VISION BASED CONTROL OF A MOBILE ROBOT FOR DRIVING APPLICATIONS

By

Gregory S. Grebe

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: COMPUTER AND SYSTEMS ENGINEERING

Examining Committee:

_____

John Wen, Thesis Adviser

_____

Agung Julius, Member

_____

Brett Fajen, Member

Rensselaer Polytechnic Institute
Troy, New York

March 2017
(For Graduation May 2017)

CONTENTS

LIST OF FIGURES iv

ABSTRACT v

1. Introduction 1
   - 1.1 Motivation 1
   - 1.2 Problem Statement 2
   - 1.3 Summary of Results 2
   - 1.4 Organization of Thesis 3

2. Literature Review 4
   - 2.1 Autonomous Vehicles 4
   - 2.2 Vision-Based Robotic Control 5
   - 2.3 Human Visual Perception for Driving 8

3. Controlled Braking 13
   - 3.1 Tau Based Controller 13
   - 3.2 Algorithm Implementation 17
   - 3.3 Simulation Results 19

4. Vehicle Following 24
   - 4.1 The Duckiebot 24
     - 4.1.1 Robot Kinematics 25
     - 4.1.2 Camera Model 28
     - 4.1.3 System Software Architecture 29
     - 4.1.4 Drive Service 30
     - 4.1.5 Camera Service 31
   - 4.2 Lateral Controller 32
   - 4.3 Visual Angle Based Controller 33
   - 4.4 Software Simulations 38
     - 4.4.1 Algorithm Implementation 38
     - 4.4.2 Simulation Results 40
   - 4.5 Robot Implementation 43

4.5.1 Vehicle Detection Implementation 43
4.5.2 Vehicle Following Implementation 47
4.5.3 Experimental Results 48

5. Conclusion and Future Work 54
5.1 Summary and Comparison of Approaches 54
5.2 Directions for Future Work 55
5.2.1 Alternative Tau Measurement Methods 55
5.2.2 Relative State Observers 56
5.2.3 Alternative Lateral Guidance 56
5.2.4 Alternative Robot Hardware 57
5.2.5 Other Robotic Applications 57

REFERENCES 58

APPENDICES

A. DuckieRR – Duckiebot on Robot Raconteur 62
A.1 Installation 62
A.2 Running Services 62
A.3 Running Clients 63
A.4 Writing New Clients 63
A.5 Example: DriveByAngle Client 64

LIST OF FIGURES

2.1 Time To Contact Geometry 9

3.1 Controlled Braking - Ideal Case 19

3.2 Controlled Braking - Baseline Test 20

3.3 Controlled Braking - Effect of Closed Loop Rate 22

3.4 Controlled Braking - Safer Approach 23

4.1 The Duckiebot 25

4.2 Kinematics Model for the Duckiebot 26

4.3 Relationship Between Robot and Camera Frames 28

4.4 Block Diagram of Robot System Architecture 31

4.5 Vehicle Following Dynamics Root Locus 37

4.6 Nonlinear Simulation Step Responses 37

4.7 Vehicle Kinematics In Polar Coordinates for Matching a Pose 39

4.8 Discrete Time Simulation - Step Response with Variable Damping Ratios 41

4.9 Discrete Time Simulation - Trajectory Tracking 42

4.10 Discrete Time Simulation - Sudden Deceleration 42

4.11 Vehicle Tag Detection 44

4.12 Camera Calibration Setup 46

4.13 General Experimental Setup for Duckiebot Following 50

4.14 Comparisons of the Commanded Velocity Profiles Between Leader and Autonomous Follower 51

4.15 Autonomous Duckiebot Performing a Lane Switch 52

ABSTRACT

Autonomous navigation of a shared roadway is a challenging task that requires an advanced model of the environment. Many autonomous vehicles currently achieve this using an expensive sensor suite based primarily on radar or LiDAR for navigation and control, while relegating vision to more contextual tasks. Nevertheless, vision can be extremely powerful and, as evidenced by human visual perception, provides useful information for the control of a vehicle. In this research, we examine the task of controlling vehicle headway using purely vision based controllers. The main idea is to design robot or vehicle controllers that mimic human visual perception models for the same task. Towards this end, we propose two controllers based on human visual perception models used in vehicle braking and vehicle following scenarios.

The first controller is based on the long studied time-to-contact visual variable, τ ("tau"), as it relates to controlled braking for collision avoidance. In this implementation τ is measured directly from the image features and their derivatives. The control scheme is then based on regulating $\dot{\tau}$ ("tau-dot"), which can be shown to have some very simple relationships to desired deceleration profiles. Ultimately, however, it is shown in simulation that the computations of τ and $\dot{\tau}$ using this method leads to large measurement errors, to the detriment of the overall controller.

The second controller is based on using the visual angle of a lead vehicle to control and keep a constant following distance. The model is shown to be more numerically stable than a tau based approach, and can be easily tuned to match a desired second order system response. This algorithm is evaluated in both simulation and on a physical robot hardware implementation. Several driving scenarios are presented to showcase the algorithm's effectiveness, including videos of the results.

1. Introduction

1.1 Motivation

Driverless vehicles are poised to dramatically change the way humans get from place to place, making roadway travel more convenient, safe, and efficient. Recent advances in autonomous vehicles by major commercial companies including Waymo (formerly the Google self-driving car project) [1], Tesla [2], and even the taxi service Uber in partnership with Volvo [3], have already made driverless car technology directly available to the masses. As a result, nearly every major automobile manufacturer is now working to have their own driverless car technology ready for market by 2020 [4].

To successfully navigate in a complex and dynamic environment, autonomous vehicles and robots typically require a good model of their surroundings. As such they rely on an advanced suite of sensors, often including some combination of radar, laser range-finding (LiDAR), ultrasound, infrared, and video [5]. With a few exceptions, computer vision is rarely used alone to directly determine vehicle control inputs. It is most commonly fused with one of these other sensors for general environment context understanding (i.e. identifying vehicles from bicyclists).

These sensors suites are certainly more robust than vision processing alone, but they come at a high cost, both financially and computationally. The cost of simple camera sensors on the other hand continues to decrease, and recent advances in computational power are making advanced vision algorithms more feasible. By examining human visual perception as it relates to driving, we know that vision can offer much more information to a vehicle controller beyond simple context clues. In fact, humans are able to accomplish driving tasks using vision alone. While it is true that there is still a large gap between human and machine vision capabilities as it relates to driving [6], in principle, any driving task that can be solved with human vision should eventually be solvable with machine vision as well. Similarly, there are many more ways we can incorporate the information provided by vision sensors directly into our vehicle controllers.

1.2 Problem Statement

Current autonomous vehicle algorithms rely on expensive laser or sonar sensors, particularly to achieve longitudinal, or headway, control of the vehicle. This thesis therefore attempts to address the following questions:

(i) *Can a purely vision based approach be used to either supplement or replace more expensive sensors for the task of vehicle headway control?*

(ii) *What kind of visual models are used in human visual perception for this driving task, and can they similarly be applied to autonomous vehicles or other robots?*

We examine this in the context of two specific driving scenarios: controlled braking for obstacle avoidance, and maintaining a constant headway behind a lead vehicle (vehicle following).

1.3 Summary of Results

In Chapter 2 we will introduce two visual models well known in the field of human visual perception to be used in driving scenarios: the *tau* (τ) model for controlled braking, and the *drive by visual angle* (DVA) model for headway keeping. Both of these models are based on simple image features that can generally be extracted in a variety of ways, some more robustly than others. We re-formulate the human visual perception models into a purely image based controller for use on an autonomous vehicle or robot.

This work makes use of a simple and inexpensive robot testbed for vision based driving and navigation tasks, the *Duckiebot*. As part of this work, the existing software architecture for the robot was updated and expanded to enable faster and simpler prototyping of these vision algorithms. The DVA based controller was then subsequently evaluated on the robot.

Both controllers are shown to able to accomplish their desired task of either bringing the vehicle to a safe stop, or following at a specified distance. However, the tau based controller is severely affected by measurement errors and the response is ultimately far from the original model. The DVA based controller, on the other hand, proves to be both flexible and effective for controlling headway.

1.4 Organization of Thesis

The remainder of this thesis is organized as follows: Chapter 2 briefly introduces the existing work in this broad research field. Chapters 3 and 4 then cover all of the work related to the controlled braking and vehicle following controllers respectively, including the theory, implementation details, and results. Chapter 5 concludes our discussion by comparing the two control schemes, highlighting our contributions, and discussing possible directions for future work.

2. Literature Review

The main research topics of this thesis fall into three broad categories: *Autonomous Vehicles*, *Vision-Based Robotic Control*, and *Human Visual Perception for Driving*. In this chapter, these topics will be briefly introduced through a survey of existing literature, with some background knowledge presented where appropriate.

2.1 Autonomous Vehicles

As alluded to in the introduction, autonomous passenger vehicles are an extremely fast growing industry now primarily driven by the consumer market. Besides additional convenience, driverless cars have the potential to increase road safety, dramatically reduce driver error related accidents, and even reduce fuel consumption by improving roadway efficiency. As most technological advances go it has a relatively rich history in research and academia. The authors in [5] give a fairly detailed history of the autonomous passenger car and its advancements over the past 50 years, with research in the US beginning in the late 1940s.

More recently, DARPA sponsored the Grand Challenge in 2005 [7] and subsequent Urban Challenge in 2007 [8]. Both aimed to test the limits of autonomous vehicle technology in either an unstructured all terrain environment or an urban setting. These challenges had mixed success, demonstrating some major advances in the field, and highlighting some of the state-of-the-art's current shortcomings. Nevertheless, the challenges are widely viewed as being a major contributor for renewed research efforts towards fully autonomous vehicles [5, 9]. In the years since, major advancements have been made by companies like Google, whose autonomous vehicles have now self-driven over 2 million miles [1], bringing autonomous vehicles to a point where they are ready for consumer use.

As previously mentioned, these cars rely on an advanced array of sensors, but at the heart of most current implementations is typically a LiDAR system, which is used to produce a very detailed and accurate 3D map of the environment around the vehicle [10]. These sensors are typically very expensive, and require a

lot of computation power to process the detailed depth maps produced. Although computer vision is not typically used as the primary sensor, is still plays an important role. For example, computer vision is currently the most robust and precise method to detect and track lane markings [5], and the problem has been extensively studied by multiple groups. Most apply a dynamic observer to estimate and track the lane curvature and relative vehicle pose [11–14]; others propose some other Bayesian filtering approach [15, 16]. Vision is also often fused with other sensors and used for context understanding, for example fusing radar or laser data with vision data to differentiate between pedestrians, cyclists, and other vehicles. It is not however typically used for things like headway keeping or the longitudinal control of the vehicle, as Radar or LiDAR would give a more robust and accurate measurement. Nevertheless, this thesis assumes that such sensors are not available, and attempts to accomplish similar tasks using vision alone.

2.2 Vision-Based Robotic Control

Vision-based robotic control has a relatively long history in the field of robotics. Recent advancements in computational power have made real-time vision algorithms feasible, and vision is increasingly being incorporated into feedback control. Vision-based robotic control, or *visual servoing* as it is often called, has been extensively researched in the past few decades [17,18], with many robotics textbooks and tutorials now covering the topic in detail [19,20]. Visual servoing is commonly applied to manipulators in which a rigid relationship between the camera, the robot joints, and the task space can be defined. The problem has also been studied for mobile robots and vehicles [21–23], however it is well understood that for a nonholonomic mobile robot the general visual servoing problem is non-trivial since no linear controller exists to control the robot motion [24].

Several distinctions can be made between different types of visual servoing problems. The first deals with the number of cameras in the system. Several approaches use a two camera, or *stereo*, approach to simplify some of the computer vision analysis. However, this thesis will assume a single camera approach.

The problem can also be differentiated by the placement of the camera in the

system. When the camera is attached to the robot, typically on a mobile base or a manipulator end effector, it is referred to as "eye-in-hand." Alternatively the camera location can be fixed, with a view to observe the robot's task environment. This is sometimes called "eye-to-hand". We focus only on the "eye-in-hand" configuration.

The problem is traditionally split into two fundamentally different approaches: Position-Based Visual Servo (PBVS), and Image-Based Visual Servo (IBVS) [17]. The approaches differ in the underlying parameter space that is being controlled. In PBVS image features are extracted, and then related back to a pose in the task space using some knowledge of the system geometry. The control task can then be accomplished directly in the task space. In IBVS, the image feature space is directly controlled, without any transformation to the task space. The latter method is advantageous in that it is robust to both camera calibration and robot calibration errors, and directly using the image features for control may reduce the computation time. However, the controller design is significantly more challenging since the relationships between the camera motion and the feature motion are highly nonlinear and coupled. In this work we examine only Image-Based Visual Servoing techniques.

The IBVS problem can be formulated in a general way [19], in which changes in image features are mapped to the camera's motion through an "interaction matrix", more commonly called the "image Jacobian"

$$\dot{\boldsymbol{p}} = \boldsymbol{J}_p \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix}, \tag{2.1}$$

where $\boldsymbol{p}$ gives some image feature (points, lines, planes, ellipses, etc. . .), $\boldsymbol{J}_p$ is the image Jacobian, and $\boldsymbol{v}$ and $\boldsymbol{\omega}$ are the camera linear and angular velocities respectively. This method can then be extended to use multiple features by simply stacking the image Jacobians for each individual feature, i.e.

$$\begin{bmatrix} \dot{\boldsymbol{p}}_1 \\ \dot{\boldsymbol{p}}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_{p1} \\ \boldsymbol{J}_{p2} \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix}. \tag{2.2}$$

Typical robotics applications are more concerned with the inverse relationship, namely *what camera velocities are required to move the image features to a desired location*. A simple pseudo-inverse control law can be applied

$$\begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} = \lambda \boldsymbol{J}_p^+ (\boldsymbol{p}_d - \boldsymbol{p}), \tag{2.3}$$

however, care must be taken in choosing an appropriate number and configuration of feature points to ensure that the Jacobian is full rank. In general this is not always possible and singularity configurations for the robot or the image features may exist.

Furthermore, the image Jacobian is still a function of the z coordinate, or depth, of each point. The IBVS formulation is usually tolerant to errors in z, but works best when z can either be assumed constant, or can be dynamically estimated online, for example by using a nonlinear estimator [25]. Because this formulation still requires knowledge of the depth, it is not entirely useful for longitudinal control in which the depth is exactly what we wish to control. The authors in [18] show that this formulation is however useful for lateral control, particularly the lane tracking problem since lane features are invariant to motion along the optical axis.

The research in [22] also tackles the problem of unknown depth and image Jacobian singularities, specifically for applications to visual servoing of a nonholonomic mobile robot. They adopt a two step approach. In the first, epipolar geometry is used, rather than an image Jacobian, to compute a relative orientation between two scenes and drive the robot such that any orientation offset is removed. Ultimately the epipoles cannot be used to remove the longitudinal error, since for motion along the optical axis the epipoles are identically zero. The second step then computes any required translation along the optical axis directly using the feature points.

In [23], the same authors examine a leader-follower formation control scenario, in which only passive vision sensors are used. In their approach, a panoramic camera is used to provide a view angle to the lead robot and compute a relative heading. The distance however, still can not be computed and is instead estimated online by a nonlinear observer (Extended Kalman Filter). Furthermore, the leader and follower

may communicate with one another to coordinate formation velocities. While the use of a nonlinear observer could certainly improve any estimate of the headway, this work will not examine any estimators, nor will we assume communication between the two vehicles.

2.3 Human Visual Perception for Driving

Largely unrelated to robotics or control theory, there has been a significant interest in the neuroscience and cognitive science fields to model human visual perception, particularly related to driving tasks. Work by Lee in the late 1970s ([26] and [27]) directly examined the use of the optic flow field in human visual perception and directly related it to driving scenarios, including lane keeping and braking. Of particular interest is his introduction of the visual variable τ ("tau"), which relates the visual expansion of the image of an object to the time to collision/contact with that object. The derivation of τ (from [28]) is now briefly presented.

Consider a visual object at a distance $Z(t)$ from the observer along the optical axis, and with size S. Modeling the eye as a simple pinhole, the object's projection will form the visual angle $\theta(t)$ (see Figure 2.1). Applying the small angle approximation, we have

$$\theta(t) = \frac{S}{Z(t)} \tag{2.4}$$

We are interested in how the optical feature changes with changes in our relative motion. By differentiating, and then using Equation 2.4 to remove S from the result we obtain the following relationships (with the dependence on the time variable, t, assumed and suppressed from the notation) :

$$\text{TTC} = \frac{Z}{-\dot{Z}} = \frac{\theta}{\dot{\theta}} = \tau \tag{2.5}$$

We notice that the units for $Z/\dot{Z}$ is simply a time. More specifically, this term gives the time-to-contact (TTC), and is exactly equivalent to the visual variable we called τ. This suggests that visual perception alone can provide the information required to regulate headway, without a need for measuring or estimating distances and velocities.

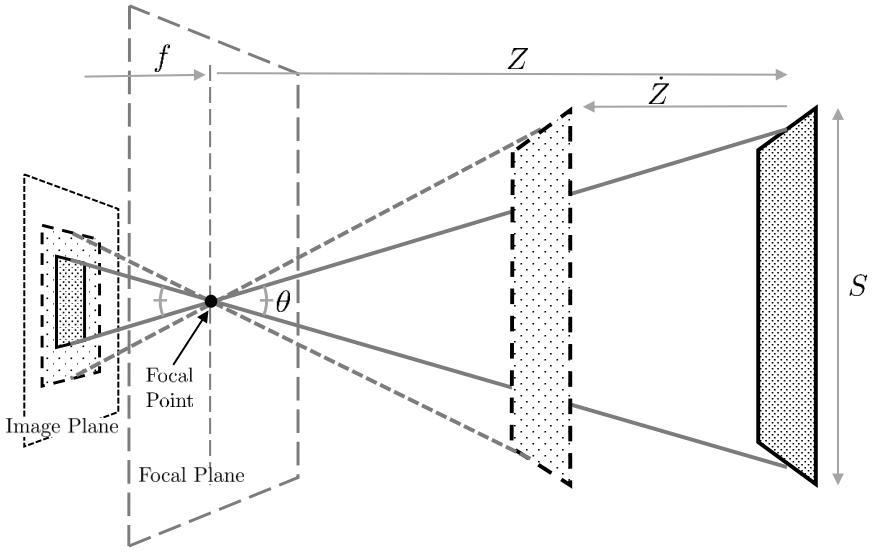


Figure 2.1: Time To Contact Geometry (Adapted from [29])

In [26], Lee specifically examines the use of the tau variable in vehicle braking situations. He notes that if a driver is to avoid collision, they must decelerate at a rate that will bring them to a stop within the distance separating them from the obstacle. Therefore, using standard kinematic relationships we have the following requirements

$$\frac{\dot{Z}^2}{2\ddot{Z}} \leq Z \quad \text{or} \quad \frac{Z\ddot{Z}}{\dot{Z}^2} \leq 0.5 \tag{2.6}$$

We can specify this relationship in terms of τ, or rather $\dot{\tau}$ ("tau-dot"), by noting that $Z\ddot{Z}/\dot{Z}^2 = 1 + \dot{\tau}$. Therefore, the drivers rate of deceleration is sufficient if and only if

$$\dot{\tau} \geq -0.5. \tag{2.7}$$

$\dot{\tau} = -0.5$ is a particularly efficient approach, as it specifies a constant deceleration profile, which minimizes jerk and therefore changes in the braking force. $\dot{\tau} > -0.5$ would result in a larger initial deceleration which then gradually decreases, leading to a more prolonged approach. Any values below -0.5, however, would require an exponentially increasing deceleration, which near the end of the approach would be

physically impossible to produce and still avoid a collision. $\dot{\tau} = -0.5$ is therefore the critical boundary point between a safe stop and a collision [28].

Lee contends that human drivers have an internal sense of a safe margin value, $\dot{\tau}_m$, required to avoid collision, and their deceleration profile will therefore match one for which $\dot{\tau}$ is held constant at this value. His trials with human drivers he believed supported this 'constant $\dot{\tau}$ strategy' hypothesis, however subsequent researchers tend to disagree. The authors in [28], for example, note that a constant $\dot{\tau}$ profile could be obtained without any visual feedback at all, simply by maintaining constant pressure on the brake. Furthermore, such a strategy completely ignores the dynamics of both the controller (the driver's perceptual-motor dynamics) and the plant (the vehicle). Given the finite precision and limitations in perceptual thresholds and perceptual-motor delays it is unlikely that humans are even capable of maintaining a constant value of $\dot{\tau}$ [30]. A more appropriate model would treat $\dot{\tau}_m$ as a value to be tracked in a classic feedback control structure, where the driver constantly makes braking adjustments based on whether $\dot{\tau}$ strays too far from the desired value. It is still unclear how the driver would then use $\dot{\tau}$ to determine how much braking to apply. The author in [30] notes that although $\dot{\tau}$ informs the driver whether to increase or decrease deceleration, it does not inform by how much. It is possible to have different situations in which the $\dot{\tau}$ error is the same, but the required change in deceleration to null this error depends on the current speed and TTC. This suggests that humans likely use more than just $\dot{\tau}$ in any braking task.

Since Lee's initial work, several researchers have tested the hypothesis that humans are even sensitive to τ and $\dot{\tau}$ as visual variables, and whether it can conclusively be shown that we use this information, including for tasks beyond vehicle braking. The general consensus has largely been that yes, we are sensitive to these variables, however, in many cases they provide a highly simplistic model as there are many other sources of visual information available that may either supplement or replace τ and $\dot{\tau}$ depending on the task. A study in human grasping and catching tasks by [31] determined that binocular vision does not provide any advantage over monocular vision for estimating TTC, suggesting that any sense of depth obtained from stereo imaging is likely not as important as the expansion rates. This further

validates τ's usefulness for a robotics application in which we would like to use only a single camera. Several authors have noted, however, that humans are also limited by our ability to detect small changes in expansion, therefore τ (and similarly $\dot{\tau}$) will generally only be useful for a finite time interval before contact [28]. A camera system attempting to measure τ would similarly be limited by factors like its resolution and spatial sampling, which would likely cause an even smaller time interval for which τ can be used.

Although tau and time-to-contact is certainly a useful model for regulating headway, it is by no means the only model that has been studied with respect to visual perception and driving. As the authors in [32] explain, Lee's method has some major shortcomings, especially for the task of following a lead vehicle. Namely, following at a fixed distance requires matching the lead vehicle's velocity. Visually this can be accomplished by keeping $\dot{\theta}$ at zero, which in Lee's model corresponds to keeping τ at infinity. While mathematically valid, this isn't a very easy model to work with. The authors further postulate that pure velocity matching approaches are also insufficient for following, as they fail to account for the distance between the vehicles required to keep a safe headway. This becomes a problem, for example, if the lead vehicle were to suddenly brake. The authors therefore introduce the 'Drive by Visual Angle' (DVA) model for vehicle following:

$$\ddot{x} = k_1 \left(\frac{1}{\theta} - \frac{1}{\theta_d} \right) + k_2 \dot{\theta} \tag{2.8}$$

where $\ddot{x}$ gives the forward acceleration and θ_d gives the visual angle corresponding to a desired headway. Examining Equation 2.4, we see that the visual angle, θ, is inversely proportional to the distance between the vehicles. Since the size of the vehicle can be assumed constant, the reciprocal of the visual angle, $1/\theta$, can directly be used in place of estimating distance. Likewise, to minimize changes in velocity the rate of change of the visual angle, $\dot{\theta}$ can be used. The DVA model postulates that humans control acceleration based on a simple weighting of these two factors. This suggests that a robot could therefore control it's acceleration in a similar way.

It should be noted that these variables have not been strictly studied only from the perspective of human visual perception. They have also attracted the attention

of many researchers in the computer vision field who are similarly interested in extracting these variables from an image or sequence of images robustly. Extracting the visual angle is essentially just a feature tracking problem, and many different methods and features can and have been used [33]. Tau or time-to-contact, on the other hand has a more interesting history in computer vision. Many methods attempt to estimate TTC directly from the optical flow field, however extracting optical flow is itself a computationally expensive process and can be relatively imprecise. The research in [34] gives a 'direct method', without the need to estimate optical flow, based on directly using the derivatives of the whole image brightness. However, this method is limited by the relative motion between the vehicles and relies upon most of the image containing the object of interest. While robust feature detection is certainly important for any real world application, it is beyond the scope of this thesis.

As the tau based approach and the DVA model are both applicable to controlling headway for different driving tasks, both will be examined in this thesis to determine their usefulness in robotic vision applications. We reiterate that these models are by no means the de-facto standard for human visual perception in their respective tasks. In fact, as we have mentioned, there are typically many other visual variables that humans also use to accomplish these tasks. These models do however provide a good base visual model for evaluation.

3. Controlled Braking

In this chapter we describe the full implementation and results of the controlled braking strategy. In Section 3.1 we detail the proposed controller based on the previously presented tau model. We then present the details of our simulation in Section 3.2 before presenting the results in Section 3.3.

3.1 Tau Based Controller

The human visual perception literature tells us that controlled braking using visual feedback can be accomplished by estimating the time to contact (TTC), or τ, from the rate of expansion of an image feature, and then regulating $\dot{\tau}$. Several additional considerations are required in order to utilize this kind of strategy in a robotic application. While humans are able to efficiently internalize visual information like τ and $\dot{\tau}$ without a need to quantize it, in a robotics application we must use actual measured values and design a control law around it. Furthermore, tau was derived as a continuous time variable, and therefore additional considerations must be made to make it useful for a discrete time implementation. In general, our approach will be to control our acceleration profile such that $\dot{\tau}$ tracks a desired reference $\dot{\tau}_{\text{desired}}$.

The general dynamics for $\dot{\tau}$ are given by

$$\begin{aligned} \dot{\tau} &= (Z\ddot{Z}/\dot{Z}^2) - 1 \\ &= 1 - (\theta\ddot{\theta}/\dot{\theta}^2) \end{aligned} \tag{3.1}$$

We note that control based on $\dot{\tau}$ poses a particular challenge as we cannot actually measure $\dot{\tau}$ or τ for that matter, but must compute it from the visual variable θ and its derivatives. Due do the discrete nature of a digital camera, we likewise can't truly measure derivatives of θ. We are therefore forced to approximate the required derivatives numerically from sequential images. In general, this kind of numerical approximation amplifies noise in our reading, and multiple differentiations can lead

to an unstable measurement. Since computing $\dot{\tau}$ directly from image features would require differentiating twice, it is unlikely we would be able stabilize a controller designed to track $\dot{\tau}_{\text{desired}}$ using this measurement. Our control strategy therefore needs some other way of determining $\dot{\tau}$.

We note that the first half of Equation 3.1 can be written in terms of τ,

$$\dot{\tau} = (-\tau \ddot{Z}/\dot{Z}) - 1, \tag{3.2}$$

and recall that Z is the separating distance between the vehicles, so $\dot{Z} = -\dot{x}$ ...

$$\dot{\tau} = \frac{-\tau \ddot{x}}{\dot{x}} - 1. \tag{3.3}$$

We will still require at least a single numerical differentiation in order to calculate τ, but this should be more stable than higher order derivatives. That being said, we have already touched on the fact that τ has its own numerical instabilities, namely that if $\dot{\theta}$ is 0, τ is infinite. Our eyes are capable of perceiving very small and (seemingly) continuous changes in the visual angle of an object as we approach it, and we can therefore at least tell that we are in fact approaching or receding from the object, even if we cannot tell exactly how fast. In a digital image, our measurements are quantized to whole pixel values, and the camera parameters of resolution and spatial sampling largely dictate how small of a change we can detect. Furthermore, from the concept of perspective projection we know that as an object moves further away, the change in distance required for an image change of a single pixel nonlinearly increases. We therefore expect to see very large gaps in which, although we are still approaching the object, the TTC as given by the image measurements will be infinite. The proposed solution is therefore to implement a sample and hold strategy, in which we simply ignore all measurements of infinity and update τ only when the visual angle changes. We keep track of the actual time since the visual angle changed, to more accurately compute the derivative once it does change. Ultimately the use of a digital camera leads to very poor estimates of τ at further distances, however this is also true of humans so it is not necessarily a prohibiting factor. As we continue to approach the object, the measured value of τ should converge to

the actual time to contact. Furthermore, if somehow the velocities did happen to exactly match so that τ truly is infinite it is largely inconsequential to our braking controller since at that point we are not on a collision course.

One final problem we have not yet addressed with Equation 3.3 is the velocity term, $\dot{x}$. This suggests that we must have some way of knowing or estimating our velocity. There are several ways that this could be accomplished. If we are to keep with using only visual feedback, velocity can typically be estimated using the process known as visual odometry, in which the optical flow is used. While this is certainly an option, in many cases it is also fair to assume that we have tight control over our commanded velocity, as many drive systems already have a built in low level feedback control loop. In this work we simply assume tight control over our velocity, and Equation 3.3 then gives an expression for $\dot{\tau}$ in terms of known or measurable values. We can now simply apply a feed-forward controller and solve for the acceleration required to obtain $\dot{\tau}_{\text{desired}}$

$$\ddot{x} = -\frac{\dot{x}}{\tau}(\dot{\tau}_{\text{desired}} + 1). \tag{3.4}$$

Now that we have determined the general control scheme, we can augment it to make it more applicable to mobile robots and autonomous vehicles. First, it must be noted that our current braking strategy will result in the vehicle coming to a stop right at the moment of contact with the lead vehicle or obstacle. In other robotics tasks, for example grasping an object, this may be the desired outcome. However, for vehicles we would generally like a larger safety margin to stop in. Furthermore, in a camera based implementation, we are limited by the camera's field of view. If we were to come too close to the obstacle, at some point we will no longer be able to detect τ and would therefore require an open loop approach for the final deceleration.

To address this we simply measure with respect to a different reference frame, for example a point on the ground some fixed distance behind the obstacle. To compute τ the "size" of the feature would then be given by the distance to the horizon, or some similar fixed visual reference. Alternatively, we could continue directly measuring the expansion of the obstacle, but we must know a priori the

visual angle of the object at the moment of contact with the reference frame. (i.e. if we would like to stop 1 meter from the vehicle we need to know the visual angle we would measure when we are 1 meter from the object.) This obviously loses some generality, as it is unrealistic in a driving scenario to have prior knowledge of every obstacle we may encounter. It may be fair however to assume that certain groups of obstacles (i.e. all vehicles) would have similar, although not identical, visual profiles as we approach them and, provided we recognize a vehicle, we simply use a value that will always give us at least the desired margin. In our case, this approach is slightly easier to implement. We therefore apply a *teach-by-showing* approach in which the visual angle of the object at the desired distance, denoted θ_{desired}, is saved in memory before we attempt a controlled approach.

If we apply this desired safety margin we note that it is now possible to obtain values of $\tau \leq 0$ without an actual collision occurring. This would cause our commanded acceleration to be either infinity or positive, neither of which are desired if we would like to avoid collision. If we have passed our safety margin threshold we have no choice but to apply full deceleration and come to a stop.

Since this controller is purely for braking, we would generally require some rule for determining when this controller should be applied. One possible choice is to use a threshold on τ. Given our current velocity and some desired limit on the maximum rate of deceleration, we can determine the minimum possible TTC value for which our requirements will be satisfied,

$$\tau_{th} = \frac{V}{2D_{\max}} \tag{3.5}$$

where V is the current velocity and $D_{\max}$ is the maximum allowed deceleration. In a typical car scenario, $D_{\max}$ should be selected to ensure the comfort of any human passengers. Although most robots generally do not have any human comfort requirements, we may still wish to impose a limit on the deceleration for safety reasons or to avoid wear on the brakes or drivetrain. In this work, we simply assume the velocity is the max the vehicle can achieve and choose the desired deceleration response. This generally increases our τ_{th} so that we begin braking sooner.

3.2 Algorithm Implementation

The robot we are using for our implementation unfortunately does not have tight control over the velocity commands as it lacks any feedback on the wheel velocities. It also lacks any active braking so it is largely an inappropriate test bed for the algorithm. We are therefore only able to evaluate the controlled braking strategy in a simulation environment.

A simulation has been created in Matlab that models a robot with ideal kinematics and a pinhole camera approaching an obstacle. We have created the simulation on approximately the Duckiebot's scale with respect to the size of the obstacles and the magnitude of the velocity / acceleration commands. We have also placed some saturation limits on the kinematics, although the values used are not exactly that of the Duckiebot. In this simulation we assume that the motion is purely along the optical axis and we therefore only control velocity. The kinematics are also assumed to be perfect such that at each timestep of the controller the robot has moved the precise distance given by numerically integrating the velocity command. We use the pinhole camera model to determine the pixel locations of the obstacle in an image. We assume that the controller is able to perfectly measure these pixel locations, and do not implement any object detection algorithm in the simulation.

Since we implement a safety margin to give ourselves a larger area to stop in we do not need to worry as much if we are not able to keep $\dot{\tau}$ exactly at or above -0.5. We therefore generally set $\dot{\tau}_{\text{desired}} = -0.5$, but will also examine the response using other values. We also have not incorporated a more advanced speed controller for when the braking control is not active, and generally just assume we may continue at maximum velocity until braking is triggered. The full τ based breaking strategy is summarized algorithmically in Algorithm 3.1. This algorithm as presented would be run every loop iteration to determine if breaking action is necessary.

For reference Figure 3.1 shows the ideal case where the controller is able to exactly measure τ without any error. We see a linear τ profile for each case (Figure 3.1a), which is expected for a constant $\dot{\tau}$ approach. It should be noted that in reality τ goes to infinity once the vehicle stops but for these plots we have held the value at zero for a slightly nicer visual (you can see the effect of this on the $\dot{\tau} = -0.7$ plot

Algorithm 3.1 Tau Based Braking Control

procedure TauBasedBraking($\theta, \theta_{\text{prev}}, \theta_{\text{desired}}, \tau_{\text{prev}}$)
  $\hat{\dot{\theta}} \leftarrow (\theta - \theta_{\text{prev}})$
  if $\hat{\dot{\theta}} \neq 0$ **then** ▷ We have a new reading
    $\dot{\theta} \leftarrow \hat{\dot{\theta}}/(T \cdot i)$ ▷ i counts loops since update, T is the framerate
    $\tau \leftarrow (\theta_{\text{desired}} - \theta)/\dot{\theta}$
    $i \leftarrow 1$
  else
    $\tau \leftarrow \tau_{\text{prev}}$ ▷ Our reading did not change
    $i \leftarrow i + 1$
  end if

  if $\tau > \tau_{th}$ **then** ▷ No need to brake yet
    $\ddot{x} \leftarrow$ AccelerationController$(\cdots)$ ▷ run some other controller

  else ▷ We need to brake
    if $\tau \leq 0$ **then**
      $\ddot{x} \leftarrow \ddot{x}_{\min}$
    else
      $\ddot{x} \leftarrow (-\dot{x}/\tau)(\dot{\tau}_{\text{desired}} + 1)$
    end if
  end if
  $\dot{x} \leftarrow \dot{x} + \ddot{x}/T$
  $\tau_{\text{prev}} \leftarrow \tau$
end procedure

which changes quickly enough that it spikes towards $-\inf$ before the vehicle actually stops). The commanded acceleration (Figure 3.1b) and velocity (Figure 3.1c) profiles are also shown. We reiterate that these too are characteristic of the specified $\dot{\tau}$ value. Most notably, for $\dot{\tau} = -0.5$ we see that the acceleration (deceleration) profile is constant throughout the trial until the vehicle actually comes to a stop. We also see that $\dot{\tau} = -0.7$, which we have said should ultimately result in a very quick deceleration right at the end does exactly that. In fact, the requested deceleration saturates the available braking power almost immediately. This reiterates the fact that $\dot{\tau}_{\text{desired}}$ should be kept above -0.5 for a safe and controlled stop.

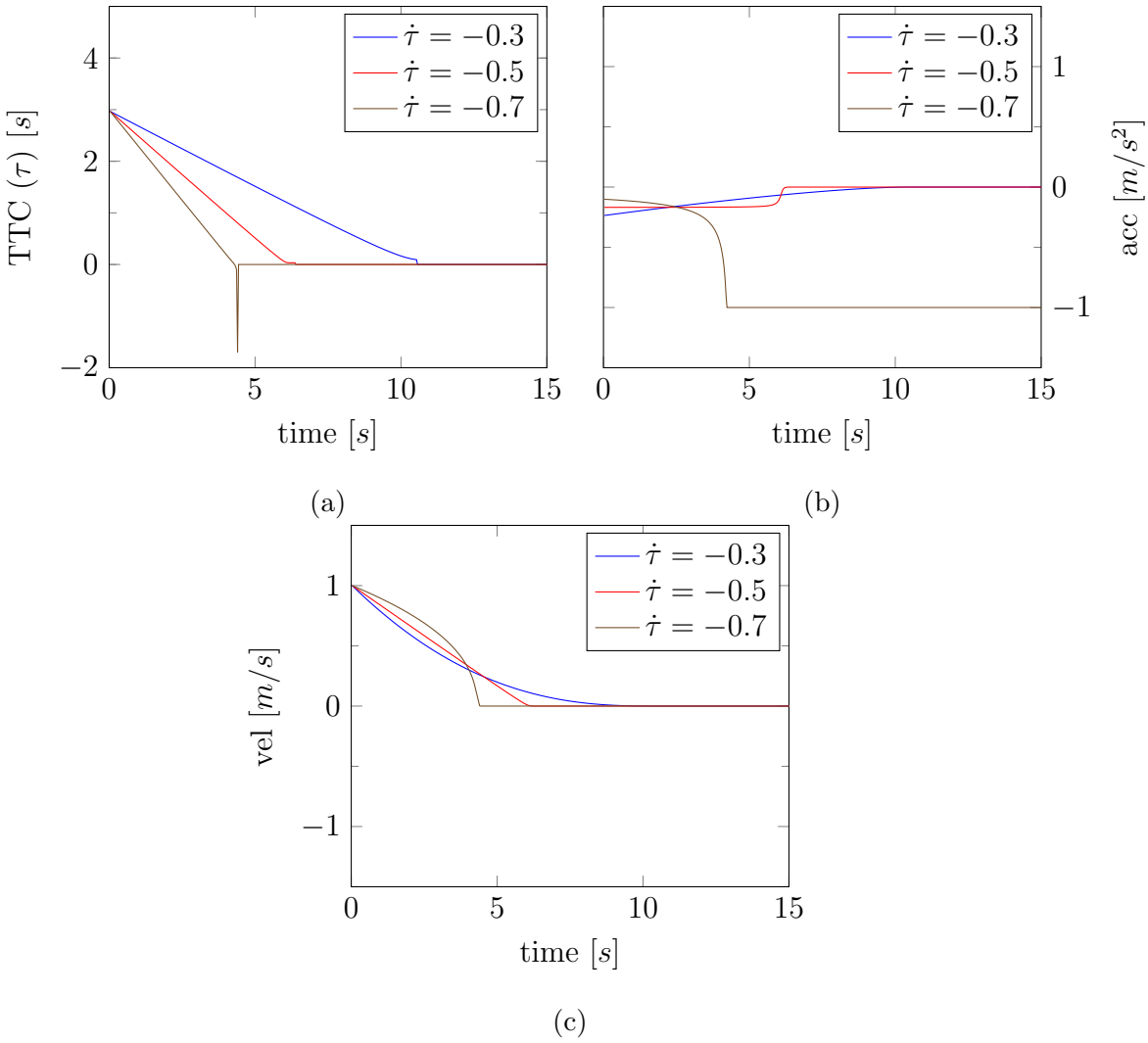


Figure 3.1: Controlled Braking - Ideal Case
(Fig 3.1a) Time to Contact Measurements. (Fig 3.1b) Commanded Deceleration Profile. (Fig 3.1c) Commanded Velocity profile.

3.3 Simulation Results

In general, our strategy is able to successfully bring the robot to a stop in a more controlled manner than simply applying full deceleration, however, due to inaccuracies in the measured τ value the response does not resemble the ideal case and we are generally not able to satisfy any deceleration constraints placed on the controller.

A baseline test for our full controller and $\dot{\tau}_{\text{desired}} = -0.5$ is shown in Figure 3.2. The robot starts at it's maximum velocity and the braking controller is immediately

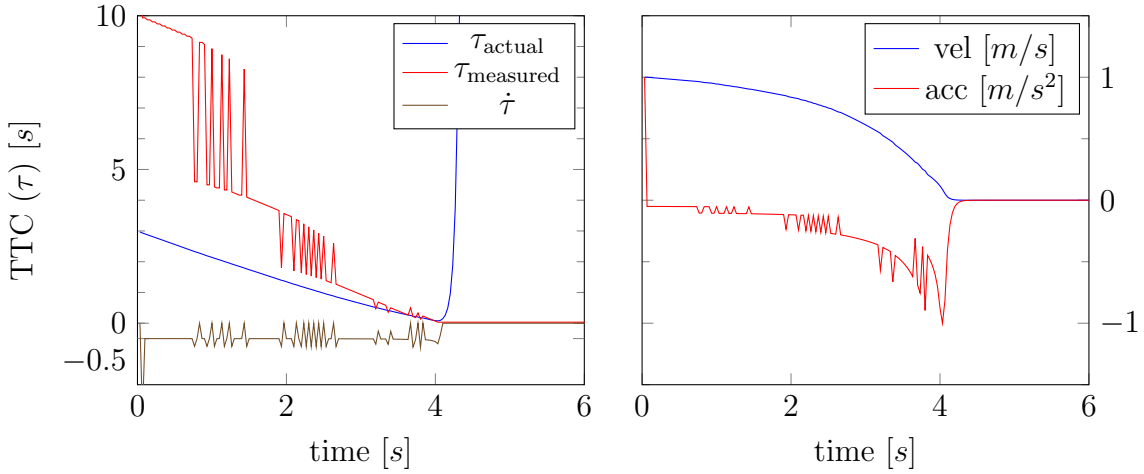


Figure 3.2: Controlled Braking - Baseline Test
(Fig 3.2a) Time to Contact Measurements. (Fig 3.2b) Kinematics Measurements.

applied. There are several interesting artifacts of our implementation immediately noticeable from these plots. First, as predicted, τ_{measured} is very inaccurate when we are further away. We have an initial error of approximately $7s$ between the measured value and the actual value provided by the simulation. Around the $1s$ mark, we first notice some of the instabilities caused by both numerically differentiating and the camera projection. The measured value seemingly bounces between two levels, but continues to generally decrease. These levels are a result of our camera's limited spatial sampling and the fact that we must measure changes in the image at whole pixel values. This leads to an almost quantization of the values τ can take, and contributes the most to our measurement error. Finally, with less than a second until collision, the reading settles and very closely approximates τ_{actual}. Once the vehicle stops we see that τ_{actual} goes to infinity as expected. Examining $\dot{\tau}$, the value generally stays right around -0.5 as desired, but is effected by the sudden changes in the τ reading. Similarly, while $\dot{\tau}$ is approximately stable our commanded deceleration profile remains approximately constant. However, the sudden drops in measured τ force us to decelerate faster to ensure we stop in time. The error in our measurement therefore causes a very large deviation in the response from the ideal case. Since our initial τ_{measured} is much larger than the actual time to contact, our

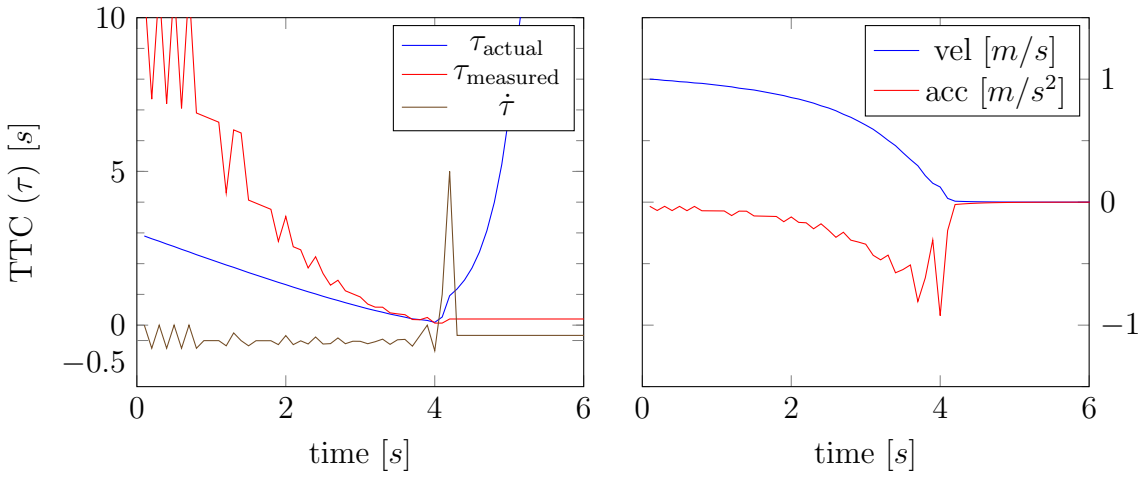
initial deceleration is generally not large enough and we are forced to compensate as we approach and τ_{measured} becomes more accurate. This is problematic as we therefore may not be able to satisfy any maximum allowed deceleration constraints and still avoid exceeding our safety margin, or in some cases even actual collision.

To emphasize that this error in τ_{measured} is really an artifact of the camera we also examine the effect of the closed loop rate of the controller. If this were not the case, we might hypothesize that since the measurement of τ is effected by the numerical differentiation, we might expect it to be effected by the sampling rate, and increasing the sampling rate might therefore give us a better estimate. We are practically limited by the framerate of a camera and the speed at which we can process images. Few practical imaging systems can process much higher than 60 frames per second (or Hz) so this is taken as our maximum. The baseline experiment used a closed loop rate of 30Hz. Figure 3.3 shows the same experiment repeated with the loop closed at 10Hz and 60Hz. Ultimately we see that the closed loop rate has little effect on the accuracy of τ, and we confirm that the camera quantization is truly the limiting factor. We see that τ_{measured} has approximately the same trend in both the 10Hz and 60Hz test as the baseline test at 30Hz. We therefore see very little change in the commanded acceleration between the three tests, and conclude that the closed loop rate could be fairly low and still obtain the same response.

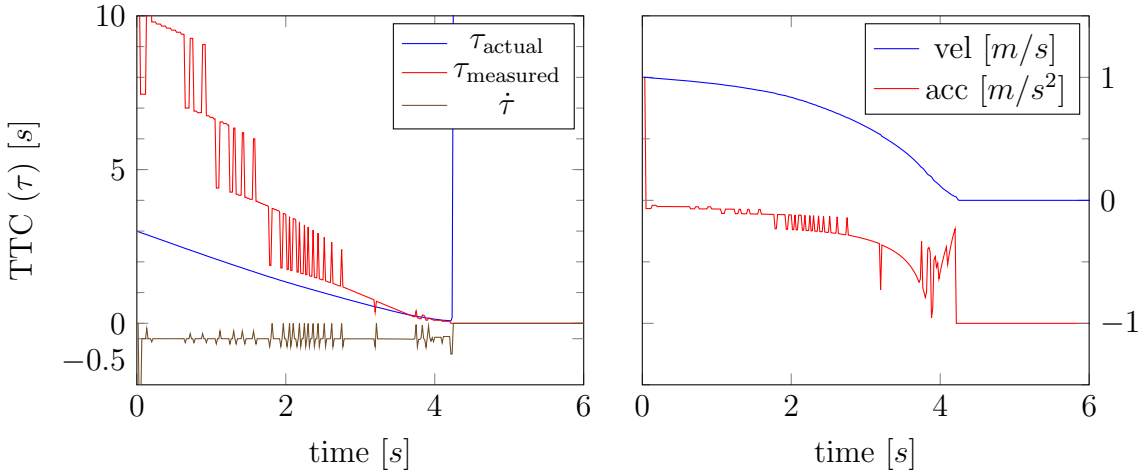
The response for $\dot{\tau}_{\text{desired}} = -0.3$ might be a little safer, but ultimately suffers from some of the same problems (Figure 3.4). We note specifically how drastically different this response is from the ideal case given in Figure 3.1.

We do not examine the case of applying our theoretically allowable τ_{th}, since we can already see that our response differs so much from the ideal in which this might be valid. We already know that our measured τ will differ significantly from the actual τ and therefore by the time our measured τ crosses τ_{th} it will already be too late for the algorithm to be effective. We may still be able to stop, but not with any reasonable expectation of a controlled deceleration.

Ultimately, we must conclude that, although the theory appears to be sound, the error in the measurement of τ severely limits the controller's effectiveness. It may be possible to correct for some of this measurement error by incorporating an



(a) 10Hz Closed Loop Rate



(b) 60Hz Closed Loop Rate

Figure 3.3: Controlled Braking - Effect of Closed Loop Rate

online nonlinear observer to estimate the distance to the obstacle; however, this was not examined further in this thesis and is left for future work.

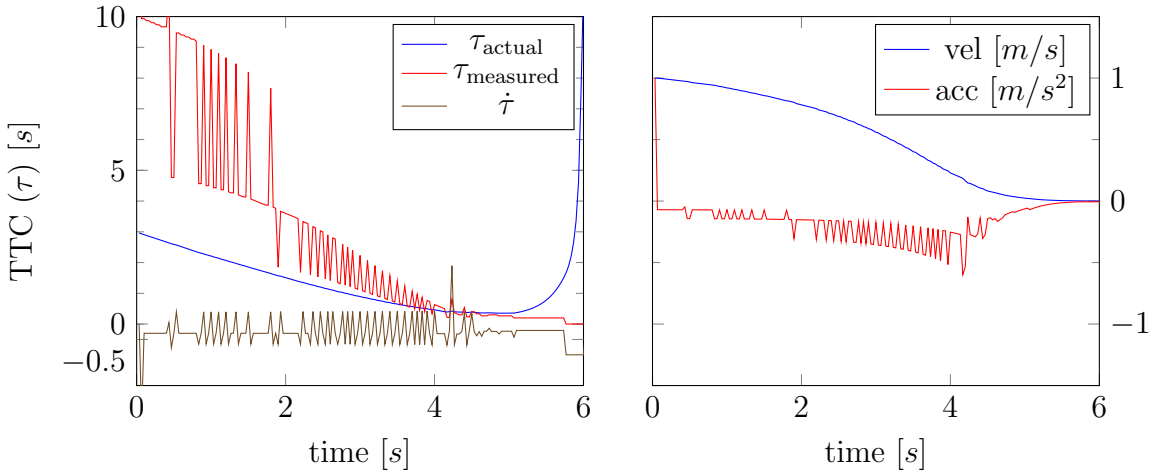


Figure 3.4: Controlled Braking - Safer Approach ($\dot{\tau} = -0.3$)

4. Vehicle Following

In this chapter we describe the full implementation and results of the vehicle following algorithm on a robotic system. In Section 4.1 we will first introduce the robot used for this implementation, the *Duckiebot*. We then introduce two controllers. Since this will be implemented on a real robot we require some method of lateral control, this is discussed in Section 4.2. We then discuss how the DVA model is transformed into a control law in Section 4.3. The algorithm is first implemented and examined in simulation (Section 4.4) before it is finally implemented on our physical robot (Section 4.5). The results for both simulation and physical experiment are presented in their respective Sections.

4.1 The Duckiebot

Very few robots are built with purely vision based control in mind. Luckily, however, the Duckiebot (Figure 4.1) was created for exactly that purpose. This whimsical little robot started as a course in advanced autonomy at Massachusetts Institute of Technology (MIT) [35]. The creators created the system as an "*open* [source], *inexpensive*, and *flexible* platform for autonomy education and research" [36]. As such it appears to be well suited for examining this algorithm.

The Duckiebot is composed of very few parts and is relatively inexpensive. It is built on a popular differential-drive hobbyist chassis, uses a single Raspberry Pi 2 [37] for all computations, and only a single monocular RGB camera with a fish eye lens for all sensing. The Raspberry Pi runs native Linux Ubuntu, allowing for a simple development environment to be created.

Despite its hardware simplicity, it's software architecture is comparable in complexity to a full scale autonomous vehicle. MIT built the Duckiebot software on top of the popular Robot Operating System (ROS) architecture. Although ROS is very powerful and well suited to the complexity of this system, one of the aims of this thesis was to allow for faster and easier prototyping of control algorithms, as well as simplify the control structure to make it even more approachable as an

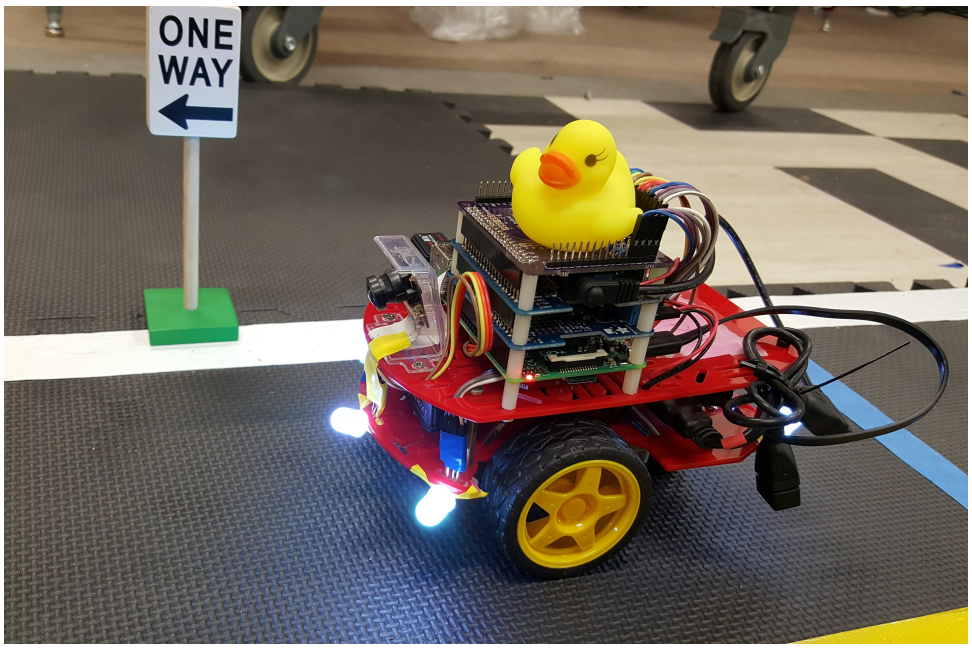


Figure 4.1: The Duckiebot

educational resource for inexperienced students. This was largely accomplished by the conversion of key parts of the Duckiebot's existing infrastructure from ROS to Robot Raconteur (RR). This will be discussed in more detail in Sections 4.1.3–4.1.5.

We first briefly introduce the robot kinematics, and review the pinhole camera model in Sections 4.1.1 and 4.1.2 respectively. In Section 4.1.3 we then discuss the overall system architecture. Finally, we detail the implementation of the two software modules that give us control over the Duckiebot's wheels and camera in Sections 4.1.4 and 4.1.5 respectively.

4.1.1 Robot Kinematics

The Duckiebot is a two-wheeled differentially-driven robot, with the main wheel axis forward placed and a rear-centrally placed ball caster for support. The kinematics of such a nonholonomic wheeled mobile robot are typically simplified by using a unicycle model [24]. The modeled unicycle wheel is placed centrally along the actual wheel axis. We therefore use this location to define the robot's position in the inertial frame (see Figure 4.2).

Wheeled vehicles cannot translate or slip in the lateral direction, leading to

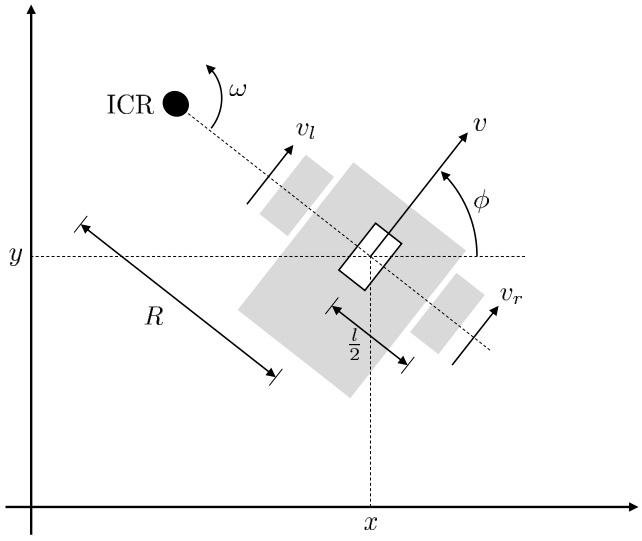


Figure 4.2: Kinematics Model for the Duckiebot

the nonholonomic constraint

$$\dot{x}\sin\phi - \dot{y}\cos\phi = 0 \tag{4.1}$$

The unicycle kinematic model is then typically given by

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\phi}\end{bmatrix} = \begin{bmatrix}\cos\phi & 0\\ \sin\phi & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}, \tag{4.2}$$

where $[v \quad \omega]^T$ is a 2-dimensional twist specifying the wheel's linear velocity and angular velocity about the instantaneous center of rotation (ICR) respectively.

Because the Duckiebot is a differential-drive robot, we must map the given twist into separate left and right wheel angular velocities, ω_l and ω_r. There is a one-to-one mapping between these values that can be determined as follows:

For rotating objects, the instantaneous linear velocity is related to the angular velocity by $v = \omega r$, where r is the radius from the axis of rotation. We note that although the linear velocity of each wheel will be different, they must give the same

angular velocity about the ICR, namely

$$\omega = \frac{v_l}{R - \frac{l}{2}} = \frac{v_l}{R + \frac{l}{2}}. \tag{4.3}$$

We can solve this expression for R,

$$R = \frac{l}{2}\frac{(v_l + v_r)}{(v_r - v_l)}, \tag{4.4}$$

and then substitute into 4.3 to obtain

$$\omega = \frac{v_r - v_l}{l} = \frac{r(\omega_r - \omega_l)}{l}. \tag{4.5}$$

Finally, we can solve for the linear velocity,

$$v = \omega R = \frac{r(\omega_l - \omega_r)}{2}. \tag{4.6}$$

We can similarly use equations 4.5 and 4.6 to solve for the inverse kinematics mapping: the required left and right wheel angular velocities given the desired twist,

$$\omega_l = \frac{v - 0.5\omega l}{r} \tag{4.7}$$

$$\omega_r = \frac{v + 0.5\omega l}{r} \tag{4.8}$$

For this implementation all distance measures are assumed to be in units of meters (m). Similarly, linear velocities and accelerations are given in units of m/s and m/s^2 respectively. Angles are in radians unless otherwise specified. Angular velocities are therefore in units of rad/s.

While the kinematics model generally describes the robot motion, it must be noted that the Duckiebot is clearly a prototypical system and thus will suffer from many nonidealities including unmodeled friction, wheel slippage, actuator dead-zones, and actuator saturation. Furthermore, without any direct feedback on the actuators, it is nearly impossible even with vision to ensure commanded velocities match the executed velocities. Therefore, in any case the commanded velocity is an

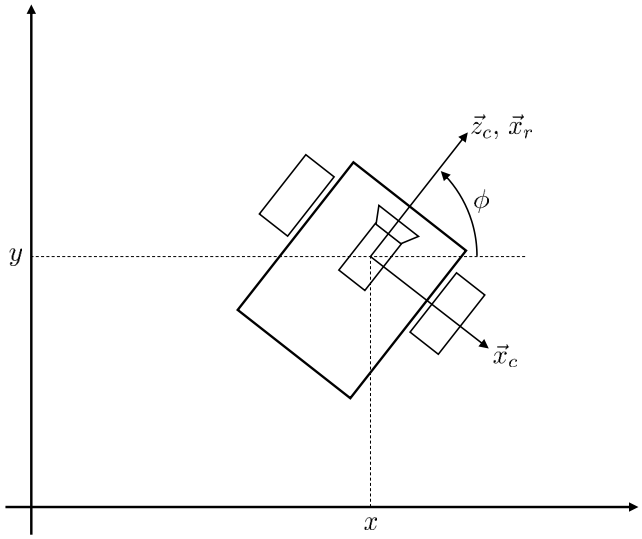


Figure 4.3: Relationship Between Robot and Camera Frames

open loop mapping.

4.1.2 Camera Model

For simplicity, the Duckiebot's camera will be modeled as a pinhole camera. It is assumed to be mounted such that the optical center is located at $[x\ y]^T$, the center of the vehicle in the 2-D inertial reference frame. We also assume that the camera's optical axis, $\vec{z}_c$, is aligned with the robot's forward axis, and it's $\vec{x}_c$ axis is parallel to the ground (see Figure 4.3). Our robot frame $(\mathcal{O}, \vec{x}_r, \vec{y}_r)$ therefore coincides with the $(\mathcal{O}, \vec{z}_c, -\vec{x}_c)$ frame.

For a pinhole camera, Cartesian coordinates given in the camera frame, $P = [X\ Y\ Z]^T$, are related to points in the image frame, $p' = [u\ v]^T$, by

$$u = f\frac{X}{Z} \qquad\qquad v = f\frac{Y}{Z}\,, \tag{4.9}$$

where f is the camera's focal length. We generally represent this using homogeneous coordinates as

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}, \tag{4.10}$$

where the scaling factor $\lambda = Z$.

Points in the image frame are subsequently mapped to pixel values, $p = [c\ r]^T$, according to the spatial quantization properties of the camera:

$$\begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & c_0 \\ 0 & s_y & r_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \tag{4.11}$$

where s_x and s_y are the spatial quantization scale factors (m/pixel), and c_0 and r_0 give the pixel location of the optical center.

Combining equations 4.10 and 4.11 we obtain the perspective projection equations:

$$\lambda \begin{bmatrix} c \\ r \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} fs_x & 0 & c_0 \\ 0 & fs_y & r_0 \\ 0 & 0 & 1 \end{bmatrix}}_{K} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}. \tag{4.12}$$

The matrix K is commonly called the *intrinsic camera matrix* as it stores all of the intrinsic properties of the camera. If the K matrix for a camera is known, the camera is said to be *intrinsically calibrated*.

Using this mapping we can uniquely determine the pixel location of any coordinate in the camera frame. Unfortunately, the inverse is not true. We can only determine a camera frame coordinate from a pixel location up to the scale factor, λ.

4.1.3 System Software Architecture

As was previously mentioned, there are two software packages that have been used to implement the Duckiebot's main system architecture: ROS [38] and Robot Raconteur [39]. Both architectures act as a sort of middleware that enables a modular hierarchy and data passing among different modules. For some applications the two architectures may also interface with one another and be used in conjunction.

ROS is one of the most popular and widely used robotics architectures. It is also the basis of MIT's original implementation for the Duckiebot system. While it is certainly powerful, some of it's intricacies and overhead make it unfriendly, especially

to those not already familiar with the architecture as a whole. Furthermore, it is not particularly easy to quickly prototype a new algorithm in ROS, since every piece of code must first be integrated into the larger software architecture.

Robot Raconteur serves essentially the same function as ROS, and implements some very similar features; however, it differs in that it uses a much simpler *service / client* structure to facilitate communication between different modules. Each service explicitly defines and implements an interface to some specific functionality or piece of hardware, and this interface is then available for use by any (or multiple) client(s). The required client setup is minimal, in fact, in the current releases of Robot Raconteur for python, all required setup can be accomplished in a single line of code by importing the python module `RobotRaconteur.Client`. Robot Raconteur also provides a service auto-discovery feature which can automatically find and make the connections to any desired services. One aim of this thesis was to simplify the Duckiebot's complex architecture to allow for faster prototyping in a more 'barebones' implementation. Towards this end, several key modules of MIT's original implementation have therefore been converted to Robot Raconteur services and then expanded upon.

The full robot controller is split into three self-contained modules, all running on the robot processor. The *Drive* module interfaces with the wheels to control the robots motion, while the *Camera* module interfaces with the on-board camera. These two modules are implemented as Robot Raconteur services. We can then implement the controller as a Robot Raconteur client so that we may connect and interact with these peripherals. This hierarchy is shown visually in Figure 4.4.

4.1.4 Drive Service

The Raspberry Pi interfaces with the robot's motors through 2 *Pi-HATs* (Hardware Attached on Top) which, as the name suggests, sits on top of the Pi's GPIO header pins. One HAT is a dedicated PWM signal generator, and the other is a motor controller capable of driving up to 4 motors. Both are commercially available from Adafruit, and come with drivers written in python to interface with them. MIT's implementation already includes ROS nodes written in python to make use

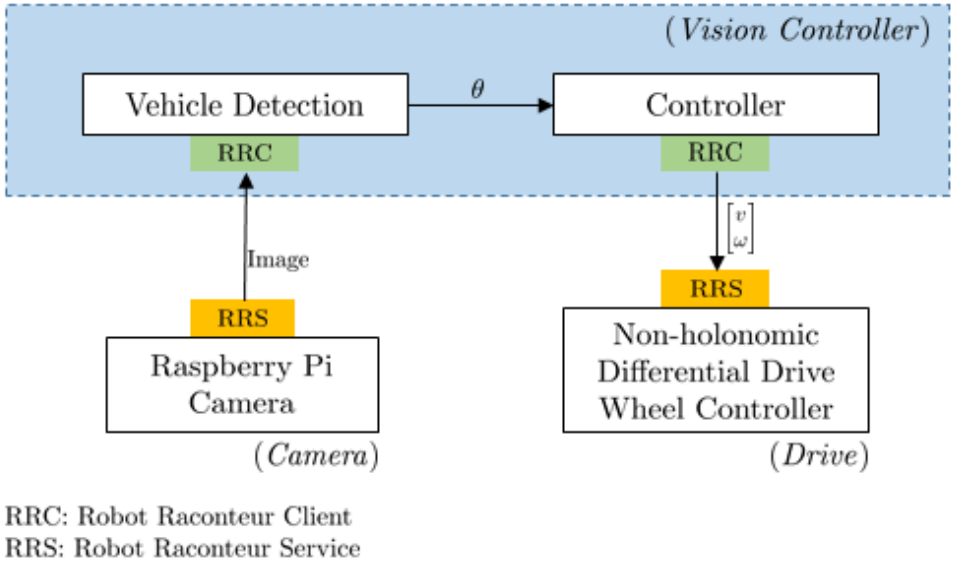


Figure 4.4: Block Diagram of Robot System Architecture

of these drivers and control the wheels, as well as implementing the forward and inverse kinematics. In this case these nodes have simply been stripped of their ROS components and replaced with similar Robot Raconteur calls to create a single service.

The *Drive* service at its core provides an interface to two functions that control the wheels. In the first, the client may directly specify the normalized left and right wheel velocities to send to the motor. In the second, the client supplies the desired 2D-twist which is then converted, using the inverse kinematics mapping, to the appropriate wheel velocities and sent to the motors. We will generally specify a twist in this implementation, as it is easier to work with in our controller.

4.1.5 Camera Service

Images are captured directly from the camera in a separate Robot Raconteur service written in C++. In this case, the use of C++ to handle the camera and Robot Raconteur interfacing gives us a significant performance boost over python, and we are able to obtain a maximum framerate of 30 frames per second (fps) if desired. Any client wishing to access the camera may then connect to the *Camera*

service and either directly request an image, or subscribe to a pipe to be automatically sent an image at some predefined framerate. Since in this implementation only a single program will be accessing the camera at any one time, we will generally choose to simply request a new image on every loop. Common camera settings like the image format, resolution, and framerate can also be queried or set from the Robot Raconteur service. In this implementation we do not use the color information in the image, therefore to save on both bandwidth and the need to perform a color conversion later we set the camera to directly capture grayscale images at a resolution of 640×480 pixels.

Wrapping the camera in a Robot Raconteur service is also beneficial to debugging and calibrating the vehicle detection algorithm (to be discussed in a later section). As the robot does not directly have a display we have no way of visualizing the detection performance as the algorithm runs. We can however connect to the *Camera* service from any other computer on the network over Robot Raconteur, provided the service is running on the robot. This allows us to run just the detection algorithm, with visualizations, on a separate computer and tweak and tune parameters as necessary. Once the parameters give a good performance we can upload a configuration file back to the robot and run the controller directly on the robot hardware. It is also possible to actually run the entire controller on the separate computer, but generally the bandwidth required to transmit the images is very large and the maximum possible closed loop rate (including the image processing in the controller) drops by about half for our system.

4.2 Lateral Controller

The lateral guidance controller implemented for this thesis is simply proportional control with the goal of keeping the image of our target vehicle near the optical center of the image. In the future lane detection and tracking could also be used, but this is generally more advanced and has not yet been implemented as a Robot Raconteur service. It is first useful to note that in our case, steering control can be decoupled from the velocity control, therefore any method we choose will not have an effect on the velocity controller (to be discussed in the next section). Since the

Duckiebot lacks any other feedback mechanism on the wheels to ensure a straight course, vision must be used in some way. For the purposes of this experiment we will assume the lead vehicle will follow a generally straight course, simplifying the task for the controller.

Assuming that the target has already been located in the image, we simply compute the location of the centroid. In nearly all practical cases the centroid of a bounding box is sufficient to estimate the objects centroid. The error signal is then defined by the normalized lateral distance from the optical center

$$e_{\text{lat}} = \frac{c_0 - c}{c_0}, \tag{4.13}$$

and the control law is simply given as

$$\omega = k_{\text{lat}} e_{\text{lat}}. \tag{4.14}$$

4.3 Visual Angle Based Controller

In this section we will discuss the controller used to control the vehicles velocity and track a lead vehicle. We currently assume that the object has already been detected in the image, and the visual angle can simply computed from the width or height of the bounding box. Note that for practical purposes, since the size of the bounding box in pixels is directly proportional to the visual angle, it could also be directly used as the measured variable.

The DVA model given in Equation 2.8

$$\ddot{x} = k_1 \left(\frac{1}{\theta} - \frac{1}{\theta_d} \right) + k_2 \dot{\theta} \tag{2.8 Revistited}$$

proposes a control law used by humans for following a lead vehicle based on the visual angle and it's derivatives. The authors in [32] sought to fit the gains k_1 and k_2 to human driver data; however, we would like to use them to tune the model and shape our response.

In order to determine our system dynamics, we would like to rearrange Equation 2.8 such that everything in terms of θ and its derivatives. For notational

convenience, we assume that both vehicles' motion is along the global $\vec{x}$ axis and denote the leaders position as x_L and our own position as x_F. We would therefore have the obvious relationship

$$Z = x_L - x_F. \tag{4.15}$$

Without loss of of generality, we may take the position of the lead vehicle to be at the origin, and thus remove x_L from Equation 4.15. Now taking derivatives we obtain the obvious expression for $\ddot{x}_F$.

$$\ddot{x}_F = -\ddot{Z} \tag{4.16}$$

If we may assume the small angle approximation of θ is valid, we know from Equation 2.4

$$\theta = \frac{S}{Z} \tag{2.4 Revisited}$$

that Z is also inversely proportional to the visual angle. We can similarly differentiate this expression twice to obtain

$$\ddot{Z} = \frac{2S\dot{\theta}^2}{\theta^3} - \frac{S\ddot{\theta}}{\theta^2} \tag{4.17}$$

Making the appropriate substitutions in 2.8 and rearranging, we may obtain

$$\ddot{\theta} = \frac{\theta^2}{S}\left[k_2\dot{\theta} + k_1\left(\frac{1}{\theta} - \frac{1}{\theta_d}\right) + \frac{2\dot{\theta}^2 S}{\theta^3}\right]$$

where everything is in terms of derivatives of θ and constant terms. Since we do not know the value of S we would ideally like to remove it from our expression as well. We therefore multiply out the $1/S$ factor and allow the constants k_1 and k_2 to absorb this value, resulting in

$$\ddot{\theta} = \theta^2\left[k_2\dot{\theta} + k_1\left(\frac{1}{\theta} - \frac{1}{\theta_d}\right) + \frac{2\dot{\theta}^2}{\theta^3}\right] \tag{4.18}$$

We note immediately that our dynamics are highly nonlinear.

To further examine the dynamics, we must linearize this system about an

equilibrium point. We intuitively know that if we are at our desired visual angle ($\theta = \theta_d$) and the visual angle is not changing ($\dot{\theta} = 0$) we should not change our current acceleration, therefore $(\theta, \dot{\theta}) = (\theta_d, 0)$ is an equilibrium point. It is easy to verify from Equation 4.18 that this is indeed true. We may use Equation 4.18 to solve for the Jacobian

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \mathbf{J} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} 0 & 1 \\ -(2\theta_d\dot{\theta}^2 - 2k_2\theta_d\dot{\theta}\theta^3 + 2k_1\theta^3 - k_1\theta_d\theta^2)/(\theta^2\theta_d) & (k_2\theta^3 + 4\dot{\theta})/\theta \end{bmatrix} \tag{4.19}$$

Substituting our equilibrium condition, we obtain the linearized system dynamics

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \mathbf{J}_{\text{eq}} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

$$\mathbf{J}_{\text{eq}} = \begin{bmatrix} 0 & 1 \\ -k_1 & k_2\theta_d^2 \end{bmatrix} \tag{4.20}$$

The eigenvalues (which give the poles) for this system are at

$$\frac{1}{2}\left[k_2\theta_d^2 \pm \sqrt{k_2^2\theta_d^4 - 4k_1}\right]. \tag{4.21}$$

We therefore know the for the system to be stable, k_2 must be negative. In many cases, a critically damped response may be desired, we can directly solve for this as well from the eigenvalues. The system will have a repeated root and therefore be critically damped, if the argument under the square root evaluates to zero. Therefore we may choose either gain (k_1 or k_2) and the following equivalent expressions give

the other

$$\begin{aligned} k_{1_{\text{crit}}} &= \frac{1}{4}k_2^2\theta_d^4 \\ k_{2_{\text{crit}}} &= -\frac{2\sqrt{k_1}}{\sigma_d^2} \end{aligned} \tag{4.22}$$

We may use a root locus analysis to more completely analyze the effect of different gains on our system. From Equation 4.20 we have the following linearized dynamics

$$\ddot{\theta} = -k_1\theta + k_2\theta_d^2\dot{\theta} \tag{4.20 Expanded}$$

which we can convert to the Laplace domain assuming zero initial conditions,

$$\begin{aligned} s^2\Theta - k_2\theta_d^2\Theta s + k_1\Theta = 0 \\ s^2 - k_2\theta_d^2 s + k_1 = 0 \end{aligned}. \tag{4.23}$$

By comparing the obtained characteristic equation to a standard second order system characteristic equation ($s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$) we parameterize the system as follows:

$$\begin{aligned} k_1 &= \omega_n^2 \\ k_2 &= -2\frac{\zeta\omega_n}{\theta_d^2} = -2\frac{\zeta\sqrt{k_1}}{\theta_d^2} \end{aligned} \tag{4.24}$$

Figure 4.5 shows the root locus for k_2 plotted with a constant k_1. We see that k_1 specifies the location of the open loop poles at $\pm j\sqrt{k_1}$ on the imaginary axis. This therefore defines the the radius of the circle the closed loop poles travel on. For a desired second order like response we may use k_1 to influence the convergence rate, and k_2 to specify the damping.

This is further confirmed by simulating the step response for the nonlinear dynamics, which directly incorporate the desired control law. We choose desired values of ω_n and ζ and compute the required k_1 and k_2 gains as given in Equation 4.24. The initial and reference angle conditions are obtained by projecting a $1m \times 1m$ square at distance of $2m$ and $1m$ from our modeled camera respectively. As expected, the obtained step response closely resembles the response expected for the given natural frequency and damping ratio (Figure 4.6).

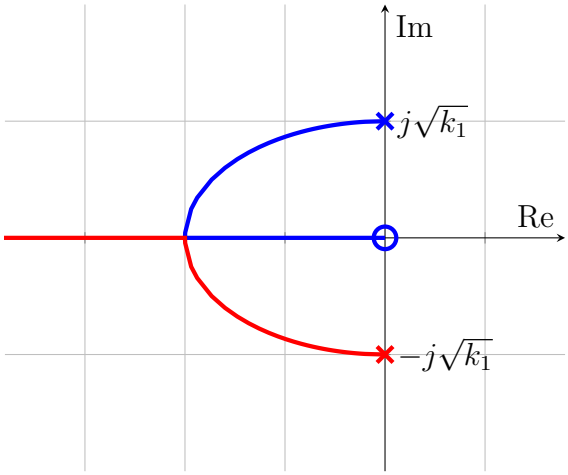


Figure 4.5: Vehicle Following Dynamics Root Locus
Root Locus vs k_2, with k_1 constant

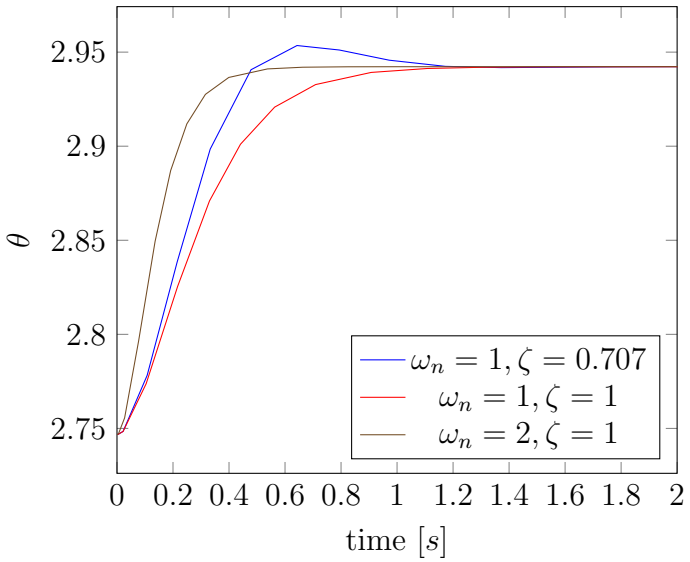


Figure 4.6: Nonlinear Simulation Step Responses

For our robotic system, we desire a close to critically damped response, since any overshoot corresponds to a shorter possible stopping distance if the lead vehicle were to suddenly brake.

4.4 Software Simulations

4.4.1 Algorithm Implementation

Since the eventual hardware implementation of this controller driving based on the visual angle will be inherently discrete, an effort has also been made to simulate the discrete control and better estimate the expected performance of the on-robot implementation. This has currently been implemented in Matlab.

Several vehicle characteristics are built into this simulation, including maximum and minimum accelerations and velocities, however, for the most part the values used do not accurately reflect the parameters of the Duckiebot. It is also assumed for this simulation that the kinematics model gives the exact robot dynamics, and the robot's relative position can therefore be determined by simply numerically integrating the velocity command at every time step. We assume that all motion from both vehicles is strictly along the optical axis, therefore the lead vehicle always remains centered in view and we only need to control velocity. This is obviously not the case with the real robot implementation.

We also make an effort to model the visual angle extraction process, using an ideal camera model. The exact coordinates of the lead vehicle are known to the simulation, therefore we can use the perspective projection equations to map them to a pixel value and extract a more realistic 'measured' value for the visual angle. This allows us to directly examine and model the effect that perspective projection and camera quantization has on our controller. Specifically the camera limits the accuracy of our measurement of θ.

The continuous time control law is converted to a discrete time difference equation using the Tustin approximation (bilinear transform),

$$\ddot{x}[k] = k_1 \left(\frac{1}{\theta[k]} - \frac{1}{\theta_d} \right) + k_2 \frac{2}{T} \frac{\theta[k] - \theta[k-1]}{\theta[k] + \theta[k+1]} \tag{4.25}$$

This generally approximates continuous time differentiation in the limit as the sampling period T approaches 0, and therefore works best at faster sampling times. Since image processing is known to be computationally intensive, we are generally able to achieve only moderate closed loop rates. A simulation environment is

therefore a useful way to evaluate if this approximation will be sufficient for control.

Because we have only velocity level control available to us, we cannot directly implement a commanded acceleration. To approximate this, the desired acceleration is numerically integrated to determine the velocity command. In general we expect this approximation may have some effect on the quality of our controller.

Remark. *If our loop can be closed sufficiently fast, we could potentially ignore the dynamics, and implement solely kinematic control for keeping a following distance, or more precisely matching a desired pose. This is traditionally done by converting the vehicle kinematics to polar coordinate form, which then directly models the following distance and heading angle. From* [20]*, we would therefore have the following kinematic expressions (see also Figure 4.7)*

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos\alpha & 0 \\ \frac{\sin\alpha}{\rho} & -1 \\ \frac{-\sin\alpha}{\rho} & 0 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}, \quad \alpha \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right] \tag{4.26}$$

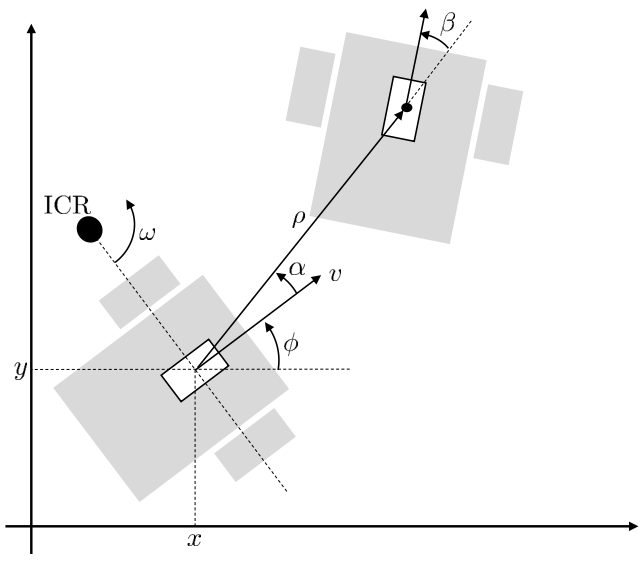


Figure 4.7: Vehicle Kinematics In Polar Coordinates for Matching a Pose

We can then apply either a linear control law

$$v = k_\rho(\rho - \rho_d), \quad \omega = k_\alpha\alpha + k_\beta\beta \tag{4.27}$$

or some form of a Jacobian pseudo-inverse controller. We could still approximate ρ, which just gives our separating distance, using $1/\theta$. The angle α is essentially the error we are already using in our lateral controller. The angle β would either require some additional method to estimate, or can just be ignored, as it would typically be fairly small. We could also incorporate $\dot{\theta}$ as a feed-forward, or predictive term for the velocity. In that case, our final control law for the linear velocity would be very similar to that given by Equation 2.8, except replacing the vehicle acceleration $\ddot{x}$ with velocity $\dot{x}$.

4.4.2 Simulation Results

In a simulation environment we have the capability of fully specifying the lead vehicle's velocity profile and examining how our control law reacts. We therefore are able to test several different scenarios to examine tracking performance. We use a closed loop rate of 10Hz since this is approximately the rate we can achieve on the Duckiebot. In addition to standard step and ramp responses we have also implemented sinusoidal velocity profiles and sudden decelerations to better predict the expected performance of the algorithm.

We parameterize k_1 and k_2 in the same way as previously discussed, however we note that in a discrete time system we usually desire w_n to be sufficiently less than our sampling frequency, ω_s, but still fast enough to ensure good convergence, thus placing a limit on k_1. These gains must largely be tuned to fit the response, but to simplify matters we specify that $\omega_n \leq \omega_s/5$ and therefore choose $k_1 = (\omega_s/5)^2$. This leaves only the desired damping ζ to be tuned in order to fully specify k_2. We examined several cases of ζ on a step input to determine the approximate discrete system dynamics (Figure 4.8). We ultimately note that both an underdamped, and critically damped step response still include a considerable amount of oscillation. This is somewhat expected since we have introduced saturation constraints on the control input, and the system therefore cannot fully realize the desired response. By

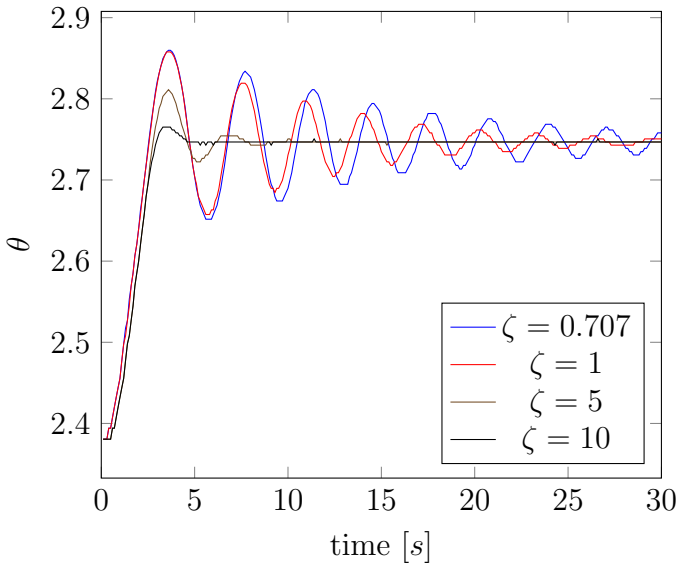


Figure 4.8: Discrete Time Simulation - Step Response with Variable Damping Ratios

significantly overdamping the controller we are able to remove the oscillations. This similarly suggests that the physical implementation will likely require additional damping.

Using an appropriate damping coefficient so that the system settles quickly and without oscillations we then examine several other trajectories of interest (Figure 4.9). We note that we can still track both a ramp and even a sinusoidal velocity profile using the visual angle alone. Finally, we examine the system's response to sudden decelerations. In all situations tested, the lead vehicle brakes at its maximum possible deceleration, and the system is similarly able to come to a stop in a safe manner (Figure 4.10). Although this is largely a function of the actual velocities and accelerations / decelerations of the vehicle, as well as the following distance, this strategy may also have the potential to be used for braking situations. We note however, that it does not guarantee a constant deceleration profile, which would be more comfortable for passengers in a vehicle and a more efficient method for braking.

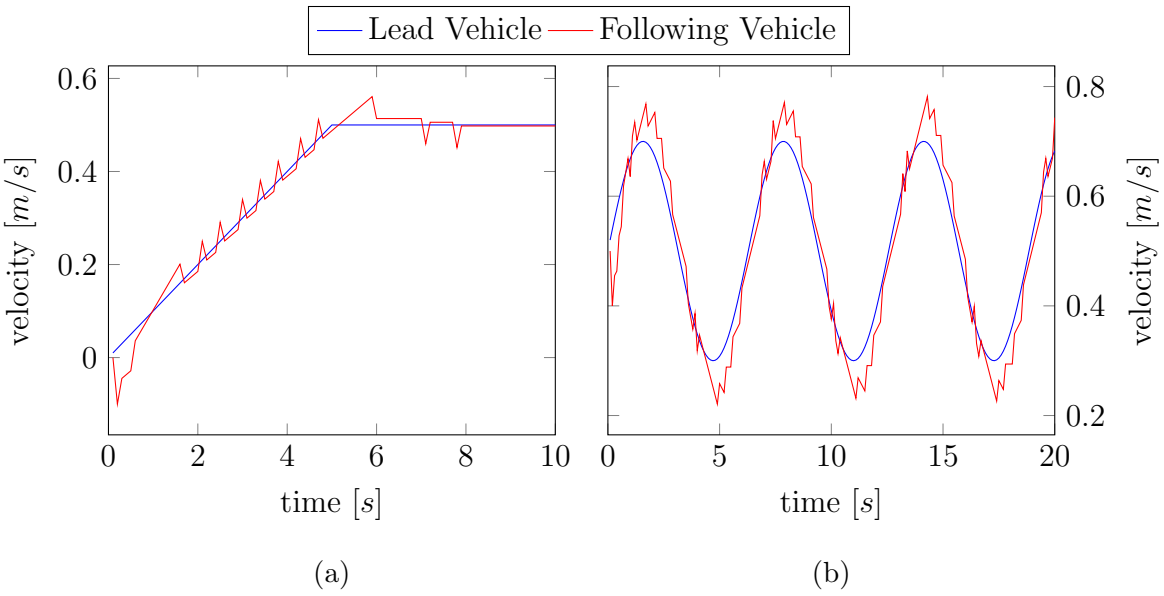


Figure 4.9: Discrete Time Simulation - Trajectory Tracking
(Fig 4.9a) Ramp response. (Fig 4.9b) Sinusoidal response.

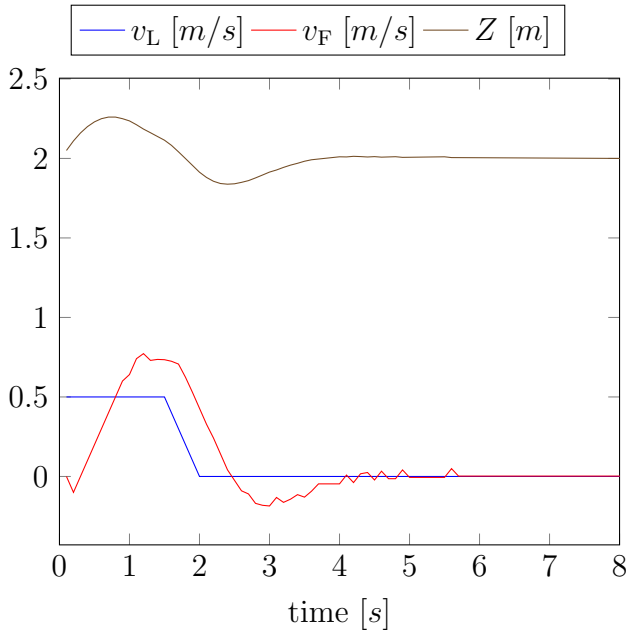


Figure 4.10: Discrete Time Simulation - Sudden Deceleration
The two vehicles start with different initial velocities. The lead vehicle then brakes while the following vehicle is still accelerating. The desired separation distance (Z) is $2m$.

4.5 Robot Implementation

The majority of our on-robot implementation was written in the Python programming language, making extensive use of both the OpenCV [40], and Numpy [41] libraries to avoid re-writing existing common computer vision or linear algebra algorithms.

Unfortunately, image processing is very computationally intensive and generally slow, especially on our small processor. We are able to achieve a closed loop rate of only 10Hz for the implemented algorithms, which generally limits our controller bandwidth.

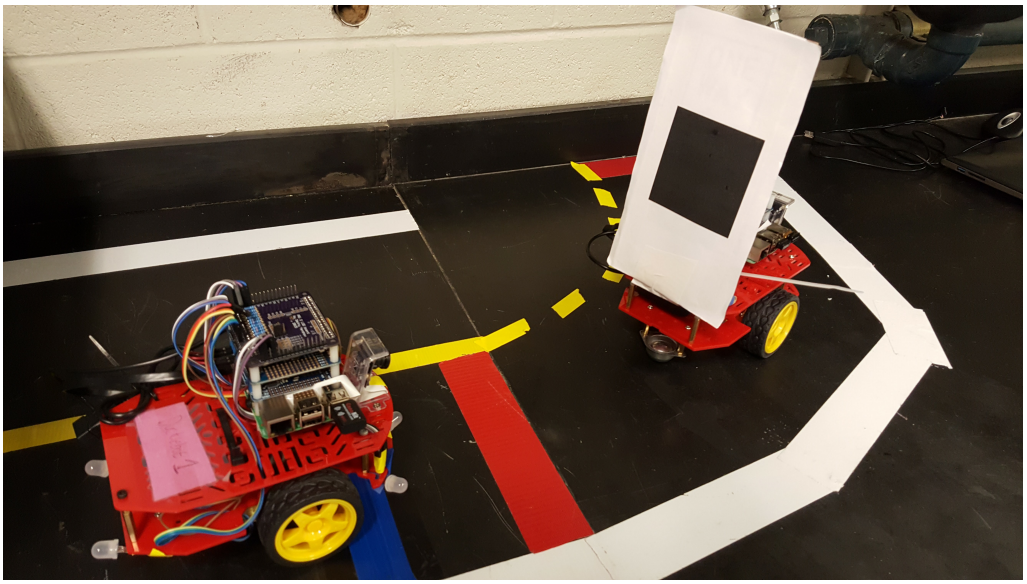
As an aside: although python is an interpreted language and is therefore generally slower than a compiled language like C/C++, it can interface with compiled C/C++ libraries, as is the case for both OpenCV and Numpy. This is beneficial in that we can quickly and flexibly prototype software in python and get excellent speed performance from the underlying libraries. Fortunately/unfortunately this also means that the performance in python is nearly optimal on our given hardware and we will not see a significant performance boost by porting the software to a compiled language. We of course learned this the hard way after the fact.

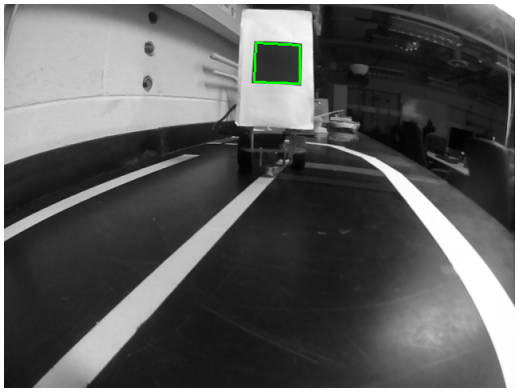
4.5.1 Vehicle Detection Implementation

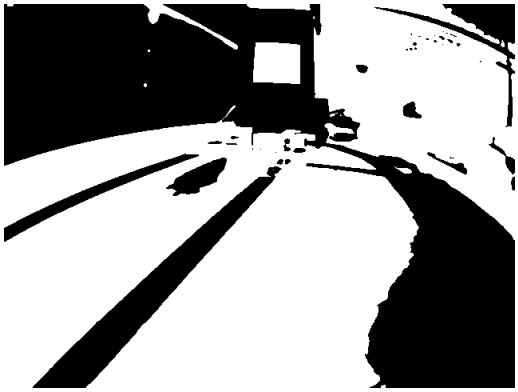
To facilitate the lead vehicle detection in this implementation, we make use of a simple tag that can be affixed to another Duckiebot or placed in the environment. The tag used in this implementation is simply a black square (or rectangle) on a field of white (Figure 4.11). Our detection algorithm is therefore based primarily on thresholding the grayscale image to highlight the darkest portions, and then applying geometric template matching to locate squares. The vertices of the largest square, which typically corresponds to the tag, is then output (if it is detected) for use in the control.

It should be noted that because the detection tag used in this implementation is so simplistic, the detection is particularly prone to false positives and is sensitive to the orientation of the square (i.e. it does not detect well when the square is viewed at an angle). While at close to mid ranges the largest square will typically be the

(a)







(b) (c)

Figure 4.11: Vehicle Tag Detection
(4.11a) A tag can be placed directly on another duckiebot. (4.11b) The camera view and detection (green outline). (4.11c) The binary thresholded image (white corresponds to the darkest pixels in the image)

tag, this is not necessarily true as the tag moves further away. This implies that for a situation like we see in the braking controller, where the tag must be detected reliably from far away, we typically need to use a much larger tag. The thresholding operation is also largely sensitive to lighting conditions and must therefore be recalibrated for new environments. One option that was explored but ultimately not used in this work was the use of color based thresholding. While this certainly has the potential to further reduce the areas of interest in the image and may be

useful for detecting other types of objects or tags, it ultimately suffers the same sensitivity to changes in lighting and therefore does not offer significant benefit in our implementation. A more visually distinct tag and some other method of detection based on feature matching would likely be needed to more reliably ensure the tag is correctly detected.

Although it does not severely impact our implementation in all cases, the Duckiebot camera's fisheye lens also introduces several nonidealites. The wider angle of view comes at the expense of radial lens distortion. Our detection algorithm is generally unaffected if the square stays near the center of the image, where the image is least distorted. However, at closer ranges or if the lead vehicle moves too far to the side the square no longer appears square and we lose our detection. We can remove the effect of this distortion by calibrating the camera and computing a look up table to remap the pixel locations in the distorted image to their undistorted locations. This of course comes at the expense of another costly image manipulation, but it is generally recommended. It should be noted that in general extracting a visual angle is completely invariant to the internal camera parameters, and therefore in a general setting camera calibration would not be required. It is only due to our chosen detection scheme that we must apply this remapping.

In the future, this remapping may be directly incorporated into the Camera Robot Raconteur service to remove the burden from the client, but it is currently implemented on the client side. The camera parameters generally need only be determined once as they typically do not change. The calibration was therefore implemented offline and then the values are simply passed to the program at runtime.

Camera calibration is often performed using a known pattern, such as a chessboard (Figure 4.12), so that the expected and measured feature points can be be compared in order to compute the intrinsic parameters and distortion coefficients. Once these parameters are known the remapping tables can similarly be computed. At runtime, we must then use this precomputed table to remap every image. In this work, we used OpenCV's implementation of both camera calibration and the distortion correction remapping. For a more complete treatment of the theory behind camera calibration and distortion correction see [33].

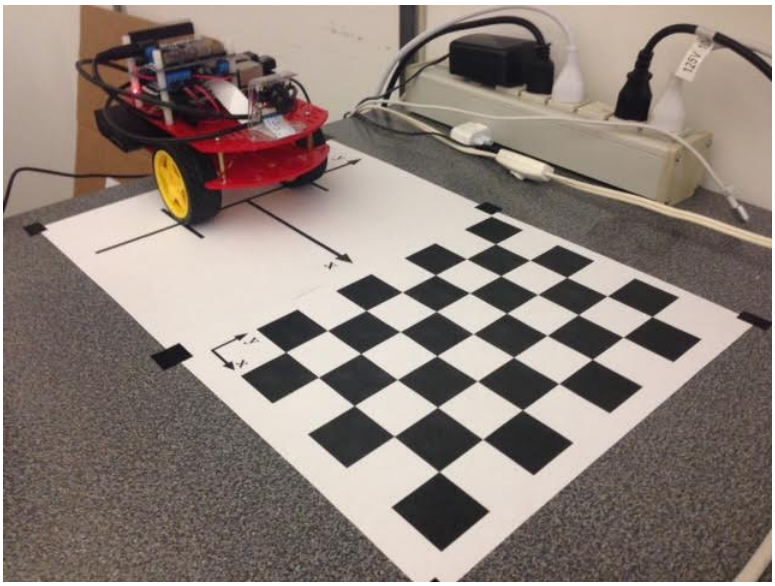


Figure 4.12: The camera calibration setup for the Duckiebot. (Photo Courtesy of [35])

The full Vehicle Detection algorithm is described in Algorithm 4.1. The functions called by this algorithm refer almost exclusively to similarly named OpenCV methods.

Algorithm 4.1 Vehicle Detection

```
procedure VehicleDetection(rawImage)
    Image ← Remap(rawImage,distortionMap)                    ▷ Undistort
    Image ← GaussianBlur(Image)                              ▷ To denoise
    threshIm ← Threshold(Image, minThreshold)
    contours ← FindContours(threshIm)
    for all contours do
        vertices ← ApproximatePolygon(contour)
        if Size(vertices) = 4 AND
          Area(vertices)>AreaThreshold AND
          Angles(vertices)≈ 90° then
            add to squares                    ▷ It is a sufficiently large square
        end if
    end for
    return square in squares with maximum area
end procedure
```

4.5.2 Vehicle Following Implementation

To make debugging and tuning simpler most of the parameters required by the program are stored in a configuration file and loaded during an initialization routine. This includes parameters for the vehicle detector, the saved camera intrinsic parameters, and any controller gains. Once the program has been initialized with the appropriate parameters, we then connect to both the *Camera* and *Drive* Robot Raconteur services, which must already be running on the robot.

We run an initial detection routine before starting the main loop. This constitutes the teach-by-showing phase in which we determine the desired visual angle to be used by the controller as a reference. The vehicle must therefore be manually placed the desired distance from the lead vehicle. The program will print the location of the detected tag to the terminal to ensure that tag has actually been detected. Obviously it is difficult to determine if the reported detection is actually the desired tag, therefore the detection parameters should be calibrated before use to ensure that the tag is generally detected.

Once the program is fully initialized the main control loop begins. We request the image from the *Camera* service and then call the detection routine. If the tag is detected we compute the location of the centroid to use for lateral steering control.

The error from the optical center is determined and used as feedback to keep the vehicle on course. We compute the visual angle directly from the width (in pixels) of the detected tag. We then directly apply Equation 4.25 to determine the new required acceleration, and from that the velocity we will command.

So long as we have found the vertices of the tag in the image we can update our controller, otherwise we must use a fallback option. To avoid the robot wandering away, our implemented fallback option stops the vehicles forward motion, but allows it to retain the previous iteration's steering error. In this way the robot may continue to attempt correcting for steering if that was the cause of the lost detection. We have also implemented a failsafe that will stop the program completely if the tag is not detected for some number of iterations.

The program is summarized algorithmically in Algorithm 4.2.

4.5.3 Experimental Results

The physical Duckiebot implementation was tested using the general setup shown in Figure 4.13. A tag is affixed to another lead Duckiebot, whose motion will be controlled through open loop command using a joystick. Both robots are placed on a straight piece of roadway (although the road isn't generally required), starting with the desired separation of approximately $0.5m$ or less.

The results seen in these experiments are summarized in the video available at `https://youtu.be/Qlrwq6SFWCk`. To generally summarize the results, the robot is largely capable of following the lead vehicle at the desired distance, although the actual gap tends to be slightly larger while the vehicles are moving and then returns to approximately the desired once the lead vehicle stops. We also tend to see small movements or oscillations after the lead vehicle has come to a stop in which the following vehicle tries to correct for a small differences in the visual angle or the lateral offset. These inconsistencies with the simulation are largely a result of errors incurred by relying on open loop control of the wheel velocities. Without additional feedback on the wheels, either using visual odometry or more simply a wheel encoder, it would be impossible to remove these errors using the current hardware. Some of these minor inconsistencies could also be improved with

Algorithm 4.2 Vehicle Following

```
procedure VehicleFollowing(configFile)
    InitializeParameters(configFile)

    cam ← RobotRaconteurConnect('camera')
    drive ← RobotRaconteurConnect('drive')

    θ_desired ← GetInitialDetection

    while True do                                          ▷ main loop
        rawImage ← cam.CaptureImage( )                     ▷ Request Image
        verts ← VehicleDetection(rawImage)

        if verts is not empty then
            noDetectionCounter ← 0
            c,r ← DetectCentroid(verts)
            θ ← CalculateAngle(verts)

            θ_err ← (1/θ) − (1/θ_desired)
            θ̇ ← (2/T)(θ − θ_prev)/(θ + θ_prev)             ▷ Tustin Approx

            a ← k1 θ_err + k2 θ̇                            ▷ acceleration
            v ← v + a/T                                    ▷ velocity
        else
            noDetectionCounter ← noDetectionCounter + 1
            if noDetectionCounter < noDetectionLimit then
                c ← c_prev
                θ ← θ_desired
                v ← 0
            else
                return                                     ▷ Exit the loop
            end if
        end if
        e_lat ← (c0 − c)/c0                                ▷ normalized column error
        ω ← k_lat e_lat

        drive.CarCmd(v, ω)                                 ▷ Send Twist Command
    end while
end procedure
```

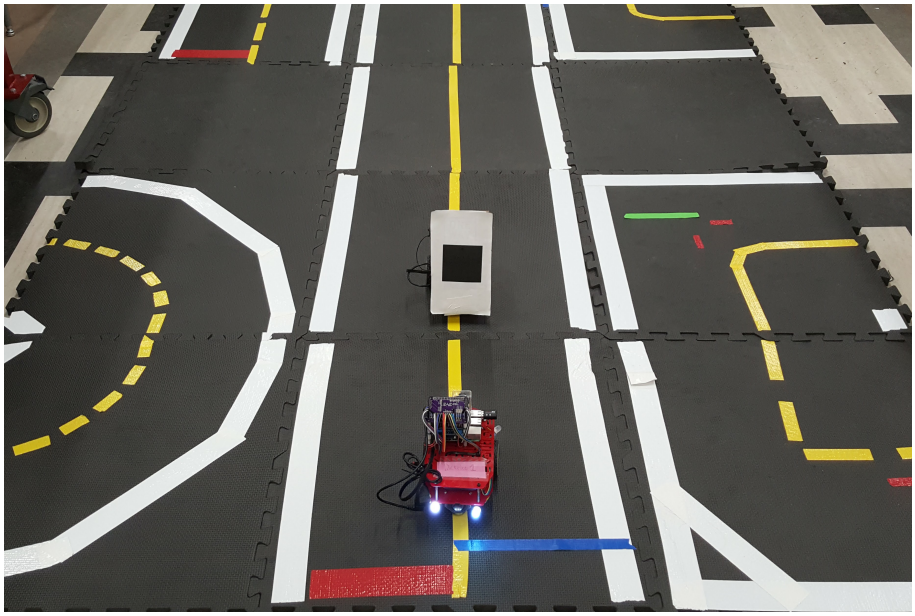


Figure 4.13: General Experimental Setup for Duckiebot Following

further tuning of the gains k_1 and k_2. Despite these small errors, for the purposes of demonstrating the algorithm, these experiments provided a fair representation of the desired response.

Figure 4.14 shows the velocities commanded by both the lead and following vehicle. We notice that the following vehicle commands the same general velocity profile as the lead vehicle, including both forwards and backwards motion. The backwards motion (Figures 4.14c–4.14d) was generally tested only in small bursts as it is more difficult for both Duckiebots to maintain straight backwards motion. It should be noted that the locations where the following vehicle's velocity suddenly drops to zero indicate a loss in tracking, where we have for safety reasons manually set the velocity to zero. We also notice that the velocity in nearly all trials was kept fairly low. In Figure 4.14e we increased the velocity to approximately twice the other trials and note that eventually the following Duckiebot lost tracking and stops before we stopped the lead Duckiebot.

The lateral tracking is also reasonable, but has mixed results largely due to both the detection algorithm used and the vehicle kinematics. Lateral tracking during approximately straight motion performs well, but the Duckiebots noticeably have a harder time generating small angular velocities which tends to lead to a

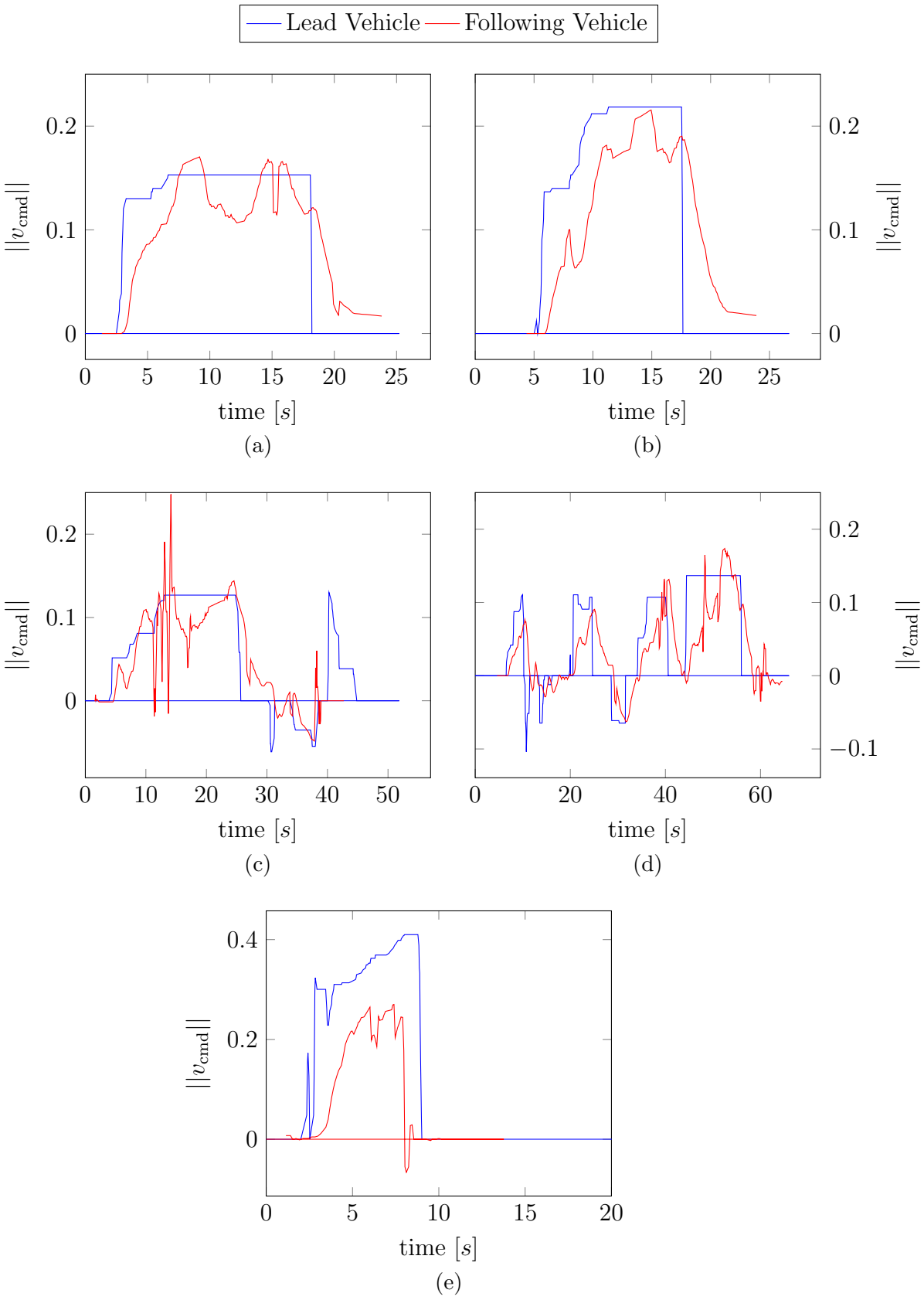


Figure 4.14: Comparisons of the Commanded Velocity Profiles Between Leader and Autonomous Follower

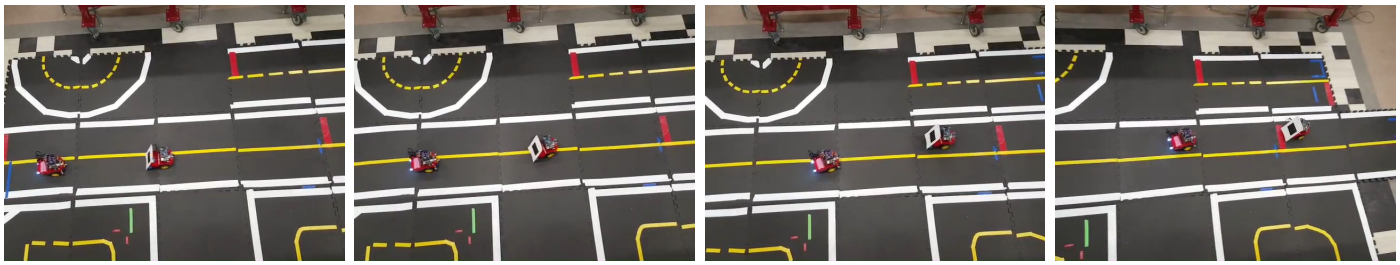


Figure 4.15: Autonomous Duckiebot Performing a Lane Switch

more jerky steering. This is largely unavoidable in a differential drive robot without any actuator feedback. Small heading changes (i.e. a lane change or wide arcs) were also tested, although we have not plotted the angular velocities for each run because it is much harder to draw any comparison between them due to the constant correction required to maintain a straight course. Despite these slight imperfections in the heading control, the autonomous Duckiebot is still able to follow the lead vehicle in very wide arcs; Figure 4.15 shows the Duckiebots performing a lane switch. If the lead vehicle changes its heading too quickly however, as expected our tag detection algorithm loses tracking as it cannot detect the tag at an angle. As mentioned, because of the Duckiebot's jerky steering and differential drive setup large 'jerky' heading changes are difficult to avoid even for the joystick controlled lead Duckiebot. This arguably might not be such a problem for a car-like vehicle that cannot physically turn its wheels in opposite directions and produce a pure rotation. Since the view would change more gradually our simplistic detection scheme may still work fairly well, however there will always be a critical angle at which the detection scheme used in this thesis is no longer applicable.

As has been mentioned, since the Duckiebots have such little mass, they are capable of accelerating and decelerating very quickly, even without active braking. Furthermore their top speed has been measured to be only slightly higher than $0.5m/s$ so they are not ultimately moving that fast to begin with. Although this strategy appears to work well for the Duckiebot, further study is needed to determine if the algorithm holds up at faster speeds, and with a larger vehicle that can generate more momentum.

Ultimately, despite the several ways in which the Duckiebot is an imperfect

testbed, we are still able to conclude that the information provided by a visual angle alone could be beneficial in controlling headway. While it is less robust, and unlikely to outperform a radar or laser system, we are still able to perform the same task of maintaining headway using visual cues alone.

5. Conclusion and Future Work

5.1 Summary and Comparison of Approaches

In this thesis, two vision based strategies for controlling headway on a mobile robot or autonomous vehicle have been proposed and implemented. Both strategies are based on human visual perception models, and have been shown to provide sufficient feedback information for a robotic application to control headway either for maintaining a constant following distance or controlled braking approach. In fact, these same algorithms could be applicable to a wide variety of use cases beyond the driving applications that we have applied them to. The actual implementation of these algorithms had mixed success, largely due to measuring errors; however, it is likely that these errors could be reduced with further research, so it does not necessarily condemn the algorithms theoretical basis.

Both methods are essentially based on extracting the visual angle of the object from the image features. Ultimately this can be done in a variety of ways. In this research we use a geometric template matching approach, but find that for a full scale implementation a more robust method would likely be required.

The tau based braking controller has some significant challenges, largely due to the fact that τ and $\dot{\tau}$ must be determined from some combination of the visual angle measurements and kinematic measurements. The controller was shown to be effective in the ideal case when τ closely approximates the actual time-to-contact. However, the proposed method of measuring τ suffers from errors introduced by the camera projection and quantization, and in the end, this method of measuring τ was too unreliable for the controller to be effective. We do note that even with these measuring errors, our implementation was still able to bring the simulated vehicle to a stop in a more controlled manner than simply applying full braking force. Furthermore, there are additional ways to measure τ from an image that may prove more accurate so that the proposed controller is more effective.

The visual angle based vehicle following controller had greater success, both in simulation and the hardware implementation on our Duckiebot robot. The algo-

rithm is in a way simpler than tau as it eliminates many of the numerical complexities and is based directly on the visual angle measurements rather than a compound variable. This method was shown to be effective for keeping a constant headway, and may even be applicable in braking scenarios, although future research with a better robot or vehicle testbed would be required to confirm this.

Ultimately we have shown that although vision may not be as robust as other sensors, it can still provide a great deal of information useful for feedback control. While a pure vision approach to autonomous driving tasks may not be the most practical, this work emphasizes the fact that there is still useful visual information that can directly be incorporated into the vehicle's controller.

5.2 Directions for Future Work

In this section we suggest various improvements to the current implementation that could be made, as well as some other possible use cases for the visual control methods discussed in this work.

5.2.1 Alternative Tau Measurement Methods

Measurement of Time to Contact (τ) from image sequences is an active area of research in the computer vision field. We used a very simplistic measurement in this work so perhaps an alternative method would prove more accurate.

Future work would examine several other methods for estimating τ in the hopes of improving the accuracy of the measurement. The algorithm in [34] directly estimates τ from the entire image or a specific region of interest without any need to detect feature points. The work in [25] also reports success with making the IBVS task more stable by incorporating an estimator to determine the distance to the features of interest. A similar estimator could be used to correct or replace the τ measurement obtained directly from the image features. This is also related to the next section.

5.2.2 Relative State Observers

Many autonomous vehicle applications incorporate some tracking or estimation of the obstacles and lead vehicles once they have been detected. It is clear that there are many cases in which this kind of predictive control could augment the controllers designed in this work, either to give a more accurate distance estimate or predict changes in velocity. This is especially important for vision based approaches in which lost detections are almost guaranteed due to changes in lighting, shadows cast on the roadway, or some other occlusion. Similarly, we have only examined these scenarios with a single vehicle, and several efforts have been made to reduce false positive detections. In a real driving scenario this would be much more difficult. A state observer could similarly be applied to the image features to ensure we are tracking the correct object of interest.

Future work would attempt to incorporate an observer, ideally still based on the image features, to track the vehicle position. The authors in [20, 25] give examples of such an estimator being applied to Image Based Visual Servoing problems.

5.2.3 Alternative Lateral Guidance

Very little has been done in this implementation to incorporate more complex lateral control methods with the proposed longitudinal controllers. Lane guidance systems are commonly used in autonomous road vehicles and therefore would likely constitute a better strategy for lateral control than simply following a lead vehicle. The longitudinal controller would then only take effect when a lead vehicle is detected to adjust for velocity. We also recognize that in many situations, steering also plays a large role in obstacle avoidance and collision prevention besides controlling speed.

Future work will implement the lane detection and pose estimation as a separate Robot Raconteur service so that it may also be used in the vehicle control. Many methods exist for both the lane line extraction and the pose estimation steps (see Chapter 2). For example, MIT's original Duckiebot implementation uses a color thresholding and Hough Transform approach to lane detection and a Grid-Based Bayesian filter for pose estimation [36]. Ultimately this method is still sensitive to

changes in lighting conditions and the Bayesian Filter is computationally demanding so other methods may also need to be explored.

5.2.4 Alternative Robot Hardware

We have already discussed several ways in which changes to our robot hardware, the Duckiebot, would be beneficial, mostly related to adding some form of additional velocity feedback to the wheels. There are also several other changes to the hardware that would be beneficial for further evaluation of the designed algorithms. While the algorithms discussed in this work do not require active braking, they have been designed with a system that employs active braking in mind (i.e. a passenger car). Our current robotic hardware implementation is not capable of actually generating a braking force, or of reaching the required speeds where such a braking force might be necessary.

Future work will attempt to implement these same algorithms on a larger scale system with both tighter velocity control and faster top speeds, as well as active braking. Ideally these algorithms or derivatives thereof would eventually be implemented in a full size car.

5.2.5 Other Robotic Applications

Although we have solely examined these models in the case of a mobile robot and a driving application, the same models could also be applied in other situations, including manipulation and grasping tasks. In particular grasping problems that are either performed autonomously or through shared control with a human may benefit from the use of time to contact information in particular, as it allows for a pure feedback approach that does not necessarily need to be preplanned. These controllers could then supplement more traditional image based visual servoing techniques. Future work unrelated to driving could apply these same models to manipulator robots to examine more use cases and determine if they can provide similar benefits.

REFERENCES

[1] Waymo, "Waymo, A Self Driving Technology Company," 2017. [Online]. Available: https://waymo.com, Accessed on: Feb. 18, 2017.

[2] Tesla, "Tesla Autopilot," 2017. [Online]. Available: https://www.tesla.com/autopilot, Accessed on: Feb. 19, 2017.

[3] Uber, "Pittsburgh, Your Self-Driving Uber is Arriving Now," 2016. [Online]. Available: https://newsroom.uber.com/pittsburgh-self-driving-uber/, Accessed on: Feb. 19, 2017.

[4] D. Muoio, "These 19 companies are racing to put driverless cars on the road by 2020," 2016. [Online]. Available: http://www.businessinsider.com/companies-making-driverless-cars-by-2020-2016-8, Accessed on: Feb. 19, 2017.

[5] A. Broggi *et al.*, "Intelligent Vehicles," in *Springer Handbook of Robotics*, 1st ed., B. Siciliano and O. Khatib, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ch. 62, pp. 1627–1655.

[6] E. D. Dickmanns, "The development of machine vision for road vehicles in the last decade," in *Proc. IEEE Intell. Veh. Symp.*, vol. 1, Jun. 2003, pp. 268–281.

[7] Darpa, "Darpa Grand Challenge," 2005. [Online]. Available: http://archive.darpa.mil/grandchallenge05/, Accessed on: Feb. 18, 2017.

[8] Darpa, "Darpa Urban Challenge," 2007. [Online]. Available: http://archive.darpa.mil/grandchallenge/, Accessed on: Feb. 18, 2017.

[9] A. Broggi *et al.*, "The VisLab Intercontinental Autonomous Challenge: an extensive test for a platoon of intelligent vehicles," *Int. J. Veh. Autonomous Syst.*, vol. 10, no. 3, pp. 147–164, 2012.

[10] E. Guizzo, "How Google's Self-Driving Car Works," 2011. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works, Accessed on: Feb. 24, 2017.

[11] D. Pomerleau and T. Jochem, "Rapidly adapting machine vision for automated vehicle steering," *IEEE Expert*, vol. 11, no. 2, pp. 19–27, Apr. 1996.

[12] E. D. Dickmanns and B. D. Mysliwetz, "Recursive 3-D Road and Relative Ego-State Recognition," in *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, Feb. 1992, pp. 199–213.

[13] C. J. Taylor *et al.*, "A comparative study of vision-based lateral control strategies for autonomous highway driving," *Int. J. of Robotics Res.*, vol. 18, no. 5, pp. 442–453, 1999.

[14] J. C. McCall and M. M. Trivedi, "Video-based lane estimation and tracking for driver assistance : Survey , system , and evaluation Video-based lane estimation and tracking," *IEEE Trans. Intell. Transp. Syst.*, vol. 7, no. 1, pp. 20–37, Mar. 2006.

[15] N. Apostoloff and A. Zelinsky, "Robust vision based lane tracking using multiple cues and particle filtering," in *Proc. IEEE Intell. Veh. Symp.*, Jun. 2003, pp. 558–563.

[16] C. Coué *et al.*, "Bayesian occupancy filtering for multitarget tracking: An automotive application," *Int. J. Robotics Res.*, vol. 25, no. 1, pp. 19–30, 2006.

[17] L. E. Weiss, A. C. Sanderson, and C. P. Neuman, "Dynamic Sensor-Based Control of Robots with Visual Feedback," *IEEE J. Robot. Autom.*, vol. 3, no. 5, pp. 404–417, 1987. [Online]. Available: http://www.cs.cmu.edu/~lew/PUBLICATION%20PDFs/VISUAL%20SERVOING/JRA%201987.pdf, Accessed on: Mar. 30, 2017.

[18] B. Espiau, F. Chaumette, and P. Rives, "A new approach to visual servoing in robotics," *IEEE Trans. Robot. Autom.*, vol. 8, no. 3, pp. 313–326, 1992. [Online]. Available: http://www.irisa.fr/lagadic/pdf/1992_itra_espiau.pdf, Accessed on: Mar. 30, 2017.

[19] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE Trans. Robot. Autom.*, vol. 12, no. 5, pp. 651–670, Aug. 1996.

[20] P. Corke, *Robotics, Vision and Control - Fundamental Algorithms in MATLAB*. Springer Berlin Heidelberg, 2011.

[21] Y. Ma, J. Košecká, and S. S. Sastry, "Vision guided navigation for a nonholonomic mobile robot," *IEEE Trans. Robot. Autom.*, vol. 15, no. 3, pp. 521–536, 1999.

[22] G. L. Mariottini, "Image-based Visual Servoing for Mobile Robots: the Multiple View Geometry Approach," Ph.D. dissertation, Università di Siena (Siena, Italy), 2006.

[23] G. L. Mariottini *et al.*, "Vision-based Localization and Control of Leader-Follower Formations," *IEEE Trans. Robot.*, vol. 25, no. 6, pp. 1431–1438, 2009.

[24] A. De Luca, G. Oriolo, and M. Vendittelli, "Control of Wheeled Mobile Robots: An Experimental Overview," *Lecture Notes in Control and Inform. Sci.*, vol. 270, pp. 181–226, 2001.

[25] A. De Luca, G. Oriolo, and P. R. Giordano, "On-Line Estimation of Feature Depth for Image-Based Visual Servoing Schemes," in *IEEE Int. Conf. Robotics and Automation*, Apr. 2007. [Online]. Available: https://www.dis.uniroma1.it/~labrob/pub/papers/ICRA07_depth.pdf, Accessed on: Mar. 30, 2017.

[26] D. N. Lee, "A theory of visual control of braking based on information about time-to-collision," *Perception*, vol. 5, no. 4, pp. 437–459, 1976. [Online]. Available: http://www.cog.brown.edu/courses/cg195/pdf_files/CG195Lee_1976.pdf, Accessed on: Mar. 30, 2017.

[27] D. N. Lee and H. Kalmus, "The Optic Flow Field: The Foundation of Vision [and Discussion]," *Philosoph. Trans. of the Roy. Soc. of London. Series B, Biological Sci.*, vol. 290, no. 1038, pp. 169–179, 1980.

[28] B. G. Bardy and W. H. Warren Jr, "Visual control of braking in goal-directed action and sport," *J. Sports Sci.*, vol. 15, no. 6, pp. 607–620, 1997. [Online]. Available: http://www.cog.brown.edu/research/ven_lab/Publications/BardyWarren_BrakingSport_JSS97.pdf, Accessed on: Mar. 30, 2017.

[29] N. A. Browning, "A neural circuit for robust time-to-contact estimation based on primate mst," *Neural Computation*, vol. 24, no. 11, pp. 2946–2963, Nov. 2012.

[30] B. R. Fajen, "Calibration, Information, and Control Strategies for Braking to Avoid a Collision," *J. Experimental Psychology: Human Perception and Performance*, vol. 31, no. 3, pp. 480–501, 2005. [Online]. Available: http://panda.cogsci.rpi.edu/resources/papers/Fajen2005a.pdf, Accessed on: Mar. 30, 2017.

[31] G. Savelsbergh, H. Whiting, and R. J. Bootsma, "Grasping tau." *J. Experimental Psychology: Human Perception and Performance*, vol. 17, no. 2, pp. 315–322, 1991. [Online]. Available: http://okazaki1.webs.com/ArtigosMestrado/25.Savelsberg1991.pdf, Accessed on: Mar. 30, 2017.

[32] G. J. Andersen and C. W. Sauer, "Optical Information for Car Following: The Driving by Visual Angle (DVA) Model," *Human Factors*, vol. 49, no. 5, pp. 878–896, 2007.

[33] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, 2nd ed. Pearson Education Limited, 2012.

[34] B. K. P. Horn, Y. Fang, and I. Masaki, "Time to contact relative to a planar surface," in *2007 IEEE Intell. Veh. Symp.*, Jun. 2007, pp. 68–74.

[35] MIT, "What is Duckietown?" 2016. [Online]. Available: http://duckietown.mit.edu/index.html, Accessed on: Feb. 25, 2017.

[36] L. Paull *et al.*, "Duckietown: an Open, Inexpensive and Flexible Platform for Autonomy Education and Research," *Int. Conf. Robotics and Automation*, 2017, submitted. [Online]. Available: https://drive.google.com/file/d/0B49dGT7ubfmSZFJmZEJwak5hcTQ/view?usp=sharing, Accessed on: Mar. 30, 2017.

[37] Raspberry Pi Foundation, "Raspberry Pi 2 Model B," 2017. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/, Accessed on: Feb. 25, 2017.

[38] Open Source Robotics Foundation, "Robot Operating System (ROS)," 2017. [Online]. Available: http://www.ros.org/, Accessed on: Feb. 25, 2017.

[39] Wason Technology, "Robot Raconteur," 2017. [Online]. Available: https://robotraconteur.com/, Accessed on: Feb. 25, 2017.

[40] Itseez, "Open Source Computer Vision (OpenCV)," 2017. [Online]. Available: http://opencv.org/, Accessed on: Feb. 25, 2017.

[41] NumPy developers, "NumPy," 2016. [Online]. Available: http://www.numpy.org/, Accessed on: Mar. 2, 2017.

APPENDIX A
DuckieRR – Duckiebot on Robot Raconteur

The effort to convert MIT's original Duckiebot implementation to an all Robot Raconteur based version is ongoing. The most recent version is available at `github.com/rpiRobotics/DuckieRR`.

A.1 Installation

The DuckieRR project is self contained, including it's dependencies, for easy installation on a Duckiebot. The github repository should be cloned to the Duckiebot before proceeding.

```
$ git clone https://github.com/rpiRobotics/DuckieRR
```

The entire project can be installed using the included `install.sh` file.

```
$ cd DuckieRR
$ bash install.sh
```

Note that this install will take some time, as it downloads and builds several different dependencies as well as the main code.

A.2 Running Services

Robot Raconteur services provide access to all robot peripherals and some additional functionality. Custom code can then be implemented as a Robot Raconteur client and connect to the services. The desired services must first be running on the robot.

All executables and python scripts for the services, once they have been built, are installed to the `build/bin` directory.

If the service is written in python it can be run as follows:

```
$ python build/bin/<ServiceName>Node.py [options]
```

If the service is written in C++ it can be run as follows:

```
$ ./build/bin/<ServiceName>Node [options]
```

`<ServiceName>` should be replaced with the name of the service, and any additional options can be passed at the end.

All services have a `--port PORT` option to specify the Robot Raconteur port number for an external TCP connection (`PORT` is replaced with an actual port number). If the option is not specified a port number is auto generated. The full connection URL is printed to the terminal once the service starts. Some services also have configurable settings and may be passed a YAML file using the `--config CONFIGFILE` option.

The specific options available ultimately depends on the service. To see all available options, the service usage can be printed using the `-h` or `--help` options.

A.3 Running Clients

All clients currently implemented are available in the `clients` folder. All services that the client connects to must already be running on the robot.

There is a simple example client `DuckieClient` that has been implemented in both Python and Matlab. We will not discuss the details of running Robot Raconteur clients on Matlab, for that the reader is referred to the Robot Raconteur Documentation [39]. This client demonstrates basic motion, as well as capturing images from the Camera service.

Any client written in Python is run in essentially the same manner as services

```
$ python clients/<ClientName>.py [options]
```

where the available options are specific to the client.

A.4 Writing New Clients

Clients can be written in any language supported by Robot Raconteur. We suggest placing the code inside the `clients` folder, however this is not required in general. The sample clients listed in the previous section show how to connect to services, and are therefore a good reference. For more information on Robot

Raconteur clients in general we again refer the reader to the Robot Raconteur Documentation [39].

If writing C++ clients, the client will require 'Thunk' code for each service so that Robot Raconteur knows how to interface with it. The C++ Thunk code for all services has already been generated and is available in `dependencies/DuckiebotRobdef/CPP_Thunk`. The thunk code is also installed to the system as a library so that it can directly be included and linked to if desired. The naming convention for these libraries is `Duckiebot__<Service>`. Note that some services that depend on commonly used types may also require the `Duckiebot` library.

A.5 Example: DriveByAngle Client

The Robot Raconteur client that implements the algorithm discussed in this work is available in the `clients/DriveByAngle` folder. It requires both the *Drive* service and the *Camera* service so both must be launched on the robot. Assuming you are already connected to the robot and have access to it's terminal you would therefore launch the services as follows:

Change to the DuckieRR folder if you have not already

```
$ cd DuckieRR
```

then, in one terminal run

```
$ python build/bin/DriveNode.py
```

and in another terminal run

```
$ ./build/bin/CameraNode
```

Note that both will also generate a TCP connection and therefore allow a `--port` option. This may be useful if you wish to connect to these clients from another computer over a TCP connection, and would like to specify the port number. However since the client will be run locally the TCP connection won't be used in our example.

We then start the client in a third terminal as follows:

```
$ python clients/DriveByAngle/DriveByAngle_onDuckie.py
```

You may note that several other clients are also included in this folder. They implement essentially the same algorithm however with additional debugging options (i.e. visualizing the camera output or turning the wheels on/off). The `_onDuckie` files have removed all of this bloat and simply runs the program.

If the detection algorithm has been recalibrated, or different controller settings are required a YAML config file can also be passed to the program to load these settings rather than the defaults. For example

```
$ python clients/DriveByAngle/DriveByAngle_onDuckie.py
   --config default.yaml
```

loads the default configuration (this actual command is of course redundant since the program loads this file by default).

The program will start by running the initial detection routine. The Duckiebot should therefore be placed at the desired distance from the lead vehicle (with the tag). If a tag is detected, the coordinates of its vertices are printed to the screen. To finish the initial detection routine press `Enter` and the main controller will start.

The tag can now be moved around in the environment and the Duckiebot will follow! The program can be quit by simply pressing `Enter`.

If the tag is lost at any time an error will be printed to the screen detailing how many loops the tag has been lost for. If the Duckiebot is consistently losing the tag, the detection algorithm may need to be recalibrated. If the tag is lost for more than some specified number of loops (50 by default), the program will automatically exit.