Northern Illinois University

Huskie Commons

Graduate Research Theses & Dissertations Graduate Research & Artistry

2020

Adaptive Object Detection for Autonomous Vehicles

Christopher Wolfe
zb191511@gmail.com

Follow this and additional works at: https://huskiecommons.lib.niu.edu/allgraduate-thesesdissertations

Part of the Electrical and Computer Engineering Commons, and the Robotics Commons

Recommended Citation

Wolfe, Christopher, "Adaptive Object Detection for Autonomous Vehicles" (2020). *Graduate Research Theses & Dissertations*. 7787.
https://huskiecommons.lib.niu.edu/allgraduate-thesesdissertations/7787

This Dissertation/Thesis is brought to you for free and open access by the Graduate Research & Artistry at Huskie Commons. It has been accepted for inclusion in Graduate Research Theses & Dissertations by an authorized administrator of Huskie Commons. For more information, please contact jschumacher@niu.edu.

ABSTRACT

ADAPTIVE OBJECT DETECTION FOR AUTONOMOUS VEHICLES

Christopher Wolfe, MS
Department of Electrical Engineering
Northern Illinois University, 2020
Hasan Ferdowsi, Director

Autonomous vehicles are gradually entering our daily lives. The goal of fully autonomous commercially available vehicles is becoming closer to reality each day as the contributions from researchers and various institutions are being added to the overall body of knowledge. Object detection is a critical component of an autonomous or semi-autonomous vehicle and draws extensively on results from many fields such as image processing and statistics. In this thesis, we consider ideas from the study of real-time computing and control systems to present a novel method of real-time adaptive object detection. We present a conceptual framework of the method as it applies to an automated vehicle control system. The application controls an object recognition detection sequence through using the aggregate channel features (ACF) detection algorithm. Our proposed method incorporates awareness of computational resources and feedback from the vehicle motion planner as inputs to the perception algorithm. We provide a complete model for analysis and simulation in MATLAB and Simulink environment. Experimental results are provided across a set of parameters, showing results consistent with the expectations in the proposed framework. The results show promising performance in the simulated scenario of highway driving on a straight road. Several possibilities for extension of the model are possible.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

DECEMBER 2020

ADAPTIVE OBJECT DETECTION FOR AUTONOMOUS VEHICLES

BY

CHRISTOPHER WOLFE
©2020 Christopher Wolfe

A THESIS SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE

MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

Thesis Director:
Hasan Ferdowsi

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor, Prof. Hasan Ferdowsi, for his guidance and patience as I learn to conduct research. I would also like to thank Prof. Mansour Tahernezhadi for his continued support and guidance throughout my academic journey. His emphasis on scientific participation and problem solving through reasoning is truly inspiring.

Additionally, I would like to thank my fellow graduate students in the Department of Electrical Engineering for their encouragement, and always bringing new ideas and inspirations to our research community.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

Chapter

1.BACKGROUND 1

Autonomous Vehicle Technology 1

Perception and Object Detection 2

Multi-Sensor Data Fusion 3

Vehicle Reaction Time 4

Visual Object Detection 6

Vehicle Controller Performance 7

Related Work 8

Reflection of Literature 10

Problem Statement and Objectives 11

2.CONCEPTUAL FRAMEWORK 13

Controller Model 13

Adaptive Techniques 14

Anytime Algorithms 14

Chapter Page

Resource-Bounded Algorithms 14

Evaluation of Algorithm Processing Time 15

Imaging Sequence Manipulation 15

Vision Detection Sequence 18

Automated Sequence Generation 21

Distribution of Secondary Detections 22

3.MODEL CONSTRUCTION 25

MATLAB and Simulink 25

Visual Object Detector 27

Object Tracking 29

Radar Simulation 30

Motion Planning 30

Low level Control 34

Vehicle Dynamics Modeling 35

Model Summary and Contributions 37

4.EXPERIMENT RESULTS AND REFLECTION 39

Chapter Page

Straight Road Scenario 39

Experiments I – III 40

Experiments IV – VI 44

Discussion 48

Future Work 50

Summary 51

REFERENCES 52

APPENDIX – MATLAB CODE 56

LIST OF TABLES

Page

Table 1: Table of Significant Works Related to Adaptive Object Detection 9

Table 2: Motion Planner State Logic Table 32

Table 3: Summary of Simulink Model Components 38

Table 4: Summary of Results.. 48

LIST OF FIGURES

Page

Figure 1: Elements of an autonomous vehicle control system 2

Figure 2: Sensor data fusion example 4

Figure 3: Vehicle controller conceptual model 13

Figure 4: Sequence of primary processes 16

Figure 5: Sequence of primary processes with secondary process 17

Figure 6: Vehicle with multiple sensors 18

Figure 7: Primary and secondary detection regions 19

Figure 8: Simulink model for simulation of autonomous vehicle control 26

Figure 9: Example of camera image with bounding box, as result of ACF object detector 28

Figure 10: Simulink model of vehicle perception 28

Figure 11: Simulink model for motion planning 33

Figure 12: Simulink model of low-level control 35

Figure 13: Simulink model for vehicle dynamics 37

Figure 14: Straight road highway driving scenario 39

Figure 15: Experiment I, II, and III sensor layout 40

Figure 16: Experiment I ($\beta = 0$, vision only) 41

Figure 17: Experiment II ($\beta = 0.2$, vision only) 42

Figure 18: Experiment III ($\beta = 0.3$, vision only) 43

Figure 19: Experiment IV, V, and VI sensor layout 44

Page

Figure 20: Experiment IV (β = 0.0, vision and radar) 45

Figure 21: Experiment V (β = 0.2, vision and radar) 46

Figure 22: Experiment VI (β = 0.3, vision and radar) 47

1. BACKGROUND

Autonomous Vehicle Technology

Modern automobiles are becoming increasingly automated. The U.S. Department of Transportation (DOT) has recently issued several publications addressing the growing need for research in Automated Driving Systems (ADS) [1][2]. These publications provide guidance for industry and nonprofit institutions in realizing the future of transportation. The agency recognizes that this technological progression can only be safely achieved through extensive research and validation of the proposed technologies. Development in ADS is expected to bring benefits to the public including reduction of traffic accidents, improved mobility for persons with disabilities, and increased economic output. At present, there are no ADS-dedicated (fully autonomous) commercial vehicles available. However, many existing vehicles already include semi-autonomous elements to improve vehicle safety [3].

Research institutions and many large technology companies worldwide are now participating in the study of autonomous vehicles [4], [5]. Vehicle autonomy overlaps significantly with the general study of mobile robotics; this in turn borrowing results from dynamic control, computer vision, artificial intelligence, and optimization, among others. Many research objectives can be carried out in laboratory environments and on test tracks using scaled representations of traffic environments. Accurate simulations also are increasingly valuable in the development of autonomous vehicles.

It should be noted that creating autonomous vehicles is a non-trivial task, even in a laboratory environment. Several hardware and software systems must operate concurrently and in real-time. Figure 1 illustrates the flow of information through the components of a typical autonomous vehicle control system:

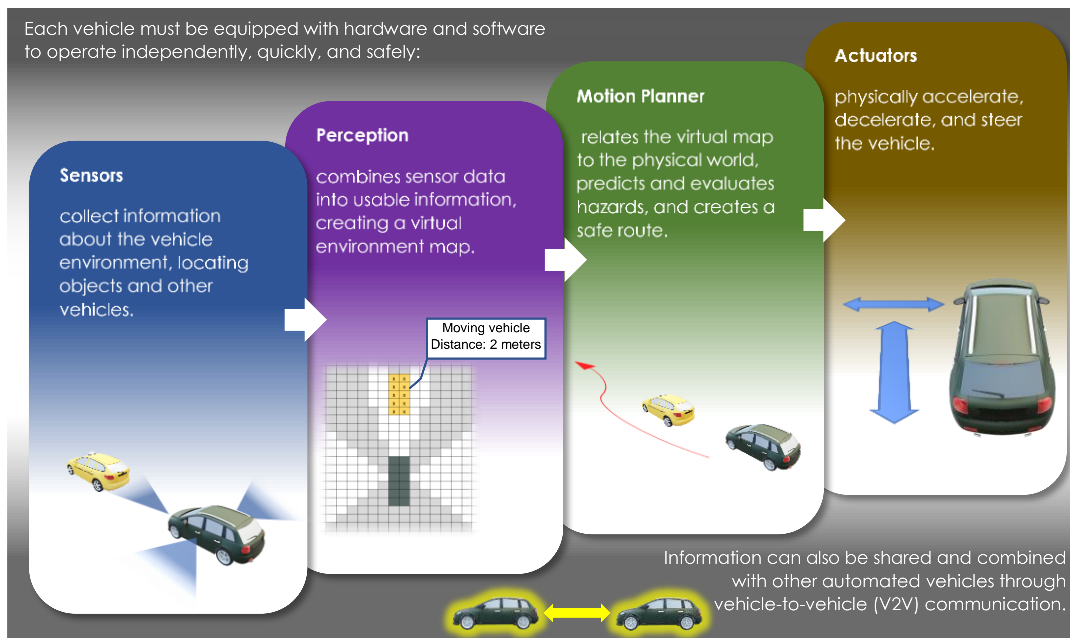


Figure 1: Elements of an autonomous vehicle control system

Perception and Object Detection

This research explores object detection, which is part of a vehicle's perception system. Object detection includes design and construction of an adequate sensor network, integration into the vehicle's real-time perception system, and appropriate handling of all data generated.

Object detection is critical in automated driving. The perceived environment must be a suitable representation of the physical environment, including the hazards and unknowns encountered by human drivers. The required system must be capable of measuring relative

distance, position, and size of objects, as well as heading and predicted position in the case of moving objects.

Many different types of sensors are useful in object detection. Examples are visual imaging (camera), laser range-finders (LiDAR), infrared imaging, ultrasonic, and radar [6]. Each of these rely on different measurement media that may perform better or worse depending on the vehicle's environment, especially considering various weather conditions.

Multi-Sensor Data Fusion

Research in object detection focuses on the use of multiple sensors (Figure 2). Practical sensors have limitations on useful range and coverage area. Because of this, multiple sensors are necessary to achieve adequate coverage under all foreseeable conditions, so consideration must be given to the sensor network design [7].

The presence of multiple sensors gives rise to another non-trivial problem: handling overlaps in sensor coverage. In this competitive configuration, the agreement or disagreement of two different sensors can be used to create a measure of confidence among individual measurements [8].

The preceding concepts all fall under the study of multi-sensor data fusion. There is no single unifying set of rules on how to handle the combination of sensor data. Instead, automated vehicle engineers must carefully justify the data-joining process through appropriate reasoning and supporting logic, hence the need for continued research.

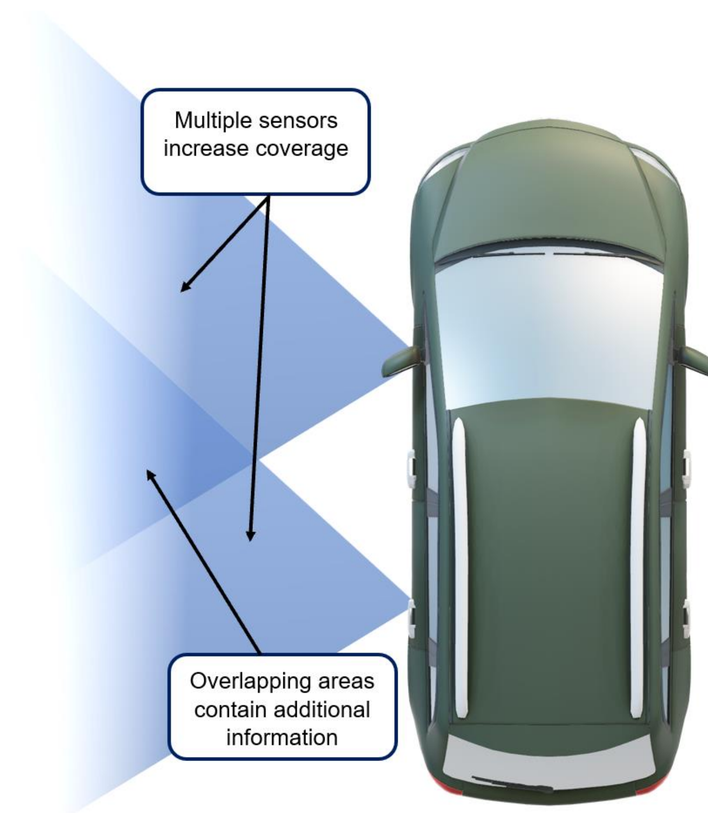


Figure 2: Sensor data fusion example

Vehicle Reaction Time

A 2019 study estimates human driver reaction time between 0.22 and 0.44 seconds to identify a road hazard, with roughly 0.2 additional seconds needed to decide how to respond [9]. This is analogous to autonomous vehicles – faster perception algorithms result in lower overall vehicle reaction time.

To illustrate, one can consider a basic kinematics problem. Suppose two physically identical vehicles are travelling side-by-side along a straight highway at a constant speed of 70 mph (31.29 m/s), and both drivers start to notice a road hazard at the same time. Let vehicle A

have reaction time 0.5 s and let vehicle B have reaction time 0.2 s. After their respective reaction times have passed, the vehicles brake with maximum deceleration such that the tires do not slip.

Let us compare the stopping distance of the two vehicles. To determine the total stopping distance for each vehicle, we need to model this scenario and solve an initial value problem (IVP) with one or more kinematic equations of rigid body motion which could reasonably depend on the following parameters:

- mu – coefficient of static friction between road and tire
- Cd – longitudinal air drag coefficient of the vehicles
- m – mass of vehicle as a single rigid body

Certainly, each of these parameters receives considerable attention in the mechanical design of the vehicles. However, it is easy to see that the IVP has the same solution for both vehicles if defined to begin at moment the physical braking mechanism begins. Before braking, the physical parameters are irrelevant since the velocity is constant. Thus, if we only wish to compare the stopping distance between these two vehicles, we do not need to solve the IVP, and the relative stopping distance is independent of the vehicle physics. In the scenario considered, we can simply multiply each vehicle's velocity and reaction time:

For vehicle A, we have 31.3 m/s * 0.5 s ≈ 15.6 m.

For vehicle B, we have 31.3 m/s * 0.3 s ≈ 9.4 m.

Subtracting the two quantities, we see the difference in stopping distance strictly due to reaction time is six meters – roughly the length of a full-size truck. The conclusion here from this simple problem is that there is no substitute for good vehicle reaction time – any seemingly small

improvements in reaction time can directly improve stopping distance regardless of the vehicle's mechanical properties.

Visual Object Detection

Researchers in object detection classify objects based on the methods used to successfully identify them, as object detectors may be well suited only for certain types of objects. Many early attempts at visual object recognition were motivated by the difficult problems of facial recognition and terrain identification. While many facets of image processing have advanced significantly in recent decades, this initial research has seeded the methods that we use today in autonomous systems.

In 1973, Fischler and Elschager proposed an image-matching algorithm that detects objects based on the success or value of expected feature detection in the presence of noise using learned data [10]. Despite initially being used in facial recognition, this is one of the earliest examples of an algorithm that has the basic structure of modern object detection methods useful in the context of autonomous vehicles: statistical image processing based on feature finding with a trained reference data set.

Object detection, like all other digital processing, had made the most significant advancements through computational power and storage of the past few decades. However, the key improvements are based in machine learning methods, such as convolutional neural networks (CNN). In 1995, LeCun and Bengio apply CNN to image processing in the well-known article in [11]. Recently, region-based convolutional neural networks (R-CNN) have become an increasingly popular method for machine learning in object detection as used in the widely-known Fast-R-CNN and Faster-R-CNN [12] detectors. The popular YOLO object detector was designed

for real-time detection performance and uses a simpler network compared to R-CNN [13]. Despite these advancements, it is still widely accepted that occluded objects remain a difficult problem in object detection. In 2017, Wang et al provide an adversarial alternative to R-CNN that is much better suited for occluded objects [14]. Also, in 2017, Dollar et. al introduce the method of fast feature pyramids with the aggregate channel features (ACF) object detector [15].

Vehicle Controller Performance

Modern vehicles depend on robust hardware and software called real-time systems where the controlling processor must be capable of performing time-critical tasks. This type of system is already a requirement for non-autonomous vehicles, and is also found in avionics and industrial control [16], [17].

A familiar example of this type of system is anti-lock brakes (ABS). Several sensors monitor the vehicle's motion and the speed of the driving wheels. The electrical control unit (ECU) collects data and based on a programmed control law drives a valve and pump to limit the braking torque delivered to the wheels. To achieve a safe and reliable response, this entire process must be completed in a few milliseconds, repeatedly and without interruption [18].

In autonomous vehicles, the required hardware and software demands are far greater, especially considering the complexity of visual object detection. However, we can combine some results from related areas. We next look at some existing work that has results useful to investigating the computational aspects of object detection.

Related Work

Here we consider a selection of published work involving the general study of visual object detection, and robot/vehicle localization. In particular, the papers chosen here demonstrate an adaptive component based on feedback of a computation time.

In 1994, Kelly [19] describes an approach to vehicle perception described as a throughput problem, considering the tradeoffs between the perception algorithm complexity and safe vehicle speed as a resulting increase or decrease in processing time, measured in flops. In 2003, Kwok, et al. present a localization algorithm that reduces the number of used samples in real-time based on computational load [20]. In this case, the computational load is measured as a percent total available, and performance of several load capacities are given. In 2013, Kim et al. provide a framework for vehicle system management focusing mainly on graceful degradation. Additionally, the framework proposes automatic selection of sensors based on the situation. In 2016, Lu et al. propose an adaptive object method not specific to autonomous driving, but clearly useful for real-time visual object detection in autonomous vehicles. The method extends on the region-based proposals such as those of the popular Fast R-CNN, by training a network to consider specific areas for zoom and nearby regions of existing objects in forming proposal regions.

In Table 1, some of the developments and challenges are identified. We find that several key ideas developed in these methods can be applied to adaptive object detection for autonomous vehicles.

Table 1: Table of Significant Works Related to Adaptive Object Detection

Article	Original Application	Major Developments	Challenges and possible extension	Key Adaptive Elements
Kelly, 1994 "*Adaptive perception for autonomous vehicles*" [19]	Environment mapping for mobile robots with laser rangefinder	Discussion of general throughput problem (tradeoffs). Determines maximum safe vehicle speed as a function of resources	Limited computational power available in 1994. Application limited to simple mobile robot.	Algorithm tracks its own cycle time and adjusts complexity accordingly
Kwok et. al, 2003 "*Adaptive real-time particle filters for robot localization*" [20]	Robot localization with laser rangefinder	Complete statistical model provided. Identifies optimum computational power for least localization error	Mixes sample sets due to time-varying environment characteristics.	Sample size adjusted in real-time to optimize computational efficiency
Kim et al., 2013 "*Towards Dependable Autonomous Driving Vehicles:* *A System-Level Approach*"[21]	Fault recovery and sensor management for automated driving	Complete framework for adaptive graceful degradation. Software library for process scheduling timing feedback.	Complete selection scheme for sensor modalities needs development.	Feedback from recent task cycle time leading to appropriate vehicle action
Lu et al., 2016 "*Adaptive Object Detection Using Adjacency and Zoom Prediction*"[22]	General object detection	Model is trained and adapted to identify important regions of interest.	Region proposals based only on prior image data. Can be expanded to additional prior knowledge.	Computational resources focused on image area of interest

In addition to the key works above, there are continual developments both inside and outside the realm of autonomous driving. More recently, a 2017 patent describes an adaptive object detection system intended for detecting and tracking faces with mobile devices in real-time [23]. The system manages computational resources by selecting images from a queue which will undergo the computationally expensive detection algorithm. Although this is not intended for use in automated vehicles, it illustrates a need for management of computational resources in the general problem of object detection and tracking. In a 2018 dissertation, Merfels describes a localization system that maintains a constant computational time for the algorithm, by adjusting the number of hidden nodes in a probabilistic graph [24]. This is done to prevent situations where the algorithm cannot keep up with the incoming sensor data as other algorithms are sharing the same processor. In this case, a PID controller is used to regulate deviations in computation time by adjusting algorithm complexity.

Reflection of Literature

While there are many sources in literature for embedded systems with an awareness of available computational resources, the availability of complete frameworks unifying adaptive object detection in the context of autonomous vehicles is lacking. Thus, there are opportunities for further research in the framework surrounding these systems. Based on the challenges and key ideas from Table 1, we arrive at the following questions:

1. Can a general framework be developed for adaptive object detection considering computational resources and sensor selection?
2. Can the method be applied to an existing object recognition algorithm without modification?

3. How can the method incorporate feedback from the rest of the vehicle control system?

Each of these questions can be their own topic of study. However, with the growing complexity of autonomous vehicles and the increased interest and participation of researchers, we can see the significance of a framework unifying the adaptive techniques with the vehicle control system model that is applicable to a modern autonomous vehicle. Our work here focuses on this need.

Problem Statement and Objectives

We consider the ideas present in the previous related literature, and the general concepts mentioned in the introduction to formulate a complete model to represent a modern automated driving system. In this thesis, we seek the following objectives:

1. Design a framework to optimize a real-time perception system, by managing algorithm execution and sensor selection, determining system parameters, and observing computational time.
2. Focus on a practical, real-time implementation in a simulated environment.
3. Determine the resulting performance and tradeoffs given the above variations.

For the first objective, we focus our attention to the concepts described in the study of and control systems and real-time embedded systems. Chapter 2 is dedicated to the construction of this framework. Here we discuss a detailed vehicle controller model space and a temporal model of the vehicle algorithm processing and task scheduling. We present an algorithm for automated detection sequence generation based on *a priori* knowledge of the object recognition processing time.

Regarding the second objective, we present a complete vehicle and environment model in Chapter 3 built in the MATLAB® and Simulink® software environment. Where necessary, we describe the background and mathematical framework for each component of the model.

Finally, for the third objective, we assess the feasibility of the presented framework by introducing performance metrics for the detection algorithm and run multiple simulations in the Simulink model. In Chapter 4, we see the numerical results and comparison for the simulations.

2. CONCEPTUAL FRAMEWORK

Controller Model

Controls engineers often think of systems in terms of a controller model where inputs and outputs are physical devices. Figure 3 shows an autonomous vehicle system described with such a model.

Figure 3: Vehicle controller conceptual model

Inputs to the control system of an autonomous vehicle are commonly visible light imaging (camera), laser imaging (LIDAR), radar, or acoustic (sonar) form the inputs to an autonomous vehicle. The handling of sensor input data is called perception, which includes all image processing elements. The next stage, motion planning, consists of localization, motion behavior, and environment mapping. The general vehicle trajectory is supplied by the motion planner to the low-level control system, consisting of lateral and longitudinal control. Outputs to the system are steering, throttle, and brake.

As most of the vehicle control system is implemented in dedicated devices, we next turn our attention to the effects of realizing vehicle control in hardware and software.

Adaptive Techniques

The discussion so far shows that individual perception component processing time is additive and not constant. Algorithms have long been proposed to have different behavior or results depending on the actual elapsed algorithm running time. We consider some classes of these algorithms in the following sections.

Anytime Algorithms

There are several ways to incorporate algorithm processing time in the design of real-time systems. One subclass of resource-aware algorithm is called anytime algorithms. In Anytime algorithms, the quality of results generally improves as a function of computation time [25].

In the context of autonomous vehicles, Anytime algorithms have been successfully developed for tree-search motion planning algorithms such as D* and RRT* [26]. In the case of Anytime motion planning, it is easy to imagine why Anytime algorithms are useful: there may be many feasible solutions to reach the desired target, but finding the optimal solution is not as critical as finding a feasible solution quickly.

Resource-Bounded Algorithms

Extending this idea to a feedback control system, algorithm parameters can be adjusted in real-time such that computational load is maintained to some level. In 2005, Thrun, et al. present such algorithms as resource-adaptive in the context of robot localization. [27]

In other parts of the vehicle control system, there may be no output available until the algorithm is completely finished processing. In this case, an Anytime algorithm is not useful. However, we can instead consider a collection of processes and examine their contributions to

the system both in process results and process time. An abstract treatment of resource-bounded algorithms is presented in [28].

Evaluation of Algorithm Processing Time

Algorithm performance and efficiency has historically been considered in the computed number of floating-point operations per second (FLOPS). However, complicated processes such as those used in computer vision or autonomous vehicles may be difficult or impossible to measure due to the large number of system states available. Additionally, system resources may not be completely dedicated to the task under consideration, and the computation time could further vary depending on other system task requirements.

Instead, we can estimate process time by recording the system clock value before the process (τ_{start}) and after the process (τ_{end}), and subtracting the difference:

$$\tau = \tau_{end} - \tau_{start}.$$

Further, we can record a finite quantity of the computed values τ to a list in memory. Taking the mean of several recent process times can be used as a predictor for future values.

Imaging Sequence Manipulation

Suppose a priori knowledge of image processing time is available for four different inputs, A, B,C, and D, such that each are predicted to finish in 25 ms. Then, assuming each sensor is processed sequentially, we have the following sequence, with total processing time of 100 ms as illustrated in Figure 4.

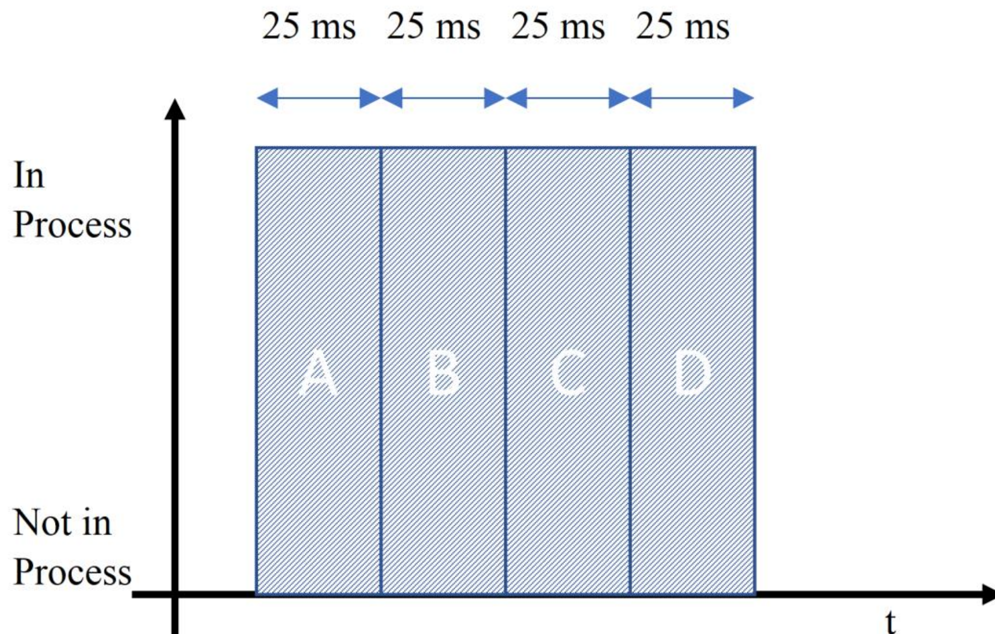


Figure 4: Sequence of primary processes

At this point, the system integrator must assess the predicted figures with regards to the overall system design. The 100 ms value implies that each sensor will be processed periodically every 100 ms. If this period is too high, the designer must find a way to free up resources elsewhere or revise the image processing algorithm. If the value is acceptable, then there may be a small excess time available. We seek to make use of these small time segments.

Suppose for example, that 125 ms is acceptable for the full processing sequence. Then we may choose how to allocate an additional 25 ms. This is illustrated in Figure 5.

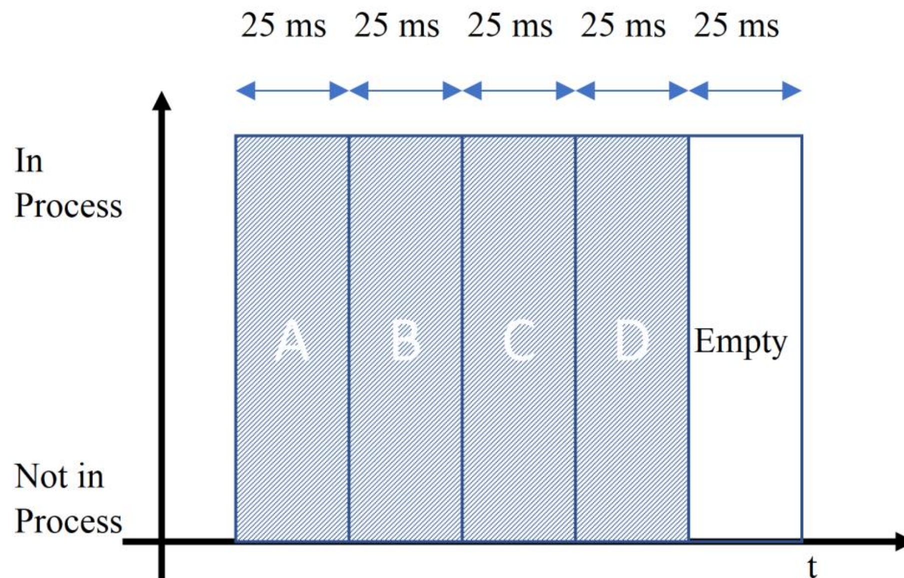


Figure 5: Sequence of primary processes with secondary process

The questions the system designer may ask at this point are:

1. What is the best use of free processing time in different machine states?
2. What is an acceptable upper bound for the processing sequence?

In the context of autonomous vehicles, both of these questions may not have an obvious answer as they are two variables in already complicated system. However, in the following section, we formulate an algorithm to determine the time allocation automatically, and we experimentally see results different values of upper bounds in Chapter 4. Ultimately, the system designer must accept a lower detection frequency of certain sensors while others are raised.

Vision Detection Sequence

Consider a vehicle equipped with multiple imaging sensors and denote each with a letter, as shown in the example in Figure 6.

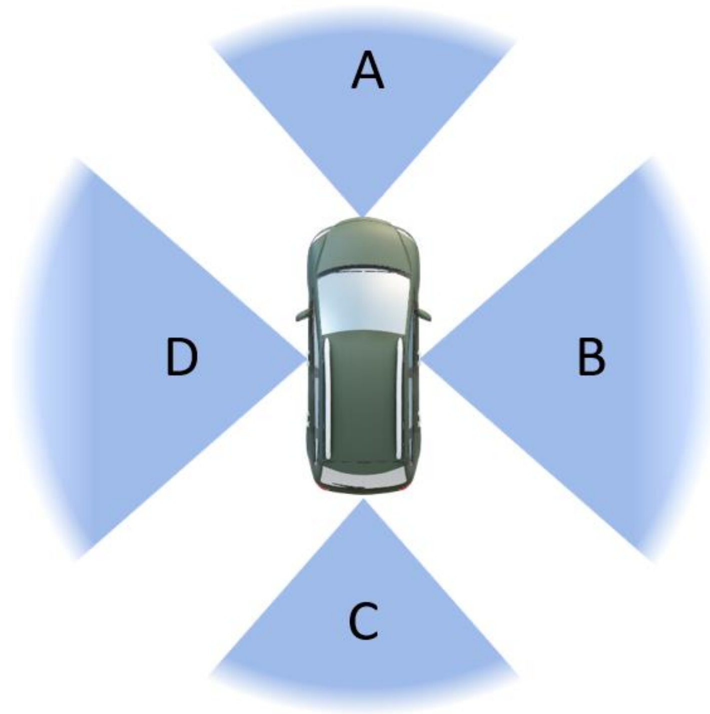


Figure 6: Vehicle with multiple sensors

Now consider a detection coming from one of the sensors. Define the complete detected image as the primary detection and define a subset of the image the secondary detection. Denote the primary detection by an uppercase letter, and the secondary detection by a lowercase letter.

Figure 7 shows an example of a forward-facing sensor image with primary and secondary regions.

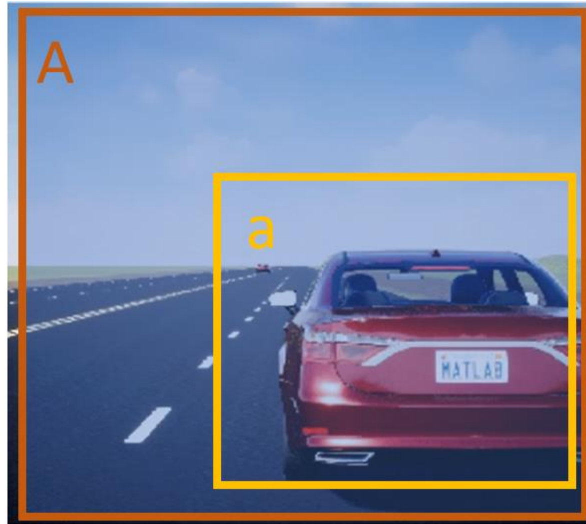


Figure 7: Primary and secondary detection regions

We can then define a sequence of n primary detections as entries from the set of primary or secondary detections, where the sequence proceeds from left to right. For example, the sequences $s_1 = \{A, B, C, D\}$ or $s_2 = \{a, b, c, d\}$ visit each of the four sensors listed in Figure 6. We can also combine primary detections such as in the sequence $s_3 = \{A, a, B, b\}$.

Ideally, the detections should be fast (multiple times per second). Because of this, we may look at the infinite sequence consisting of combined frames, or elementary sequences. For example, the infinite sequence

$$s_4 = \left\{ \overbrace{A, a, B, b}^{frame\ 0}, \overbrace{A, a, B, b}^{frame\ 1}, \cdots \right\}$$

can be thought of as successive frames of s_3.

Additionally, the detections from each sensor should occur at regular intervals, meaning variance among the time between detections should be minimized.

Define the detection period t_i of sensor i as the difference between the two consecutive detections (primary or secondary) originating from the same sensor.

Further define the average detection period $\overline{t_i}$ for sensor i as

$$\overline{t_i} = \frac{1}{n}\sum_{k=1}^{n} t_{i_k}$$

and the detection time variance σ_i^2 as

$$\sigma_i^2 = \frac{1}{n-1}\sum_{k=1}^{n}\left(t_{i_k} - \overline{t_i}\right)^2$$

where n samples are taken over many frames. We can extend these definitions to infinite sequences by taking the limit as n approaches infinity.

For example, if the time between each element of the sequence in s_4 is τ, then for sensor A, we have a detection period 1τ within each frame (from A to a), and a detection period 3τ of between frames (from a to A in the next frame). This yields the sequence of time intervals $\{1\tau, 3\tau, 1\tau, 3\tau, \cdots\}$ and the average detection period is

$$\overline{t_i} = \lim_{n\to\infty}\left(\frac{1}{n}\sum_{k=1}^{n} t_{i_k}\right)$$

$$= \lim_{n\to\infty}\frac{1}{n}(1\tau + 3\tau + 1\tau + 3\tau + \cdots)$$

$$= \lim_{n\to\infty}\frac{2n\tau}{n}$$

$$= 2\tau\ .$$

The detection time variance is

$$\sigma_i^2 = \lim_{n\to\infty} \frac{1}{n-1} \sum_{k=1}^{n} \left(t_{i_k} - \overline{t_i}\right)^2$$

$$= \lim_{n\to\infty} \frac{1}{n-1} \sum_{k=1}^{n} \left(t_{i_k} - 2\tau\right)^2$$

$$= \lim_{n\to\infty} \frac{1}{n-1} n\tau^2$$

$$= \tau^2.$$

We can reduce the variance by rearranging the sequence. If we instead use sequence $\{A, B, a, b\}$ for each frame, we have the infinite sequence $\{A, B, a, b, A, B, a, b, \cdots\}$ and sensor A (and also B) has detection intervals $\{2\tau, 2\tau, 2\tau, 2\tau, \cdots\}$. In this arrangement, we still have detection period $\overline{t_i} = 2\tau$, but the variance σ_i^2 has been reduced to zero. In practice, there will be small variations in the sensor acquisition and system processing time regardless of whatever sequence we use, so σ_i^2 can never be completely reduced.

Automated Sequence Generation

We can automate the sequence creation process given an arbitrary number of sensors and secondary detections. Assume that each sensor must include its primary detection once in each frame. Then let m_p = quantity of primary detections per frame = quantity of sensors and let m_s be the number of secondary detections per frame. Let β be the desired upper bound for frame processing time. We use the word “desired” here as the resulting processing time may not exactly match the calculated time when the sequence is executed. We can estimate the time needed for

running the primary or secondary detections by looking at prior data as described in previous sections. Assume for now that these estimated values are available as inputs to the system.

Let $\overline{t_1}$ represent the estimated time for a primary detection and $\overline{t_2}$ represent the estimated time for a secondary detection. From this, we can determine the amount of available time, t_{free}, by

$$t_{free} = max(\beta - m_p\overline{t_1}, 0)$$

And the number of available partitions or "slots" m_{free} by:

$$m_{free} = floor\left(\frac{t_{free}}{\overline{t_2}}\right) + 1$$

Note the max function in the definition of t_{free} is to handle the possibility of no available time. In this case we cannot run any secondary detections and must force the system to run a sequence consisting of only primary sensor process.

Distribution of Secondary Detections

Now we must handle the distribution of secondary sequences. Recall in the earlier example, $\{A, B, a, b\}$ outperformed $\{A, a, B, b\}$ in terms of detection time variance as the measurements were more evenly distributed. We can extend this idea to arbitrary numbers of sensors by defining a permutation (or arrangement of values) from 1 to n such that the numbers are visited non-sequentially. By letting $n = m_p$ we can create a permutation to distribute secondary detections in between primary detections with lowered detection time variance. Let $s_p = \{1, 2, \cdots, m_p\}$. Then define a permutation w_p as

$$w_p = \left\{x_i \middle| x_i = \begin{cases} s_i & i\ even \\ s_i + c & i\ odd \end{cases},\ c = ceil(m_p/2), i \in s_p\right\}.$$

This is just one scheme for permutation, but others are possible, such as no permutation, or simply reversing the entries. To illustrate the effect of w_p, the sequence $\{1, 2, 3, 4, 5, 6\}$ can be permuted by $w_p = \{1, 4, 2, 5, 3, 6\}$.

Now define the *origin sequence* s_0 by:

$$s_0 = \left\{x_i \middle| x_i = ceil\left(\frac{i\, m_s}{m_{free}}\right) + 1, i \in \left\{1, 2, \cdots, m_{free}\right\}\right\}.$$

and define *permuted sequence* s_w by

$$s_w = \left\{x_i \middle| x_i = s_{0_{j}}, j = mod\left(w_{p_i}, m_p\right), i \in \left\{1,, \cdots, m_{free}\right\}\right\}.$$

The origin sequence is necessary for when the number of free slots exceeds the number of secondary detections. For example, suppose for sensors A through E we have $m_{free} = 10$, $m_p = 5$, and $m_s = 3$. This results in the origin sequence

$$s_0 = \{1\ ,1\ ,1\ ,2, 2, 2, 3, 3, 3, 3\}$$

and the permutation $w_p = \{1, 4, 2, 5, 3\}$, we assign them to the sensors according to sensor number in order given by w_p. This can be represented by the augmented matrix with

$$\left[\begin{array}{c|cc} 1 & 1 & 2 \\ 2 & 2 & 3 \\ 3 & 1 & 3 \\ 4 & 2 & 3 \\ 5 & 1 & 3 \end{array}\right].$$

Replacing the numerical values with the alphabetic sensor designations, we have:

$$\begin{bmatrix} A & a & b \\ B & a & c \\ C & b & c \\ D & a & b \\ E & a & c \end{bmatrix}.$$

Reshaping this matrix into a row vector of size $m_p + m_{free}$, we have:

$$[A\ a\ b\ B\ a\ c\ C\ b\ c\ D\ a\ b\ E\ a\ c].$$

We have reached a solution for the detection distribution where the primary detections are evenly distributed, and the secondary detections are distributed throughout the primary distributions. Note it is not always possible to perfectly distribute all detections, and even under an ideal distribution may not exactly follow the expected frame execution times. Additionally, this method does not necessarily provide an optimal distribution for sequences. However, it is easy to see that by rearranging the sequence, there will be reduction in mean detection time and detection time variance for a given sensor (for example ‘A’ and ‘a’ being of the same sensor).

3. MODEL CONSTRUCTION

MATLAB and Simulink

MATLAB (MathWorks, Natick, MA) is a software package for numerical calculation [29]. Widely used in education and research, the software has many built in functions for use in statistics, automatic control, and image processing, making it an attractive choice for autonomous systems. Recently, MATLAB has introduced the Automated Driving Toolbox specifically for the study of autonomous and semi-autonomous vehicles [30].

Simulink is a component of MATLAB designed for modeling and control of dynamical systems [31]. Simulink projects are created in the form of “models”, which are edited using a very convenient graphical user interface similar to control flow diagrams.

According to the conceptual model discussed in Chapter 2, we now introduce the Simulink model shown in Figure 8. In the model, we represent each major component of the vehicle control system with a Simulink subsystem. The plant to be manipulated (vehicle state) is considered in the vehicle dynamics modelling. The vehicle interacts in the simulated environment and is updated in the next simulation cycle. The simulated environment is merely a visual rendering using the Unreal Engine for purposes of visual object detection, which includes the road, ground, sky, and other vehicles. Each sensor input sensor is implemented with an individual block that generates sensor data based on the defined properties such as sensor type, orientation, and update frequency.

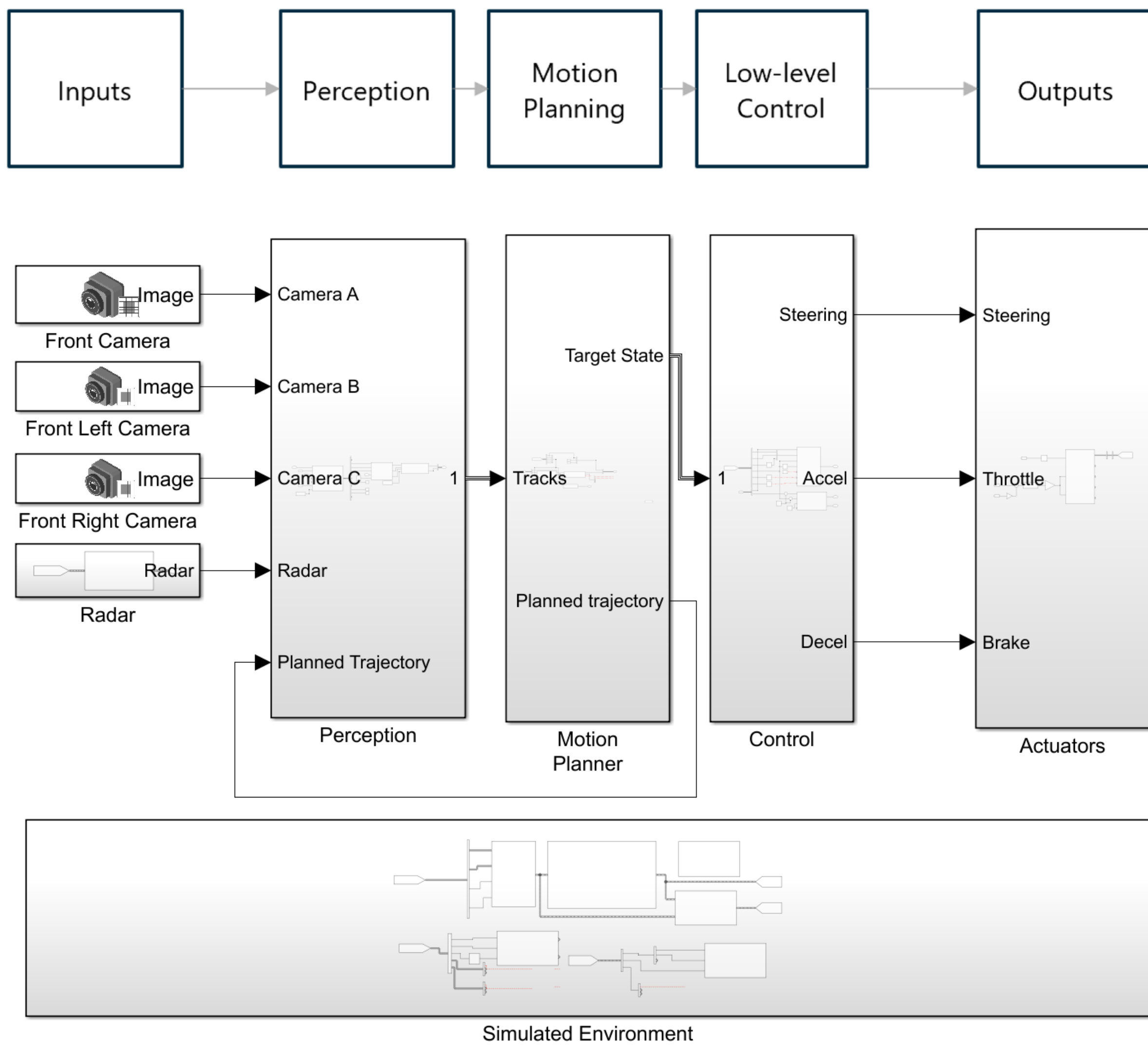


Figure 8: Simulink model for simulation of autonomous vehicle control

Note that the interconnections shown in Figure 8 are not exhaustive. In particular, vehicle localization and odometry are assumed available at all times. In an actual vehicle, localization and odometry are estimates of the true values using dedicated sensors. In our model, localization and odometry are available by the position and velocity components of the bus signal *BusEgoPos* providing values that are always accurate to the simulation.

Additionally, special attention is given to the feedback loop between the motion planning and perception subsections. A novelty of this model is the introduction of direct feedback from the motion planner module so as to provide the regions of interest for the adaptive detection module with resource awareness. We now present a detailed discussion of each subsystem.

Visual Object Detector

Vehicle detection from RGB camera images can be achieved according to the methods described in [32] and [15]. For a given image, several image channels are computed based on histograms of oriented gradients. Feature pyramids are constructed based on the channel information, allowing the algorithm to detect multiple scales without rescaling the original image. The detector compares the aggregated channel features (ACF) from the image to that of a known (training) dataset.

The MATLAB Computer Vision Toolbox includes an implementation of the ACF object detector instantiated by function vehicleDetectorACF(). The detector is pre-trained with images of actual vehicles to detect vehicle features. After instantiation, the detector is called with detect() function and returns a list of bounding box coordinates and confidence levels. An example image with bounding box is shown in Figure 9.

Figure 9: Example of camera image with bounding box, as result of ACF object detector

Using the bounding box information, we can estimate vehicle position by using our knowledge the camera orientation, width, and focal length. The MATLAB function, imageToVehicle() can be used to convert the bounding box edges to vehicle coordinates [33]. The Simulink model for the perception module is shown in Figure 10.

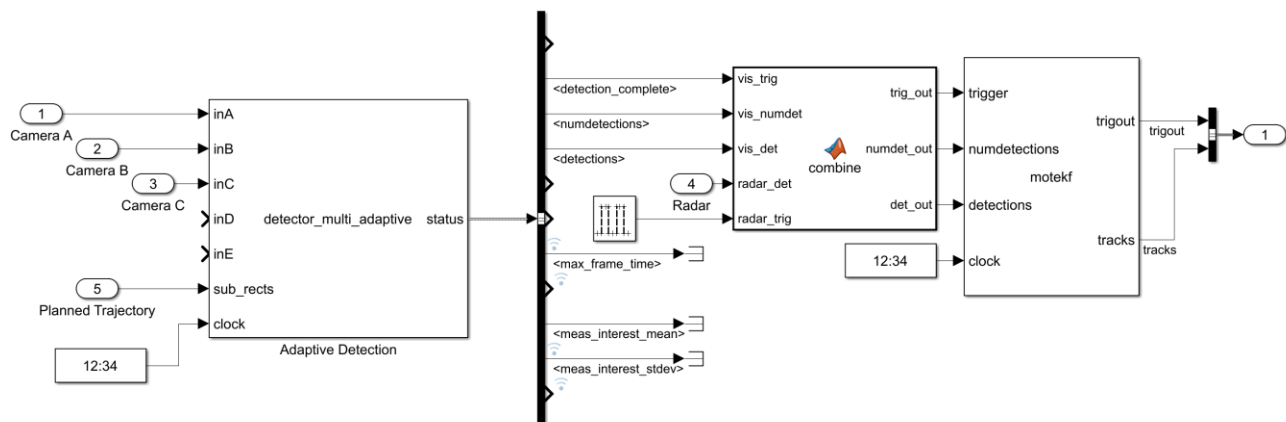


Figure 10: Simulink model of vehicle perception

Object Tracking

In our model, we are initially considering only other vehicles detected with vision sensors, which means we can strictly use visual bounding boxes as input data to the next stage in the vehicle perception. The bounding boxes are expected to have considerable variation, occasional missing measurements, and double measurements (multiple bounding boxes for a single object). Thus, we need an object tracking module to assess the state of the environment by keeping track of candidate and confirmed detected objects. The Global Nearest Neighbor (GNN) object detection method is based on the Munkres assignment problem described in [34]. The GNN method is shown to be superior in correctly tracking multiple overlapping targets [35].

The MATLAB Automated Driving Toolbox includes an implementation of GNN tracking that includes object state estimation with Kalman Filtering [36]. The function multiObjectTracker() allows for a state estimation function handle. Here we use the extended Kalman filter for its feasibilities in nonlinear state space models described by [37]:

$$\vec{x}_{k+1} = f(k, x_k) + \vec{w}_k \text{ (state transition function)}$$

$$\vec{y}_{k+1} = h(k, x_k) + \vec{v}_k \text{ (measurement function)}$$

where f, g are nonlinear functions of the current state x_k, and w_k and v_k are white Guassian noise describing the measurement and state transition process noise, respectively. In this case, the state vectors have four elements $[x, \dot{x}, y, \dot{y}]$ which will be the resulting form of our tracker output.

In our model, the object tracking function is encapsulated as a MATLAB System block we call *motekf* for "multi object tracker with extended Kalman filter". This block simplifies the

object tracker results into a five column matrix, the first column being the confirmed track ID, and the four remaining columns as the elements of the vector $[x, \dot{x}, y, \dot{y}]$ describing the tracked object state.

Radar Simulation

We include an optional radar simulation. Radar detections are generated with the Simulink block *Radar Detection Generator* [38]. This block models radar detections with simulated noise, false positives, specific locations of vehicle edges, and sensor properties. The detection frequency can also be adjusted. We fuse the vision and radar data using a custom MATLAB function called *combine*, which considers measurements from the bus output of the radar detection generator, and from the matrix output of the adaptive vision detector. The two trigger sources are also combined, updating the object tracker when either type of detection is ready.

Motion Planning

The vehicle motion planner is responsible for assessing the operating state of the vehicle and determining the necessary actions. The motion planner curve layer generates trajectories based on the vehicle's knowledge of the roadway geometry and the behavioral layer calculates the appropriate reference (setpoint) values to send to the motion planner. To make these assessments, the motion planner must relate objects detected in the ego vehicle's frame of reference F_E to the world frame of reference F_O. Let R be the rotation matrix defined by

$$\boldsymbol{R} = \begin{bmatrix} \sin(\psi) & cos(\psi) \\ cos(\psi) & -\sin(\psi) \end{bmatrix}$$

Where ψ is the angle of the vehicle with respect to the x-axis.

Now let $d_p = [x, y]^T$ and $d_v = [\dot{x}, \dot{y}]^T$ be the ego vehicle position and velocity in world coordinates. Further, let T_p be the position transformation matrix and T_v as the velocity transformation matrix, defined as:

$$\boldsymbol{T}_p = \begin{bmatrix} R & d_p \\ 0_{2\times1} & 1 \end{bmatrix} \text{ and}$$

$$\boldsymbol{T}_v = \begin{bmatrix} R & d_v \\ 0_{2\times1} & 1 \end{bmatrix}.$$

Then, a point $s^E = \left[s_x, s_y, 1\right]^T$ in the ego frame F_O is transformed to a point s^O in the world frame by

$$s^O = \boldsymbol{T}_p \, s^E.$$

Similarly, a velocity $v^E = \left[v_x, v_y, 1\right]^T$ in the ego frame F_E is transformed to velocity v^O in the world frame by

$$v^O = \boldsymbol{T}_p \, v^E.$$

Applying these transformations to the incoming tracked objects presents their complete state in world coordinates. These transformations are implemented in the model as a MATLAB Function block *tracks_to_world*.

In the curve layer, the motion planner generates a trajectory curve based on the current state of the vehicle (localization), desired pose of the vehicle (destination), and curvature of the road in world coordinates. For a straight road, this is simply a line positioned in between the lane nines.

The behavioral layer contains the vehicle decision-making process. In our model, the ego vehicle must respond to objects observed to be in its path. In the case of multiple objects detected

in lane, the behavioral layer determines which is the most important object (MIO), defined as the object closest to the ego vehicle. For our model, the behavioral layer functions similarly to Adaptive Cruise Control (ACC) found on modern autonomous and many commercial semi-autonomous vehicles [39]. In the implementation in [40], the module depends on velocity control and obeys a set of fuzzy membership rules for the position control. In our model, we use a simplified implementation with state transition logic, as shown in Table 2.

Table 2: Motion Planner State Logic Table

Present State	Transition Logic	Next State
-1 (Decelerate)	Maximum distance reached	0
0 (Hold)	Min or max. distance reached	-1 or 1
1 (Accelerate)	Minimum distance reached	0
2 (Full Stop)	None	2
Any	Followed vehicle velocity drops below threshold	2

Additionally, the motion planner includes a trajectory output for direct connection to the perception module. In the case of a straight road scenario it is represented by a matrix of constants defining a straight-ahead reduced detection window in the object recognition system. Our Simulink model for motion planning is shown in Figure 11.

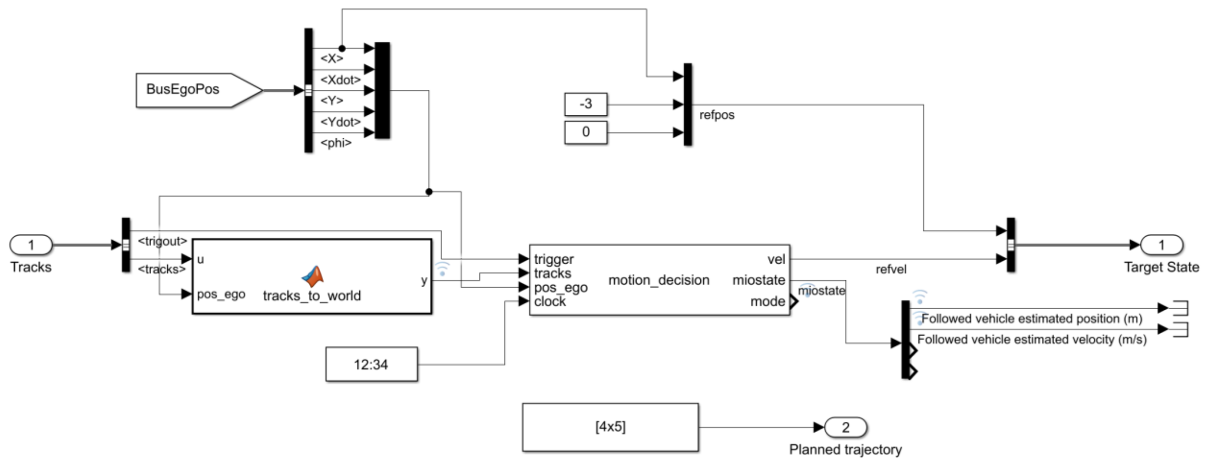


Figure 11: Simulink model for motion planning

Low level Control

Low level lateral control is accomplished with the Stanley method as described by Hoffman et al. in [41]. The method was named after the Stanford Racing Team's "Stanley" vehicle used by in the DARPA Grand Challenge 2005. In this method, the steering control law for the wheel angle $\delta(t)$ is given by

$$\delta(t) = \left(\psi(t) - \psi_{ss}(t)\right) + \tan^{-1}\frac{ke(t)}{k_{soft} + v(t)}$$

$$+ k_{d,yaw}\left(r_{meas} - r_{traj}\right) + k_{d,steer}(\delta_{meas}(i) - \delta_{meas}(i+1)).$$

where $\psi(t)$ is yaw angle $\psi_{ss}(t)$ is steady state yaw angle (which depends on vehicle mass m and tire stiffness C_y). The constant k is determined experimentally, k_{soft} tuned to low speed performance, and gains $k_{d,yaw}$, $k_{d,steer}$ are yaw rate feedback gain, and steering angle feedback gain. Included in MATLAB is an implementation of the Stanley lateral control. The MATLAB default values of $k_{d,yaw} = k_{d,steer} = 0.1$ are used, with m and C_y matched to the kinematic model properties (discussed in the next section).

For longitudinal control, the Stanley method is a PI controller with integrated anti-windup on the integral term. Included in MATLAB is an implementation of the Stanley longitudinal control. The MATLAB default values of $k_p = 2.5$, and $k_i = 1$ are used. This implementation also limits the acceleration and deceleration to adjustable maximum values. The default values of 3 m/s^2 and 6 m/s^2 are chosen, respectively.

The Simulink model for low level control is shown in Figure 12.

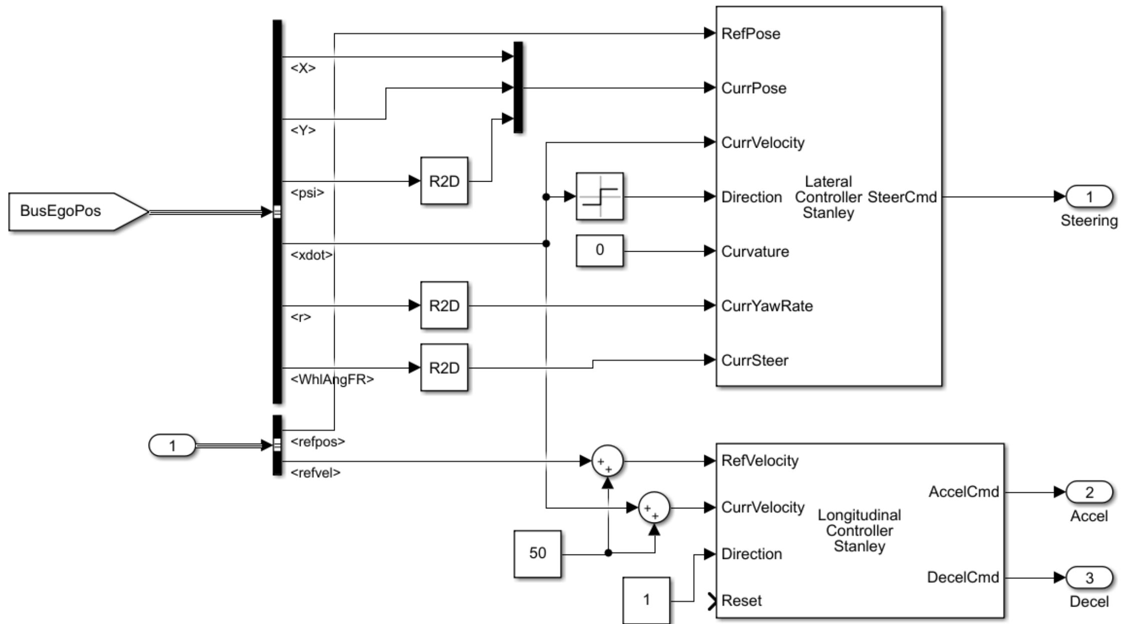


Figure 12: Simulink model of low-level control

Vehicle Dynamics Modeling

In our simulation, the vehicle physics model should accurately represent real-world vehicle dynamics. As is commonly used in the study of automated driving, we use the bicycle model [42] which relates the vehicle x-axis state in world coordinates, $\dot{X}(t)$, y-axis state in world coordinates $\dot{Y}(t)$, vehicle heading angle $\psi(t)$, side-slip angle β, and velocity v in the body frame of reference, with respect to the vehicle center of gravity.

There are two well-known implementations of bicycle model – kinematic and dynamic. The kinematic bicycle model assumes the vehicle direction of motion is along the same axis as the front wheels, which is suitable for low-speed driving. The dynamic bicycle model operates

without this assumption, instead accounting for inertial dynamics. This is important at highway speeds as the angle assumption can no longer be made[43].

The dynamic bicycle model is described by the following system of differential equations [44]:

$$\begin{cases} \dot{x} = \dot{\psi}\dot{y} + a_x \\ y = -\dot{\psi}\dot{x} = \psi\dot{y} + \frac{2}{m}(F_{c,f}\cos\delta_f + F_{c,r}) \\ \ddot{\psi} = \frac{2}{I_z}\left(l_f F_{c,f} - l_r F_{c,r}\right) \\ \dot{X} = \dot{x}\cos\psi - \dot{y}\cos\psi \\ \dot{Y} = \dot{x}\sin\psi - \dot{y}\cos\psi \end{cases}$$

Where m is the vehicle mass, I_z is yaw inertia. $F_{c,f}$ and $F_{c,r}$ are the lateral tire forces at the front and rear wheels, respectively.

The MATLAB Autonomous Driving Toolbox includes an implementation of the dynamic bicycle model. We use default values for the block parameters, described by [45].

In the simulated model, we use the “Vehicle Body 3DOF” block with force calculated by the product of vehicle mass and acceleration (Newton’s second law). To avoid discontinuities caused by abrupt changes in input, the incoming acceleration signal is dampened slightly by a filter with transfer function:

$$H(s) = \frac{0.5}{0.5s+1}.$$

The Simulink model for vehicle dynamics is shown in Figure 13.

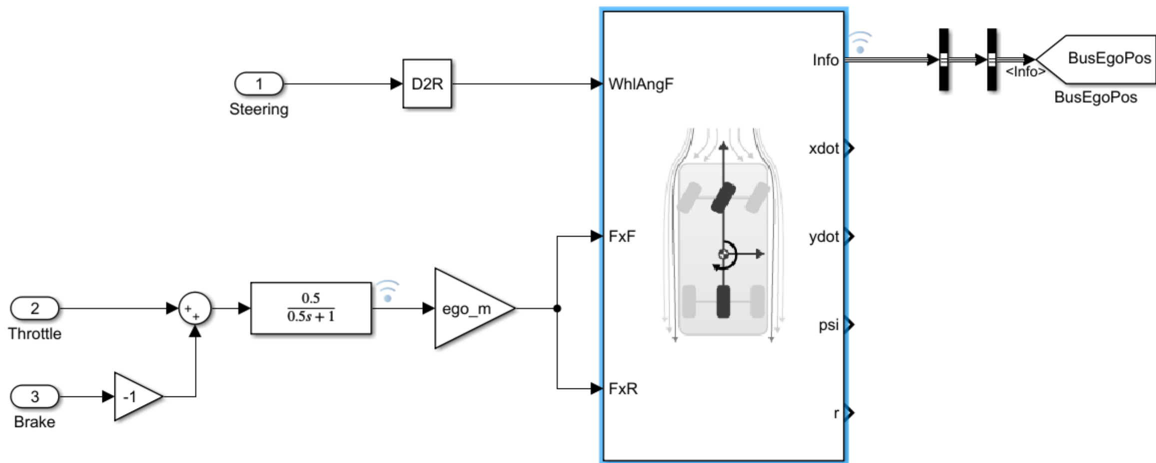


Figure 13: Simulink model for vehicle dynamics

We choose the Simulink environment sampled at a fixed period of $T_s = 10\ ms$. The sampling rate for all Simulink blocks is also set to T_s. The simulation ends after 20 s. Automatic solver selection is used for numerical solutions of differential equations as described in the preceding sections.

Model Summary and Contributions

The completed model uses a combination of premade components and original components with the goal of being simple yet flexible enough to be extended to several scenarios. Table 3 summarizes the main components of each subsystem and lists our original contributions to the model to distinguish the blocks and functions provided with MATLAB. For our developed components, complete MATLAB code for each component is provided in the appendix.

Table 3: Summary of Simulink Model Components

Subsystem	**MATLAB Provided Components**	**Developed Components**
Perception	Functions used in *Detector*_multi_adaptive: vehicleDetectorACF(), imageToVehicle() Functions used in *motekf* block: multiObjectTracker()	*Detector_multi_adaptive* block (sequence generation algorithm, image preparation, process time evaluation) *combine* function *Motekf* block: (encapsulation only)
Motion Planning	None	*tracks_to_world* block (coordinate transformations) *motion_decision* block (behavioral layer and velocity output)
Low-Level Control	Stanley lateral and longitudinal control blocks	None
Actuators/Dynamics	Bicycle model block	None

4. EXPERIMENT RESULTS AND REFLECTION

Straight Road Scenario

In this scenario, we consider an automated vehicle following another vehicle as in Figure 14. Images from the cameras are used to detect and track other vehicles. The relative position and velocities of the tracked objects are estimated. Using this estimated information, we can control the ego vehicle motion based on the observed environment. All simulations are performed on an Intel i7-8650U processor with 16GB RAM and nVIDIA GeForce GTX 1060 graphics. The GPU is only used for the scenario rendering. It is not used in the detection algorithm. Measured detection times are scaled (multiplied) by a factor of 0.2 to closer represent an on-board vision processing system.

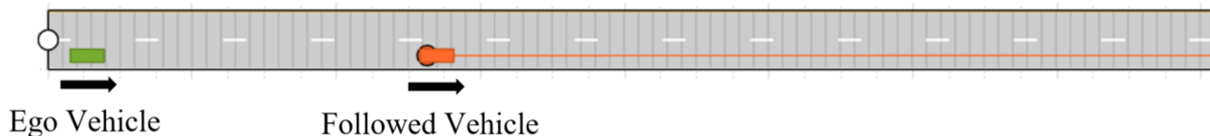


Figure 14: Straight road highway driving scenario

The scenario starts with both vehicles cruising at 29 m/s (65 mph). The autonomous vehicle is using adaptive cruise control (ACC). After some time has passed, the followed vehicle rapidly decelerates and comes to a complete stop. The automated vehicle must respond to the slowdown and decelerate to avoid a crash. We define the reaction time in all experiments as the time it takes between the moment where the followed vehicle reaches 27 m/s, and the ego vehicle estimating the velocity at 27 m/s (and therefore responding).

Experiments I – III

The automated vehicle is equipped with three front facing sensors at angles of -30, 0 and +30 degrees with respect to the vehicle's longitudinal axis as illustrated in Figure 15. We assess the performance of the adaptive perception system by comparing the vehicle response under different configurations of sequence generation. First, for experiment I, we let $\beta = 0$, which is more accurate to say we wish β minimized. In this case, the sequence is not adaptive. We simply process the frames in order of sensor designation: A-B-C. We then increase β to 0.2 and 0.3 in Experiments II and III, respectively. There are only vision detections considered in experiments I through III. Figure 16, Figure 17, and Figure 18 show the results of experiments I, II, and III, respectively. The top plot compares the estimated and true positions of the vehicles. The middle plot compares the estimated and true positions of the vehicles. The bottom plot shows the measured sequence length vs. the desired sequence length.

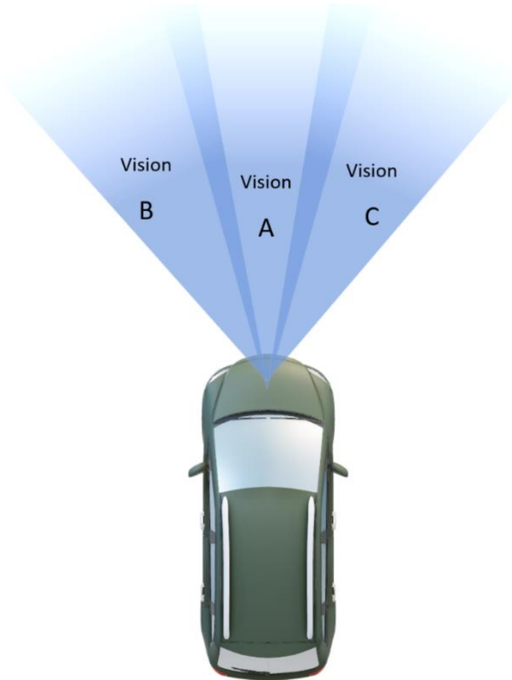


Figure 15: Experiment I, II, and III sensor layout

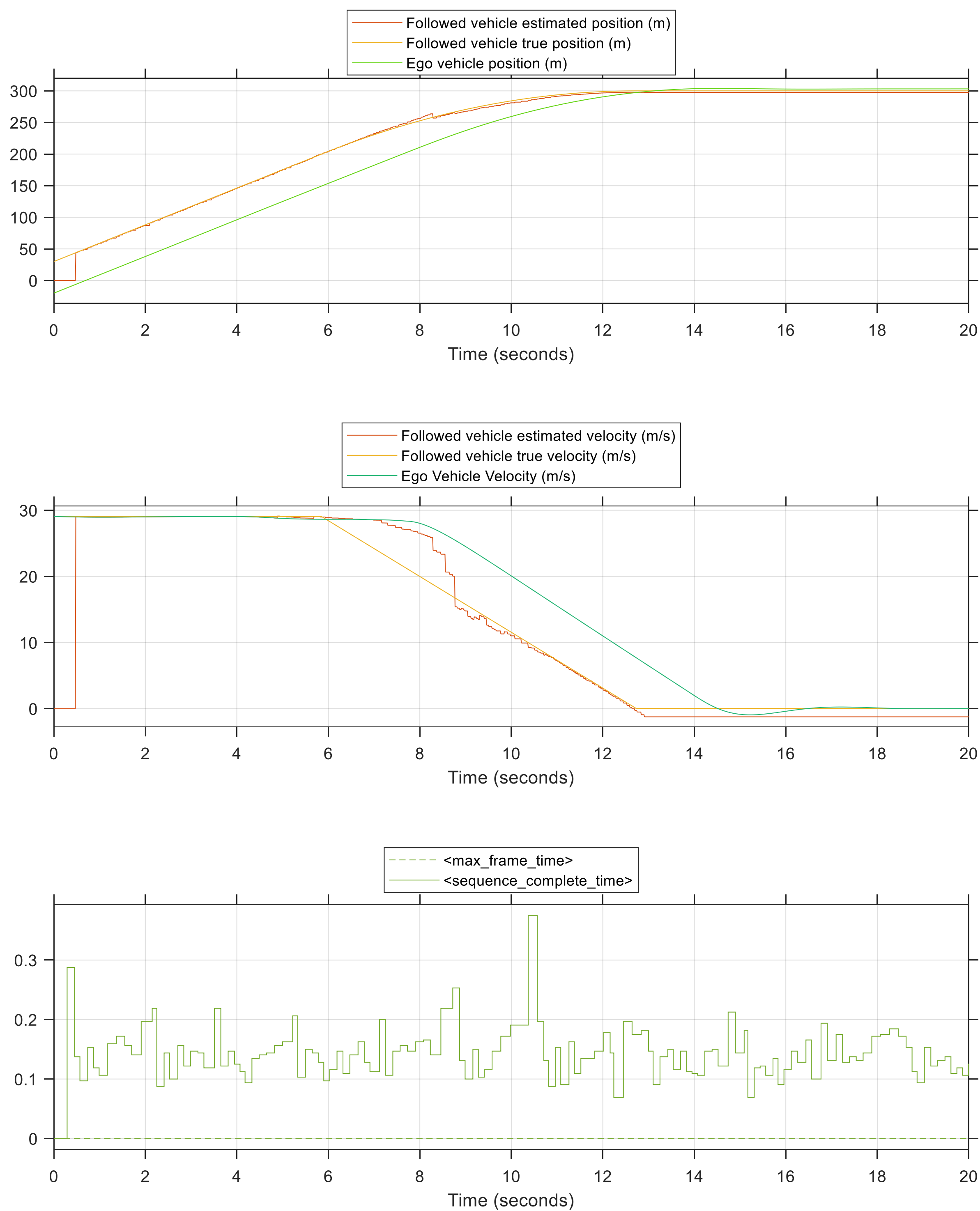


Figure 16: Experiment I (β = 0, vision only)

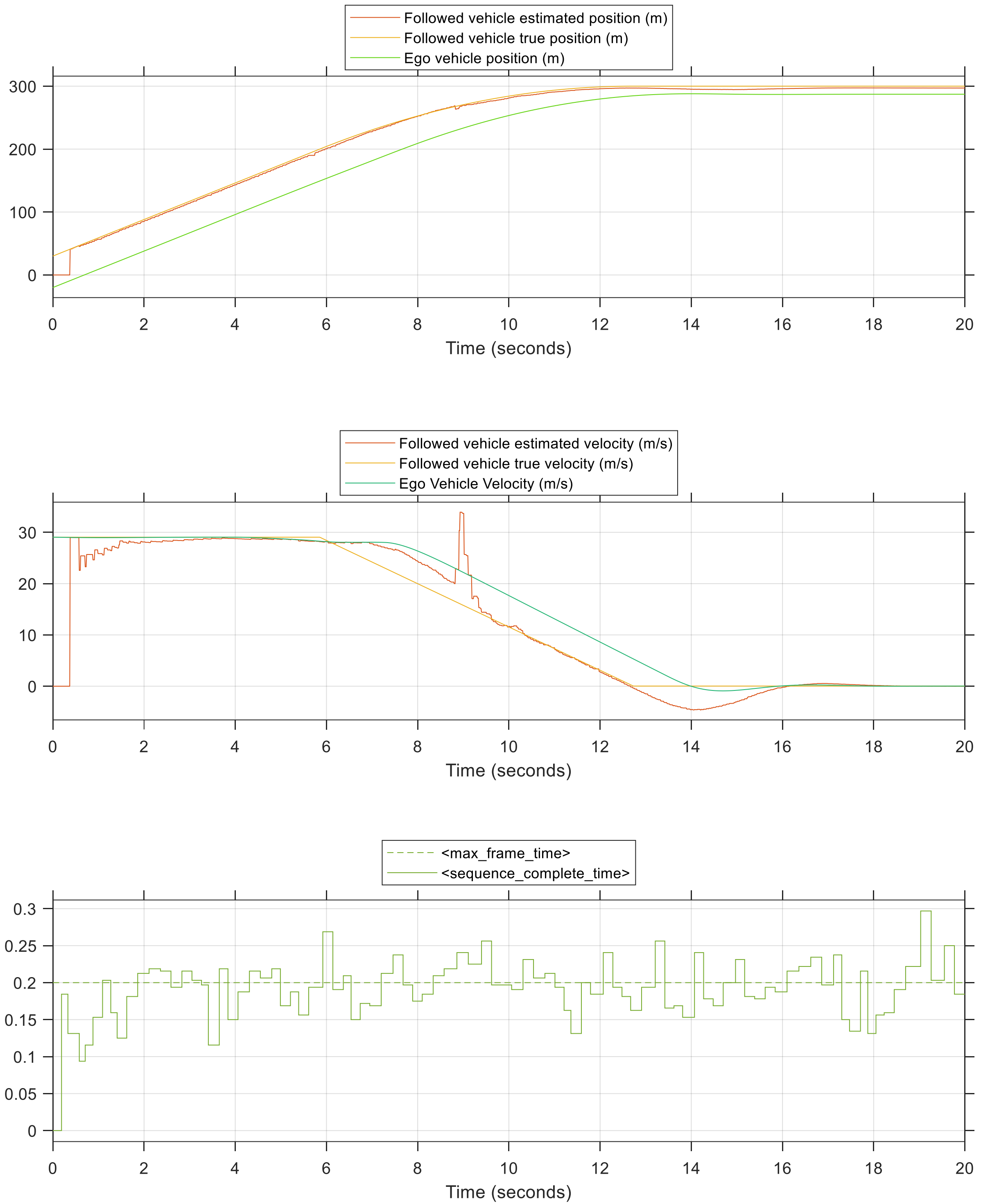
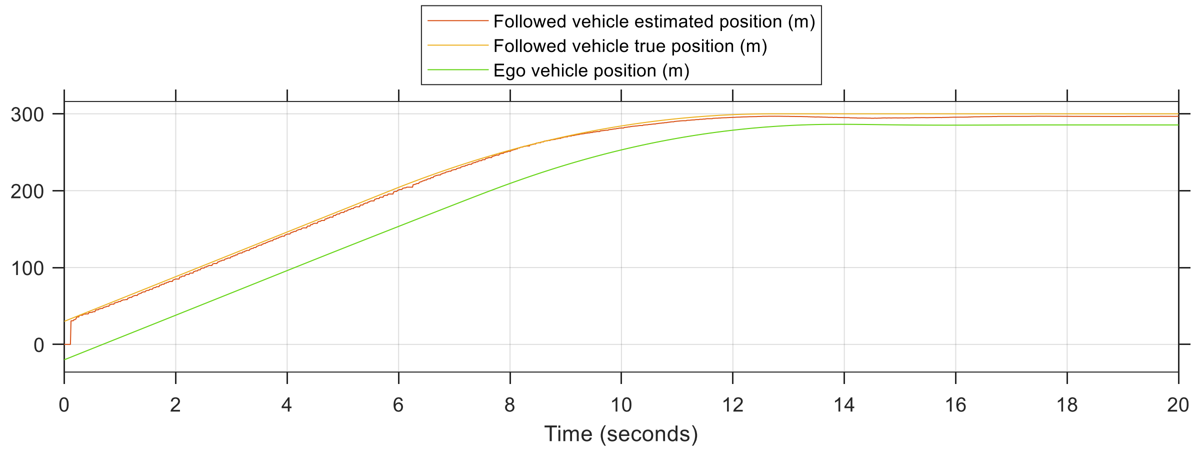
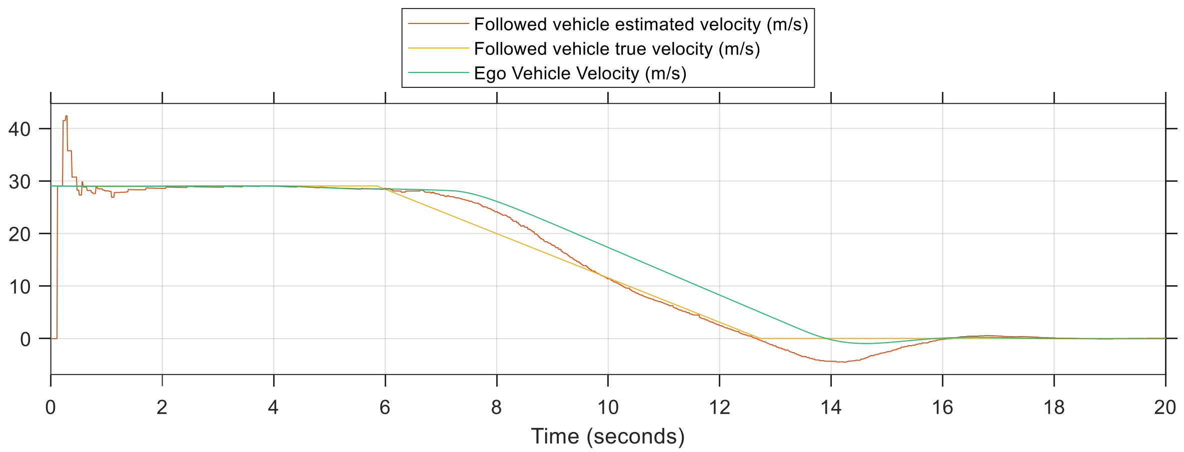


Figure 17: Experiment II (β = 0.2, vision only)





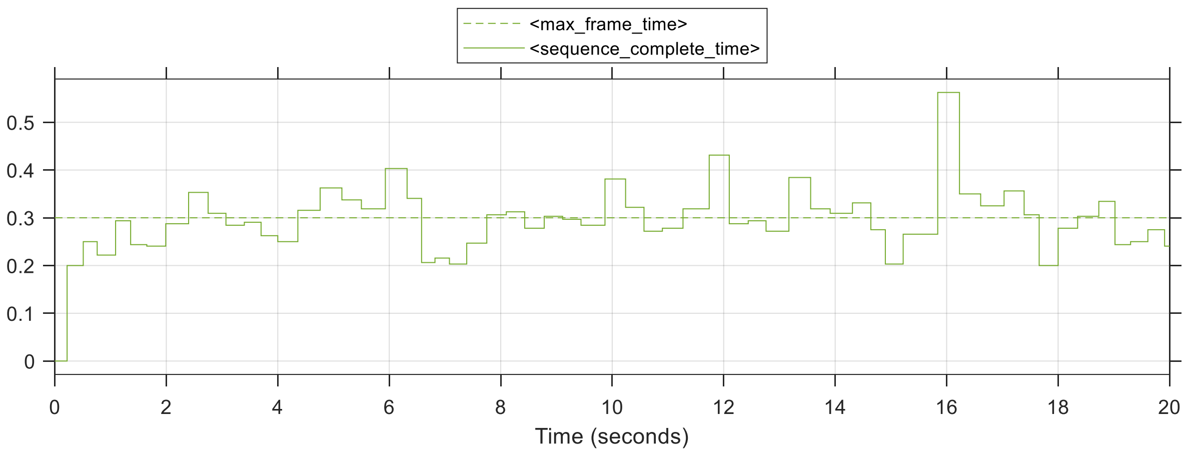


Figure 18: Experiment III (β = 0.3, vision only)

Experiments IV – VI

We repeat the previous three experiments with the three vision sensors, but this time we add a simulated radar sensor to the front of the vehicle (Figure 19). The radar sensor operates at 10 Hz and has a 20 degree detection angle. Again we start with β=0, and increase β to 0.2 and 0.3. Figure 20, Figure 21, and Figure 22 show the results of experiments IV, V, and VI, respectively. The top plot compares the estimated and true positions of the vehicles. The middle plot compares the estimated and true positions of the vehicles. The bottom plot shows the measured sequence length vs. the desired sequence length.

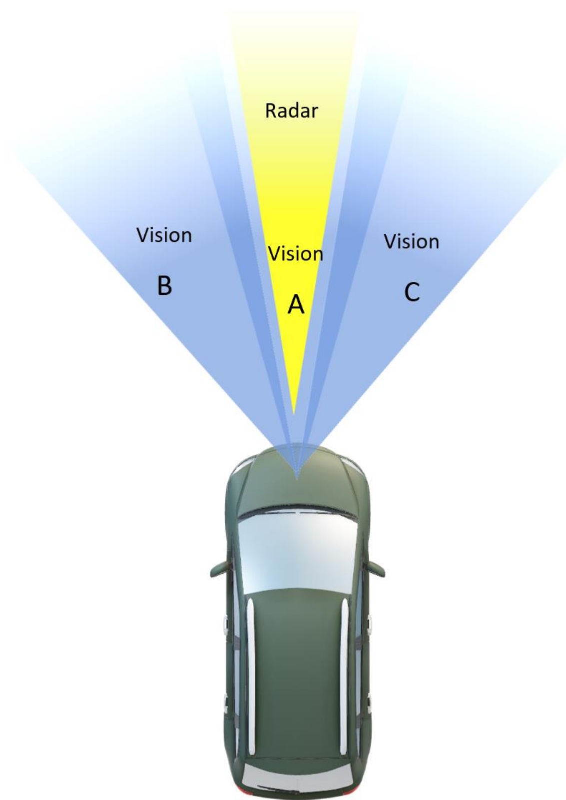
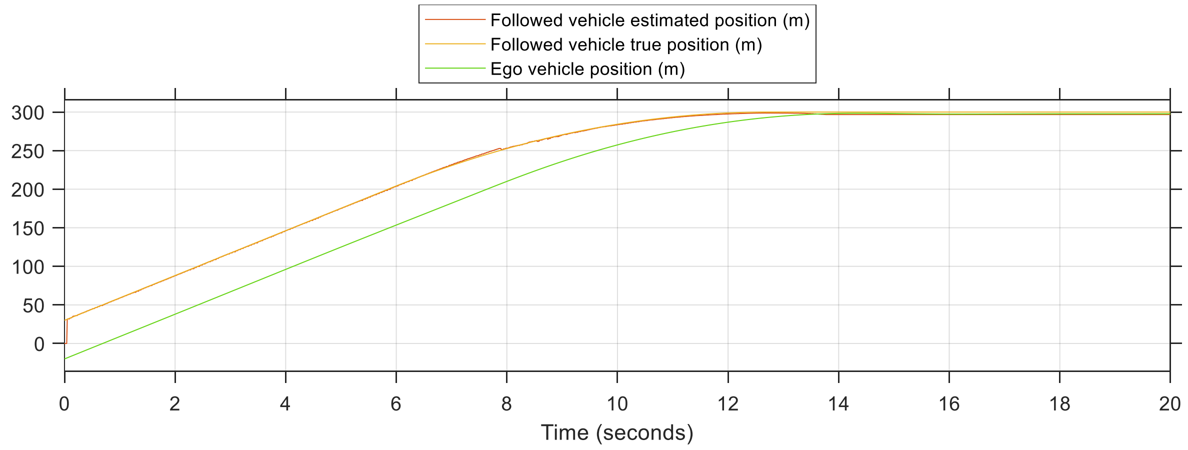
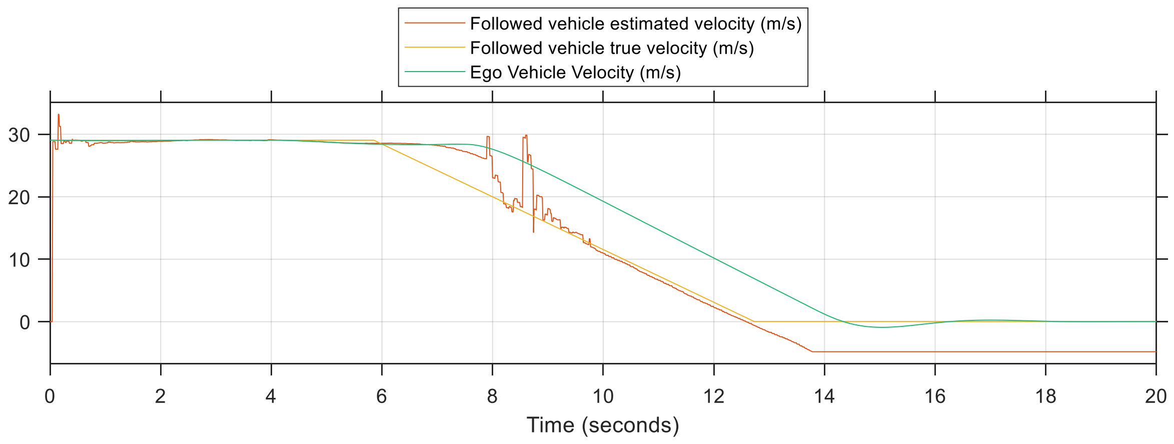


Figure 19: Experiment IV, V, and VI sensor layout





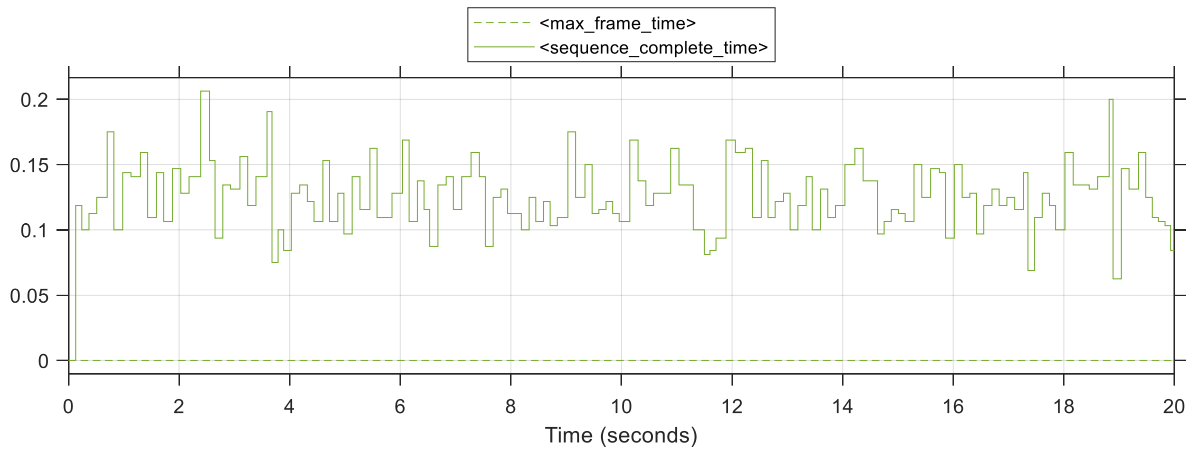


Figure 20: Experiment IV (β = 0.0, vision and radar)

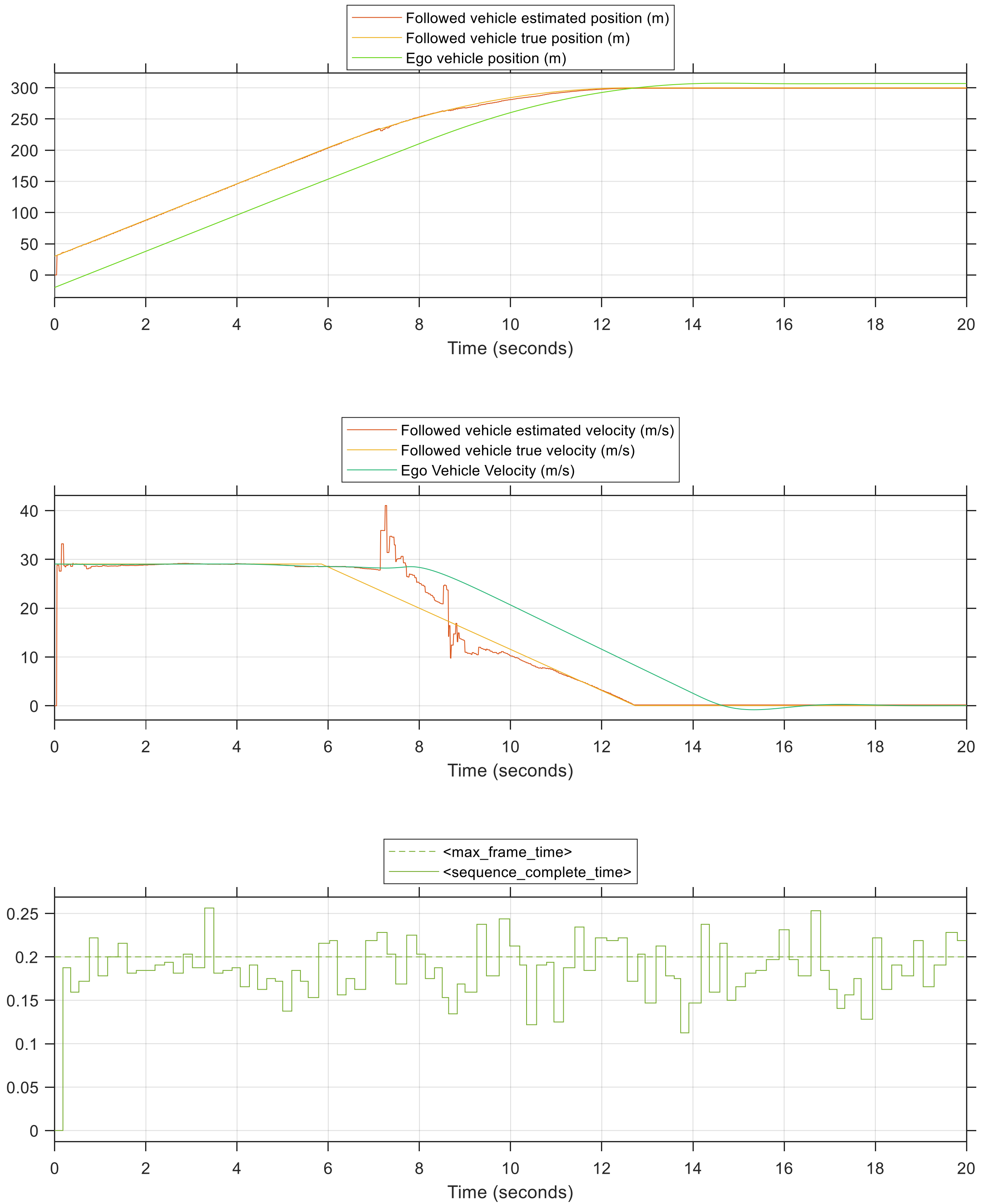


Figure 21: Experiment V (β = 0.2, vision and radar)

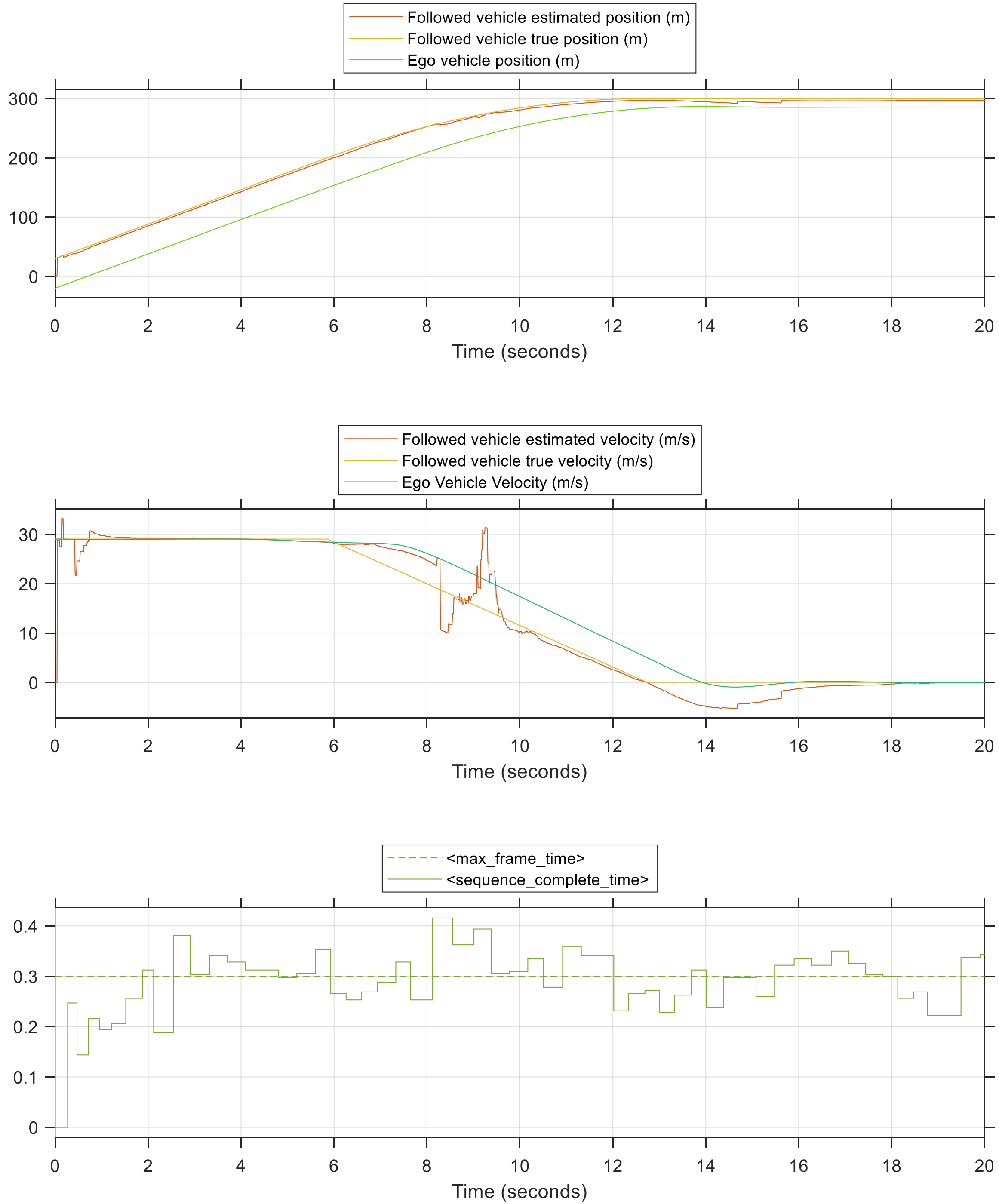


Figure 22: Experiment VI (β = 0.3, vision and radar)

The results of the six experiments are collected in Table 4.

Table 4: Summary of Results

Visual Detections	Desired frame time β	Actual frame time (mean, s)	Generated Sequence (typical)	Mean Detection Interval sensor A (s)	Detection Interval Std. Dev. sensor A (s)	Reaction Time (s) (vision)	Reaction Time (s) (vision + radar)
Primary Only	0 (minimized)	0.139	ABC	0.154	0.026	1.51	1.27
Primary + Secondary	0.2	0.195	AaBaCa	0.061	0.026	1.01	1.37
Primary + Secondary	0.3	0.305	AaaBaaCaa	0.042	0.024	0.88	0.96

Discussion

The first point of discussion is the generated sequence. As expected, we see the generated sequence length grows with increasing frame bound β. Additionally, resulting frame time value 0.195 when $\beta = 0.2$ and 0.305 when $\beta = 0.3$ very closely match the desired frame time. Further, the mean detection interval decreases as expected. This shows reasonable performance in prediction of frame processing time with the available hardware configuration. The deviation in detection intervals appears close in all three configurations, indicating there will be variance no matter what, and it is generally unaffected by the change in sequence. However, it is likely that more complicated sequences will have greater detection variance for a particular sensor.

Our next topic of discussion is the vehicle reaction time. With vision only, we see an inverse relationship between mean detection interval and reaction time. Qualitatively, we see the least artifacts (large sections of noise) in the velocity plots when $\beta = 0.3$ increases. This is an

expected result. However, the presence of such artifacts shows a revised metric for vehicle reaction time may be necessary. When radar is added, the reaction time is reduced even without the adaptive algorithm applied. This is to be expected, as modern vehicles use radar for distance estimation for this reason – increased sensing reliability. However, the presence of these additional measurements is not without a cost. There tends to be more artifacts in the velocity plots for experiments IV through VI. Further, the reaction time actually increased when β increased from 0 to 0.2. This is likely because of the high error in the vision measurements competing with the more regular radar measurements resulting in an overall increase in object tracker state covariance. Therefore, the noise characteristics and measurement frequency of the radar and vision detection algorithm must be considered. Under different conditions, such as low light, there may be even further discrepancy, however, the radar will continue to function even if the vision sensor cannot. We conclude with the following:

1. Vision-based position and velocity estimates can be improved by the adaptive technique and/or radar measurements.
2. The characteristics of the different types of sensor object tracker play a significant role in the estimate, which may help or hinder the result.
3. The best choice for overall reliability is to combine the vision and radar with the adaptive algorithm, and, if permissible, a large value for β.

Finally, we relate this work back to the objectives listed in Chapter 1. We have provided a method of object detection that exhibits adaptive qualities in terms of the perceived environment and available computational resources. The method is shown to be effective using an existing visual object recognition algorithm. The Simulink model, although simplified

compared to an actual automated driving system, demonstrates a complete feedback network including all major subsystems of the vehicle controller conceptual model. We see that by increasing the upper bound for frame processing time β, we find the tradeoff between decreased primary image detection frequency and increased secondary image detection frequency can be valuable to the vehicle reaction time.

Certainly, this particular set of experiments is considered under specific conditions, but the concept can easily be extended. Especially in research environments, the availability of complete reference models is invaluable, and our proposed model provides students and researchers interested in autonomous vehicles with a platform to evaluate adaptive perception, while easily being adapted and extended to different tasks. We consider some extensions in the next section.

Future Work

In this model, we only considered a straight road model with an ego vehicle and one other vehicle. The most obvious extension to this work is additional scenarios. We can also investigate a generalization of the adaptive techniques described here and generalize it to other types of sensors and different models of subsequent control components in the vehicle.

For the straight road, additional vehicles can be added to test the system performance with multiple objects. We can also vary the speed of the vehicles and assess the system performance against stationary and moving objects. We may also consider a curved road. Given the adaptive nature of the proposed system and feedback from motion planner, as well as the multiple sensor capability, this is a natural extension of the work presented here.

A physical implementation with laboratory vehicle is also important for verifying the results in a practical setting. A physical environment provides the optimal challenge for the proposed algorithm, as many artifacts must be considered that are not easily simulated: such as weather, varying lighting condition, and sensor noise.

Summary

In this research, we have explored the feasibility of a situationally aware object detection method for use in autonomous vehicle control. We introduced a general framework for process feedback and resource allocation in real-time, with an application to visual object detection. A complete simulation model was presented containing several detailed elements of an autonomous vehicle control system. We performed experiments showing feasibility among a range of parameters.

This work illustrates the growing complexity of modern control and automation applications, and the ever-important need to stay current on the latest techniques in related areas such as image processing, feedback control, real-time embedded systems, artificial intelligence, and optimization. The study of autonomous vehicles is not limited to these areas.

Intelligent systems are increasingly becoming integral to everyday life. It is a very exciting opportunity to participate in the study and development of autonomous driving technology. In our work, we illustrated how the additions of seemingly tiny increments of information to autonomous vehicle are what allows it to grow in performance. In the same way, the combined contributions of universities and institutions across the world are what will realize the intelligent and automated world of the future.

REFERENCES

[1] "Automated Driving Systems: A Vision for Safety." https://www.transportation.gov/av/3 (accessed Oct. 09, 2019).

[2] U. S. DOT, "Automated Vehicles 3.0 Preparing for the Future of Transportation." https://www.transportation.gov/av/3 (accessed Oct. 09, 2019).

[3] NHTSA, "Automated Vehicles for Safety." https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety (accessed Oct. 09, 2019).

[4] C. Urmson *et al.*, "Autonomous driving in traffic: Boss and the urban challenge," *AI Mag.*, vol. 30, no. 2, pp. 17–17, 2009.

[5] SAE International, *SAE International, Ford, General Motors, and Toyota form Automated Vehicle Safety Consortium*, Apr. 04, 2019.

[6] S. Liu, L. Li, J. Tang, S. Wu, and J.-L. Gaudiot, "Creating autonomous vehicle systems," *Synth. Lect. Comput. Sci.*, vol. 6, no. 1, pp. i–186, 2017.

[7] A. Appriou, *Uncertainty theories and multisensor data fusion*. ISTE : Wiley, 2014.

[8] W. Elmenreich, "Sensor fusion in time-triggered systems," 2002. http://www.academia.edu/download/3575988/10.1.1.69.9299.pdf (accessed Sep. 04, 2018).

[9] MIT News Office, "Study measures how fast humans react to road hazards," *MIT News*, Aug. 07, 2019.

[10] M. A. Fischler and R. A. Elschlager, "The representation and matching of pictorial structures," *IEEE Trans. Comput.*, vol. 100, no. 1, pp. 67–92, 1973.

[11] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *Handb. Brain Theory Neural Netw.*, vol. 3361, no. 10, p. 1995, 1995.

[12] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015, pp. 91–99.

[13] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016, pp. 779–788.

[14] X. Wang, A. Shrivastava, and A. Gupta, "A-fast-rcnn: Hard positive generation via adversary for object detection," 2017, pp. 2606–2615.

[15] P. Dollár, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 8, pp. 1532–1545, 2014.

[16] M. Chetto, *Real-time systems scheduling: fundamentals*. John Wiley & Sons, 2014.

[17] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, “Period optimization for hard real-time distributed automotive systems,” 2007, pp. 278–283.

[18] J. Wang, *Real-time embedded systems*. John Wiley & Sons, 2017.

[19] A. Kelly, “Adaptive perception for autonomous vehicles,” CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1994.

[20] C. Kwok, D. Fox, and M. Meila, “Adaptive real-time particle filters for robot localization,” 2003, vol. 2, pp. 2836–2841.

[21] J. Kim, R. Rajkumar, and M. Jochim, “Towards dependable autonomous driving vehicles: a system-level approach,” *ACM SIGBED Rev.*, vol. 10, no. 1, pp. 29–32, 2013.

[22] Y. Lu, T. Javidi, and S. Lazebnik, “Adaptive object detection using adjacency and zoom prediction,” 2016, pp. 2351–2359.

[23] T. Ruf, A. Ernst, J. Garbas, and A. Papst, “Apparatus and method for resource-adaptive object detection and tracking,” Oct. 2017.

[24] C. Merfels, “Sensor fusion for localization of automated vehicles.” https://d-nb.info/1173898506/34 (accessed Apr. 04, 2019).

[25] S. J. Russell and S. Zilberstein, “Composing Real-Time Systems.,” 1991, vol. 91, pp. 212–217.

[26] D. González, J. Pérez, V. Milanés, and F. Nashashibi, “A review of motion planning techniques for automated vehicles,” *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1135–1145, 2015.

[27] Sebastian Thrun; Wolfram Burgard; Dieter Fox, *Probabilistic Robotics*. Cambridge, Massachusetts: MIT Press, 2005.

[28] Y. Moses, “Resource-bounded knowledge,” 1988, pp. 261–275.

[29] “MATLAB - MathWorks.” https://www.mathworks.com/products/matlab.html (accessed Dec. 13, 2018).

[30] “Automated Driving Toolbox Documentation.” https://www.mathworks.com/help/driving/index.html?s_tid=srchtitle (accessed Oct. 21, 2020).

[31] "Simulink - Simulation and Model-Based Design - MATLAB & Simulink." https://www.mathworks.com/products/simulink.html (accessed Oct. 21, 2020).

[32] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," 2009, pp. 304–311.

[33] "Convert bird's-eye-view image coordinates to vehicle coordinates - MATLAB imageToVehicle." https://www.mathworks.com/help/driving/ref/birdseyeview.imagetovehicle.html?s_tid=srchtitle (accessed Oct. 28, 2020).

[34] J. Munkres, "Algorithms for the assignment and transportation problems," *J. Soc. Ind. Appl. Math.*, vol. 5, no. 1, pp. 32–38, 1957.

[35] P. Konstantinova, A. Udvarev, and T. Semerdjiev, "A study of a target tracking algorithm using global nearest neighbor approach," 2003, pp. 290–295.

[36] "Track objects using GNN assignment - MATLAB." https://www.mathworks.com/help/driving/ref/multiobjecttracker-system-object.html?s_tid=srchtitle#mw_a97ab2f0-7332-474f-b54c-ba98d8ac4559 (accessed Oct. 20, 2020).

[37] S. Haykin, *Kalman filtering and neural networks*, vol. 47. John Wiley & Sons, 2004.

[38] "Create detection objects from radar measurements - Simulink." https://www.mathworks.com/help/driving/ref/radardetectiongenerator.html (accessed Sep. 30, 2020).

[39] A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," *IEEE Trans. Intell. Transp. Syst.*, vol. 4, no. 3, pp. 143–153, 2003.

[40] W. Pananurak, S. Thanok, and M. Parnichkun, "Adaptive cruise control for an intelligent vehicle," 2009, pp. 1794–1799.

[41] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," 2007, pp. 2296–2301.

[42] D. Wang and F. Qi, "Trajectory planning for a four-wheel-steering vehicle," 2001, vol. 4, pp. 3320–3325.

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

[44] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," 2015, pp. 1094–1099.

[45] "Implement a single track 3DOF rigid vehicle body to calculate longitudinal, lateral, and yaw motion - Simulink." https://www.mathworks.com/help/driving/ref/bicyclemodel.html?s_tid=srchtitle (accessed Oct. 20, 2020).

APPENDIX – MATLAB CODE

```
classdef motion_decision < matlab.System

    % Block Memory
    properties (GetAccess='private', SetAccess='private', Hidden)
        MIOvel;
        MIOvel_last;
        mode;
        MIOstate;
        MIOvalid_last;
        MIOvalid;   %Same as isLaneClear
        vel;
        accum;
    end


    % Block settings
    properties
        cruisevel = 30;
        mindist = 37;
        maxdist = 43;
        Ts = 0.01; %sample time
        cutoff_vel = 28; % E-stop condition
        v0 = 30; % Initial ego velocity
        tstartup = 2.5; % Time to reach steady state condition
    end

    methods (Access = protected)

        % Setup
        function setupImpl(obj)
            obj.mode = 0;
            obj.MIOvel = 0;
            obj.MIOvel_last = 0;
            obj.MIOstate = [0 0 0 0]; %[x xdot y xdot xdotdot 0]
            obj.MIOvalid = false;
            obj.MIOvalid_last = false;
            obj.vel = obj.v0;

        end

        % Step
        function [vel,miostate,mode] = stepImpl(obj, trigger, tracks, pos_ego, clock)

            if trigger
                obj.MIOvel_last = obj.MIOvel;
                obj.MIOvalid_last = obj.MIOvalid;

                numtracks = 0;
                isLaneClear = true;
                % Count valid tracks
                laneboundaries = [-1 -5];
                veh_width = 1.8;
                for i = 1:(size(tracks,1))
                    if tracks(i,1)>0
                        numtracks = numtracks + 1;
```

```
        % Check if each track is in our lane
        tracky = tracks(i,4);
        if tracky <= laneboundaries(1) + veh_width / 2 ...
                & tracky >= laneboundaries(2) - veh_width / 2
            % MIO exists
            isLaneClear = false;

            obj.MIOvalid = true;
        end
    end
end

obj.MIOvel = -1;


if ~isLaneClear
    % Find index of MIO
    [MIOnearest, MIOindex] = min(tracks(1:numtracks,2));
    reldist = pos_ego(1) - tracks(MIOindex,1);
    % Find x velocity of MIO
    obj.MIOvel = tracks(MIOindex,3);
    obj.MIOstate(1:4) = tracks(MIOindex,2:5);
else
    obj.vel = obj.cruisevel;
end

reldist = 0;

% Settling timer
if clock >= obj.tstartup
    steadystate = true;
else
    steadystate = false;
end

% Estimate MIO acceleration
if obj.MIOvalid_last & obj.MIOvalid

    disp('MIOvel')
    disp(obj.MIOvel)
    disp('MIOvel_last')
    disp(obj.MIOvel_last)

   if obj.MIOvel_last > obj.cutoff_vel...
           & obj.MIOvel < obj.cutoff_vel &...
           steadystate
        obj.mode = 2;
   end

end

% If steady state reached, allow mode transitions
if steadystate

    % Mode transitions
    switch obj.mode
```

```
                % State -1
                case -1
                    if reldist < obj.maxdist | isLaneClear
                        obj.mode = 0;
                    end
                % State 0
                case 0
                    if ~isLaneClear & reldist < obj.mindist
                        obj.mode = -1;
                    elseif ~isLaneClear & reldist > obj.maxdist
                        obj.mode = 1;
                    end
                % State 1
                case 1
                    if reldist > obj.mindist | isLaneClear
                        obj.mode = 0;
                    end
                % State 2
                case 2
                    ;
            end

            obj.vel = 0;
            % Decide velocity
            switch obj.mode
                % Decelerate
                case -1
                    obj.vel = obj.MIOvel - 0.5;
                % Hold speed
                case 0
                    if isLaneClear | steadystate
                        obj.vel = obj.cruisevel;
                    else
                        obj.vel = obj.MIOvel;
                    end
                % Accelerate
                case 1
                    obj.vel = obj.MIOvel + 0.5;
                % Emergency Braking
                case 2
                    obj.vel = 0;

            end
        end
    end
    vel = obj.vel;
    mode = obj.mode;
    miostate = obj.MIOstate;
end

function [vel,miostate,mode] = getOutputSizeImpl(obj)
    % Return size for each output port
    %y = [10 2];
    vel = [1 1];
    miostate = [1 4];
    mode = [1 1];
```

```
        end

        function [vel,miostate,mode] = getOutputDataTypeImpl(obj)
            % Return data type for each output port
            %y = 'string';
            vel = 'double';
            miostate = 'double';
            mode = 'double';
        end

        function [vel,miostate,mode] = isOutputComplexImpl(obj)
            % Return true for each output port with complex data
            %y = false;
            vel = false;
            miostate = false;
            mode = false;
        end

        function [vel,miostate,mode] = isOutputFixedSizeImpl(obj)
            % Return true for each output port with fixed size
            %y = true;
            vel = true;
            miostate = true;
            mode = true;
        end
    end
end
```

```
function seq =
create_vision_sequence(classes,classes_which_priority,t1sum,t2ave,max_frame_time)

    classeslower = lower(classes);
    mp = size(classes,2);

    ms = 0;
    for i=1:mp
        if classes_which_priority(i)
            ms = ms + 1;
            classes_priority(i) = lower(classes(i));
        else
            classes_priority(i) = ' ';
        end
    end
    % Determine available time to stay within bound
    free_space = max(max_frame_time - t1sum,0);

    % Find number of available slots
    if ms >0
        mfree = floor(free_space/t2ave);
    else
        mfree = 0;
    end

    % Find next lowest number of free spots, as a
    % multiple of mp and
    mfree_sup = ceil(mfree/mp)*mp;

    % Distribute secondary detections appropriately
    origin_sequence = [ceil([1:mfree]*ms/mfree) zeros(1, mfree_sup-mfree)];


    % Walk through the classes 1 through n in a staggered order
    class_nums = 1:ceil(mp/2)*2;
    walk_order = zeros(1,mp);
    w1 = class_nums(1:ceil(mp/2));
    w2 = [class_nums(ceil(mp/2)+1:end) 0];
    for i=1:mp
        is_even = mod(i-1,2);
        walk_order(mp-i+1) =
class_nums((~is_even)*w1(ceil(i/2))+is_even*w2(ceil(i/2)));
    end

    % Create a new sequence
    num_new_bins_per_primary = ceil(mfree/mp);
    total_new = mp + num_new_bins_per_primary*mp;
    detection_sequence_new = strings([1,total_new]);

    % Distribute primary
    for i = 1:mp
        detection_sequence_new((i-1)*num_new_bins_per_primary+1)=classes(i);
        for k=1:num_new_bins_per_primary;
        end
    end
```

```
    % Distribute Secondary
    for i = 1:mp
        detection_sequence_new((i-1)*num_new_bins_per_primary+1)=classes(i);
    end

    walk_order = mp:-1:1;

    sw = zeros(1,mfree_sup);
    i = 1:mfree_sup;
    k = mod(i-1, mp)+1 ;
    j = walk_order(k) + floor((i-1)./mp)*mp;
    sw(i) = origin_sequence(j);

    aug = [[1:mp]' , reshape(sw,[mp,ceil(mfree_sup/mp)])];
    augstr = strings(size(aug));

    for i = 1:mp
        for j = 1:ceil(mfree_sup/mp)+1
            if j==1
                augstr(i,j) = classes(aug(i,j));
            else
                if aug(i,j) > 0
                    augstr(i,j) = classeslower(aug(i,j));
                else
                    augstr(i,j) = ' ';
                end
            end
        end
    end


    swstr = reshape(augstr',[1 mp + mfree_sup]);

    % Matlab converts single entry string arrays to just a string
    % Workaround
    if size(swstr,2)>1
        swstr2 = convertStringsToChars(swstr);
        swstr3 = [swstr2{:}];
    else
        swstr3 = swstr;
    end

    %Purge blank slots
    whereis_blank = strfind(lower(swstr3)," ");
    for i = flip(whereis_blank)
        swstr3(i)=[];
    end

    %swstrtotal = swstrtotal + "_" + swstr3;

    seq=swstr3;
end
```

```
classdef detector_multi_adaptive < matlab.System
    % Adaptive Multi-Sensor Visual Object Detector

    % Block Memory
    properties (GetAccess='private', SetAccess='private', Hidden)
        detector;
        storage;
        ylast;
        Tpd;
        Tp;
        ready;
        process_acc;
        request_sequence;
        sequence;
        sequence_length;
        sequence_step;
        t1ave;
        t2ave;
        sequence_accum_time;
        sequence_complete_time;
        primary_times_hist;
        secondary_times_hist;
        detections;
        num_detections;
        sensor;
        image_result;
        bInit;
        current_sensor;
        meas_interest_mean; % Sensor interval mean
        meas_interest_stdev; % Sensor interval stdev
        meas_interest_last; % Storage for mean sensor metric measurement
        meas_interest_hist;

        frame_time_mean; % Average Frame time
        frame_time_last; % Storage forAverage Frame time measurement
        frame_time_hist;
    end


    % Block settings
    properties
        Ts = 0.01;          % Sample time
        singlegrid = 0;     % For legacy plot output
        display = 0;        % For legacy plot output
        num_sensors = 1;        %Number of sensors
        max_frame_time = 0.5; %Upper bound for sequence time
        Tp0 = 0.25;             %Initial detection time (scaled)
        tscale = 1; %Hardware time scaling factor
        camera_resolution = [720 720];
        sensor_of_interest; % Sensor selection for mean sensor metric


    end

    methods (Access = protected)
```

```
        % Setup
        function setupImpl(obj)
            obj.bInit = 0;
            obj.ready = 1;
            obj.request_sequence = 1;
            obj.detector = vehicleDetectorACF();

            obj.primary_times_hist = zeros(10,5);
            obj.primary_times_hist(:,1:obj.num_sensors)=obj.Tp0;
            obj.secondary_times_hist = zeros(10,5);
            obj.t1ave=0;
            obj.sequence_accum_time=0;
            obj.sequence_complete_time=0;
            obj.image_result = uint8(zeros(720,720,3));
            reltranslation = [1, 0, 1.4];
            focalLength = [1109, 1109];
            opticalCenter = [360, 360];
            imageSize = [720, 720];
            intrinsics = cameraIntrinsics(focalLength,opticalCenter,imageSize);
            obj.sensor = monoCamera(intrinsics,reltranslation(3));
            obj.current_sensor = uint8('A');
            obj.meas_interest_mean=0; % Sensor interval mean
            obj.meas_interest_stdev=0; % Sensor interval stdev
            obj.meas_interest_hist=zeros(20,1); % Storage for mean sensor metric
measurement
            obj.meas_interest_last=0; % Storage for mean sensor metric measurement

            obj.frame_time_mean=0; % Average Frame time
            obj.frame_time_hist=zeros(20,1); % Storage for Average Frame time
measurement

        end

        % Step
        function [status] = stepImpl(obj, inA, inB, inC, inD, inE, sub_rects, clock)
            % Assume we are not sending any new detections, until
            % a detection process finishes
            sendDetections = false;
            % Assume no detections until we find some
            num_valid_detections = 0;
            if obj.request_sequence
                    % Generate new sequence
                    classes = convertCharsToStrings({'A' 'B' 'C' 'D' 'E'});
                    classes = classes(1:obj.num_sensors);
                    classes_which_priority = (sum(sub_rects,1)~=0)';
                    num_sec = sum(classes_which_priority)
                    t1sum = sum(mean(obj.primary_times_hist))
                    area_pri = obj.camera_resolution(1)*obj.camera_resolution(2)
                    ave_area_sec = sum(sub_rects(3,:).*sub_rects(4,:))/num_sec

                    % Estimate t2 time by ratio of pixels covered
                    if num_sec > 0
                        t2est = t1sum * (ave_area_sec/area_pri)/obj.num_sensors
                    else
                        t2est = 0;
                    end
```

```
                    % Call sequence generator
                    obj.sequence =
create_vision_sequence(classes,classes_which_priority,t1sum,t2est,obj.max_frame_time);
                    disp("Sequence " + obj.sequence);
                    obj.sequence_length = strlength(obj.sequence);
                    obj.sequence_step = 1;
                    obj.request_sequence = 0;

            end

            % If we are ready, continue a sequence
            if obj.ready | ~obj.bInit
                obj.bInit = true;
                % Send to status
                % Decide which sensor to capture from
                ch = char(obj.sequence);
                out.current_sensor = ch(obj.sequence_step);

                % Record time
                before = cputime;
                isPrimary = false;

                % Set up detection based on current sequence position
                switch(out.current_sensor)
                    case 'A'
                        imageIn = inA; sensor_index = 1; isPrimary = true;
                    case 'B'
                        imageIn = inB; sensor_index = 2; isPrimary = true;
                    case 'C'
                        imageIn = inC; sensor_index = 3; isPrimary = true;
                    case 'D'
                        imageIn = inD; sensor_index = 4; isPrimary = true;
                    case 'E'
                        imageIn = inE; sensor_index = 5; isPrimary = true;
                    case 'a'
                        imageIn = imcrop(inA,sub_rects(:,1)); sensor_index = 1;
                    case 'b'
                        imageIn = imcrop(inB,sub_rects(:,2)); sensor_index = 2;
                    case 'c'
                        imageIn = imcrop(inC,sub_rects(:,3)); sensor_index = 3;
                    case 'd'
                        imageIn = imcrop(inD,sub_rects(:,4)); sensor_index = 4;
                    case 'e'
                        imageIn = imcrop(inE,sub_rects(:,5)); sensor_index = 5;
                    otherwise
                        disp('error')
                end

                % Get sensing interval metric
                if obj.sensor_of_interest == sensor_index
                       meas_diff = clock - obj.meas_interest_last;
                       obj.meas_interest_last = clock;
                       obj.meas_interest_hist = ...
                           circshift(obj.meas_interest_hist,1,1);
                       obj.meas_interest_hist(1) = meas_diff;
```

```
                disp(obj.meas_interest_hist)
                obj.meas_interest_mean = mean(obj.meas_interest_hist);
                obj.meas_interest_stdev = std(obj.meas_interest_hist);
            end



            [bboxes,scores] = detect(obj.detector,imageIn);

            now = cputime;
            obj.Tp=(now-before)*obj.tscale;
            %Tp = toc(before);
            obj.Tpd=uint16(ceil(obj.Tp/obj.Ts));

            % Compute object locations
        bboxes2=[];
        which = find(scores>20);
        y = zeros(10,2);

        num_valid_detections = size(which,1);




            % Gather significant detections and draw bounding boxes if
             if num_valid_detections > 0
                 bboxes2 = bboxes(which,:);
                 scores2 = scores(which,:);
                 % Draw bounding boxes
                 image_result =
insertObjectAnnotation(imageIn,'rectangle',bboxes2,scores2);
             else
                 image_result = imageIn;
             end
             r =sub_rects(:,sensor_index);
             if isPrimary
                % If primary, just copy the annoted image
                obj.image_result = image_result;
             else
                % if secondary, paste input image into black image
                obj.image_result = zeros(obj.camera_resolution(1), ...
                    obj.camera_resolution(2), 3);

                obj.image_result(r(2):r(2)+r(4),r(1):r(1)+r(3),:) = image_result;
             end

            % Image offset from reduced detection
            offset = [0 0];
            if ~isPrimary
                offset = [r(1) r(2)];
            end



            obj.detections = zeros(10,2);
            % Compute Object Locations
```

```
                for i = 1:(min(num_valid_detections,10))
                    ctr = bboxes2(i,1)+bboxes2(i,3)/2;
                    hgt = bboxes2(i,2)+bboxes2(i,4);
                    loc = imageToVehicle(obj.sensor,[ctr,hgt]+offset);
                    obj.detections(i,:) = loc;
                end
                obj.num_detections = num_valid_detections;
                %disp(obj.detections);
                % Add to total sequence time counter
                obj.sequence_accum_time = obj.sequence_accum_time + obj.Tp;
                % Store process time in appropriate matrix
                disp('Detected in ' + string(obj.Tp) + ' sec('+ string(obj.Tpd) + '
cycles)');
                if isPrimary
                    % Roll matrix column by 1
                    obj.primary_times_hist(:,sensor_index) = ...
                        circshift(obj.primary_times_hist(:,sensor_index), 1, 1);
                    % Store value
                    obj.primary_times_hist(1,sensor_index) = obj.Tp;
                else
                    % Roll matrix column by 1
                    obj.secondary_times_hist(:,sensor_index) = ...
                        circshift(obj.secondary_times_hist(:,sensor_index), 1, 1);
                    % Store value
                    obj.secondary_times_hist(1,sensor_index) = obj.Tp;

                end

                % Update summaries
                obj.t1ave = mean(obj.primary_times_hist);
                obj.t2ave = mean(obj.secondary_times_hist);

                % Update status
                obj.ready = 0;
                obj.process_acc = 1;
            else
                % Increment time step
                obj.process_acc = obj.process_acc + 1;

                % Check if detection is complete
                if obj.process_acc >= obj.Tpd;
                    %disp('Detect complete');
                    obj.ready = 1;
                    obj.process_acc = 0;
                    sendDetections = true;

                    % Increment sequence step
                    obj.sequence_step = obj.sequence_step + 1;

                    % Check if sequence is complete
                    if obj.sequence_step > obj.sequence_length
                        obj.sequence_step = 1;
                        obj.request_sequence = 1;
                        obj.sequence_complete_time = obj.sequence_accum_time;
                        obj.sequence_accum_time = 0;
```

```
                    % Update frame time history matrix
                    obj.frame_time_hist = ...
                       circshift(obj.frame_time_hist,1,1);
                    obj.frame_time_hist(1) = obj.sequence_complete_time;
                    % Compute mean
                    obj.frame_time_mean = mean(obj.frame_time_hist)

                end
            end

        end

        % Send status to bus
        status.ready = uint8(obj.ready);
        status.current_sensor = obj.current_sensor;
        status.primary_times_hist = obj.primary_times_hist;
        status.secondary_times_hist = obj.secondary_times_hist;
        status.sequence_complete_time = obj.sequence_complete_time;
        status.t1ave = obj.t1ave;
        status.t2ave = obj.t2ave;
        status.max_frame_time = obj.max_frame_time;

        % Send the results from previously completed detection
        status.detections = obj.detections * sendDetections;
        status.detections_preview = obj.detections;
        status.detection_complete = logical(obj.ready);
        status.numdetections = obj.num_detections * sendDetections;
        status.image_result = uint8(obj.image_result);
        status.meas_interest_mean = obj.meas_interest_mean;
        status.meas_interest_stdev = obj.meas_interest_stdev;
        status.frame_time_mean = obj.frame_time_mean;
    end

    function [status] = getOutputSizeImpl(obj)
        % Output port size
        status = 1;
    end

    function [status] = getOutputDataTypeImpl(obj)
        % Output data type
        status = 'Bus: BusDetectorStatus2';
    end

    function [status] = isOutputComplexImpl(obj)
        % Output complexity
        status = false;
    end

    function [status] = isOutputFixedSizeImpl(obj)
        % Output fixed/variable
        status = true;
    end
end
end
```

```
classdef motekf < matlab.System

    % Encapsulates the multi-object tracker function
    % With simplified output

    % Block Memory
    properties (GetAccess='private', SetAccess='private', Hidden)
        tracker;
        tracks_last;
    end


    % Block settings
    properties(Nontunable)
        FilterInitializationFcn = 'initcvkf';
        ConfirmationThreshold = [2 3];
        DeletionThreshold = [5 5];
    end

    methods (Access = protected)

        % Block Setup
        function setupImpl(obj)
            obj.tracker = multiObjectTracker('FilterInitializationFcn', ...
                obj. FilterInitializationFcn, ...
                'ConfirmationThreshold', ...
                obj.ConfirmationThreshold, ...
                'DeletionThreshold', ...
                obj.DeletionThreshold);
                obj.tracks_last = zeros(10,5);
        end

        % Step
        function [trigout, tracks] = stepImpl(obj, trigger, numdetections, detections,
clock)
            if trigger
                % Assume no confirmed tracks until we determine some
                confirmedtracks = {};
                if numdetections>0
                    dets = {};
                    for i = 1:numdetections
                        loc = [detections(i,1),detections(i,2),0];
                        dets(i) = { ...
                            objectDetection(clock,loc, ...
                            'SensorIndex',1) ...
                            };

                    end
                    % Send detections to tracker
                    [confirmedtracks,tentativetracks] = obj.tracker(dets, clock);
                else
                    if obj.tracker.NumTracks > 0
                        [confirmedtracks,tentativetracks] = obj.tracker({}, clock);
                    end
                end
```

```
                % Get Tracks
                tracks = zeros(10,5);
                numconf = size(confirmedtracks,1);
                if numconf>0
                    for i = 1:size(confirmedtracks)
                        tracks(i,1) = double(confirmedtracks(i).TrackID);
                        tracks(i,2:5) = confirmedtracks(i).State(1:4);
                    end
                end
                obj.tracks_last = tracks;
            else
                tracks = obj.tracks_last;
            end
            trigout = trigger;
        end

        function [trigout, tracks] = getOutputSizeImpl(obj)
            % Return size for each output port
            %y = [10 2];
            tracks = [10 5];
            trigout = [1 1];
        end

        function [trigout, tracks] = getOutputDataTypeImpl(obj)
            % Return data type for each output port
            %y = 'string';
            tracks = 'double';
            trigout = 'boolean';
        end

        function [trigout, tracks] = isOutputComplexImpl(obj)
            % Return true for each output port with complex data
            %y = false;
            tracks = false;
            trigout = false;
        end

        function [trigout, tracks] = isOutputFixedSizeImpl(obj)
            % Return true for each output port with fixed size
            %y = true;
            tracks = true;
            trigout = true;
        end
    end
end


function [trig_out,numdet_out,det_out] =
combine(vis_trig,vis_numdet,vis_det,radar_det,radar_trig)
%Combine radar and vision measurements
det_buffer = zeros(10,2);
radar_numdet=0;
vision_numdet=0;
%Radar bias offset value
radar_offset=[1 0];
```

```
% Combine vision measurements when triggered
if(vis_trig)
    vision_numdet = vis_numdet;
    det_buffer(1:vision_numdet,:) = vis_det(1:vision_numdet,:);
end

% Combine radar measurements when triggered
if(radar_trig)
    radar_numdet = radar_det.NumDetections;
    for i = 1:radar_numdet;
        variance = 0.2;
        radar_noise = sqrt(variance)*randn(1,2);
        det_buffer(vision_numdet+i,:) = ...
            radar_det.Detections(i).Measurement(1:2)' + ...
            radar_offset + radar_noise;
    end
end

% Send output
trig_out = vis_trig | radar_trig;
numdet_out = vision_numdet + radar_numdet;
det_out = det_buffer;
```