Trajectory Tracking Control of Unmanned Ground Vehicles using an Intermittent Learning Algorithm

Pavan Kumar Gundu

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

Ryan M. Gerdes, Chair

Kyriakos G. Vamvoudakis

Ryan K. Williams

July 03, 2019

Blacksburg, Virginia

Keywords: Q-Learning, Autonomous Driving, Intermittent Protocols, Platooning, String Stability

Trajectory Tracking Control of Unmanned Ground Vehicles using an Intermittent Learning Algorithm

Pavan Kumar Gundu

(ABSTRACT)

Traffic congestion and safety has become a major issue in the modern world's commute. Congestion has been causing people to travel billions of hours more and to purchase billions of gallons of fuel extra which account to congestion cost of billions of dollars. Autonomous driving vehicles have been one solution to this problem because of their huge impact on efficiency, pollution, and human safety.

Motion control is a key area of autonomous driving research that handles moving parts of vehicles in a deliberate and controlled manner. A widely worked on problem in motion control concerned with time parameterized reference tracking is trajectory tracking. Having an efficient and effective tracking algorithm embedded in the autonomous driving system is the key for better performance in terms of resources consumed and tracking error. Many tracking control algorithms in literature rely on an accurate model of the vehicle and often, it can be an intimidating task to come up with an accurate model taking into consideration various conditions like friction, heat effects, ageing processes etc. And typically, control algorithms rely on periodic execution of the tasks that update the control actions, but such updates might not be required, which result in unnecessary actions that waste resources.

The main focus of this work is to design an intermittent model-free optimal control algorithm in order to enable autonomous vehicles to track trajectories at high-speeds. To obtain a solution which is model-free, a Q-learning setup with an actor-network to approximate the optimal intermittent controller and a critic network to approximate the optimal cost, resulting in the appropriate tuning laws is considered.

Trajectory Tracking Control of Unmanned Ground Vehicles using an Intermittent Learning Algorithm

Pavan Kumar Gundu

(GENERAL AUDIENCE ABSTRACT)

A risen research effort in the area of autonomous vehicles has been witnessed in the past few decades because these systems improve safety, comfort, transport time and energy consumption which are some of the main issues humans are facing in the modern world's highway systems. Systems like emergency braking, automatic parking, blind angle vehicle detection are creating a safer driving environment in populated areas. Advanced driver assistance systems (ADAS) are what such kind of systems are known as. An extension of these partially automated ADAS are vehicles with fully automated driving abilities, which are able to drive by themselves without any human involvement.

An extensively proposed approach for making traffic throughput more efficient on existing highways is to assemble autonomous vehicles into platoons. Small intervehicle spacing and many vehicles constituting each platoon formation improve the traffic throughput significantly. Lately, the advancements in computational capabilities, in terms of both algorithms and hardware, communications, and navigation and sensing devices contributed a lot to the development of autonomous systems (both single and multiagent) that operate with high reliability in uncertain/dynamic operating conditions and environments.

Motion control is an important area in the autonomous vehicles research. Trajectory-tracking is a widely studied motion control scenario which is about designing control laws that force a system to follow some time-dependent reference path and it is important to have an effective and efficient trajectory-tracking control law in an autonomous vehicle to reduce the resources consumed and tracking error.

The goal of this work is to design an intermittent model-free trajectory tracking control

algorithm where there is no need of any mathematical model of the vehicle system being controlled and which can reduce the controller updates by allowing the system to evolve in an open loop fashion and close the loop only when an user defined triggering condition is satisfied. The approach is energy efficient in that the control updates are limited to instances when they are needed rather than unnecessary periodic updates. Q-learning which is a model-free reinforcement learning technique is used in the trajectory tracking motion control algorithm to make the vehicles track their respective reference trajectories without any requirement of their motion model, the knowledge of which is generally needed when dealing with a motion control problem.

The testing of the designed algorithm in simulations and experiments is presented in this work. The study and development of a vehicle platform in order to perform the experiments is also discussed. Different motion control and sensing techniques are presented and used. The vehicle platform is shown to track a reference trajectory autonomously without any human intervention, both in simulations and experiments, proving the effectiveness of the proposed algorithm.

Acknowledgments

I would like to thank and acknowledge all the wonderful individuals who have guided and supported me during my graduate studies at Virginia Polytechnic Institute and State University.

First of all, I would like to thank my advisor Dr. Ryan Gerdes for his support and guidance throughout the course of my research. In helping me clearly define the problem he was very instrumental and also in keeping me concentrated when I would go off track. I also extend my gratitude to him for being my main source of financial support during the course of my graduate studies and research at Virginia Tech.

I would also like to thank my co-advisor Dr. Kyriakos Vamvoudakis who has a great command and experience in the area of modern control, for guiding me when I got stuck or overwhelmed with a problem.

Also, I would like to thank my committee member Dr. Ryan Williams for taking the time to read my thesis and providing me with valuable inputs to help improve my work.

I would like to extend special thanks to my family and friends for their continuous support and encouragement.

Finally, I thank all the undergraduate students that I have worked with, for their continual help and support.

This work was supported in part, by NATO under grant No. SPS G5176, by an NSF CAREER under grant No. CPS-1851588, and by NSF under grant No. SATC-1801611.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Traffic congestion has caused urban Americans to travel an extra 4.8 billion hours and to purchase an additional fuel of 1.9 billion gallons, which accounts to a congestion cost of $101 billion, in 2010 [33]. In 2014, a wastage of 7 billion hours of time and 3 billion gallons of fuel was witnessed in America due to congested traffic [34]. This noticeable increase in wastage of resources over the last few decades and the huge impact on pollution, human welfare of present highway systems is what has been fueling the autonomous driving vehicles research which may decrease traffic jams and better driver's safety and comfort.

The recent breakthroughs in computational abilities, in terms of both hardware and algorithms, communications, and navigation and sensing devices have made the development of autonomous single and multi-agent systems that operate with great reliability, in the presence of uncertain and dynamic operating conditions, environments, and goals feasible [17]. These systems try to make an accurate characterization of their surroundings and self-state using the knowledge base and the available data from sensors which in turn is used to take optimal decisions focusing at interacting with the surroundings (other vehicles, pedestrians, buildings) in a timely manner.

1.2 Intermittent Model-free Trajectory Tracking

One key area of autonomous driving research is motion control in which trajectory tracking is a widely worked on problem concerned with the development of control algorithms that can make a vehicle track a time parameterized reference autonomously [9]. Better performance in terms of resources used and tracking error can be achieved by using an efficient and effective trajectory tracking technique in the autonomous vehicle. The recently focused approaches for trajectory tracking like Model predictive control (MPC) [25], [30], [28] are computationally exhaustive which might make them less appealing to automotive manufacturers as they might need extra computational resources like graphical processing units (GPUs) to compute control inputs at high rates, which could increase the vehicle costs noticeably. Many other trajectory tracking control algorithms in the literature rely on the knowledge of an accurate system model of the vehicle/robot [25], [29], [18]. Often, it is a difficult task to come up with simple and accurate equations representing a system. It can be a daunting task to account for additional conditions like heat effects, friction, characteristics dispersion due to mass production, aging processes, hysteresis effects in the system equations. Most of the control strategies being developed from modern control theory are not being utilized in the industrial world because of their dependency often on an accurate mathematical model of the system under consideration. But, model-free control techniques [15], [16], which require no mathematical model of the system being controlled, have been used to a large extent in a variety of practical situations within a few years [5], [6], [19], [40], [41].

To perform well, control techniques may not require periodic execution of tasks that update control actions, resulting in avoiding unnecessary executions that waste resources. Autonomous vehicles are one such system where wastage of resources can be reduced by avoiding extraneous control updates which could use energy reserves unnecessarily. Even

though, desired closed-loop performance might not need the periodic execution of control tasks, controllers are still usually implemented in a time-triggered fashion despite the fact that communication resources which have to be shared between the control and other tasks can be scarce along with the computational resources.

Two recently focused control techniques with an aperiodic and reduced number of controller updates are self-triggered control and event-triggered control [23]. The scheduling context of different strategies used in studying the control problems under computational and communication constraints is from where these techniques are developed [8], [13], [11]. These techniques allow the system being controlled to update with open-loop control and when a triggering condition is satisfied they close the loop. Lower computation and sparse communication can decrease the wastage of system resources while guaranteeing performance and stability. A mix of ideas from adaptive control [24] and optimal control [26] theories, along with reinforcement learning [27], [31] can be used to take care of the performance and stability issues. Actor/critic networks, a widely used reinforcement learning technique [35], that employs an actor to estimate the optimal control policy and a critic to evaluate the actor's selected control policies are considered here.

1.2.1 Related Work

Authors of [38] proposed a model-free optimal control technique in it to regulate the states of continuous time linear system to the origin. But in many cases, the states tracking a reference trajectory with optimal performance is necessary rather than just regulating them to an origin, like in the case of an autonomous vehicle tracking a reference trajectory. A Hamiltonians based approach was used in [39] to derive an event-triggered optimal tracking algorithm for known non-linear and linear systems by employing an augmented system

technique. An approach to achieve tracking without the knowledge of system model using Q-learning is developed in this work, as in [38], to derive a similar algorithm. Q-learning, a technique in reinforcement learning, which is model independent was developed mainly for discrete-time systems, where based on previous state and action observations optimal actions are chosen [7], [43]. It helps in determining the optimal control policy easily by learning an action-based value function that can estimate the cost incurred by performing a certain action while in a certain state. Systems with uncertainties in the motion model can be controlled using Q-learning because of its model-free nature.

1.3 Experimental Platform

An existing scaled vehicle that behaves similarly to the full-scale everyday cars was studied, and few hardware and software developments were made on this vehicle as part of this work in order to provide an experimental platform to test the simulation results on. The vehicle has its own sensors and actuators as part of the sensing and control systems.

The vehicle studied is capable of traveling at speeds of up to 10m/s and is collision tolerant. Using this kind of scaled vehicle rather than a full-scale one would be cost-effective [14], [12], [21] keeping in mind the collisions that might happen while testing. Also, the high-speed nature of the vehicle would be in good comparison to the nature of full-scale vehicles, not having that capability might underestimate the speeds at which full-scale vehicles move. The experimental platform is discussed in more detail in some of the later sections.

1.4 Platooning and String Stability

Since the research on autonomous driving technology has begun, researchers have been trying to improve passenger safety and comfort while also tackling the problem of traffic congestion to improve the throughput of current highways with minimal infrastructure changes. However, the stop-and-go nature of traffic noticeably affects the vehicles' fuel economy [33]. Therefore, for increasing the throughput, it is not sufficient to just improve the individual vehicles' performance. An extensively proposed approach to get an even efficient traffic throughput on existing highways is to develop vehicle platoon systems which can have innovative capabilities [22]. When applied to passenger vehicles, their main goals are to increase the vehicle density on highways, avoiding traffic jams and improve fuel efficiency [36]. In addition, vehicle platoons can better the safety of each vehicle constituting them in comparison to independent vehicles [10].

When dealing with platoons, the control design has to be such that, not only the performance requirements of individual vehicles are met but also those of the whole system together are taken care of. One such kind of requirement is string stability. A vehicle platoon is string stable if the distance and speed errors attenuate as they propagate upstream the platoon [46]. In mathematical terms, if the respective transfer functions from the distance error and speed error of a vehicle to that of its following vehicle have a magnitude less than or equal to 1, the platoon is said to satisfy string stability [36].

The inter-vehicle spacings in platoons can be variable or constant. While in constant spacing policy, the desired inter-vehicle spacing is constant and doesn't depend on the velocity of the vehicles, in variable spacing policy, the desired inter-vehicle spacing depends on the velocity and is variable. The constant spacing policy is the one widely employed because it can achieve very high traffic capacity when compared to the variable spacing policy. But

it has been shown that with constant inter-vehicle spacing in the platoon, it is impossible to guarantee string stability when each vehicle is receiving relative distance and velocity information only w.r.t. the vehicle in front of it [46]. A comparison between performances of various constant inter-vehicle spacing policies can be found here [37] and it shows that constant spacing policy can be made string stable if the lead vehicle transmits its velocity or velocity and acceleration information to all the other vehicles in the platoon.

An effort to implement the proposed trajectory tracking algorithm on a group of vehicles is also made in this work to achieve platooning with constant inter-vehicle spacing while satisfying the performance requirements.

1.5 Thesis Outline

Development of the event-triggered model-free optimal trajectory tracking controller is discussed in Chapter 2. The application of this controller in simulations on the experimental vehicle considering its motion model and dynamics is also presented in Chapter 2. More details about the experimental ground vehicle platform design and development (hardware and software) are provided in Chapter 3. Experimental validation of the control algorithm on the vehicle is discussed in Chapter 4. Chapter 5 talks about the application of the trajectory tracking algorithm on a group of vehicles to achieve constant inter-vehicle spacing platooning and how the reference trajectories for each vehicle are generated for the platooning purpose. Finally Chapter 6 concludes and talks about future work.

Chapter 2

Model-free Event-triggered Optimal Tracking Control

2.1 Overview

A model-free event-triggered optimal controller to make an autonomous vehicle track a reference trajectory at high-speed using Q-learning is developed in this chapter. The event-triggered nature of the controller lets it limit the control updates to instances when an event is triggered rather than unnecessary periodic updates. The algorithm is shown to be applicable for any generic linear time-invariant continuous time system and then its application to the experimental vehicle is discussed in the form of simulations.

The problem is formulated by augmenting the system and reference data and then coming up with a triggering condition to tell the controller when to update the control using a sampled version of the augmented states. Then a relation between the infinite (continuous state sampling) and limited (event-triggered sampling) bandwidth costs is derived. Further, the algorithm is made model-free by using Q-learning, and relevant tuning laws that tune online and predict the optimal controller and the proposed Q-function are derived using the actor/critic approximation technique of reinforcement learning to perform trajectory tracking autonomously. Finally, the equilibrium point of the closed-loop system is shown to be globally asymptotically stable by employing an impulsive systems model.

This chapter is structured as follows. The problem is formulated, and the system and path dynamics are augmented in Section 2.2 which would form a base for the online optimization problem to be solved. Section 2.3 derives the event-triggering condition for the controller such that optimality and stability of the equilibrium point are not compromised. A Q-learning based model-free formulation to make the algorithm independent of motion model and an actor/critic structure is provided in Section 2.4. Application of the proposed algorithm in simulations on the experimental vehicle considering its dynamics is presented in Section 2.5.

Table 2.1 below shows the notations used in this chapter hereafter.

Table 2.1: Notations used through Section 2.4.

$\mathbb{R}^+$	Set of positive real numbers
$\overline{\lambda}(B)$	Maximum eigenvalue of a matrix B
$\underline{\lambda}(B)$	Minimum eigenvalue of a matrix B
$\|.\|$	Frobenius norm and Euclidean norm for a matrix and vector respectively
$\otimes$	Kronecker product
vech(B)	Half-vectorization of a symmetric $n \times n$ matrix B is a column vector of length $n(n+1)/2$ obtained by vectorization of only the upper (or lower) triangular part of B
mat$(\cdot)$	Inverse vech$(\cdot)$ operation called as matricization

2.2 Problem Formulation

Consider the following linear time-invariant continuous time system,

$$\dot{x}(t) = Ax(t) + Bu(t),\ x(0) = x_0,\ t \geq 0$$

here $x(t) \in \mathbb{R}^n$ is the state vector, $u(t) \in \mathbb{R}^m$ is the system's control input, while $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times m}$ are the system (or state) and input matrices. The A and B matrices will be considered unknown.

The reference path is modeled using the following equations $\dot{z}(t) = A_d z(t),\ z(0) = z_0,\ t \geq 0$ where $z(t) \in \mathbb{R}^n$ denotes the bounded reference trajectory and $A_d \in \mathbb{R}^{n\times n}$ the unknown path matrix, even though this approach will not require them.

For the system to track its reference, the following error is defined, $e_{\text{track}}(t) := x(t) - z(t),\ t \geq 0$, with dynamics, $\dot{e}_{\text{track}}(t) = Ax(t) + Bu(t) - A_d z(t),\ t \geq 0$.

The augmented state $x_{\text{aug}} := [e_{\text{track}}^{\text{T}} \quad z^{\text{T}}]^{\text{T}} \in \mathbb{R}^{2n}$ is considered and its dynamics are written as

$$\dot{x}_{\text{aug}}(t) = A_{\text{aug}} x_{\text{aug}}(t) + B_{\text{aug}} u(t),\ t \geq 0. \tag{2.1}$$

A sampled version of the augmented state is considered as follows in-order to make the system save resources

$$\hat{x}_{\text{aug}}(t) := \begin{cases} x_{\text{aug}}(r_j), & \forall t \in (r_j, r_{j+1}] \\ x_{\text{aug}}(t), & t = r_j. \end{cases}$$

The error between the sampled and continuous version of the augmented state is defined as

$$e(t) := \hat{x}_{\text{aug}}(t) - x_{\text{aug}}(t),\ t \geq 0$$

The goal here is to find an optimal controller u_d, called the event-triggered (intermittent) controller which takes the form $u_d := k(\hat{x}_{\text{aug}}(t))$ and optimizes a cost function identical to the one with continuously updating control without any need of the plant (system) model nor of the reference.

$$J(x_{\text{aug}}(0); u_d) = \int_0^\infty e^{-\gamma\tau} \big(x_{\text{aug}}^{\text{T}} Q_{\text{aug}} x_{\text{aug}} + u_d^{\text{T}} R u_d\big) \mathrm{d}\tau, \tag{2.2}$$

with user-defined matrices $Q_{\text{aug}} := \begin{pmatrix} Q & 0_{n\times n} \\ 0_{n\times n} & 0_{n\times n} \end{pmatrix}$ and $R \succ 0$ which incorporate penalties associated with deviations from operating points and taking actions, respectively, where $0_{n\times n}$ is a zero matrix of size $n \times n$ and $Q \succeq 0$, and with $\gamma \in \mathbb{R}^+$ a discount factor.

Equivalently, the aim is to obtain optimal controller $u_d^\star$ such that

$$J(x_{\text{aug}}(0); u_d^\star) \leq J(x_{\text{aug}}(0); u_d),\ \forall u_d$$

which also can be expressed as the following optimization problem,

$$J(x_{\text{aug}}(0); u_d^\star) = \min_{u_d} J(x_{\text{aug}}(0); u_d)$$

given the dynamics in (2.1) as conditions.

The ultimate goal is to find a value function that is approximately equal to the value function $V^\star$ for (2.1) with input as $u_c := k(x_{\text{aug}}(t))$ (the continuous sampled controller), defined as,

$$V^\star(x_{\text{aug}}(t)) := \min_{u_c} \int_t^\infty \frac{1}{2} e^{-\gamma(\tau - t)} \big(x_{\text{aug}}^{\text{T}} Q_{\text{aug}} x_{\text{aug}} + u_c^{\text{T}} R u_c\big) \mathrm{d}\tau,\ \forall t, \tag{2.3}$$

without any information of the motion and reference models, and given an event-triggered controller which is not periodic. The next section will talk about the optimal event-triggered control framework and derivation of the event-triggering condition for the controller.

2.3 Event-triggering Condition

For the continuous-time sampled controller u_c, the continuous-time sampled Hamiltonian corresponding to the system (2.1) and value function (2.3) is defined as

$$\mathcal{H}(x_{\text{aug}}, u_c, \frac{\partial V^\star}{\partial x_{\text{aug}}}) = \frac{\partial V^\star}{\partial x_{\text{aug}}}^{\text{T}} (A_{\text{aug}} x_{\text{aug}} + B_{\text{aug}} u_c) + \\ + \frac{1}{2} x_{\text{aug}}^{\text{T}} Q_{\text{aug}} x_{\text{aug}} + \frac{1}{2} u_c^{\text{T}} R u_c - \gamma V^\star, \ \forall x_{\text{aug}}, u_c. \tag{2.4}$$

The continuous sampled optimal control associated with (2.4) is

$$u_c^\star := u_c^\star(x_{\text{aug}}) = -R^{-1} B_{\text{aug}}^{\text{T}} \frac{\partial V^\star}{\partial x_{\text{aug}}}, \ \forall x_{\text{aug}}. \tag{2.5}$$

The time-triggered value function can be expressed as the following function of augmented state because the system (2.1) is linear.

$$V^\star(x_{\text{aug}}) = \frac{1}{2} x_{\text{aug}}^{\text{T}} P x_{\text{aug}}, \ \forall x_{\text{aug}}, \tag{2.6}$$

where $P \in \mathbb{R}^{2n \times 2n}$ is the unique solution to below Algebraic Riccati Equation (ARE)

$$A_{\text{aug}}^{\text{T}} P + P A_{\text{aug}} - \gamma P + P B_{\text{aug}} R^{-1} B_{\text{aug}}^{\text{T}} P + Q_{\text{aug}} = 0. \tag{2.7}$$

Substituting (2.6) into (2.5), it follows that

$$u_c^\star := u_c^\star(x_{\text{aug}}) = -R^{-1} B_{\text{aug}}^{\text{T}} P x_{\text{aug}}, \ \forall x_{\text{aug}}, \tag{2.8}$$

It is clear that complete information about the dynamics of the augmented system, i.e., the matrices A_{aug} and B_{aug} are necessary to calculate the value of P from (2.7).

The sampled version of (2.8) to avoid the continuous control updates is defined as follows

$$u_d^\star := u_d^\star(\hat{x}_{\text{aug}}) = -R^{-1}B_{\text{aug}}^{\text{T}}P\hat{x}_{\text{aug}},\ \forall \hat{x}_{\text{aug}}. \tag{2.9}$$

Because controllers $u_c^\star$, $u_d^\star$ are linear functions of x_{aug}, $\hat{x}_{\text{aug}}$ respectively and system (2.1) is linear, a Lipschitz constant $L \in \mathbb{R}^+$ exists satisfying, $\|u_c^\star(x_{\text{aug}}) - u_d^\star(\hat{x}_{\text{aug}})\| = \|R^{-1}B_{\text{aug}}^{\text{T}}P(x_{\text{aug}} - \hat{x}_{\text{aug}})\| \leq \|R^{-1}B_{\text{aug}}^{\text{T}}P\|\|(x_{\text{aug}} - \hat{x}_{\text{aug}})\| \leq L\|e\|,\ \forall x_{\text{aug}}, \hat{x}_{\text{aug}}$.

Now the following lemma is ready to be stated.

Lemma 2.1. *Given the aperiodically updated controller as in (2.9), the Hamiltonian*

$$\begin{aligned}\mathcal{H}(x_{aug}, u_d^\star, \frac{\partial V^\star(x_{aug})}{\partial x_{aug}}) \equiv \frac{\partial V^\star(x_{aug})}{\partial x_{aug}}^T \big(&A_{aug}x_{aug} \\ &- B_{aug}R^{-1}B_{aug}^T P\hat{x}_{aug}\big) + \frac{1}{2}\hat{x}_{aug}^T P B_{aug}R^{-1}B_{aug}^T P\hat{x}_{aug} \\ &+ \frac{1}{2}x_{aug}^T Q_{aug}x_{aug} - \gamma V^\star(x_{aug}),\ \forall x_{aug}, \hat{x}_{aug}.\end{aligned}$$

meets the below inequality,

$$\left\|\mathcal{H}(x_{aug}, u_d^\star, \frac{\partial V^\star(x_{aug})}{\partial x_{aug}})\right\| \leq \frac{\overline{\lambda}(R)}{2}L^2\left\|e\right\|^2.$$

Proof. Check Lemma 1 of [39]. □

Theorem 2.2. *Assume a positive definite and radially unbounded function $V = \frac{1}{2}x_{aug}^T P x_{aug}$ exists, where P is the solution to (2.7). Then, system (2.1) $\forall t \in (r_j, r_{j+1}]$, $j \in \mathbb{N}$ closed looped with event-triggered controller (2.9), with cost-functional (2.2) and triggering condition*

$$\|e\|^2 \leq \frac{(1-\beta^2)\underline{\lambda}(Q)}{L^2\overline{\lambda}(R)}\|e_{track}\|^2 + \frac{\underline{\lambda}(R)}{L^2\overline{\lambda}(R)}\|u_d^\star\|^2$$

for $\gamma = 0$ and user-specified constant $\beta \in (0,1)$ has an equilibrium point which is asymptot-

ically stable, and for $\gamma \neq 0$ *the error* e_{track} *is bounded uniformly and ultimately. Also, the intermittent control policy* (2.9) *is optimal while the optimal value is*

$$J^\star(x_{aug}(0); u_c^\star, u_d^\star) = \frac{1}{2}\int_0^\infty e^{-\gamma\tau}\big(u_c^\star - u_d^\star\big)^T R\big(u_c^\star - u_d^\star\big)\, d\tau + J^\star(x_{aug}(0); u_c{}^\star),$$

where $J^\star(x_{aug}(0); u_c^\star)$ *is the optimal cost in the case of the time-triggered controller* (2.8).

Proof. See Theorem 1 of [39]. □

2.4 Model-free Online Tuning

The value function (2.6) should be expressed in the form of a function of x_{aug} and u_d for representing the Q-function.

The Q-function $\mathcal{Q}(x_{\text{aug}}, u_d) : \mathbb{R}^{2n+m} \to \mathbb{R}$ can be written as follows

$$\begin{aligned}
&\mathcal{Q}(x_{\text{aug}}, u_d) := V^\star(x_{\text{aug}}) + \mathcal{H}(x_{\text{aug}}, u_d, d, \frac{\partial V^\star}{\partial x_{\text{aug}}}) - \mathcal{H}(x_{\text{aug}}, u_c^\star, \frac{\partial V^\star}{\partial x_{\text{aug}}}) \\
&= V^\star(x_{\text{aug}}) + \frac{1}{2}x_{\text{aug}}^{\text{T}} P(A_{\text{aug}}x_{\text{aug}} + B_{\text{aug}}u_d) \\
&+ \frac{1}{2}(A_{\text{aug}}x_{\text{aug}} + B_{\text{aug}}u_d)^{\text{T}} P x_{\text{aug}} + \frac{1}{2}u_d^{\text{T}} R u_d + \frac{1}{2}x_{\text{aug}}^{\text{T}} Q_{\text{aug}} x_{\text{aug}} \\
&- \gamma V^\star(x_{\text{aug}}),\ \forall x_{\text{aug}}, u_d,
\end{aligned} \tag{2.10}$$

where $\mathcal{H}(x_{\text{aug}}, u_c^\star, \frac{\partial V^\star}{\partial x_{\text{aug}}}) = 0$ and optimal cost with time-triggering controller updates is $V^\star(x_{\text{aug}}) = \frac{1}{2}x_{\text{aug}}^{\text{T}} P x_{\text{aug}}$.

The Q-function (2.10) can be written as a function of x_{aug} and u_d in the following way

$$\mathcal{Q}(x_{\text{aug}}, u_d) := \frac{1}{2}U^{\text{T}}\bar{Q}U = \frac{1}{2}\text{vech}\big(\bar{Q}\big)^{\text{T}}\big(U \otimes U\big),\ \forall x_{\text{aug}}, u_d, \tag{2.11}$$

where $\bar{Q} = \begin{pmatrix} Q_{x_{\text{aug}}x_{\text{aug}}} & Q_{x_{\text{aug}}u_d} \\ Q_{u_d x_{\text{aug}}} & Q_{u_d u_d} \end{pmatrix} \in \mathbb{R}^{(2n+m)\times(2n+m)}$ is positive definite with $Q_{x_{\text{aug}}x_{\text{aug}}} = (1-\gamma)P + Q_{\text{aug}} + PA_{\text{aug}} + A_{\text{aug}}^{\text{T}}P$, $Q_{x_{\text{aug}}u_d} = PB_{\text{aug}}$, $Q_{u_d x_{\text{aug}}} = Q_{x_{\text{aug}}u_d}^{\text{T}} = B_{\text{aug}}^{\text{T}}P$, $Q_{u_d u_d} = R$ and $U := \begin{pmatrix} x_{\text{aug}}^{\text{T}} & u_d^{\text{T}} \end{pmatrix}^{\text{T}}$.

It is shown in the following lemma that (2.11) and (2.3) are equal when $u_d = u_d^\star$.

Lemma 2.3. *With the Q-function and optimal time-triggered controller* (2.11) *and* (2.3) *respectively,*

$$\begin{aligned} \mathcal{Q}^\star(x_{aug}, u_d^\star) &:= \min_{u_d} \mathcal{Q}(x_{aug}, u_d) \\ &= V^\star(x_{aug}) + \frac{1}{2}(u_c^\star - u_d^\star)^T R(u_c^\star - u_d^\star), \ \forall x_{aug}. \end{aligned}$$

Proof. Check Lemma 2 of [38]. □

The optimal control policy is obtained as follows by solving $\frac{\partial \mathcal{Q}(x_{\text{aug}}, u_d)}{\partial u_d} = 0$,

$$u_d^\star = \arg\min_{u_d} \mathcal{Q}(x_{\text{aug}}, u_d) = -Q_{u_d u_d}^{-1} Q_{u_d x_{\text{aug}}} \hat{x}_{\text{aug}} \tag{2.12}$$

A model-independent online learning of action-based value function's parameters is developed in the following subsection by using a reinforcement learning approach.

2.4.1 Actor/Critic Approximators

Here, the critic and actor approximators estimate the parameters of Q-function (2.11) and optimal controller (2.12) respectively.

By assuming as $W_c := \frac{1}{2}\text{vech}\big(\bar{Q}\big)$, the Q-function (2.11) can be written as $\mathcal{Q}^\star(x_{\text{aug}}, u_d^\star) = W_c^{\text{T}}\big(U \otimes U\big)$, where $W_c \in \mathbb{R}^{\frac{1}{2}(2n+m)(2n+m+1)}$ would be the actual parameters.

Because the ideal parameters for calculating $u_d^\star$ and $\mathcal{Q}^\star$ are not known, *weight estimates* for the approximators are considered as follows, for the critic approximator $\hat{W}_c := \frac{1}{2}\text{vech}\big(\hat{\bar{Q}}\big)$. Then the estimated Q-function would be

$$\hat{\mathcal{Q}}(x_{\text{aug}}, u_d) := \hat{W}_c^{\text{T}}\big(U \otimes U\big),\ \forall x_{\text{aug}}, u_d \tag{2.13}$$

Similarly, the actor approximator with estimated actor weights can be written as

$$\hat{u}_d = \hat{W}_a^{\text{T}}\hat{x}_{\text{aug}}, \tag{2.14}$$

where $\hat{W}_a \in \mathbb{R}^{2n\times m}$.

Taking advantage of integral reinforcement learning technique from [42] the value function (2.3) is written as the following equation

$$\begin{aligned} e^{-\gamma T}V^\star(x_{\text{aug}}(t)) = V^\star(x_{\text{aug}}(t-T)) - \frac{1}{2}\int_{t-T}^{t} e^{-\gamma(\tau-t+T)} \\ \big(x_{\text{aug}}^{\text{T}}Q_{\text{aug}}x_{\text{aug}} + u_d^{*T}Ru_d^\star\big)\text{d}\tau \end{aligned} \tag{2.15}$$

where $T \in \mathbb{R}^+$ is a fixed time interval and small.

By employing the result from Lemma 2.3, (2.15) is written as

$$\begin{aligned} e^{-\gamma T}\mathcal{Q}^\star(x_{\text{aug}}(t), u_d^\star(t)) = \mathcal{Q}^\star(x_{\text{aug}}(t-T), u_d^\star(t-T)) \\ -\frac{1}{2}\int_{t-T}^{t} e^{-\gamma(\tau-t+T)}\big(x_{\text{aug}}^{\text{T}}Q_{\text{aug}}x_{\text{aug}} + u_d^{*T}Ru_d^\star\big)\text{d}\tau. \end{aligned} \tag{2.16}$$

To find the tuning law for the estimated critic weights, an error $e_c \in \mathbb{R}$ that is eventually driven to zero is defined. Considering equation (2.16) and the estimated Q-function (2.13),

(2.16) is re-written as

$$\begin{aligned} e_c := e^{-\gamma T}\hat{\mathcal{Q}}(x_{\text{aug}}(t),\hat{u_d}(t)) - \hat{\mathcal{Q}}(x_{\text{aug}}(t-T),\hat{u_d}(t-T)) + \\ \frac{1}{2}\int_{t-T}^{t} e^{-\gamma(\tau-t+T)}\big(x_{\text{aug}}^{\mathrm{T}}Q_{\text{aug}}x_{\text{aug}} + \hat{u_d}^{\mathrm{T}}R\hat{u_d}\big)\mathrm{d}\tau \\ = e^{-\gamma T}\hat{W}_c^{\mathrm{T}}\big(U(t)\otimes U(t)\big) + \frac{1}{2}\int_{t-T}^{t} e^{-\gamma(\tau-t+T)}\big(x_{\text{aug}}^{\mathrm{T}}Q_{\text{aug}}x_{\text{aug}} \\ + u_d^{\mathrm{T}}Ru_d\big)\mathrm{d}\tau - \hat{W}_c^{\mathrm{T}}\big(U(t-T)\otimes U(t-T)\big). \end{aligned}$$

Now, for finding the tuning law for the estimated actor weights, error $e_a \in \mathbb{R}^m$ is defined as $e_a := \hat{W}_a^{\mathrm{T}}x_{\text{aug}}(r_j) + \hat{Q}_{u_du_d}^{-1}\hat{Q}_{u_dx_{\text{aug}}}x_{\text{aug}}(r_j),\ t = r_j$ where $\hat{Q}_{u_du_d}^{-1}$ and $\hat{Q}_{u_dx_{\text{aug}}}$ values are extracted from $\hat{W}_c$. Tuning laws for $\hat{W}_c$ and $\hat{W}_a$ are found in the next subsection by driving the errors e_c and e_a to zero.

The squared norms of errors e_c and e_a are defined independently as follows,

$$K_1 = \frac{1}{2}\left\|e_c\right\|^2,\ K_2 = \frac{1}{2}\left\|e_a\right\|^2$$

2.4.2 Learning Algorithm

A tuning law for the critic weights estimates $\hat{W}_c$ is obtained by applying gradient descent on K_1 and normalization as

$$\dot{\hat{W}}_c = -\alpha_c\frac{1}{\big(1+\sigma^{\mathrm{T}}\sigma\big)^2}\frac{\partial K_1}{\partial \hat{W}_c} = -\alpha_c\frac{\sigma}{\big(1+\sigma^{\mathrm{T}}\sigma\big)^2}e_c^{\mathrm{T}}, \tag{2.17}$$

where $\sigma := e^{-\gamma T}U(t)\otimes U(t) - U(t-T)\otimes U(t-T)$, while $\alpha_c \in \mathbb{R}^+$ is a gain that is constant and controls the critic weights estimates' convergence speed.

The tuning law for actor weights estimates will have a non-periodic nature because tuning happens only at times of triggering. Hence, the actor weights estimates are updated based on an impulsive system [20] as follows

$$\begin{cases} \dot{\hat{W}}_a & = 0, \qquad \forall t \in (r_j, r_{j+1}] \\ \hat{W}_a^+ & = \hat{W}_a - \alpha_a \frac{1}{(1+x_{\text{aug}}(t)^{\text{T}} x_{\text{aug}}(t))} \frac{\partial K_2}{\partial \hat{W}_a} \\ & = \hat{W}_a - \alpha_a \frac{x_{\text{aug}}(t)}{(1+x_{\text{aug}}(t)^{\text{T}} x_{\text{aug}}(t))} e_a^{\text{T}}, \qquad t = r_j, \end{cases} \tag{2.18}$$

where $\alpha_a \in \mathbb{R}^+$ is a gain that is constant and controls the actor weights estimates' convergence speed. Defining weights estimation errors as $\tilde{W}_c := W_c - \hat{W}_c$ and $\tilde{W}_a := -Q_{x_{\text{aug}} u_d} Q_{u_d u_d}^{-1} - \hat{W}_a$, their dynamics are derived as

$$\dot{\tilde{W}}_c = -\alpha_c \frac{\sigma \sigma^{\text{T}}}{\left(1 + \sigma^{\text{T}} \sigma\right)^2} \tilde{W}_c, \tag{2.19}$$

and

$$\begin{cases} \dot{\tilde{W}}_a = 0, \qquad \forall t \in (r_j, r_{j+1}] \\ \tilde{W}_a^+ = \tilde{W}_a - \alpha_a \frac{x_{\text{aug}}(t) x_{\text{aug}}(t)^{\text{T}}}{(1+x_{\text{aug}}(t)^{\text{T}} x_{\text{aug}}(t))} \tilde{W}_a \\ \qquad\qquad - \alpha_a \frac{x_{\text{aug}}(t) x_{\text{aug}}(t)^{\text{T}}}{(1+x_{\text{aug}}(t)^{\text{T}} x_{\text{aug}}(t))} \tilde{Q}_{x_{\text{aug}} u_d} Q_{u_d u_d}^{-1}, \ t = r_j. \end{cases} \tag{2.20}$$

2.5 Simulations

This section talks about the application of the proposed algorithm on the experimental vehicle to make it track a reference trajectory in simulations. The vehicle employs a split controller [32] where the trajectory tracking (high-level) control algorithm is separated from the management of the vehicle dynamics using a low-level controller. The high-level controller commands a desired linear and angular velocity and the purpose of the low-level controller

is to achieve them. The advantage of the split-controller is when moving on to a full-scale highway vehicle or to a vehicle with different dynamics, it is enough to just change the low-level controller keeping the high-level controller unchanged. In order to realize the desired velocities, the low-level controller runs at a much higher rate compared to the high-level controller.

2.5.1 Low-level Controller

A differential drive vehicle is a two-wheeled drive system with one wheel on each side and independent actuators for each wheel. If v_L and v_R are the left and right wheel velocities of the differential drive system and v and ω denote the system's linear and angular velocity, they are related as follows

$$v_L = v - \omega L/2$$
$$v_R = v + \omega L/2$$

where L is the width of the system's wheelbase. The vehicle considered is a differential drive system but with two wheels on each side, and the drag of the extra wheel modifies the above relation as below

$$v_L = v - \omega L$$
$$v_R = v + \omega L$$

which was found by testing the turning accuracy of the vehicle on flat ground.

A PWM signal is sent to each motor to control its velocity. So, the ultimate goal of the low-level controller is to accept the desired left and right wheel velocities from the high-level

controller and send a PWM signal to each motor so that the desired velocities are achieved. A PID velocity controller is employed as the low-level controller to track the commanded wheel velocities as shown below.

$$e(t) = v_{Ld}(t) - v_L(t) \tag{2.21}$$

$$u(t) = u(t - T_{ll}) + k_p(e(t) - e(t - T_{ll})) + k_i e(t) T_{ll} + \frac{k_d}{T_{ll}}(e(t) - 2e(t - T_{ll}) + e(t - 2T_{ll})) \tag{2.22}$$

where v_{Ld} is the desired left wheel speed, v_L is the measured left wheel speed, $u(t)$ is the PWM duty-cycle (control input), T_{ll} is the time-period of the low-level controller and k_p, k_i, and k_d are the controller gains determined by tuning. The above controller is applied to each of the motors.

2.5.2 High-level Controller

The trajectory tracking control algorithm designed in Chapter 2 considered to be used as the high-level controller is applicable only to linear systems. Because the vehicle has a non-linear model, for the chosen controller to be applicable, tracking in x-direction and y-direction is considered separately which would help in the following way.

Let x denote the position, v_x the speed and u_x the acceleration of the vehicle in the x-direction. Then the motion model in the x-direction is given by,

$$\begin{pmatrix} \dot{x} \\ \dot{v_x} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ v_x \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u_x \tag{2.23}$$

which is a linear system on which the controller can be used. The controller gives the required acceleration in the x-direction, given the current position and speed, and the target position and speed in the x-direction. The rate at which the desired acceleration is updated is equal to the rate at which the high-level controller runs. The above arguments also apply in the y-direction. In experiments, the current position and speed in both directions are supposed to be obtained using onboard sensors but in simulations, they are considered to be known.

The high-level controller independently gives the desired accelerations in x and y-direction which need to be converted into desired left and right wheel velocities for the low-level controller. If u_x and u_y denote the desired accelerations in x and y-direction respectively, $1/T_{hl}$ denotes the rate at which the high-level controller runs, v_{x0} and v_{y0} are the current speeds of the vehicle in x and y-direction respectively, then the desired speeds in x and y-direction until the next iteration of the high-level controller can be calculated as

$$v_{xd} = v_{x0} + u_x T_{hl}$$

$$v_{yd} = v_{y0} + u_y T_{hl}$$

which in turn can be converted into desired linear and angular speeds using the following equations,

$$v_d = \sqrt{v_{xd}^2 + v_{yd}^2}$$

$$\omega_d = \frac{v_{xd} u_y - v_{yd} u_x}{v_{xd}^2 + v_{yd}^2}$$

From the desired linear and angular speeds, desired left and right wheel speeds can be

obtained as follows,

$$v_{Ld} = v_d - \omega_d L \tag{2.24}$$

$$v_{Rd} = v_d + \omega_d L \tag{2.25}$$

where L is the width of the vehicle's wheelbase. These desired wheel speeds are fed into the low-level controller which tries to track them by calculating and applying a PWM duty cycle to each motor.

2.5.3 Updating Target Point

For the purpose of simulations, a set of GPS coordinates defining a Cassini Oval shaped race track is considered. In order to make the tracking happen at a constant linear speed, target points that are equally spaced are considered on the track because the target updates at a constant rate. The value of this equal spacing (look-ahead distance) is obtained by multiplying the desired constant linear speed with the time between consecutive target updates dt.

$$lh\_dist = des\_speed * dt \tag{2.26}$$

where lh_dist is the look-ahead distance and des_speed is the desired constant linear speed. So, assuming that the vehicle starts at some point on the track, its first target point should be lh_dist ahead, its second target point should be lh_dist euclidean distance ahead from its first target and so on if it has to move at a constant linear speed. To generate target points which are lh_dist distance apart, the original points which define the race track with 0.4 meters spacing are converted into points with lh_dist spacing based on des_speed

and dt. The equally spaced points on the reference trajectory are the target positions in x and y-direction updated every dt seconds from one to its next. But the target speeds in both directions are also needed for the trajectory tracking control in this case, which are determined as follows.

Let (x_r, y_r) denote the current position of the vehicle and (x_1, y_1) denote the current target position on the race track. So, the current target position in the x-direction is x_1 and that in the y-direction is y_1. Consider the future target positions to be (x_2, y_2),(x_3, y_3),(x_4, y_4),... These points are equally spaced and as discussed earlier, this spacing is equal to lh_dist meters. In order to find the current target speeds in x and y-direction, the speeds the vehicle should have at (x_1, y_1) in x and y-direction to achieve tracking should be determined. Because the vehicle should cover the gap between (x_1, y_1) and (x_2, y_2) in dt secs, the speed of the vehicle at (x_1, y_1) should be $(x_2 - x_1)/dt$ in the x-direction and $(y_2 - y_1)/dt$ in the y-direction. The target speeds at the future target positions are also determined in a similar manner.

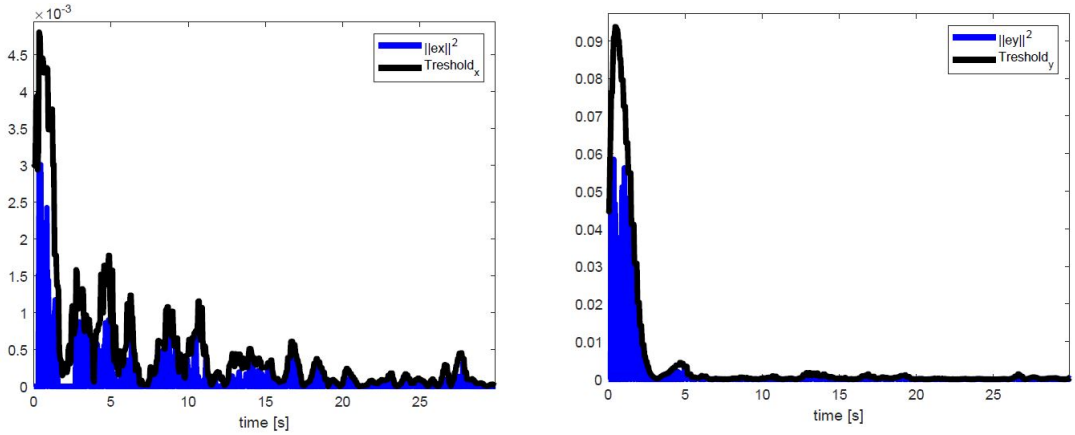
Now that everything required for performing the simulations is present, the next subsection talks about the simulation parameters and results obtained.

2.5.4 Simulation Parameters and Results

As mentioned earlier, a set of 2D points that define a Cassini Oval shaped race track is taken as reference. This race track which is approximately 400m long is at the Utah State University Electric Vehicle & Roadway Research Facility. The x-coordinates of all point are grouped to form the reference positions in the x-direction and similarly in the y-direction. The vehicle is expected to track the targets in x and y-direction independently which are updated periodically, resulting in 2D navigation along the track. For this, the learning

strategy is run in both directions giving two required event-triggered control inputs, one in each direction, which are none other than the desired accelerations in x and y-direction.

The simulation is run for a duration of 200 seconds with high-level and low-level controller rates of 20 Hz and 100 Hz respectively. The targets in x and y-direction are updated every 0.05 secs and the desired constant linear speed is chosen to be 4 m/s which makes the look-ahead distance to be 0.2 m. The initial position of the vehicle is chosen on the track as shown in Fig. 2.4 with zero initial speeds in both directions. Initial values of the critic and actor weights are assigned randomly in the range $[-1, 1]$. The user-defined parameters are chosen as $L = 17,\ \beta = 0.6,\ \gamma = 2,\ R = 0.01,\ Q = 0.001I_2$, and the weights tuning gains as $\alpha_a = 0.001,\ \alpha_c = 10$. During the first 50s of the simulation, probing noise is added to the control inputs u_x and u_y. The evolution of the triggering gaps and thresholds in both

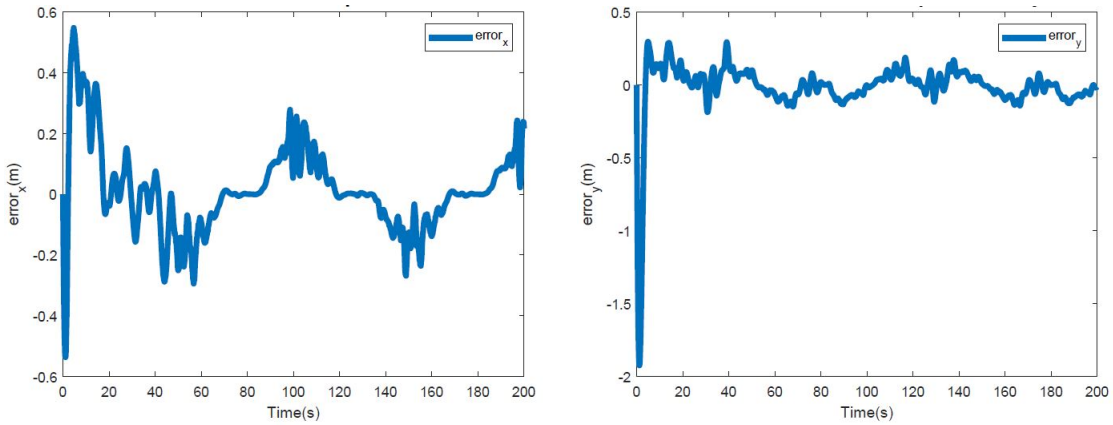


(a) x-direction (b) y-direction

Figure 2.1: Evolution of the thresholds and triggering gaps for the first 30s

directions for the first 30s can be seen in Fig. 2.1 which clearly shows that triggering occurs when the error reaches the threshold. Triggering happens 2392 and 2311 times in x and y-direction respectively out of 4000 iterations of the high-level controller.

Position errors in x and y-direction are plotted in Fig. 2.2, and the plots of actual and desired states are in Fig 2.3. The error plots show that the learning algorithm is effective



(a) x-direction (b) y-direction

Figure 2.2: Position errors in x and y-direction

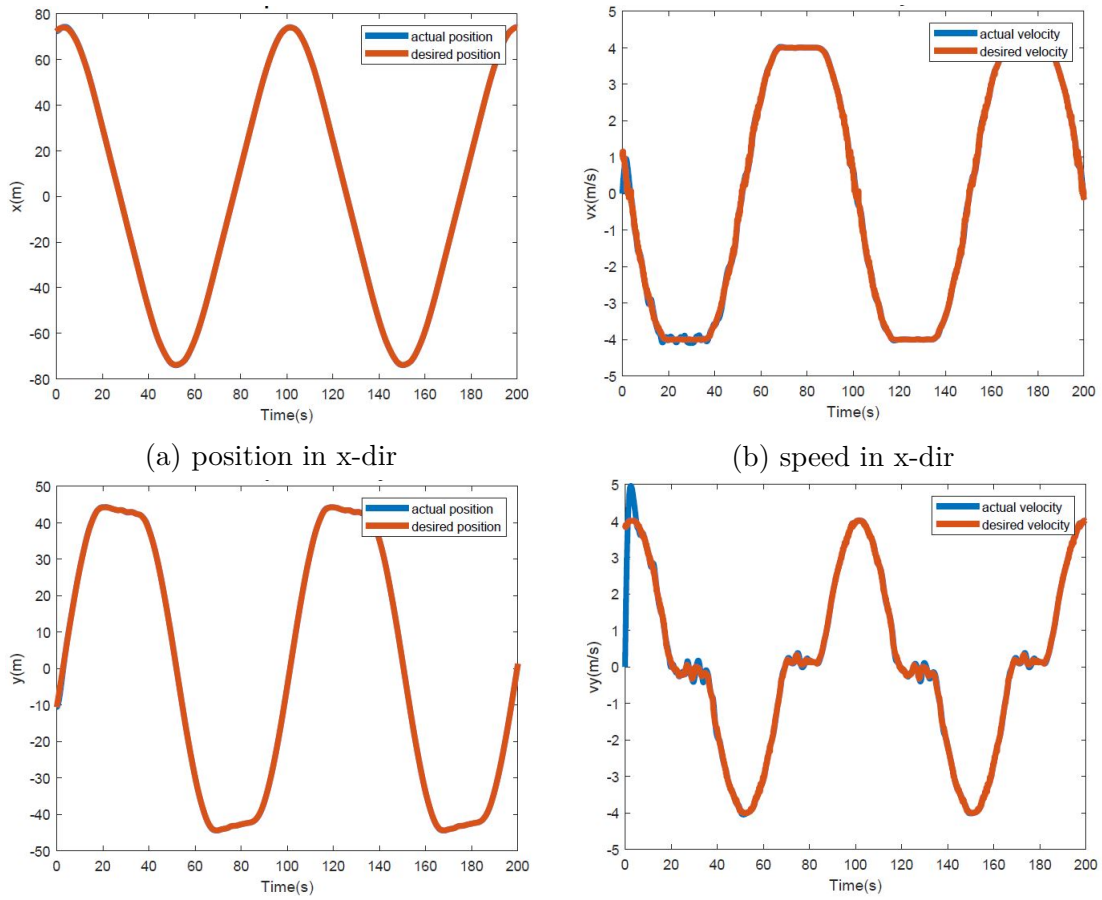
Table 2.2: Average position, speed and cross-track errors for various desired constant linear speeds - single vehicle simulation.

Desired speed(m/s)	Position error(m)	Speed error(m/s)	Cross-track error(m)
2	0.1101	0.0282	0.0894
4	0.1872	0.0625	0.1415
6	0.2573	0.0907	0.2013
8	0.3080	0.1206	0.2461

at tracking the desired trajectory with low errors even at speeds of 4m/s ($\approx 9mph$). The initial high values of errors are because of the probing noise added to the control inputs and because of the online learning. It can be noticed that the errors tend to be high when the vehicle tries to track the curvy sections of the track when compared to the straight sections where the tracking error is almost zero.

The actual path taken by the vehicle and the desired path is shown in Fig. 2.4. The actual and reference linear speeds of the vehicle are plotted in Fig. 2.5, and the cross-track error with respect to the reference trajectory is in Fig. 2.6.

Without any apriori information regarding the motion model of the vehicle and with aperiodic updates of the control inputs, the proposed model-free event-triggered trajectory track-



(a) position in x-dir

(b) speed in x-dir

(c) position in y-dir

(d) speed in y-dir

Figure 2.3: Actual and desired states in x and y-direction

ing optimal controller could make the vehicle track its 2D reference trajectory (race track) with online learning.

The linear speed plot shows that the vehicle tracks the desired trajectory at a constant speed after starting from zero initial speed. The cross-track error follows a pattern similar to the position errors when tracking the curvy and straight sections of the track. Table 2.2 shows the values of average position error, speed error and cross-track error for different desired constant linear speeds.

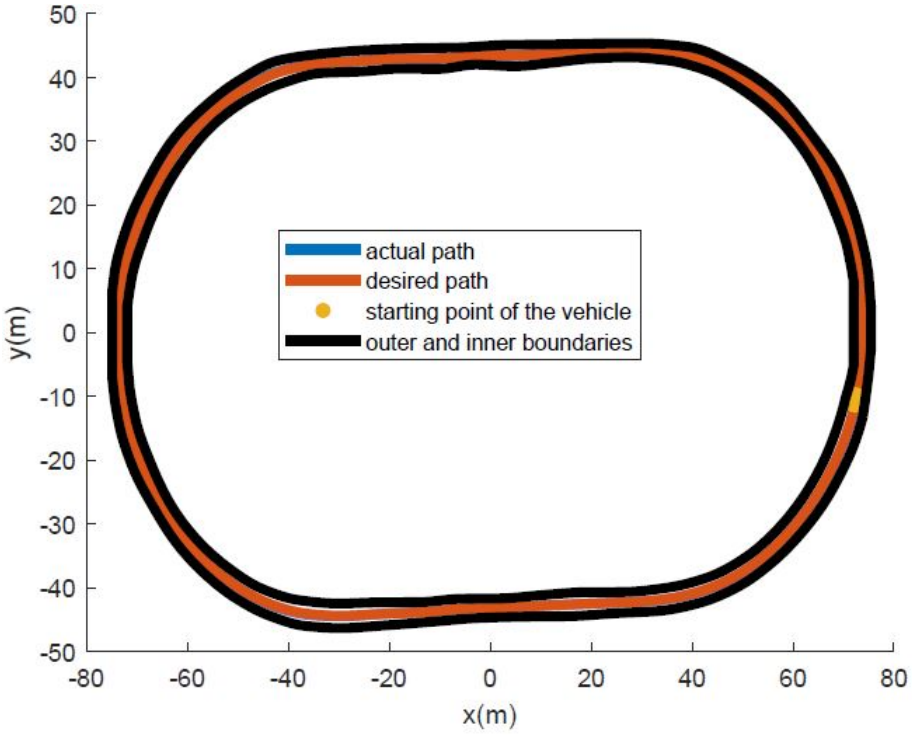


Figure 2.4: The actual and desired path of the vehicle on the race track. The yellow mark is the starting point of the vehicle.

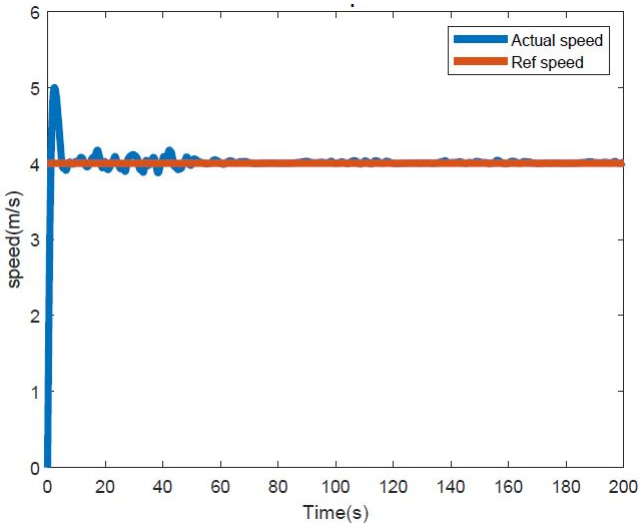


Figure 2.5: Actual and desired linear speed of the vehicle.

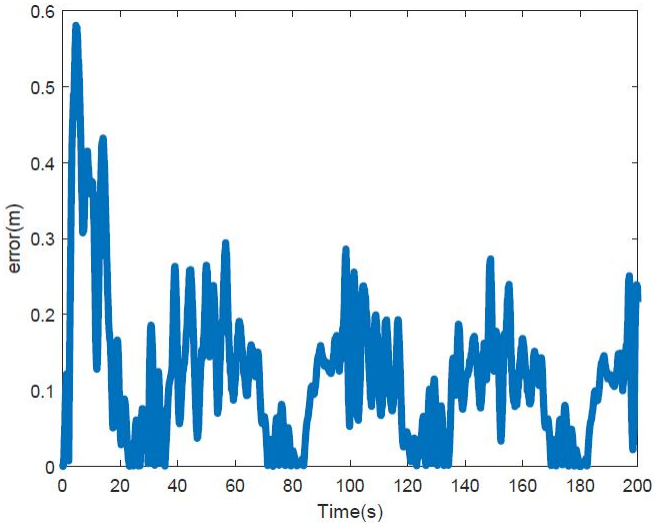


Figure 2.6: Cross-track error w.r.t. the desired trajectory

Chapter 3

Experimental Platform Design and Development

The simulation results show that the model-free event-triggered trajectory tracking control algorithm works well. But experimental validation on a physical platform is necessary to make sure that the proposed method works in practice.

A previously developed Unmanned Ground Vehicle (UGV) platform was studied in an effort to validate the simulation results in experimentation. The vehicle platform considered is approximately 1/10th the size of a standard car. This platform was originally designed and developed by Samuel Mitchell and Daniel Dunn. But I and Kevin Kawecki have made few major changes to this system according to the requirements of this research. This chapter mainly focuses on the design of two major segments of an experimental platform, which are hardware and software.

The following sections talk about the hardware and software of the vehicle platform, and the major changes made to the already existing system.

3.1 Hardware

The vehicle studied and used for experimental validation was purchased from Battlekits [1]. It is capable of moving at speeds of up to 10m/s which satisfies the requirement of the platform being analogous to a standard highway vehicle. Also, the vehicle is collision tolerant which makes experimentation more feasible. Using this kind of a scaled vehicle rather than a full-scale one will be cost effective keeping in mind the collisions that might happen while testing.

More details about the hardware are discussed in Appendix A.

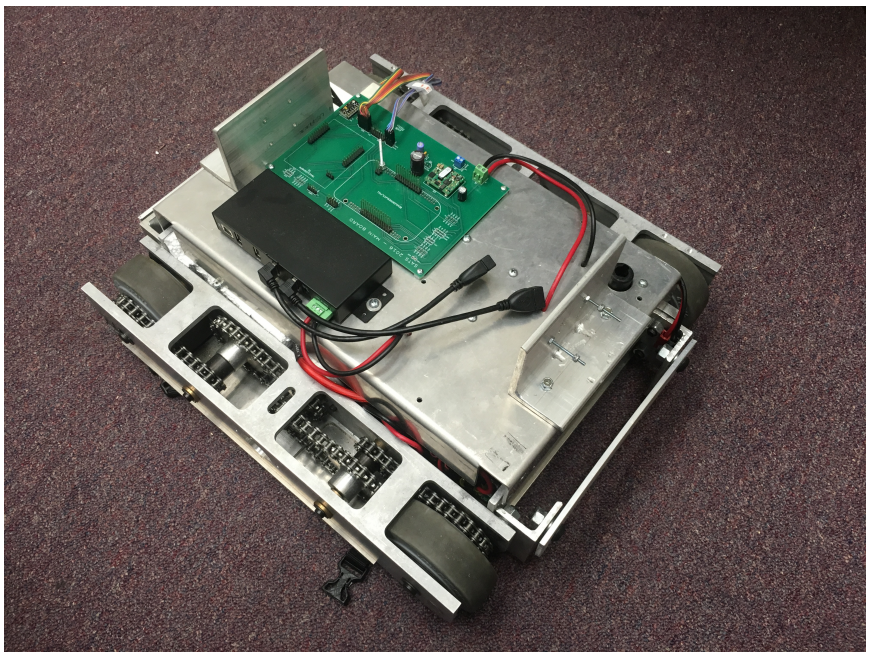


Figure 3.1: The experimental vehicle platform. There is one motor on each side of the vehicle. It can travel at speeds up to 10m/s.

3.2 Software

3.2.1 Robot Operating System (ROS)

Robot Operating System (ROS) is a collection of software frameworks for the development of software on robots [44]. It is a middle-ware usually written in Python/C++ which allows the passing of messages between tasks (nodes), flexibility in hardware and scheduled implementation of tasks. It was chosen for this work because of its open-source nature, huge community support, flexibility, and simple interface. The software system structure implemented in ROS is shown in Figure 3.2.

Initially, the software system used to operate on a low-cost, community-supported development platform called BeagleBone Black with the Ubuntu distribution of Linux. It had an AM335x 1GHz ARM Cortex-A8 processor. But recently, the development platform was changed to UP-Board with 1.92GHz Intel ATOM x5-Z8350 processor on which the system currently operates, the main reason for this change is the better processing speed of the UP-board. The UP-board also has an on-board Graphics Processing Unit (GPU) which can come handy when performing image processing in the future, for example, to detect lanes and objects.

3.2.2 Nodes/Tasks

Inside ROS various tasks like reading and processing data from a sensor, using sensor data in the control algorithm to calculate required control input, writing PWM signals to actuators are done in the form of Python/C++ scripts called as nodes. This sub-section describes tasks performed in various nodes of the software system. This collection of tasks to execute on the vehicle are placed in a package called *sats_car_ros*. Links called as topics are used

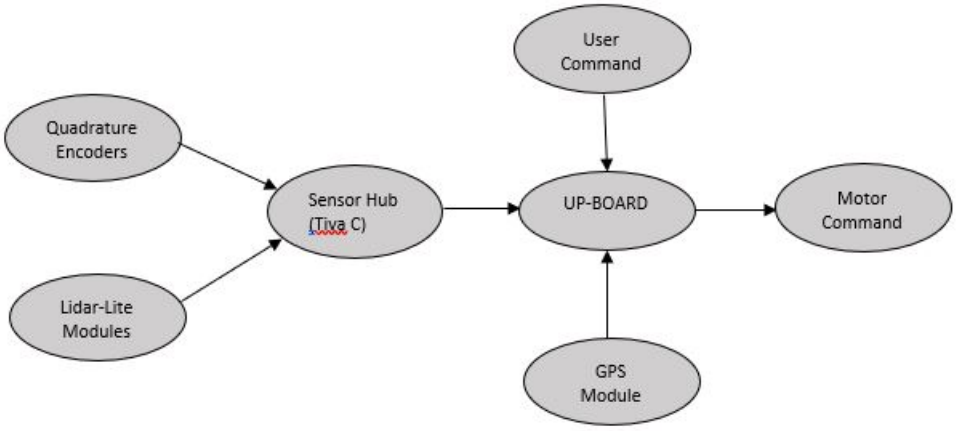


Figure 3.2: Data flow on the experimental vehicle platform. The UP-board receives sensor data from the sensor hub (Tiva C) and the GPS module, user commands from the command station, calculates the output commands and writes them to files from which the motor-drivers read them.

to send information from one node to another in the form of messages so that their respective tasks can be performed.

Controller Node The controller node executes the code for both the high-level trajectory tracking controller and the low-level PID controller. It accepts information about the vehicle's position and speed in x,y-directions which are required by the high-level controller to calculate the desired left and right wheel speeds. This node also needs information about the preceding vehicle's position and speed in the case of platooning. In addition, it takes in the current wheel speeds and uses them together with the desired wheel speeds produced by the high-level controller to implement the low-level PID controller which calculates the required PWM output. The PWM duty-cycle value is read by the motors from files to which the controller node writes the calculated PWM value.

Detailed information about the implemented high-level and low-level controllers is in Chapter 2. The controller node gathers all the information required by the high-level and low-level

controllers from the Sensor node and GPS node. In the case of platooning, the lead vehicle follows a GPS path which is predefined while all the other vehicles follow a reference path generated by their respective preceding vehicles as will be discussed in Chapter 5.

Sensor Node The sensor node reads information from the sensor hub (Tiva C), verifies and processes the received information, and publishes it onto various topics to which the controller node subscribes. The sensor hub (Tiva C) receives information from the Quadrature Encoders which measure the wheel velocities and for use in the platooning case, the Lidar-Lite units which measure relative distance and speed between vehicles.

Each message from the sensor hub has three sections: start code 1Byte, sensor data 24Bytes, and CRC 2Bytes. The checksum is calculated on the first 25Bytes using the CRC-CCITT method.

When a proper message which satisfies the checksum is received by the sensor node, it processes the data and publishes it onto a topic in the form of a formatted message. The controller node subscribes to this topic to receive some of the required measurements. When an erroneous message is received by the sensor node, it is discarded.

GPS Node This node provides information about the vehicle's GPS coordinates and speed in x,y-directions to the controller node. It accesses and processes data from the GPS module through USB. The package which implements this node was taken directly from the internet [3], which is one of the major advantages of ROS. ROS packages for almost any robotic application can be found online. The package used basically parses NMEA [4] strings produced by the GPS module and publishes a very simple GPS message.

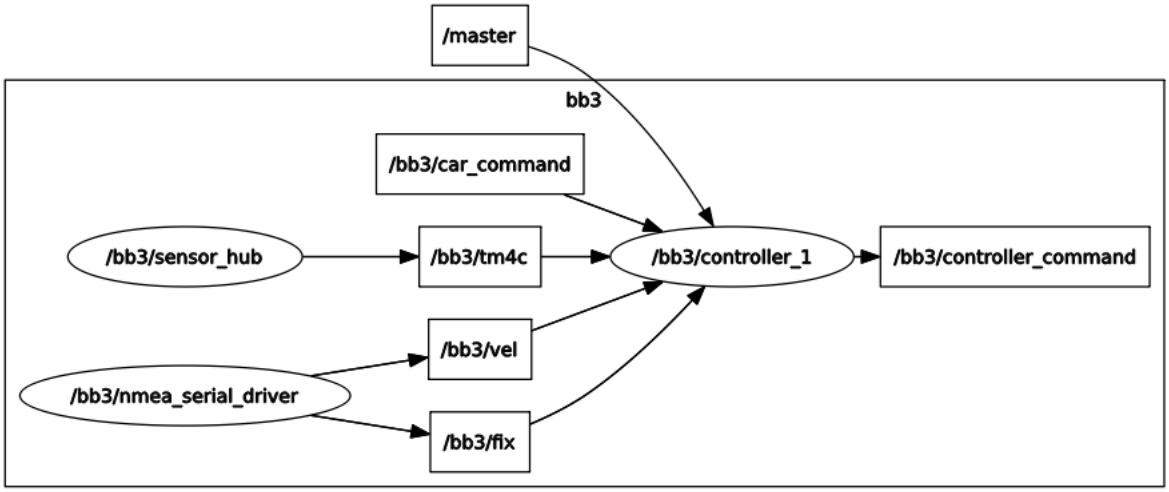


Figure 3.3: The interconnection of the ROS nodes on the UP-board. The vehicle operates within its own namespace (e.g. /bb3).

3.2.3 Safety

The controller node on the vehicle subscribes to a common topic called "/master". This topic contains a bool type variable using which the PWM values can be forced to a value that results in switching off the motors and stopping of the vehicle even though the desired PWM values are being calculated by the controller.

In order to safely stop the vehicle when ROS execution is halted, the PWM values are set to stationary by making use of a ROS functionality which enables to run commands when exiting a node. This also provides safety when the controller node encounters an error and exits but the motors maintain their most recent speeds.

An emergency shutdown function was also implemented using XBee modules which can come handy when the wireless connection is weak/lost and the vehicle continues to operate without commands from the command station.

3.3 Command Station

A command station which is a MacBook Pro running Ubuntu distribution of Linux operates the vehicle through a wireless connection. The vehicle and the command station are connected to an ASUS router wirelessly. The command station is also used to obtain and visualize data from the vehicle.

The vehicle receives configuration commands from the base (command station), which initialize the ROS nodes on the vehicle. Once the nodes are initialized, the vehicle performs all the required calculations on its board. In case of emergency, the base station can send user commands to stop the vehicle from moving as described in Subsection 3.2.3.

3.3.1 Wi-Fi Router

An ASUS AC1900 Dual Band 802.11ac Gigabit wireless adapter is used as the Wi-Fi access point which can operate at both 2.4 GHz and 5 GHz. The 5 GHz operating frequency was chosen in this work because of its faster data rates at shorter distances compared to 2.4 GHz. It has three detachable 9 dB antennas which expand the network's range and deliver signal to required areas. Signal strengths of up to 92% could be obtained at a distance of 10m using this wireless adapter at 5 GHz. And no internet is needed while configuring the vehicle from the command station using this access point.

3.4 Miscellaneous

Earlier, the electronics power system and the motor electrical system used to be powered by the same set of batteries and so were not isolated from each other, which made the

Beaglebone shut down when the motors drew a large amount of current or when the battery voltage was low. A new Printed Circuit Board (PCB) was designed by Kevin Kawecki for the UP-board in which the systems are powered by their own set of batteries making them isolated and avoiding the shutdown issue.

The ROS package *sats_car_ros* has a collection of tasks/nodes required to operate the vehicle and provides a simple interface. The *rqt* framework in ROS can be used to visualize real-time control and sensor data from the vehicle on the command station.

With all the required hardware and software components to implement the trajectory tracking control algorithm, this experimental platform allows the testing of the simulations in Chapter 2.

Chapter 4

Experimental Validation

A testing platform of scaled autonomous vehicle as discussed in Chapter 3 was developed to validate the simulation results experimentally. Providing a means of testing the theoretical findings on a physical system is the purpose of this platform. As mentioned in Chapter 2, the vehicle is similar to a differential drive system driven by DC motors and can reach speeds of up to 10m/s. The wheel velocities were determined using Quadrature encoders. The vehicle had an onboard micro-controller to implement the control algorithm.

A split-controller structure was used where the trajectory tracking controller in the form of a high-level controller commands desired speeds for left and right wheels while a low-level (PID) controller converts these desired speeds into PWM duty cycles to write to the motor controller so that the commanded speeds can be tracked. The vehicle obtains its precise position and speed in each direction using an onboard RTK-GNSS module which employs the RTK positioning technique. The idea is to validate the control algorithm on a single vehicle first and then move on to platooning like in the case of simulations.

4.1 Single Vehicle Experiments

To start with, the 3rd-floor corridor of Whittemore Hall was chosen to perform the trajectory tracking experiments. A path was chosen in the corridor as shown in Fig. 4.1 and its GPS coordinates were taken to provide a reference to the vehicle in order to track it. Length of

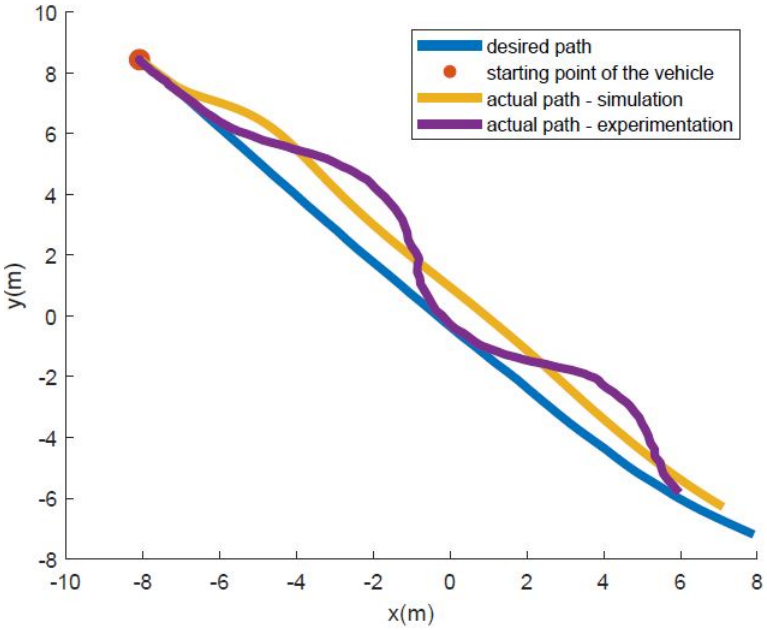
the path is approximately 24m. The vehicle is supposed to track the updating targets in x and y-direction independently resulting in 2D trajectory tracking.

Table 4.1: Low-level PID controller gains with low settling time and overshoot.

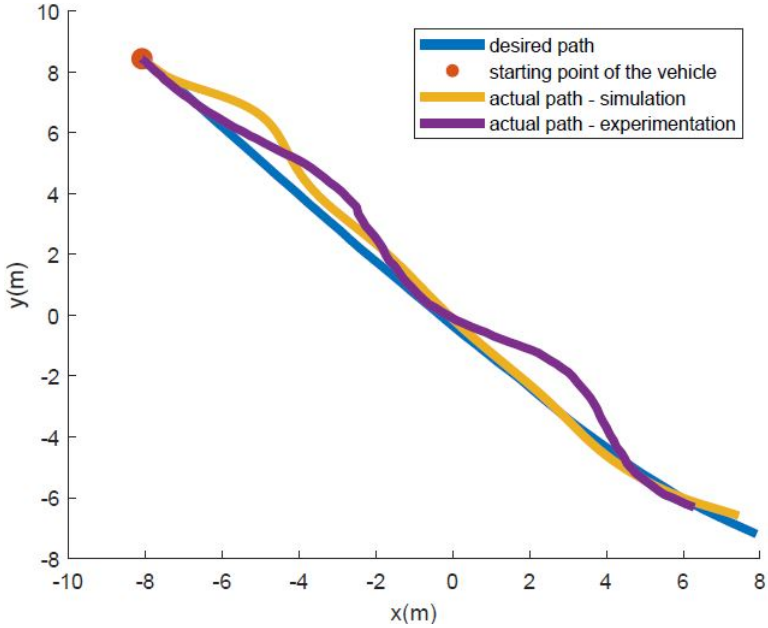
Gain parameter	Value
k_p	2.35
k_i	6.25
k_d	0.04

Figure 4.1: Path chosen for experiments in the corridor of the 3rd-floor - Whittemore Hall

The experiment is run for a duration of 22s with high-level and low-level controller rates at 5 Hz and 10 Hz respectively. The target update rate is 2.5 Hz in both directions and

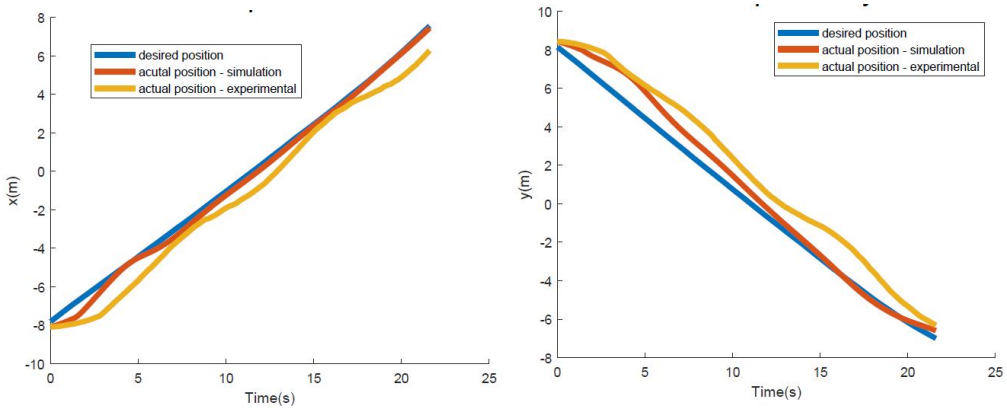


(a) Iteration 1



(b) Iteration 3

Figure 4.2: Actual and desired paths of the vehicle in simulations and experiments.



(a) x-direction (b) y-direction

Figure 4.3: Actual and desired positions of the vehicle during iteration 3.

the desired constant linear speed is set to 1 m/s which makes the look-ahead distance to be 0.4m. The initial position of the vehicle is shown in Fig. 4.1 by the cross mark and it starts with zero initial speeds in both directions.

Initial critic and actor weights are chosen randomly in the range $[-1, 1]$. The user defined parameters are chosen as $L = 17,\ \beta = 0.6,\ \gamma = 2,\ R = 0.01,\ Q = 0.001I_2$, and the weights tuning gains as $\alpha_a = 0.001,\ \alpha_c = 10$. Probing noise is added to the control inputs during the first 5.5s of the simulation. The linear speed of the vehicle is saturated to 1.1m/s for the initial testing phase.

The sensor hub reports data at a rate of 50 Hz and the RTK-GNSS module at a rate of 5Hz. The length of the vehicle's wheelbase is 0.319m.

Values of the low-level controller gains k_p, k_i and k_d are shown in Table 4.1. It was realized during the experimentation that the chosen GNSS module does not report speed measurements when the rover is moving at linear speeds less than 0.5m/s in which case the speed measurements are assumed to be equal to the last measured values for now. Because the length of the path (24m) is not long enough for the ideal weights to be learned in one iteration, multiple iterations of the experiment are performed using the weights learned from one

iteration as the initial weights for the next iteration.

The experimental tracking results obtained are as shown in Fig. 4.2 for multiple iterations. The figure also shows the simulation results obtained in MATLAB for the same scenario with experimental constraints included.

The path plots show that the results get better with iterations but for yet unknown reasons the actual path of the vehicle is oscillating close to the reference path. Further iterations above three didn't improve the experimental results. The plots of actual and desired positions in x and y-directions are in Fig 2.3 for iteration 3. Going ahead, the idea is to choose a reference path that is longer, avoiding the discontinuity in learning and also to do more research into finding the reasons for the oscillations.

Chapter 5

Platooning Control

5.1 Overview

The application of the trajectory tracking control algorithm developed in Chapter 2 and reference trajectory generation for a group of vehicles (similar to the experimental vehicle platform) to achieve platooning with constant inter-vehicle spacing is talked about in this chapter. It is assumed that each vehicle in the platoon except the lead vehicle i.e., each following vehicle has information regarding the relative position and relative speed to its preceding vehicle in determining its target position and speed. Polynomial interpolation is used to determine and update the reference of each following vehicle with time.

It is shown through simulation results that a constant inter-vehicle spacing policy can satisfy the string stability performance requirement if all the vehicles in the platoon receive the lead vehicle's speed information and use it in their respective controllers. Constant spacing policy can achieve very high traffic throughput, whereas the throughput is limited for constant time-headway policy where the desired inter-vehicle spacing depends on the speed of the vehicles and is variable.

The structure of this chapter is as follows. The reference trajectory generation technique for the following vehicles using their respective preceding vehicles' position and speed data is discussed in Section 5.2. Section 5.3 shows the incorporation of the leader's speed data

into the tracking control algorithm for the following vehicles. Application of the control algorithm on all the vehicles of the platoon to track their respective reference trajectories and to achieve platooning with constant inter-vehicle spacing is presented in the form of simulations in Section 5.4.

5.2 Reference Trajectory Estimation for Following Vehicles

In this estimation strategy, each following vehicle measures the relative position and relative speed in x and y-direction to its preceding vehicle. The following vehicles use on-board lidars to get these measurements in practice. It is assumed that the trajectory of the lead vehicle is feasible to all the vehicles behind.

Considering a coordinate system fixed to the ground, the current position and speed of the following vehicle in x-direction is represented as $X(t) = [x(t), v_x(t)]^T$. Similar analysis also follows in the y-direction. The states of the vehicle preceding $X_p(t) = [x_p(t), v_{xp}(t)]^T$ are calculated by the vehicle following making use of its current states and the measurement data obtained from the sensors. The preceding vehicle's position and speed information is stored in the following vehicle's memory to obtain intermediate points by means of interpolation at required time instants.

5.2.1 Constant Distance Spacing

Inorder to make the following vehicle maintain a constant distance of say L_f relative to its predecessor, it can be made to track a point on the predecessor's path that is at distance L_f from the predecessor's current position. Distance traveled by the preceding vehicle at time

t given by a distance function $L_p(t)$ is defined as

$$L_p(t) = \int_0^t \sqrt{v_{xp}^2(\tau) + v_{yp}^2(\tau)}\mathrm{d}\tau \tag{5.1}$$

where $v_{yp}(t)$ is the speed of the preceding vehicle in y-direction at time t as calculated by the following vehicle.

At current time instant t_0 the following vehicle should track the interpolated position and speed of its preceding vehicle in x and y-direction at time $t = T_i$ where T_i is given by

$$L_p(T_i) := \begin{cases} L_p(t_0) - L_f, & L_p(t_0) \geq L_f \\ 0, & L_p(t_0) < L_f, \end{cases}$$

if it has to follow its preceding vehicle at a distance L_f.

Using the memorized preceding vehicle's position and speed data, and value of T_i calculated from $L_p(t)$ the following vehicle determines its target position and speed in x and y-direction at the current time instant t_0 to maintain an inter-vehicle distance of L_f with the help of interpolation. Then the trajectory tracking control algorithm is used by the following vehicle for target tracking, following the predecessor at a constant distance in a platoon.

5.3 Inter-vehicle Communication

Because the vehicles in the platoon are supposed to move with constant inter-vehicle spacing, using just a predecessor following information flow topology cannot guarantee string stability. Broadcasting at least leader's (the leading vehicle in the platoon) speed information to following vehicles in the platoon through V2V communication is required to ensure string stability. The following Subsection 5.3.1 talks about how the model-free event-triggered

trajectory tracking optimal control task is formulated for the following vehicles using also the leader's speed in x and y-direction.

5.3.1 Problem Formulation

Assuming that the dynamics of each following vehicle in the platoon can be expressed in the form of a pair of linear systems in x and y-direction like in the case of the leader, consider the following linear time-invariant continuous-time system

$$\dot{x}(t) = Ax(t) + Bu(t),\ x(0) = x_0,\ t \geq 0$$

here $x(t) \in \mathbb{R}^n$ is the state vector, $u(t) \in \mathbb{R}^m$ is the system's control input, while $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ are the system (or state) and input matrices. The A and B matrices will be considered unknown. In x-direction, the state vector would be position and speed, and the control input would be acceleration in that direction. Similarly, for the y-direction.

The reference path is modeled using the following equations,

$$\dot{z}(t) = A_d z(t),\ z(0) = z_0,\ t \geq 0$$

where $z(t) \in \mathbb{R}^n$ denotes the bounded reference trajectory and $A_d \in \mathbb{R}^{n\times n}$ the unknown path matrix, even though this approach will not require them. The actual reference is modeled into references in x and y-directions separately because one direction is considered at a time. For each following vehicle, the reference trajectory would be the trajectory generated using the estimation method in Section 5.2. Only for the leader, the reference trajectory is a set of GPS way-points defining a race track as shown in Section 2.5.

For each following vehicle to track its reference, the following error is defined in each direction

$e_{\text{track}}(t) := x(t) - z(t),\ t \geq 0$, with dynamics,

$$\dot{e}_{\text{track}}(t) = Ax(t) + Bu(t) - A_d z(t),\ t \geq 0 \tag{5.2}$$

Now, here is where the leader's speed information is also taken into consideration and incorporated into the Intermittent Learning algorithm. Together with position and speed error of the following vehicle with respect to its target position and speed, also considered is the speed error with respect to the leader's speed in each direction. The following new error vector in each direction is defined, $e_{\text{track1}}(t) := [e_{\text{track}}^{\text{T}} \quad x_2(t) - v_L(t)]^{\text{T}} \in \mathbb{R}^{n+1}$, where $x_2(t)$ is the speed of the following vehicle and $v_L(t)$ is the leader's speed.

The following steps are also performed in each direction separately. The augmented state $x_{\text{aug}} := [e_{\text{track1}}^{\text{T}} \quad z^{\text{T}}]^{\text{T}} \in \mathbb{R}^{2n+1}$ is considered and its dynamics are written as

$$\dot{x}_{\text{aug}}(t) = A_{\text{aug}} x_{\text{aug}}(t) + B_{\text{aug}} u(t),\ t \geq 0. \tag{5.3}$$

A sampled version of the augmented state is considered as follows in order to make the system save resources

$$\hat{x}_{\text{aug}}(t) := \begin{cases} x_{\text{aug}}(r_j), & \forall t \in (r_j, r_{j+1}] \\ x_{\text{aug}}(t), & t = r_j. \end{cases}$$

Error between the sampled and continuous version of the augmented state is defined as $e(t) := \hat{x}_{\text{aug}}(t) - x_{\text{aug}}(t),\ t \geq 0$. The goal here is to find an optimal controller u_d, called as the event-triggered (intermittent) controller which takes the form $u_d := k(\hat{x}_{\text{aug}}(t))$ and optimizes a cost function identical to the one with continuously updating control without any need of the plant (system) model nor of the reference.

$$J(x_{\text{aug}}(0); u_d) = \int_0^\infty e^{-\gamma\tau} \left(x_{\text{aug}}^{\text{T}} Q_{\text{aug}} x_{\text{aug}} + u_d^{\text{T}} R u_d \right) \mathrm{d}\tau, \tag{5.4}$$

with user-defined matrices $Q_{\text{aug}} := \begin{pmatrix} Q & 0_{(n+1)\times n} \\ 0_{n\times(n+1)} & 0_{n\times n} \end{pmatrix}$ and $R \succ 0$ which incorporate penalties associated with deviations from operating points and taking actions, respectively, where $0_{n\times n}$ is a zero matrix of size $n \times n$ and $Q \succeq 0$, and with $\gamma \in \mathbb{R}^+$ a discount factor.

Equivalently, the aim is to obtain optimal controller $u_d^\star$ such that,

$$J(x_{\text{aug}}(0); u_d^\star) \leq J(x_{\text{aug}}(0); u_d), \forall u_d,$$

which also can be expressed as the following optimization problem,

$$J(x_{\text{aug}}(0); u_d^\star) = \min_{u_d} J(x_{\text{aug}}(0); u_d)$$

given the dynamics in (5.3).

The ultimate goal is to find a value function that is approximately equal to the value function $V^\star$ for (5.3) with input as $u_c := k(x_{\text{aug}}(t))$ (the continuous sampled controller), defined as,

$$V^\star(x_{\text{aug}}(t)) := \min_{u_c} \int_t^\infty \frac{1}{2} e^{-\gamma(\tau - t)} \big(x_{\text{aug}}^{\mathrm{T}} Q_{\text{aug}} x_{\text{aug}} + u_c^{\mathrm{T}} R u_c\big) \mathrm{d}\tau, \ \forall t, \tag{5.5}$$

without any information of the motion and reference models, and given an event-triggered controller which is not periodic.

The main change made in the above formulation is to make the controller to also react based on the leader's speed which is incorporated using the extra error state. This helps the following vehicles to react faster than in the case of using just a predecessor following information flow topology in which case, only the errors in the states with respect to the preceding vehicle are used. The Intermittent Learning algorithm with the above modification in the formulation can be used on each of the following vehicles assuming that they have some way to get position and speed information of their respective preceding vehicles, and

speed information of the leader vehicle. The remaining parts of the controller design are similar to those in Chapter 2.

5.4 Multi-Vehicle Simulations

This section discusses the application of the proposed trajectory tracking control algorithm on a group of 5 vehicles each similar to the experimental vehicle platform to make them move in a platoon with constant inter-vehicle spacing and constant speed while staying on a race track. Each vehicle in the platoon uses a split-level controller similar to that in Section 2.5. A low-level controller to track the high-level commanded left and right wheel velocities is implemented on each of the vehicles in the form of a PID controller as described in Section 2.5. The target points for the leader come from the 2D points defining the race track and those for the following vehicles are determined from the paths of their respective preceding vehicles such that constant inter-vehicle spacing is maintained. The simulation environment for the leading vehicle is exactly similar to that used in Section 2.5.

5.4.1 Simulation Parameters and Results

The simulations are performed on the same track as that used in the single-vehicle simulations. As mentioned, the platoon consists of 5 vehicles, the dynamics of which are similar to those of the experimental vehicle platform. Each vehicle is expected to track its periodically updating targets in x and y-direction separately resulting in 2D navigation along the track.

With a low-level and high-level controller rate of 100 Hz and 20 Hz respectively on each vehicle, the simulation is run for a 200s duration. The target update rate is 20 Hz in both (x and y) directions. The desired constant linear speed of the lead vehicle is set to be 2

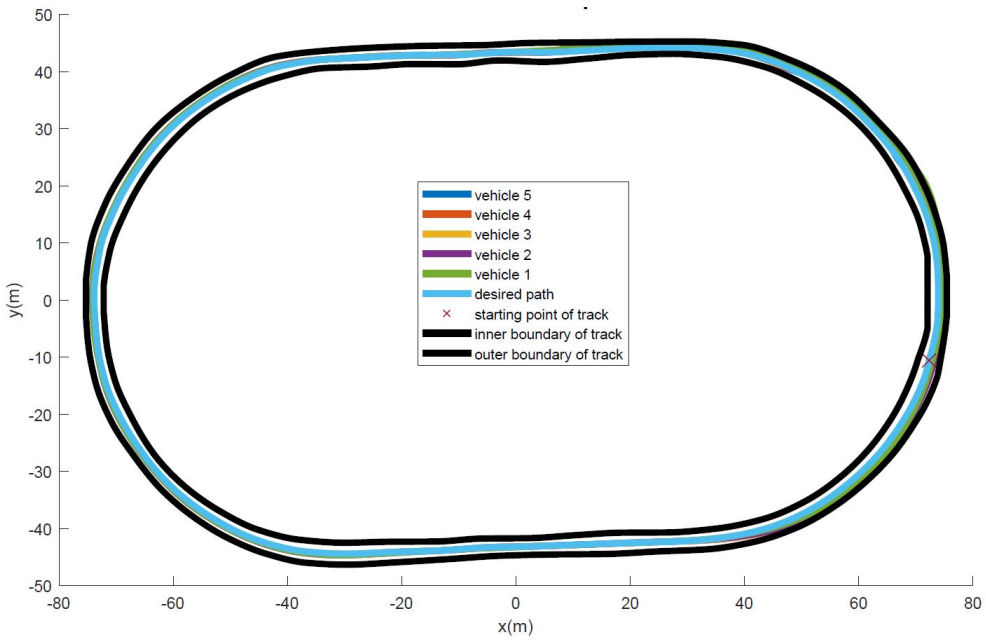


Figure 5.1: The actual paths of the vehicles on the race track and the desired path of the lead vehicle. The cross mark shows the starting point of the vehicles.

m/s for the first 150s and changed to 4 m/s for the remaining duration of the simulation. This change in speed simulation helps to assess the string stability of the platoon as will be discussed later.

Each vehicle starts from the same initial position on the track as shown in Fig. 5.1 with zero initial speeds in both directions. The actor and critic initial weights are chosen randomly in the range $[-1, 1]$. The desired constant inter-vehicle spacing is set to a value of 0.8m. The user defined parameters are chosen as $L = 17,\ \beta = 0.6,\ \gamma = 2,\ R = 0.01,\ Q = 0.001I_2$ for the lead vehicle and $Q = 0.001I_3$ for the following vehicles, and the weights tuning gains as $\alpha_a = 0.001,\ \alpha_c = 10$. During the first 50s of the simulation, probing noise is added to the control inputs u_x and u_y of each vehicle.

Fig. 5.1 shows a plot of all 5 vehicles' paths on the race track resulting from the simulation. It also shows the desired trajectory of the lead vehicle. In all the plots, vehicle 5 is the lead and vehicle 1 is the rear vehicle in the platoon.

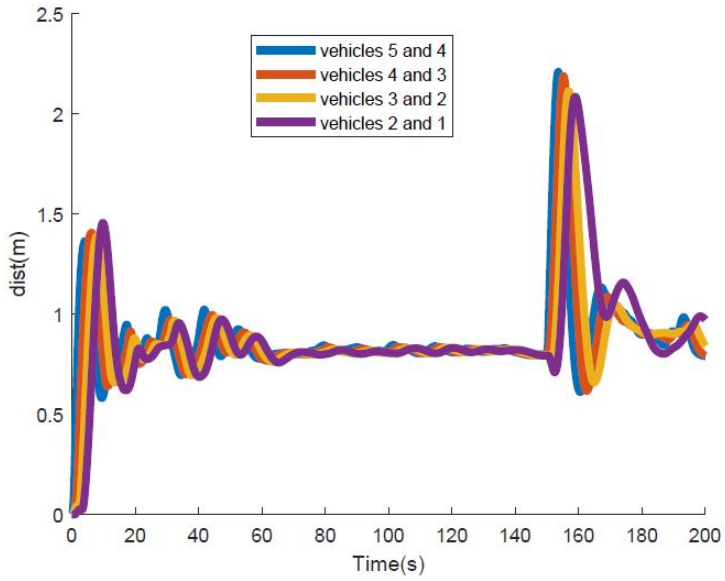


Figure 5.2: Inter-vehicle spacing between consecutive vehicles. The desired spacing, in this case, is 0.8m.

The inter-vehicle spacing between consecutive vehicles is shown in Fig. 3.2, which shows that the vehicles maintain the desired spacing, especially after the initial learning happens, proving the effectiveness of the proposed reference trajectory generation method for the following vehicles.

The linear speeds of the vehicles are plotted in Fig. 5.3 from which it can be seen that the vehicles move at the desired constant speeds. Figs. 5.4 and 5.5 show the plots of errors in position and errors in speed respectively of the vehicles with respect to their reference trajectories. At 150s, when the desired linear speed of the lead vehicle is changed from 2m/s to 4m/s, the errors increase but again they decrease gradually with time.

The error plots show that the platoon is string stable in terms of position error and speed error of the vehicles w.r.t. their respective references. The cross-track error plot of each vehicle w.r.t the reference track of the leader is shown in Fig. 5.6. The platoon isn't string stable in terms of the cross-track error because the proposed controller doesn't depend on

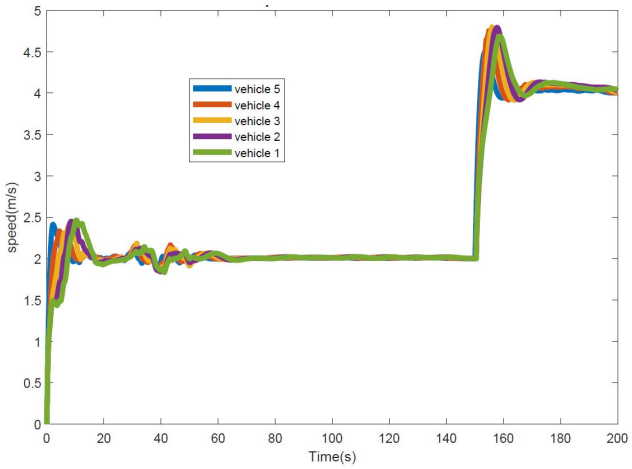


Figure 5.3: Linear speeds of the vehicles increased from 2m/s to 4m/s at 150s.

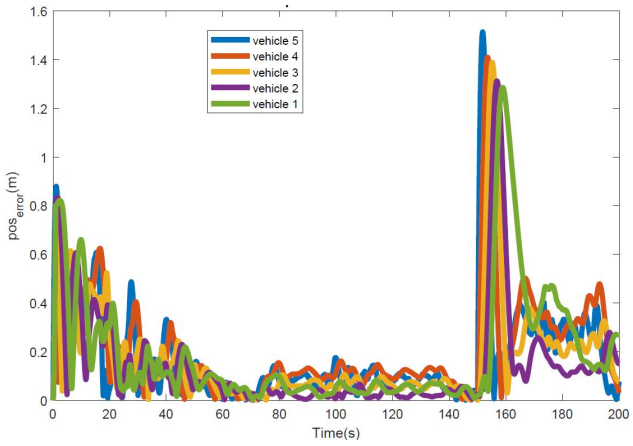


Figure 5.4: Error in the position of each vehicle w.r.t. its updating reference in terms of Euclidean distance.

this error in any way.

The numerical simulation results provide support to the theoretical findings in this chapter and Chapter 2.

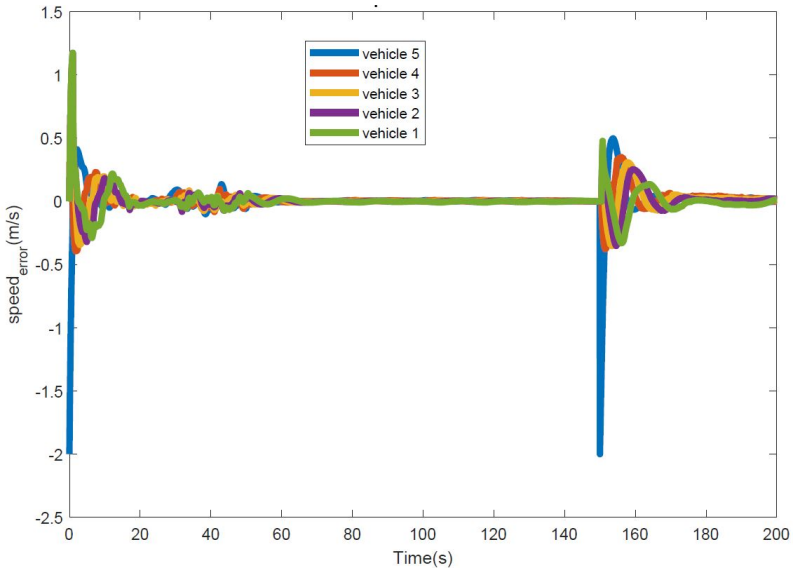


Figure 5.5: Error in the linear speed of each vehicle w.r.t. its varying reference.

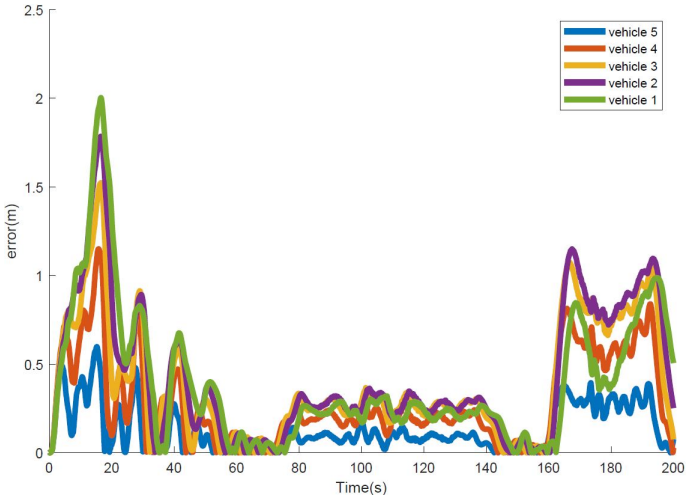


Figure 5.6: Cross-track error of each vehicle w.r.t. the reference track of the lead vehicle.

Chapter 6

Conclusion

As part of this work, a model-free event-triggered trajectory tracking optimal controller was developed. Results of an experimental vehicle platform using the controller to track a set of GPS coordinates defining a race track in simulations were shown in Chapter 2, which show the feasibility of the proposed trajectory tracking control technique.

In order to experimentally validate the simulation results, the already existing vehicle platform was studied and few major changes were made to it according to the requirements of this work. Details about the design and development (hardware and software) of the experimental vehicle platform were presented, as part of which various sensing and communication techniques to be used by the vehicle to sense its surroundings and its own states were proposed.

A technique called Real-time kinematic (RTK) positioning that enhances the precision of GNSS (GPS, GLONASS, Galileo, etc.) based positioning data was studied as part of this research. RTK GNSS modules were bought and used to implement this technique on the vehicle to get its accurate position data.

Experiments were performed to validate the single vehicle trajectory tracking results. Even though the experimental results obtained aren't as good as the simulation results, they show that the presented algorithm is applicable in practice and the results can be made better with further efforts (for example - using better sensors, choosing longer test track to avoid

discontinuous learning).

The developed optimal tracking controller was implemented on a group of vehicles (each similar to the experimental vehicle) in simulations to make them move in a platoon with constant inter-vehicle spacing while satisfying the string stability requirements in terms of position and speed errors of the vehicles w.r.t. their respective references, the results of which were shown in Chapter 5. A method for generating the reference trajectories for the following vehicles from the trajectories of their respective preceding vehicles was also presented in Chapter 5. It was shown that constant spacing policy can be made string stable by letting the lead vehicle transmit its speed information to all the other vehicles in the platoon.

Improving the experimental results for the single vehicle trajectory tracking and implementing the constant spacing platooning in experiments would be the next steps of this research. Future research efforts can also focus on making the optimal controller work for non-linear systems so that the tracking problem need not be separated into tracking in the x-direction and that in the y-direction. Making the vehicle platoon string stable in terms of cross-track error would also be worked on as part of the future work.

Bibliography

[1] Battlekits - combat robot kits. URL http://battlekits.com.

[2] Emlid - reach m+ rtk gnss module. URL https://docs.emlid.com/reachm-plus/.

[3] Ros - nmea strings parser, . URL http://wiki.ros.org/nmea_navsat_driver.

[4] Nmea protocol, . URL https://gpsd.gitlab.io/gpsd/NMEA.html.

[5] Hassane Abouaïssa, Michel Fliess, Violina Iordanova, and Cédric Join. Freeway ramp metering control made easy and efficient. *arXiv preprint arXiv:1206.5937*, 2012.

[6] Sébastien Andary, Ahmed Chemori, Michel Benoit, and Jean Sallantin. A dual model-free control of underactuated mechanical systems, application to the inertia wheel inverted pendulum. In *2012 American Control Conference (ACC)*, pages 1029–1034. IEEE, 2012.

[7] DP Bertsekas and JN Tsitsiklis. Neuro-dynamic programming. athena scientific, belmont, ma, 1996. *There is no corresponding record for this reference.*

[8] Roger W Brockett and Daniel Liberzon. Quantized feedback stabilization of linear systems. *IEEE transactions on Automatic Control*, 45(7):1279–1289, 2000.

[9] Davide Calzolari, Bastian Schürmann, and Matthias Althoff. Comparison of trajectory tracking controllers for autonomous vehicles. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE, 2017.

[10] Jason Carbaugh, Datta N Godbole, and Raja Sengupta. Safety and capacity analysis

of automated and manual highway systems. *Transportation Research Part C: Emerging Technologies*, 6(1-2):69–99, 1998.

[11] Anton Cervin and Johan Eker. Feedback scheduling of control tasks. In *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*, volume 5, pages 4871–4876. IEEE, 2000.

[12] Riyad A El-laithy, Jidong Huang, and Michael Yeh. Study on the use of microsoft kinect for robotics applications. In *Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium*, pages 1280–1288. IEEE, 2012.

[13] Nicola Elia and Sanjoy K Mitter. Stabilization of linear systems with limited information. *IEEE transactions on Automatic Control*, 46(9):1384–1400, 2001.

[14] Ishmaal T Erekson. Modified trajectory shaping guidance for autonomous path following control of platooning ground vehicles. 2016.

[15] Michel Fliess. Model-free control and intelligent pid controllers: towards a possible trivialization of nonlinear control? *IFAC Proceedings Volumes*, 42(10):1531–1550, 2009.

[16] Michel Fliess and Cédric Join. Model-free control. *International Journal of Control*, 86 (12):2228–2252, 2013.

[17] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *Journal of guidance, control, and dynamics*, 25(1):116–129, 2002.

[18] Miaomiao Fu, Jun Ni, Xueyuan Li, and Jibin Hu. Path tracking for autonomous race car based on gg diagram. *International Journal of Automotive Technology*, 19(4):659–668, 2018.

[19] Pierre-Antoine Gédouin, Emmanuel Delaleau, Jean-Matthieu Bourgeot, Cédric Join, Shabnam Arbab Chirani, and Sylvain Calloch. Experimental comparison of classical pid and model-free control: position control of a shape memory alloy active spring. *Control Engineering Practice*, 19(5):433–441, 2011.

[20] WM Hassad, V Chellaboina, and Sergey G Nersesov. Impulsive and hybrid dynamical systems, 2006.

[21] Karol Hausman, Jörg Müller, Abishek Hariharan, Nora Ayanian, and Gaurav S Sukhatme. Cooperative control for target tracking with onboard sensing. In *Experimental robotics*, pages 879–892. Springer, 2016.

[22] J Karl Hedrick, Masayoshi Tomizuka, and P Varaiya. Control issues in automated highway systems. *IEEE Control Systems Magazine*, 14(6):21–32, 1994.

[23] WPMH Heemels, Karl Henrik Johansson, and Paulo Tabuada. An introduction to event-triggered and self-triggered control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3270–3285. IEEE, 2012.

[24] Petros Ioannou and Bariş Fidan. *Adaptive control tutorial*. SIAM, 2006.

[25] Gregor Klančar and Igor Škrjanc. Tracking-error model-based predictive control for mobile robots in real time. *Robotics and autonomous systems*, 55(6):460–469, 2007.

[26] Frank L Lewis, Draguna Vrabie, and Vassilis L Syrmos. *Optimal control*. John Wiley & Sons, 2012.

[27] Frank L Lewis, Draguna Vrabie, and Kyriakos G Vamvoudakis. Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *IEEE Control Systems Magazine*, 32(6):76–105, 2012.

[28] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 36 (5):628–647, 2015.

[29] B Madhevan and M Sreekumar. Tracking algorithm using leader follower approach for multi robots. *Procedia Engineering*, 64:1426–1435, 2013.

[30] Ivan Maurović, Mato Baotić, and Ivan Petrović. Explicit model predictive control for trajectory tracking with mobile robots. In *2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 712–717. IEEE, 2011.

[31] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. John Wiley & Sons, 2007.

[32] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[33] David Schrank, Tim Lomax, and Bill Eisele. 2012 urban mobility report. *Texas Transportation Institute*, pages 1–57, 2011.

[34] David Schrank, Bill Eisele, and Tim Lomax. 2014 urban mobility report: powered by inrix traffic data. Technical report, 2015.

[35] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[36] D Swaroop and J Karl Hedrick. String stability of interconnected systems. *IEEE transactions on automatic control*, 41(3):349–357, 1996.

[37] DVAHG Swaroop and J Karl Hedrick. Constant spacing strategies for platooning in automated highway systems. *Journal of dynamic systems, measurement, and control*, 121(3):462–470, 1999.

[38] Kyriakos G Vamvoudakis and Henrique Ferraz. Model-free event-triggered control algorithm for continuous-time linear systems with optimal performance. *Automatica*, 87: 412–420, 2018.

[39] Kyriakos G Vamvoudakis, Arman Mojoodi, and Henrique Ferraz. Event-triggered optimal tracking control of nonlinear systems. *International Journal of Robust and Nonlinear Control*, 27(4):598–619, 2017.

[40] Jorge Villagra and Carlos Balaguer. A model-free approach for accurate joint motion control in humanoid locomotion. *International Journal of Humanoid Robotics*, 8(01): 27–46, 2011.

[41] Jorge Villagra and David Herrero-Pérez. A comparison of control techniques for robust docking maneuvers of an agv. *IEEE Transactions on Control Systems Technology*, 20 (4):1116–1123, 2011.

[42] Draguna Vrabie, Kyriakos G Vamvoudakis, and Frank L Lewis. *Optimal adaptive control and differential games by reinforcement learning principles*, volume 2. IET, 2013.

[43] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[44] Wikipedia contributors. Robot operating system — Wikipedia, the free encyclopedia, 2019. URL https://en.wikipedia.org/w/index.php?title=Robot_Operating_System&oldid=901201487. [Online; accessed 17-June-2019].

[45] Wei Yan. A two-year survey on security challenges in automotive threat landscape. In *2015 International Conference on Connected Vehicles and Expo (ICCVE)*, pages 185–189. IEEE, 2015.

[46] Diana Yanakiev and Ioannis Kanellakopoulos. A simplified framework for string stability analysis in ahs. In *Proceedings of the 13th IFAC World Congress*, volume 182, pages 177–182, 1996.

Appendices

Appendix A

Vehicle Hardware

The vehicle studied and used for experimental validation was purchased from Battlekits [1]. It is capable of moving at speeds of up to 10m/s which satisfies the requirement of the experimental vehicle platform being analogous to the standard highway vehicles. Also, the vehicle is collision tolerant which facilitates the experimental validation. Using this kind of scaled vehicle rather than a full-scale one will be cost effective keeping in mind the collisions that might happen while testing.

This appendix talks about the hardware components of the vehicle.

A.1 Electronics Platform

The electronics platform that encloses the batteries and motors was designed by Daniel Dunn. The platform provides an area on the top to mount the electronics system, and to place the GPS module and the lidar units. On the underside of the platform, power distribution block to the motor electrical system is mounted.

A.2 Vehicle Shell

In order to protect the electronics on the vehicle from sunlight, a shell was designed for it by Jackson Reid. An aerodynamic vehicle shell, and also a bumper structure in the front

and back of the vehicle to prevent the shell from breaking when the vehicle collides with its surroundings were designed by him as shown in Figure A.1. The shell was buckled using

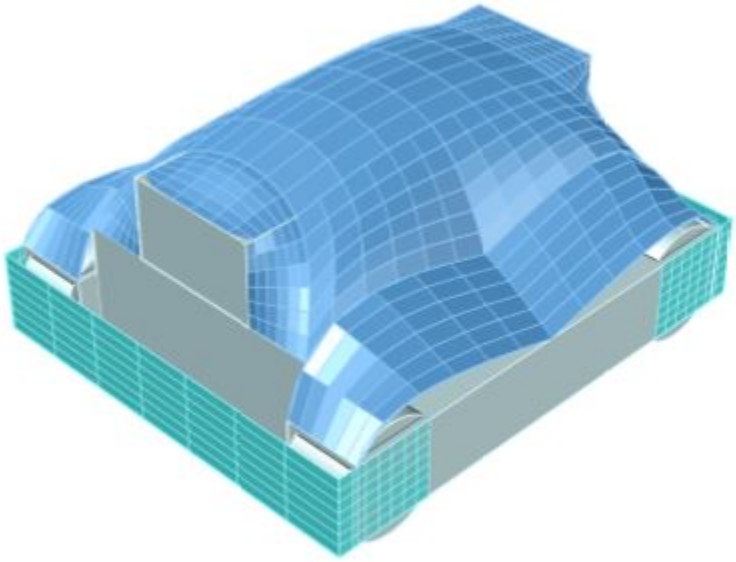


Figure A.1: Vehicle shell and bumper system designed by Jackson Reid

side-release buckles on one side and was hinged on the other side to the vehicle. This allowed for fast access to the electronics platform under the shell while securing the shell well enough from falling off the vehicle.

A.3 UP-board

Initially, the vehicle had a Beaglebone Black with 1GHz ARM Cortex-A8 processor as its onboard computer because of its low-cost and huge community support. The onboard computer was recently changed to UP-board which has a 1.92GHz Intel ATOM x5-Z8350 processor

with Ubuntu distribution of Linux. It has an extensive number of peripherals and is just the size of a credit card. It is equipped with 4x USB 2.0 and 1x USB 3.0 OTG ports as compared to 1x USB 2.0 port on Beaglebone Black. A USB Hub was needed with Beaglebone Black but that wasn't needed in the case of UP-board owing to its enough number of USB ports.

A.4 Tiva C Microcontroller

The Tiva C EK-TM4C123GXL Microcontroller acts as the sensor hub which interfaces with the sensors - quadrature encoders, lidar-lite modules and sends measurement data to the UP-board through UART. It is from Texas Instruments and has an ARM Cortex-M4 processor. The peripheral devices are configured using the peripheral driver library for ease of use.

A.5 USB Wireless Adapter

The vehicle is connected to the Wi-Fi router using a 802.11b/g USB wireless adapter to send and receive information from the command station which is also connected to the router. The adapter was configured such that the onboard computer connects to the Wi-fi access point through it as soon as the board is switched on.

A.6 Battery System

The motor electrical system powers up using a 36V supply generated by connecting three 12V batteries in series. The chargers in the lab can charge these batteries at up to 2A currents and usually takes around 20 minutes to completely charge the 36V supply. The electronics power system uses a 7.4V battery supply to power up.

A.7 Quadrature Encoders

Two quadrature encoders, one for each motor, are present in the vehicle which can determine the wheel velocities. These wheel velocities can in-turn be used to calculate the linear speed and angular speed of the vehicle. The quadrature encoders are interfaced to the sensor hub (Tiva C) using the quadrature encoder input (QEI) modules. The encoders are pretty accurate at determining the wheel velocities but not the vehicle position because of the possible slip in the wheels when the vehicle moves.

A.8 Lidar-LITE Module

A Lidar-LITE module on the vehicle provides distance to target which in the platooning case is the preceding vehicle. It is different from a true Lidar module in its means of operation. It sends a pulse of light and detects the reflected pulse with reference to shift in pixels, while in a conventional Lidar module, light in the form of a pulsed laser is used. This makes the Lidar-LITE module cheap compared to the Lidar module. It is interfaced to the sensor hub (Tiva C) using the I2C communication protocol. Numerical differentiation of the relative distance data from the Lidar-LITE module provides relative speed information which is accurate.

A.9 GNSS Units

Real-time kinematic (RTK) positioning technique enhances the precision of GNSS (GPS, GLONASS, Galileo etc.) based positioning data. Emlid Reach M+ RTK GNSS modules are used to implement this technique to get accurate position data of the vehicle. One unit

can be used as a base with one unit on the vehicle as a rover. More details about how RTK positioning technique works can be found in the below subsection. The GNSS units can also provide speed data, and operate at a minimum/maximum frequency of 1Hz/14Hz. More details on how to use these units can be found at [2].

A.9.1 RTK Positioning

The common GNSS positioning technique is referred to as code-based positioning in which the receiver utilizes codes received from four or more satellites to calculate ranges to them. Then knowing the locations of the satellites and the ranges from them, it determines its position with an accuracy of a few meters.

Real-time kinematic (RTK) positioning technique uses carrier-based ranging to determine the position of the receiver that is orders of magnitude more accurate when compared to that estimated through code-based positioning. The RTK technique is quite complicated but the basic idea is to mitigate or get rid of errors common to a base and rover receiver pair. This technique can provide accuracies of up to 1 cm i.e., centimeter-level positioning. The accuracy in position attainable by the rover is dependent mainly on its distance from the base (commonly called as baseline) and precision of the corrections sent by the base. The corrections in-turn depend on the accuracy of the base's location and the quality in satellite observations of the base station. To minimize environmental effects like multipath and interference, location selection is important and so is the quality of base and rover antennas and receivers.