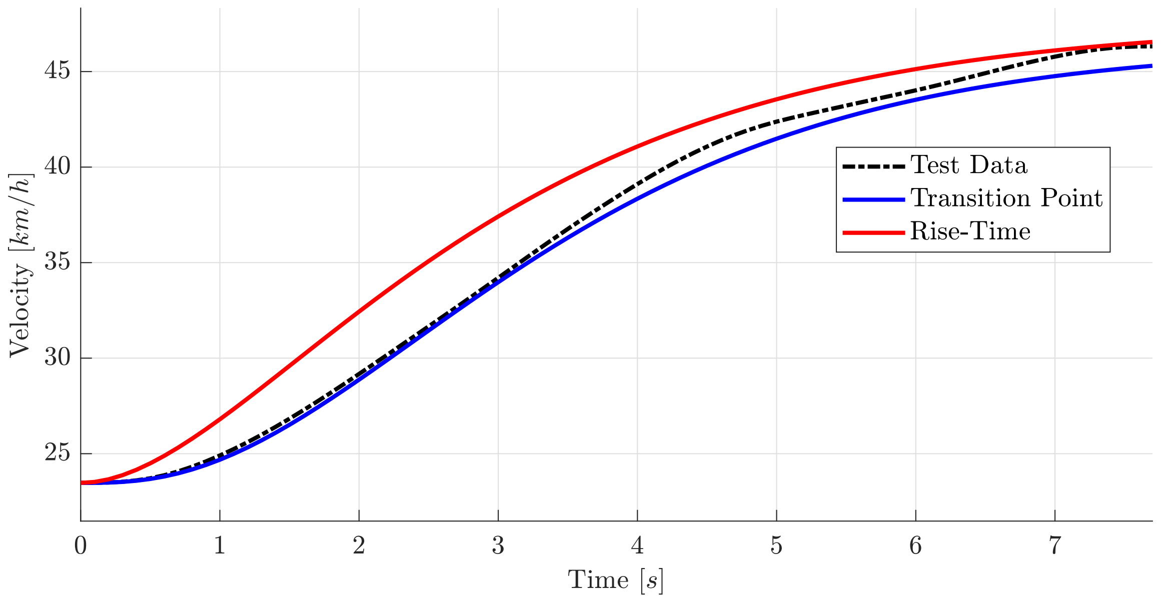


Figure 3.15: The system identification results of the Transition-Point and Rise-Time methods exhibiting similar behavior to a test trajectory.

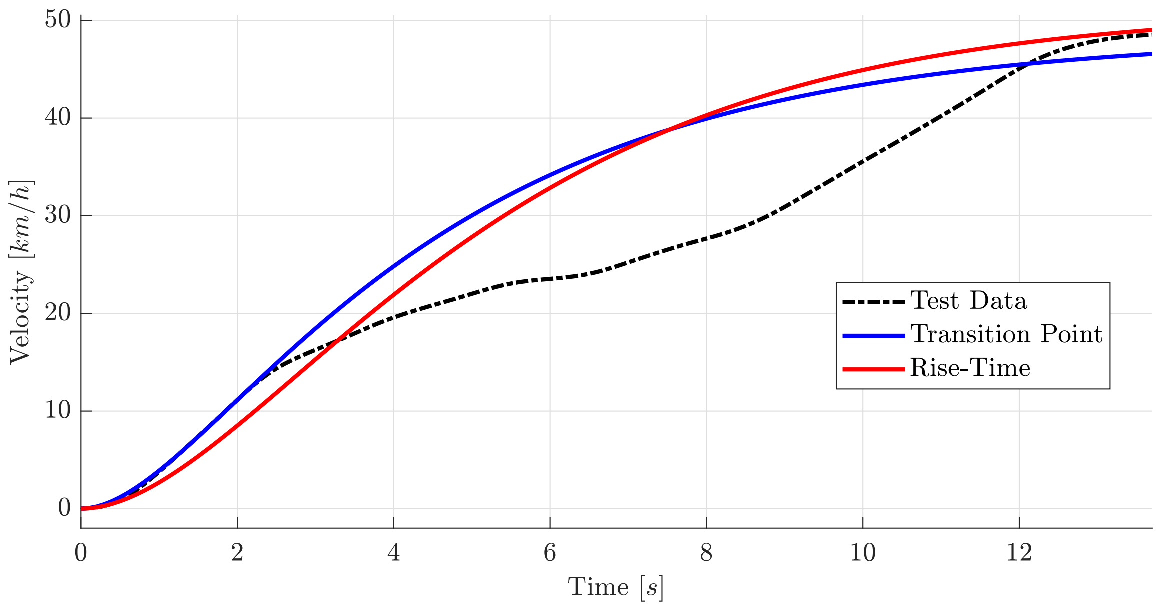


Figure 3.16: The system identification results of the Transition-Point and Rise-Time methods exhibiting dissimilar behavior to a test trajectory.

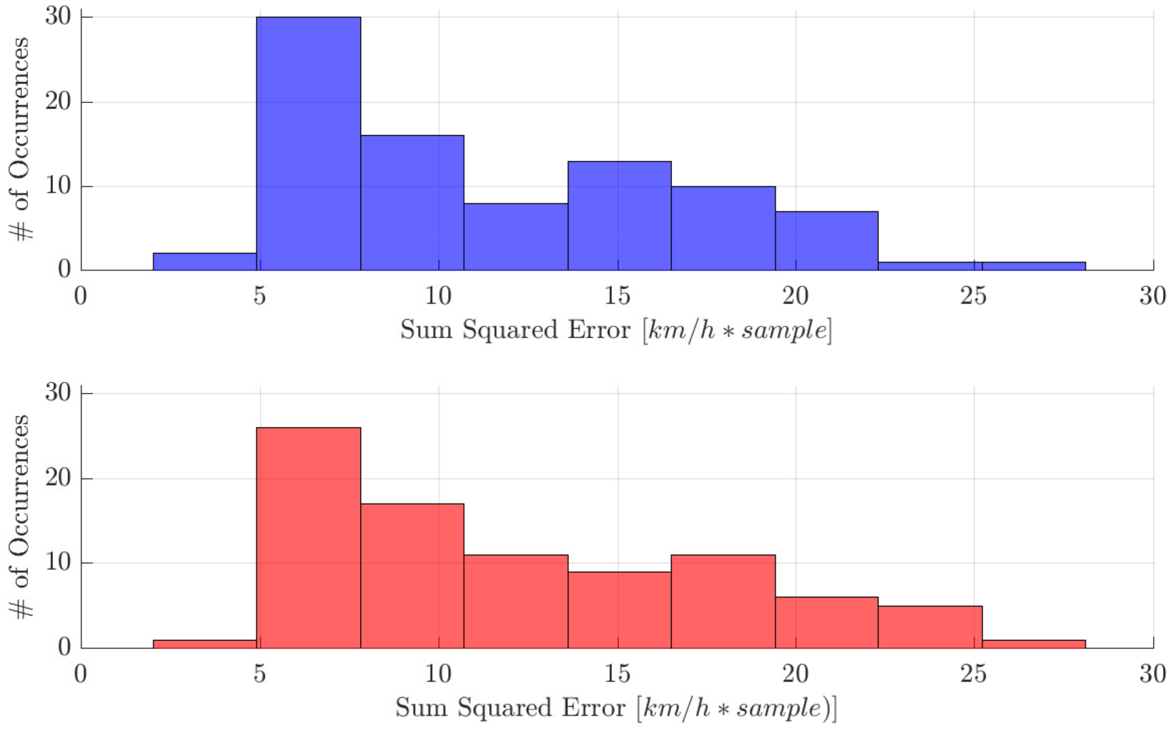


Figure 3.17: Sum-squared error results from the modeling and simulation of 89 test velocity trajectories. Transition-Point (top) and Rise-Time (bottom).

Because the results of each methods are comparable, we choose a prediction method based on real-world practicality. With this in mind, transition-Point identification method often creates situations where test data is not used in final model creation due to the fact that all predicted models do not necessarily have the same order. This unnecessary wastefulness is unpractical and inefficient in situations where test-data is limited. Therefore, in the sense of efficiency, the Rise-Time method is more effective than the Transition-Point and will be used in the prediction of driver acceleration behavior in Section 3.4.

The final section of this chapter will focus on the modeling of driver acceleration behaviors using both the system identification approach discussed in this section and the regression based method discussed in Section 3.2.

3.4 Evaluation of Driver Modeling Methods

The previous sections of this chapter discussed two methods used in predictive model building. In Section 3.2, three formal statistical time-series segmentation methods are presented, compared, and the most effective of the three, the SWAB method, is used in conjunction with an inverse function fit regression to create a model. Section 3.3 presents two system identification based modeling approaches. Both methods are used with training data to build a model, their ability to predict new trajectories is compared, and the more efficient of the two, the Rise-Time method, is selected to be used in the prediction of velocity trajectories. This section presents the application of the segmentation and system identification modeling approaches to the focus of this work: the modeling of driver behavior with a specific focus on positive acceleration behaviors.

The data used in this work comes from twelve test drives conducted with the same driver using the electric B-Class presented in Chapter 2. As previously mentioned, each modeling technique requires training data to build a model and testing data to evaluate it. In order to utilize the test drive data as efficiently as possible, each data-set will serve eleven times as training data and once as testing data. For example, if data set 1 is the current testing data-set, then data sets $[2, \ldots, 12]$ are used as training sets to build a model that will be used to predict behaviors present in data-set 1. The process continues sequentially through the series until all data sets have been used to build and evaluate models. This type of model building and evaluation allows the assessment of not only the model qualities, but also the effectiveness and reliability of the model building methods. A method capable of reliably constructing accurate models is perfect for a driver assistance system application.

Prior to modeling, the following processing operations are conducted on the training and testing data-sets. First, the required data, namely velocity, is extracted from the data set, re-sampled at a frequency of 10Hz, and filtered using a Hanning filter with a length of 5 samples. Acceleration is calculated from the velocity data using the differentiation method described in Equation (3.21) and the acceleration phases are found using the heuristic data classification method. Note that only acceleration phases that are not influenced by a leader car, as described in Figure 3.11, are accepted. After the initial data processing procedure, the training data sets are used in combination with the regression and system identification based modeling methods to create two driver acceleration models as a function of time which are used to predict the velocity trajectories present in the test data set. The entire modeling method is repeated in a loop for all twelve data sets, resulting in 12 regression models, of the form shown in Equation (3.31), and 12 system identification ones, of the form shown in (3.44). Velocity trajectories from the corresponding testing data sets are predicted using each model by plugging in the initial velocities and time durations to each respective model for each respective curve.

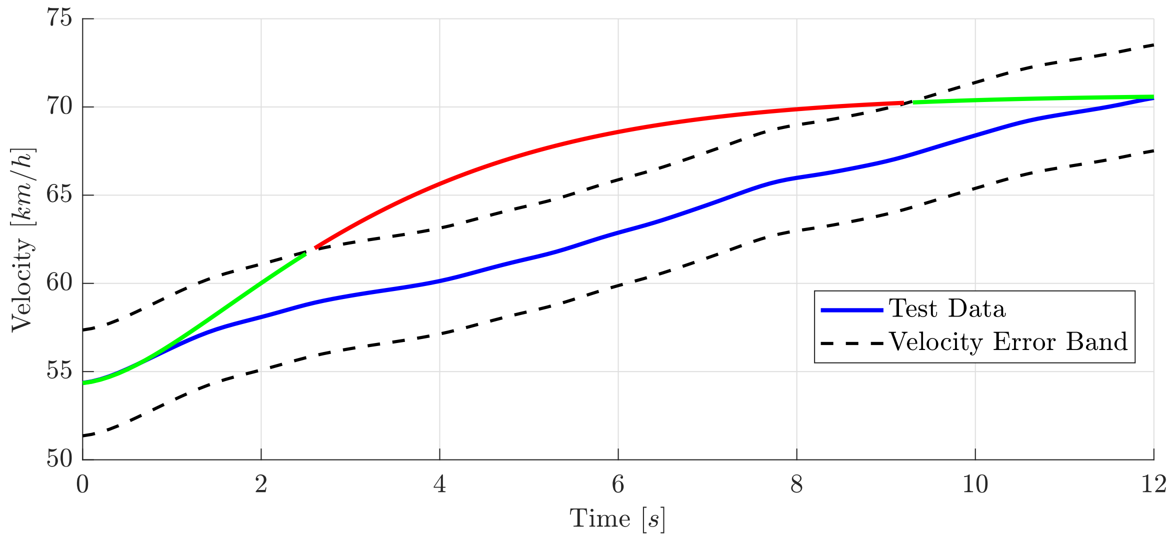


Figure 3.18: Demonstration of the band error evaluation measure.

The predicted velocity trajectories are evaluated with respect to their similarity to the original test trajectory using three metrics: energy consumption, sum-squared error, and the velocity band error. Energy consumption is used because of its direct relationship to acceleration during driving. Similar energy consumptions imply similar driving cycles. Economical behavior is also a desirable trait for use in a range-extension system like the ones the driver models would be applied to. The sum-squared error and the velocity band error are also chosen as comparison metrics because they directly measure similarity between real and predicted curves. The band error is defined as the percentage of the trajectory time duration that the predicted curve falls within an $x\ \frac{km}{h}$ - wide-band around the test curve. An example is shown in Figure 3.18 using a band width of $3\frac{km}{h}$ where the samples of the predicted trajectory that fall within the band are shown in green and the ones that do not are shown in red.

The results displayed in the following figures are calculated using the SWAB and segment combination parameters shown in Sections 3.1.3 and 3.1.5 for the function fit method. For the System Identification method, the heuristic classification parameters shown in 3.1.1 and the classification groups shown in Table 3.8 are used.

Using the longitudinal vehicle model presented in Chapter 2, the energy consumption of the predicted trajectories is calculated and compared in Figure 3.19. Energy consumption has a direct mathematical relationship to driving velocity and acceleration and is, therefore, an ideal metric to measure curve similarities. Economical behavior is also desirable in a model that has a potential use in a range-extension

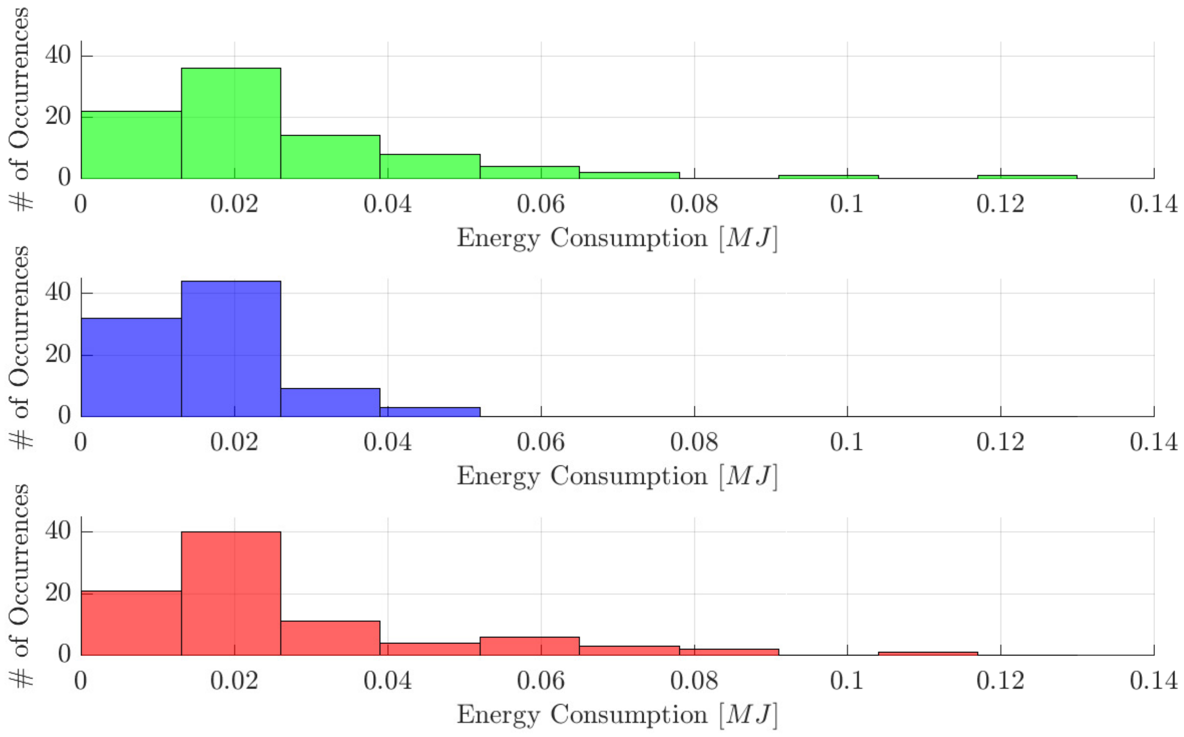


Figure 3.19: Energy consumption comparison between test data (top), function-fit (middle), and system identification (bottom) predictions.

driver assistance system. The first figure shows the total energy consumption for all curves for the test data and curves predicted using the function fit and system identification methods. While the general shape of the plots is similar, with the majority of the data located around the $0.02MJ$ mark, it is important to note that the function fit method energy consumption is much more consistent and has almost no cases with a consumption over $0.04MJ$ whereas the system identification predictions and test data show similar energy consumptions spread out over a wider range. This is confirmed in Figure 3.20, which shows the differences between the energy consumptions of the test data and the two respective prediction methods. Here, positive ΔE values represent instances where the predicted curve consumed less energy than the test data and vice versa. With an average consumption difference of $-3.01e-7MJ$, the system ID predictions consumed slightly more than the test data whereas the function fit method, with an average consumption difference of $9.3e-3MJ$, consumed slightly less. Because the magnitudes here are small and represent only a small fraction of a gallon of fuel (1 gallon $= 132MJ$), each method demonstrates a sufficient level of accuracy with respect to fuel consumption.

Figure 3.21 shows the velocity band error percentage between the prediction meth-

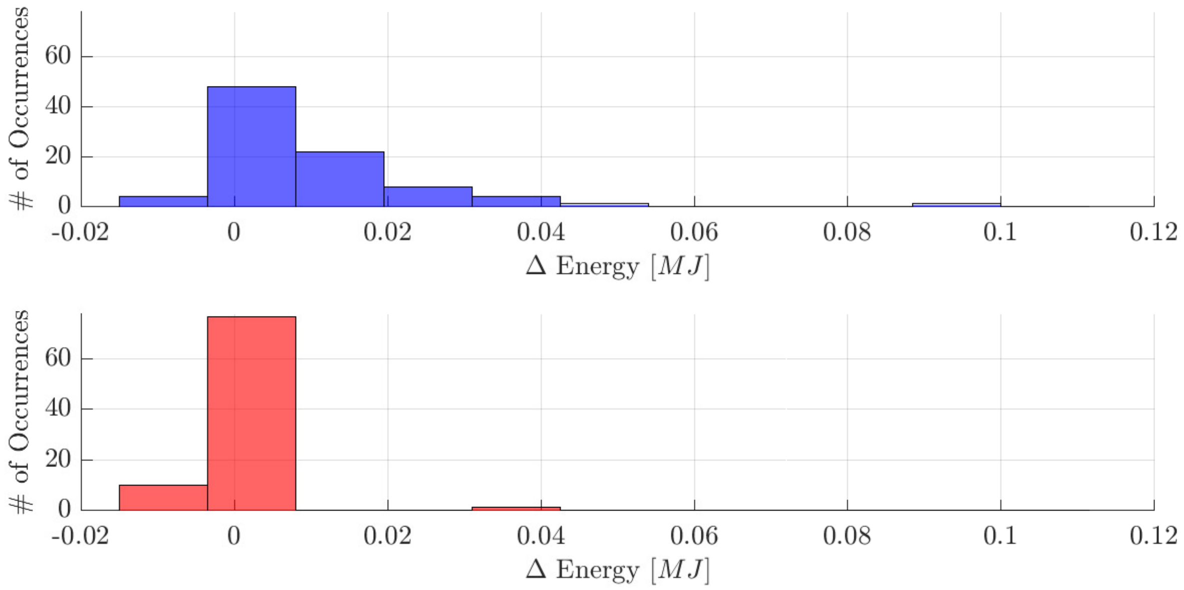


Figure 3.20: Difference in energy consumption comparison between test data (top), function-fit (middle), and system identification (bottom) predictions.

ods and the test data. Here, the general shape of the histogram plots are similar between the function fit and system identification methods with the average error being 44.3% and 48.8% for each approach. The plots show two concentrations in the band error data: one near zero and another grouping around the $60-70\%$ mark. This means that both methods are either almost entirely accurate or relatively inaccurate in predicting the exact shape of the test data curves.

The sum-squared error normalized by curve length is shown in Figure 3.22 with the average errors being 0.6 and $0.54 km/h * sample$ for each respective method. Similarly to the velocity band error percentage, the sum squared errors have averages and histogram shapes that are similar to one another with the highest concentration of points occurring around the 0.5 mark meaning that both methods have a relatively low overall error when compared directly to the test data. The length of the curves used range between 50 and 300 samples, meaning that over the entire length, the error is acceptably low.

Finally, Figures 3.23 and 3.24 show all test data curves compared with the predicted curves from the function fit and system identification methods. Test data curves are depicted in solid lines and predicted curves are shown with dotted lines. While a visual comparison is difficult in this case, a few trends in the data are recognizable. The function fit method, for example, tends to predict curves with consistent slopes that do not always reach the final velocity value of the test data. In contrast,

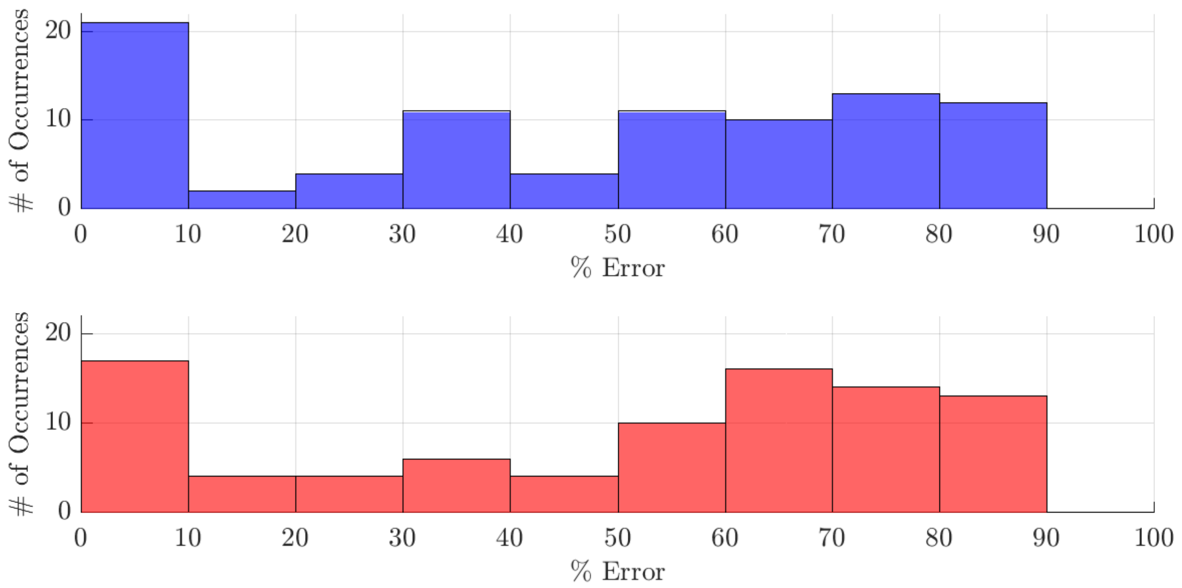


Figure 3.21: Percent band error results comparison between the function-fit (top) and system identification (bottom) prediction methods.

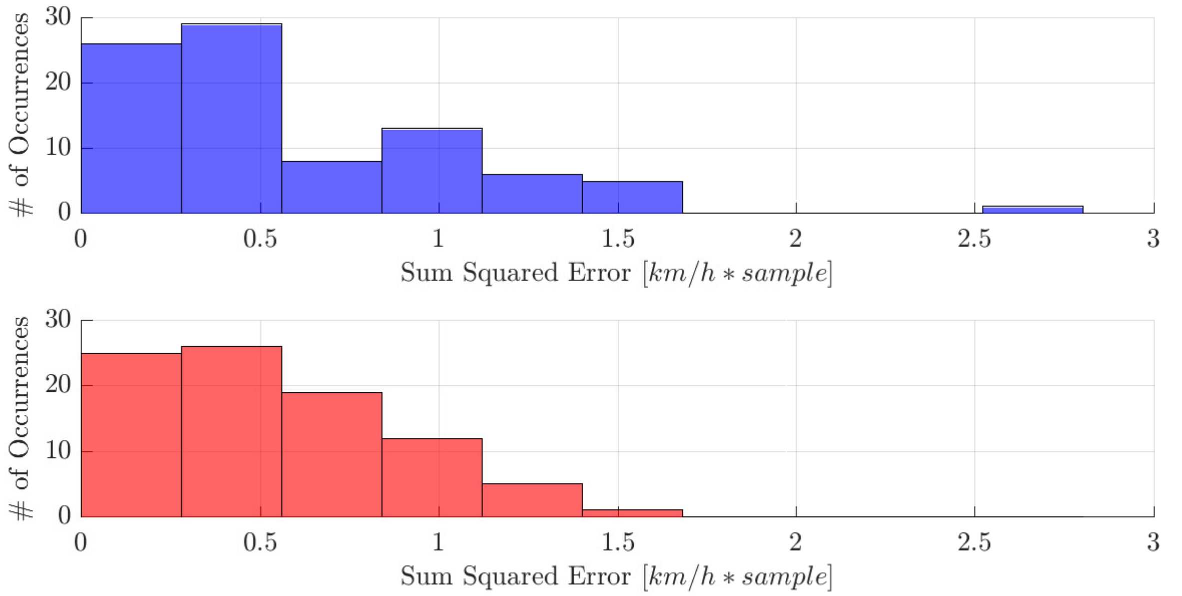


Figure 3.22: Sum-squared error results comparison between the function-fit (top) and system identification (bottom) prediction methods.

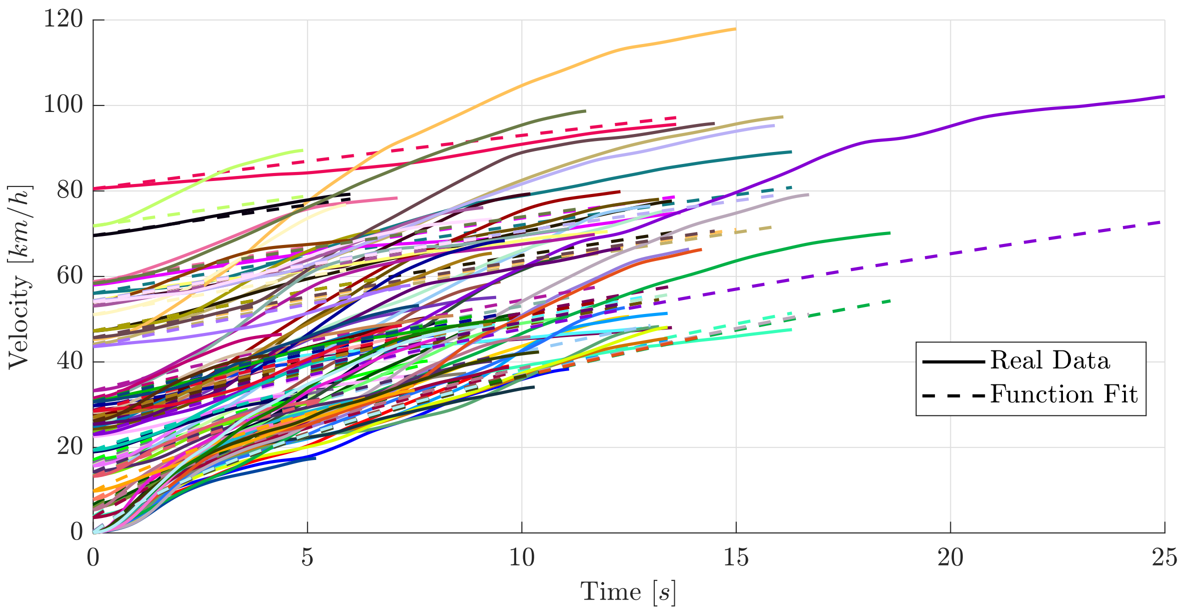


Figure 3.23: Velocity trajectory prediction results using the function-fit method.

the system identification method predictions have a higher acceleration variation and nearly always reach the final test data velocity value.

When evaluated and directly compared to the real-world test data, both the function fit and system identification based approaches to modeling and predicition of velocity behavior demonstrate accurate, efficient approaches. The overall energy consumption of each method is similar, with the system identification method requiring slightly more energy in most cases, the velocity band error demonstrates an accurate prediction with a probability of around 50%, and sum-squared average between the test and predicted curves is, on average, relatively low. In conclusion, both methods can be used to accurately model and predict driver acceleration behavior. The system identification approach in particular is able to reliably achieve desired accelerations with curve shapes similar to those observed in the test data.

In this chapter we have investigated and evaluated two driver behavior modeling methods based on pattern recognition. A training data trajectory is broken down into homogeneous segments and the behavior in these segments is used to build a model that exhibits similar behavior. In the next chapter we investigate an approach that does not require breaking down a trajectory before a model can be built: the prediction of entire velocity curves using a machine learning approach.

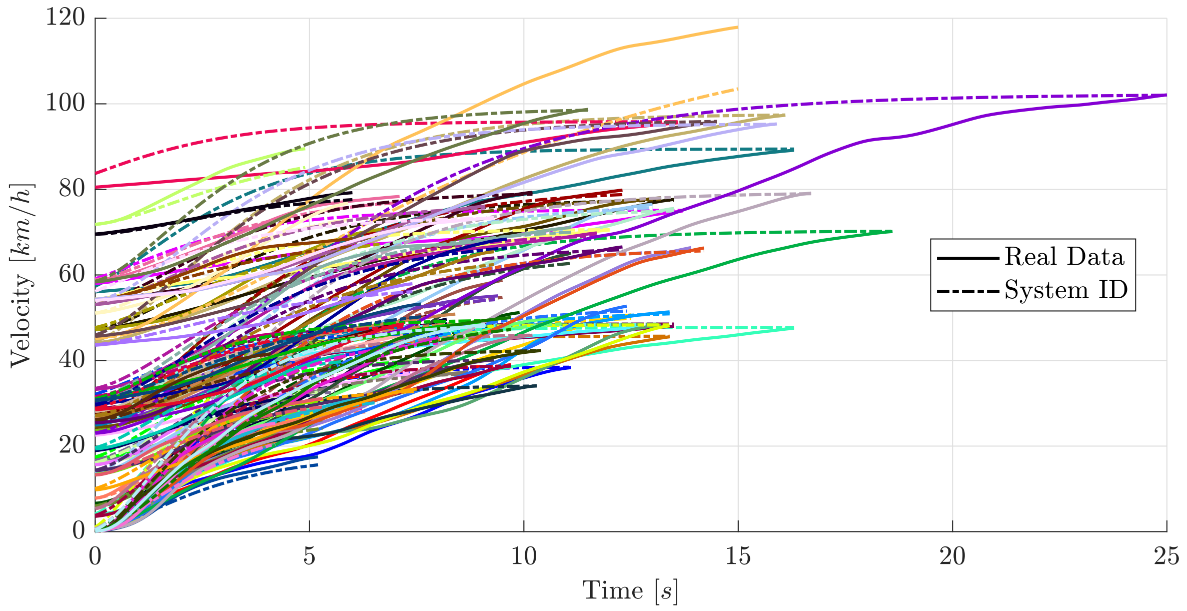


Figure 3.24: Velocity trajectory prediction results using the system identification method.

CHAPTER 4
MODELING USING MACHINE LEARNING

Machine learning is a broad tool capable of classifying and building regression models from large data sets. A brief background of the subject can be found in Chapter 2. In the specific application to analyzing driving behavior, machine learning has been used to classify driver aggressiveness using pattern recognition [64], identify unique drivers from a group [53], identify aggressive driving events for insurance purposes [34], and identify driver characteristics like drowsiness, inattentiveness, or sobriety [45]. Where the literature is focused on the identification of certain driving behaviors and events, this work focuses on using machine learning for constructing a model capable of behavior prediction. Specifically, we use machine learning to build a model by analyzing an entire velocity trajectory at once, eliminating the pattern recognition and data analysis procedures required in Chapter 3.

In this chapter, support vector machine algorithms available in the Matlab *Machine Learning* toolbox are used to build a driving cycle model capable of predicting accurate velocity behavior with respect to a specific driver with a potential application in building driver acceptance of a DAS. A short introduction to machine learning, support vector machines, and their application to regression-type problems is presented before discussing their specific use in this work. The results of the several machine learning training algorithms are presented and evaluated in the final section of this chapter.

4.1 Machine Learning Fundamentals

There are two main problem categories in the field of machine learning known commonly as classification and regression. Classification problems can be thought of as pattern recognition with the goal of grouping similar data into homogeneous categories. Regression problems instead have the goal of recognizing patterns in the relationships between variables in order to construct a function capable of mimicking and predicting behavior. The prediction of driving velocities can, in this sense, be classified as a regression type machine learning problem and will therefore be the focus in this work.

Similar to the model building process discussed in Chapter 3, machine learning regression problems typically have two phases: training and testing. As the name implies, the first phase consists of using a machine learning algorithm to construct, or train, a model using data with known behavior. The algorithm is fed predictor and response data, which is used to identify patterns within the predictor variables that result in the given responses. Once the algorithm has constructed a sufficient model, it's ability to predict response behavior using new predictor data can be tested and evaluated against the real responses. If the constructed algorithm accurately predicts the response data, the process is complete. If not, the training phase must be repeated using either more training data, different prediction variables, or a combination of the two.

There exist a number of possible training algorithms suited to regression-type problems including regression trees, neural networks, and support vector machines. Of the available algorithms, support vector machines (SVM's) are utilized in this work due to their relative ease of use [30]. This section presents a brief background and introduction to support vector machine learning algorithms and training them in both the linear and nonlinear cases. These are then tested for velocity trajectory prediction in Section 4.2.

4.1.1 Support Vector Machines

Support vector machines are based on statistical learning theory pioneered in the late 1960's, the 70's, and the 80's by Vapnik and Chervonenkis in [60] and [62]. In short, SVM's build optimally located hyperplanes that serve as divisions between various types of data and the name "Support Vector Machines" stems from the vectors that make up the dividing hyperplanes. In the most basic form, the goal is to construct learning algorithms for pattern recognition in a classification sense by creating a hyper plane to serve as a divide between two types of data. This section provides an overview of the fundamentals of support vector machine classification beginning with the linear separable case. More in depth explanations of the theory can be found in both [61] and [31].

Given n training data $(x_1, y_1), \ldots, (x_n, y_n)$ samples with inputs $x \in \mathbb{R}^m$ and output $y \in \{1, -1\}$, we assume that the data can be linearly separated by hyperplanes, d, with normal vector to that hyperplane, w, and coefficient b,

$$d = (w \cdot x) - b \tag{4.1}$$

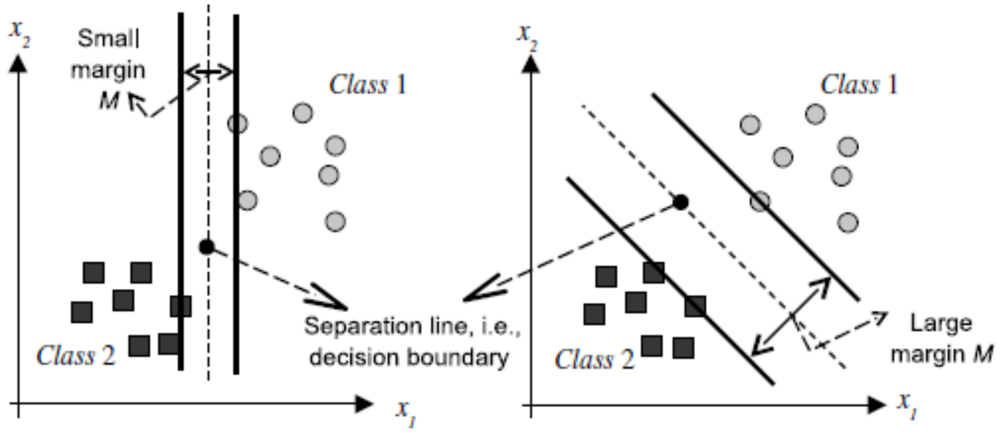


Figure 4.1: Two possible separating hyperplanes in the two-dimensional case [31]. The fit on the right is considered a better fit than on the left due to the larger margin of separation.

The hyperplane is said to have separated the training data vectors if they are successfully classified without error and the distance to the closest vector is maximized. Two examples with varying quality are shown in Figure 4.1. In the case that the data is strictly separable, d_i can be defined to have a value greater than $+1$ if $y_i = 1$ and a value less than -1 for $y_i = -1$, defined here as $y_i\left[(x_i \cdot w) - b\right] \geq 1$. The hyperplane generation problem can be constructed as an quadratic programming (QP) problem with the goal is minimizing the norm of the plane normal vector:

$$\Psi(w) = \frac{1}{2}(w \cdot w), \tag{4.2}$$

subject to the $y_i\left[(x_i \cdot w) - b\right] \geq 1$ constraint. This optimization problem can be solved by finding the saddle point of the Lagrangian, shown in (4.3), with respect to the KKT conditions.

$$L(w, b, \alpha) = \frac{1}{2}\left(w \cdot w\right) - \sum_{i=1}^{n} \alpha_i \{\left[(x_i \cdot w) - b\right] y_i - 1\} \tag{4.3}$$

From the KKT conditions, we derive that the optimal hyperplane is a linear combination of the ith training data components and Lagrange multiplier, α_i. The vectors for which the Lagrange multiplier is nonzero are the ones that form the "support" vectors.

$$w = \sum_{i=1}^{n} y_i \alpha_i x_i \tag{4.4}$$

Plugging this result into (4.3), we arrive at the optimization problem, (4.5), only dependent on the various α_i's that can be solved using a constrained optimization solver.

$$\min_{\alpha} P(\alpha) = \min_{\alpha} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j (x_i - x_j) - \sum_{i=1}^{n}\alpha_i \tag{4.5}$$

subject to the constraints:

$$\alpha_i \geq 0 \quad \forall i \tag{4.6}$$

and

$$\sum_{i=1}^{n}\alpha_i y_i = 0 \tag{4.7}$$

However, it is often the case that the training data in question is not linearly separable. Vapnik, et al. proposed a solution by introducing the slack variables, ξ_i, to both the cost function and constraints shown in (4.2) allowing a "soft" separating margin between the training data classes ([12, 52]) resulting in:

$$\Psi(w, \xi) = \frac{1}{2}(w \cdot w) + C\left(\sum_{i=1}^{n}\xi_i\right) \tag{4.8}$$

subject to

$$y_i\left[(x_i \cdot w) - b\right] \geq 1 - \xi_i, \quad \forall i \tag{4.9}$$

where the constant C is a trade off variable between a larger margin and a small number training samples that breach it. Similarly to the purely separable case, this optimization problem can be solved using the Lagrangian subject to the corresponding KKT conditions, resulting in the same quadratic programming problem shown in (4.5) with a slightly modified constraint that creates an upper bound C on the Lagrange multipliers shown in (4.10).

$$0 \leq \alpha_i \leq C, \quad \forall i \tag{4.10}$$

Furthermore, SVM's can be generalized to use non-linear classification functions that must be linear with respect to parameters like w and b, but are not required to have a linear relationship to the training data [9]. Following from the result shown in (4.4), the nonlinear classification functions are defined directly as a function of the Lagrange multipliers. Shown in (4.11), K is the kernel function that compares the training data vector x_j with some input vector x_i. In the original publication, Boser proposes kernels like a potential or radial basis function but other functions, like those available in the Matlab *Machine Learning* toolbox, include quadratic, cubic, and Gaussian functions.

$$d = \sum_{j=1}^{n} y_j \alpha_j K(x_i, x_j) - b \tag{4.11}$$

By plugging this in to the original QP, we arrive at the final, non-linear, non-separable SVM quadratic programming problem, (4.12), as function of the Lagrange multipliers *alpha*. Note that if the kernel function K is simply the linear difference between vectors x_i and x_j, we arrive back at the linear case in (4.5).

$$\min_{\alpha} P(\alpha) = \min_{\alpha} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{n} \alpha_i \tag{4.12}$$

subject to the box constraint

$$0 \leq \alpha_i \leq C, \quad \forall i \tag{4.13}$$

and the linear constraint

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{4.14}$$

When solved, the optimization problem and corresponding constraints presented in (4.12) yield a trained algorithm capable of solving classification-type problems. In the next section, these results are generalized to the regression-type machine learning problem.

4.1.2 Support Vector Machine Regression

Vapnik generalizes the classification approaches to regression-type problems for the linear and nonlinear-type loss functions in [61]. According to the reference, a linear SVM regression will take place if, first, the regression estimation is defined as a set of linear functions $f(x, \alpha) = (w \cdot x) + b$, similar to those shown in (4.1), that seek to mimic the response variables y as a function of the predictors x. Second, the problem must be defined as one of risk minimization with respect to an ε-intensive loss function $|\cdot|_\varepsilon$, where

$$|y - f(x, \alpha)|_\varepsilon = \begin{cases} 0 & \text{if} \quad |y - f(x, \alpha)| \leq \varepsilon \\ |y - f(x, \alpha)| - \varepsilon & \text{otherwise} \end{cases} \tag{4.15}$$

A graphic depiction of the loss function is presented in Figure 4.2. Finally, risk must be minimized according to the SRM principle which guarantees a sequence of risks converges asymptotically to the smallest risk. The theorem and proof can be found in Chapter 4.2 of [61].

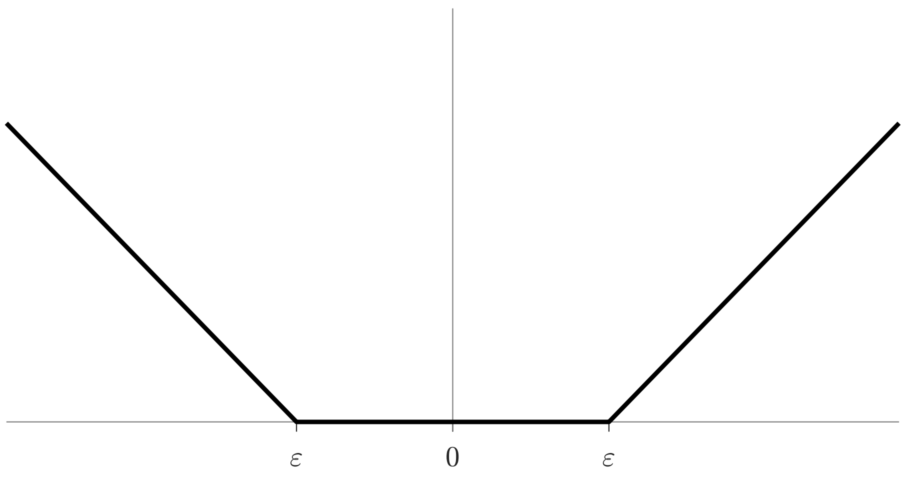


Figure 4.2: The ε-intensive loss function.

Provided training data $(x_1, y_1), \ldots, (x_n, y_n)$, where x_i are predictors and y_i are responses, and satisfied preconditions defined above, we can solve the empirical risk minimization problem with respect to a linear ε-intensive loss function,

$$R_{\text{emp}}(w, b) = \frac{1}{n} \sum_{i=1}^{n} |y - (w \cdot x) - b|_{\varepsilon}, \tag{4.16}$$

by treating it as a QP optimization by minimizing both the vector product of w and the C-weighted sums of the slack variables ξ_i and ξ_i^* shown in (4.17).

$$\Psi(w, \xi^*, \xi) = \frac{1}{2}(w \cdot w) + C \left(\sum_{i=1}^{n} \xi_i^* + \sum_{i=1}^{n} \xi_i \right) \tag{4.17}$$

subject to the constraints,

$$\begin{aligned}
y_i - (w \cdot x_i) - b \quad & \leq \varepsilon + \xi_i^*, \quad \forall i, \\
(w \cdot x_i) + b - y_i \quad & \leq \varepsilon + \xi_i, \quad \forall i, \\
\xi_i^* \quad & \geq 0, \quad \forall i, \\
\xi_i \quad & \geq 0, \quad \forall i
\end{aligned}$$

Similarly to the classification problems in Section 4.1.1, the QP can be solved using the Lagrangian and the corresponding KKT conditions, resulting in the regression coefficients as a function of Lagrange multipliers α and α^*:

$$w = \sum_{i=1}^{n} (\alpha_i^* - \alpha_i) x_i, \tag{4.18}$$

where the multipliers can be found by solving the constrained optimization problem shown in (4.19).

$$\min_{\alpha,\alpha^*} P(\alpha, \alpha^*) = \min_{\alpha,\alpha^*} \varepsilon \sum_{i=1}^{n} (\alpha_i^* + \alpha_i) - \sum_{i=1}^{n} y_i(\alpha_i^* - \alpha_i) + \frac{1}{2} \sum_{i,j=1}^{n} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)(x_i \cdot x_j) \tag{4.19}$$

subject to the constraints,

$$\sum_{i=1}^{n} \alpha_i^* = \sum_{i=1}^{n} \alpha_i,$$

$$0 \leq \alpha_i^* \leq C, \quad \forall i,$$
$$0 \leq \alpha_i \leq C, \quad \forall i$$

The results for the linear case can be further generalized to use a kernel-based regression equation, where, instead of the linear function, $f(x, \alpha) = (w \cdot x) + b$, we have functions of the form:

$$f(x; v, \beta) = \sum_{i=1}^{n} \beta_i K(x, v_i) = b \tag{4.20}$$

where β_i is a constant defined as $\alpha_i^* - \alpha_i$, $\forall i$, v_i is a vector, and K is a kernel function as defined in Section 4.1.1. Using the same quadratic optimization approach in (4.17), we arrive at the kernel function based optimization problem shown in (4.21).

$$\min_{\alpha,\alpha^*} P(\alpha, \alpha^*) = \min_{\alpha,\alpha^*} \varepsilon \sum_{i=1}^{n} (\alpha_i^* + \alpha_i) - \sum_{i=1}^{n} y_i(\alpha_i^* - \alpha_i) + \frac{1}{2} \sum_{i,j=1}^{n} (\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j) K(x_i, x_j) \tag{4.21}$$

subject to

$$\sum_{i=1}^{n} \alpha_i^* = \sum_{i=1}^{n} \alpha_i,$$

$$0 \leq \alpha_i^* \leq C, \quad \forall i,$$
$$0 \leq \alpha_i \leq C, \quad \forall i$$

Solving this optimization problem is equivalent to training a machine learning algorithm. The resulting solution is a regression equation capable of mimicking and predicting response data. Note that in the *Matlab* toolbox, validation methods like k-fold cross-validation and hold out are used to improve the regression function. These methods break up training data, solve the SVM optimization for some parts, and use the others to test the fit of the model. More information can be found in [24], [67], and references therein.

In the next section, the results of several trained SVM algorithms are presented and tested in their ability to predict velocity trajectories.

4.2 Trajectory Prediction

The training phase of the machine learning process involves the selection of predictor and response variables followed by training a SVM algorithm as presented in Section 4.1.2. This involves solving an optimization problem that results in a function that predicts the response variables with respect to the predictors. Once training is completed, the testing phase evaluates the capability of the regression function to predict the response associated with the same predictor variables from a new data set and compares the predicted response with a known one. The effectiveness of a machine learning algorithm is highly dependent on the selection of predictor variables used in the training phase and finding the "correct" predictors is rarely straightforward. In this section, we present five different sets of predictor variables, create models using a machine learning SVM algorithm, and compare their abilities to accurately predict driving cycle data. As in the pattern recognition models discussed in Chapter 3, the success of our model will be based on how accurately it can predict velocity behavior, as accuracy is vital to building driver trust in a DAS application.

4.2.1 Prediction Algorithm Training

The focus of this chapter is the modeling and prediction of individual driver velocity behavior. Therefore the selection of our machine learning response variable is obvious: velocity as a function of time $v(t)$. The predictor variables are chosen to most generally represent driving behavior. The first, speed limit $v_{\mathrm{SL}}(t)$, is because it is a variable that is easily measured and plays a significant role in driving behavior. Speed limit changes characterize how drivers accelerate (or decelerate) and constant speed limits characterize constant velocity behavior. The second predictor, distance traveled $d(t)$, is used as a measure to place a driver within speed limit zones.

When considering driving velocity as a function of the speed limit, there are two types of behavior to consider. First, how a driver behaves when the speed limit is constant and, second, when there is either a positive or negative change. In order for a machine learning training algorithm to distinguish these relationships, the predictor variables must provide the correct information. We structure our predictors by considering the relationships between driving velocity and speed limit in these two cases. In the first case, the relationship is simple; velocity behavior with respect to a constant speed limit requires only knowledge of the current velocity, speed limit, and the difference between the two. The second, however, is more complicated because velocity changes are now a function of the changing speed limit. More specifically, we need to know how, if the driver begins to accelerate/brake before a limit change or after, that behavior is unique to a specific change, and the nature of the specific acceleration/deceleration.

With this information in mind, we present five successive prediction variable structures used in conjunction with velocity as the response. Each prediction variable structure is used to train a model that is tested, visually inspected, and used as feedback to improve the results in the next trial. A formal evaluation and comparison of all methods is presented in the final subsection.

Trial 1

The predictors in the first trial are structured to inform the machine learning algorithm where the vehicle is located at every point in time with regard to the past and future speed limits. Typically, each response variable sample in a training set has a corresponding set of predictor variables. Because our data was recorded as a function of time, we have a response-predictor pair for every time sample. As such, for each v^i, $i = 1, \ldots, n$, in the n-length training set we introduce five specific predictors:

- v^i_{SL} : current speed limit
- $v^i_{\text{SL,next}}$: next-nearest, different speed limit
- $v^i_{\text{SL,prev}}$: first-previous, different speed limit
- $d^i_{\text{SL,next}}$: distance to next-nearest, different speed limit

- $d^i_{\mathrm{SL,prev}}$: distance to first-previous, different speed limit

and further shown in Figure 4.3.

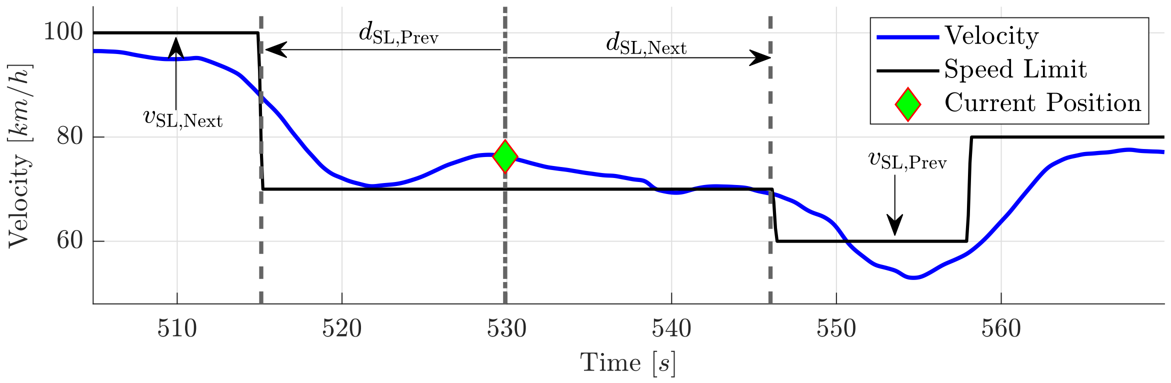


Figure 4.3: The predictor variables used in the first trial training..

In this trial, the algorithm has direct knowledge of the distance to past and future changes in speed limit, as well as magnitudes of the speed limit changes themselves. Additionally, the speed limit values have been set to zero when the vehicle velocity is zero to compensate for stoplights, crosswalks, etc. in areas where the speed limit would otherwise be higher. A training phase using this predictor-response structure was completed on the data shown in Figure 4.4 using a linear support vector machine (LSVM) kernel algorithm combined with a hold out validation. The LSVM algorithm with a hold-out validation was chosen for relative simplicity and ease of use. Another algorithm is tested in Trial 3. The $16,000$ time-sample training data set was conducted by the same driver as the data in Chapter 3 with the same electric B-Class. The trajectory contains a variety of driving situations including multiple acceleration, deceleration, constant velocity, and stand-still phases over varying speed limits, making it an ideal set with which to train the SVM algorithm.

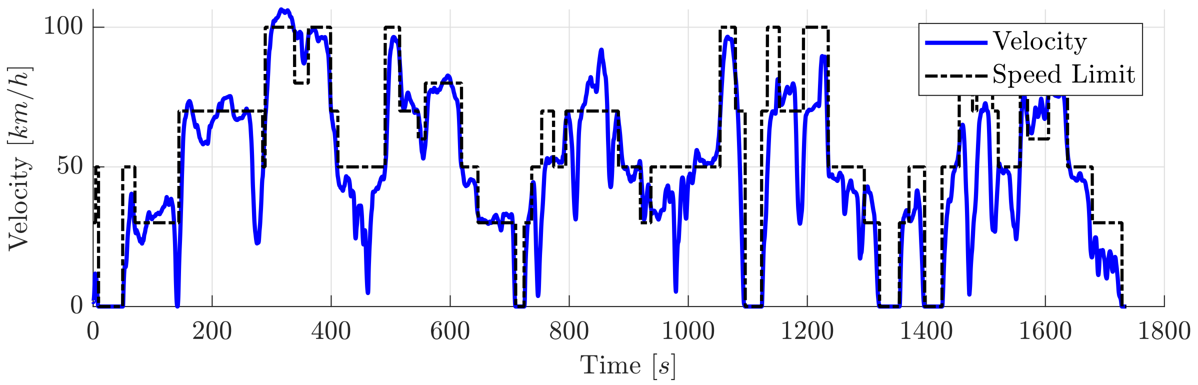


Figure 4.4: Velocity trajectory used to train the machine learning regression function.

The trained SVM algorithm results in a function f, capable of predicting the responses, $\vec{Y}$, as a function of the predictors, $\vec{X}$. In this case, $\vec{Y}$ is a $1 \times m$ vector of predicted velocity values and $\vec{X}$ is a $5 \times m$ matrix of predictor variables where m is the length of the testing data set.

$$\vec{Y} = f(\vec{X}) \tag{4.22}$$

Figure 4.5 shows the results of the testing phase. Similar to the training data, the testing set contains a variety of different velocity and acceleration behaviors and was recorded using the same driver and car as the training set. A visual inspection reveals that while this algorithm is able to predict the relative magnitude of the velocity trajectory, the realistic shape is not present. A possible cause for this discrepancy is that the predictor variables are environment based and contain no information on how the driver performs, causing the predicted trajectory to jump at the same time as the speed limit changes. Another is that the predictor variables provide the training algorithm with information that is possibly too in the past or future to be relevant in current decision making. For example, if a driver is at the halfway point of a $50km$ speed limit zone, the previous and next speed limit zones have no influence on how he/she drives. Therefore, the predictors in this situation may cause the training algorithm to build relationships between variables that have no current correlation. In the next trial, we chose new predictors in an attempt to improve the fit accuracy.

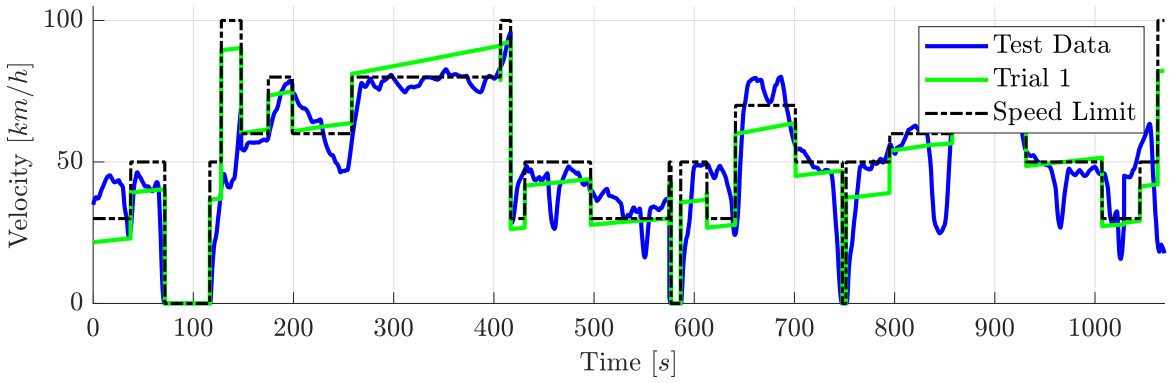


Figure 4.5: Machine learning Trial 1 velocity prediction results.

Trial 2

The Trial 1 results make it obvious that the machine learning training algorithm needs more information than just the relative location of the vehicle to the nearest past and future speed limit changes to predict velocity behavior. Therefore, we discard the previous predictors and introduce new ones to improve the prediction accuracy: the previous velocity sample, to provide the algorithm with exact driver behavior information, and what we refer to as a speed limit window as shown in Figure 4.6.

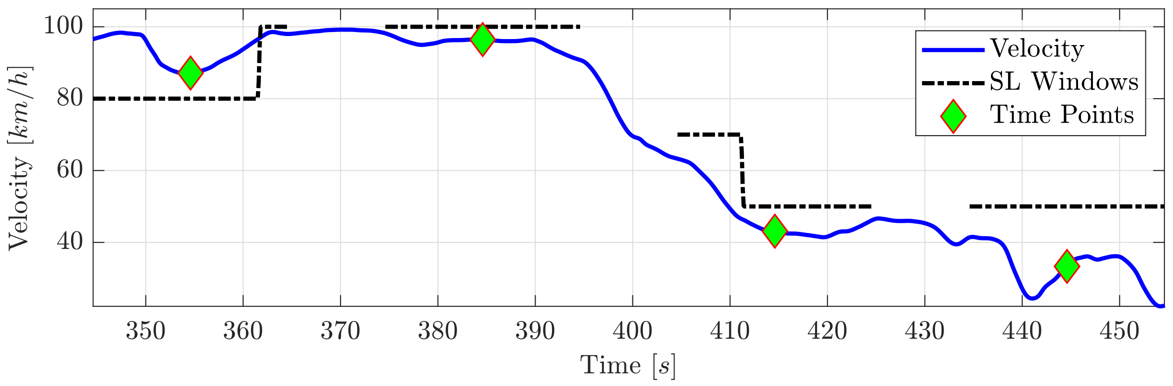


Figure 4.6: Several plotted examples of the speed limit window predictor variable with a $10s$ width.

The speed limit window is an l-second length time window spanning the vehicle's current time position and containing past and future speed limit information. Used as a predictor, this gives the training algorithm speed limit information for each velocity response, while ensuring that the data belongs only to a relevant time frame. With these two predictors, we have $(1/t_\text{s}) * l + 1$ variables instead of the five used in Trial 1, where t_s is the data sampling rate.

Note that the use of a previous velocity value as a predictor value complicates the prediction process by creating a recursive loop. Each future velocity value predicted by the algorithm is dependent on the previously predicted one. Therefore, once the regression function has been trained by the machine learning algorithm, a loop is required to generate the predicted velocity trajectory, with an example step for calculating the kth velocity value shown in (4.23).

$$v^k = f(v^k_{\text{SL,window}}, v^{k-1}) \tag{4.23}$$

Using the same LSVM training algorithm in Trial 1, the same training data set, the same testing set, and the loop described above, a new model was trained with the velocity prediction results shown in 4.7. This algorithm shows a slight improvement over the Trial 1 results, demonstrating more smooth transitions between different velocities in contrast to the harsh jumps present in Trial 1. However, the predicted curve does not match the curvature or relative magnitude of the real data and requires further improvement to accurately predict the test trajectory.

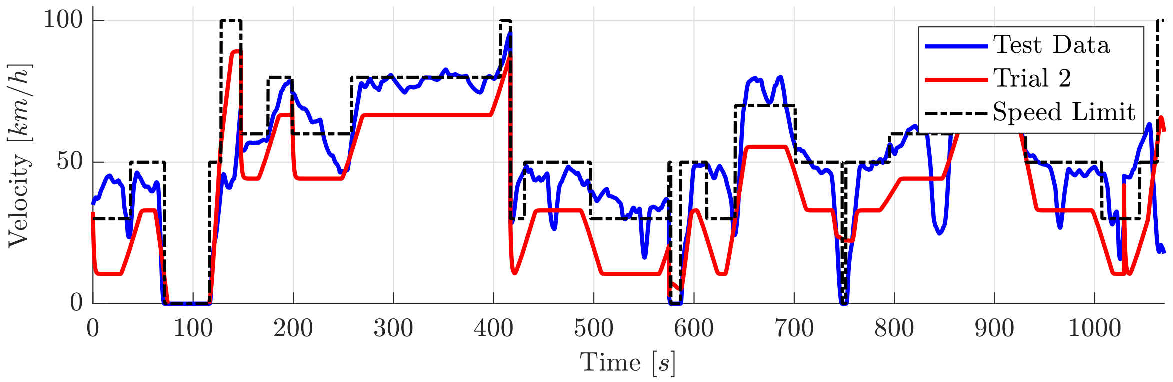


Figure 4.7: Machine learning Trial 2 velocity prediction results.

Trial 3

The result of Trial 2 show an improvement over Trial 1 but fail to accurately predict the curvature, or acceleration behavior, of the test data. In the previous trials, the training algorithm was given information in terms of velocity, in order to predict velocities. Therefore, we introduce driver acceleration behavior to the training data by using a different response variable entirely: $\Delta v^k = v^k - v^{k-1}$. Because the sampling rate of the test data is constant, the term Δv effectively depicts acceleration behavior. Otherwise, the predictor variables consist of the same speed limit window used in Trial 2 and j previous velocity values instead of the single one. By using multiple past velocity values, we provide the training algorithm with more velocity acceleration. As a result of the new predictor-response structure, our training algorithm constructs a function of the form:

$$\Delta v^k = f(v^k_{\text{SL,window}}, v^{k-1}, \dots, v^{k-j}) \tag{4.24}$$

where v^k is calculated by adding Δv^k to v^{k-1}. Used in the same loop fashion as in Trial 2, we predict the test data velocity trajectory using again an LSVM training algorithm. The results are shown in Figure 4.8.

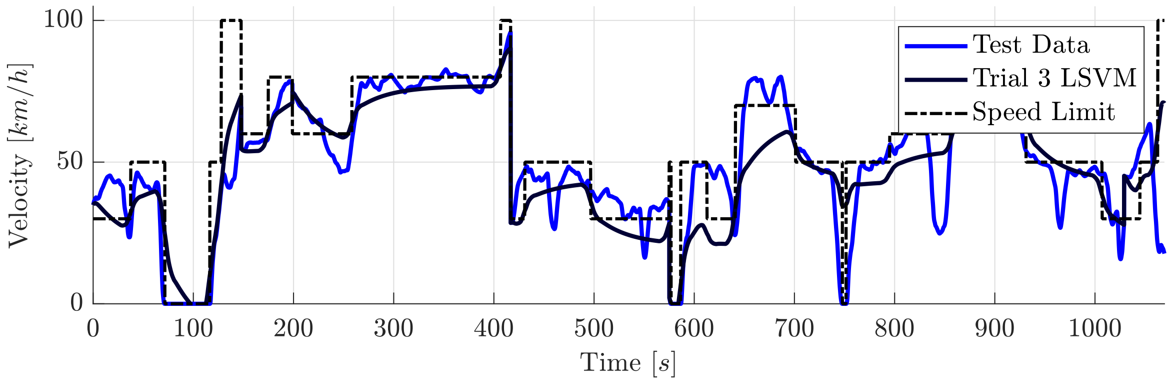


Figure 4.8: Machine learning Trial 3 velocity prediction results.

The results of Trial 3 show another improvement over the previous two attempts. While there are large discrepancies between the relative magnitudes of the real and predicted curves, the SVM algorithm is able to generate curvature that matches the test data reasonably well. Two additional case studies are conducted using the same predictor-response variable structures but with slightly different training methods. First, a quadratic support vector machine (QSVM) training algorithm was chosen to test if a more advanced kernel would be more capable of replicating the complex curvatures and magnitudes present in the test data. The results, shown in Figure 4.9, demonstrate visually that the QSVM is capable of more sporadic behavior and a lower magnitude prediction accuracy than the LSVM.

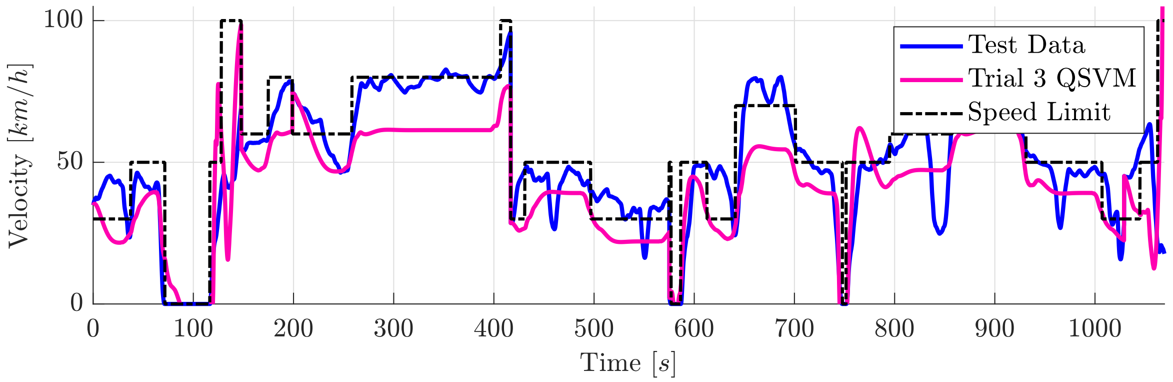


Figure 4.9: Machine learning Trial 4 velocity prediction results.

The second case study uses an LSVM with standardized prediction data. In this work, standardization of data is normalizing scale of the training data in order to make relationships between training data variables more or less pronounced to the machine learning algorithm. A standardized example of the training data shown in Figure 4.4

on the scale $[-1, 1]$ is presented in Figure 4.10. Note that the full curvature of the trajectory is preserved, despite having a smaller scale and negative vertical shift.

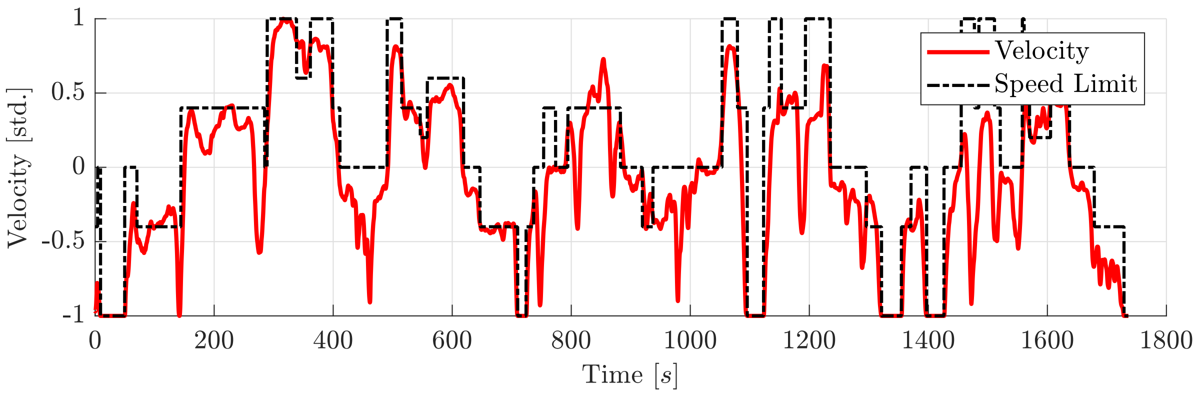


Figure 4.10: Standardized velocity trajectory used to train the machine learning regression equation in Trial 5.

Because the training data is standardized, it follows naturally that the testing data must be similarly scaled to allow the SVM algorithm to predict the data. Figure 4.11 shows predicted testing data that was standardized, used in the SVM algorithm, and rescaled back to the original values. The standardized training algorithm shows similar prediction results to the original LSVM case, but with slightly smoother curvature. A formal evaluation of each trial is presented and discussed in the final section of this chapter.

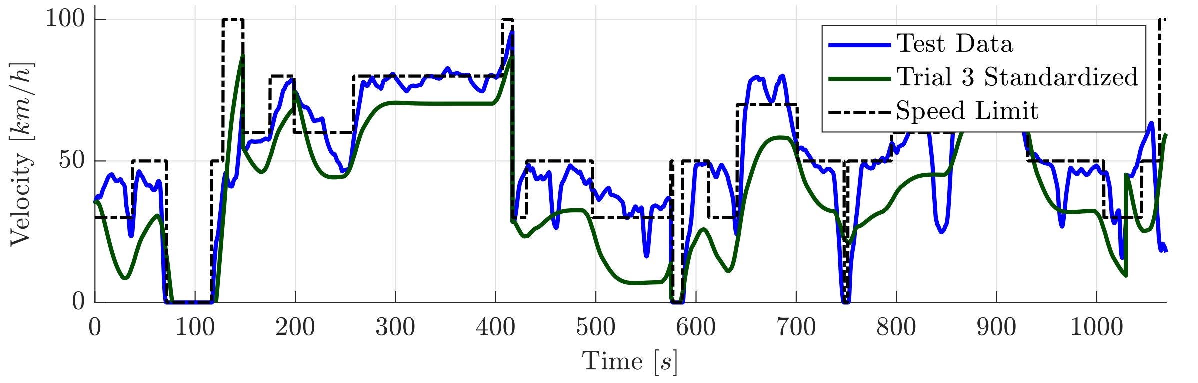


Figure 4.11: Machine learning Trial 5 velocity prediction results.

4.2.2 Prediction Algorithm Evaluation

The five trials presented in Section 4.2.1 are now evaluated in their capability to accurately predict velocity trajectories. Similar to the pattern recognition modeling, predicting behavior using machine learning to build driver trust in a DAS requires a high level of predicted velocity accuracy. Therefore, we choose evaluation metrics like fuel consumption, which generally reflects velocity behavior during driving, to measure general curve prediction accuracy and compare energy use. To directly measure accuracy with respect to the test-data curves, we select the length-normalized sum-squared error (SSE) on both velocity and acceleration in addition to the velocity band error presented in Section 3.4. The evaluation metrics of each trial are shown in Table 4.1 using a velocity band with a width of $5km/h$.

Table 4.1: Results and comparison of the machine learning velocity prediction trials.

Trial#	E (MJ)	ΔE (MJ)	SSE-v $(\frac{km}{h*sample})$	SSE-a $(\frac{m}{s^2*sample})$	Band-Error (%)
1	0.807	0.064	0.471	0.078	95.5
2	0.682	0.118	0.6113	0.022	97.0
3	0.790	0.080	0.356	0.016	77.92
4	0.773	0.097	0.576	0.035	85.4
5	0.657	0.214	0.528	0.017	85.9

First and foremost, we see that are large discrepancies present between the predicted and test trajectories. Where, Trial 5 has the lowest fuel consumption of $0.657MJ$, each trial required less energy than the test data with percent differences ranging between 8 and 33%. Furthermore, we see velocity and acceleration SSE values between $0.4 - 0.61km/s * sample$ and $0.16 - 0.78m/s^2 * sample$, respectively. Note that while each trial demonstrates similar velocity SSE values to the pattern recognition case, the pattern recognition curves have lengths ranging between 50 and 300 samples in contrast to the $10,000+$ samples from this test curve. An average error of $0.5km/h * sample$ over $10,000$ samples results in significant discrepancies and the same holds true for the acceleration data. The band error measurement shows similar inaccuracies with the lowest error percentage at nearly 78%. While Trial 3 demonstrates the most similar behavior to the test curve, the errors present in the fuel consumption, SSE, and velocity band metrics are too large to consider this method for use in a driver assistance system. Achieving acceptable SSE and velocity band error values of $0.05m/s^2 * sample$ and 95% require further development of the training data and algorithms.

CHAPTER 5
CONCLUSION

Range extension driver assistance systems have the potential to improve the fuel consumption of road vehicles by suggesting energy optimized speeds to the driver. However, because the systems have no direct control over the vehicle itself, they are completely dependent on driver trust and acceptance. In this work, we have explored two methods for modeling and predicting driver behavior that could be used to personalize speed commands to suit individual driving styles.

The first method is pattern recognition based and involves the segmentation of a driver's velocity trajectory and drawing information about the behavior from individual segments. Four different time-series segmentation techniques were investigated and the most time-efficient and effective, the SWAB, was selected. Using both function-fit regression and system identification techniques, in conjunction with the SWAB segments, we built and tested velocity models in their ability to predict driving behavior. When compared to predicted trajectories, both methods exhibited high levels of accuracy with regard to the energy consumption and curve shape. Each model shows great potential for improving a driver assistance system.

Where the first method requires breaking down a time-series to analyze behavior and build a model, the second method is able to process the entire trajectory at once. Using an SVM based machine learning approach, we explored five different training and testing techniques to model and predict driver behavior using velocity as a function of the speed limit. While the approach shows promise, the results are not able to sufficiently predict velocity trajectories of a specific driver. More development is required to achieve an algorithm capable of predicting driver behavior with a degree of accuracy suitable to a DAS.

5.1 Future Work

The focus of this work was the identification and prediction of individual driver behaviors, with a specific focus on positive acceleration phases. Future case studies could be conducted using the same techniques to model and predict deceleration, constant velocity, as well as general behavior when driving behind a leader car. Furthermore, both methods presented here exhibit different approaches suitable for generally analyzing, modeling, and predicting behavior in multivariate time-series data. While the machine learning algorithm shows only potential at this point, the pattern recognition based approach can be used “out of the box” to analyze any time-series.

Implementing the identified driver models in a driver assistance system was beyond the scope of this work. However, an implementation of both pattern recognition based models in a simulation environment to test the potential for building driver acceptance and improving fuel economy would be an excellent subject for future work. Positive simulation results would merit the use of the models in a real-world test using the electric B-Class.

REFERENCES

[1] A. A. Abdelgadir and J. Y. Alsawalhi, “Energy management optimization for an extended range electric vehicle,” in *Modeling, Simulation, and Applied Optimization (ICMSAO), 2017 7th International Conference on*, Sharjah, United Arab Emirates, 2017.

[2] N. AbuAli and H. Abou-zeid, “Driver behavior modeling: Developments and future directions,” *International Journal of Vehicular Technology*, 2016.

[3] C. C. Aggarwal, *Data Classification Algorithms and Applications*. CRC Press, 2015.

[4] J. Bechtloff, *Regelungstechnik*. Vogel Buchverlag, 2012.

[5] R. M. Bell and Y. Koren, “Lessons from the netflix prize challenge,” *SIGKDD Explorations*, vol. 9, 2007.

[6] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1957.

[7] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, 1981.

[8] C. Bingham, C. Walsh, and S. Carroll, “Impact of driving characteristics on electric vehicle energy consumption and range,” in *IET Intelligent Transport Systems*, vol. 6, Quebec, Canada, 2012, pp. 29–35.

[9] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT ’92, Pittsburgh, Pennsylvania, USA: ACM, 1992.

[10] J. Broughton and C. Baughan, “The effectiveness of antilock braking systems in reducing accidents in great britain,” *Accident Analysis and Prevention*, vol. 1, 347355, 2002.

[11] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust Region Methods*. Society for Industrial and Applied Mathematics, 2000.

[12] C. Cortes and V. Vapnik, “Support vector networks,” *Machine Learning*, vol. 20, 1995.

[13] C. Dora and M. Phillips, “Transport, environment, and health,” World Health Organization, Tech. Rep., 2000.

[14] D. Dörr, D. Grabengiesser, and F. Gauterin, “Online driving style recognition using fuzzy logic,” in *2014 IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*, Qingdao, China, 2014.

[15] N. R. Draper and H. Smith, *Applied Regression Analysis*. John Wiley & Sons, Inc., 1998.

[16] P. Fancher, Z. Bareket, and R. Ervin, “Human-centered design of an acc-with-braking and forward-crash-warning system,” *Vehicle System Dynamics*, vol. 36, no. 2-3, pp. 203–223, 2001.

[17] J. C. Ferreira, J. de Almeida, and A. R. da Silva, “The impact of driving styles on fuel consumption: A data-warehouse-and-data-mining-based discovery process,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, pp. 2653 –2662, 2015.

[18] J. M. Gavilan and F. V. Morente, “Three similarity measures between one-dimensional data sets,” in *Revista Colombiana de Estadstica*, vol. 37, 2014, pp. 79–94.

[19] O. Gietelink, “Design and validation of advanced driver assistance systems,” PhD thesis, The Netherlands TRAIL Research School, 2007.

[20] S. Giljum, F. Hinterberger, M. Bruckner, E. Burger, J. Frhmann, S. Lutter, E. Pirgmaier, C. Polzin, H. Waxwender, L. Kernegger, and M. Warhurst, “Over-consumption? our use of the worlds natural resources,” Friends of the Earth Europe, Tech. Rep., 2009.

[21] B. Gu and G. Rizzoni, “An adaptive algorithm for hybrid electric vehicle energy management based on driving pattern recognition,” in *ASME International Mechanical Engineering Congress and Exposition*, Chicago, Illinois, USA, 2006.

[22] L. Guzzella and A. Sciarretta, *Vehicle Propulsion Systems*. Springer, 2013.

[23] W. Harrington and V. McConnell, “Motor vehicles and the environment,” Resources for the Future, Tech. Rep., 2003.

[24] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer, 2008.

[25] G. Heppeler, M. Sonntag, and O. Sawodny, “Fuel efficiency analysis for simultaneous optimization of the velocity trajectory and the energy management in

hybrid electric vehicles," in *Preprints of the 19th World Congress The International Federation of Automatic Control*, Cape Town, South Africa, 2014.

[26] D. Herron. (2013). Teslas technology inside the mercedes-benz b-class electric drive, The Long Tail Pipe.

[27] B. Higgs and M. Abbas, "A two step segmentation algorithm for behavioral clustering of naturalistic driving styles," in *Proceedings of the 16th International IEEE Annual Conference on TuA2.4 Intelligent Transportation Systems (ITSC 2013)*, The Hague, The Netherlands, 2013.

[28] ——, "Segmentation and clustering of car-following behavior: Recognition of driving patterns," *IEEE Transactions on Intelligent Transport Systems*, vol. 16, no. 1, 2016.

[29] J. Himberg, K. Korpiaho, H. Mannila, J. Tikanmaki, and H. Toivonen, "Time series segmentation for context recognition in mobile devices," in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, San Jose, CA, USA, 2001.

[30] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classifcation," 2016.

[31] T.-M. Huang, V. Kecman, and I. Kopriva, *Kernel Based Algorithms for Mining Huge Data Sets*. Springer, 2006.

[32] I. Jenhani, N. B. Amor, and Z. Elouedi, "Decision trees as possibilistic classifiers," *International Journal of Approximate Reasoning*, vol. 48, 2007.

[33] S. il Jeon, S. tae Jo, Y. il Park, and J. moo Lee, "Multi-mode driving control of a parallel hybrid electric vehicle using driving pattern recognition," *Journal of Dynamic Systems, Measurement, and Control*, 2002.

[34] J. F. Junior, E. Carvalho, B. V. Ferreira, C. de Souza, Y. Suhara, A. Pentland, and G. Pessin, "Driver behavior profiling: An investigation with different smartphone sensors and machine learning," *PLOS ONE*, 2017.

[35] M. Karmakar and A. K. Nandi, "Driving assistance for energy management in electric vehicle," in *Power Electronics, Drives and Energy Systems (PEDES), 2016 IEEE International Conference on*, Trivandrum, India, 2016.

[36] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "Segmenting time series: A survey and novel approach," in *In an Edited Volume, Data mining in Time Series Databases. Published by World Scientific*, 1993, pp. 1–22.

[37] A. Kumar, “From mass customization to mass personalization: A strategic transformation,” *International Journal of Flexible Manufacturing Systems*, vol. 19, pp. 533–547, 2007.

[38] Y. LeCun, “Learning process in an asymmetric threshold network,” *Disordered systems and biological organization*, 1986.

[39] A. Lew and H. Mauch, *Dynamic Programming A Computational Tool.* Springer, 2007.

[40] A. Lie, C. Tingvall, M. Krafft, and A. Kullgren, “The effectiveness of esp (electronic stability program) in reducing real life accidents,” *Traffic Injury Prevention*, vol. 1, pp. 37–41, 2004.

[41] C.-C. Lin, H. Peng, S. Jeon, and J. M. Lee, “An adaptive longitudinal driving assistance system based on driver characteristics,” in *Proceedings of the 2002 Advanced Vehicle Control Conference*, Hiroshima, Japan, 2002.

[42] N. Lin, C. Zong, M. Tomizuka, P. Song, Z. Zhang, and G. Li, “An overview on study of identification of driver behavior characteristics for automotive control,” *Mathematical Problems in Engineering*, 2014.

[43] X. Lin, D. Grges, and S. Liu, “Eco-driving assistance system for electric vehicles based on speed profile optimization,” in *IEEE Conference on Control Applications (CCA)*, Antibes, France, 2014.

[44] H. T. Luu, L. Nouveliere, and S. Mammar, “Dynamic programming for fuel consumption optimization on light vehicle,” in *6th IFAC Symposium Advances in Automotive Control*, Munich, Germany, 2010.

[45] G. A. M. Meiring and H. C. Myburgh, “A review of intelligent driving style analysis systems and related artificial intelligence algorithms,” *MDPI*, 2015.

[46] J. E. Meseguer, C. T. Calafate, J. C. Cano, and P. Manzoni, “Assessing the impact of driving behavior on instantaneous fuel consumption,” in *IEEE Consumer Communications and Networking Conference (CCNC)*, vol. 12, 2015.

[47] F. Morlock, G. Heppeler, U. Wohlhaupter, and O. Sawodny, “Range extension for electric vehicles by optimal velocity planning considering different driver types,” in *To appear in: 2017 IEEE Conference on Control Technology and Applications*, Kohala Coast, Hawaii, 2017.

[48] Y. G. N. Dembski and A. Soliman, “Analysis and experimental refinement of real-world driving cycles,” *SAE Technical Paper Series*, 2002.

[49] L. Nouveliere, S. Mammar, and H.-T Luu, “Energy saving and safe driving assistance system for light vehicles: Experimentation and analysis,” in *Networking, Sensing and Control (ICNSC), 2012 9th IEEE International Conference on*, IEEE, 2012.

[50] M. Peden, R. Scurfield, D. Sleet, D. Mohan, A. A. Hyder, E. Jarawan, and C. Mathers, “World report on road traffic injury prevention,” World Health Organization, Tech. Rep., 2004.

[51] H. Peng, “Evaluation of driver assistance systemsa human centered approach,” *University of Michigan*,

[52] J. C. Platt, “Sequential minimal optimization: A fast algorithm for training support vector machines,” Microsoft Research, Technical Report MSR-TR-98-14, 1998.

[53] Z. F. Quek and E. Ng, “Driver identification by driving style,” 2013.

[54] E. Rendon-Velez, “Classification and overview of advanced driver assistance systems according to the driving process,” in *ASME 2010 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, ASME, vol. 30, Quebec, Canada, 2010.

[55] M. Richardson, A. Prakash, and E. Brill, “Beyond pagerank: Machine learning for static ranking,” in *Proceedings of the 15th international conference on World Wide Web, WWW*, L. Carr, D. D. Roure, A. Iyengar, C. Goble, and M. Dahlin, Eds., 2006.

[56] F. Rosenblatt, “The perceptron–a perceiving and recognizing automaton,” Cornell Aeronautical Laboratory - Cornell University, Tech. Rep., 1957.

[57] D. E. Rumelhart, G. E. Hinton, and R. J. Willams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, 1986.

[58] B. Shi, L. Xu, H. Jiang, and W. Meng, “Comparing fuel consumption based on normalised driving behaviour: A case study on major cities in china,” *IET Intelligent Transport Systems*, vol. 11, no. 4, pp. 189–195, 2017.

[59] A. Smola and S. Vishwanathan, *Introduction to Machine Learning.* Cambridge University Press, 2008.

[60] V. Vapnik and A. Chervonenkis, *Theorie der Zeidenerkennung.* Akademia-Verlag, 1979.

[61] V. N. Vapnik, *The Nature of Statistical Learning Theory.* Springer, 1999.

[62] V. Vapnik, *Estimation of Dependencies Based on Empirical Data*. Spring, 1982.

[63] J. Wang, L. Zhang, D. Zhang, and K. Li, "An adaptive longitudinal driving assistance system based on driver characteristics," in *IEEE Transactions on Intelligent Transport Systems*, 2013.

[64] W. Wang and J. Xi, "A rapid pattern-recognition method for driving styles using clustering-based support vector machines," in *American Control Conference (ACC)*, 2016.

[65] J. Willmert, *Bellman k-segmentation algorithm*, `http://homepages.spa.umn.edu/~willmert/science/ksegments/`, 2014.

[66] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, and M. Norouzi, *Googles neural machine translation system: Bridging the gap between human and machine translation*, Cornell University Library, 2016.

[67] S. Yadav and S. Shukla, "Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification," in *IEEE 6th International Conference on Advanced Computing*, 2016.

[68] Y. Zhang, W. C. Lin, and Y.-K. S. Chin, "A pattern-recognition approach for driving skill characterization," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 4, pp. 905 –916, 2010.

PART 2:

REALIZING SIMULTANEOUS LANE KEEPING AND ADAPTIVE SPEED REGULATION ON ACCESSIBLE MOBILE ROBOT TESTBEDS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Enforcing multiple, sometimes conflicting control objectives is a challenge present in modern advanced driver assistance systems. Drivers are capable of activating multiple modules simultaneously where safety must be guaranteed at all times. Examples includes adaptive speed regulation, where the vehicle must achieve a desired speed while maintaining a safe distance to any preceding vehicle, and lane keeping, where a vehicle is kept safely within the bounds of a lane.

Provably safe algorithms for both adaptive speed regulation and lane keeping are introduced and used to run experiments on two robotic testbeds. The underlying algorithms are based on control Lyapunov functions for performance, a control barrier functions for safety, and a real-time quadratic program for mediating the conflicting demands between the two. The Robotarium, a robotic testbed that allows students, as well as researchers less experienced with hardware, to experiment with advanced control concepts in a safe and standardized environment, is compared with a more expensive OptiTrack based Khepera robot testbed.

CHAPTER 1
INTRODUCTION

Designing controllers that enforce different and sometimes conflicting objectives is a recurring challenge in many real systems. This is especially crucial for robotic or automotive systems, in which stringent safety-critical specifications must be guaranteed at all times, while also providing the performance expected by a user [12]. Advanced Driver Assistance Systems (ADAS) are a prime example, where passenger and commercial vehicles are outfitted with multiple safety or comfort modules [6]. Lane keeping, for example, controls a vehicle's steering to maintain position in a lane, while adaptive cruise control regulates a vehicle's speed to a driver-set value when there is no preceding vehicle in the lane, and maintains a safe following distance when a leader vehicle is detected [23, 24]. Because ADAS control modules can be activated concurrently in today's vehicles, designing provably correct control software for the simultaneous operation of two or more control modules is crucial and has attracted considerable attention (see [15, 27, 7] and references therein).

Set invariance is a popular method to specify and prove safety properties [4], which are often established through the use of barrier functions (also known as certificates). The barrier function has proved popular because it provides a certificate of set invariance without the difficult task of computing a system's reachable set [21, 20]. Inspired by the automotive safety-control problems, a control barrier function (CBF) is proposed in [2], which extends the normal barrier function condition to only requiring a single sub-level set to be controlled invariant, and extends barrier functions from ODEs to control systems. When CBFs are combined with control Lyapunov functions (CLFs) representing a control objective through a quadratic programming (QP) framework, families of control policies that guarantee safety can be designed. Simply put, the controller mediates the control objectives whenever safety and performance are in conflict.

In this work, we use both the Khepera robot testbed [13] and the Robotarium testbed [19] to explore the real-time hardware implementation of adaptive speed regulation and lane keeping simultaneously using the CBF-CLF-QP approach. Exploring a hardware implementation of CBF-CLF-QPs on two different testbeds allows us to check for potential challenges that arise due to modeling errors, sensor sampling rates, or accuracy limitations of real systems, paving the way for future testing of the algorithms in full-sized vehicles. Furthermore, the implementation serves as an educational example to show how the Robotarium allows students to work with a reasonably sophisticated safety-critical control problem in a (personally) safe and relatively inexpensive setting. For comparison purposes, the Khepera testbed, which uses a costly OptiTrack camera system and Khepera robots, is also used to implement the CBF-CLF-QP algorithms.

1.1 Outline

The work is organized as follows. Chapter 2 presents an introduction to nonlinear control techniques, the analytic model used to represent the Khepera and Robotarium robots, and a brief introduction to the two robotic testbeds used to conduct experiments.

Chapter 3 presents the control methods. In Section 3.1, the control barrier functions used to guarantee safe driving behaviors are discussed. Similarly, Section 3.2 presents the control Lyapunov functions used to achieve control objectives. The two concepts are then formulated as constraints to a quadratic programming problem in Section 3.3.

In Chapter 4, we present the experimental implementation methods for the experiments in Section 4.1 and the experimental results are shown and discussed in Section 4.2.

Finally, a conclusion of the methods and experiments discussed in this work is presented in Chapter 5

CHAPTER 2
BACKGROUND

This chapter presents, first, an introduction to the nonlinear control methods used to develop the controllers discussed in Chapter 3, followed by the unicycle model used to represent the robots in testbed experiments, and, finally, a description of the robotic testbeds themselves.

2.1 Nonlinear Controls

The field of controls involves the mathematical modeling and control of real world systems. In the classical approach, systems are represented using a straightforward, linear model that simplifies the system dynamics and controllers. While this approach is mathematically and computationally appealing, the simplified dynamics are often unable to replicate real world behavior of more complex systems like a robot or an automobile [25]. Nonlinear control methods have, therefore, gained steady attention and popularity. This section introduces the fundamentals of nonlinear control methods used to develop the QP-based controller presented in Chapter 3. Excellent resources for a more in depth explanation of the concepts described here can be found in [11, 25].

A linear system, in its simplest form, is a differential equation:

$$\dot{x} = Ax \tag{2.1}$$

with some initial condition $x_0 = x(t_0)$. In the case that A is a square constant matrix, the solution to the linear system is shown in (2.2).

$$x(t) = e^{A(t-t_0)}x_0 \tag{2.2}$$

The solution to the states, x, as functions of time always have closed forms. This means the system's behavior can always be controlled with the addition of a Bu input term, and stability can be calculated using the values in A; a stable system requires that any bounded input to the system produces a bounded output. However, these linear dynamics cannot be used to describe most real world systems, requiring a more complex model: the nonlinear system.

$$\dot{x} = f(x) + g(x)u \tag{2.3}$$

The system shown in (2.3) is in what is referred to as “affine” form, meaning that the states, x, have a linear relationship to the control input, u. Unlike the linear case, this system has no closed form solution for $x(t)$, because the functions $f(x)$ and $g(x)$ are no longer constant matrices, but any nonlinear (or linear) functions with varying stability properties. So the question becomes, how do we control the behavior and stability of such a vague system? To start, stability is defined to have three classes: normal, asymptotic, and exponential where each case describes a systems behavior when near a stable, or equilibrium, point. Normal stability guarantees that if a system starts close to an equilibrium point, then it will always stay within an arbitrary bound of that point [11]. The other two cases guarantee not only that the system will stay near the equilibrium point, but will converge to it at either an asymptotic or exponential rate [11]. The stability of a system can be checked using a Lyapunov function, which proven by Lyapunov, is a positive definite function, like the energy of a system, whose derivative can be used to determine the stability of a system. For example, the derivative of a Lyapunov function, $V(x)$, is defined as:

$$\dot{V}(x) = \sum_{i=1}^{n} \frac{\partial V}{\partial x_i}\dot{x}_i = \frac{\partial V}{\partial \mathbf{x}} f(x) \tag{2.4}$$

where n is the number of system states and negative semi-definite and definite derivatives imply stability and asymptotic stability respectively. Exponential stability is given by a Lyapunov function bounded by two positive definite functions, whose derivative is bounded by a negative definite function [11].

Having established stability, we introduce feedback linearization: a popular method used for controlling nonlinear systems using a vector of outputs y, functions of the state that should be driven to zero, and the affine nonlinear system. To feedback linearize a single output, single input system, the derivative of the output is taken γ times with respect to x until the control input appears, where γ is known as the relative degree. The original outputs and the all derivatives up to degree $\gamma - 1$, form a new state vector η:

$$\eta(x) = \begin{bmatrix} y(x) \\ L_f y(x) \\ \vdots \\ L_f^{(\gamma-1)} y(x) \end{bmatrix} \tag{2.5}$$

where L is the Lie derivative with respect to the function f and the final derivative of degree γ is $y^{(\gamma)}(x) = L_f^{\gamma} y(x) + L_g L_f^{(\gamma-1)} y(x) u$. The input, $u(x)$, can be solved as a function of the output and its derivatives:

$$u(x) = \frac{1}{L_g L_f^{(\gamma-1)} y(x)} \left(-L_f^{\gamma} y(x) + y(x)^{(\gamma)} \right) \tag{2.6}$$

resulting in a feedback control law as a function of the state, that when plugged back into the system, effectively linearizes the dynamics. Note that the controlled outputs states, $\eta(x)$, in addition to the uncontrolled states, $z(x)$ represent the feedback linearized normal form of the system, as shown in equation (2.7). The total number of controlled states and uncontrolled ones, or zero dynamics, must be equal to the original number of states in the system as explained in [11].

$$\begin{aligned} \dot{\eta} &= f(\eta, z) + g(\eta, z) u, \\ \dot{z} &= q(x, z) \end{aligned} \tag{2.7}$$

where $u \in U$, $\eta \in X$, and $z \in Z$.

This feedback linearized normal form can be taken another step further using control Lyapunov ([22]) and control barrier functions ([5]) together in a quadratic programming based control approach [16, 17]. Assuming that $f(0, z) = 0$, meaning that the system will not leave the set Z for zero-valued output states, a CLF is defined:

Definition 1. *[2] A continuously differentiable function $V : X \to \mathbb{R}$ is an exponentially stabilizing control Lyapunov function if there exist positive constants c_1,c_2,$c_3 > 0$ such that*

$$c_1 ||\eta||^2 \leq V(\eta) \leq c_2 ||\eta||^2 \tag{2.8}$$

$$\inf_{u \in U} \left[L_f V(\eta, z) + L_g V(\eta, z) u \right] \leq -c_3 V(\eta) \tag{2.9}$$

The existence of V results in a family of control values, shown in (2.10), that guarantee stable, exponential convergence of an output to zero as long as the second constraint is satisfied [2] (and references therein).

$$K_{\text{clf}}(\eta, z) = \{u \in U : L_f V(\eta, z) + L_g V(\eta, z)u + c_3 V(\eta) \leq 0\} \tag{2.10}$$

A controller (2.11) than minimizes the control input, while still satisfying the CLF constraint can be derived from this family of feasible control values using a minimization technique.

$$u^* = \text{argmin}\{||u|| : u \in K_{\text{clf}}(\eta, z)\} \tag{2.11}$$

Furthermore, $V(\eta(\mathbf{x}))$ can be calculated using the vector η and the matrix P from solving the algebraic Ricatti equation.

$$V(x) = \eta(x)^T P \eta(x) \tag{2.12}$$

Following from the CLF result, we review some basic results regarding control barrier functions in [28]. Given a continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$, a closed set $\mathcal{C}$ is defined by

$$\mathcal{C} = \{x \in \mathbb{R}^n : h(x) \geq 0\}. \tag{2.13}$$

Assuming that $\mathcal{C}$ is nonempty and has no isolated points, namely, $\text{Int}(\mathcal{C}) \neq \emptyset$ and $\overline{\text{Int}(\mathcal{C})} = \mathcal{C}$.

Consider an affine control system of the form shown in (2.3) with f and g locally Lipschitz continuous, $x \in \mathbb{R}^n$, and $u \in U \subset \mathbb{R}^m$.

Definition 2. *[28] Given a set $\mathcal{C} \subset \mathbb{R}^n$ defined by* (2.13)*, the continuously differentiable function $h : \mathbb{R}^n \to \mathbb{R}$ is called a (zeroing) control barrier function defined on set $\mathcal{D}$ with $\mathcal{C} \subseteq \mathcal{D} \subset \mathbb{R}^n$, if there exists a constant $\gamma > 0$ such that*

$$\sup_{u \in U} [L_f h(x) + L_g h(x)u + \gamma h(x)] \geq 0, \ \forall x \in \mathcal{D}. \tag{2.14}$$

Given a CBF h, for all $x \in \mathcal{D}$, define the set

$$K_{\text{zcbf}}(x) = \{u \in U : L_f h(x) + L_g h(x)u + \gamma h(x) \geq 0\}. \tag{2.15}$$

The following result guarantees the forward invariance of $\mathcal{C}$ when inputs are selected from $K_{\text{zcbf}}(x)$.

Theorem 1. *[28] Let $\mathcal{C} \subset \mathbb{R}^n$ be a set defined by* (2.13) *for a continuously differentiable function h. If h is a CBF on $\mathcal{D}$, then any locally Lipschitz continuous controller*

$u : \mathcal{C} \to U$ satisfying $\forall x \in \mathcal{D}$, $u(x) \in K_{\mathrm{zcbf}}(x)$, will render the set $\mathcal{C}$ forward invariant.

This is an important result because, similarly to the CLFs, a min-norm controller for CBFs can be defined as

$$u^* = \operatorname{argmin}\{||u|| : u \in K_{\mathrm{zcbf}}(x\} \tag{2.16}$$

meaning that when both CLF and CBF constraints are obeyed, a controller can be found that satisfies both control objectives. In the closed form, the controller is found by solving a quadratic programming problem [9]. An example QP with one CLF and one CBF is shown in (**??**).

$$\begin{aligned} \mathbf{u}^*(\mathbf{x}) = & \underset{\mathbf{u}}{\operatorname{argmin}}\, \mathbf{u}^T H \mathbf{u} \\ s.t. \quad & L_f V(\eta, z) + L_g V(\eta, z) u \leq -\alpha V(\eta) + \delta, \\ & L_f h(x) + L_g h(x) u \leq \gamma h(x) \end{aligned} \tag{2.17}$$

where δ are slack variables to soften the CLF constraint, and α and γ are positive constants. The QP-based control input generation method introduced here is used furtherin Chapter 3 to construct a controller capable of achieving the ADAS control objectives of this work.

2.2 Analytical Model

In this section, the robot model used in conjunction with the CBF-CLF-QP control algorithm and the experimental implementations is presented. Both experimental test beds, as explained in Section 2.3, use two-wheeled, differential drive robots, leaving the unicycle robot model as an ideal modeling choice.

2.2.1 The Unicycle Robot Model

The standard unicycle model has three states and is given in (2.18) as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v\cos(\psi) \\ v\sin(\psi) \\ \omega \end{bmatrix}. \tag{2.18}$$

Figure 2.1 shows the coordinates $(x, y), \psi, v, \omega$ representing the 2D position, the orientation, and the longitudinal and angular velocities of the robot, respectively.

When the longitudinal force and the angular torque are taken as inputs to the model, there are two additional states

$$\dot{v} = \frac{u_l}{m} \tag{2.19}$$
$$\dot{\omega} = \frac{u_a}{I_z}, \tag{2.20}$$

where u_l and u_a are the force and torque control inputs, respectively, I_z is the moment of inertia about the z-axis, and m is the mass of the robot. Note that the relative degree, or number of times the states must be differentiated before the input term appears, of the x and y states for u_l and u_a are not equal, which is inconvenient for the input-output feedback linearization explained in Section 3. To overcome this, we choose a point of interest located a distance $a > 0$ forward of the wheel axis, as done in [8], [14] and references therein. This point is shown in Figure 2.1. Noting that the change of coordinates modifies the derivative of the longitudinal velocity term, with the addition of a centripetal acceleration term represented by $a\omega^2$, the final unicycle model used in this work is:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{\psi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos(\psi) - a\omega\sin(\psi) \\ v\sin(\psi) + a\omega\cos(\psi) \\ \frac{u_l}{m} - a\omega^2 \\ \omega \\ \frac{u_a}{I_z} \end{bmatrix}. \tag{2.21}$$

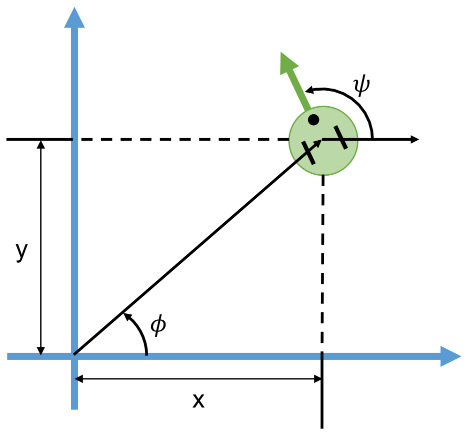
Our state vector is defined moving forward as $\mathbf{x} = [x, y, v, \psi, \omega]^\top$.

The angular position of the robot with respect to the origin is denoted by ϕ (see Figure 2.1) and is useful for the polar coordinate based path tracking algorithm used in the experiments. Clearly, $\phi = \text{atan}(y/x)$ and its time derivative is

$$\dot{\phi} = \frac{\sqrt{(a\omega\cos(\phi - \psi) - v\sin(\phi - \psi))^2}}{\sqrt{x^2 + y^2}}. \tag{2.22}$$

As explained in the experimental implementation section, the robots follow a path defined in polar coordinates:

$$R_{path} = R + b\sin(n\phi). \tag{2.23}$$



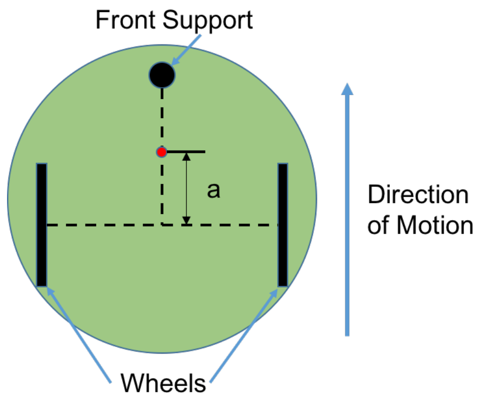


Figure 2.1: (left) States of the unicycle robot. (right) Modified point of interest.

where R is the mean radius of the path, b is the amplitude of the sinusoidal variation of the path, and n is the number of periods in the path.

2.3 Experimental Testbeds

This subsection introduces the two testbeds that are used for the experiments conducted in Chapter 4: the Khepera robots and the Robotarium.

2.3.1 Khepera Robot Testbed

The Khepera robot testbed was provided by the GRITS lab at the Georgia Institute of Technology [13]. A Khepera robot is shown in Figure 2.2.

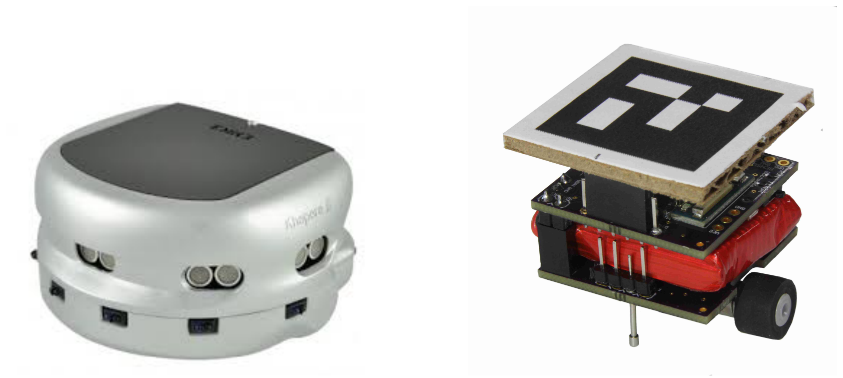


Figure 2.2: (left) Khepera III robot, (right) GRITSBot from the Robotarium.

Sensing. A model-based solution to the speed regulation and lane keeping control problems requires knowledge of each robot's position, orientation, and velocity. In the Khepera robot testbed, the position and orientation data are collected using 10 OptiTrack S250e motion capture cameras.

Actuation. The Khepera III robot uses two DC motors, where each motor actuates a single wheel in the differential drive system. The two motors are powered by a shared 7.4V, 1350mah LiPo battery. The input signal to each motor corresponds to shaft speed and is transmitted to the motor via pulse-width modulation (PWM). For later use, it is important to note that the PWM signal, because it commands motor shaft speed, does not correspond to either the force or torque control input used in the model. The force and torque inputs from the adaptive speed regulation and lane keeping controllers will be integrated through the model to produce equivalent motor speeds, which will then be converted to a PWM-command signal for use in the control loop and the embedded electronics.

Embedded Computing. Each Khepera III robot is equipped with a 600MHz ARM processor and 128Mb RAM, embedded Linux, and a WiFi module for communicating via a wireless router. Control inputs are computed on a centralized computer and sent to the robot via WiFi.

2.3.2 The Robotarium

The Robotarium was conceived because multi-robot testbeds constitute an integral and essential part of the multi-robot research cycle, yet they can be prohibitively expensive, complex, and time-consuming to develop, operate, and maintain. As a swarm-robotic testbed that can be accessed remotely through a web interface (`www.robotarium.org`, the Robotarium gives users the flexibility to test a variety of multi-robot algorithms (see [19],[26]). In particular the Robotarium tackles the challenge of robust, long-term, and safe operation of large groups of robots with minimal operator intervention and maintenance.

The Robotarium utilized in this work contains 20 miniature ground robots, the GRITSBots (see [18]). These inexpensive, differential-drive robots simplify the operation and maintenance of the Robotarium through features such as: (i) automated registration with a server and overhead tracking system, (ii) automatic battery charging, and (iii) wireless (re)programming.

Unlike the Khepera testbed, the Robotarium also offers a MATLAB-based simulator that closely approximates the behavior of the GRITSBots through a parameterized unicycle model and a model of measurement latency. Therefore, controls code developed using the Robotariums simulator can be deployed onto the Robotarium with little to no modifications. This simulator gives users the ability to rapidly iterate through simulation and testing phases, allowing for a straightforward implentation process.

Sensing. Similar to the Khepera testbed, the Robotarium relies on centralized overhead tracking. Instead of an OptiTrack system, however, the Robotarium employs a web camera-based setup that uses a single Microsoft Lifecam HD camera running at an update rate of 30 Hz and a resolution of 1280x720 pixels. ArUco tags (ArUco is an OpenCV-based library for Augmented Reality applications) attached to each GRITSBot allows the system to determine the robot's position and orientation.

Actuation. A GRITSBot is equipped with two miniature stepper motors, each actuating a single wheel. The advantage of stepper motors is that their velocity can be determined without encoders by simply counting the number of steps a motor has moved. The additional complexity of controlling stepper motors is handled via a custom motor board that houses an Atmega168 microcontroller and executes a velocity controller onboard. Each GRITSBot is powered by a single 3.7V, 400 mAh LiPo battery resulting in a runtime of up to 40 minutes on a single charge.

Embedded Computing. A GRITSBot is equipped with an ESP8266, a WiFi-enabled microcontroller equipped with 160 KB of RAM running at 160 MHz. Given these specifications, a GRITSBot is not capable of hosting an operating system, yet it is powerful enough to handle wireless communication, pose estimation, low-level control, as well as high-level behaviors. Similar to the Khepera-based setup, control inputs are computed on a centralized computer and sent to the robot via WiFi.

CHAPTER 3
CONTROL METHODS

The approach to simultaneously achieve adaptive speed regulation and lane keeping is introduced in this chapter, using the fundamentals covered in Section 2.1. By encoding the safety specifications as CBF conditions and the performance objectives as CLF conditions with relaxation parameters, the control policy is generated online by solving a QP that combines CBFs and CLFs as constraints.

3.1 Control Barrier Functions

Following from the definition of CBFs in Section 2.1, we present the applications to adaptive speed regulation and lane keeping.

Adaptive Speed Regulation. Similar to the adaptive cruise control on vehicles, adaptive speed regulation in mobile robots requires the following robot to always maintain a safe time-headway with the lead robot, and achieve a user-defined longitudinal velocity whenever possible.

While achieving the user-set speed is a soft constraint that will be discussed in the next subsection, maintaining a safe time-headway is a hard constraint, which can be expressed as $D \geq \tau v_f$ where D is the distance between the lead and following robots, v_f is the speed of the following robot, and τ is the minimum allowable time headway, in seconds, between the two robots. Therefore, the following CBF is chosen for this speed regulation safety specification:

$$h_{asr} = D - \tau v_f. \tag{3.1}$$

Lane Keeping. The objective of lane keeping is to keep the robots within its lane boundary. Therefore, the lane keeping specification for the robot can be expressed as $|y_{lat}| \leq d_{\max}$ where y_{lat} represents the lateral displacement of the robot w.r.t. the desired path in road fixed coordinates, and $d_{\max}$ is the width of the path. Different CBFs can be used, such as the one introduced in [3]:

$$h_{lk} = d_{max} - sgn(v_{lat})y_{lat} - \frac{1}{2}\frac{v_{lat}^2}{a_{max}}, \tag{3.2}$$

where $sgn(\cdot)$ is the sign function, a_{max} is the maximum allowable lateral acceleration and v_{lat} is the lateral velocity of the robot in road-fixed coordinates, or the following CBF:

$$h_{lk} = 1 - \frac{y_{lat}^2}{d_{max}^2} - \frac{1}{2} v_{lat}^2. \tag{3.3}$$

Both (3.2) and (3.3) ensure that $h_{lk} \geq 0$ implies $|y_{lat}| \leq d_{\max}$.

3.2 Control Lyapunov Functions

While the safety specifications need to be respected at all times, there are three performance objectives that should be achieved whenever possible. First, $v \to v_d$, where v_d is the desired longitudinal velocity of the following robot. Second, $\omega \to 0$, which serves to reduce jittering in the robots angular movement and create a smoother behavior with respect to angular velocity along the course. Finally, $(x, y) \to (R_{path}\cos(\phi), R_{path}\sin(\phi))$ where the right hand side is the tracking point in the desired path. To implement these performance objectives, the following three outputs must be driven to zero:

$$\begin{aligned}
\eta_1 &= v - v_d, \\
\eta_2 &= \omega, \\
\eta_3 &= \begin{bmatrix} x - R_{path}\cos(\phi) \\ y - R_{path}\sin(\phi) \end{bmatrix}.
\end{aligned}$$

It is interesting to point out that driving η_2 and η_3 to zero are contradictory objectives, since η_2 being zero requires the robot to move in a straight line while η_3 being zero requires the robot to track the desired path with a curved trajectory. We show how these conflicting objectives are considered as "soft constraints" and are balanced in a QP framework by some relaxation variables in Subsection 3.3, as well as simulation and experiment results in Section 4.2.

As defined in Chapter 2, for $i = 1, 2, 3$, to achieve exponential convergence of η_i to zero (without regard to other outputs), a special class of control Lyapunov functions $V(x)$ termed *exponentially stabilizing control Lyapunov function (ES-CLF)* [1] are used. For the outputs η_1, η_2, the control Lyapunov functions are taken as

$$V_1(x) = (v - v_d)^2, \tag{3.4}$$

$$V_2(x) = \omega^2. \tag{3.5}$$

For the output η_3, because

$$\dot{\eta}_3 = \begin{bmatrix} \dot{x} - \dot{R}_{path}\cos(\phi) + \dot{\phi}R_{path}\sin(\phi) \\ \dot{y} - \dot{R}_{path}\sin(\phi) - \dot{\phi}R_{path}\cos(\phi) \end{bmatrix},$$

where $\dot{R}_{path} = nb\dot{\phi}\cos(n\phi)$ and $\dot{\phi}$ is given in (2.22), the output η_3 has relative degree 2. Implementing input-output linearization defined in Section 2.1 and using the technique in [1] yields the following CLF:

$$V_3(x) = [\eta_3^\top, \dot{\eta}_3^\top]P[\eta_3^\top, \dot{\eta}_3^\top]^\top,$$

where

$$P = \begin{bmatrix} \sqrt{3} & 0 & 1 & 0 \\ 0 & \sqrt{3} & 0 & 1 \\ 1 & 0 & \sqrt{3} & 0 \\ 0 & 1 & 0 & \sqrt{3} \end{bmatrix}.$$

For each $V_i,\ i = 1, 2, 3$, the set of control inputs that exponentially stabilizes η_i is given as

$$K_i(\mathbf{x}) = \{u | L_f V_i(\mathbf{x}) + L_g V_i(\mathbf{x})u + c_i V_i(\mathbf{x}) \leq 0\} \tag{3.6}$$

where $c_i (i = 1, 2, 3)$ is a positive constant, which is a tunable parameter specifying the convergence rate.

Note that it is impossible to input/output linearize the robot system (2.21) for the output $[\eta_1, \eta_2, \eta_3^\top]^\top$, because there are only two inputs. However, the total length of the output vector is 6, or three times the number of outputs, meaning that the system can be feedback linearized three times simultaneously using the inputs and outputs.

3.3 CBF-CLF-based Quadratic Programs

The CLFs and CBFs developed in the preceding subsections can be unified in a QP to generate a min-norm controller as follows:

$$\begin{aligned} \mathbf{u}^*(\mathbf{x}) = & \underset{\mathbf{u}=[u_l,u_a,\delta_1,\delta_2,\delta_3]^\top}{\operatorname{argmin}} \quad \mathbf{u}^T H \mathbf{u} \\ s.t. \quad & A^i_{\text{clf}}(\mathbf{x})\mathbf{u} \leq b^i_{\text{clf}}(\mathbf{x}),\ i = 1, 2, 3, \\ & A_{\text{asr}}(\mathbf{x})\mathbf{u} \leq b_{\text{asr}}(\mathbf{x}), \\ & A_{\text{lk}}(\mathbf{x})\mathbf{u} \leq b_{\text{lk}}(\mathbf{x}), \end{aligned} \tag{3.7}$$

where

$$\begin{aligned} A^1_{\text{clf}}(\mathbf{x}) &= \left[L_g V_1(\mathbf{x}), -1, 0, 0\right], \\ b^1_{\text{clf}}(\mathbf{x}) &= -L_f V_1(\mathbf{x}) - c_1 V_1(\mathbf{x}), \\ A^2_{\text{clf}}(\mathbf{x}) &= \left[L_g V_2(\mathbf{x}), 0, -1, 0\right], \\ b^2_{\text{clf}}(\mathbf{x}) &= -L_f V_2(\mathbf{x}) - c_2 V_2(\mathbf{x}), \\ A^3_{\text{clf}}(\mathbf{x}) &= \left[L_g V_3(\mathbf{x}), 0, 0, -1\right], \\ b^3_{\text{clf}}(\mathbf{x}) &= -L_f V_3(\mathbf{x}) - c_3 V_3(\mathbf{x}), \\ A_{\text{asr}}(\mathbf{x}) &= \left[L_g h_{asr}(\mathbf{x}), 0, 0, 0\right], \\ b_{\text{asr}}(\mathbf{x}) &= -L_g h_{asr}(\mathbf{x}) - \gamma_1 h_{asr}(\mathbf{x}), \\ A_{\text{lk}}(\mathbf{x}) &= \left[L_g h_{lk}(\mathbf{x}), 0, 0, 0\right], \\ b_{\text{lk}}(\mathbf{x}) &= -L_g h_{lk}(\mathbf{x}) - \gamma_2 h_{lk}(\mathbf{x}), \end{aligned}$$

$H := \operatorname{diag}\{p_1, ..., p_5\} \in \mathbb{R}^{5\times 5}$ are the weight matrix with penalty weight $p_i > 0$, γ_1, γ_2 are given positive constants, and $\delta_i \geq 0 (i = 1, 2, 3)$ are relaxation parameters. These relaxation variables enable us to have controllers with different, potentially conflicting, objectives, whose priority can be changed by tuning p_i, $i = 3, 4, 5$, with larger value implying more priority on that objective.

The optimization problem (3.7) can be solved by QP solvers such as the *quadprog* function in *MATLAB*. The inputs u_l, u_a generated are applied to the robot (2.21), which ensure that it always satisfies the safety specifications and achieves the performance objectives when δ_i are sufficiently small.

CHAPTER 4
EXPERIMENTAL IMPLEMENTATION

The controller developed in Chapter 3 is now used in an experimental application on both the Khepera and Robotarium testbeds. First, the implementation of the controller is discussed followed by the experimental results.

4.1 Controller Implementation

This section explains the implementation of the adaptive speed regulation and lane keeping control algorithms on the Khepera and Robotarium testbeds. For detailed information on both testbeds, please refer to Section 2.3. The implementation differs from that of a standard vehicle because the actuators are not "force-torque-based", but rather, "speed-based". The implementations methods for both the Khepera robots and the Robotarium follow the same general steps shown in Figure 4.1, with the exception of a few noted differences.

To start, pose data on both the Khepera testbed and the Robotarium are acquired through an overhead tracking system and include the 2D position and orientation of each robot. While the Khepera testbed relies on the proprietary OptiTrack motion capture system to provide pose data using reflective infrared markers (at 50 Hz), the Robotarium uses a single web camera and an OpenCV-based tag tracker in conjunction with ArUco tags (at 30 Hz). The Robotarium's tracker uses open-source software packages and is also freely available at `https://github.com/robotarium`.

The acquired 2D position data represent the center position of the robot and these values must be shifted in order to coincide with the modified unicycle model described in Section 2.2. This shift is done according to

$$x_{shift} = x + a\cos(\psi),\ y_{shift} = y + a\sin(\psi).$$

Following the shift of coordinates, the states for each robot are assembled in the order shown in (2.21). The 2D position states, x and y, are taken from the shift calculation in the previous step, while ψ is drawn directly from the data acquisition hardware and ϕ is calculated from the position data using the *atan2()* function in MATLAB. Longitudinal velocity v and angular rate ω are taken from the velocity and angular velocity control inputs sent to the robots in the previous loop. In order

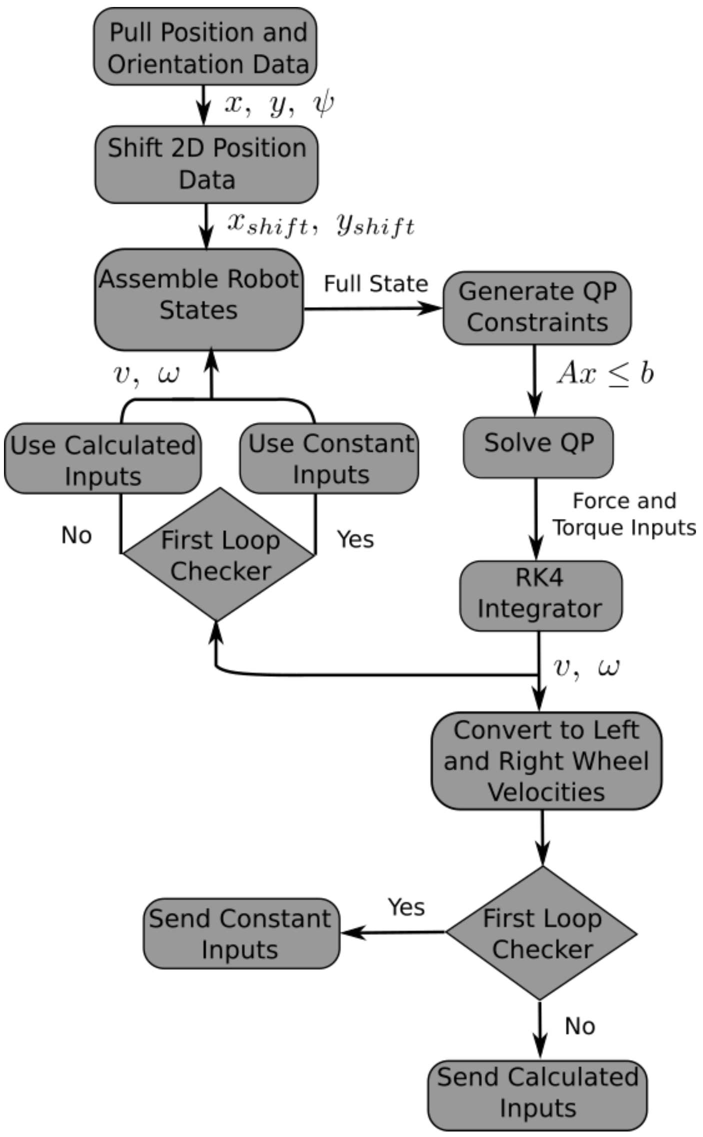


Figure 4.1: Flowchart describing the control generation loop in the experimental implementation.

to avoid a singularity in the CLF based controller on the first loop, the longitudinal velocities of both robots are set to their desired values, v_{d_l} and v_{d_f}, and the angular velocities are set to zero. This initialization causes the robots to have a nonzero positive velocity before the QP-based controller takes full control.

The assembled states $\mathbf{x}$ are used to calculate the matrices $A^i_{\text{clf}}, b^i_{\text{clf}}, A_{\text{asr}}, b_{\text{asr}}, A_{\text{lk}}, b_{\text{lk}}$ for the QP (3.7) in Section 3. MATLAB's *quadprog* function is used to solve the QP (3.7) for the force and torque inputs u_l, u_a in real time. Because both the Khepera and Robotarium robots receive longitudinal and angular velocities as inputs, the force and torque control inputs are integrated using the model's kinematics and a fourth-order Runge-Kutta integration method.

Linear and angular velocity inputs computed by solving the CBF-CLF-QP (3.7) are converted to wheel velocities and sent to the robots via WiFi. As noted, in the initial loop, constant velocity commands are sent to avoid a controller singularity. Both the Khepera-based testbed and the Robotarium rely on Matlab-based APIs to send wheel velocity commands via UDP sockets to the robots. While velocity updates are sent to the Khepera robots at 50 Hz, the Robotarium's robots receive updates at 30 Hz , which is the update rate limitation imposed by the web camera.

4.2 Experimental Results

In this section, we demonstrate the effectiveness of the CBF-CLF-QP controller through experiments on Khepera robots and the Robotarium.

In the experiments, two different scenarios are created to demonstrate that the QP framework is capable of handling different control objectives while always ensuring safety specifications. In the first, the path tracking controller is turned off for a period of time during the experiments. Specifically, the constraint with V_3 is removed from the QP (3.7) when the "off" mode is conducted, and added to the QP (3.7) again when the "on" mode is conducted, with all the other constraints kept the same. By doing this, a scenario where the robot attempts to leave the lane is simulated.

The second type are referred to as "Decaying Path Tracker" experiments. Specifically, the path tracking controller decays when the "decaying mode" is conducted, by changing the variable c_3 in the CLF constraint (3.6) and its corresponding weight entry p_3 in the matrix H in (3.7) as follows:

$$c_3(t) = c_3^{des} + (c_3(0) - c_3^{des})e^{-(t-t_{decay})}, \tag{4.1}$$

$$p_3(t) = p_3^{des} + (p_3(0) - p_3^{des})e^{-(t-t_{decay})}. \tag{4.2}$$

As c_3, p_3 decreases, the QP prioritizes the path tracking objective less; when $p_3 \approx 0$, the path tracking constraint can be considered to be removed from the QP (i.e., the "off" mode), in which case the resulting controller has no intention to track the desired path. This creates another scenario where the follower robot attempts to leave the safety of the lane on multiple occasions.

The parameters for all experiments are shown in Table 4.1, where R, b, n are the parameters of the desired path defined by (2.23), d_{max} is the width of the lane in (3.2), τ is the time-headway in (3.1) and v_{d_f} and v_{d_l} are the desired velocities for the following and lead robots, respectively. For each experiment, the initial conditions are the same: the following robot starts at the position $(x, y) = (R, 0)$, with the lead robot positioned ahead by 25% to 50% of a path revolution, and the robots are oriented tangent to the path at their starting positions with a small longitudinal and zero angular velocity.

Table 4.1: Experiment Parameters

Para.	R	b	n	$d_{\max}$	τ	v_{d_f}	v_{d_l}
Fig.4.5 and Fig.4.8	0.9	0.23	3	0.15	1.8	0.2	0.1
Fig.4.15	0.25	0.1	2	0.15	1.8	0.2	0.1
Fig.4.11	0.25	0.06	3	0.04	3	0.075	0.05
Unit	m	m	-	m	s	m/s	m/s

4.2.1 Experiments on Khepera Robots

This subsection summarizes the execution of both the on/off and decaying path tracking experiments on the Khepera robot testbed, where CBFs h_{asr} in (3.1) and h_{lk} in (3.3) are used.

Figure 4.2 shows the values of the following robot CBFs h_{asr} and h_{lk} during the on/off experiment, with the simulation results, run under the same conditions, displayed as well. Here, the path tracking controller turns off at $t = 20s$ and resumes at $t = 45s$. As can be seen from Figures 4.2, both CBFs are positive for all time, which means that the safety specifications are always satisfied.

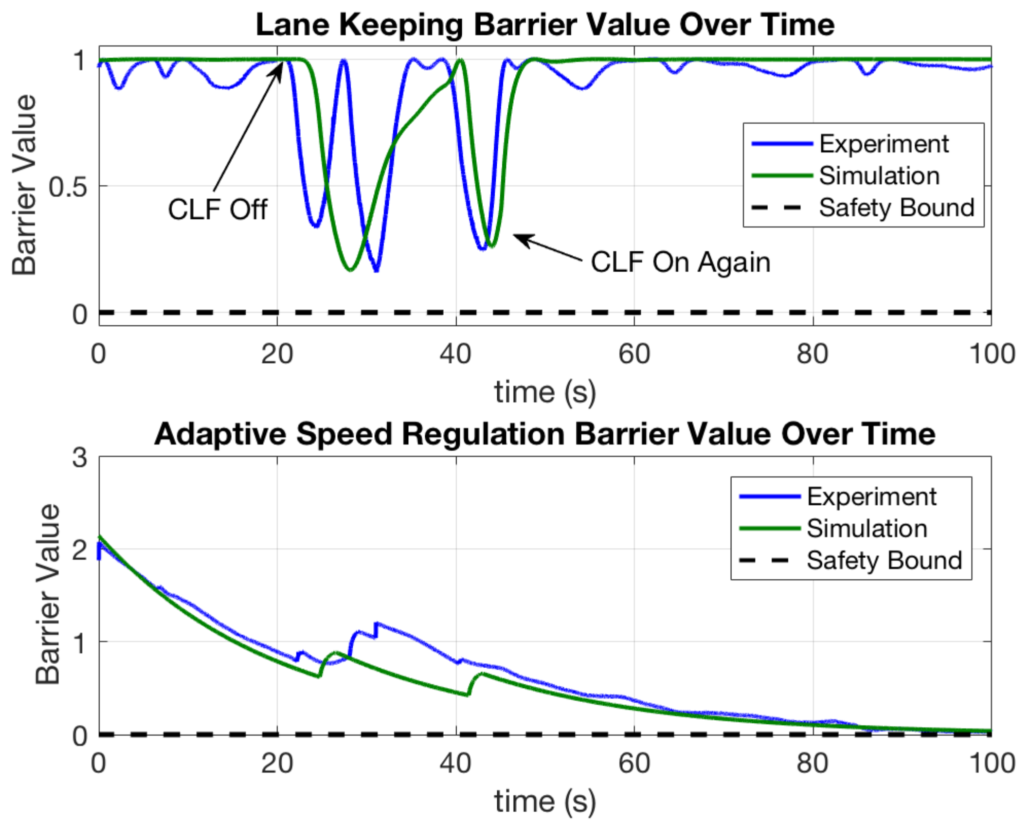


Figure 4.2: The value of CBFs in the on/off experiment and simulation on Khepera robots. (top) Value of the CBF h_{lk} where positiveness implies that the robot is within the boundary. (bottom) Value of the CBF h_{asr} where non-negativeness implies the specification $D \geq \tau v_f$ is satisfied.

Figure 4.3 shows the value of CLFs V_1, V_2, V_3 for the same on/off experiment and simulation, where penalty weights $p_3 = 10^5$, $p_4 = 1$, and $p_5 = 10^3$ are used such that the controller put more emphasis on V_1 (achieving the desired speed) and V_3 (tracking the path) while less on V_2 (reducing the angular velocity). As can be seen from Figure 4.3, before 20 seconds, the values of V_1, V_2, V_3 are quite smooth; when the tracking controller turns off at $t = 20$, the value of V_2, V_3 fluctuates quite a bit since the penalty weight on V_2 is small and removing the constraint of V_3 from the QP poses no restriction on V_3 during this period; when the tracking controller turns on again, the value of V_1, V_2, V_3 become smooth again. The mismatch between the experimental and simulation data in Figure 4.2 and Figure 4.3 can be attributed to calibration and modeling errors.

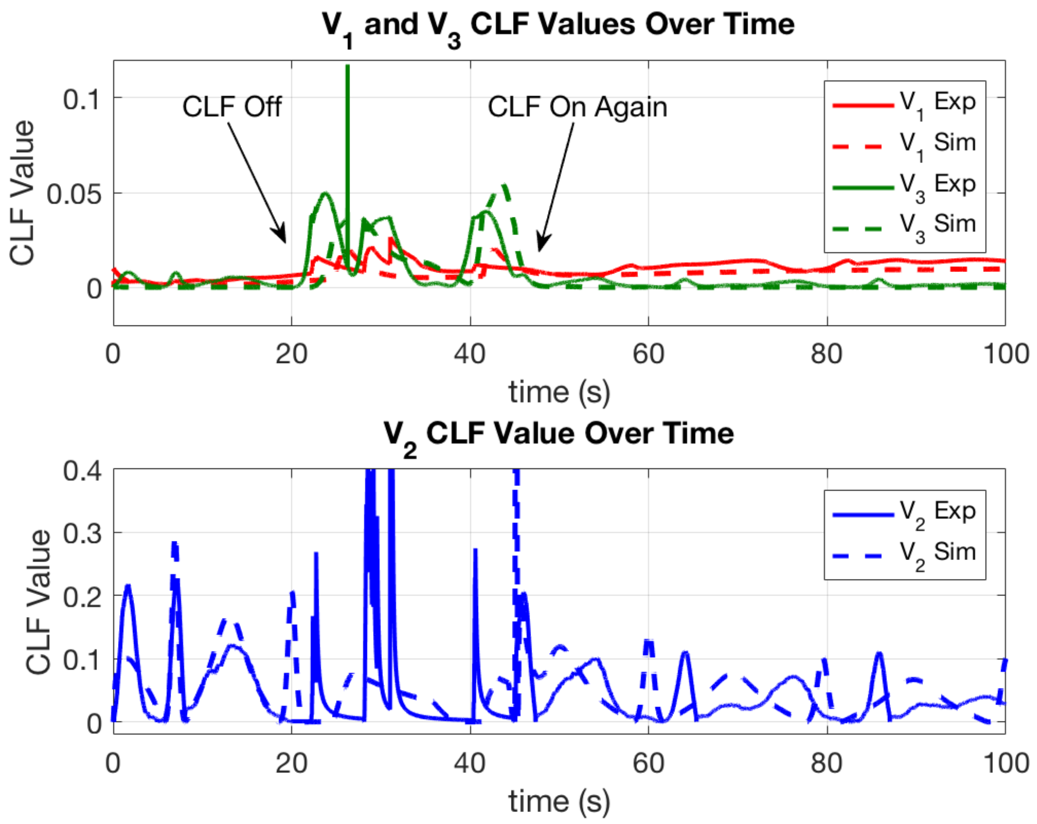


Figure 4.3: The value of CLFs in the experiment and simulation on Khepera robots. (top) Value of V_1 and V_3. (bottom) Value of V_2.

Figure 4.4 shows the following Khepera robot's trajectories based on the on/off experimental data. The time-lapse images in Figure 4.5 show a point during the period when the tracking controller turns off. It can be seen that even when the tracking objective is removed at that point, the robot is repelled back to the lane when it attempts to leave due to the constraint of the lane keeping CBF.

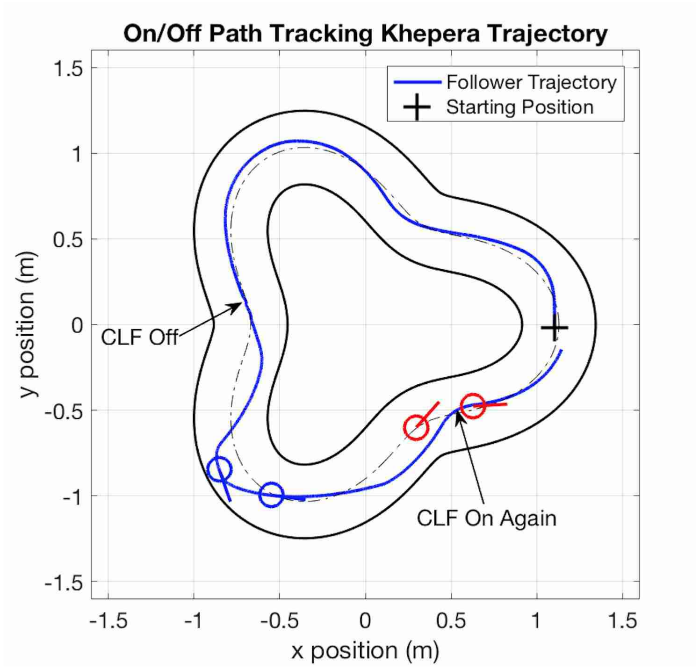


Figure 4.4: Trajectories of the Khepera robots during the on/off path tracking experiment. During the "off mode" between 20 seconds and 45 seconds, the following robot remains within the lane boundary because of the lane keeping CBF.

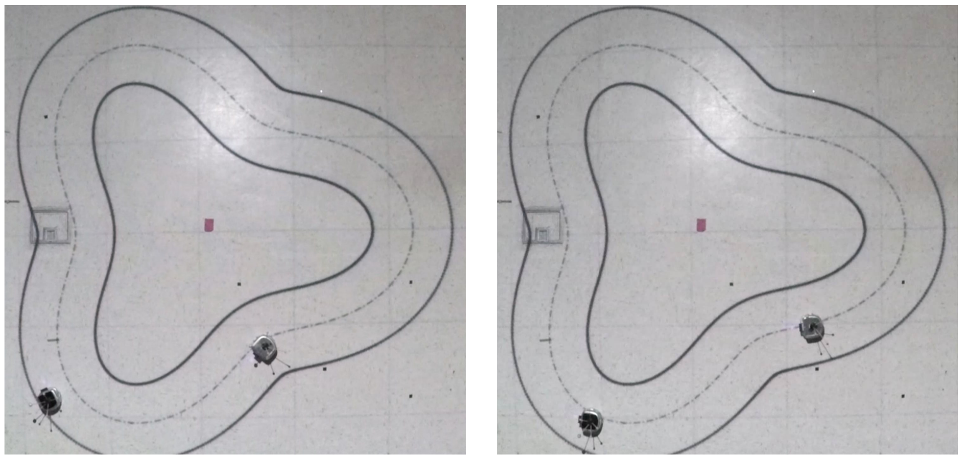


Figure 4.5: Time-lapse images of the Khepera robot during the "off" mode. The Khepera robot can be kept inside the lane due to the lane keeping CBF. (left) The following robot approaches the lane boundary. (right) The following robot is repelled from the lane boundary.

The values of the lane keeping and adaptive speed regulation barrier functions in the decaying path tracker experiment and simulation with the Khepera robots is shown in Figure 4.6. Here, the path tracking CLF begins to decay just after the 10s mark, leaving the robot free to attempt to leave the lane as shown by the fluctuating barrier function values. Despite the absence of the path tracker, the both barriers in both the experiment and simulation cases remains positive, meaning that the safety constraints are always obeyed. The discrepancy between the simulation and experimental values can be attributed to measurement inaccuracies and modeling errors.

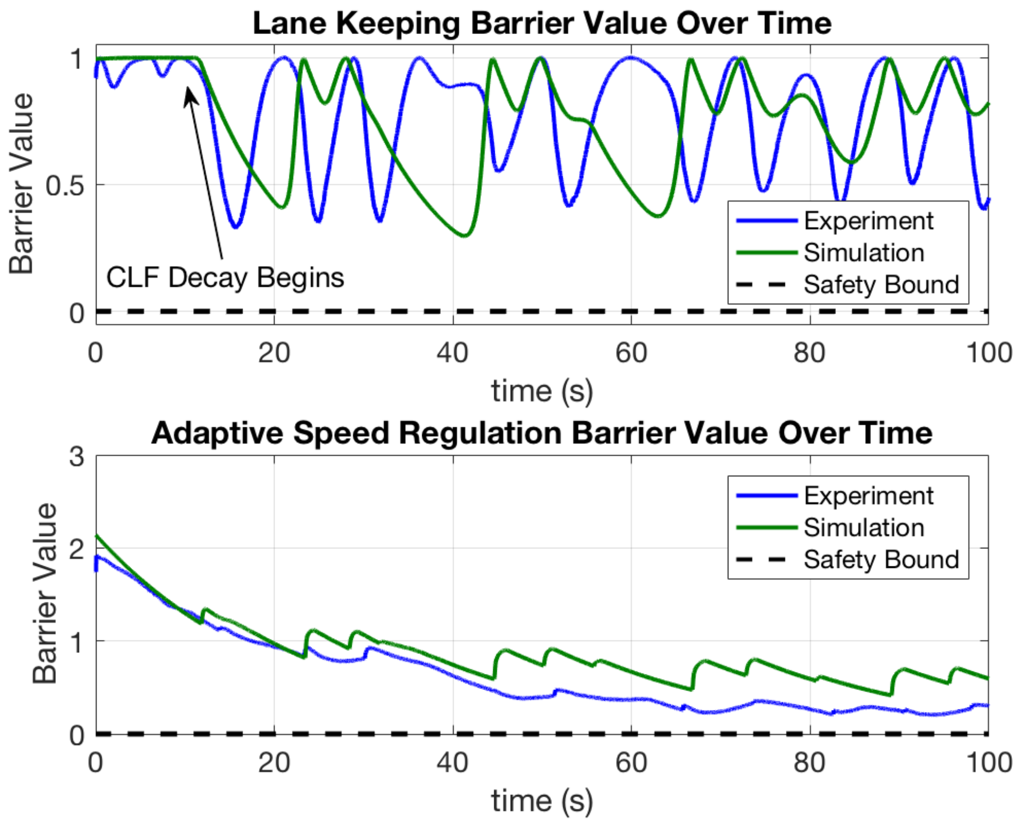


Figure 4.6: Barrier function values during the decaying path tracking experiment with Khepera robots.

Figures 4.7 shows the plotted trajectory of the follower robot , accompanied by time lapse images in 4.8, during the decaying path tracker experiment. Note how the robot is successfully repelled away from the edge of the lane in the bottom right corner of the images.

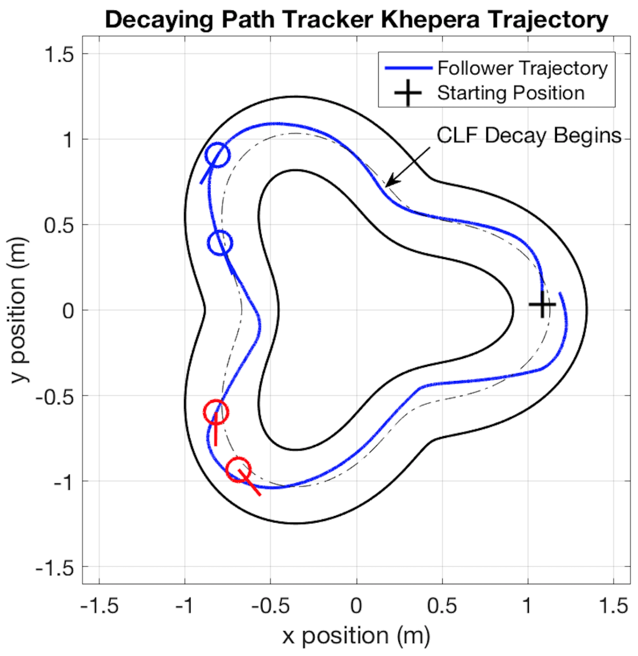


Figure 4.7: Khepera robots' trajectories during the decaying path tracking experiment.

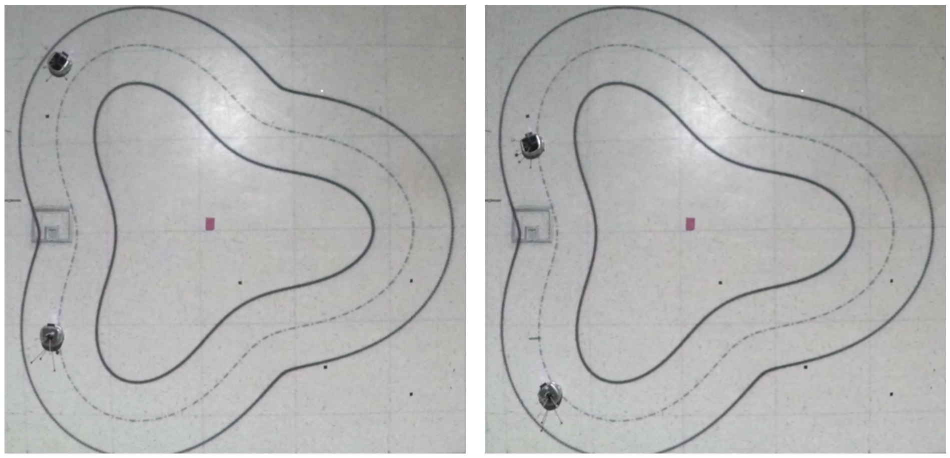


Figure 4.8: Time-lapse images during Decay Khepera Experiment. (left) The robot approaches edge of the lane. (right) The robot is turned away from lane edge by CBF.

4.2.2 Experiments on the Robotarium

This subsection summarizes the execution of the on/off and decaying path tracking experiments on the Robotarium testbed, where CBFs h_{asr} in (3.1) and h_{lk} in (3.2) are used. Another experiment, where is path tracking controller is on for the entire duration, is also presented.

Figure 4.9 shows the value of CBFs h_{asr} and h_{lk} of the following robots during the on/off experiment on the Robotarium, with the corresponding simulation results depicted as well. The path tracking controller turns off at $t = 10s$ and resumes at $t = 42s$. As shown in Figure 4.9, both CBFs are positive for all time, which means that the lane keeping and adaptive speed regulation specifications are always satisfied. Compared with the results on the Khepera robots in Figure 4.2, the value of h_{asr} and h_{lk} here are both noisier. This difference is likely due to the size differences between the two testbeds and the fact that the Robotarium runs at a lower update rate (30Hz) than the Khepera testbed (50Hz).

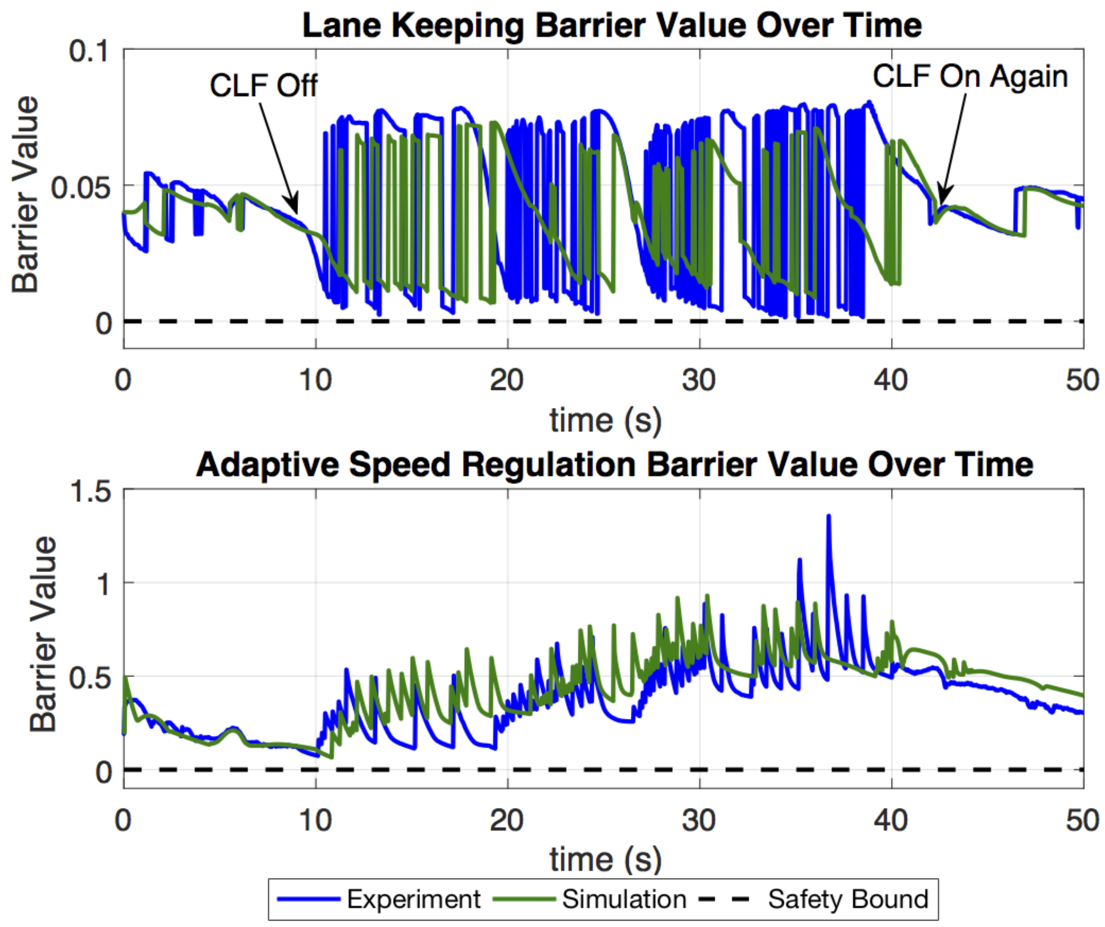


Figure 4.9: The value of CBFs in the experiment and simulation on Robotarium with on/off path tracking CLF. (top) Value of the CBF h_{lk} where positiveness implies satisfaction. (bottom) Value of the CBF h_{asr} where non-negativeness implies satisfaction.

Figure 4.10 shows the following robots' trajectories based on the experimental data on Robotarium. Figure 4.11 shows two time-lapse images during the "off" mode of the experiment. It can be seen that the following robot approaches the lane boundary and is repelled from the boundary because of the lane keeping CBF.

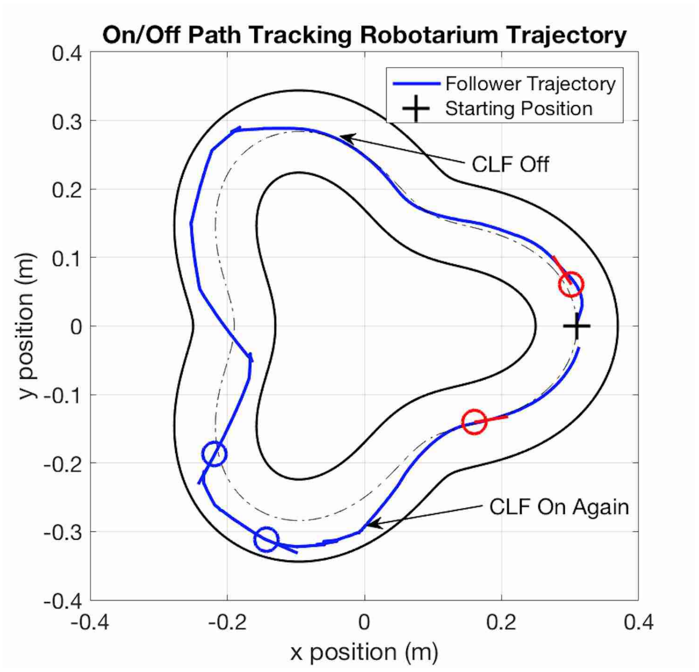


Figure 4.10: Trajectories of the following robots on Robotarium during the on/off path tracking experiment.

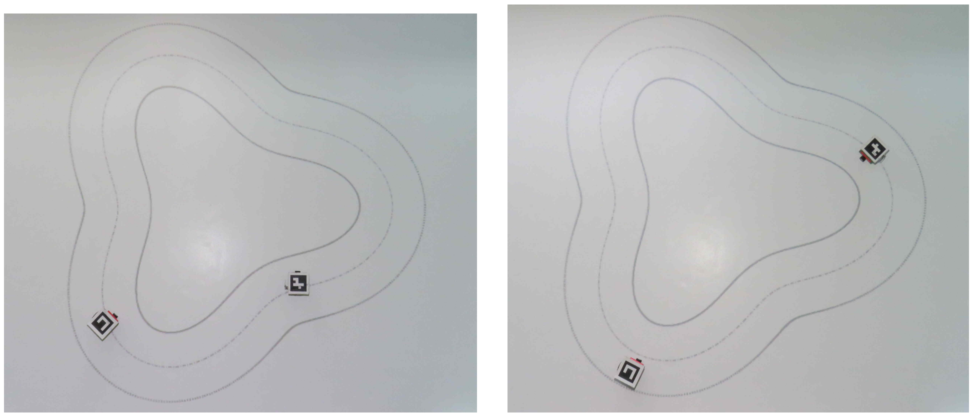


Figure 4.11: Time-lapse images of the Robotarium during the off mode. (left) The following robot approaches the lane boundary. (right) The following robot is repelled from the lane boundary.

Figure 4.12 shows the trajectories of the following robot during the decaying path tracker experiment. It can be seen that the robots are kept within the lane boundary, despite the decaying tracking CLF. Figure 4.13 shows the value of CBFs h_{asr} and h_{lk} during the experiment along with the corresponding simulation results, which indicate that the safety specifications are satisfied for all time.

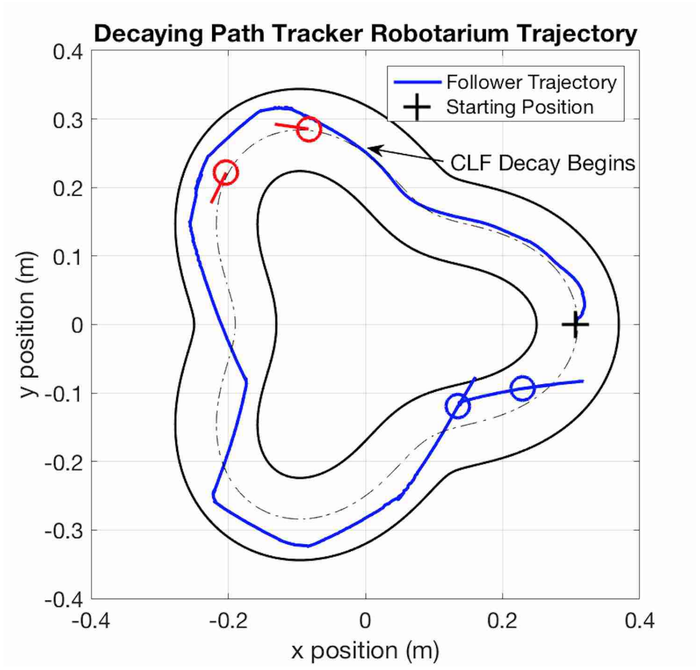


Figure 4.12: Robots' trajectories in Robotarium experiment with the decaying path tracking CLF.

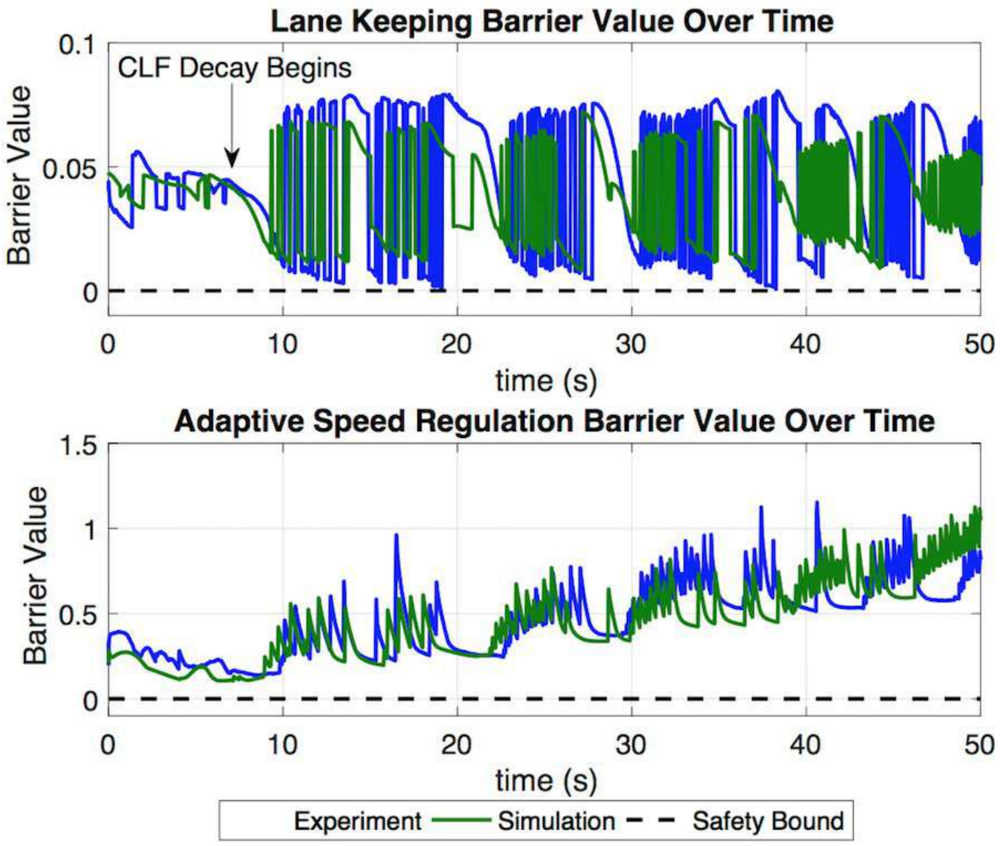


Figure 4.13: The value of CBFs h_{asr} and h_{lk} in the experiment and simulation on Robotarium with the decaying path tracking CLF.

As a comparison, Figure 4.14 shows the value of CBF h_{asr} when the path tracking controller is turned on for the entire Robotarium experiment, with the corresponding simulation results depicted as well. Taking given model and calibration errors into account, h_{asr} remains predominantly positive for all time, meaning that the adaptive speed regulation specification is always satisfied. Particularly, when h_{asr} is close to 0, the minimum allowable time headway τ is achieved. Figure 4.15 shows two time lapse images of the experiment, where the following robot approaches and maintains a safe headway to the leader.

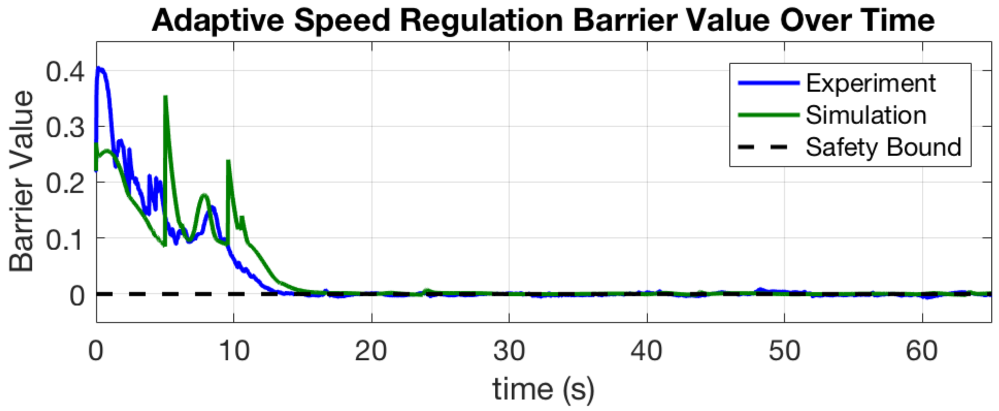


Figure 4.14: The value of CBF h_{asr} in the Robotarium experiment and simulation when the path tracking controller is turned on for all time. Non-negativeness of h_{asr} means satisfaction of the adaptive speed regulation specification.

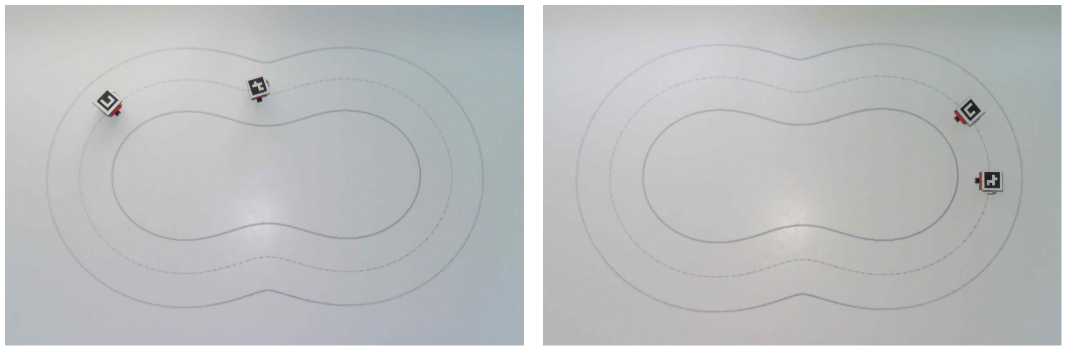


Figure 4.15: Time lapse of the adaptive speed regulation experiment in Robotarium. (left) Minimum headway is not reached after a quarter revolution around the path. (right) Minimum headway maintained eventually.

Video results for the on/off and decaying experiments on the Khepera robots and Robotarium, in addition to the pure path tracking Robotarium experiment, can be found at [10].

CHAPTER 5
CONCLUSION

In this work, the real-time implementation of lane keeping and adaptive speed regulation was experimentally evaluated on two robot testbeds, based on a CBF-CLF-QP approach. Our results showed the effectiveness of the CBF-CLF-QP framework for multi-objective controller design with safety constraints, and its potential for implementation on ADAS control software. These results were achieved on accessible mobile testbeds—a key advantage of this approach is that it provides students hands-on experience with rather sophisticated control software where safety, in the sense of formal methods, is a primary factor. Additional advantages include the low cost of the experiments, and in the case of the Robotarium, the fact that multiple groups of faculty and students can compare results on a common platform. The hope is that this will allow for the rapid prototyping and deployment of safety-critical controllers among a wide audience of researchers.

With regard to the potential for future work, the control algorithms developed here, were done so with an automotive application in mind. Therefore, the next logical step would be use both the lane keeping and adaptive cruise control CLFs and CBFs on a test vehicle. This advancement would require modeling the complex dynamics of an automobile as well as a test bed capable of measuring and applying the required inputs and outputs.

REFERENCES

[1] A. D. Ames, K. Galloway, K. Sreenath, and J. W. Grizzle, “Rapidly exponentially stabilizing control lyapunov functions and hybrid zero dynamics,” *Automatic Control, IEEE Transactions on*, vol. 59, no. 4, pp. 876–891, 2014.

[2] A. D. Ames, J. W. Grizzle, and P. Tabuada, “Control barrier function based quadratic programs with application to adaptive cruise control,” in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, 2014, pp. 6271–6278.

[3] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, “Control barrier function based quadratic programs for safety critical systems,” *Automatic Control, IEEE Transactions on*, 2017 (to appear).

[4] F. Blanchini, “Survey paper: Set invariance in control,” *Automatica*, vol. 35, no. 11, pp. 1747–1767, 1999.

[5] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, NY, USA: Cambridge University Press, 2004.

[6] M. Campbell, M. Egerstedt, J. P. How, and R. M. Murray, “Autonomous driving in urban environments: Approaches, lessons and challenges,” *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 368, no. 1928, pp. 4649–4672, 2010.

[7] S. Dai and X. Koutsoukos, “Safety analysis of automotive control systems using multi-modal port-hamiltonian systems,” in *19th ACM International Conference on Hybrid Systems: Computation and Control*, 2016.

[8] C. De La Cruz and R. Carelli, “Dynamic modeling and centralized formation control of mobile robots,” in *IECON 2006-32nd Annual Conference on IEEE Industrial Electronics*, IEEE, 2006, pp. 3880–3885.

[9] K. S. Galloway, K. Sreenath, A. D. Ames, and J. W. Grizzle, “Torque saturation in bipedal robotic walking through control lyapunov function based quadratic programs,” *CoRR*, vol. abs/1302.7314, 2013.

[10] *Implementing simultaneous lane keeping and adaptive speed regulation in the robotarium*, `https://youtu.be/h1QyibYE1gI`.

[11] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2002.

[12] J. C. Knight, "Safety critical systems: Challenges and directions," in *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, IEEE, 2002, pp. 547–550.

[13] S. G. Lee and M. Egerstedt, "Controlled coverage using time-varying density functions," *IFAC Proceedings Volumes*, vol. 46, no. 27, pp. 220–226, 2013.

[14] F. N. Martins, M. Sarcinelli-Filho, T. F. Bastos, and R. Carelli, "Dynamic modeling and adaptive dynamic compensation for unicycle-like mobile robots," in *Advanced Robotics, 2009. ICAR 2009. International Conference on*, IEEE, 2009, pp. 1–6.

[15] P. Nilsson, O. Hussien, A. Balkan, Y. Chen, A. Ames, J. Grizzle, N. Ozay, H. Peng, and P. Tabuada, "Correct-by-construction adaptive cruise control: Two approaches," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 4, pp. 1294–1307, 2016.

[16] K. PengTee, S. S. Ge, and E. H. Tay, "Barrier lyapunov functions for the control of output-constrained nonlinear systems," *Automatica*, vol. 45, no. 4, pp. 918–927, 2009.

[17] PeterWieland and F. Allgoewer, "Constructive safety using control barrier functions," in *IFAC Proceedings Volumes*, 2007, pp. 462–467.

[18] D. Pickem, M. Lee, and M. Egerstedt, "The GRITSBot in its natural habitat - a multi-robot testbed," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015, pp. 4062–4067.

[19] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The robotarium: A remotely accessible swarm robotics research testbed," *ArXiv:1604.00640*, 2016.

[20] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *Hybrid Systems: Computation and Control*, 2004, pp. 477–492.

[21] S. Prajna, A. Jadbabaie, and G. J. Pappas, "A framework for worst-case and stochastic safety verification using barrier certificates," *IEEE Transactions on Automatic Control*, vol. 52, no. 8, pp. 1415–1428, 2007.

[22] E. D. Sontag, "A lyapunov-like characterization of asymptotic controllability," *SIAM Journal on Control and Optimization*, vol. 21, no. 3, pp. 462–471, 1981.

[23] K. L. Talvala, K. Kritayakirana, and J. C. Gerdes, “Pushing the limits: From lanekeeping to autonomous racing,” *Annual Reviews in Control*, vol. 35, no. 1, pp. 137–148, 2011.

[24] A. Vahidi and A. Eskandarian, “Research advances in intelligent collision avoidance and adaptive cruise control,” *Intelligent Transportation Systems, IEEE Transactions on*, vol. 4, no. 3, pp. 143–153, 2003.

[25] M. Vidyasagar, *Nonlinear Systems Analysis*. Englewood Cliffs, New Jersey 07632: Prentice Hall Inc., 1978.

[26] L. Wang, A. D. Ames, and M. Egerstedt, “Multi-objective compositions for collision-free connectivity maintenance in teams of mobile robots,” *ArXiv preprint arXiv:1608.06887*, 2016.

[27] X. Xu, J. W. Grizzle, P. Tabuada, and A. D. Ames, “Correctness guarantees for the composition of lane keeping and adaptive cruise control,” *ArXiv preprint, arXiv:1609.06807*, 2016.

[28] X. Xu, P. Tabuada, A. D. Ames, and J. W. Grizzle, “Robustness of control barrier functions for safety critical control,” in *IFAC Conference on Analysis and Design of Hybrid Systems*, 2015, pp. 54–61.