ANALYZING THE EFFECTS OF GEOMETRIC LANE CONSTRAINTS ON RADAR-BASED SENSING OF AVAILABLE VEHICLE HEADWAY USING MAPPED LANE GEOMETRY AND CAMERA REGISTRATION OF LANE POSITION

A Thesis in
Mechanical Engineering
by
Krishna Prasad Varadarajan Srinivasan

© 2019 Krishna Prasad Varadarajan Srinivasan

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2019
The thesis of Krishna Prasad Varadarajan Srinivasan was reviewed and approved by the following:

Sean N. Brennan  
Professor of Mechanical Engineering  
Thesis Adviser

Hosam K. Fathy  
Bryant Early Career Professor of Mechanical Engineering

Karen A. Thole  
Distinguished Professor of Mechanical Engineering  
Head of the Department of Mechanical Engineering

*Signatures are on file in the Graduate School.
Abstract

Commercial trucks are currently equipped with a single front-facing RADAR mounted on the front bumper, as this is a sensor useful for Cooperative Adaptive Cruise Control, Emergency Braking Systems, and many similar Connected and Automated vehicle functions that require longitudinal vehicle control. This paper investigates the use of a bumper-mounted RADAR to perform traffic characterization around the ego-vehicle, with a particular goal to obtain an estimate of the furthest headway that can be considered as a reliable estimate of open maneuvering space, such that there are no vehicles within the same lane as the ego-vehicle, i.e. the ego-lane. This available headway in front of a vehicle is an important parameter in an ongoing study whose goal is to obtain improvements in fuel economy for highway driving of a tractor-trailer. But headway availability depends on the correct attribution of RADAR-measured vehicles to be either within the ego-lane, or outside the lane. The attribution of lane designations to specific RADAR targets depend strongly on lane geometry and the ability to align RADAR measurements to the ego-lane. This work investigates how knowledge of lane geometries, as well as sensor performance characteristics, may improve the trust in a RADAR measurement of open headway distance in front of a vehicle.

Specifically, several strategies for associating local traffic either within or excluded from the ego lane are considered, and the possible sources of error in headway calculations are investigated for each strategy. The strategies differ by the availability of lane geometry information and RADAR target association to lane constraints, which in turn is assumed to be supplied by the use of map data and secondary camera measurements. It is observed that combining RADAR with lane-geometry maps help detect headway up to and beyond RADAR range limit with exceptions tied to a particular road. These errors can further be minimized by using camera measurements subject to accuracies in registering lane markers. The results of analyzing highway routes reveals that: RADAR data without lane geometry strongly limits the trusted headway measurement from this sensor; that lane geometry maps can greatly increase the trusted range, usually beyond the RADAR’s range limits; and that image-based registration of lane markers to lane geometry should have accuracies allowing full-range trust in the RADAR target measurements of headway.
Table of Contents

List of Figures vi
List of Symbols vii
Acknowledgments viii

Chapter 1
Introduction 1
  1.1 Motivation .................................................. 1
  1.2 Research Goals ............................................... 2
  1.3 Thesis Outline ............................................... 3

Chapter 2
Literature Review 5

Chapter 3
Data collection platform 9

Chapter 4
Methods to calculate geometric limits on headway ranges 12
  4.1 Method 1: Conventional RADAR sensing ..................... 13
  4.2 Method 2: Map-based RADAR ................................ 17
  4.3 Method 3: Map-based RADAR and Camera Fusion .......... 21
  4.4 Comparison of methods ...................................... 22

Chapter 5
Conclusions 24

Appendix A
Script for headway estimation 25

Appendix B
Script to plot camera distance to pixel error 36
Appendix C
Script to calculate yaw-angle orientation errors using different methods 40

Appendix D
Helper classes used in the main scripts 45

Appendix E
Script to illustrate ambiguous headway arising due to misorientation of ROI and the actual lane 87

Bibliography 90
List of Figures

1.1 The maximum distance available for correct headway measurements depends strongly on the overlap between the ROI and actual lane, as well as the geometry of the vehicle to be detected. ......................................................... 3

3.1 Instrumented Volvo VNL 300 truck used for this research work ................. 10
3.2 Figure depicting the specifications of the Delphi RADAR unit used in this work . 11

4.1 Flowchart depicting the assumptions made in processing the sensor information for headway calculations in the three strategies listed above ......................... 13
4.2 Example headway calculation showing point where lane geometry errors would have allowed a 1.5 meter wide vehicle within the lane to be missed. .............. 15
4.3 Headway ROI extent calculation using Method 1 for a sample route, for three different target vehicle dimensions ................................................................. 16
4.4 The histogram of headways calculated via Method 1 for a highway route in the Pennsylvania region, assuming three different types of target vehicles. ........ 17
4.5 Calculated yaw-angle from 4 repeated traversals of a roadway showing significant repeatability. In one traversal, a lane change maneuver is clearly visible. .... 19
4.6 The histogram of standard deviation of yaw angle, binned over route stations.................. 20
4.7 Headway calculated using method 2, using various estimates of ego-vehicle yaw-angle. ................................................................. 21
4.8 The distance versus pixel lateral extent for a 10cm lane marker .................... 22
4.9 The distance versus pixel lateral extent for a 10cm lane marker .................... 23
List of Symbols

$P_i$  $i^{th}$ point
$P_{i-1}$  $(i - 1)^{th}$ point
$X_i$  $x$-coordinate of $i^{th}$ point
$Y_i$  $y$-coordinate of $i^{th}$ point
$W$  Vehicle width
$N$  Number of points
$\psi$  Yaw-angle
$\Delta \psi$  Yaw-angle orientation error
$H_{\text{max}}$  Maximum headway
$T$  Traversal number
$s$  Station coordinate
$\psi(s)$  Station averaged yaw-angle
$e_{\psi}$  Yaw-orientation error
$\sigma$  Standard deviation
The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000801. The findings here are those of the author and do not represent the views of the sponsor, Penn State, nor any other entity.

I would like to acknowledge the contribution of the Intelligent Vehicles and Systems Group (IVSG), particularly the members of the NEXTCAR-IVSG sub-group at Penn State for creating a thriving research environment. I would like to acknowledge the support and help of Robert Leary, Mohamaed Wahba, and Evan Pelletier in helping me complete this graduate research work. The camaraderie and the enthusiasm shared by the group members helped me further my knowledge in the field as well as further fuelled my ambition to establish a career in the field of robotics, and connected and autonomous vehicles.

I would like to thank Dr. Sean Brennan for his continuous support throughout my Master’s journey at the university. His contribution to my understanding of the research field can only be superseded by his tremendous support in my personal life.

I would like to express my gratitude towards the Department of Mechanical and Nuclear Engineering at the Pennsylvania State University for an amazing environment and providing me with teaching assistantships during my Master’s program. This enabled me that I could pursue my research and academics without worrying about the finances.

Last but not the least, I want to convey my sincere gratitude to the drivers, technicians, and the staff at the LTI Test Track for helping me with the experiments performed on the test track as part of this thesis.
Dedication

To my parents for their blessings and for everything that I am able to achieve.

To my brother and little sisters for all the love.

To my research advisor, Dr. Sean Brennan and the ‘Intelligent Systems and Vehicles Group’ for all the guidance, discussions, and for making me a better and a more informed engineer.

To dreams, belief, and perseverance.
Chapter 1

Introduction

1.1 Motivation

In the field of Connected and Autonomous Vehicles (CAV), the measurement of surrounding traffic is important for predicting key vehicular behaviors. There are many factors showing a need for estimating local traffic, particularly traffic ahead of a vehicle that would affect maneuverability options. For heavy trucks, the fuel used in operation is one of the major operating costs incurred by a vehicle in its lifetime. Characterizing and estimating the traffic around the vehicle, and using that information to plan maneuver trajectories and avoiding aggressive driving can result in fuel savings [1], [2], [3]. Additionally, traffic and headway information is useful for route planning: It has been shown that for every 1% increase in trip time, the fuel consumption increases by 1.1% [4], and the primary factor which governs the choice of a route is the traffic condition along a given route [5]. Hence, characterizing the available free space in front of the vehicle, or headway, has the potential for major fuel optimization.

The characterization of headway for a vehicle depends particularly on the open space available in front of the vehicle of interest (the “ego-vehicle”), but in particular the space within the ego-vehicle's lane ahead. Because a vehicle's inference of the detection region of interest (ROI) may not align with the actual lane geometry, it is possible that vehicles that should be included within the ROI for a headway calculation may be incorrectly omitted. This situation is illustrated in Figure 1.1 which shows a situation where the RADAR headway estimation algorithm assumes a straight lane ahead to define the ROI, and yet the actual lane deviates due to curvature. This discrepancy illustrates that the headway estimation, in the presence of lane definition errors, is limited to a particular distance.

It may seem that all targets seen by a RADAR system should generally be tracked by commercial-grade sensors, and for the commercial-grade RADAR sensor used in this work, the
Delphi ESR 2.5 RADAR, this is the case. However, even when targets are seen, the lane in which these targets are operating may still be unclear, especially in the common situation for commercial trucks where there are vehicles in the passing lane, and the truck is in the slow lane. In the case, for example, where a lane is veering to the right and there is a vehicle ahead in the passing lane, it can become very unclear from RADAR data alone whether the vehicle ahead is entering the ego-lane, or simply remaining within the passing lane. Thus, designating the lane in front of the ego-vehicle is crucial to avoid misclassification of vehicles ahead of the ego vehicle in the ego-vehicle's lane. From the RADAR measurements, the bearing angle and range of the detected targets can be used to resolve for their positions with respect to the ego-vehicle. But, aggregation of different sources of error discussed in this work show that, even with such tracking, there may still be ambiguous headway measurements. Hence, resolving the detected targets from the RADAR to the lane designations of the ego-vehicle helps in relieving this ambiguity in headway availability.

1.2 Research Goals

The goal of this work is to define this distance limitation in headway calculations, based on the overlap between the ROI and the actual lane – at some distance the mismatch of overlap between the ROI and the actual lane could allow vehicles to be missed, resulting in an incorrect estimation of the available headway. This maximum range of trusted headway is strongly a function of road and vehicle geometry, and is subject to common errors including lane curvature, vehicle orientation within the lane, and registration of the visible lane to inferred lane ROI. To produce larger ranges of trusted headway availability, the reduction of these errors are analyzed respectively by considering the inclusion of a lane geometry map, analysis of vehicle orientation errors in vehicle yaw measurement, and the use of a camera system to match lane markings to the ROI. This analysis assumes that the the radar acts as a perfect detection system, and thus error analysis does not include artifacts of the RADAR itself; further, this work assumes that registration errors are known through prior calibration, and thus the focus of the thesis is on the impacts of these errors rather than the map-to-image or RADAR-to-image data fusion processes, which is an active topic of other researchers, as discussed in the next section.
Figure 1.1: The maximum distance available for correct headway measurements depends strongly on the overlap between the ROI and actual lane, as well as the geometry of the vehicle to be detected.

In addition to the space available in front of a vehicle, the headway available depends on the density of nearby traffic, and the occurrence of lane cut-ins and cut-outs. Initial work to detect headway focused on tracking individual vehicle targets from the ego-vehicle; however, for normal highway driving, the uncertainty in the lane definition ROI greatly increased the difficulty in determining if a vehicle was actually cutting-in or departing the ego lane, or if the ego-lane itself was veering ahead of the vehicle. Thus, appropriate definition of the ego-lane ROI can be seen as a necessary precursor in headway calculations, to thereby infer surrounding traffic cut-ins and lane departures.

For commercial trucks, radar systems are currently available on many models for Adaptive Cruise control and for safety systems/warnings; therefore, the primary goal of this work is to understand if traffic characterization can be performed on a mobile truck by only using one front-facing RADAR, and the advantage of using camera and map systems to increase the utility of RADAR data. The focus of this work is driven by the fact that, for modern CAVs, vision [6], RADAR [7], and LiDAR sensor systems [8] generally constitute the sensor suite of a vehicle’s perception module.

### 1.3 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 discusses the state-of-the-art in the field of traffic estimation, focusing particularly on RADAR-based headway estimation between
the ego-vehicle and the vehicles in the ego lane. Data collection platform and sensors used in this research work are discussed Chapter-3. This is followed by a description of the three strategies in which the headway estimation errors are analyzed in this work in Chapter-4, providing mathematical expressions for error calculation in each. This is then followed by a section examining an experiment-driven analysis of headway estimation strategies. A discussion follows then analyzing the contributions, comparing the resulting headway ranges across strategies. Finally, conclusions are presented in Chapter 5, summarizing the main contributions of the thesis.
Chapter 2

Literature Review

RADAR is preferred for headway measurement because of its unique advantages relative to other sensors. A RADAR uses the Doppler effect to directly measure velocity of vehicle, unlike other sensors, which use the difference between two measurements to determine the vehicle speed. As discussed by Kocić, et al., [9], this plays a crucial role in fusion algorithms, as vehicle velocity obtained using RADAR is measured as an independent parameter, which helps in enhancing convergence rates when fused with measurements from other sensors. Long-range RADAR sensors can usually measure up to a distance of 200 meters which enables the ego-vehicle to detect headway in a larger preview horizon. Compared to other long-range sensors such as LIDAR and camera-based systems, RADARs are affected the least in inclement weather conditions such as fog or poor lighting conditions. Fritsche, et al., [10] also demonstrate this where enhancement in localization ability is achieved by leveraging the detection capability of RADARs in smoky environments, for example on a mobile robot, fusing information from RADAR and LiDAR to navigate a smoky room. RADAR sensors continue to be improved for automotive applications, including for example the development of the 77 GHz. RADAR [11]. As noted by Bloecher and Dickmann [12], RADAR sensing is an integral part of ADAS (Advanced Driver Assistance Systems) and CAV applications, and in safety-critical applications of autonomous cars; this same work, however, notes interference challenges of RADAR signals, and and mitigation strategies while being deployed on a vehicle on road.

There are many examples of research that utilize RADAR for CAV applications. RADAR forms an integral part of a vehicle’s perception sensor module for functionalities such as emergency braking, blind spot detection, and vehicle following. For example, research by authors with DaimlerChrysler AG used RADAR to develop the adaptive brake assistance functionalities as an improvement over the conventional brake assistance system [13]. RADAR is widely used in the Adaptive Cruise Control (ACC) systems deployed in CAV applications, including collision avoidance and ACC [14]. RADARs can further be used for implementing auto-braking systems; for
example, the work by Kim, et al., [15] proposes a design and implementation of an auto-braking system for pre-crash safety meant for using RADAR on deployment in vehicles. RADARs have also been used to understand driving behaviors: Andrej Ivanco at Clemson University [16] performed a study where headway is used in the bid to understand society’s reluctance towards the adoption of autonomous cars. He performed a data analysis using RADAR data collected from SHRP2 [17] of around 3800 trips from 39 vehicles, validating a positive linear relationship between headway distance and vehicle speed.

There are many different ways to measure headway and local traffic, including the fusion of RADAR data with other sensor sources. In 1992, Ito, et al., presented methods to combine stereo and optical flow lines to obtain headway measurements [18]. Suzuki and Nakatsuji present a method to use particle filters and dual filters on the acceleration rate and velocity data measurements obtained using a probe vehicle in a six-vehicle platoon to estimate headway [19]. Gehrig and Stein [20] use only a vision-based camera to detect objects with the motivation of collision avoidance. LiDARs can also be used to obtain headway measurements, including work by Benalie, et al., where a laser scanner mounted on the front of the vehicle is combined with a fuzzy controller to obtain headway and velocity measurements [21]. Measurements from RADAR have been used to obtain information about the shape of the road, as shown by Ma, et al., [22]. Lee, et al., combine RADAR measurements with longitudinal dynamic model of the vehicle to estimate the curvature of the road [23]. RADAR and camera measurements have been commonly fused to define a ROI, and obtain headway and other properties of target vehicles [24], [25], [26]. Fusing RADAR data with other sensor modules for measuring local traffic not only enhances measurement quality due to leveraging properties of other sensors, but it also facilitates the use of simpler and lower-cost RADAR technologies, as mentioned by Sole, et al., [26]. Millela, et al., combine RADAR with vision cameras to perform ground segmentation in their work detailed in [27]. Millela and Reina also present fusing multiple sensors including RADARs, to detect obstacles, targets, and free space availability for perception for autonomous driving [28].

There are a smaller number of research publications that address headway estimation in particular for CACC (Cooperative Adaptive Cruise Control) or similar deployments. In one such work, Ge and Orosz present a method to use headway and velocity obtained from GPS and DSRC radios to propose an optimal connected cruise controller [29]. Use of headway in (C)ACC applications further has motivated researchers to analyze the safety critical headway for collision avoidance. The work by Kester, et al., is one such example where they measure critical headway in the event of non-ideal vehicle-to-vehicle communication conditions, such as emergency breaking [30]. Finally, headway is also used to maintain the inter-vehicle platoon spacing. Ko and Chang discuss a method to use the turn signals of the vehicle and combine it with the vehicle headway to detect lane cut-ins in a platoon [31]. On the other hand, Harfouch, et al., use headway to propose a CACC method to overcome the vehicular spacing homogeneity assumption in platoons [32].
The work in this thesis is motivated by algorithms that seek fuel-efficient longitudinal control, and integral to this is headway measurement which constrains the control decisions available. The importance of headway as a parameter in control strategies for fuel efficiency is demonstrated by the work of Luo, et al., [33]. In this work, an NMPC (Nonlinear Model Predictive Control) framework is used to propose a novel ACC framework for hybrid vehicles which demonstrates an enhanced fuel efficiency while providing improved coordination between fuel efficiency and tracking safety versus conventional ACC. The contribution of headway in the bigger picture of optimizing fuel economy is further highlighted by the Li, et al., who use MPC (Model Predictive Control) to execute vehicle-following decisions with the objective to enhance fuel economy and reduce tracking errors [34]. More than 5% reduction in fuel cost is observed by replacing an LQ control algorithm with MPC by Ruina, et al., while deploying ACC on an electric vehicle [35]. MPC algorithms are also used in other research work in reducing fuel cost. By exploiting the potential of introduction of CAVs, Xu and Peng show that 7% reduction in fuel cost is obtained when using MPC and EKFC (Equivalent Kinetic energy and Fuel Conversion) algorithms for vehicle speed optimization compared to a constant-speed cruise control system in their test scenarios [36]. Qu, et al., utilized a moving horizon control scheme to model longitudinal and lateral dynamics to follow vehicle trajectory and consider the fuel economy. [37]. Further, work by Li, et al., proposes a multi-objective ACC scheme incorporating fuel economy, tracking ability, and the desired driver response for to show reduction in fuel consumption compared to traditional ACC [38]. These examples demonstrate the use of headway, either in terms of time or distance, to formulate the expression of longitudinal dynamics of the vehicle and in turn optimize fuel economy.

There is a need to characterize RADAR information in consideration of more ubiquitous lane maps. Maps play an integral role in CAV applications both for localization purposes and as a look-up feature for performing parameter estimation in the desired preview horizon. Easy access to maps either, through open-source platforms (such as Open Street Maps), or pre-registration through an on-board (D)GPS sensor means that valuable feature information about the upcoming global environment (such as lane width, curvature, and road shape) can be fused with other measurements. One such work is presented by Kim, et al., where road curb information from LiDAR point clouds is fused with a global map obtained from DGPS for localization of the vehicle [39]. Similarly, there are other examples in the literature that fuse information from LiDAR and maps [40]. Similar fusion of mapped features to RADAR detection is not as common in the literature. In the research of Dickmann, et al., enhancements in vehicle perception capabilities are demonstrated on a Mercedes Benz S-class research vehicle equipped with digital maps along with other sensors [41]. Information about lane markers can also be obtained using an on-board camera systems which provide useful information about relating detected object information with respect to the ego vehicle’s lane as demonstrated throughout the literature [42], [43], [44]. Combining RADAR with lane maps help counter errors generally associated with RADAR
measurements.

There are several ways to characterize errors in RADAR. External factors affecting the performance of an automotive RADAR include factors such as signal interference with the environment or discontinuity in the power supply as described in [45], [46], and [47]. The primary measurement errors, which are intrinsic to the RADAR, include errors in range, bearing angle, angular velocities, and measurements obtained from combining the primary measurements, such as position of the detected object. These errors affect the object detection capability of the RADAR, and subsequently affect collision avoidance, (C)ACC, and other features in CAV applications. Herzel and Kissinger describe the root cause of errors in angular velocity measurements and propose a method to overcome these errors, specifically in the scenario of collision avoidance [48]. Error in bearing angle measurements can be intrinsic and could be worsened by poor sensor mounting. Suzuki et al., discuss error in measuring the angle from a RADAR mounted to the bumper of the vehicle, and propose a real-time correction algorithm [49]. Another major drawback in using RADAR as a stand-alone sensor for perception is that a RADAR returns multiple data points per detected object in most scenarios. This is acknowledged in the work of Föllster and Rohling where they address this problem using data association between different measurements [50]. Multiple RADARs are also deployed in many vehicles instrumented in CAV applications. Hence, errors can also occur due to interference between these RADARs, as analyzed by Bechtler, et al., [51]. Finally, Stanislas and Peynot in characterized the Delphi ESR 2.5 RADAR, which is used in this work, and tabulated a comparison of various measurement errors between the specifications provided by the manufacturer and their experiments [52]. However, their experiments are intended for close-range detection driven by their application domain (ground robotics).
Chapter 3

Data collection platform

In this research, a Volvo VNL 300 truck, seen in Figure 3.1, was instrumented with sensors and logging devices for collecting data from various systems on the truck. This figure also depicts the instrumentation capabilities used in this study for external measurements; inertial measurements of yaw rate are available through the vehicle’s factory-integrated measurements on the CAN bus. Sensor data was logged directly from the vehicle’s CAN bus, through serial communication into the Robot Operating System, or by direct measurement to DAQ cards using a Simulink Real-Time target PC (a SpeedGoat, specifically). For perceiving the environment around the truck, a front-facing RADAR was mounted behind the bumper of the truck, using a Delphi ESR 2.5 RADAR unit which is capable of identifying up to 64 targets using simultaneous short-range and long-range detection. The mounting location matched commercial RADAR unit placements for this vehicle. The characteristics of the RADAR are depicted in Figure 3.2. As depicted in the figure, the RADAR is capable of performing simultaneous short- and long-range detection. The field of view and maximum range for short-range detection is $\pm 45^\circ$ up to 60 meters and that for long-range detection is $\pm 10^\circ$ up to 174 meters. The RADAR data and the GPS data were logged through the Robot Operating System (ROS); GPS data was collected using a Hemisphere A325 unit at a frequency of 20Hz $(0.05\text{secs.})$. 
The default data-logging frequency of the RADAR is $> 400\text{Hz}$. However, this data logging and publishing rate was found to be excessive for the purpose of this work, simply because the RADAR publishes the data even if only one target is seen within a time-stamp. In order to log the data in a more efficient manner, a ROS package was written to publish the data once 64 targets are identified by the RADAR. This resulted in a new data-logging frequency of $25\text{Hz} \cdot (0.04\text{secs.})$. Further, the target’s range in meters ($m$), range-rate in meters per second ($m/s$), bearing-angle (in degrees, with counter-clock-wise direction being positive), and track-validity in boolean (1 or 0) were published from the available topics from Delphi. Since, the vehicle and traffic’s longitudinal dynamics (particularly headway) are dynamically much slower as compared to the RADAR’s publishing frequency, even with this modification, the data collected by the RADAR was found to be largely unaffected by this modification.
Figure 3.2: Figure depicting the specifications of the Delphi RADAR unit used in this work
Methods to calculate geometric limits on headway ranges

The calculation of the geometric conditions for the maximum possible open headway available depends heavily on the nature of sensor information available. Strategies for the deployment of a headway algorithm can be developed based on different combinations of the above sensor information. Three methods are considered in this work:

1. **Method 1: Conventional RADAR Sensing** Use the RADAR by itself, calculating headway assuming that the lane ahead projects straight from the ego-vehicle ahead. In this case, headway accuracy is most limited by the straight ego-lane assumption and thus will depend strongly on the local road curvature actually experienced by the vehicle.

2. **Method 2: Map-based RADAR** Using the RADAR and GPS with a possible yaw sensor estimate. This method corrects the forthcoming lane representation by using GPS position to index a route database of lane geometry (i.e., a map). For this method, headway accuracy is most limited by the orientation error of the vehicle relative to the lane projection. If a yaw estimate is available, the projection is further corrected to better align the vehicle to the map.

3. **Method 3: Map-based RADAR and Camera Fusion** Using the RADAR, this method assumes that both map and camera information are available. Like Method 2, the GPS or similar position measurements are used to index a map database, but as an additional correction, the front-facing camera image is registered to the map geometry. In this process, the headway accuracy is most limited by the errors in camera-to-map and camera-to-radar calibrations, and this error is assumed to be known from prior work, and fixed throughout the route.
Assumptions made in processing the sensor information for headway calculations in all the three methods are described in the flowchart in Figure 4.1.

![Flowchart](image)

Figure 4.1: Flowchart depicting the assumptions made in processing the sensor information for headway calculations in the three strategies listed above

The strategies are illustrated and discussed in detail in the sub-sections below.

### 4.1 Method 1: Conventional RADAR sensing

The first method assumes that the lane geometry in front of the vehicle is unknown. When using a RADAR unit without other information sources, it is difficult to know the geometry of the lane in front of the ego-vehicle. To infer whether or not RADAR targets are within the ego-lane, the assumption is usually made that the road ahead is a straight road. This is illustrated in Fig. 1.1. Additionally, the road is assumed to be flat, the RADAR is assumed to be operating with perfect accuracy and responsiveness, and the road is assumed to be within the RADAR’s operational area of coverage.

Under these assumptions, the headway is limited by the minimum of either the range of the radar, or the location at which the deviation in the actual road curvature away from a straight lane assumption is equal to the width of a vehicle, as shown in Fig. 1.1. If a vehicle’s route is known a-priori, as is a common case for commercial trucks, the maximum headway available geometrically can be approximately calculated by comparing the straight road assumption to the
actual profile of the road ahead.

In this study, this headway calculation was performed by using the current and prior GPS positions of the vehicle to project a straight-lane assumption into the true lane path of the vehicle, as recorded in post-processing of the vehicle’s route. First, a line projection based on the $i$th and $(i-1)$ route points is created in world coordinates, where each point $P_i = (X_i, Y_i)$:

$$l(P_i, P_{i-1}) : Y = \frac{Y_i - Y_{i-1}}{X_i - X_{i-1}} \cdot X + Y_i$$  \hspace{2cm} (4.1)

The route is then examined for all points $N$ such that $N > i$ to find the first point that deviates from this projection more than a vehicle width, $W$. Specifically,

$$N_{max} = \min_{N > i} \text{dist}(P_N, l(P_i, P_{i-1})) > W$$  \hspace{2cm} (4.2)

where $\text{dist}(P_N, l(P_i, P_{i-1}))$ represents the distance from the line projection to the test point $P_N$.

An example of this calculation performed over a sample route is shown in Fig. 4.2. At the point labeled “Max Headway”, a vehicle that is actually within the ego-lane, for a vehicle 1.5 meters wide, could be mis-classified even if detected by radar, simply due to misorientation of the ROI with the lane. Thus, this maximum headway point, based on the road geometry, would be the maximum value of headway that should be trusted as open, assuming that no obstacle detection occurs via the RADAR sensor to this point.
Figure 4.2: Example headway calculation showing point where lane geometry errors would have allowed a 1.5 meter wide vehicle within the lane to be missed.

This process can be repeated along a vehicle’s route, to obtain estimates of how headway varies during a typical highway drive, under different assumptions of the size of potential leading vehicles. In this work, a motorcycle is assumed to be 1 meter wide, a sedan 1.6 meters, and a tractor trailer to be 2.6 meters. Fig. 4.3 shows the resulting headway calculated for these three types of vehicles using method 1 versus the station coordinates, where station coordinates represent how far the vehicle has traveled since the start of a route. The results here are typical of those found from highway testing of this algorithm; they show, for example, that it is fairly routine that the headway limited by the geometric curvature of a highway lane will limit the trusted ROI of a RADAR system to far less than the RADAR’s range reported by manufacturers.
Figure 4.3: Headway ROI extent calculation using Method 1 for a sample route, for three different target vehicle dimensions

Fig. 4.4 shows a histogram of the calculated headway using this method, for the three different vehicles mentioned earlier, binned across an hour-long drive that is used for cycle testing truck technologies (Note: this cycle location is proprietary and is intentionally not shown). The histogram reveals that, for a typical sedan being a lead vehicle, the most common largest headway that should be utilized is approximately 60 meters (vertical dotted line), which is approximately 1/3 the actual available range of the RADAR. This result reveals an important motivation for this work: that the data interpretation of long-range sensors such as RADAR, LIDAR, and camera systems have a clear need to associate their detected targets as members of the vehicle’s ego-lane. Without such association, the useful range of the sensor would be far less than the detection range.
Fig. 4.4 also illustrates how, in long-range target sensing, the road geometry information may become more important than the target size. Examining the vertical solid bar in this histogram, the bins to the left – which represent near-range targets – have occurrence frequencies that depend strongly on the target size. However, bins to the right – representing long-range targets – show a distribution that depends primarily on headway alone. An explanation for this is that, at least in the routes that the authors have tested in the North Atlantic region, it appears that headway depends primarily on road curvature. In other words, roads that are straight and long enough to allow large headways commonly connect to curves ending these segments, with curvature large enough to make it difficult to discriminate any lead vehicles within the lane thereafter.

### 4.2 Method 2: Map-based RADAR

The second method assumes that the lane geometry in front of the vehicle is known from a map database, but that the map is not registered to camera information; in this case, the in-lane headway detection limits are limited primarily by the point at which the yaw orientation error causes a deviation away from the assumed lane. Once this deviation is at least a vehicle’s width
away from the true lane position, then a vehicle within the ego-lane could be misclassified as being in the wrong lane. Thus, the error in headway availability is limited by the range where the yaw-angle errors would introduce at least a vehicle’s width of error. Assuming a yaw-angle orientation error of $\Delta \psi$, the maximum headway would be given by:

$$H_{\text{max}} = \frac{W}{\Delta \psi}$$  \hspace{1cm} (4.3)

A common method of estimating absolute yaw angle in an Earth-fixed coordinate system is to use sequential GPS position measurements to obtain a global yaw orientation, a method used here to evaluate pointing accuracy. Assuming the vehicles yaw angle, $\psi$, is approximated as:

$$\psi = \text{atan} \left( \frac{Y_i - Y_{i-1}}{X_i - X_{i-1}} \right)$$  \hspace{1cm} (4.4)

where $(X_i, Y_i)$ denotes the global XY position in ENU coordinates, obtained via a GPS measurement. In Fig. 4.5, the calculated yaw angle for a testing route is plotted showing 4 traversals. One can observe that the yaw angle of the vehicle will vary even while staying within a lane, but that this variation is small in magnitude, it will depend on lane change events - one of which is shown in this same figure, and it may change by station position, i.e. $e_{\psi1} = \psi(s, T) = \psi(s) - \psi(T)$, where $s$ is the station coordinate, and $T$ is the traversal number, $\psi(s)$ is the station-averaged yaw angle, and $\psi(T)$ is the yaw angle measured during traversal $T$.

Another means of approximating the vehicle’s orientation yaw error within a lane is to assume yaw error is represented by some multiple of the variance in yaw measurements observed across repeated traversals of the lane. Specifically, $e_{\psi2} = K \cdot \sigma_{\psi(s,T)}$, with a typical value of $K$ being 2 to represent 2 standard deviations. In the experiments performed for this study, the replicated yaw angle measurement for at least four traversals at each station coordinate within the route enables the calculation of this per-station variance error, $e_{\psi2}$. In Fig. 4.6, the histogram of this variance across the entire route is shown.

Finally, one can approximate the yaw orientation error using the aggregated yaw error statistics over the entire route. In this case, the error is simply the $2\sigma$ variance bound calculated from aggregating all yaw deviations across all traversals, across the entire route, $e_{\psi3} = 2 \cdot \sigma_{\psi(\forall s, \forall T)}$. This calculation’s result is also shown in Fig. 4.6, where one observes that the $2 - \sigma$ yaw-angle deviation within a lane averages 0.43 degrees throughout all traversals of the route.
Figure 4.5: Calculated yaw-angle from 4 repeated traversals of a roadway showing significant repeatability. In one traversal, a lane change maneuver is clearly visible.
Figure 4.6: The histogram of standard deviation of yaw angle, binned over route stations.

Each of the three methods of calculating yaw orientation errors – $e_{\psi_1}$, $e_{\psi_2}$, or $e_{\psi_3}$ – were used with Eq. 4.3 to calculate their associated headways; plots of these results are shown in Fig. 4.7, along with the range limit of the RADAR sensor commercially available on the research truck. One observes that the in-lane yaw error method, $e_{\psi_1}$ routinely gives headways far less than the RADAR’s full range. The per-station standard deviation, $e_{\psi_2}$, and the averaged variance across all deviations, $e_{\psi_3}$, both predict headway ranges well above the maximum sighting range of the RADAR, and thus both would appear to not usually limit RADAR performance. However, the data shown here – which only includes highway-speed driving segments – reveals that unless one assumes statistical properties for yaw error that are aggregated across an entire route, the headway metrics that are a function of station, namely $e_{\psi_1}$ and $e_{\psi_2}$, both indicate that there are routinely locations on highway-speed lanes where the geometry of the lane ahead of the vehicle effectively nullifies the long-range capabilities of a RADAR sensor, and in some real-world cases suggesting that headway availability should only be trusted to as little as 50 meters. This is significant as NHTSA braking standards require a truck to stop within 76 meters (250 feet) for trucks at a speed of 60 mph. Thus, RADAR systems that are unable to designate obstacles as within-lane in under 76 meters may either be required to react to near-lane obstacles that are otherwise safe, or might, in some road geometry conditions, require other means such as camera systems to detect lane designations. These plots show that such conditions are rare, but could
4.3 Method 3: Map-based RADAR and Camera Fusion

The third method assumes that the camera system is able to register map information and RADAR targets to the lane. Thus, the primary source of lane-tracking error is due to this registration error, and thus is able to correct for in-lane yaw orientation errors. Because the camera calibration would produce a yaw-offset error, Eq. 4.3 can again used in this method, except with the camera’s lane-tracking error as the source of yaw-angle uncertainty.

One can estimate an imaging system’s uncertainty in angular measurement by recognizing that the yaw-angle certainty in orientation lock improves as features are used further toward the horizon. If one examines the relationship between pixel extent and the width of a lane marker for a typical HD lane-tracking camera – as is used in the test vehicle of this study – one obtains the plot as shown in Fig. 4.8. This plot indicates that, at 80 meters ahead, a typical lane stripe will cover a pixel extent of 1 pixel. The corresponding angular accuracy to match lane markers can similarly be calculated for the same camera viewing angle and pixel count, and one finds that the angular acuity of 1 pixel or 2 pixel coverage – which is usually easily obtained via regression fitting of all lane pixels within an image – is sufficient to far exceed the range limits of RADAR, as discussed in the following section.

Figure 4.7: Headway calculated using method 2, using various estimates of ego-vehicle yaw-angle.
4.4 Comparison of methods

Fig. 4.9 shows a comparison of the various methods of headway calculation discussed in this work, presented only for situations in the test route where the vehicle was at highway speeds (above 40 mph). The results reveal that the use of RADAR alone, Method 1, will generally result in so much ambiguity of lane position that one cannot use this technique to discern available headway to the limits of the RADAR’s maximum sighting range. And in many situations, the lane designations would not be suitable for emergency braking requirements. For Method 2 – which uses RADAR with GPS-based maps of lane geometries – one finds that the use of a mean error metric seems to guarantee lane awareness beyond the RADAR’s range. However, closer analysis of the per-station standard deviation shows that some road geometries will produce situations of limited headway awareness, below the RADAR range. Finally, the camera-based deviation calculations (Method 3) show that if lane-designations are matched to the limit of the camera’s pixel capabilities, it generates headway prediction accuracies several times further than the range limitations of RADAR.
The conclusions one can draw from these results is that RADAR alone is not sufficient for determining long-range clearance of the ego-lane, but augmenting RADAR with lane-geometry maps will nearly always enable headway attribution up to and beyond the RADAR limit, albeit with exceptions that are strongly related to the particular road. These errors are correctable with camera systems, subject to accuracies in visual registration of lane markers to the limit of camera sensor performance. The use of camera systems for the registration of mapped information is a very active area of current research, with clear benefits for the challenges considered in this study.

Figure 4.9: The distance versus pixel lateral extent for a 10cm lane marker
Chapter 5

Conclusions

The objective of this work is to present an investigation of the different sources of error in a proposed headway estimation algorithm. Three different strategies with varying level of sensor information is analyzed and the resulting errors in headway estimation are calculated. The results illustrate that attribution of external targets to specific lane designations is very difficult if RADAR is used alone, and if assuming straight-path forward travel. When road geometry information is added, one can attribute traffic to lane positions often beyond the range limit of RADAR; however, experimental testing still reveals some situations on typical highways where lane curvature would still cause traffic attribution errors. Geometric calculations assuming typical image-based registration of lane markers reveals that the use of a camera sensor to track lane markers could correct traffic attribution errors, thus enabling headway clearance estimation at least to the range of commercial RADAR systems.

It should be noted that these methods are are by no means the only errors that one might encounter when trying to estimate headway using only one front-facing RADAR mounted behind the bumper of the ego-vehicle. Significant improvements could be obtained through the use of model-based pose estimation, data fusion techniques, V2x communication systems, and similar strategies. These are active areas of research that could greatly benefit the goals of this study, to estimate the in-lane region where headway can be accurately calculated.
Appendix A

Script for headway estimation

clear
close all

% Data parameters
file_names = {'2018_04_18_MixedB1_HalfLoadedTrailer.mat'; ... 
'2018_04_18_MixedB2_HalfLoadedTrailer.mat'; ... 
'2018_04_18_MixedB3_HalfLoadedTrailer.mat'; ... 
'2018_04_18_MixedB4_HalfLoadedTrailer.mat'; ... 
'2018_04_18_MixedB5_HalfLoadedTrailer.mat'};

traversal_ind = 4;
speed_threshold = 20;

% Parameters for aligning each traversal by station
index_to_match = 4000; % A random point on each path to match to the first traversal
crop_to_common_station = 1;
interpolate_station = 1;
station_decimation = 0.5;

% Camera pixel uncertainty (use script_camera_distance_to_pixel_error ) to
% calculate these. This depends on how many pixels you think it will take
% to measure a lane marker.
sigma_yaw_from_camera = 0.001281; % 1-pixel width (0.073408 deg)
sigma_yaw_from_camera = 0.002559; % 2-pixel width (0.146631 deg)
% sigma_yaw_from_camera = 0.003834; % 3-pixel width (0.219669 deg)

start_offset_times = [550 0 0 0];
end_offset_times = [0 0 0 1175];

utilities = Utilities();

% Create instance of Headway class.
headway = Headway();
headway.maximum_sighting_distance = 200;
headway.show_live_plot = 0;
headway.minimum_longitudinal_velocity = 5;

% Plot parameters
font_size = 16;
line_width = 2;
sigma_multiplier = 2; % Specifies the number of sigmas to plot

% Load the sensor data
if ~exist('data', 'var')
    data = {};
data.traversal = {};
    for i = 1:length(file_names)
        all_data = load(strcat('data/', file_names{i}));
        % Create a structure for the GPS data
data.traversal{i}.time = all_data.NonCANdata.RawGPS.Time;
data.traversal{i}.latitude = all_data.NonCANdata.RawGPS.Latitude;
data.traversal{i}.longitude = all_data.NonCANdata.RawGPS.Longitude;
data.traversal{i}.altitude = all_data.NonCANdata.RawGPS.Height;
data.traversal{i}.east_velocity = all_data.NonCANdata.RawGPS.VEast;
data.traversal{i}.north_velocity = all_data.NonCANdata.RawGPS.VNorth;
    end
end
data.traversal{i}.yaw_rate = all.data.VolvoCANdata.
  VDC2.X.A._YawRate(:,2);

if i == 1

  % Create instance of Map class and process the GPS data
to calculate
  % station and project the latitude/longitude/altitude
  points into global XYZ
  % We will set the first GPS reading as our 'base station'
to calculate the
  % global XYZ coordinates and station.
  map_config = {};
  map_config.reference_latitude = data.traversal{1}.
    latitude(1);
  map_config.reference_longitude = data.traversal{1}.
    longitude(1);
  map_config.reference_altitude = data.traversal{1}.
    altitude(1);
  m = Map(map_config);

end

data.traversal{i} = m.processGPS(data.traversal{i});

inds = find(data.traversal{i}.time >= start_offset_times(i) &
  data.traversal{i}.time <= data.traversal{i}.time(end) -
  end_offset_times(i));

% Trim the data to the desired times
data.traversal{i}.time = data.traversal{i}.time(inds);
data.traversal{i}.station = data.traversal{i}.station(inds);
data.traversal{i}.latitude = data.traversal{i}.latitude(inds);
data.traversal{i}.longitude = data.traversal{i}.longitude(inds);
data.traversal{i}.altitude = data.traversal{i}.altitude(inds);
data.traversal{i}.east_velocity = data.traversal{i}.
  east_velocity(inds);
data.traversal{i}.north_velocity = data.traversal{i}.north.velocity(inds);
data.traversal{i}.U = data.traversal{i}.U(inds);
data.traversal{i}.V = data.traversal{i}.V(inds);
data.traversal{i}.X = data.traversal{i}.X(inds);
data.traversal{i}.Y = data.traversal{i}.Y(inds);
data.traversal{i}.Z = data.traversal{i}.Z(inds);
data.traversal{i}.yaw = data.traversal{i}.yaw(inds);
data.traversal{i}.yaw_rate = data.traversal{i}.yaw_rate(inds);

% Zero each time
data.traversal{i}.time = data.traversal{i}.time − data.traversal{i}.time(1);
end

data = m.alignDataByStation(data, index_to_match, crop_to_common_station, interpolate_station, station_decimation);

for i = 1:length(file_names)
data.traversal{i}.highspeed_flag = double(data.traversal{i}.U > speed_threshold);
    data.traversal{i}.highspeed_flag(data.traversal{i}.highspeed_flag == 0) = NaN;
end
fprintf('Data is loaded!
');
end

%%

% HEADWAY CALCULATION FOR DIFFERENT VEHICLE WIDTHS
We can vary the width of the vehicle that is in our lane ahead of us below.

% Approximated width of a motorcycle
vehicle_width_1 = 1.0; % meters

% Width of a Honda Accord
vehicle_width_2 = 1.6; % meters

% Maximum truck width in North America
% https://ops.fhwa.dot.gov/FREIGHT/publications/size_regs_final_rpt/
% index.htm
vehicle_width_3 = 2.6; % meters

% Let's set some configuration parameters for these tests
config = {};
config.method = 'straight_road';

% Highway lane width in rural and urban
% https://safety.fhwa.dot.gov/geometric/pubs/mitigationstrategies/
% chapter3/3_lanewidth.cfm
config.lane_width = 3.6;

fprintf('Processing the GPS data for vehicle width = %.2fm...
', config.vehicle_width);
headway_width_1 = headway.processFullTraversal(data.traversal{
    traversal_ind}, config);
fprintf('Finished processing the data!\n')

fprintf('Processing the GPS data for vehicle width = %.2fm...
', config.vehicle_width);
headway_width_2 = headway.processFullTraversal(data.traversal{
    traversal_ind}, config);
            traversal_ind}, config);
    fprintf('Finished processing the data!\n')

  config.vehicle_width = vehicle_width_3;
  fprintf('Processing the GPS data for vehicle width = %.2fm\n',
          config.vehicle_width);
  headway_width_3 = headway.processFullTraversal(data.traversal{
    traversal_ind}, config);
  fprintf('Finished processing the data!\n')

  all_yaw_measurements = zeros(length(file_names), length(data.traversal{
    1}.station));
  for i = 1:length(file_names)
    all_yaw_measurements(i,:) = data.traversal{i}.yaw .* data.
    traversal{i}.highspeed_flag;
  end

  % It's possible that a yaw angle could be a multiple of 2*pi, which
  % makes it difficult to find a standard deviation. For instance, the yaw
  % measurement % could be pi or 3*pi, which are the same angle, but one is 3.14 and
  % the
% other is 9.42... which doesn't help for finding statistics. First we will
% convert them to 0 - 2*pi, and then unwrap them back out.
all_yaw_measurements = unwrap(mod(all_yaw_measurements, 2*pi));

% Find the standard deviation of the yaw measurements
yaw_std = std(all_yaw_measurements);

% Remove large spikes (isoutlier removes 3-sigma)
yaw_std_no_outliers = yaw_std;
yaw_std_no_outliers(isoutlier(yaw_std, 'mean')) = NaN;

% Find the mean yaw standard deviation. Some of the deviations are Nans,
% which screw up calculating the mean. Luckily MATLAB has a nifty function
% that calculates the mean without the Nans!
mean_yaw_std = nanmean(yaw_std_no_outliers);
headway_from_mean_yaw_deviation = (vehicle_width_3 ./ (sigma_multiplier * mean_yaw_std));

% Calculate the headway based on the standard deviation at a particular
% station coordinate and based on the mean of those deviations.
headway_method_2_std = vehicle_width_3 ./ (sigma_multiplier * yaw_std_no_outliers);

% The yaw deviations are very noisy, so they cause really large spikes in
% the headway estimate. Let's cap these at a max value.
headway_method_2_std(headway_method_2_std > headway_from_mean_yaw_deviation) = headway_from_mean_yaw_deviation;

headway_method_2_mean = headway_from_mean_yaw_deviation * ones(1, length(data.traversal{traversal_ind}.station));

% Calculate the headway based on the uncertainty of measuring a lane marker
headway_camera = vehicle_width_3 / (sigma_multiplier *
% Headway method 3 = headway_camera * ones(1, length(data.traversal{traversal_ind}.station));

%%

% Histogram plot of headway for three different vehicle types
figure(19200)
clf
h1 = histogram(headway_width_1);
hold on
h2 = histogram(headway_width_2);
h3 = histogram(headway_width_3);
yl = ylim;
plot([75 75],yl,'k','LineWidth',3) % Transition from vehicle geometry to road geometry
plot([57.5 57.5],yl,'k--','LineWidth',3) % Typical sedan
xlabel('Headway (m)', 'FontSize', fontsize)
ylabel('Observations', 'FontSize', fontsize)
legend({'Motorcycle: ', num2str(vehicle_width_1), 'm'},... 
       {'Sedan: ', num2str(vehicle_width_2), 'm'},...
       {'Truck: ', num2str(vehicle_width_3), 'm'}),...
       'FontSize', fontsize)
set(gca,'FontSize', fontsize)
saveas(gcf,'figures/histogram_of_headway_different_vehicle_widths_mixedb5', 'epsc')
saveas(gcf,'figures/histogram_of_headway_different_vehicle_widths_mixedb5', 'fig')
saveas(gcf,'figures/histogram_of_headway_different_vehicle_widths_mixedb5', 'png')

% Headway vs. time for different vehicle widths
min_station_ind = find(data.traversal{traversal_ind}.station == 16630);
max_station = find(data.traversal{traversal_ind}.station == 17780);
figure(115200)
clf
plot(data.traversal{traversal_ind}.station(min_station_ind:max_station), headway_width_1(min_station_ind:max_station),...
33

LineWidth’, line_width) hold on
plot(data.traversal{traversal_ind}.station(min_station_ind:max_station),’
LineWidth’, line_width)
plot(data.traversal{traversal_ind}.station(min_station_ind:max_station),’
LineWidth’, line_width)
xlim([min(data.traversal{traversal_ind}.station(min_station_ind:
max_station)) max(data.traversal{traversal_ind}.station(
min_station_ind:max_station))])
ylim([0 headway.maximum_sighting_distance])
xlabel(’Station (m)’, ’FontSize’, font_size)
ylabel(’Headway (m)’, ’FontSize’, font_size)
legend([strcat(’Motorcycle: ’, num2str(vehicle_width_1), ’m’), ...
strcat(’Sedan: ’, num2str(vehicle_width_2), ’m’), ...
strcat(’Truck: ’, num2str(vehicle_width_3), ’m’)], ’FontSize’, font_size)
set(gca, ’FontSize’, font_size)
saveas(gcf, ’figures/
headway_vs_station_different_vehicle_widths_mixedb5’, ’epsc’)
saveas(gcf, ’figures/
headway_vs_station_different_vehicle_widths_mixedb5’, ’fig’)
saveas(gcf, ’figures/
headway_vs_station_different_vehicle_widths_mixedb5’, ’png’)

%%

% Plot of a section of yaw vs. station. You can set the range below.
min_station = 5000;
max_station = 7000;
min_station_ind = find(data.traversal{traversal_ind}.station ==
min_station);
max_station_ind = find(data.traversal{traversal_ind}.station ==
max_station);
figure(28800)
clf
hold on
legend_labels = cell(length(file_names),1);
for i = 1:length(file_names)
    plot(data.traversal{1}.station(min_station_ind:max_station_ind),
         rad2deg(mod(data.traversal{1}.yaw(min_station_ind:
                     max_station_ind), 2*pi)), 'LineWidth', line_width)
    legend_labels{i} = ['Traversal ' num2str(i)];
end
xlim([min_station max_station])
ylim([220 260])
xlabel('Station (m)', 'FontSize', font_size)
ylabel('Yaw (deg)', 'FontSize', font_size)
legend(legend_labels, 'FontSize', font_size)
set(gca, 'FontSize', font_size)

saveas(gcf, 'figures/yaw_vs_station_repeatability_with_lane_change', 'epsc')
saveas(gcf, 'figures/yaw_vs_station_repeatability_with_lane_change', 'fig')
saveas(gcf, 'figures/yaw_vs_station_repeatability_with_lane_change', 'png')

% Histogram of the yaw deviations (multiplied by the sigma factor)
figure(38400)
clf
hold on
histogram(rad2deg(sigma_multiplier*yaw_std_no_outliers));
yl = ylim;
plot([rad2deg(sigma_multiplier*mean_yaw_std) rad2deg(2*mean_yaw_std) ]
     ,yl, 'k', 'LineWidth',3)
text(rad2deg(sigma_multiplier*mean_yaw_std)*1.2, yl(2)*0.95, ['2−\sigma = '
     num2str(rad2deg(sigma_multiplier*mean_yaw_std), '%.2f') ' deg'], 'FontSize', font_size)
xlabel('Yaw Standard Deviation, 2−\sigma (deg)', 'FontSize', font_size)
ylabel('Observations', 'FontSize', font_size)
set(gca, 'FontSize', font_size)

saveas(gcf, 'figures/yaw_standard_deviation_histogram', 'epsc')
saveas(gcf, 'figures/yaw_standard_deviation_histogram', 'fig')
saveas(gcf, 'figures/yaw_standard_deviation_histogram', 'png')
% Plot of each headway method
figure(57600)
clf
plot(data.traversal{traversal_ind}.station, headway_width_3)
hold on
plot(data.traversal{traversal_ind}.station, headway_method_2_std)
plot(data.traversal{traversal_ind}.station, headway_method_2_mean, 'LineWidth', line_width)
plot(data.traversal{traversal_ind}.station, headway_method_3, 'LineWidth', line_width)
plot(data.traversal{traversal_ind}.station, headway.
   maximum_sighting_distance * ones(1, length(data.traversal{traversal_ind}.station)), 'LineWidth', line_width)
xlim([min(data.traversal{traversal_ind}.station) max(data.traversal{traversal_ind}.station)])
xlabel('Station')
ylabel('Headway (m)')
legend('In-lane Yaw Error (Method 1)', 'Standard Deviation (2-sigma) (Method 2)', 'Mean Standard Deviation (2-sigma) (Method 2)', 'Camera-based Deviation (Method 3)', 'Maximum Sighting Range')
set(gca, 'FontSize', font_size)
saveas(gcf, 'figures/headway_vs_station_multiple_methods', 'epsc')
saveas(gcf, 'figures/headway_vs_station_multiple_methods', 'fig')
saveas(gcf, 'figures/headway_vs_station_multiple_methods', 'png')
Appendix B

Script to plot camera distance to pixel error

1 % clear , clc
2 close all
3
4 minimum_x = 3; % meters
5 maximum_x = 200; % meters
6 number_of_pixels_offset = 1;
7
8 lane_marker_width = 0.1016; % meters (4")
9 x_decimation = 0.1; % meters
10
11 % Camera pose
12 t_x = 0;
13 t_y = 0;
14 t_z = 1.5;
15 roll = 0;
16 pitch = 0;
17 yaw = 0;
18
19 % Camera calibration parameters
20 focal_x = 779.67316;
21 focal_y = 778.997918;
22 skew = 0;
23 center_x = 518.087984;
24 center_y = 396.89055;
image_width = 1024;
image_height = 768;

% Plot parameters
font_size = 16;
line_width = 2;

% Create instance of Camera class
camera = Camera(t_x, t_y, t_z, roll, pitch, yaw, ...
                focal_x, focal_y, skew, center_x, center_y);

% Create a matrix to store the XYZ data of a single lane marker, centered
% laterally in front of us, going out to a very far distance.
x = minimum_x:x:decimation:maximum_x;
xyz = [x; lane_marker_width * ones(1,length(x)); zeros(1,length(x))];
px = camera.projectToImage(xyz);
px_width = center.x - px(1,:);

% Find the last pixel width that equals our minimum detectable pixel width
min_lookaheaed_distance_ind = find(px_width < number_of_pixels_offset ,1);
min_lookaheaed_distance = x(min_lookaheaed_distance_ind);

yaw_uncertainty_at_min_lookaheaed_distance = atan2(lane_marker_width, 
                                                min_lookaheaed_distance);

fprintf('Yaw uncertainty @ minimum lookahead distance of %.1f m: 
        %.6f (rad) %.6f (deg)\n', ...
        min_lookaheaed_distance, ...
yaw_uncertainty_at_min_lookaheaed_distance, ...
r
rad2deg(yaw_uncertainty_at_min_lookaheaed_distance));

figure(115200)
clf
plot(x, px_width, 'LineWidth', line_width)
hold on
yl = ylim;
plot([min_lookaheaed_distance min_lookaheaed_distance],yl,'k','
% We can constrain the minimum and maximum forward distance (x). This
% calculates the pixel coordinates of these world points so we can
% limit
% our homography calculation below.
xyz_limits = [minimum_x maximum_x; 0 0; 0 0];
px_limits = camera.projectToImage(xyz_limits);

% Let's start from the bottom of the image in the center and
calculate the
% the width (in meters) of a single pixel as we move up the image.
px = px_limits(2,1):-1:px_limits(2,2);
pixels = [px_limits(1,1)*ones(1,length(px))-number_of_pixels_offset; px];

% We can use the inverse of the homography to calculate distance in
% meters
% for a particular pixel, assuming a flat road (z=0) in this case.
xyz = camera.projectFromImageToCameraFrame(pixels);
xyz = xyz ./ xyz(3,:);
figure(19200)
clf
plot(xyz(:,1),xyz(:,2), 'LineWidth', line_width)
xlabel('Distance ahead of vehicle (m)', 'FontSize', font_size)
ylabel(['Width of ' num2str(number_of_pixels_offset) ' pixels (m)'], 'FontSize', fontsize)
grid on
set(gca, 'FontSize', fontsize)
Appendix

Script to calculate yaw-angle orientation errors using different methods

clear
 clf
 close all

% Data parameters

file_names = {'2018_04_18_MixedB1_HalfLoadedTrailer.mat'; ...
     '2018_04_18_MixedB2_HalfLoadedTrailer.mat'; ...
     '2018_04_18_MixedB3_HalfLoadedTrailer.mat'; ...
     '2018_04_18_MixedB4_HalfLoadedTrailer.mat'; ...
     '2018_04_18_MixedB5_HalfLoadedTrailer.mat'};

speed_threshold = 20;

start_offset_times = [550 0 0 0];
end_offset_times = [0 0 0 1175];

utilities = Utilities();

% Plot parameters

font_size = 16;

% Load the sensor data
% if ~exist('data', 'var')

data = {}; 
data.traversal = {}; 
for i = 1:length(file_names)

    all_data = load(strcat('data/', file_names{i}));

% Create a structure for the GPS data
    data.traversal{i}.time = all_data.NonCANdata.RawGPS.Time;
    data.traversal{i}.latitude = all_data.NonCANdata.RawGPS.Latitude;
    data.traversal{i}.longitude = all_data.NonCANdata.RawGPS.Longitude;
    data.traversal{i}.altitude = all_data.NonCANdata.RawGPS.Height;
    data.traversal{i}.east_velocity = all_data.NonCANdata.RawGPS.VEast;
    data.traversal{i}.north_velocity = all_data.NonCANdata.RawGPS.VNorth;
    data.traversal{i}.yaw_rate = all_data.VolvoCANdata.VDC2_X_A_YawRate(:, 2);
    
if i == 1

    % Create instance of Map class and process the GPS data
to calculate
    % station and project the latitude/longitude/altitude
    % points into global XYZ
    % We will set the first GPS reading as our 'base station'
to calculate the
    % global XYZ coordinates and station.
    config.reference_latitude = data.traversal{1}.latitude(1);
    config.reference_longitude = data.traversal{1}.longitude(1);
    config.reference_altitude = data.traversal{1}.altitude(1);
    m = Map(config);
data.traversal{i} = m.processGPS(data.traversal{i});

inds = find(data.traversal{i}.time >= start_offset_times(i) &
            data.traversal{i}.time <= data.traversal{i}.time(end) -
            end_offset_times(i));

% Trim the data to the desired times
data.traversal{i}.time = data.traversal{i}.time(inds);
data.traversal{i}.station = data.traversal{i}.station(inds);
data.traversal{i}.latitude = data.traversal{i}.latitude(inds);
data.traversal{i}.longitude = data.traversal{i}.longitude(inds);
data.traversal{i}.altitude = data.traversal{i}.altitude(inds);
data.traversal{i}.east_velocity = data.traversal{i}.east_velocity(inds);
data.traversal{i}.north_velocity = data.traversal{i}.north_velocity(inds);
data.traversal{i}.U = data.traversal{i}.U(inds);
data.traversal{i}.V = data.traversal{i}.V(inds);
data.traversal{i}.X = data.traversal{i}.X(inds);
data.traversal{i}.Y = data.traversal{i}.Y(inds);
data.traversal{i}.Z = data.traversal{i}.Z(inds);
data.traversal{i}.yaw = data.traversal{i}.yaw(inds);
data.traversal{i}.yaw_rate = data.traversal{i}.yaw_rate(inds);

% Zero each time
data.traversal{i}.time = data.traversal{i}.time - data.traversal{i}.time(1);

end

% end

index_to_match = 4000;
crop_to_common_station = 1;
interpolate_station = 1;
data = m.alignDataByStation(data, index_to_match,
crop_to_common_station, interpolate_station);

%%

% Combine the yaw measurements into one matrix so we can easily find
% the
% standard deviation
figure
hold on
all_yaw_measurements = zeros(length(file_names), length(data.traversal{1}.station));
legend_labels = cell(length(file_names), 1);
for i = 1:length(file_names)
    all_yaw_measurements(i,:) = data.traversal{i}.yaw;
    removeOutliers(data.traversal{i}.yaw, deg2rad(1));
    plot(data.traversal{i}.station, rad2deg(mod(data.traversal{i}.yaw, 2*pi)));
    legend_labels{i} = [‘Traversal ‘ num2str(i)];
end
xlabel(‘Station (m)’, ’FontSize’, font_size)
ylabel(‘Yaw (deg)’, ’FontSize’, font_size)
legend(legend_labels, ’FontSize’, font_size)
set(gca, ’FontSize’, font_size)

for i = 1:length(file_names)
data.traversal{i}.highspeed_flag = double(data.traversal{i}.U > speed_threshold);
data.traversal{i}.highspeed_flag(data.traversal{i}.highspeed_flag == 0) = NaN;
end

all_yaw_measurements_{1,:) = all_yaw_measurements(1,:).* data.traversal{1}.highspeed_flag;
all_yaw_measurements_{2,:) = all_yaw_measurements(2,:).* data.traversal{2}.highspeed_flag;
all_yaw_measurements_{3,:) = all_yaw_measurements(3,:).* data.traversal{3}.highspeed_flag;
all_yaw_measurements_{4,:) = all_yaw_measurements(4,:).* data.traversal{4}.highspeed_flag;
traversal{4}.highspeed_flag;

yaw_std = std(mod(all_yaw_measurements, 2*pi));

yaw_std_outliers = utilities.removeOutliers(yaw_std, 1.5);

figure
histogram(rad2deg(yaw_std_outliers));

figure
hold on
plot(data.traversal{1}.station, rad2deg(yaw_std))
plot(data.traversal{1}.station, rad2deg(yaw_std))
plot(data.traversal{1}.station, rad2deg(yaw_std_outliers))
xlabel('Station (m)', 'FontSize', fontsize)
ylabel('Yaw Error, 1-sigma (deg)', 'FontSize', fontsize)
set(gca, 'FontSize', fontsize)
Helper classes used in the main scripts

Class for coordinate transformation

```matlab
classdef Transform < handle

    % Class properties and variables
    properties
        frames
        origin_frame
    end

    methods

        % Constructor for class.
        function obj = Transform(obj)
            obj.frames = {};
        end

        function point = rotate(obj, point, roll, pitch, yaw)
            R = Transform.rotationMatrix(roll, pitch, yaw);
        end

end
```
point = R * point;

end

function point = translate(obj, point, x, y, z)

T = Transform.translationMatrix(x, y, z);

point = T * point;

end

function [x, y, z] = transformToGlobal(obj, from_frame, point)

if nargin == 2
    point = [0 0 0 1]’;
end

x_from = obj.frames.(from_frame).x;
y_from = obj.frames.(from_frame).y;
z_from = obj.frames.(from_frame).z;
roll_from = obj.frames.(from_frame).roll;
pitch_from = obj.frames.(from_frame).pitch;
yaw_from = obj.frames.(from_frame).yaw;

R_from = Transform.rotationMatrix(roll_from, pitch_from, yaw_from);

T_from = Transform.translationMatrix(x_from, y_from, z_from);

E_from = R_from * T_from;

x_to = obj.frames.(obj.origin_frame).x;
y_to = obj.frames.(obj.origin_frame).y;
z_to = obj.frames.(obj.origin_frame).z;
roll_to = obj.frames.(obj.origin_frame).roll;
pitch_to = obj.frames.(obj.origin_frame).pitch;
yaw_to = obj.frames.(obj.origin_frame).yaw;

R_to = Transform.rotationMatrix(roll_to, pitch_to, yaw_to);

T_to = Transform.translationMatrix(x_to, y_to, z_to);

E_to = R_to * T_to;

E = E_from * E_to;

if size(point,1) == 3
    point = [point; ones(1, size(point,2))];
end

origin = E \ point;

x = origin(1,:);
y = origin(2,:);
z = origin(3,:);
end

function [x, y, z] = transformToLocal(obj, from_frame, point)

if nargin == 2
    point = [0 0 0 1]';
end

x_from = obj.frames.(from_frame).x;
y_from = obj.frames.(from_frame).y;
z_from = obj.frames.(from_frame).z;
roll_from = obj.frames.(from_frame).roll;
pitch_from = obj.frames.(from_frame).pitch;
yaw_from = obj.frames.(from_frame).yaw;

R_from = Transform.rotationMatrix(roll_from, pitch_from, yaw_from);
T.from = Transform.translationMatrix(x.from, y.from, z_from);

E_from = R_from * T_from;

x_to = obj.frames.(obj.origin_frame).x;
y_to = obj.frames.(obj.origin_frame).y;
z_to = obj.frames.(obj.origin_frame).z;
roll_to = obj.frames.(obj.origin_frame).roll;
pitch_to = obj.frames.(obj.origin_frame).pitch;
yaw_to = obj.frames.(obj.origin_frame).yaw;

R_to = Transform.rotationMatrix(roll_to, pitch_to, yaw_to);

T_to = Transform.translationMatrix(x_to, y_to, z_to);

E_to = R_to * T_to;

E = E_from * E_to;

if size(point,1) == 3
    point = [point; ones(1, size(point,2))];
end

origin = E * point;

x = origin(1,:);
y = origin(2,:);
z = origin(3,:);

end

% setPose
function obj = setOffsetFromOrigin(obj, frame, x, y, z, roll, pitch, yaw)
if nargin == 3

    if isstruct(x)
        pose = cell2mat(cat(1,struct2cell(x)));
    else
        pose = x;
    end

    x = pose(1);
    y = pose(2);
    z = pose(3);
    roll = pose(4);
    pitch = pose(5);
    yaw = pose(6);
end

if ~isfield(obj.frames, frame)
    obj.frames.(frame) = {};
end

obj.setPosition(frame, x, y, z);
obj.setOrientation(frame, roll, pitch, yaw);
end

function obj = setOrigin(obj, frame, x, y, z, roll, pitch, yaw)

    if nargin == 3

        if isstruct(x)
            pose = cell2mat(cat(1,struct2cell(x)));
        else
            pose = x;
        end

        x = pose(1);
        y = pose(2);
        z = pose(3);
    end

end
roll = pose(4);
pitch = pose(5);
yaw = pose(6);

end

if ~isfield(obj.frames, frame)
    obj.frames.(frame) = {};
end

obj.origin.frame = frame;
obj.setPosition(frame, x, y, z);
obj.setOrientation(frame, roll, pitch, yaw);
end

function obj = setPosition(obj, frame, x, y, z)
    if nargin == 3
        position = x;
        x = position(1);
        y = position(2);
        z = position(3);
    end

    obj.frames.(frame).x = x;
    obj.frames.(frame).y = y;
    obj.frames.(frame).z = z;
end

function obj = setOrientation(obj, frame, roll, pitch, yaw)
    if nargin == 3
        orientation = roll;
        roll = orientation(1);
pitch = orientation(2);
yaw = orientation(3);

end

obj.frames.(frame).roll = roll;
obj.frames.(frame).pitch = pitch;
obj.frames.(frame).yaw = yaw;

end

function [x, y, z] = getOrigin(obj)

x = obj.frames.(obj.origin_frame).x;
y = obj.frames.(obj.origin_frame).y;
z = obj.frames.(obj.origin_frame).z;
roll = obj.frames.(obj.origin_frame).roll;
pitch = obj.frames.(obj.origin_frame).pitch;
yaw = obj.frames.(obj.origin_frame).yaw;

R = Transform.rotationMatrix(roll, pitch, yaw);
T = Transform.translationMatrix(x, y, z);
E = R * T;
origin = E \ [0 0 0 1]';
x = origin(1);
y = origin(2);
z = origin(3);

end

end

methods (Static)

function R = rotationMatrix(roll, pitch, yaw)
\[
\begin{align*}
\text{Rx} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\text{roll}) & \sin(\text{roll}) & 0 \\ 0 & -\sin(\text{roll}) & \cos(\text{roll}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ; \\
\text{Ry} &= \begin{bmatrix} \cos(\text{pitch}) & 0 & -\sin(\text{pitch}) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\text{pitch}) & 0 & \cos(\text{pitch}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ; \\
\text{Rz} &= \begin{bmatrix} \cos(\text{yaw}) & \sin(\text{yaw}) & 0 & 0 \\ -\sin(\text{yaw}) & \cos(\text{yaw}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} ;
\end{align*}
\]

\[
\text{R} = \text{Rx} \ast \text{Ry} \ast \text{Rz} ;
\]

end

function \( \text{T} = \text{translationMatrix}(x, y, z) \)

\[
\begin{align*}
\text{T} &= \text{eye}(4) ; \\
\text{T}(1,4) &= -x ; \\
\text{T}(2,4) &= -y ; \\
\text{T}(3,4) &= -z ;
\end{align*}
\]

end
end

classdef Map < handle

% Class properties and variables
properties

config
gps
end
methods

% Constructor for class.
function obj = Map(config)

    obj.config = config;

    if ~isempty(obj.config) && ~isempty(obj.config.reference_latitude)
        obj.gps = GPS(obj.config.reference_latitude, obj.config.reference_longitude, obj.config.reference_altitude);
    else
        obj.gps = GPS();
    end
end

function data = processGPS(obj, data)

    ENU = obj.gps.WGS84TA2ENU(data.latitude, data.longitude, data.altitude);
    station = obj.calculateStation(ENU);

    data.station = station;
    data.X = ENU(1,:);
    data.Y = ENU(2,:);
    data.Z = ENU(3,:);

    data.yaw = atan2(diff(data.Y), diff(data.X));
    data.yaw = [data.yaw; data.yaw(end)];
    data.yaw = unwrap(data.yaw);

    % Calculate lateral and longitudinal velocity from INS
    data.U = data.east_velocity .* cos(data.yaw) + data.
north_velocity.*sin(data.yaw);
data.V = -data.east_velocity.*sin(data.yaw) + data.
north_velocity.*cos(data.yaw);

end

function [id, station, lateral_offset, heading_map] =
    stationHere(obj,latitude_x,longitude_y,altitude_z,origin_x
    ,origin_y,origin_z,S,X_map,Y_map,psi_map,type)

    if type == "lla"
        enu = obj.g.WGSLLA2ENU(latitude_x,longitude_y,
                                 altitude_z,origin_x,origin_y,origin_z);
        x = enu(1);
        y = enu(2);
    elseif type == "xyz"
        x = latitude_x;
        y = longitude_y;
    else
        error("fcn_station_here: Parameter 'type' must be lla
                or xyz\n")
    end

% nearest_station_ind
id = knnsearch([X_map Y_map],[x y]);
s = S(id);
x_map = X_map(id);
y_map = Y_map(id);
heading_map = psi_map(id);

delta_X = -x_map + x;
delta_Y = -y_map + y;
\% station = \text{delta}_X \cdot \cos(-\text{heading}_\text{map}) - \text{delta}_Y \cdot \sin(-\text{heading}_\text{map}) + s;
\%
lateral\_offset = \text{delta}_X \cdot \sin(-\text{heading}_\text{map}) + \text{delta}_Y \cdot \cos(-\text{heading}_\text{map});

id = \text{knnsearch}([X\_map \ Y\_map],[x \ y], 'k',2);

id = \text{sort}(id);

map1 = [X\_map(id(1)); \ Y\_map(id(1))];
map2 = [X\_map(id(2)); \ Y\_map(id(2))];
location = [x; y];

s = S(id(1));
ab = map2 - map1;
ab\_squared = \text{dot}(ab, ab);
ap = location - map1;
t = \text{dot}(ap, ab) \div \text{ab\_squared};
point\_on\_line = map1 + ab \cdot \text{diag}(t);
lateral\_offset = \text{sqrt}\(((\text{point\_on\_line}(1)-\text{location}(1))^2+(\text{point\_on\_line}(2)-\text{location}(2))^2)));
heading\_map = \text{psi}_\text{map}(id(1)); \% \text{atan2}(\text{map2}(2)-\text{map1}(2),\text{map2}(1)-\text{map1}(1));
station = s + \text{sqrt}\(((\text{point\_on\_line}(1)-\text{map1}(1))^2+(\text{point\_on\_line}(2)-\text{map1}(2))^2));

\text{end}

\text{function} \ data = \text{alignDataByStation} (\text{obj}, \text{data}, \text{index\_to\_match}, \text{crop\_to\_common\_station}, \text{interpolate\_station}, \text{station\_decimation})

station\_offsets = \text{zeros}(1,\text{length}(\text{data\_traversal}));
id\_offset = \text{zeros}(1,\text{length}(\text{data\_traversal}));
station\_offsets(1) = 0;
id\_offset(1) = 1;
\text{for} \ i = 2:\text{length}(\text{data\_traversal})

\ X = \text{data\_traversal}\{i\}\_X(\text{index\_to\_match});
\ Y = \text{data\_traversal}\{i\}\_Y(\text{index\_to\_match});
Z = data.traversal{i}.Z(index_to_match);

[id, station, lateral_offset, heading_map] = obj.
    stationHere(X,Y,Z,0,0,0,data.traversal{1}.station,
    data.traversal{1}.X,data.traversal{1}.Y,data.
    traversal{1}.yaw,"xyz");
station_offsets(i) = data.traversal{i}.station(
    index_to_match) - station;
id_offset(i) = id(1);

    data.traversal{i}.station = data.traversal{i}.station
    - stationOffsets(i);

end

if crop_to_common_station == 1

    % And we will chop off the beginning and ends of the
    % data so they all have
    % the same length of data. First we need to find the
    % maximum and minimum
    % station.
    start_stations = zeros(1,length(data.traversal));
end_stations = zeros(1,length(data.traversal));
for i = 1:length(data.traversal)
    start_stations(i) = data.traversal{i}.station(1);
    end_stations(i) = data.traversal{i}.station(end);
end
max_start_station = max(start_stations);
min_end_station = min(end_stations);

% Now crop the data
for i = 1:length(data.traversal)
    
    inds = find(data.traversal{i}.station >=
        max_start_station & data.traversal{i}.station
        <= min_end_station);

    % Trim the data to the desired times
    data.traversal{i}.time = data.traversal{i}.time(}
inds(1)) − data.traversal{ i }.time(inds(1));
data.traversal{ i }.station = data.traversal{ i }.station(inds);
data.traversal{ i }.latitude = data.traversal{ i }.latitude(inds);
data.traversal{ i }.longitude = data.traversal{ i }.longitude(inds);
data.traversal{ i }.altitude = data.traversal{ i }.altitude(inds);
data.traversal{ i }.east_velocity = data.traversal{ i }.east_velocity(inds);
data.traversal{ i }.north_velocity = data.traversal{ i }.north_velocity(inds);
data.traversal{ i }.U = data.traversal{ i }.U(inds);
data.traversal{ i }.V = data.traversal{ i }.V(inds);
data.traversal{ i }.X = data.traversal{ i }.X(inds);
data.traversal{ i }.Y = data.traversal{ i }.Y(inds);
data.traversal{ i }.Z = data.traversal{ i }.Z(inds);
data.traversal{ i }.yaw = data.traversal{ i }.yaw(inds);
data.traversal{ i }.yaw_rate = data.traversal{ i }.yaw_rate(inds);

end

end

if interpolate_station == 1

% Interpolate the data to be on the same station
decimation First we need
% to find the minimum and maximum station.
start_stations = zeros(1,length(data.traversal));
end_stations = zeros(1,length(data.traversal));
for i = 1:length(data.traversal)
    start_stations(i) = data.traversal{ i }.station(1);
    end_stations(i) = data.traversal{ i }.station(end);
end
min_start_station = min(start_stations);
max_end_station = max(end_stations);
new_station = min_start_station:station_decimation:
    max_end_station;
for i = 1:length(data.traversal)

    % It's possible that there are non-unique station
    % measurements, so you cannot interpolate the
    % data.
    % This grabs only the unique data indices
    [~,unique inds] = unique(data.traversal{i}.station);

    data.traversal{i}.latitude = interp1(data.
        traversal{i}.station(unique inds), data.
        traversal{i}.latitude(unique inds),
        new_station,'linear','extrap');
    data.traversal{i}.longitude = interp1(data.
        traversal{i}.station(unique inds), data.
        traversal{i}.longitude(unique inds),
        new_station,'linear','extrap');
    data.traversal{i}.altitude = interp1(data.
        traversal{i}.station(unique inds), data.
        traversal{i}.altitude(unique inds),
        new_station,'linear','extrap');
    data.traversal{i}.east_velocity = interp1(data.
        traversal{i}.station(unique inds), data.
        traversal{i}.east_velocity(unique inds),
        new_station,'linear','extrap');
    data.traversal{i}.north_velocity = interp1(data.
        traversal{i}.station(unique inds), data.
        traversal{i}.north_velocity(unique inds),
        new_station,'linear','extrap');
    data.traversal{i}.U = interp1(data.traversal{i}.
        station(unique inds), data.traversal{i}.U( 
        unique inds), new_station,'linear','extrap');
    data.traversal{i}.V = interp1(data.traversal{i}.
        station(unique inds), data.traversal{i}.V( 
        unique inds), new_station,'linear','extrap');
    data.traversal{i}.X = interp1(data.traversal{i}.
        station(unique_inds), data.traversal{i}.X( 

unique_inds), new_station,'linear','extrap');
data.traversal{i}.Y = interp1(data.traversal{i}.station(unique_inds), data.traversal{i}.Y(unique_inds), new_station,'linear','extrap');
data.traversal{i}.Z = interp1(data.traversal{i}.Z(unique_inds), data.traversal{i}.Z(unique_inds), new_station,'linear','extrap');
data.traversal{i}.yaw = interp1(data.traversal{i}.station(unique_inds), data.traversal{i}.yaw(unique_inds), new_station,'linear','extrap');
data.traversal{i}.yaw_rate = interp1(data.traversal{i}.station(unique_inds), data.traversal{i}.yaw_rate(unique_inds), new_station,'linear','extrap');
data.traversal{i}.station = new_station - min_start_station;

end

end

end

end

methods (Static)

function station = calculateStation(X, Y, Z)

if nargin == 1
    ENU = X;
    X = ENU(:,1);
    Y = ENU(:,2);
    Z = ENU(:,3);
end

station = sqrt(diff(X).^2 + diff(Y).^2 + diff(Z).^2);
station = cumsum(station);
station = [0; station];
function [left_lane_edge, right_lane_edge] = 
    createRoadBoundaries(lane_center, road_yaw, lane_width)

    road_yaw = road_yaw - road_yaw(1); 
    left_lane_edge = [lane_center(1,:); \ (lane_width/2) * cos(road_yaw - pi/2); \ lane_center(2,:); \ (lane_width/2) * sin(road_yaw - pi/2)];

    right_lane_edge = [lane_center(1,:); \ (lane_width/2) * cos(road_yaw - pi/2); \ lane_center(2,:); \ (lane_width/2) * sin(road_yaw - pi/2)];

end

end

end

Class for obtaining ENU coordinate from GPS data

classdef GPS < handle

    % Class properties and variables
    properties
        % General parameters
        reference_latitude = 40.8623031194444;
        reference_longitude = -77.8362636138889;
        reference_altitude = 333.817;

    end

    properties (Constant)
        A_EARTH = 6378137;
        flattening = 1/298.257223563;

end
NAV_E2 = (2 - GPS.flattening) * GPS.flattening; % also e^{-2}

methods

function obj = GPS(reference_latitude, reference_longitude, reference_altitude)

    if nargin == 3
        obj.reference_latitude = reference_latitude;
        obj.reference_longitude = reference_longitude;
        obj.reference_altitude = reference_altitude;
    else
        fprintf('Using default base station coordinates at LTI test track\n')
    end

end

function ENU = WGSLLA2ENU(obj, latitude, longitude, altitude, reference_latitude, reference_longitude, reference_altitude)

    if nargin == 4

        reference_latitude = obj.reference_latitude;
        reference_longitude = obj.reference_longitude;
        reference_altitude = obj.reference_altitude;
    end

    ENU = zeros(3, length(latitude));
    for i = 1:length(latitude)

        XYZ = WGSLLA2XYZ(obj, latitude(i), longitude(i), altitude(i));

        ENU(:, i) = WGSXYZ2ENU(obj, XYZ, reference_latitude, reference_longitude, reference_altitude);
    end
% Returns the equivalent WGS84 XYZ coordinates (in meters) for a given geodetic latitude (degrees), longitude (degrees), and altitude above the WGS84 ellipsoid in meters. Note: N latitude is positive, S latitude is negative, E longitude is positive, W longitude is negative.

% Ref: Decker, B. L., World Geodetic System 1984, Defense Mapping Agency Aerospace Center.

function XYZ = WGSLLA2XYZ(obj, latitude, longitude, altitude)

    slat = sin(deg2rad(latitude));
    clat = cos(deg2rad(latitude));
    r_n = GPS.A_EARTH / sqrt(1 - GPS.NAV_E2 * slat * slat);
    XYZ = [(r_n + altitude) * clat * cos(deg2rad(longitude));
           (r_n + altitude) * clat * sin(deg2rad(longitude));
           (r_n * (1 - GPS.NAV_E2) + altitude) * slat];

    if (latitude < -90.0) || (latitude > 90.0) || (longitude < -180.0) || (longitude > 360.0)
        error('WGS lat or WGS lon out of range');
    end

end

function ENU = WGSXYZ2ENU(obj, XYZ, reference_latitude, reference_longitute, reference_altitude)

    if nargin == 2

        reference_latitude = obj.reference_latitude;
        reference_longitude = obj.reference_longitude;
        reference_altitude = obj.reference_altitude;

    end
% First, calculate the xyz of reflat, reflon, refalt

refXYZ = WGSLLA2XYZ(obj, reference_latitude, reference_longitude, reference_altitude);

% Difference xyz from reference point

diffXYZ = XYZ - refXYZ;

% Now rotate the (often short) diffxyz vector to enu frame

% Rotate about Z-axis
R1 = eye(3);
c = cos(deg2rad(90 + reference_longitude));
s = sin(deg2rad(90 + reference_longitude));
R1(1,1) = c;
R1(2,2) = c;
R1(2,1) = -s;
R1(1,2) = s;

% Rotate about X-axis
R2 = eye(3);
c = cos(deg2rad(90 - reference_latitude));
s = sin(deg2rad(90 - reference_latitude));
R2(2,2) = c;
R2(3,3) = c;
R2(2,3) = s;
R2(3,2) = -s;

R = R2 * R1;

ENU = R * diffXYZ;

end

function XYZ = ENU2WGSXYZ(obj, ENU, reference_latitude, reference_longitude, reference_altitude)
if nargin == 2

    reference_latitude = obj.reference_latitude;
    reference_longitude = obj.reference_longitude;
    reference_altitude = obj.reference_altitude;
end

% Rotate the ENU vector to XYZ frame

% Rotate about Z-axis
R1 = eye(3);
c = cos(deg2rad(90 + reference_longitude));
s = sin(deg2rad(90 + reference_longitude));
R1(1,1) = c;
R1(2,2) = c;
R1(2,1) = -s;
R1(1,2) = s;

% Rotate about X-axis
R2 = eye(3);
c = cos(deg2rad(90 - reference_latitude));
s = sin(deg2rad(90 - reference_latitude));
R2(2,2) = c;
R2(3,3) = c;
R2(2,3) = s;
R2(3,2) = -s;

R = R2 * R1;

% Rotate
diffXYZ = R \ ENU;

% Calculate the XYZ of the reference latitude, longitude, altitude
refXYZ = WGSLLA2XYZ(obj, reference_latitude, reference_longitude, reference_altitude);
% Add the difference to the reference
XYZ = diffXYZ + refXYZ;

end

function LLA = WGSXYZ2LLA(obj, XYZ)

    if ((XYZ(1) == 0.0) && (XYZ(2) == 0.0))
        longitude = 0.0;
    else
        longitude = rad2deg(atan2(XYZ(2), XYZ(1)));
    end

    if ((XYZ(1) == 0.0) && (XYZ(2) == 0.0) && (XYZ(3) == 0.0))
        error('WGS XYZ located at the center of the earth')
    else

        % Make initial latitude and altitude guesses based on spherical earth.
        rhosqrd = XYZ(1)^2 + XYZ(2)^2;
        rho = sqrt(rhosqrd);
        templat = atan2(XYZ(3), rho);
        tempalt = sqrt(rhosqrd + XYZ(3)^2) - GPS.A.EARTH;
        rhoeerror = 1000.0;
        zerror = 1000.0;

        % Newton’s method iteration on templat and tempalt makes
        % the residuals on rho and z progressively smaller. Loop
        % is implemented as a 'while' instead of a 'do'
        % to simplify
        % porting to MATLAB

        while ((abs(rhoeerror) > 1e-6) || (abs(zerror) > 1e-6))
            slat = sin(templat);
            clat = cos(templat);
q = 1 - GPS.NAV_E2 * slat * slat;

r_n = GPS.A_EARTH / sqrt(q);

drdl = r_n * GPS.NAV_E2* slat * clat / q; % d(r_n )/d(latitude)

rhoerror = (r_n + tempalt) * clat - rho;
zerror = (r_n * (1 - GPS.NAV_E2) + tempalt) * slat - XYZ(3);

%

rational = [drhoerror/dlat 
            drhoerror/dalt 
            dzeror/dlat 
            dzeror/dalt] 

% Find Jacobian

% Apply correction = inv(Jacobian)*errorvector

invdet = 1.0 / (aa * dd - bb * cc);

latitute = rad2deg(templat);

end

end
LLA = [latitude; longitude; altitude];
end

function LLA = ENU2WGSLLA(obj, ENU, reference_latitude, reference_longitude, reference_altitude)

if nargin == 2

    reference_latitude = obj.reference_latitude;
    reference_longitude = obj.reference_longitude;
    reference_altitude = obj.reference_altitude;
end

LLA = zeros(3, size(ENU, 2));
for i = 1:size(ENU, 2)

    XYZ = ENU2WGSXYZ(obj, ENU(:,i), reference_latitude, reference_longitude, reference_altitude);

    LLA(:,i) = WGSXYZ2LLA(obj, XYZ);

end
end
end

Headway class

classdef Headway < handle

    % Class properties and variables
    properties

        show_live_plot = 0;
    end
end
maximum_sighting_distance = 200; % meters  
minimum_longitudinal_velocity = 5; % m/s  

% Create an instance of the Transform class. This class let's you convert  
% points to and from different coordinate frames.  
tf = Transform();  
m = Map({});  

end  

methods  

% Constructor for class.  
function obj = Headway()  

% obj.tf.setOrigin('vehicle', 0, 0, 0, 0, 0);  

% Set the map coordinate frame to the origin  
% Input parameters are a frame name (map) and the X, Y, Z, roll, pitch, yaw  
obj.tf.setOrigin('map', 0, 0, 0, 0, 0, 0);  

end  

function [headway, y] =  
calculateHeadwayWithStraightRoadAssumption(obj, local_map,  
local_map_left_lane_edge, local_map_right_lane_edge,  
local_map_yaw, lane_width, vehicle_width)  

% We want to find the first lateral deviation greater than our vehicle width  
ind = find(abs(local_map(2,:)) >= vehicle_width, 1);  

% If the road is veering to the left  
if ~isempty(ind) && local_map(2,ind) > 0 && numel(  
local_map_left_lane_edge(2,:)) == numel(unique(  
local_map_left_lane_edge(2,:)))
headway = \texttt{interp1} (\texttt{local\_map\_left\_lane\_edge} (2, :), \\
\texttt{local\_map\_left\_lane\_edge} (1, :), \texttt{lane\_width/2} + \\
\texttt{vehicle\_width}, 'linear', 'extrap'); \\
y = \texttt{lane\_width}/2;

elseif \texttt{isempty} (\texttt{ind}) && \texttt{local\_map} (2, \texttt{ind}) < 0 && \texttt{numel} (\texttt{local\_map\_right\_lane\_edge} (2, :)) == \texttt{numel} (\texttt{unique} (\texttt{local\_map\_right\_lane\_edge} (2, :)))

headway = \texttt{interp1} (\texttt{local\_map\_right\_lane\_edge} (2, :), \\
\texttt{local\_map\_right\_lane\_edge} (1, :), −\texttt{lane\_width/2}− \\
\texttt{vehicle\_width}, 'linear', 'extrap'); \\
y = −\texttt{lane\_width}/2;

end

if \texttt{isempty} (\texttt{ind}) 

if \texttt{ind} ~= 1 && \texttt{ind} ~= \texttt{size} (\texttt{local\_map}, 2)

% Check the data point in front to see if it falls within
% the straight road lane
if \texttt{abs} (\texttt{local\_map} (2, \texttt{ind} + 1)) < \texttt{lane\_width}/2

\texttt{two\_points} = \texttt{local\_map} (:, \texttt{ind}:\texttt{ind} + 1);

% Otherwise it's the previous data point
elseif \texttt{abs} (\texttt{local\_map} (2, \texttt{ind} − 1)) < \texttt{lane\_width}/2

\texttt{two\_points} = \texttt{local\_map} (:, \texttt{ind}−1:ind);

end

headway = interp1(two_points(2,:), two_points(1,:), lane_width/2, 'linear', 'extrap');

else

    headway = local_map(1, ind);

end

else

    headway = inf;

end

R = [cos(local_map_yaw(ind)) -sin(local_map_yaw(ind));
     sin(local_map_yaw(ind))  cos(local_map_yaw(ind))];

v = [0 1]';
s = [1 0]';

v2 = R * v;

test = (dot(v2, s) / dot(s, s)) * s;

if isempty(ind)

    headway = inf;

else

    headway = local_map(1, ind);% + test(1);

    headway = vehicle_edge_path(1, ind);

    headway = interp1(vehicle_edge_path(1,:), lane_width/2);

    headway = headway + test(1);
% straight_road_x = 0:0.1:max(local_map(1,:));
% straight_road_y = -1 * ones(1,length(straight_road_x));
% headway = interp1(straight_road_x,
% straight_road_y, local_map(2,ind));
%
end

if headway > obj.maximum_sighting_distance
    headway = inf;
end

end

function headway = calculateHeadwayWithMap(obj, local_map, 
vehicle_width)
    headway = 1;
end

function headway = calculateHeadwayWithMapAndCamera(obj, 
local_map, vehicle_width)
    headway = 1;
end

function all_headways = processFullTraversal(obj, data, 
config)
    if obj.show_live_plot == 1
        figure(9600)
    end

    all_headways = zeros(1, length(data.latitude));

    % Now we will step through each GPS point and calculate 
    % our headway
    for i = 1:length(data.latitude)
        % Make sure we are actually moving as our GPS-based
yaw goes crazy due
% to GPS measurement noise.
if data.U(i) > obj.minimum_longitudinal_velocity

% Grab the indices for data from our current
% station to some fixed
% lookahead distance. We want the lookahead
distance to be large enough
% that it doesn't influence our headway
calculation, but this will
% reduce the number of data points ahead of the
vehicle that we are
% carrying through the projection into body-fixed
coordinates.
station inds = find(data.station >= data.station(i) & data.station < data.station(i) + obj.
maximum_sighting_distance);
data_in lookahead range = [data.X(station inds);
data.Y(station inds); data.Z(station inds)];
local_map_yaw = data.yaw(station inds);

% Set our current vehicle position
% Input parameters are frame name (vehicle) and
% the X, Y, Z, roll,
% pitch, and yaw of the current vehicle pose.
obj.tf.setOffsetFromOrigin('vehicle', data.X(i),
data.Y(i), data.Z(i), 0, 0, data.yaw(i));

% Project GPS points from global to body-fixed
[local_x, local_y, local_z] = obj.tf.
transformToLocal('vehicle',
data_in lookahead range);
local_map = [local_x; local_y; local_z];

straight_road_decimation = 0.5;
straight_road_x = 0:straight_road_decimation:obj.
maximum_sighting_distance;
straight_road_map = [straight_road_x;
zeros(1, length(straight_road_x))];
zeros(1, length(straight_road.x));

straight_road.yaw = zeros(1, size(straight_road_map, 2));

[local_map_left_lane_edge, local_map_right_lane_edge] = obj.m.
createRoadBoundaries(local_map, local_map_yaw, config.lane_width);
[straight_road_left_lane_edge, straight_road_right_lane_edge] = obj.m.
createRoadBoundaries(straight_road_map, straight_road_yaw, config.lane_width);

if strcmp(config.method, "straight_road")
    [all_headways(i), y] = obj.
calculateHeadwayWithStraightRoadAssumption(local_map, local_map_left_lane_edge, local_map_right_lane_edge, local_map_yaw, config.lane_width, config.vehicle_width);
elseif strcmp(config.method, "with_map")
    all_headways(i) = headway.
calculateHeadwayWithMap(local_map, config.vehicle_width);
elseif strcmp(config.method, "with_map_and_camera")
    all_headways(i) = headway.
calculateHeadwayWithMapAndCamera(local_map, config.vehicle_width);
end

if obj.show_live_plot == 1
    if ~exist('h1', 'var') || ~isvalid(h1)
        h1 = plot(local_map(2,:), local_map(1,:), 'k');
        hold on
        h2 = plot(local_map_left_lane_edge(2,:),
    end
local_map_left_lane_edge(1,:), 'k--');
191 h3 = plot(local_map_right_lane_edge(2,:),
local_map_right_lane_edge(1,:), 'k--')
192 h4 = plot(straight_road_map(2,:),
straight_road_map(1,:), 'r');
193 h5 = plot(straight_road_left_lane_edge
(2,:), straight_road_left_lane_edge
(1,:), 'r--');
194 h6 = plot(straight_road_right_lane_edge
(2,:), straight_road_right_lane_edge
(1,:), 'r--');
195 h7 = plot(y, all_headways(i), 'ro', 'LineWidth', 5);
196 set(gca, 'xdir', 'reverse')
197 ylim([0 obj.maximum_sighting_distance])
198 xlabel('Local-y (m)')
199 ylabel('Local-x (m)')
200 legend([h1, h4, h7], 'Actual Road', 'Assumed Lane', 'Max Headway')
201 axis equal
202 grid on

else
205 set(h1, 'XData', local_map(2,:), 'YData',
local_map(1,:));
206 set(h2, 'XData', local_map_left_lane_edge
(2,:), 'YData',
local_map_left_lane_edge(1,:));
207 set(h3, 'XData',
local_map_right_lane_edge(2,:), 'YData',
local_map_right_lane_edge(1,:));
208 set(h4, 'XData', straight_road_map(2,:),
'YData', straight_road_map(1,:));
209 set(h5, 'XData',
straight_road_left_lane_edge(2,:), 'YData',
straight_road_left_lane_edge
(1,:));
210 set(h6, 'XData',

straight_road_right_lane_edge(2,:), 'YData', straight_road_right_lane_edge(1,:));
set(h7, 'XData', y, 'YData', all_headways(i));

end

pause(0.01)

end

else

all_headways(i) = NaN;

end

end

end

end

end

methods (Static)

end

end

classdef Camera < handle

% Class properties and variables
properties

K % Intrinsic matrix
R % Rotation matrix
T % Translation matrix
% Extrinsic matrix
E

% Projection matrix
P

% Homography matrix
H

px_horizontal
px_vertical

end

methods

% Constructor for class.
function obj = Camera(t_x, t_y, t_z, roll, pitch, yaw, focal_x, focal_y, skew, center_x, center_y)

    if nargin == 7 & isa(focal_x, 'struct')

        parameters = focal_x;
        focal_x = parameters.focal_x;
        focal_y = parameters.focal_y;
        skew = parameters.skew;
        center_x = parameters.center_x;
        center_y = parameters.center_y;

    end

    obj.P = projectionMatrix(obj, t_x, t_y, t_z, roll, pitch, yaw, focal_x, focal_y, skew, center_x, center_y);

end

function P = projectionMatrix(obj, t_x, t_y, t_z, roll, pitch, yaw, focal_x, focal_y, skew, center_x, center_y)

    % Create a homogeneous intrinsic parameter matrix
    obj.K = intrinsicMatrix(obj, focal_x, focal_y, skew, center_x, center_y);

    % Create extrinsic matrix
    obj.E = extrinsicMatrix(obj, t_x, t_y, t_z, roll, pitch, yaw);
% Create projection matrix
P = obj.K * obj.E;

obj.P = P;

end

function K = intrinsicMatrix(obj, focal_x, focal_y, skew, center_x, center_y)

% Create a homogeneous intrinsic parameter matrix
K = eye(4);
K(1,1) = focal_x;
K(2,2) = focal_y;
K(1,2) = skew;
K(1,3) = center_x;
K(2,3) = center_y;

obj.K = K;

end

function E = extrinsicMatrix(obj, t_x, t_y, t_z, roll, pitch, yaw)

% Create a rotation matrix that rotates the world frame to
% orient it with the camera frame through orthogonal rotations.
% World X -> Camera Z
% World Y -> - Camera X
% World Z -> - Camera Y

R_int = [0 -1 0 0; ...
        0 0 -1 0; ...
        1 0 0 0; ...
        0 0 0 1];

% Create the rotation matrices about the camera XYZ
% These are not your standard rotation matrices because we have already
% rotated the reference frame to the camera reference frame where
% Z points out of the image plane, X to the right, and Y down if you are
% looking directly through the camera view. Therefore, a vehicle rotation
% in pitch acts about the X direction in the camera coordinate frame.
% A vehicle rotation in yaw acts about the Y direction in the camera
% coordinate frame and finally, a vehicle rotation in yaw acts about the Z
% direction in the camera coordinate frame. The vehicle coordinate frame
% is the standard ISO coordinate frame with X−forward, Y−left, Z−up.

Rx = [1 0 0 0;
     0 cos(−pitch) sin(−pitch) 0;
     0 −sin(−pitch) cos(−pitch) 0;
     0 0 0 1];

Ry = [cos(−yaw) 0 −sin(−yaw) 0;
     0 1 0 0;
     sin(−yaw) 0 cos(−yaw) 0;
     0 0 0 1];

Rz = [cos(roll) sin(roll) 0 0;
     −sin(roll) cos(roll) 0 0;
     0 0 1 0;
     0 0 0 1];

% Concatenate the rotation matrices into one rotation matrix
obj.R = Rx * Ry * Rz * R_int;

% Create a homogeneous translation matrix. The translation values are
% negative because they represent the relative
translation of the
% world frame as if you are sitting in the camera
coordinate frame.
% The difference is though, we tend to measure the camera
height and
% orientation as positive values and specifying that a
camera is at
% +1.5 m seems to make more intuitive sense than at −1.5
m.

```
obj.T = eye(4);
obj.T(1,4) = -tx;
obj.T(2,4) = -ty;
obj.T(3,4) = -tz;
```

% Combining the rotations and translations, first by
% translating the point
% to the camera coordinate frame, then applying the
% orthogonal rotation
% matrix (R) to orient the world axes with that of the
camera coordinate
% frame, and then applying the slight vehicle (ISO)
% orientation changes
% directly from an IMU.

```
E = obj.R * obj.T;
```

```
obj.E = E;
```

```
function H = homographyMatrix(obj)

% For planar homography, we will remove the Z component
% of the
% projection matrix. This will form a 3x3 matrix.
H = [obj.P(1:3,1:2) obj.P(1:3,4)];
```

```
obj.H = H;
```

```
end
```
function pixels = projectToImage(obj, points, round_pixels)

    if size(points, 1) == 3
        points = [points; ones(1, size(points, 2))];
    end

    % Multiply by the projection matrix
    pixels = obj.P * points;

    % Divide by the third row to scale back into pixel coordinates
    pixels = pixels ./ repmat(pixels(3,:), size(pixels, 1), 1);

    % Return just the u, v coordinates (top two rows)
    pixels = pixels(1:2,:);

    % If we need to access exact pixel coordinates, we need to
    % round the measurements to the nearest integer. This can be
    % set as a flag (round_pixels). By default, the pixels are NOT
    % rounded.
    if nargin == 3 && round_pixels == 1
        pixels = round(pixels);
    end

end

function xy = projectFromImageToCameraFrame(obj, pixels, H)

    if nargin == 2

        % Set z = 0 planar homography
        if isempty(obj.H)
            obj.homographyMatrix();
        end

    elseif nargin == 3


obj.H = H;

end

if size(pixels,1) == 2
    pixels = [pixels; ones(1, size(pixels,2))];
end

xy = obj.H \ pixels;

% xy = xy . / xy(3, :);
% xy(3,: ) = 0;

delimiter

function H = createHomography(obj, X_pixels, Y_pixels, X_desired_pixels, Y_desired_pixels)

A = [X_pixels Y_pixels ones(size(X_pixels)) zeros(length(X_pixels),3) -X_pixels.*X_desired_pixels -Y_pixels.*X_desired_pixels ... zeros(length(X_pixels),3) X_pixels Y_pixels ones(size(X_pixels)) -X_pixels.*Y_desired_pixels -Y_pixels.*Y_desired_pixels];

A = reshape(A', 9, 2 * length(X_pixels) );

[U, D, ~] = svd(A' * A);

[~, ind_minimum_eigenvalue] = min(diag(D));

h = U(:, ind_minimum_eigenvalue);

h = h ./ h(end);

H = [h(1:3) ';
    h(4:6) ';
    h(7:9) '];

delimiter

end
function \([\text{px}_\text{horizontal}, \text{px}_\text{vertical}] = \text{createGrid}(\text{obj},\text{x}_\text{decimation}, \text{y}_\text{decimation}, \text{x}_\text{min}, \text{x}_\text{max}, \text{y}_\text{min}, \text{y}_\text{max})\]

\[
X = \text{x}_\text{min} : \text{x}_\text{decimation} : \text{x}_\text{max};
\]

\[
Y = \text{y}_\text{min} : \text{y}_\text{decimation} : \text{y}_\text{max};
\]

\[
\text{XYZ}_\text{horizontal} = [... \\
\text{reshape}([X; X], 1, [])]; ...
\]

\[
\text{repmat}([\text{y}_\text{min} \text{y}_\text{max}], 1, \text{length}(X));
\]

\[
\text{zeros}(1, 2 \times \text{length}(X));
\]

\[
\text{XYZ}_\text{vertical} = [... \\
\text{reshape}([Y; Y], 1, [])]; ...
\]

\[
\text{zeros}(1, 2 \times \text{length}(Y));
\]

\[
\text{px}_\text{horizontal} = \text{obj}.\text{projectToImage}(...); \text{px}_\text{vertical} = \text{obj}.\text{projectToImage}(...);
\]

Auxillary class for additional functionalities

classdef Utilities < handle

    properties

    end

    methods

        function \([\text{data}, \text{outlier_ranges}] = \text{removeOutliers}(\text{obj}, \text{data}, \text{max_standard_deviation}, \text{sigma})\]

        data_mean = \text{mean}(\text{data});

        data_std = \text{std}(\text{data});
data_outliers = find(abs(abs(data)-abs(data_mean)) >
max_standard_deviation_sigma * data_std);

outlier_ranges = [];

if isempty(data_outliers)
    outlier_edge_indices = find(diff(data_outliers) ~= 1) + 1;
    if isempty(outlier_edge_indices)
        outlier_edge_indices = 1;
        number_of_outlier_regions = 1;
    else
        number_of_outlier_regions = length(outlier_edge_indices) + 1;
    end
end

outlier_ranges = zeros(number_of_outlier_regions,2);

for i = 1:number_of_outlier_regions
    if i == 1
        if number_of_outlier_regions == 1
            outlier_ranges(i,:) = [data_outliers(1)
data_outliers(end)];
        else
            outlier_ranges(i,:) = [data_outliers(1)
data_outliers(outlier_edge_indices(1) -1)];
        end
    elseif i < number_of_outlier_regions
        outlier_ranges(i,:) = [data_outliers
outlier_edge_indices(i-1)) data_outliers(
outlier_edge_indices(i-1))];

else

outlier_ranges(i,:) = [data_outliers(
outlier_edge_indices(i-1)) data_outliers(
end)];

end

end

%% Replace the outliers with the average value on the values before and after the region
data = removeRangeOfData(obj,data,outlier_ranges);

end

function data = removeRangeOfData(obj,data,ranges)

if ~isempty(ranges)

for i = 1:size(ranges,1)

left_previous_ind = ranges(i,1)-1;
right_after_ind = ranges(i,2)+1;

if left_previous_ind < 1
left_previous_ind = 1;
end

if right_after_ind > length(data)
right_after_ind = length(data);
end

if ranges(i,2) ~= length(data)
data(left_previous_ind+1:right_after_ind−1) =
    round(mean(data([left_previous_ind
               right_after_ind])));

else

    data(ranges(i,2)) = data(ranges(i,2)−1);

end

end

end

end

function m = leastSquares(obj, X,Y)

    if rcond(X' * X) > 100 * eps

        m = X' * X \ X' * Y;

    else

        m = nan * ones(size(X,2),1);

    end

end

function [x, P] = recursiveLeastSquares(obj, x, y, P, lambda)

    k = inv(lambda)*P*x / (1 + inv(lambda)*x*P*x);
    e = y - x;
    x = x + k*e;
    P = inv(lambda) * P - inv(lambda)*k*x*P;

end

function station = calculateStation(X, Y, Z)
if nargin == 1
    ENU = X;
    X = ENU(:,1);
    Y = ENU(:,2);
    Z = ENU(:,3);
end

station = sqrt(diff(X).^2 + diff(Y).^2 + diff(Z).^2);
station = cumsum(station);
station = [0; station];
end
end
end
Appendix E

Script to illustrate ambiguous headway arising due to misorientation of ROI and the actual lane

```matlab
clear, clc
close all

font_size = 16;

headway = Headway();
headway.maximum_sighting_distance = 200;

m = Map({});

vehicle_width = 0.5;
lane_width = 2;

% Build the local map
decimation = 0.5; % meters
max_x = 15; % meters
local_map_x = 0:decimation:max_x;
local_map_y = 0.005 * local_map_x.^2;
local_map_z = 0 * local_map_x;
```
local_map = [local_map_x; local_map_y; local_map_z];
local_map_yaw = atan2(diff(local_map(2,:)), diff(local_map(1,:)));
local_map_yaw = [local_map_yaw local_map_yaw(end)];
[local_map_left_lane_edge, local_map_right_lane_edge] = m. createRoadBoundaries(local_map, local_map_yaw, lane_width);

straight_road_x = 0:decimation:max(local_map(1,:));
straight_road_map = [straight_road_x;
                   zeros(1, length(straight_road_x));
                   zeros(1, length(straight_road_x))];
straight_road_yaw = zeros(1, size(straight_road_map,2));
[straight_road_left_lane_edge, straight_road_right_lane_edge] = m. createRoadBoundaries(straight_road_map, straight_road_yaw, lane_width);

[headway_straight_road, y] = headway.
calculateHeadwayWithStraightRoadAssumption(local_map,
                                          local_map_left_lane_edge, local_map_right_lane_edge, local_map_yaw,
                                          lane_width, vehicle_width);
% headway_with_map = headway.calculateHeadwayWithMap(local_map,
% vehicle_width);
% headway_with_map_and_camera = headway.
calculateHeadwayWithMapAndCamera(local_map, vehicle_width);

figure
hold on
h1 = plot(local_map(2,:), local_map(1,:), 'k.');
plot(local_map_left_lane_edge(2,:), local_map_left_lane_edge(1,:), 'k')
plot(local_map_right_lane_edge(2,:), local_map_right_lane_edge(1,:), 'k')
h2 = plot(straight_road_map(2,:), straight_road_map(1,:), 'r.');
plot(straight_road_left_lane_edge(2,:), straight_road_left_lane_edge(1,:), 'r')
plot(straight_road_right_lane_edge(2,:), straight_road_right_lane_edge(1,:), 'r')
h3 = plot(y, headway_straight_road, 'ro');
set(gca,'xdir','reverse')
xlabel('Local−y (m)','FontSize',fontsize)
ylabel('Local−x (m)','FontSize',fontsize)
legend([h1, h2, h3], {'Actual Road', 'Assumed Lane', 'Max Headway'},'FontSize',fontsize)
set(gca,'FontSize',fontsize)
axis equal
grid on

saveas(gcf,'figures/headway_simulation','epsc')
Bibliography


91


