GEORGIA DOT RESEARCH PROJECT 16-13

FINAL REPORT

A New Vehicle Concept To Provide Better Rapid Transit At Reduced Cost (Phase 1)

GDOT
Georgia Department of Transportation

OFFICE OF PERFORMANCE-BASED MANAGEMENT AND RESEARCH

15 KENNEDY DRIVE
FOREST PARK, GA 30297

GDOT Research Project. 16-13

Final Report

A NEW VEHICLE CONCEPT TO PROVIDE BETTER RAPID TRANSIT AT REDUCED COST (PHASE 1)

By
Bill Diong, Ph.D.; Professor of Electrical Engineering
Ying Wang, Ph.D.; Associate Professor of Mechatronics Engineering
Jidong Yang, Ph.D.; Associate Professor of Civil Engineering

Kennesaw State University Research & Service Foundation

Contract with

Georgia Department of Transportation

In cooperation with

U.S. Department of Transportation
Federal Highway Administration

October 2018

TECHNICAL REPORT STANDARD TITLE PAGE

1.Report No.: FHWA-GA-RP-16-13	2. Government Accession No.:	3. Recipient's Catalog No.:	
4. Title and Subtitle: A New Vehicle Concept to Provide Better Rapid Transit at Reduced Cost (Phase 1)		5. Report Date: October 2018	
		6. Performing Organization Code:	
7. Author(s): Bill Diong, PhD; Ying Wang, PhD; and Jidong Yang, PhD		8. Performing Organ. Report No.: 16-13	
9. Performing Organization Name and Address: Kennesaw State University Research & Service Foundation 1000 Chastain Road Mail Drop 0111 Kennesaw, Georgia 30144-5591		10. Work Unit No.:	
		11. Contract or Grant No.: 0015121	
12. Sponsoring Agency Name and Address: Georgia Department of Transportation Office of Research 15 Kennedy Drive Forest Park, GA 30297-2534		13. Type of Report and Period Covered: Final; June 2016–October 2018	
		14. Sponsoring Agency Code:	
15. Supplementary Notes: Prepared in cooperation with the U.S. Department of Transportation, Federal Highway Administration.			
16. Abstract: A bus rapid transit (BRT) system is a mass transit system that uses rubber-tired vehicles operating in dedicated guideways, high-occupancy vehicle lanes, and/or mixed traffic. These systems typically also have a limited number of stops and use signal priority queue jumper lanes in order to increase operational efficiency and reliability. However, a cost-benefit analysis of a BRT system, in its present form, is not unequivocally favorable. To tilt such analysis more in its favor, this paper describes the proposed Slim Modular Flexible Electric Bus Rapid Transit (SMFe-BRT) system concept, which improves upon the existing BRT concept in several key areas. Firstly, the SMFe-BRT vehicle will comprise a lead module, with a human driver, plus one or more physically separate follower modules that track the lead module's movements in an autonomous fashion. Secondly, the bodies of these modules will be slimmer, which allow the vehicles to operate in narrower lanes, thereby yielding substantial cost savings in its implementation and operation. This report presents details regarding the design and prototyping of this SMFe-BRT vehicle's key subsystems, resulting in a prototype lead module and a prototype follower module that can each travel at straight-line speeds exceeding 15 mph and cornering speeds exceeding 4 mph. In addition, this report describes the development of a leader-follower control algorithm that works properly in an indoor environment, demonstrating module straight-line tracking up to 4 mph for time intervals of several seconds long. However, the researchers encountered some challenges demonstrating proper module following in the outdoor tests. Finally, this report presents a feasibility study evaluating SMFe-BRT against the traditional BRT based on multiple criteria, including transport efficiency, environmental impacts, and finances. The study shows that the SMFe-BRT offers additional benefits in the context of both freeway and arterial operations, and is generally preferred to the traditional BRT.			
17. Key Words: Electric Bus; Bus Rapid Transit; Semi-Autonomous; Module following		18. Distribution Statement:	
19. Security Classification (of this report): Unclassified	20. Security Classification (of this page): Unclassified	21. Number of Pages: 177	22. Price:

Form DOT 1700.7 (8-69)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

EXECUTIVE SUMMARY

In the Atlanta metropolitan area, one of the fastest growing regions in the United States, transportation ranks as the top concern of businesses and residents, while congestion relief has been the top priority when allocating transportation funding. The area currently suffers from regularly jammed highways and less than highly satisfactory public transit. One initiative to increase transit ridership is bus rapid transit (BRT), such as that proposed by Cobb County at a cost of $500 million (or almost $20 million per service mile). While BRT has many benefits and has been implemented worldwide, this research project seeks to improve upon the basic model by means of a novel vehicle concept called the Slim Modular Flexible Electric Bus (SMFe-bus). The key features of this vehicle are: (1) narrower width (25–50% slimmer than a regular bus), requiring less right-of-way; (2) a "lead" module with a driver cab, and a few driverless "follower" modules/cars trailing behind it; (3) follower modules that can be easily attached and detached from the preceding module by way of "virtual coupling" to meet varying passenger demand by time of day with optimized operations; and (4) given the smaller size of the modules, each are self-propelled by in-wheel electric motors, which will allow the modules to better negotiate turns while being more friendly to the environment than fossil-fuel engines. The significance of this project is that it will lead to a system that costs less than conventional BRT (by reducing right-of-way and construction costs), while providing an equal or better level of service (LOS), and is more environmentally friendly.

This initial phase of the planned multi-stage research project was aimed at achieving the following regarding the Slim Modular Flexible Electric Bus Rapid Transit (SMFe-BRT) concept and the SMFe-bus vehicle:

1. Demonstrate a higher benefit-to-cost ratio for the SMFe-BRT approach compared to the existing BRT approach (using Cobb County's BRT proposal and Metropolitan Atlanta Rapid Transit Authority's [MARTA's] GA 400 Transit Initiative's BRT option as case studies), and determine infrastructure design and operational feature requirements.

2. Develop two-wheel-drive prototype lead and follower SMFe-bus modules with 3-hp motors and 150-Ah battery pack, capable of speeds greater than 15 mph.
3. Demonstrate straight-line following by the two-module prototype SMFe-bus at 15 mph within an 8-ft-wide path, and also proper tracking of 90-degree cornering at 4 mph within the swept path of a 40-ft city transit bus.

After about two years of work, the following was accomplished:

- For the feasibility study, the SMFe-BRT was evaluated against the traditional BRT by considering three major criteria: transport efficiency, environmental impact, and finances. The results indicate that the SMFe-BRT provides additional benefits compared to the traditional BRT for both freeway and arterial operations.
- The developed power and propulsion system for each of the lead and follower module prototypes works properly when operated by a remote-control unit, for motor throttling (forward and reverse), steering, regenerative braking, and emergency braking. Outdoor tests indicate that the design of these prototypes yields performance that meets or exceeds the technical objectives (mainly straight-line speed and cornering speed) proposed for their power, propulsion, steering, and braking systems.
- The developed leader–follower controller works properly in an indoor environment. To solve the measurement delay problem, a dual-Kalman-filter strategy and a multi-thread programming technique were integrated into the control scheme. The indoor experimental results using two autonomous vehicles validated the effectiveness and robustness of the proposed approach, and demonstrated module straight-line tracking up to 4 mph for time intervals of several seconds long. Meanwhile, the researchers observed some challenges in the outdoor tests. In particular, a regular laser sensor cannot obtain correct measurements in a bright outdoor environment.

Hence, this project fully accomplished two of its three objectives, while partially reaching the objective of demonstrating straight-line following by the two-module prototype SMFe-bus at 15 mph, and also proper tracking of 90-degree cornering at 4 mph. Therefore, substantial progress has been made toward a better BRT system that costs less than conventional BRT (by reducing right-of-way and construction costs), while providing an equal or better level of service, and is more environmentally friendly.

ACKNOWLEDGMENTS

The work described in this report was made possible by a contract with the Department of Transportation, State of Georgia, in cooperation with the U.S. Department of Transportation Federal Highway Administration. The contents of this report reflect the view of the authors who are responsible for the facts and accuracy of the data presented herein. The contents do not necessarily reflect the official view or policies of the Department of Transportation, State of Georgia, or the Federal Highway Administration. This report does not constitute a standard, specification, or regulation.

1 INTRODUCTION

1.1 Motivation

The increasing demand for transportation versus the limited opportunities for increasing capacity within many metropolitan areas calls for more effective use of the available capacity [1]. According to the Bureau of Transportation Statistics, U.S. Department of Transportation, 79.6% of Georgia residents drove alone to work in 2013, while 10.3% participated in carpooling, and only 2.1% chose public transit (considerably less than the national average of 5.2%) [2]. In the Atlanta metropolitan area (Metro Atlanta), a recent survey concluded that transportation ranks as the top concern of its businesses and residents [3], while congestion relief has been the top priority when allocating its transportation funding. The area currently suffers from regularly jammed highways and less than highly satisfactory public transit options, even though about 12% of Atlanta residents take transit regularly.

One well-known way to mitigate highway congestion is to increase transit ridership by way of improved service, although this often requires an upgrade and/or expansion of existing transit infrastructure. Bus rapid transit (BRT) has been adopted around the world, including by Pittsburgh, Cleveland, Chicago, and New York City in the United States, Bogota in Colombia, Beijing in China, Oslo in Norway, and Mexico City in Mexico [4–12], as another option for improving transit service. It uses buses and dedicated lanes with limited stops to quickly transport passengers to their destinations while offering a certain level of flexibility to meet the demand. However, given the concerns about low transit ridership and increasing financial austerity in the U.S. public sector, BRT is often viewed as not being very cost-effective because of the required dedicated lanes, which often must be added to existing roadways. For example, establishing the BRT corridor proposed by Cobb County, Georgia, to service the 25.3-mile (one direction) stretch from a station near Kennesaw State University (KSU) in Kennesaw to the existing Metropolitan Atlanta Regional Transit Authority (MARTA) Arts Center Station has been budgeted at about $500 million [13], which is equivalent to a cost of almost $20 million per service mile, although that is still lower than the cost associated with rail-based mass transit [14]. Furthermore, to meet increased passenger demand, a shorter bus headway is currently the typical system response, which quickly drives up the cost of operations. In nearby Gwinnett County, Georgia, interest in establishing BRT has recently been voiced by the county commission chairman, and while county residents are also more favorable toward transit now than any time during the past 45 years, there is still considerable concern about its cost to taxpayers versus the expected benefits [15].

In light of those considerations, the research team proposed a Slim Modular Flexible Electric Bus Rapid Transit (SMFe-BRT) system based on a novel vehicle concept. The key features of this Slim Modular Flexible Electric Bus (SMFe-bus) vehicle are: (1) narrower width (about 25% slimmer than a regular bus), requiring less right-of-way; (2) a "lead" module with a driver cab, and a few driverless "follower" modules trailing behind the lead module, to better negotiate sharp turns; (3) modules that are not physically coupled together so each follower module can be easily detached from or attached to the preceding module by way of "virtual coupling" to better meet varying passenger demand over the course of a day; and (4) given the semi-autonomous nature of the modules, each is self-propelled by in-wheel electric motors, which will allow the modules to more quickly change speed and direction while being more friendly to the environment than using fossil-fuel engines.

1.2 Review of State of the Art

The research team performed a review regarding recent and current research related to bus transportation, and the BRT concept in particular. The findings are summarized as follows:

Wang and Li proposed using the spare capacity of the dedicated BRT lanes by high-occupancy vehicles (with more than three passengers) during peak traffic hours in Hangzhou, China, to reduce traffic jams by better balancing usage of the available road resource [16]. Dodero et al. considered seven scenarios for Line 1 of the BRT system in Mexico City, Mexico, to evaluate how operations and level of service (LOS) might improve as a result of operational modifications, investments in infrastructure, and/or technology acquisitions [11]. They found that better results were obtained from implementations requiring infrastructure investment than from those involving operational modifications. But the impact of each considered implementation was limited; only a few showed improvements on the analyzed indicators as large as 10%, testifying to the difficulty of improving BRT service.

Recently, significant attention has been paid to developing systems that localize, monitor, and track buses in public transportation networks [17–19]. GPS and vehicle-to-vehicle communications are the main technological means to acquire and share such information. In particular, the locations, speeds, and directions of buses can be shared among the bus drivers, passengers, and the administrators of public transport systems. Through sharing that information and synchronizing public transportation schedules, the passengers' transfer time could be shortened and the cost of public transport could be reduced.

Another trend in recent bus transportation research is to develop electric or hybrid electric vehicles with low emission and low energy-consumption characteristics, and also improved steering control [20–22]. In addition, various transportation modes (including commuter rails, urban transit buses, electric trolley buses, and conventional diesel buses) have been compared in terms of energy use and CO_2 emissions [23], and also cost-to-benefit ratio [24]. The results from these studies indicated that plug-in hybrid and electric city buses had the best potential to reduce energy consumption and emissions while yielding a lower cost-to-benefit ratio than conventional diesel buses.

Even more recently, Alam et al. described research, including simple road tests, on the automatic control of platoons of freight trucks [25]. Their concept essentially uses a supervisory controller to adjust the speed set-points of each truck's cruise-control system to maintain adequate vehicle separations; however, the drivers of each truck control their own vehicle's steering. Related to this work, research has been ongoing at KSU regarding the control of mobile robots [26].

In summary, this literature review did not uncover any recent or ongoing research that is identical to this proposed work. However, related research supports the BRT concept as an important and valuable alternative to other mass transit options in certain settings, with ongoing efforts to advance its components to enhance passenger experience and improve the technical, economic, and environmental performance of BRT systems [27]. Furthermore, prior research indicates that more-electric propulsion systems are the best way to reduce fossil-fuel consumption and harmful emissions.

1.3 Objectives

This initial phase of the planned multi-stage research project is aimed at achieving the following objectives regarding the Slim Modular Flexible Electric Bus Rapid Transit concept and the Slim Modular Flexible Electric Bus vehicle:

1. Demonstrate a higher benefit-to-cost ratio for the SMFe-BRT approach compared to the existing BRT approach (using Cobb County's BRT proposal and Metropolitan Atlanta Rapid Transit Authority's [MARTA's] GA 400 Transit Initiative's BRT option as case studies), and determine infrastructure design and operational feature requirements.
2. Develop two-wheel-drive prototype lead and follower SMFe-bus modules with 3-hp motors and 150-Ah battery pack, capable of speeds greater than 15 mph.
3. Demonstrate straight-line following by the two-module prototype SMFe-bus at 15 mph within an 8-ft-wide path, and also proper tracking of 90-degree cornering at 4 mph within the swept path of a 40-ft city transit bus.

2 PROCEDURE

To achieve the above-mentioned objectives, the project was carried out as three sets of tasks, detailed as follows.

2.1 Task Set 1. Feasibility Study of the SMFe-BRT

2.1.1 Task 1.1 Review of Literature and Practices

The literature and practices related to the scope of this task will be reviewed and documented.

2.1.2 Task 1.2 Select Test Corridors and Gather Data

Several candidate corridors will be considered. The selection of the final study corridor will depend on data availability and practical considerations of SMFe-BRT.

2.1.3 Task 1.3 Develop and Calibrate Simulation Model(s)

Simulation models will be built for the existing (base) condition. Data necessary for calibrating the models will be gathered. Depending on data availability, the researchers expect that certain data (e.g., traffic characteristics and roadway geometry) are required to be collected or verified in the field. This calibration process is to ensure the models replicate the existing condition, which will serve as a benchmark for evaluating any modified conditions through scenario analysis as discussed in the next subtask.

2.1.4 Task 1.4 Analyze Scenarios

By implementing the SMFe-BRT concept in a simulated environment, a number of scenarios will be considered and analyzed. The scenarios will consider the geometric requirements of the slim bodies of the vehicles (e.g., dedicated narrower lanes, turning radii, access requirements, etc.) and specific performance characteristics of SMFe-BRT (e.g., demand-responsiveness and signal priority).

2.1.5 Task 1.5 Evaluate Feasibility

The feasibility of SMFe-BRT will be evaluated in a broader context, including its interactions with other vehicles in the traffic stream, the number of stops and delays, fuel consumption, emission reduction, energy saving, and life cycle cost.

2.2 Task Set 2. Design an SMFe-bus to Provide Comparable Service to the Cobb County BRT Bus

2.2.1 Task 2.1 Document State-of-the-art Practices in (Hybrid) Electric Vehicle Propulsion and Steering Systems

The literature and practices related to (hybrid) electric vehicle, i.e., (H)EV, propulsion and steering will be reviewed and documented to guide the project forward.

2.2.2 Task 2.2 Determine Vehicle and Service Specifications Proposed for Cobb County's BRT

The vehicle and service specifications proposed for Cobb County's BRT will be identified and documented. This includes parameters such as bus dimensions and capacity, bus turning radius, service route length, and peak headway. The SMFe-bus will be designed to match or exceed those identified specifications.

2.2.3 Task 2.3 Develop Laboratory Prototype of SMFe-bus Propulsion and Steering System

The procedure for developing the lab prototype for the propulsion and steering system will be as follows:

- Select and procure in-wheel motors suitable for a lab prototype of an SMFe-bus with two-wheel-drive lead and follower modules.
- Select and procure batteries and controllers compatible with the chosen in-wheel motors.
- Select and implement the steering method/system best-suited to the "virtual coupling" module-following requirement.

- Design and construct a simple chassis for this lab prototype based on the selected motors, batteries, controllers, and steering system. Develop the propulsion control system and the interface between the high-level (module-following) computer and the vehicle propulsion control system.
- Integrate the procured motors, batteries, controllers, and steering system into the lab prototype's chassis.

2.2.4 Task 2.4 Carry Out Tests and Improvements of SMFe-bus Prototype

As the procured components are received, they will be tested independently to ascertain that they are fully functional and meet the requirements. Any observed deficiencies will be corrected. Then the integrated system will be tested to ensure its functionality both before and after it is integrated into the lab prototype's chassis.

2.2.5 Task 2.5 Determine Design Specifications for an SMFe-bus Propulsion System that Can Provide Service Matching that Proposed for the Cobb County BRT

Design specifications will be developed for an SMFe-bus propulsion system that can provide service matching that proposed for the Cobb County BRT. The design and specifications will take into account the lessons learned from the lab prototype's design and test results.

2.2.6 Task 2.6 Assess Impact of a Fully Electric Propulsion System on Operations and Cost

Once the design for the SMFe-bus propulsion system has been specified, the researchers will estimate the cost of a fully electric bus and compare that to the costs of similar-capacity fossil-fuel buses and hybrid-electric buses. They will also study how the sizing of the SMFe-bus module's battery pack affects the type of battery-charging infrastructure required and how the time needed for battery charging affects service operations. This will lead to recommendations on how to balance battery pack sizing versus needed battery-charging infrastructure, and operational performance and associated costs.

2.3 Task Set 3. Develop the Hardware and Software Needed to Demonstrate the Concept of Virtual Coupling (Module-following) for Two Prototype SMFe-bus Modules

2.3.1 Task 3.1 Review Literature and Practices

The research team will thoroughly review papers related to vehicle tracking, robot tracking, and related technologies. The advantages and disadvantages of various techniques and approaches proposed in the literature will be identified. Meanwhile, practices and activities related to vehicle and/or robot tracking, which have been implemented by other researchers, will be reviewed.

2.3.2 Task 3.2 Develop Computer Vision Algorithms for Object Recognition

The researchers will develop efficient computer vision algorithms to recognize objects (vehicles) robustly and reliably. The developed algorithm can visually recognize a moving object at a desired speed. The target object may move straightly or turn with a specific angle.

2.3.3 Task 3.3 Develop the Communication Software for the Low-level Vehicle Control System

The software will be developed to allow communication between the high-level vehicle tracking system and the low-level motion control system. The researchers will establish the communication prototypes, design the communication modes, and determine the communication content.

2.3.4 Task 3.4 Develop Machine Learning Algorithms for Modular Tracking

A machine learning algorithm will be developed to determine optimal motion commands for the vehicles (modular) through fusing the visual information extracted from the vision subsystem and distance information measured by the sonar/laser sensors. After enormous offline training, the vehicles are expected to autonomously learn correct tracking actions in various situations or environments.

2.3.5 Task 3.5 Field Test and Improvement

The above algorithms and the developed system will be first tested and validated in a laboratory environment. Once they are successful, some field tests will be carried on. The test results will be evaluated and used to improve the algorithms and software/hardware design.

The work done and the results achieved are detailed in the following three chapters corresponding to the three task sets, although they are presented in the order of task set 2, 3, and then 1. This order is because task set 3 depended on the successful completion of task set 2, whereas the completion of task set 1 was independent of the others and served essentially as a justification for them.

This page is intentionally left blank.

3 TASK SET 2: Design and Prototyping of an SMFe-bus

3.1 State-of-the-art Practices in (Hybrid) Electric Vehicle Propulsion and Steering Systems

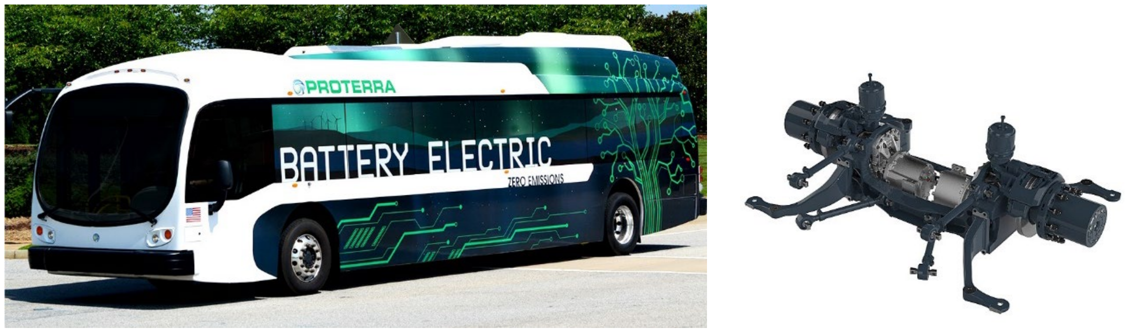
The literature and practices related to (hybrid) electric vehicle propulsion and steering were reviewed to guide the way forward on this project.

Most of the world's major bus manufacturers have (H)EVs in their product lineup. Some also sell battery electric vehicles (BEVs) or only sell BEVs; one example of the latter is Proterra (see Fig. 1a) [28], a U.S.-based company. With the ongoing trend of decreasing battery pack costs [29], BEVs appear to have the edge over (H)EVs due to their simpler powertrain design, especially for vehicles to be introduced 3–5 years from now or later. Hence, this project focused on a BEV, instead of an HEV.

Presently, most BEVs for transit applications use a single electric motor to drive the two wheels attached to a single axle. However, Proterra has recently introduced a BEV with two electric motors to independently drive the two wheels attached to a single axle (see Fig. 1b) [30], which exploits the power density of electric motors and improves steering control of the bus (i.e., it allows for torque vectoring [31]). A two-speed gearbox interposed between the motor and its corresponding wheel is used in that bus to improve hill-climbing performance, but this is at the expense of increased space, weight, and cost. For the present project, there is not a hill-climbing performance objective to be satisfied, hence a gearbox is unnecessary; moreover, the SMFe-bus vehicle is slim and lateral space is a limiting factor. Thus, the researchers focused on hub (i.e., in-wheel) motors where the motor is mounted within the hub of a wheel [32]. In-wheel motor (see Fig. 2) advantages include the following:

- No mechanical gearings from the motor shaft
- A wide range of application, including conversions and hybridization
- Concept electric cars, and light commercial vehicles
- Electromechanical topology

- Independent distribution and variation of power (torque) to individual wheels
- Front/rear wheel drive vectoring in all-wheel-drive (AWD) vehicles for better traction and road handling



(a) (b)

FIGURE 1

Proterra Bus (a) Catalyst BEV [28]; (b) DuoPower Axle Assembly [30]

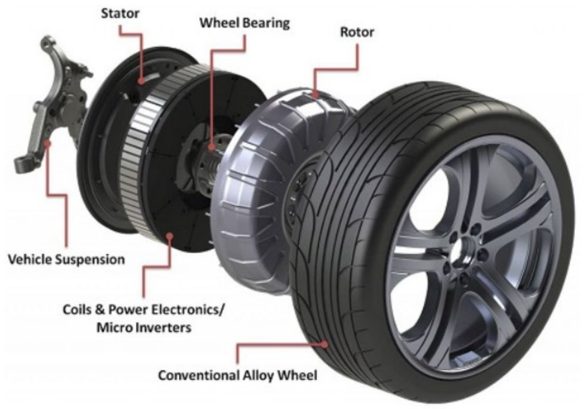


FIGURE 2

An In-Wheel Motor's Main Components [33]

Manufacturers of hub (in-wheel) motors for EVs include Protean Electric [33], Elaphe [34], and Kelly Controls [35].

The 35-ft version of the Proterra Catalyst (Fig. 1a) can probably be modified in a relatively straightforward manner to serve as the lead module of the SMFe-bus, with the follower module requiring somewhat more significant modifications (in particular, to provide it with module-following capability).

3.2 Vehicle and Service Specifications Proposed for Cobb County’s BRT (as an example)

Vehicle and service specifications were identified and documented, including parameters such as bus dimensions and capacity, bus turning radius, service route length, and peak headway. Cobb County’s Department of Transportation has proposed implementing a BRT system, called Connect Cobb, which will comprise the following:

- A corridor running from Kennesaw State to Midtown
- A 25.3-mile route with 15 stops, shown in Fig. 3
- Construction of expanded roads, as illustrated by Fig. 4

This project was budgeted at about $500 million [13], which is equivalent to a cost of almost $20 million per service mile, although that is still lower than the cost associated with rail-based mass transit [14]. The SMFe-bus will be designed (see Section 2.2.5) to match or exceed the identified specifications of Cobb County’s arterial rapid transit (ART) bus, including those shown in Fig. 5. Note that ART is sometimes used in place of BRT to emphasize that the route runs through existing high-density, mixed-use arterial corridors.

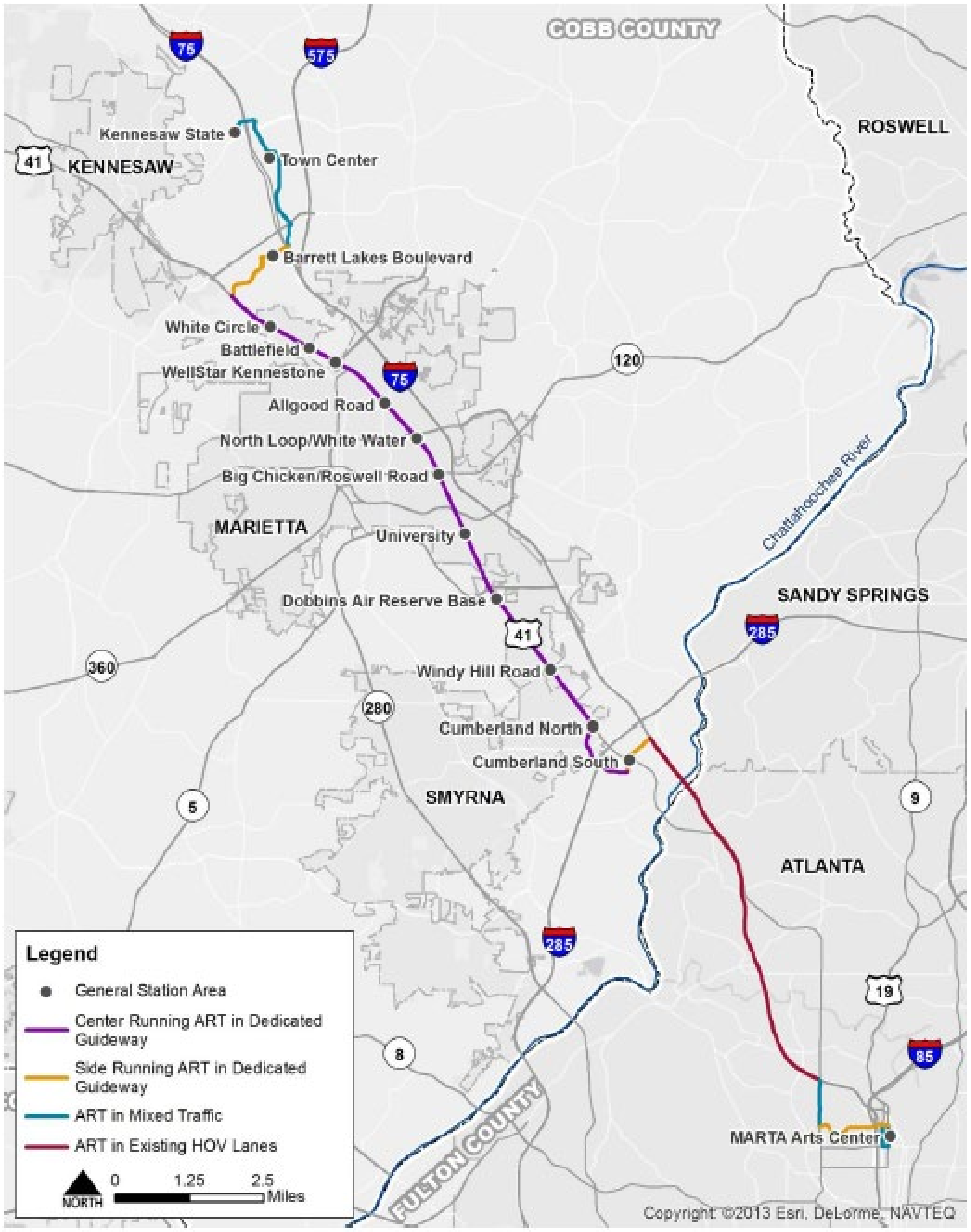


FIGURE 3

Proposed Connect Cobb ART Corridor Showing Traffic Configuration and Stations [Source: Cobb County DOT]

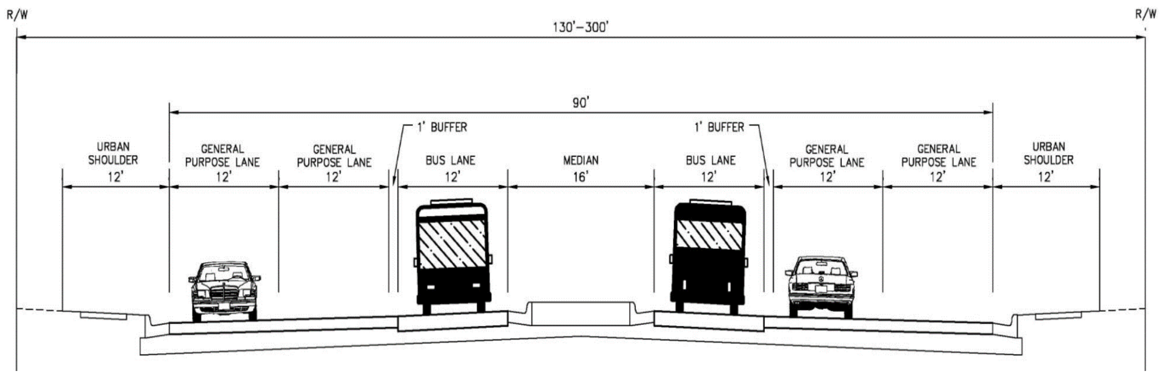
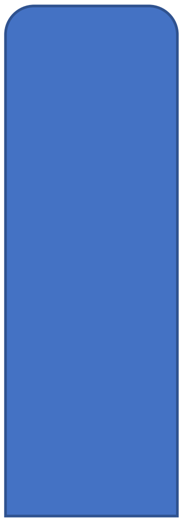


FIGURE 4

Proposed Typical Section on US 41 / Cobb Parkway [Source: Cobb County DOT]



ART Bus Specifications		**SMFe-Bus Specifications**
Compressed Natural Gas or Diesel-Electric Hybrid	**Fuel Type**	Battery Electric
110 (60 Seated plus 50 Standing)	**Capacity**	105 (24+27+27+27 seated, 4 modules, 9 rows, with 1 driver or 3 pax seated per row), or 105 (33+36+36 seated, 3 modules, 12 rows, with 1 driver or 3 pax seated per row)
62	**Length (ft)**	27 to 36
8.5	**Width (ft)**	6 to 6.25
11	**Height (ft)**	10
39	**Turning Radius (ft)**	33.1 (for each 36' long module)
68000	**Weight (lb)**	21774 to 29032
At station	**Fare Collection**	At station
60	**Max Speed (mph)**	60
Both Sides	**Door Location**	One Side

Lead
Follower
Follower

FIGURE 5

Comparing Specifications of Cobb County's ART to Proposed SMFe-BRT

3.3 Lab Prototype of SMFe-bus Vehicle

A laboratory prototype of the SMFe-bus vehicle with two modules—a lead module and a follower module (see Fig. 6)— was designed and constructed as a proof-of-concept demonstrator. Their subsystems are essentially identical except that the follower module has sensors and a high-level computer to enable it to track the lead module positioned and moving ahead of it. The design and construction processes included the following steps, and the details for each subsystem are presented in the following subsections.

- Select and procure in-wheel motors suitable for lab prototype of an SMFe-bus with two-wheel-drive lead and follower modules.
- Select and procure batteries and controllers compatible with the chosen in-wheel motors.
- Select and implement the steering method/system best suited to the "virtual coupling" module-following requirement.
- Design and construct a simple chassis for this lab prototype based on the selected motors, batteries, controllers, and steering system. Develop the propulsion control system and the interface between the high-level (module-following) computer, and the vehicle propulsion control system.
- Integrate the procured motors, batteries, controllers, and steering system into the lab prototype's chassis.

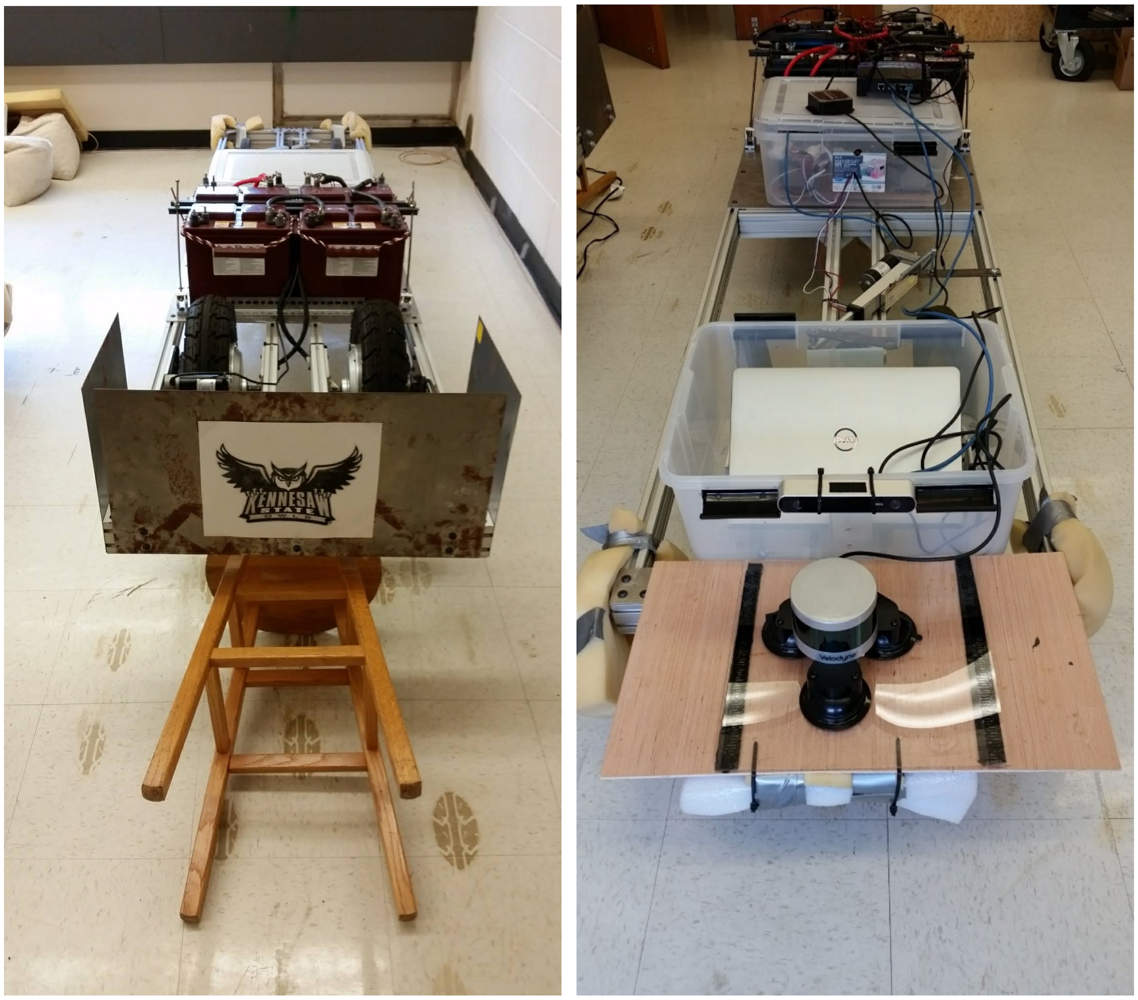


FIGURE 6

Lab Prototype of the SMFe-bus Vehicle with Two Modules:
a Lead Module and a Follower Module

3.3.1 Hub Motors and Motor Controllers

To prove the above-described concept of the SMFe-BRT vehicle and determine needed adjustments to the proposed vehicle design, a ⅓-scale prototype was developed. For this prototype (with two ⅓-scale modules), it was estimated that each module would weigh about 250 kg (10 kg for floorboard, 30 kg for rectangular chassis, 120 kg for batteries, 70 kg for hub motors and wheels, 20 kg for electronics and sensors). Furthermore, to attain linear speeds of at least 15 mph (24 km/h), the two hub motors per module were sized at 2 kW each. A few suppliers were considered, and the research team decided that the motor and motor controller would be acquired from Kelly Controls, LLC.

The selected motor controller (Kelly KLS7222H [36]) is a sinusoidal wave drive type controller that reduces the operation noise and the switching losses (by up to one-third). This motor controller uses high-power metal-oxide semiconductor field-effect transistors (MOSFETS), space vector pulse-width modulation (SVPWM), and field-oriented control (FOC) to achieve a peak efficiency of 99%. The controller drives the selected 48 V 2 kW brushless DC hub motor with 10-inch tires (see Fig. 7) [37] with the help of Hall sensors, and is able to rotate it fast enough to achieve linear speeds of up to 57.1 km/h. A 48 V 100 Ah battery pack is to be used to supply the motors via the main contactor (with a precharge resistor across its contacts) and the motor controller.

To test this prototype vehicle, it would need to be remotely controlled. This is addressed in Section 3.3.4.

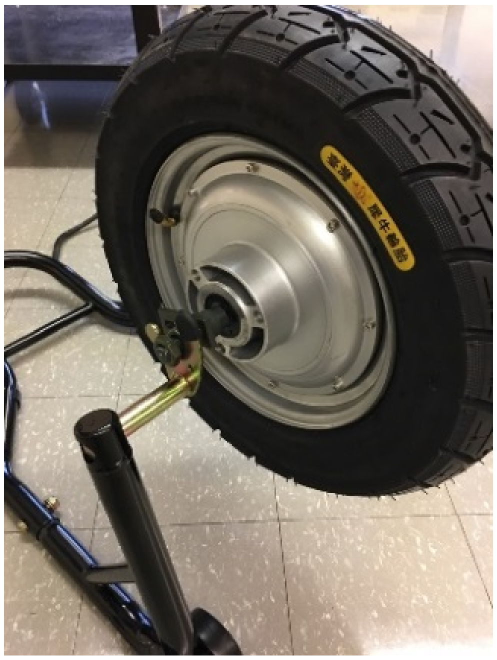


FIGURE 7

The Selected 48 V 2 kW Brushless DC Hub Motor with 10-inch Tire Mounted on a Stand for Testing

3.3.2 Batteries

One important consideration when selecting the motor and motor controller was their voltage rating. While a higher operating voltage would allow for reduced operating current, reduced wiring size, and increased operating time, it would also increase the weight and cost of the battery pack. Hence the 48 V was a reasonable compromise when considering all the above factors. Initially, the researchers chose the Trojan SCS150 (12 V) deep-cycle wet lead-acid battery to implement this pack, then for the second vehicle module they used the Duracell SL124MDC (12 V) deep-cycle wet lead-acid battery, which was found to be a better cost-performance alternative, although its capacity (20 hr) is 75 Ah instead of 100 Ah for the SCS150.

3.3.3 Chassis

To obtain a fairly lightweight yet sufficiently sturdy chassis for the prototype SMFe-bus vehicle, the researchers decided to construct it using aluminum as the main material. Then, to ease the design, reduce the machining requirements, and speed up the assembly process, they chose to use T-slotted aluminum extrusions. The chassis design factored in the requirements for the various subsystems, starting with the hub motors, and including an area for mounting/supporting the heavy battery pack and an area for placing the electronics (and also sensors for the follower module). Fig. 8 shows a rendering of the designed chassis for the scaled prototype.

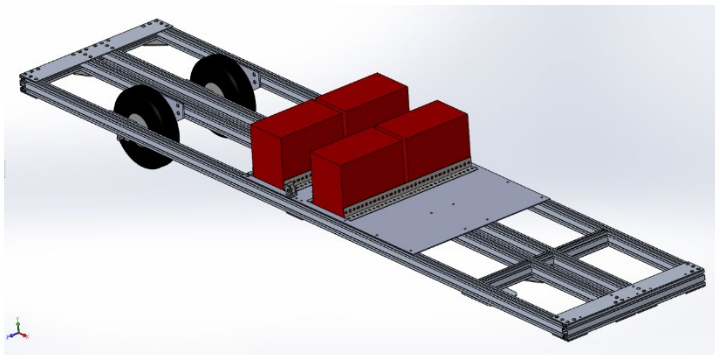


FIGURE 8

Rendering of the Designed Chassis for the Scaled Prototype

3.3.4 Radio Control System

The prototype lead module's propulsion system, which is controlled by remote means, consists of the above-mentioned hub motors and motor controllers, together with a Raspberry Pi 3 computer, a digital-to-analog converter, and a radio-frequency (RF) remote control (RC) system.

To emulate a driver's operation of the lead module, the RC system is used to communicate the Throttle (Forward or Reverse), Brake, and Left or Right commands. The joysticks on the selected Turnigy 5X [38] RC's transmitter are used to command these actions, which are then communicated to the RC receiver at a frequency of 2.4 GHz. A toggle switch on the RC transmitter can also be used to initiate emergency stopping (Brake) in case of a dangerous situation. The receiver's channels output servo-type pulse-width modulated signals (with pulses ranging between 1 and 2 ms in width) based on each transmitter joystick's position, which are read by the module's low-level Raspberry Pi 3 computer. Fig. 9 shows how the various components connect together to permit remote operation of the prototype vehicle's lead module for testing purposes.

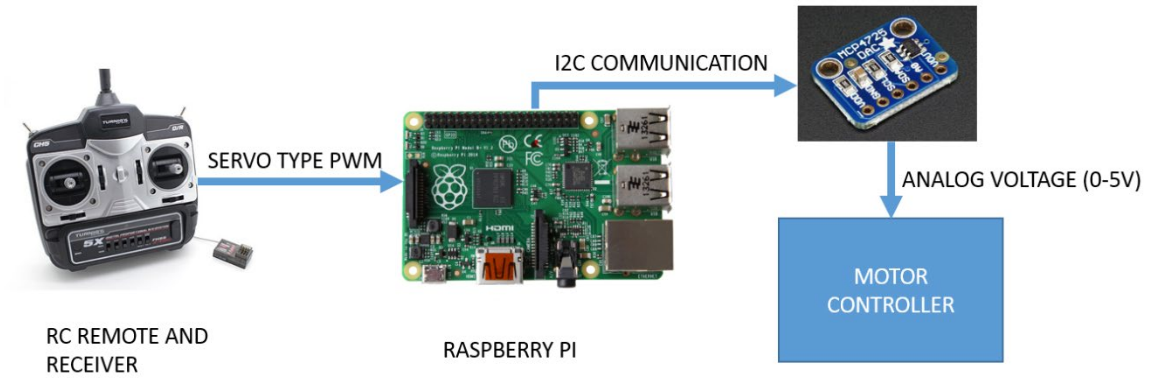


FIGURE 9

Connection Diagram for Motor Control Using RC System

3.3.5 Low-level Computer (RPi) and Electronics Board

The electronics board (see Fig. 10) of the lead and follower modules are the same, differing only in their firmware. First and foremost, the selected Raspberry Pi 3 computer [39] is powered by a component that converts the 48 V from the pack of four 12 V batteries, connected in series, down to 5 V. Then, there is a Raspberry Pi Cobbler/connector board that provides a convenient breakout of the General Purpose Input Output (GPIO) pin selection of the Raspberry Pi.

The signals output from four channels of the RC receiver are sent as inputs to the GPIO pins of the Raspberry Pi for processing. These channels represent the Forward or Reverse switch command, Throttle command, the Left or Right command, and the Brake command.

Due to the Raspberry Pi 3 lacking analog outputs, an MCP4725 digital-analog converter (DAC) [40] was selected for producing the analog voltage needed by the motor controller. This DAC, with 12-bit resolution, can output a voltage between 0 and 6.5 V while the motor controller selected for the prototype vehicle requires a throttle input of 0 to 5 V. The DAC communicates with the Pi 3 using the I2C communication protocol. A Python script is run on the Pi 3 to read the input from the RC receiver and then write an I2C signal to the DAC; this script (see "lead_mod_t9.py" in Appendix A) runs continuously on the lead module during testing, while it is run on the follower module only while maneuvering it out of the lab and to the test site. Specifically, the joystick on the RC transmitter being all the way down produces 0 V of motor throttle and the joystick all the way up produces 5 V of motor throttle. Since one DAC is needed for each of the two hub motors, the bus address of one of the DACs was changed from the default value of (0×62) to (0×63). This allows different voltages, if needed, to be applied to the left and right motors.

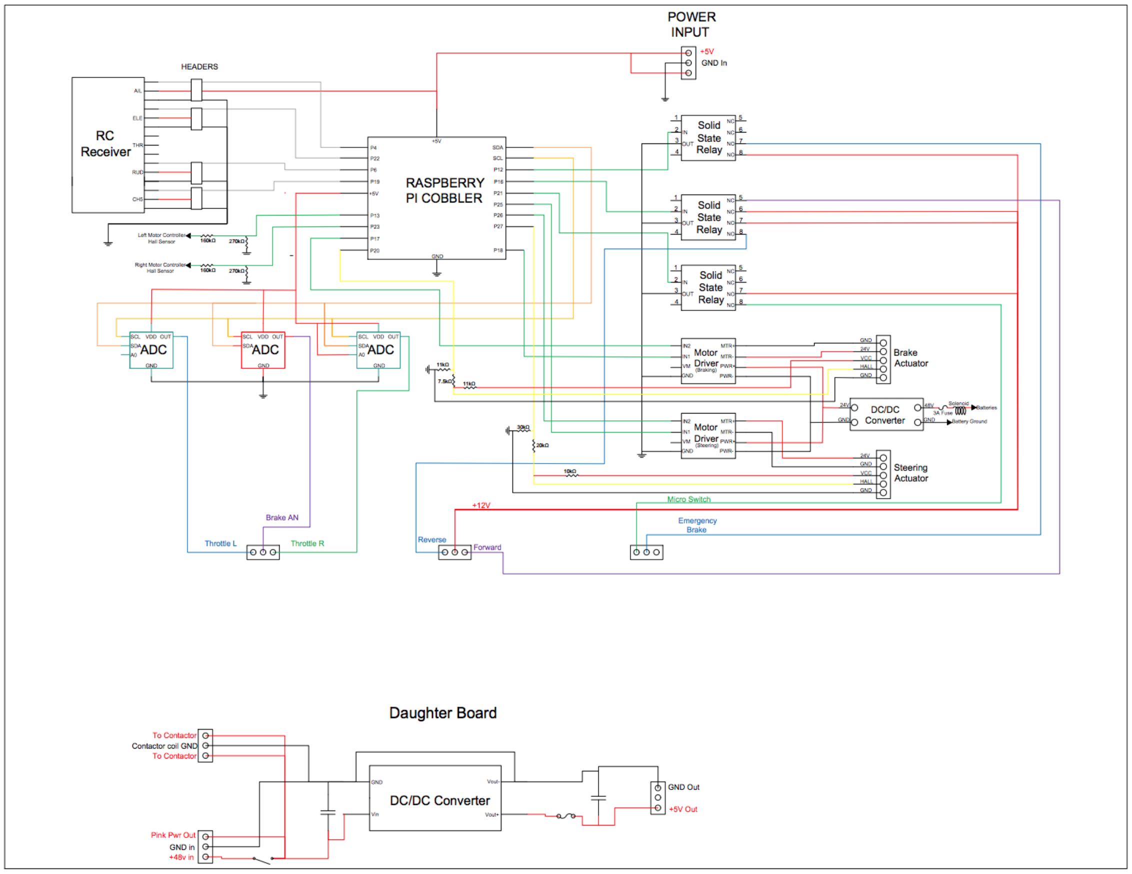


FIGURE 10

Connection Schematic for the Electronics Board

For the follower module only, the purpose of the "lmsc_v3_27.py" program is to interface between the high-level controller (laptop) and this module's subsystems that vary its speed and direction to achieve module following. A flowchart of this program is displayed as Fig. 11. When this program is executed, it first extends the braking actuator to release the brakes applied to the hub motor's rotors. The program then centers the steering using the steering actuator. After this, transmission control protocol (TCP) communication is established with the high-level controller (laptop) that sends this program the values it needs for the hub motor speed (in RPM) and the steering wheel angle (in degrees, with right of straight-ahead having positive values and left of straight-ahead having negative values), and it obtains and initializes these values immediately.

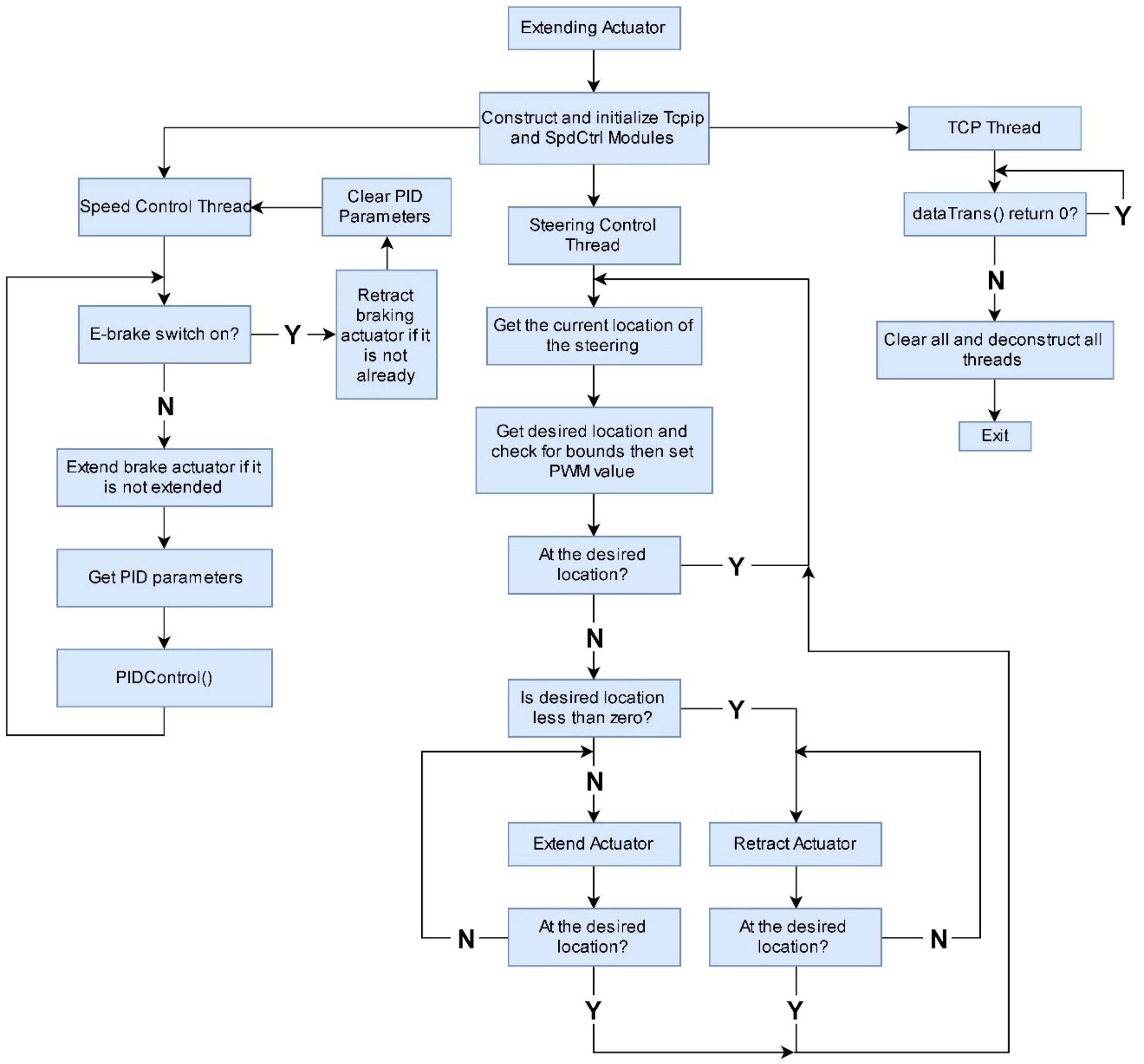

Extending Actuator
Construct and initialize Tcpip and SpdCtrl Modules
TCP Thread
dataTrans() return 0?
Y
N
Clear all and deconstruct all threads
Exit
Speed Control Thread
Clear PID Parameters
E-brake switch on?
Y
Retract braking actuator if it is not already
N
Extend brake actuator if it is not extended
Get PID parameters
PIDControl()
Steering Control Thread
Get the current location of the steering
Get desired location and check for bounds then set PWM value
At the desired location?
Y
N
Is desired location less than zero?
Y
N
Extend Actuator
Retract Actuator
At the desired location?
N
Y
At the desired location?
N
Y

FIGURE 11

"lmsc_v3_27.py" Flowchart

The motor speed thread is then set up. In this thread, the emergency brake trigger on the RF remote will be checked. If it is triggered, then the program will retract the braking actuator to apply the brake and will set the motor speed control to zero. If the emergency brake is not applied, then it will assign the desired values for the motor RPM that were received from the TCP communications earlier.

The steering thread is then set up. In this thread, the current location of the steering actuator will be acquired. The desired location for the steering that is received from the high-performance controller will be checked so that the minimum and maximum bounds are not passed. The steering actuator uses a pulse-width modulation (PWM) value so that the speed of the actuator can be varied as a function of the distance that the actuator must travel. This PWM value will be calculated in this thread and then sent to the actuator to control the steering.

Although the initial TCP communication is started at the beginning of the program, there is still a thread making use of the TCP protocol for the remainder of the operation. The TCP communication thread is then set up. In this thread, the RPM of the motors will be checked, and the proportional–integral–derivative (PID) controller's constants will be received. The desired RPM and angle for the motors and steering will then be received from the high-performance controller, and the angle will be immediately calculated to find the bit value that the analog-to-digital converter (ADC) will read from the slide pot.

After these three threads are set up, they will all be started at the same time.

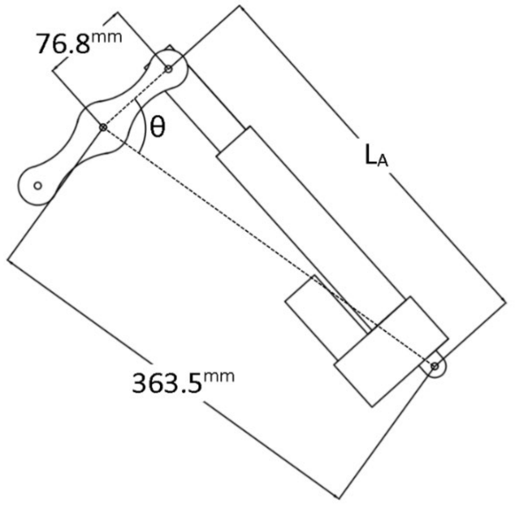
The program now enters an infinite loop that will constantly check for the thread flags to be triggered. If this flag is triggered, then all of the threads will be terminated. The voltage to the motors will be shut off, the PWM values to the steering actuator will be set to zero, the TCP communication will be closed, the emergency brake and actuator brakes will both be applied, and the GPIO pins will be deactivated.

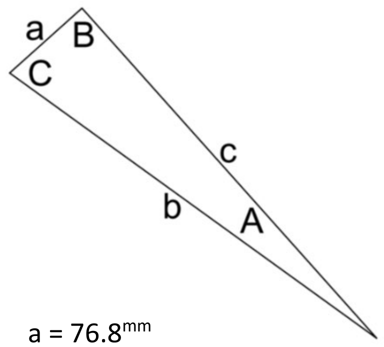
3.3.6 Steering Method/System

The steering actuation system is based on the conventional go-kart steering system (see Fig. 12 diagram). Calculations for this system were performed to find an equation that would relate the required stroke length of the steering actuator to the desired angle of the front (steering) wheels.

What is shown in the figure is the actuator arm and where the value of angle θ is being obtained. The law of cosines is used since there are two known sides for the triangle that the actuator creates. Two unknowns are left (i.e., θ and the actuator length). The θ or the actuator length can be set up as functions of one another, producing a nonlinear output. This illustration also shows the minimum and maximum values that the arm will be able to extend. Lastly, several values for the θ are calculated over the extension of the actuator length in order to find where the steering column will end.

Actuator Length [L_A]= 287^{mm} to 419^{mm} (+/- 2^{mm})
Steering Actuator = 76.8^{mm}





a = 76.8^{mm}

b = 363.5^{mm}

c = L_A

$L_A^2 = 76.8^2 + 363.5^2 - 2*74.6*363.5*\cos\theta$ 🡸 Using the law of cosines

$$\theta = \cos^{-1}\frac{L_A^2 - 138030}{-55834}$$

$$\theta_{\text{min}} = \cos^{-1}\frac{287^2 - 138030}{-55834} = 4.5^\circ$$

$$\theta_{\text{max}} = \cos^{-1}\frac{419^2 - 138030}{-55834} = 132.2^\circ$$

$$\theta_{\text{straight}} = \cos^{-1}\frac{350^2 - 138030}{-55834} = 73.85^\circ$$

$$\theta_{\text{adjusted}} = \cos^{-1}\frac{L_A^2 - 138030}{-55834} - 73.85^\circ$$

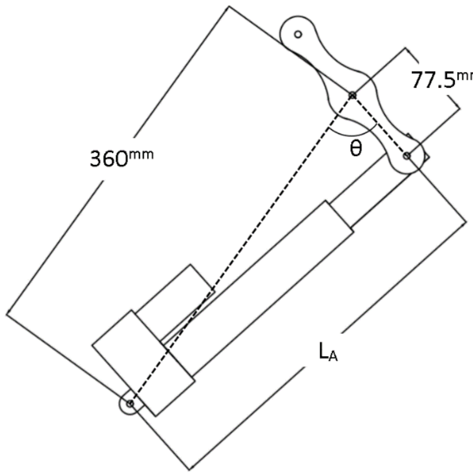
θ_{adjusted} will be zero when the wheels are straight, be positive when they are turned to the right, and negative when they are turned to the left.

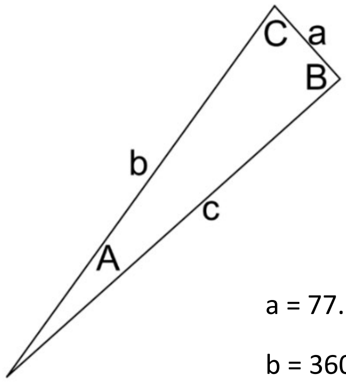
FIGURE 12

Steering Actuation System Diagram

Fig. 13 shows a tidier version of the previous figure. The measurements were retaken for the calculations, so the constant values in the equation are slightly different; this is the equation used in the other calculations in Fig. 13.

Actuator Length [L_A]= 287^{mm} to 419^{mm} (+/- 2^{mm})
Steering Actuator = 76.8^{mm}





a = 77.5^{mm}

b = 360^{mm}

c = L_A

$L_A^2 = 77.5^2 + 360^2 - 2*77.5*360*\cos\theta$ 🡐 Using the law of cosines

$L_A^2 = 135606 - 55800*\cos\theta$

$L_A^2 - 135606 = -55800*\cos\theta$

$$\theta = \cos^{-1}\frac{L_A^2 - 135606}{-55800}$$

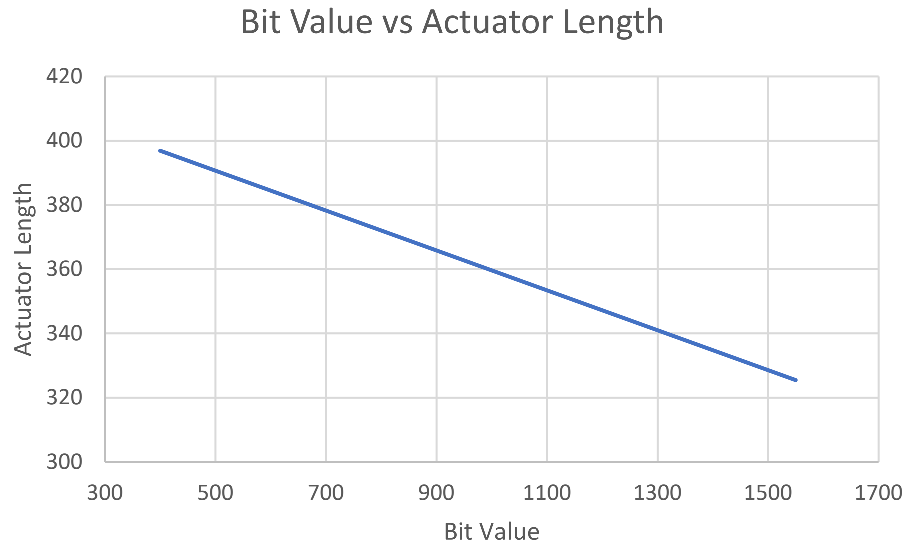
$$L_A = \sqrt{135606 - 55800\cos\theta}$$

FIGURE 13

Derivation of Equation Relating Steering Wheel Angle and Steering Actuator Length

Fig. 14 shows the length of the actuator that is being read into the Raspberry Pi via a proportionally equivalent slide potentiometer, which produces an output voltage that varies as the length of the actuator changes and moves/repositions the attached potentiometer tab along with it.

This figure first illustrates with a graph the change in the bit values read into the Raspberry Pi from the slide pot. The graph is of the actuator length versus the bit values read, using the minimum and maximum actuator lengths and respective bit values as the data points. An equation for the best-fit straight line through these points was found and then plugged into the equation presented in Fig. 13. "DL" replaces the bit value variable "b" as it is the variable name used in the Python program.



L_A = Actuator Length
b = Bit Value
$L_{A,min} = 325.44^{mm}$
$L_{A,max} = 396.88^{mm}$
$b_{min} = 1550$
$b_{max} = 400$

$$m = \frac{L_{A,min} - L_{A,max}}{b_{max} - b_{min}} = \frac{325.44 - 396.88}{1550 - 400} - .0621$$

$L_A = -.0621*b + C$

$C = L_A + .0621*b = 325.44 + .0621*1550 = 421.7$

$L_A = -.0621*b + 421.7$

FIGURE 14

Derivation of Equation Relating Steering Actuator Length and Bit Value of ADC

3.3.7 Braking Method/System

Braking of each module is done in two ways. The first is electronic or regenerative braking, which is carried out by the motor controllers as discussed in Section 3.3.1 above. But for safety (emergency stop) reasons, a mechanical disk brake system was also implemented on each module to more quickly slow down and possibly fully stop the rear wheels (with the hub motors). This system is detailed as follows.

The disk brakes on the SMFe-bus slow down and stop the vehicle while in operation. For the lead module, this will be commanded by its RC system transmitter's operator and implemented by the Raspberry Pi program for RC operation (see "lead_mod_t9.py" in Appendix A). For the follower module, this will be commanded in the event of an emergency situation by the Emergency Brake switch on its RC system transmitter, which the Raspberry Pi program for automatic following operation will interpret as discontinuing automatic following and fully applying the disk brakes.

The hydraulic brakes are engaged when a linear actuator pulls a cable attached to the brake lever. The brake lever provides the leverage for a push rod to force fluid through the brake lines and into the calipers. The pressure inside each caliper forces a piston to exert force on the brake pads and create friction against the (spinning) rotor/disk. This friction is what slows and eventually stops the wheels.

The main hardware components of the braking system consist of the following (see Fig. 10 – Connection Schematic for the Electronics Board):

1. Raspberry Pi 3
2. Adafruit DRV8871 motor driver
3. Zoom Industrial 4″ linear actuator with integrated Hall effect sensor
4. 48–24 V DC/DC converter
5. Hydraulic brake system
    a. Master cylinder with lever

b. Brake fluid lines

c. Caliper, brake pads, and rotor

As described, the brakes are applied when the 4″ actuator contracts. Conversely, the brakes are released when the actuator extends. This forward and reverse motion is controlled by altering the polarity of the voltage applied to the actuator using the DRV8871 motor driver. Moreover, the two logic inputs to the motor driver originate from the Raspberry Pi 3, which sends the signals to extend and retract, depending on the joystick position of an RC system transmitter (for the lead module), or if the Emergency Brake switch of an RC system transmitter has been activated (retract only, for both the lead and follower modules).

The control program for each SMFe-bus module is written in Python (see Appendix), which includes the braking routine, and is run on the Raspberry Pi 3 low-level controller. When the program begins its execution, the linear actuator is extended to its maximum position, which releases the brakes completely. In the main routine of the Raspberry Pi program for RC operation (lead module), the desired position and the actual position of the actuator are compared. That is, the desired position of the lead module RC system's joystick is compared with the actual position of the actuator; this actual position is determined by either incrementing or decrementing a counter based on the pulses from the Hall sensor inside the actuator. The more the joystick is pulled back, the more the actuator contracts, and the same is true for the opposite direction.

The actuator has a software-set retraction limit to stop the actuator from breaking the cable that tethers its tip to the lever on the brake's master cylinder.

3.3.8 Tests and Improvements of SMFe-bus Prototype

As the procured components were received, they were tested independently to ascertain that they were fully functional and met their requirements. Then, each of the above-mentioned subsystems was tested independently to ensure its functionality both before and after it was integrated into the lab prototype's chassis. Finally, the complete and integrated power and propulsion system, operated by a remote-control system working properly, was tested first indoors and then outdoors to assess motor throttling (forward and reverse), steering, regenerative braking, and emergency braking. Each problem detected during the tests was corrected. This resulted in module prototypes that yield performance that meets or exceeds the technical objectives (mainly straight-line speed and cornering speed) proposed for their power, propulsion, steering, and braking systems.

3.4 Design Specifications for an SMFe-bus Propulsion System that Can Provide Service Matching That Proposed for the Cobb County BRT

Design specifications were developed for an SMFe-bus propulsion system that can provide service matching that proposed for the Cobb County BRT. The design and specifications took into account the lessons learned from the lab prototype's design and test results.

Calculations were performed to determine the maximum power requirement for propelling each SMFe-bus module, assuming a curb weight of 29,032 lb for a 36-ft-long 36-passenger follower module (the corresponding lead module will have room for 33 passengers, 1 driver, and 1 wheelchair-bound passenger), in order to properly specify the electric motors. It was further assumed that the vehicle will need to accelerate from 0 to 30 mph in 9 seconds, and also reach a maximum speed of 60 mph. Hence, the motors need to be about either 176 hp (132 kW) for two hub motors or 88 hp (66 kW) for four hub motors. After estimating the motors' power rating, a current-versus-time profile was roughed out, assuming a 370 V battery pack and that the distance between each of the 14 total stops along the planned Cobb Parkway BRT corridor [41] was the same, which implied that 145 Ah of charge was needed for a one-way trip, as depicted in Fig. 15. Selecting a 600 Ah capacity battery pack (equivalent to 222 kWh) would allow the vehicle to make two round trips without recharging, although this neglects the energy needed for heating/cooling and other electrical/electronic systems.

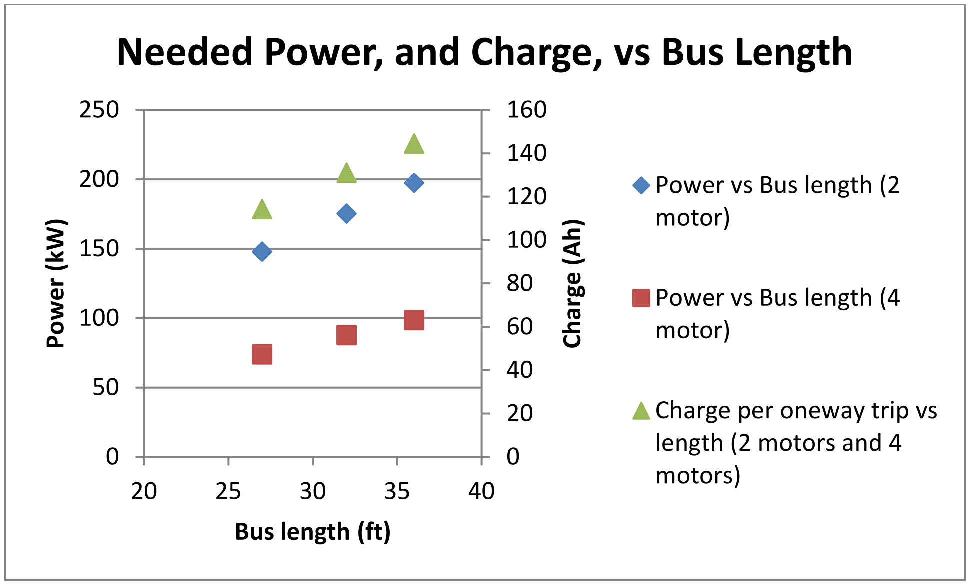


FIGURE 15

Different Power and Charge Requirements as a Function of Bus Module Length

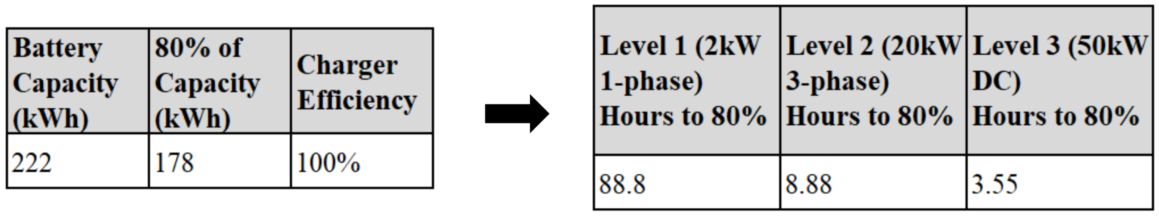
3.5 Impact of a Fully Electric Propulsion System on Operations and Cost

Once the design for the SMFe-bus propulsion system was specified, the research team could estimate the cost of a fully electric bus and compare that to the costs of similar-capacity fossil-fuel buses and hybrid electric buses. The researchers could also study how the sizing of the SMFe-bus module's battery pack affects the type of battery-charging infrastructure required and how the time needed for battery charging affects service operations. This finding then leads to recommendations on how to balance battery pack sizing versus needed battery-charging infrastructure, and operational performance and associated costs.

The SMFe-bus vehicle will be all-electric as compared to the compressed natural gas or diesel-electric hybrid BRT vehicle planned for the Connect Cobb project; hence, it will be more environmentally friendly.

As noted in Section 3.4, a 600 Ah capacity battery pack (equivalent to 222 kWh for a 375 V power and propulsion voltage) would allow the vehicle to make two round trips without recharging, although this ignores the energy needed for heating/cooling and other electrical/electronic systems.

During the day, the vehicle's modules can receive a quick charge at the terminating stations, which should allow it to be able to run continuously throughout the day without downtime for a complete recharging. Of course, if it is a follower module that is temporarily taken out of service during a low-demand period, it can be recharged during that time. At night, the vehicles will be connected to the charging stations to charge the batteries to their full capacities. There are three different types of chargers that can be used that have different ratings [42], as shown in Fig. 16. Level 1 and 2 chargers use single-phase or 3-phase AC supply and can deliver power from 2 to 20 kW. Level 3 chargers require a 3-phase AC supply and can deliver power from 20 to 240 kW, which permits fast charging.



Battery Capacity (kWh)	80% of Capacity (kWh)	Charger Efficiency
222	178	100%

Level 1 (2kW 1-phase) Hours to 80%	Level 2 (20kW 3-phase) Hours to 80%	Level 3 (50kW DC) Hours to 80%
88.8	8.88	3.55

FIGURE 16

Charging Times for Different Levels of Charging

The researchers considered how these calculations compare to the recently published specifications for a commercially available, fully electric transit bus. Table 1 compares several key features, which indicate that the calculations are quite reasonable.

TABLE 1
Comparison of Several Key Features of Proposed SMFe-bus to the Proterra FC Bus

	SMFe-bus	Proterra FC
Total energy (kWh)	222	94
Top speed (mph)	60	65
Acceleration 0–20 mph (s)	6	4.5
Motors (kW)	2×132	2×190
Curb weight (lb)	29,032	28,925
Total energy (kWh)	222	94
Gearbox	None	2-speed auto-shift

4 TASK SET 3: Development and Demonstration of Module-following Control

4.1 Review of Literature and Practices

Autonomous vehicle tracking has attracted significant attention in the intelligent transportation community. Most research activities have focused on developing a model-based leader–follower controller so that a follower vehicle can track its predecessor autonomously and reliably. For example, Loria et al. proposed a leader–follower nonlinear controller when the vehicles followed a straight path. Specifically, since the system to be controlled is not controllable (in the control theoretic sense of this word), they developed a controller that has a property of persistence of excitation to make the whole system stable [43]. Cruz-Morales et al. presented a leader–follower formation strategy for nonholonomic mobile robots, where the discrete kinematics models of the robots were derived, and the relative distance/angle model between the robots was developed. These models were employed to develop a control law for autonomous tracking [44].

Since the vehicle models are usually nonlinear, a number of researchers employed nonlinear control approaches to design a control law [45–50]. Paliotta and Pettersen [45] developed a distributed control law for leader–follower synchronization with disturbance rejection. The feedback linearization technique was utilized to derive the control law. Chen et al. [46] considered the measurement delays in a leader–follower formation control problem for nonholonomic vehicles. In particular, they extended the concept of input-to-state stability and integrated a Smith predictor with nonlinear small-gain assignment. Mori and Namerikawa [47] proposed a formation control algorithm based on a consensus algorithm and a leader–follower structure for a multi-UAV (unmanned aerial vehicle) system. They developed the control algorithm based on the Lyapunov stability theorem and linear matrix inequality (LMI) conditions when the communication between two vehicles was intermittent.

In a leader–follower control structure, since the trajectory of the leader vehicle can be naturally regarded as the predicted reference trajectory that the follower vehicle needs to follow, the model predictive control (MPC) technique was employed by many researchers to implement autonomous vehicle tracking [51–55]. For example, Maeda and Konaka presented an MPC controller to predict the predecessor's trajectory for a two-wheeled vehicle [51]. Dunbar and Caveney developed a distributed receding horizon controller for a vehicle platoon, and derived the sufficient conditions of string stability [53].

The techniques of sliding mode control, adaptive control and robust control are also very popular in designing a leader–follower controller [56–63]. For instance, Koroglu and Falcone proposed a controller for a platoon of autonomous homogeneous vehicles. They studied the string stability problem and derived the sufficient LMI conditions [56]. Chen et al. developed a sliding mode controller for distributed formation control of multiple mobile robots [58]. They also employed the backstepping method to study the formation control strategy. In addition, Chen, Torre, and Dong developed a distributed exponentially tracking controller for multiple wheeled mobile robots using adaptive control [60].

All of the above research is related to model-based control. In these research projects, the mathematical model of vehicle kinematics or dynamics was derived and then a controller was designed based on this model.

Only a few researchers have tried the model-free control methods thus far [64–68]. For example, Hung et al. employed the reinforcement learning algorithm to implement a leader–follower controller for fixed-wing UAVs in a stochastic environment [64], where Dyna-Q with a variable learning rate was employed by the agents to learn a control policy. Peng et al. proposed adaptive dynamic surface control for autonomous surface vehicles using neural networks [65]. Their developed approach was compared to a model-based control method. Rinaldi, Chiesa, and Quagliotti compared linear–quadratic regulator (LQR) control to neural network control in a leader–follower formation control task for quadrotor UAVs [66].

Leader–follower tracking control is usually appropriate to a vehicle team including two or three vehicles. If there are more than three vehicles, then the vehicles form a platoon. In a platoon of vehicles, leader–follower control is still at the core of the control strategy, but an additional issue called “string stability” must be considered, which means that the effect of disturbances will not be amplified throughout the string as the vehicle index increases. Peters and Mason proposed a leader–follower control strategy for a platoon of vehicles with non-homogeneous weights [69]. To guarantee the string stability, each follower vehicle receives not only the state of its immediate predecessor but also the state of the leader. Similar approaches were employed in other studies to implement coordinated control of a platoon of vehicles with string stability [70–75].

Clearly, controller design could be simplified if the vehicles can communicate with each other through a wired or wireless network and acquire the exact states of its neighbors and/or the leader in the platoon. As a result, it has been popular to employ explicit communication in vehicle tracking control to improve the system’s reliability and stability [76–79]. Ampountolas and Kring proposed a bus-to-bus communication strategy to acquire the current position and speed of the leader bus to improve cooperation between the buses [76]. Hu and Lemmon proposed a distributed switching control approach to achieve almost sure safety for leader–follower vehicle control. The vehicles can exchange their information over a wireless radio communication network to attain and maintain formations [78].

While explicit communications among vehicles simplify the controller design, the inevitable communication delays and disturbances could make the system unstable.

It is surprising that few researchers have employed computer vision in autonomous vehicle tracking; however, this is partly due to unreliable detections and slow response speed of a computer vision subsystem. Karras, Kyriakopoulos, and Karavas proposed a leader–follower scheme using vision-based implicit communications for underwater vehicles [80]. The relative positions between the vehicles were estimated using a computer vision algorithm, and a motion tracking controller was implemented for robust leader–follower tracking. Cruz-Morales et al. developed a leader–follower formation control strategy for nonholonomic mobile robots [44]. In particular, a Microsoft Kinect™ camera was employed to acquire the relative distance and angle between the robots, and a motion control law was derived to utilize the acquired visual information.

Here is a summary of existing research in autonomous vehicle tracking:

- Most researchers have focused on developing a model-based controller. Few investigators have paid attention to model-free control strategies.
- Explicit communication approaches have been utilized extensively to simplify the controller design.
- Computer vision is seldom applied to autonomous vehicle tracking even though the camera cost has been reduced dramatically and the computer vision algorithms have been improved significantly in recent years.

The researchers in this project contend that a model-free control approach has some advantages over a model-based control approach. Due to the complexity of a vehicle's kinematics and dynamics, it is challenging to derive its mathematical model accurately since it is quite nonlinear. Specifically, because the math model matches the real kinematics and dynamics of the vehicle only in a very small work zone, using an approximate linearized model often makes the control law fail when the vehicle is not working in the desired work zone. The researchers also argue that the explicit communication approach introduces additional controller stability issues due to unexpected communication delays and disturbances, even though it simplifies the controller design. Instead, here they propose employing some local sensing approaches (i.e., distance measurements and object detections) to improve the reliability of the vehicle tracking system. Furthermore, in view of the significant progress in computer vision research in the past years, they also propose integrating computer vision with other sensing approaches for vehicle tracking because they believe the vision subsystem can provide richer environmental information than other sensors.

4.2 Develop Computer Vision Algorithms for Object Recognition

In this project, the position of the leader vehicle is detected visually using a ZED™ camera and the SSD (Single Shot MultiBox Detector) deep learning technology [53]. In the past two or three years, there has been significant progress in deep learning or the deep neural network based on convolutional neural network (CNN) to detect visual objects. In particular, it was reported that deep learning has shown better image recognition capabilities than human eyes [53]. SSD deep learning is based on the famous VGG-16 architecture and includes additional convolutional layers to extract features at multiple scales [53]. Specifically, SSD can complete image classification and object localization in a single forward pass of the network. Another advantage of SSD is its real-time performance, "scoring over 74% mAP (mean Average Precision) at 59 frames per second on standard datasets such as PascalVOC and COCO." [54]

An SSD neural network was trained in this project to detect an owl logo pasted on the backside of the leader vehicle, as shown in Fig. 17.

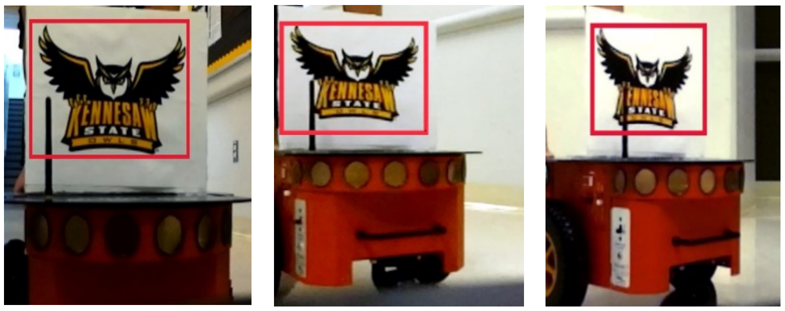


FIGURE 17

The Training Samples

To train the SSD deep neural network, 500 images of the owl logo were acquired from different orientations and distances. Then, the 500 images were input to the SSD network as the training samples. After 2,000 training iterations, the total loss of the network converges to 1.6, as shown in Fig. 18, which indicates a successful training result.

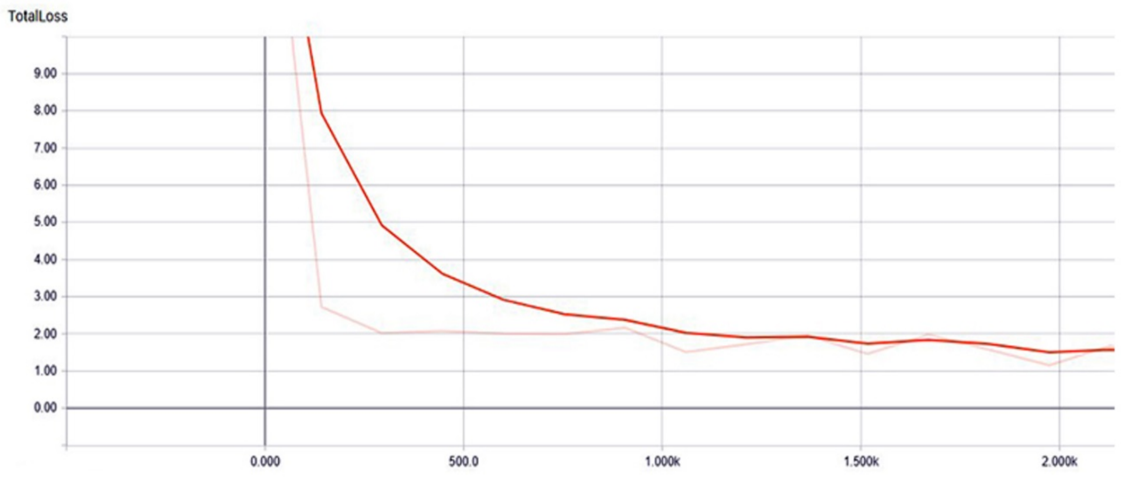


FIGURE 18

The Total Loss of the SSD Network After 2,000 Training Iterations

To improve the detection speed of the SSD technology mentioned above, an image filtering algorithm using depth information was proposed, as shown in Fig. 19.

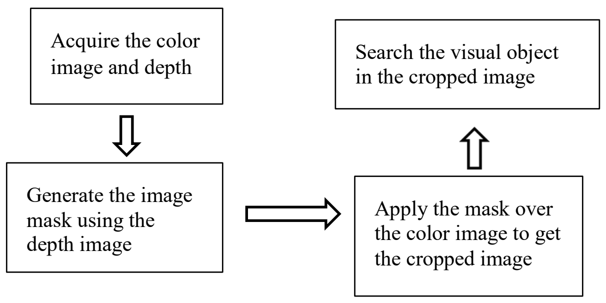

Acquire the color image and depth
Generate the image mask using the depth image
Apply the mask over the color image to get the cropped image
Search the visual object in the cropped image

FIGURE 19

The Image Filtering Algorithm Using Depth Information

In the above algorithm, the original color image and the depth image are first acquired through a ZED™ stereo camera. Then an image mask is generated using the depth image and the predefined depth range. In particular, if a pixel's depth is within the expected depth range, the value of "1" will be assigned in the corresponding position in the mask. Otherwise, "0" will be assigned in that position. Next, the image mask is applied over the original color image to generate the cropped image, which includes the objects within the expected depth range only. Finally, the SSD deep learning technology is employed to detect the object of interest on the cropped image. The image series in Fig. 20 shows the processing results using the image filtering algorithm proposed in Fig. 19.

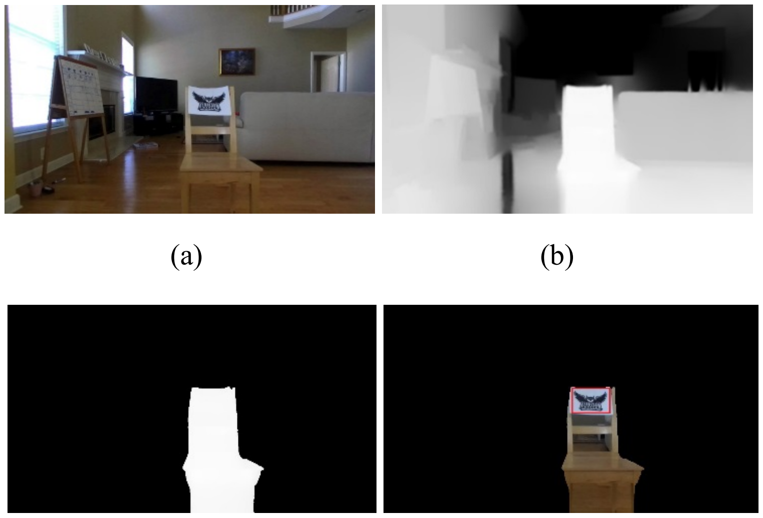

(a) (b) (c) (d)

FIGURE 20

The Image Processing Results Using the Proposed Image Filtering Algorithm:
(a) Original Color Image; (b) Depth Image;
(c) Image Mask; (d) Cropped Color Image

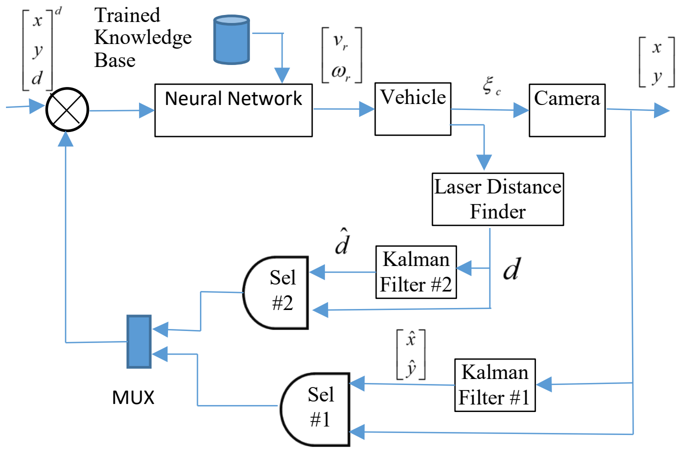
In the proposed image filtering algorithm, since the background and the objects not within the expected depth range are quickly filtered, SSD deep learning just needs to search the visual target (the owl logo here) in a small area of the cropped color image, which significantly improves the speed of the computer vision subsystem.

4.3 Develop the Communication Software for the Low-level Vehicle Control System

A local network (LAN) is set up in the follower vehicle through a wired router. The high-level control system (the laptop), the low-level control system (the RaspBerry Pi), and the laser scanner are connected in the network. A C++ TCP/IP program was developed in the laptop end, which can communicate with the Python program in the low-level control system. A high-level communication protocol was created to deliver the sensory information and control commands between the high-level and low-level control systems.

4.4 Develop Machine Learning Algorithms for Modular Tracking

To meet various challenges in the project, a model-free neural network controller with dual Kalman filters was proposed, as shown in Fig. 21.



$\begin{bmatrix} x \\ y \\ d \end{bmatrix}^d$
Trained Knowledge Base
Neural Network
$\begin{bmatrix} v_r \\ \omega_r \end{bmatrix}$
Vehicle
ξ_c
Camera
$\begin{bmatrix} x \\ y \end{bmatrix}$
Laser Distance Finder
$\hat{d}$
Kalman Filter #2
d
Sel #2
$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix}$
Kalman Filter #1
Sel #1
MUX

FIGURE 21

Leader–Follower Controller Using the Neural Network and Dual Kalman Filters for Autonomous Vehicle Tracking

In Fig. 21, the follower vehicle utilizes its onboard camera and laser distance finder to acquire the image of the leader vehicle that is moving in front of it and the distance d between them. Then the center position $\begin{bmatrix} x \\ y \end{bmatrix}$ of the leader vehicle in the image is extracted through the computer vision technology. A model-free neural network controller was proposed, which adjusts the translational and rotational velocities $\begin{bmatrix} v_r \\ \omega_r \end{bmatrix}$ of the follower vehicle based on the measured x, y, and d. When the follower vehicle is moving, since the camera is physically attached to the vehicle, the camera velocity ξ_c is changed accordingly and a different image of the leader vehicle is observed.

Meanwhile, the distance between the two vehicles is also changed due to the motions of the two vehicles. The neural network controller is trained offline so that it can adjust $\begin{bmatrix} v_r \\ \omega_r \end{bmatrix}$ continuously to keep a constant distance between the two vehicles and make $\begin{bmatrix} x \\ y \end{bmatrix}$ stay within a desired region on the image.

As mentioned, the slow response time of the computer vision subsystem and the laser sensor could cause the control system to fail totally. In this project, a dual-Kalman-filter approach was proposed to solve this problem, as shown in Fig. 21. There are two Kalman filters in the control loop, one for the computer vision subsystem and the other for the laser distance finder, which predict a new $\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix}$ or $\hat{d}$ based on their internal models. In addition to the filters, a selection unit (the "Sel" unit in Fig. 19) is designed to select a predicted measurement or a real measurement from the laser sensor or the computer vision subsystem. The selection strategy is as follows: If a new real measurement is available, the selection unit will select the new measurement as its output. Otherwise, the unit will select a measurement predicted by the Kalman filter as its output.

There are two advantages observed by introducing the two Kalman filters into the control loop. First, since the Kalman filters take less than 3 ms to predict a new measurement estimation, the controller can update the control command $\begin{bmatrix} v_r & \omega_r \end{bmatrix}^T$ within 10 ms, which is much faster than the control loop without the Kalman filters. Second, the experimental results in this project also show the dual-Kalman-filter approach improves the measurement reliability when the computer vision subsystem does not detect the leader vehicle in an image occasionally.

A model-free neural network controller was proposed to adjust the translational and rotational velocities of the follower vehicle. One of the advantages of a neural network controller is avoiding complicated modeling and subsequent linearization of the nonlinear vehicle dynamics, which is the main challenge in most model-based control approaches. The architecture of the neural network controller in this project is presented in Fig. 22.

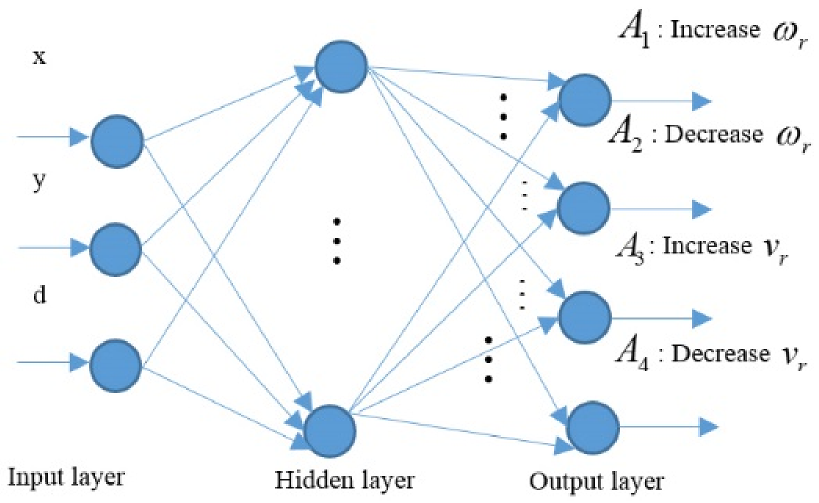


FIGURE 22

Architecture of the Neural Network Controller

In Fig. 22, the neural network controller accepts three inputs x, y, and d. In particular, x and y represent the center of the leader vehicle on the image plane, and d represents the current distance between the leader and follower vehicles. Then, the controller has four output units, and each of those represents a specific action ($A_1 \sim A_4$) of the follower vehicle: increase or decrease its translational or rotational velocities. The control strategy is learned from the samples through the backpropagation (BP) training algorithm [81], which are shown in Table 2. The objective of the controller is to keep the visual target on the rear of the leader vehicle close to the center of the image and maintain a constant distance between the two vehicles.

TABLE 2
The Training Samples of the Neural Network

Sample #	Inputs		Outputs			
	x	d	A_1	A_2	A_3	A_4
1	0	0	2	0	0	1
2	1	0	1	0	0	1
3	2	0	0	0	0	1
4	3	0	0	1	0	1
5	4	0	0	2	0	1
6	0	1	2	0	0	0
7	1	1	1	0	0	0
8	2	1	0	0	0	0
9	3	1	0	1	0	0
10	4	1	0	2	0	0
11	0	2	2	0	1	0
12	1	2	1	0	1	0
13	2	2	0	0	1	0
14	3	2	0	1	1	0
15	4	2	0	2	1	0

In Table 2, only x and d are considered because y almost does not change when the vehicles are running on a flat surface. The value of x is normalized to a number varying from 0 to 4, which represents the visual target position on the image plane from the far left side to the far right side. The normalized values of d vary from 0 to 2, which represent a "short," an "appropriate," or a "long distance" between two vehicles. The value of each output unit ($A_1 \sim A_4$) is telling how to adjust the follower vehicle's velocities. For example, $A_1 = 0$ indicates keeping the current ω_r, $A_1 = 1$ indicates increasing ω_r by a constant $\Delta\omega$, and $A_1 = 2$ indicates increasing ω_r by $2\Delta\omega$.

Before the neural network controller could run in a real vehicle tracking experiment, it was trained offline for 2,000 steps with 50 units in its hidden layer, using the samples in Table 2. The training error quickly converged to zero, as shown in Fig. 23.

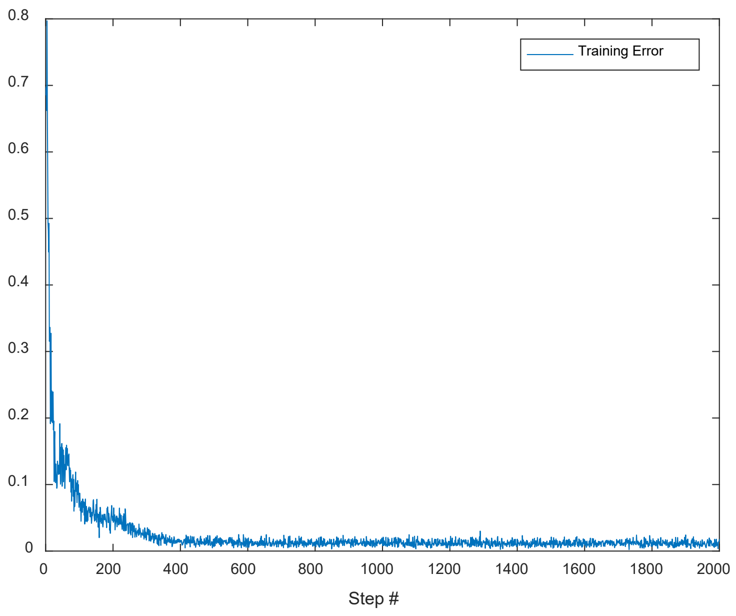


FIGURE 23

Training Error of the Neural Network in 2000 Steps Using 50 Hidden Units

Meanwhile, the histories of several selected weights of the network are shown in Figs. 24 and 25 where the weights converged quickly in the 2,000-step training, which indicates that the neural network learned the control strategy from the samples correctly.

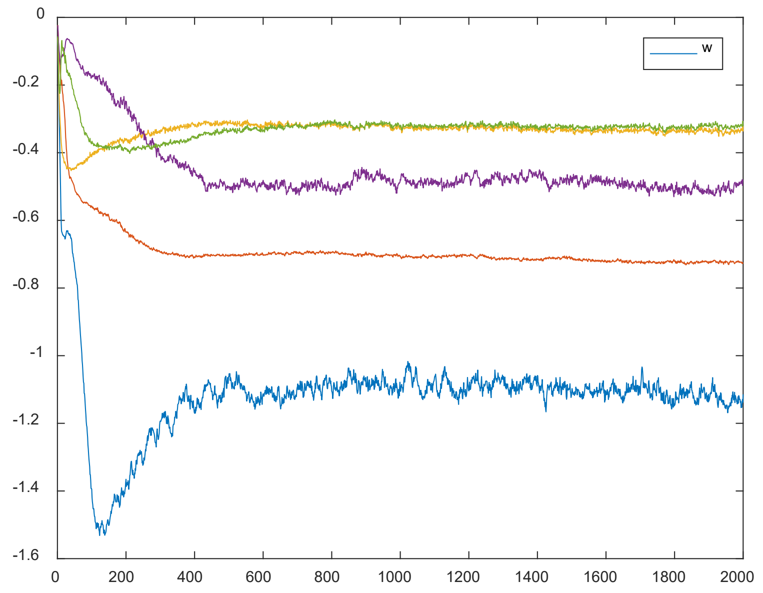


FIGURE 24

History of Selected Weights from the Input Layer to the Hidden Layer (w(1,10), w(2,20), w(1,26), w(2,35), w(1,17))

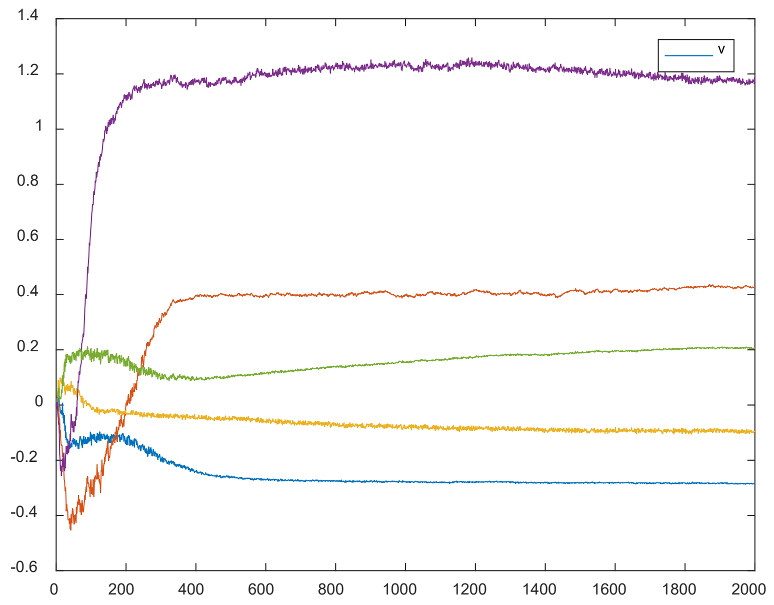


FIGURE 25

History of Selected Weights from the Hidden Layer to the Output Layer (v(2,3), v(30,4), v(18,1), v(10,2), v(36,3))

As mentioned previously, there are two Kalman filters designed for predicting the position of the leader vehicle on the image plane and the distance between the two vehicles. The model of the first Kalman filter for the computer vision subsystem is given in Equation 1.

$$\begin{cases} \mathbf{X}_k = \mathbf{F} \cdot \mathbf{X}_{k-1} + \mathbf{w}_k \\ \mathbf{Z}_k = \mathbf{H} \cdot \mathbf{X}_k + \mathbf{v}_k \end{cases} \tag{1}$$

where,

$\mathbf{X}_k = \begin{bmatrix} u_k & v_k & \dot{u}_k & \dot{v}_k & \ddot{u}_k & \ddot{v}_k \end{bmatrix}^T$ is the state variable;

(u_k, v_k) = the estimated position of the leader vehicle on the image plane at time k ;

$\mathbf{Z}_k$ = the measured position using the computer vision algorithm at time k ;

$\mathbf{F}$ = the state transition matrix;

$\mathbf{H}$ = the measurement matrix;

$\mathbf{w}_k$ = the process noise; and

$\mathbf{v}_k$ = the measurement noise at time k .

Both $\mathbf{w}_k$ and $\mathbf{v}_k$ are the zero mean Gaussian white noise, i.e. $\mathbf{w}_k \sim \mathrm{N}(0, \mathbf{Q}_k)$ and $\mathbf{v}_k \sim \mathrm{N}(0, \mathbf{R}_k)$, where $\mathbf{Q}_k$ and $\mathbf{R}_k$ are the covariance matrices.

In this project, a constant acceleration model was assumed. In other words, it was assumed that the position of the leader vehicle on the image plane is moving with a constant acceleration. As a result, the matrices $\mathbf{F}$ and $\mathbf{H}$ were derived as follows:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3}$$

It was also assumed that:

$$\mathbf{Q}_k = \begin{bmatrix} 10^{-4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 10^{-4} & 0 & 0 & 0 & 0 \\ 0 & 0 & 10^{-4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 10^{-4} & 0 & 0 \\ 0 & 0 & 0 & 0 & 10^{-4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 10^{-4} \end{bmatrix} \tag{4}$$

$$\mathbf{R}_k = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \tag{5}$$

Based on the above model, the Kalman filter continuously predicts the new positions of the leader vehicle on the image plane. Meanwhile, once a new real measurement is extracted from the image using the computer vision algorithm, it is employed to correct the filter. In the experiments in this project, it was observed that the Kalman filter usually takes less than 3 ms to predict a new position, while the computer vision algorithm takes more than 1,000 ms to return a new measurement. As a result, there are many predictions between the two adjacent measurements.

Compared to the Kalman filter used in the computer vision subsystem, the second Kalman filter for the laser sensor is much simpler, so its details are not presented here.

4.5 Field Test and Improvement

Several experiments with mobile robots were carried out to validate the proposed control strategy when the leader vehicle was moving along a straight-line, circular, or “S”-shaped trajectory. The experimental results for the circular trajectory are presented in Figs. 26 to 30. For example, the trajectories of the two vehicles are shown in Fig. 26 where the leader vehicle was moving along a circular trajectory with a radius of about 2.5 meters. As shown in the figure, the follower vehicle successfully tracked the leader vehicle.

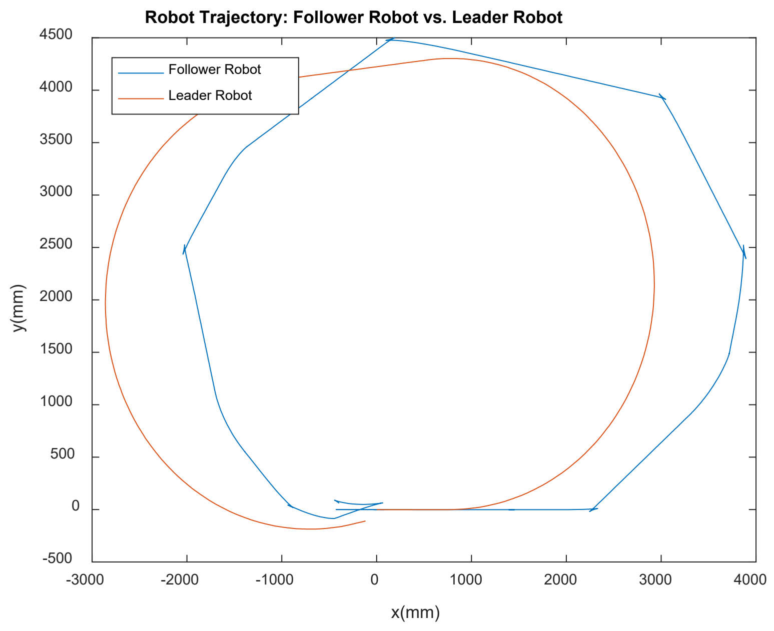


FIGURE 26

Trajectories of the Two Vehicles

Fig. 27 shows the *x*-coordinate of the visual sign on the image plane, which was extracted by the computer vision algorithm or predicted by the first Kalman filter. The computer vision subsystem takes about 1,050 ms to obtain a new measurement of the position of the visual sign.

Between every two adjacent measurements, the Kalman filter predicts a new position for the visual sign. Based on the results shown in Fig. 27, it is evident that the Kalman filter worked well.

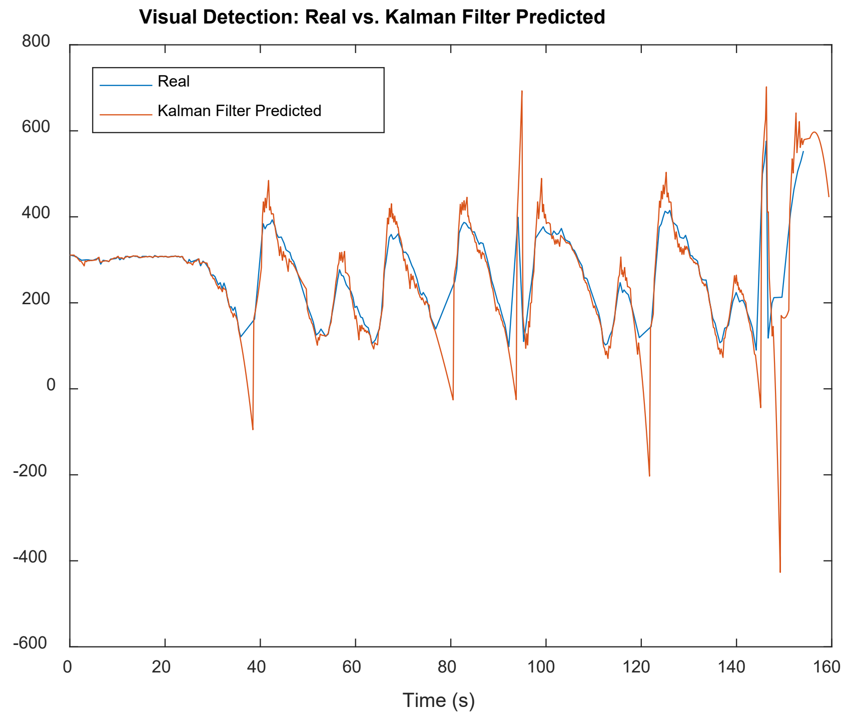


FIGURE 27

The x-coordinate of the Visual Sign on the Image Plane Extracted by the Cascade Classifier Algorithm and Predicted by the First Kalman Filter

Fig. 26 presents the distance between the two vehicles measured by the laser scanner and predicted by the second Kalman filter. It shows that the values predicted by the second Kalman match the real measurements of the laser scanner very well.

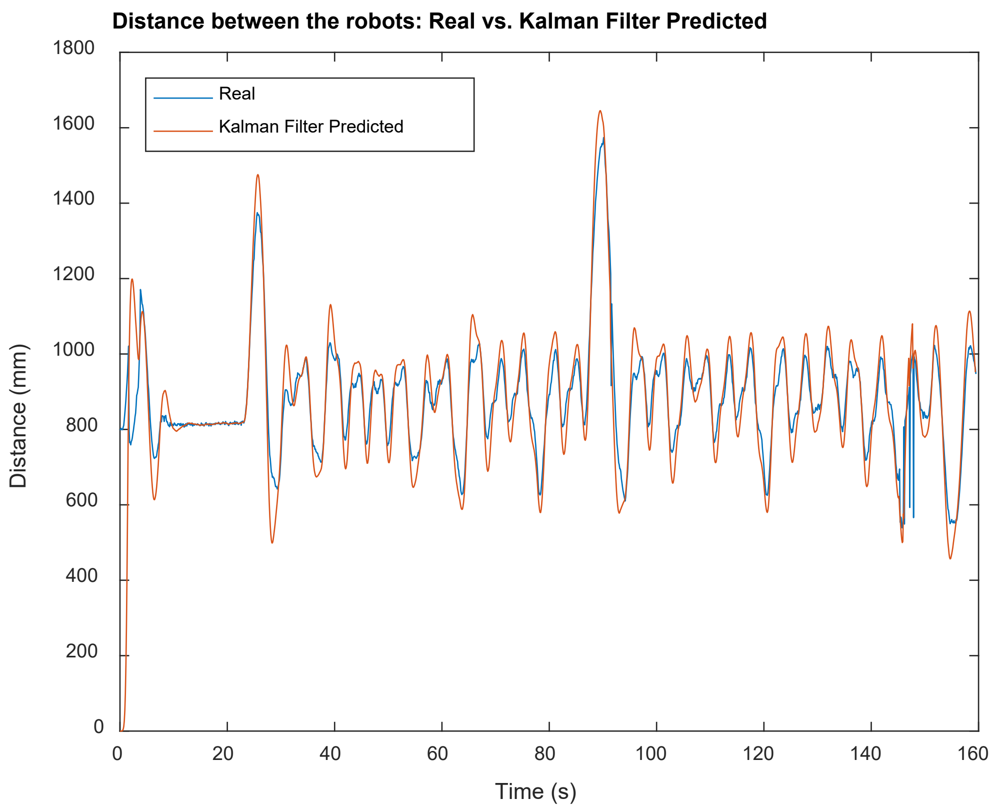


FIGURE 28

Distance between the Two Vehicles Measured by the Laser Scanner and Predicted by the Second Kalman Filter

The outputs of the neural network controller will adjust the translational and rotational velocities of the follower vehicle so that it can track the leader vehicle reliably. Therefore, it is interesting to observe how these velocities are adjusted when the leader vehicle moves along a circular trajectory. The history of the translational velocities of the two vehicles is presented in Fig. 27 while the history of the rotational velocities is shown in Fig. 28.

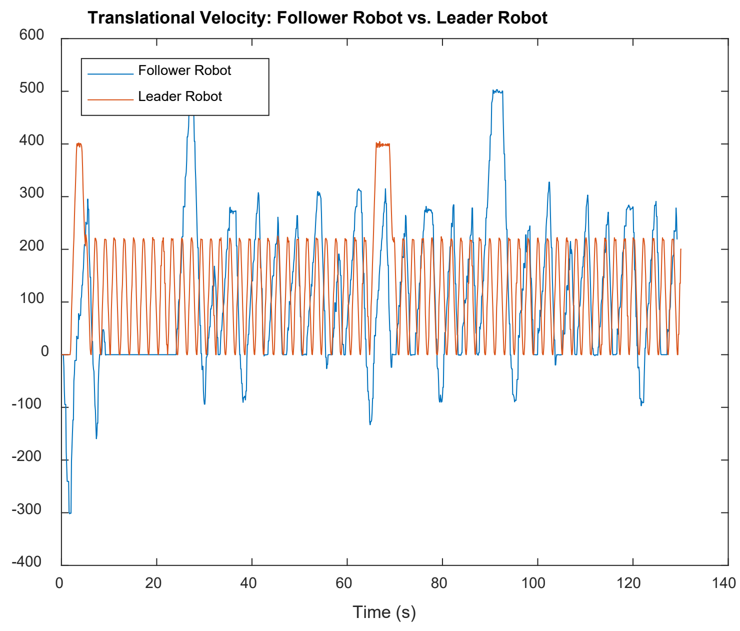


FIGURE 29

Translational Velocities of the Two Vehicles

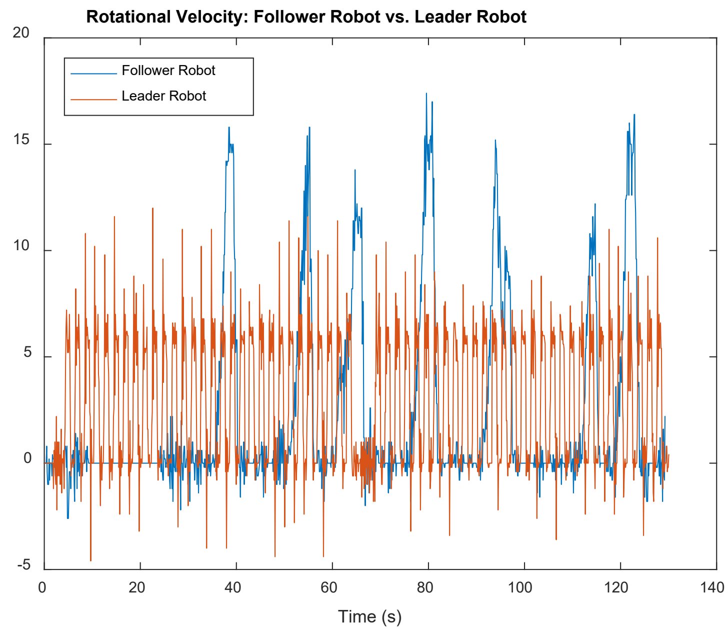


FIGURE 30

Rotational Velocities of the Two Vehicles

Based on these experimental results provided in Figs. 24 to 28, the proposed control strategy with dual Kalman filters works successfully and effectively in a two-vehicle mobile robot autonomous tracking situation.

5 TASK SET 1: Feasibility (Cost–Benefit) Study of the SMFe-BRT Concept

5.1 Motivation

Atlanta, Georgia, and its metropolitan area is one of the fastest growing regions in the nation. According to the Atlanta Regional Commission (ARC) *2014 Transportation Fact Book* [82], the region has been adding approximately 74,000 people each year since 1982. In 2011, the Atlanta metropolitan region became the ninth largest in the nation with a population size of 4.3 million. This number exceeded 5.8 million in 2017 according to the U.S. Census Bureau. Increasing population of the region and its demand for transportation infrastructure are causing severe congestion in Metro Atlanta. Total annual hours of delay have increased from 25,000 hours in 1982 to 150,000 hours in 2011. However, the supply of public transportation remains comparatively low and does not meet this increasing demand. The annual public transportation passenger miles traveled in Atlanta only range from 500 miles to 1,000 miles from the years of 1982 to 2011 [82]. As a result, the major state highways—Interstate 75, Interstate 85, and Interstate 285—suffer from severe congestion. As seen in Fig. 31, the level of service (LOS) by travel time index (TTI) indicates a LOS of F for most of the interstates in the northern part of Atlanta. Particularly, the region between Cobb County and Fulton County is one of the most congested areas in Metro Atlanta. This area experiences the highest travel demand, especially from daily commuters [41]. The only form of public transportation supplied in this area is Cobb Linc Route 10, which connects the Marietta Transfer Center with the MARTA Arts Station [83].

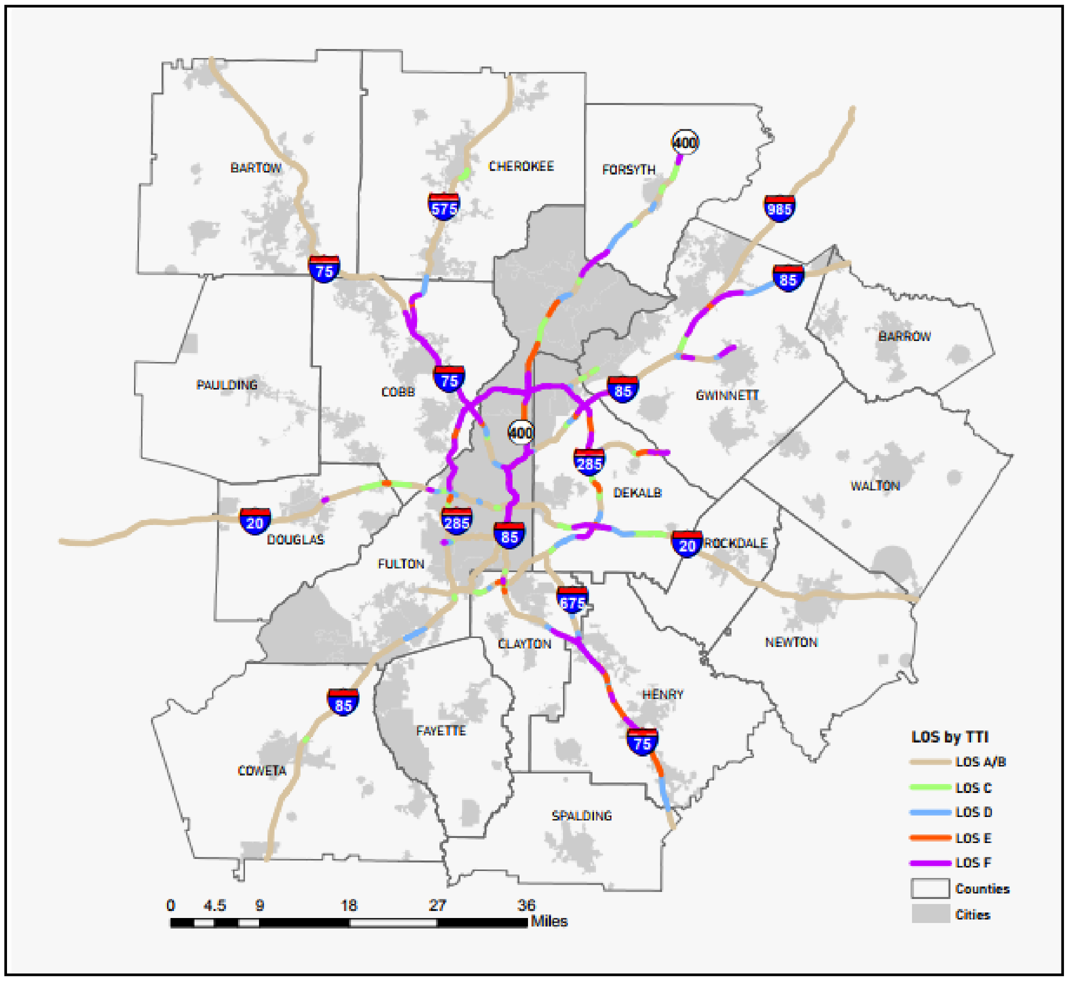


FIGURE 31

Most Congested Highway Segments, December 2014 [82]

5.2 Objectives and Overview

Bus rapid transit has gained popularity around the world in many metropolitan areas. BRT systems have been widely adopted because of their operational flexibility as well as lower operating costs [84]. Although BRT already has shown many benefits over traditional municipal buses, continued improvements to the original BRT have been made over time. This feasibility study is focused on a novel vehicle concept that the research team named Slim Modular Flexible Electric Bus Rapid Transit, which has the following key features:

- The system consists of one lead module and one or more follower modules
- Each module is 25% narrower in width compared to a traditional BRT vehicle
- The follower modules are self-propelled using in-wheel electric motors
- The modules are virtually coupled for easy detaching and attaching

Because of the narrower body of the modules, less right-of-way would be required for the dedicated bus lane, which translates to considerable savings in right-of-way and construction costs. The virtual coupling of modules permits flexibility in operations to better meet varying travel demand throughout a day. With each follower module being self-propelled by in-wheel electric motors, the SMFe-BRT provides an environment-friendly transit alternative.

To test the proposed vehicle concept and compare it with the traditional BRT, the researchers selected two of the busiest corridors in Georgia for this study: (1) Cobb Parkway, an arterial corridor, and (2) Georgia State Route 400 (GA 400), a freeway corridor. Both corridors have experienced severe congestion, especially during the peak periods of demand. The foremost reason for selecting those two locations is the fact that BRT systems are currently planned for those corridors.

5.2.1 Cobb Parkway Corridor

The Connect Cobb Corridor project was proposed in 2015 to improve the existing transit system [85]. The project runs from Kennesaw State University's Kennesaw Campus to the MARTA Arts Center station and consists of the following key features:

- Proposed bus rapid transit system
- Construction of dedicated bus lanes—center- and side-running dedicated guide lanes
- Usage of I-75 high-occupancy vehicle (HOV) lanes
- Total distance of 25.3 miles of the proposed BRT line
- BRT with a short headway of 8 minutes

- 15 proposed BRT stations

This study focuses on the section from Barret Lakes Boulevard to Akers Mill Road (indicated in purple in Fig. 32). This road section is approximately 12 miles long and includes a number of major signalized intersections (listed below), where transit signal priority will be assumed.

- Cobb Parkway + Barrett Lakes Boulevard/Greers Chapel Road
- Cobb Parkway + Progressive Way
- Cobb Parkway + Bells Ferry Road
- Cobb Parkway + Canton Road ramps (exit and entrance)
- Cobb Parkway + Allgood Road Northeast
- Cobb Parkway + North Marietta Parkway
- Cobb Parkway + Roswell Street Northeast
- Cobb Parkway + South Marietta Parkway
- Cobb Parkway + South Cobb Drive ramps (exit and entrance)
- Cobb Parkway + Terrell Mill Road Southeast
- Cobb Parkway + Windy Hill Road Southeast
- Cobb Parkway + Herodian Way
- Cobb Parkway + Cumberland Boulevard
- Cobb Parkway + Spring Road Southeast
- Cobb Parkway + Circle 75 Parkway
- Cobb Parkway + I-285 ramps (exit and entrance)
- Cobb Parkway + Akers Mill Road
- Cumberland Boulevard + Spring Road Southeast
- Cumberland Boulevard + Cumberland Parkway Southeast
- Cumberland Boulevard + Akers Mill Road
- Akers Mill Road + I-75 HOV Lane ramp

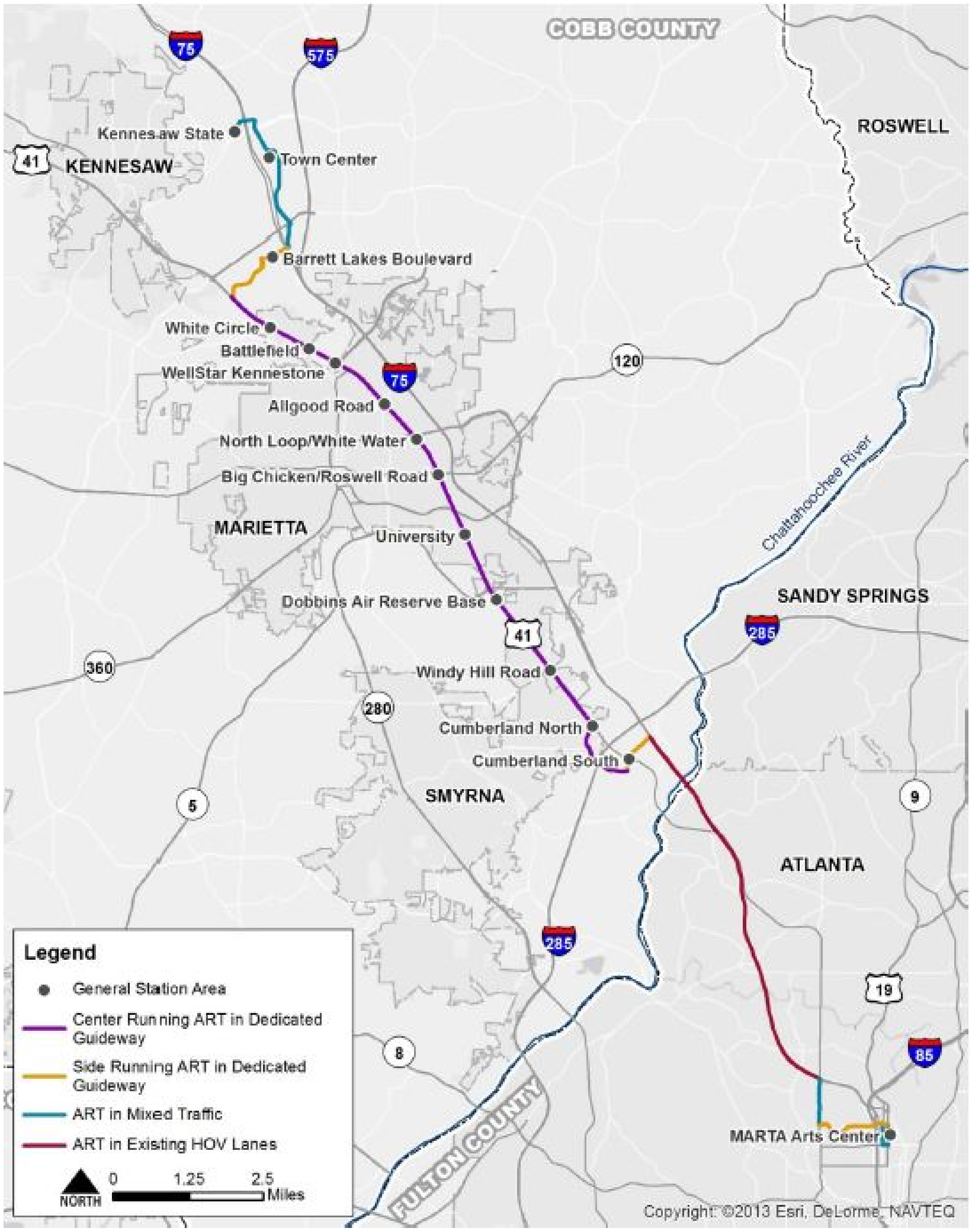


FIGURE 32

Proposed Connect Cobb Corridor Project, April 2015 [85]

5.2.2 GA 400 Corridor

As documented in the GA 400 Transit Initiative’s Environmental Impact Statement [86], there are four factors contributing to the need for the GA 400 Transit Initiative:

- Increased travel demand and resulting congestion generated by employment and population growth
- Limited existing transit mobility within northern Fulton County and inadequate connectivity to other major activity centers
- Transit travel times that are not competitive with automobile travel times
- Economic development opportunities being impacted by congestion

BRT will be added as part of the GA 400 Express Lane project. The project is approximately 12 miles long, running from the North Springs Station to Windward Parkway in northern Fulton County, as shown in Fig. 33.

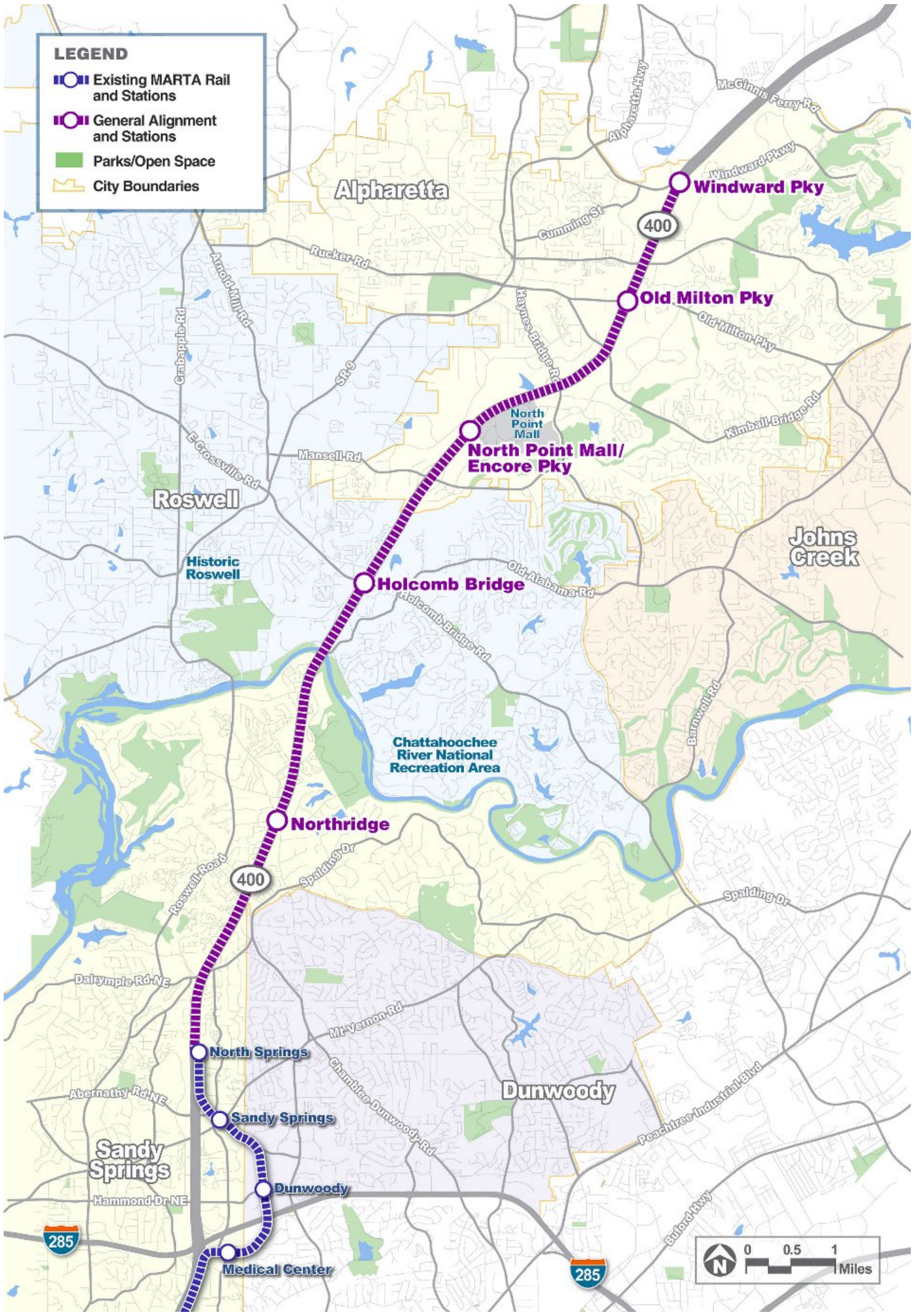


FIGURE 33

Proposed GA 400 Corridor Project, April 2015 [86]

5.3 Review of Literature

5.3.1 Introduction

The research team reviewed the literature pertaining to this study, including the following aspects:

- Assumptions for generating future conditions
- Commuting tendency in Metro Atlanta, and the studies on BRT impacts in other cities similar to Atlanta
- Environmental impacts and emission studies by vehicle types
- Methods for multicriteria evaluation of transportation projects, especially applications of the analytic hierarchy process (AHP) method

5.3.2 Future Ridership Prediction

5.3.2.1 Commuting Data in Atlanta [87]

Prior to making a prediction of future public transit ridership, it is essential to investigate the existing average percentage of demand for public transit. Residents in Metro Atlanta rely heavily on their personal vehicles. Despite existing transit services provided by MARTA's bus and rail lines, USDOT Bureau of Transportation statistics show 79.6 percent (compared to 76.3 percent nationwide) of Metro Atlanta commuters drive alone [2]. U.S. Census Bureau statistics show that the percentage of commuters who choose personal vehicles has been increasing, and those who choose public transportation has been decreasing since 1990. Data in 2014 indicate that the percentage of public transit choice in Metro Atlanta is only 10.6% (see Fig. 34).

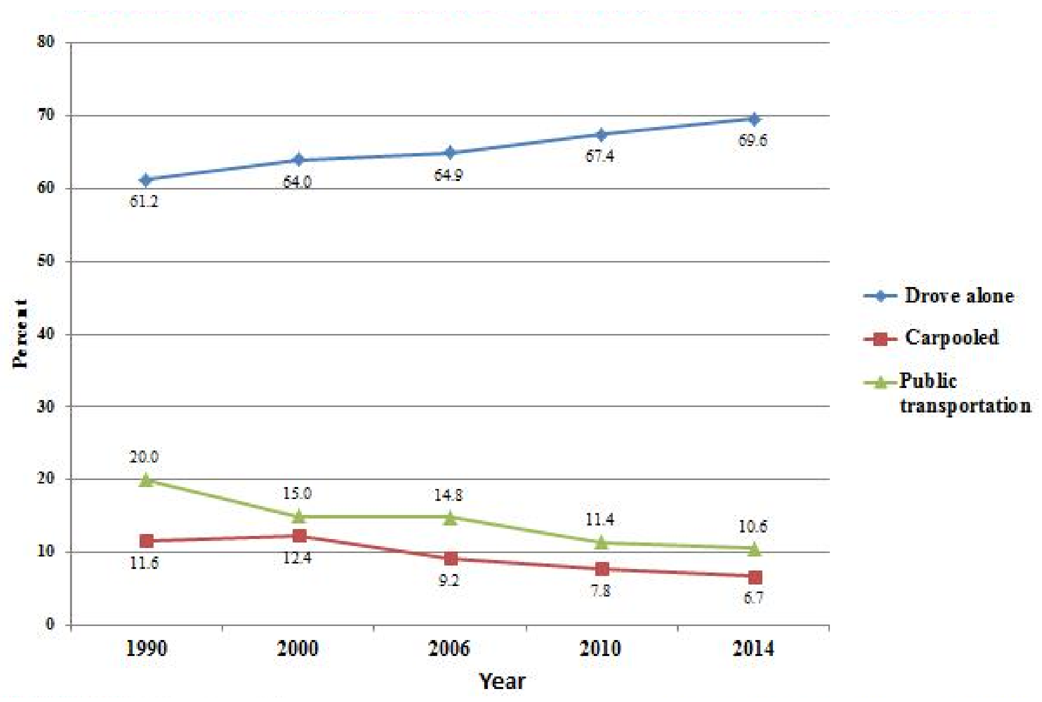


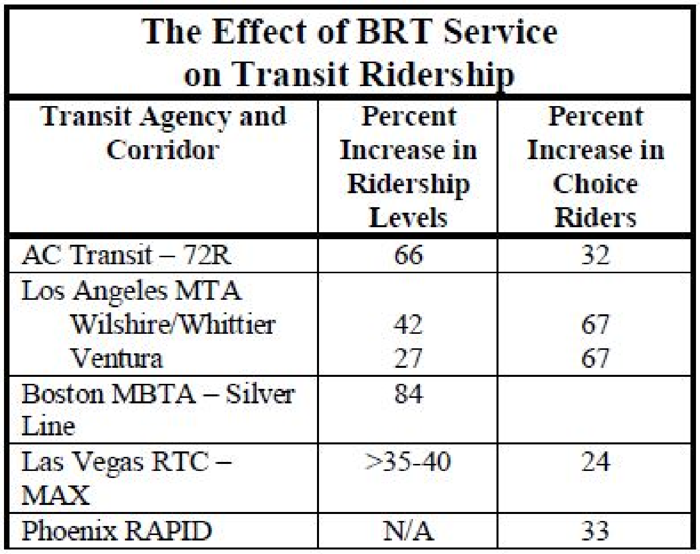
FIGURE 34

Commute Mode Share in Atlanta: 1990 to 2014 (Source: 1990 Census, 2000 Census, 2010 Census; and American Community Survey, 2006, 2010, 2014) [87]

5.3.2.2 BRT Ridership Study [88]

Due to the advanced technologies, improved designs, and features, BRT systems have gained popularity over the past decade. This popularity has promoted the public's demand for the BRT systems. To measure the changes in BRT ridership, Peak et al. collected data for 10 consecutive months in six BRT-operating cities. The results show that all six BRT-operating cities have experienced significant increases in ridership. In Las Vegas, the Metropolitan Area Express (MAX) system introduced by the Regional Transit Committee (RTC) is responsible for a 35 to 40 percent growth in ridership. Alameda–Contra Costa (AC) Transit achieved the highest (84 percent) increases in ridership in the governing districts. Table 3 summarizes the ridership increases in various cities where BRT systems have been adopted.

TABLE 3
The Effect of BRT Service on Transit Ridership



The Effect of BRT Service on Transit Ridership

Transit Agency and Corridor	Percent Increase in Ridership Levels	Percent Increase in Choice Riders
AC Transit – 72R	66	32
Los Angeles MTA		
Wilshire/Whittier	42	67
Ventura	27	67
Boston MBTA – Silver Line	84	
Las Vegas RTC – MAX	>35-40	24
Phoenix RAPID	N/A	33

It can be inferred that the introduction of BRT systems would typically induce an increase in ridership.

5.3.2.3 Ridership Responsiveness [89]

Traditionally, the *demand* for public transportation has been more responsive to its service than the monetary values, like bus fares. The data gathered from cities such as Detroit, Chesapeake/Norfolk, Madison, and Stevenage, and from Great Britain indicate that the mean transit headway elasticity is −0.47 with a standard deviation of ±0.14 for all service hours. For higher service demand during peak hours, the mean value is −0.27 with a standard deviation of ±0.14. The magnitude of elasticities indicates the relative change (e.g., percent change) in demand given a relative change (e.g., one percent change) in headway. The negative sign of the elasticity indicates that increasing headway will decrease demand. Since the magnitudes of the elasticity are less than 1.0, they imply that the demand is inelastic to headway changes in general.

5.3.2.4 Analysis of Vehicle and Person Throughput [90]

Given that personal vehicles are the major mode of commute in Metro Atlanta, the existing HOV lanes on Interstate 85 (I-85) were experiencing congestion. To further improve the serviceability of I-85, HOV lanes were converted into high-occupancy toll (HOT) lanes and opened to traffic in 2011. Guensler et al. conducted this study after the conversion took place to examine changes in vehicle occupancy, and vehicle and passenger throughput on I-85.

Since the implementation of tolls on the lanes, the average vehicle occupancy on the HOT lanes has decreased from around 2 persons per vehicle to that of the general-purpose lanes. Observed occupancies for the lanes on I-85 are shown in Table 4. Year 2012 is the latest observed year, and the average vehicle occupancy (AVO) for all lanes is in the range of 1.16–1.20 persons per vehicle.

TABLE 4
Observed Occupancy by Lane, Spring 2012, PM

Occupancy	HOT	GP1	GP2	GP3	GP4	GP5
1	85.3%	85.0%	86.0%	85.3%	83.7%	84.8%
2	12.2%	14.0%	12.9%	13.5%	14.6%	13.6%
3	0.8%	0.7%	0.6%	0.7%	1.2%	1.2%
4+	1.7%	0.4%	0.4%	0.6%	0.5%	0.5%
Sessions (n)	12	6	7	7	7	7
AVO	1.20	1.17	1.16	1.17	1.19	1.18

5.3.3 Emission Calculations

5.3.3.1 Public Transit Fuel Type [91]

Knowing the fuel type of a vehicle is the initial step for emission calculations. Most heavy and public transit vehicles historically have used diesel fuel, and diesel is still the main source of fuel for heavy vehicles being manufactured. However, with the rising concerns for the environment and air quality, alternative fuels have gained popularity. Compressed and liquefied natural gas, dual fuel engines, grid-connected, and hybrid electric are some alternative fuels that are available today.

Among the alternative fuels mentioned above, natural gas propulsion is the primary alternative for diesel public transit vehicles. In the U.S., approximately 7000 public transit vehicles are operated by natural gas. Between the two major types of natural gas propulsion system, compressed natural gas (CNG) is generally chosen over liquefied natural gas (LNG).

5.3.3.2 Automotive Emissions [92]

Depending on the purpose of a vehicle, the specific fuel type can be chosen as part of the vehicle design. To estimate emissions from in-use vehicles, Faiz et al. defined the main pollutants from automobile emissions. Harmful pollutants include carbon monoxide (CO), nitrogen oxides (NO_x), unburned hydrocarbons, volatile organic compounds (VOCs), and particulate matter (PM). Other pollutants measured include non-methane organic gases (NMOG). Since natural gas is mostly methane (CH_4), CNG vehicles were measured with much lower NMOG than gasoline or diesel vehicles; however, CNG vehicles produced higher emissions of methane. Vehicle emission pollutants mentioned above were measured for CNG transit buses and listed as follows:

- **CO, PM2.5, NO_x, THC** [93]

  The U.S. Environmental Protection Agency (EPA) measured CNG transit bus emission rates for CO, PM2.5, and NO_x. It found that vehicles' emissions are discharged at different rates, depending on the manufactured year groups and the vehicle age groups. For the most recent manufactured years from 2007 to 2013, and age groups of 0–3 years, the pollutants were measured as follows: 2.18 g/mile for NO_x, 5.93 g/mile for CO, and 0.0016 g/mile for PM2.5. Total hydrocarbon content (THC) for the same manufactured year and age group was 4.33 g/mile.

- **Nitrous Oxide (N_2O), Methane (CH_4)** [94]

  The EPA also measured N_2O and CH_4 emission rates for on-road CNG transit buses. The rates are listed as 1.97 g/mile for CH_4, and 0.175 g/mile for N_2O.

- **Carbon Dioxide (CO_2)** [95]

  In a study for the International Council on Clean Transportation, Delgado and Muncrief measured CO_2 emission rates for CNG buses as 2,250 g/mile.

- **Volatile Organic Compounds (VOCs)** [96]

  The EPA lists the ratio of VOC to THC as 0.004. With the THC rate of 4.33 g/mile provided above, the VOC rate can be calculated, using the given ratio, as 0.017 g/mile.

5.3.4 Analytic Hierarchy Process in Practice

Given the AHP as a widely used multi-criteria decision making (MCDM) framework, this subsection focuses on previous studies that used the AHP method for the evaluation of transportation projects.

5.3.4.1 The Case of Cracow, Poland [97]

In 2014, Nosal and Solecka applied the AHP method to evaluate the integrated system of urban public transport (ISUPT) in Cracow. ISUPT was considered there for mobility management purposes, and to encourage people's use of public transportation and bicycles and their choice of walking. The purpose of their study was to present the methodology of MCDM used and to apply it to assess ISUPT alternatives. They presented eight variants for ISUPT design in Cracow and created 10 evaluation criteria.

Criteria were chosen based on the survey of three interest groups: passengers, operators, and city authorities; and the 10 criteria are as follows: (1) travel time, (2) journey standard, (3) rolling stock use index, (4) environmental impact, (5) level of integration of public urban transport system, (6) reliability of urban public transport system, (7) safety of journeys, (8) profitability of urban public transport systems, (9) availability of urban public transport systems, and (10) investment costs. The importance of these criteria as weighted by the three interest groups are indicated in Fig. 35.

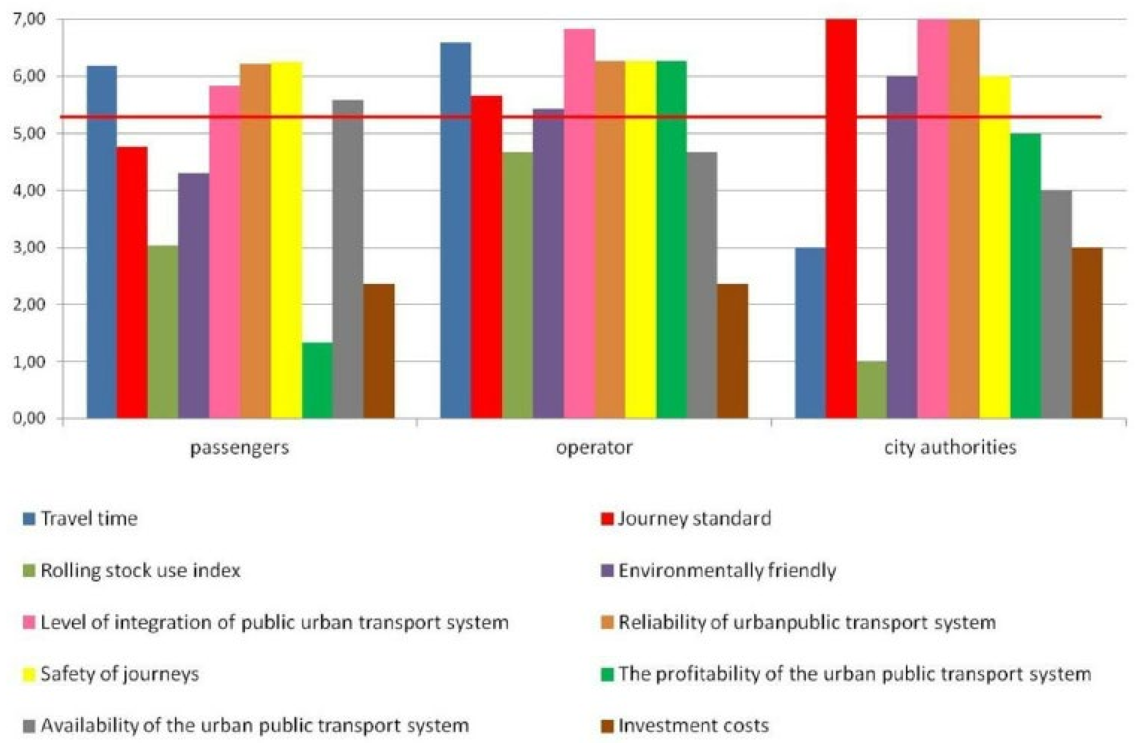


FIGURE 35

Definition of the Importance of the Criteria, Results of Surveys Conducted in Cracow[97]

5.3.4.2 The Case of Korea [98]

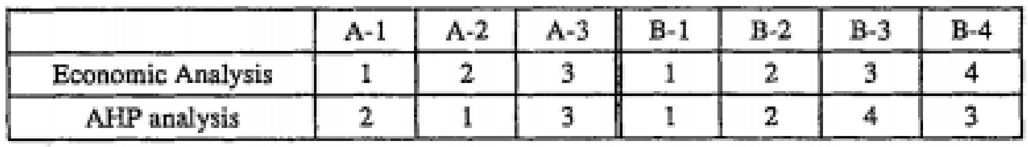
As the highway system affects the users greatly, Tabucanon and Lee emphasized improvements of the infrastructure. They used the AHP tool to select the alternative modes of highway route improvements in Korea, using the measures of effectiveness (MOEs). The AHP model was developed using the survey data and interviews of various interest groups' members. The highway users, the government, and the community members were chosen as the interest groups for the AHP model development. Among the three interest groups, the most important group was the highway users, with the weight assigned as 57%. The community members were the next important with a weight of 29%, and the government was the least important with a weight of 13%.

The significance of the community members in the literature was due to people's socio-economic status, and the diverse demand.

Each of the interest groups was given a different set of factors. Highway users were given travel time, travel cost, safety, congestion, and convenience, and they ranked the travel time factor the highest. Community members were given factors of regional equity, air pollution, noise impact, household displacement, and convenience. Because the highway system is part of the social production and distribution system, it promotes the accumulation of public and private capital and technology, and enhances market activity and income distribution and thus leads to improved living standards, regional specialization of the industry for the efficient use of national resources. It can also lead to balanced regional development and assists the achievement of regional equity and national unity [98]. The regional equity can be measured by travel cost/time to regional center and gross regional products of an area. As a result of the community survey, regional equity was valued the most by the community members.

The authors compared the preference order results. The AHP method was the main tool of evaluation in this literature; however, economic analysis, such as net present worth, benefit–cost ratio, and internal rate of return, was also conducted for comparison purposes. As shown in Table 5, the application of the AHP method yielded different priority results than the economic analysis across alternatives.

TABLE 5
The Preference Order Using Both Economic and AHP Analyses



	A-1	A-2	A-3	B-1	B-2	B-3	B-4
Economic Analysis	1	2	3	1	2	3	4
AHP analysis	2	1	3	1	2	4	3

5.3.4.3 The Case of Lithuania [99]

The rail system in Lithuania is not very attractive to its passengers because of the low speed, low level of comfort, and the low quality of railways and the dynamic characteristics of the locomotives. Sivilevicius and Maskeliunaite developed an AHP method to aid decision making for adopting a more effective rail system in Lithuania.

The evaluation criteria for this study were organized based on the railway trip quality and were divided into four main categories: (1) technical state of the track, (2) railway planning and technology, (3) price of the ticket, and (4) safety of the railroad.

5.3.4.4 The Case of Singapore [100]

The research problem lies with rapid economic development in Singapore. Singapore's desire to build a world-class transportation system encouraged the researchers to investigate alternative fuel use for the years of 2020–2030. The goal was to displace oil as a source of fuel. A multiple criteria decision method was used to identify 10 different fuel options, and then filter down to 4 main fuel alternatives. The AHP method was used for selecting the best fuel system, and the alternatives were evaluated based on economic, technical, and social considerations. Sensitivity analysis was also performed, which provides stability for the most optimum solution, especially when the parameters are sensitive to change. The AHP method was also used for forward and backward planning to determine the likely future conditions as well as the necessary policies.

The 10 preliminary alternative fuels are provided below. The alternatives were screened using the multiple criteria decision method, and narrowed down to four: (1) status quo, (2) oil and electric vehicles (EV), (3) oil and natural gas vehicles (NGV), and (4) methanol-fueled vehicles. Status quo refers to no change in transportation fuel. These four alternatives were evaluated using different criteria: consumer preference, safety, cost, supply, technology, and emission.

In addition, the policy that aids the selected alternatives was chosen from this list:

- Policy P1: to provide financial incentives to promote the use of electric vehicles
- Policy P2: to adopt stricter emissions standards for motor vehicles
- Policy P3: to provide the infrastructure to facilitate recharging of electric vehicles

- Policy P4: to lengthen the life of the certificate of entitlement for electric vehicles compared with oil vehicles

It was found, after performing three steps of forward and backward processes, that the use of electric vehicles would be the best option, along with the combination of Policies P1 and P3.

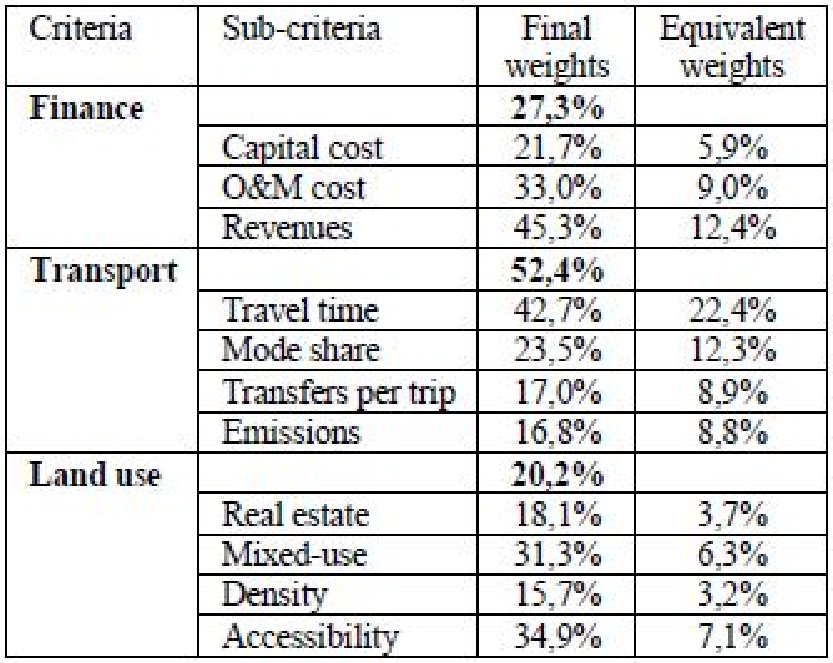
5.3.5 Application of Analytic Hierarchy Process

5.3.5.1 Weight Assignments by Experts [101]

Multicriteria decision analysis is typically used for selecting the optimal option for transportation projects. Although cost-benefit analysis (CBA) is a popular decision-supporting tool, it becomes inappropriate or less effective when the alternatives have values that are hard to monetize. In their study, Schlickmann et al. used the AHP method for evaluation of transit alternatives. The alternatives were defined as no build, BRT, and light rail transit (LRT). They selected three main criteria: finance, transport, and land use.

One of the crucial steps in performing the AHP method is to consider experts' opinions in assigning criteria weights. The study shows the recent survey data answered by 19 experts from different countries—Brazil, Canada, Germany, the Netherlands, Portugal, and the U.S. The survey data are shown in Table 6. According to the assigned weights, the biggest influencers for choosing an alternative were travel time, revenues, mode share, and operation and maintenance (O&M) costs, since they add up to 60% of the total weight.

TABLE 6
Priority Profile



Criteria	Sub-criteria	Final weights	Equivalent weights
Finance		**27,3%**	
	Capital cost	21,7%	5,9%
	O&M cost	33,0%	9,0%
	Revenues	45,3%	12,4%
Transport		**52,4%**	
	Travel time	42,7%	22,4%
	Mode share	23,5%	12,3%
	Transfers per trip	17,0%	8,9%
	Emissions	16,8%	8,8%
Land use		**20,2%**	
	Real estate	18,1%	3,7%
	Mixed-use	31,3%	6,3%
	Density	15,7%	3,2%
	Accessibility	34,9%	7,1%

5.4 Methodology

5.4.1 Microsimulation – Vissim Model Parameters

The microsimulation software, PTV Vissim, was selected as the simulation tool. This subsection discusses the Vissim network inputs that had constant values across the alternatives evaluated.

5.4.1.1 Desired Speed Decisions

Desired speeds were assumed to follow respective speed limits, i.e., 45 mph for Cobb Parkway; 35 mph for intersecting roads with Cobb Parkway; and 65 mph for GA 400.

5.4.1.2 Driving Behaviors

The drivers' behaviors were assumed to follow urban (motorized) driving behaviors. The Wiedemann 74 car-following model was used, for which the desired distance between each car is calculated by Equation 6 [102]:

$$d = ax + bx, \tag{6}$$

where

ax = standstill distance;

bx = additional space between two successive vehicles if they are moving, and

$$bx = (bx_{\text{additive}} + bx_{\text{multiplicative}} \times z) \times \sqrt{v};$$

v = vehicle speed (m/s); and

z = value of range (0.1), which is normally distributed around 0.5 with a standard deviation of 0.15.

5.4.1.3 Vehicle Compositions

According to the traffic count data gathered, the percentages of vehicles were estimated to be 3% trucks, 2% buses, and 95% personal vehicles.

5.4.1.4 Signal Controllers

Ring barrier signal controllers are the typical type of signal controller used in all Vissim models. For illustration, a typical ring barrier diagram is shown in Fig. 36.

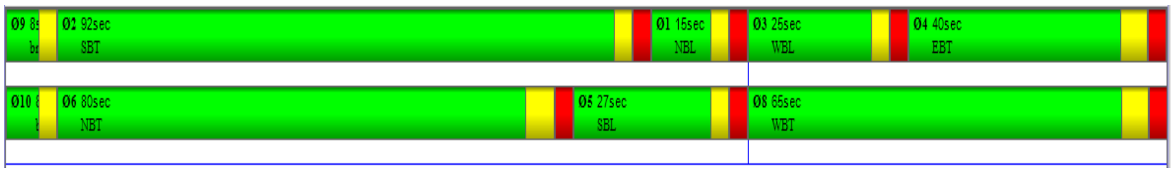


FIGURE 36

Typical Ring Barrier Diagram

5.4.1.5 Simulation Characteristics

After the Vissim models were established, the simulations were executed five times to generate network performance data. The purpose of multiple simulation runs is to capture the variance in the results. Changing random seed number within Vissim for each simulation prevents each simulation from having the same outcome.

The total length of the Vissim simulation was set for 5,400 simulation seconds (5,400 real-time seconds). The seeding period was set to be 1,800 simulation seconds. During the seeding period, a unique traffic composition was planted into the Vissim network. The actual recording periods for reporting were 3,600 simulation seconds. The simulation results were recorded and compiled for the AM peak, PM peak, and off-peak periods each under the following conditions:

- Future condition with BRT, 27% ridership increase
- Future condition with SMFe-BRT, 27% ridership increase
- Future condition with SMFe-BRT, 32% ridership increase

Note that the 27% ridership increase for the traditional BRT alternative was determined based on the literature review. Given the advantageous features of SMFe-BRT over the traditional BRT, a 32% ridership increase with SMFe-BRT is expected and was, thus, evaluated. However, 27% ridership increase for SMFe-BRT was also studied for comparison purposes.

To account for the flexibility of SMFe-BRT in meeting varying demand, the suggested numbers of modules for different operating periods or different headways are shown in Table 7. Note that for Cobb Parkway, a constant headway of 8 minutes was used for all peak and off-peak periods, and three modules were assigned for peak periods and two modules were assigned for the off-peak period. For GA 400, the numbers of modules are suggested values for different headways; the actual assignment in the simulation may be different depending on demand. The maximum of four modules allows for all boarding passengers being picked up at the busiest station.

TABLE 7
Number of Modules for SMFe Operations

Cobb Parkway

Ridership Increase	Time	Number of modules	Carrying Capacity
27%	AM peak	3	105
	PM peak	3	105
	Off peak	2	70
32%	AM peak	3	105
	PM peak	3	105
	Off peak	2	70

GA-400

Headway	Number of modules	Carrying Capacity
5 minutes	2	70
10 minutes	3	105
15 minutes	3	105
20 minutes	4	140

5.4.2 Vissim Model Development: Existing Conditions

5.4.2.1 Vehicle Volume Inputs

As vehicle input data were obtained from different agencies along Cobb Parkway, the data were compiled and organized in Excel® spreadsheets. The year 2019 is referenced as the base year, and Year 2040 is referenced as the future year in this study. Given the Atlanta Braves' stadium as a development of regional impact (DRI) in the region, traffic generated by this DRI is considered in deriving future traffic volumes.

5.4.2.2 Existing Public Transit: CobbLinc

The existing CobbLinc ridership data were obtained during the marked data collection duration. The typical duration of the data collection period is about 3 months. Since Vissim inputs are hourly based, the average ridership for AM peak, PM peak, and mid-day peak conditions were derived and used.

5.4.3 Vissim Model Development: Future Conditions

5.4.3.1 Dedicated Bus Lanes

A key feature of the traditional BRT and SMFe-BRT is the dedicated bus lanes. For Cobb Parkway, a center-running dedicated guideway will be added from the northernmost part to the southernmost part of Cobb Parkway. From the center-running dedicated lanes, the express lanes will be transformed into the side-running dedicated guideways [83]; see Figs. 37 and 38. In Vissim, at center-running locations, the existing lanes will be shifted to the side and the proposed dedicated guideways will be added in the middle.

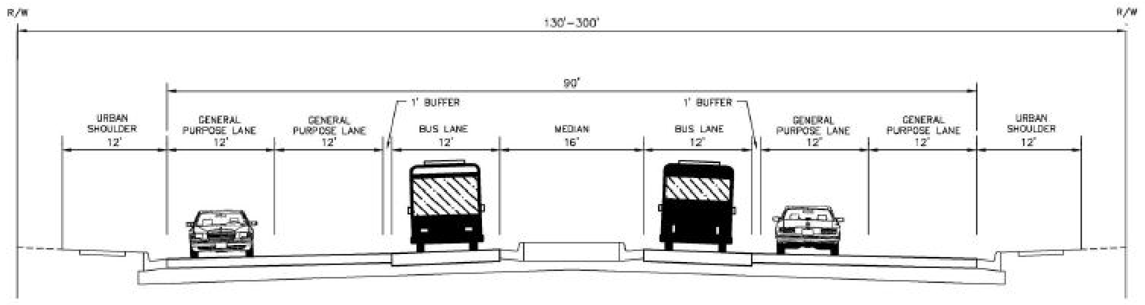


FIGURE 37

Proposed Typical Section on Cobb Parkway – Center-Running Dedicated Guideway

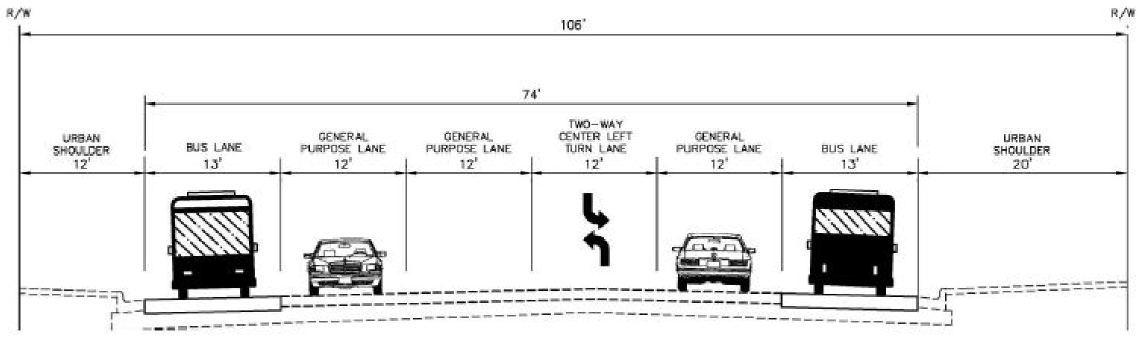


FIGURE 38

Proposed Typical Section on Akers Mill Road – Side-Running Dedicated Guideway

For the GA 400 corridor, three transit alternatives were proposed: one heavy rail transit (HRT) alternative and two BRT alternatives. The HRT alternative operates in an exclusive guideway on either side of the GA 400 right-of-way. For the two BRT alternatives, one uses the same alignment as the HRT alternative and the other operates within the planned Georgia Department of Transportation (GDOT) managed lanes along GA 400.

5.4.3.2 Vehicle Volumes Prediction

Future conditions represent the year 2040. To properly account for background traffic growth over time, vehicle volumes were factored up by an annual traffic growth rate using Equation 7 [103].

$$E_{t+n} = E_t \times (1+g)^n \tag{7}$$

where

E_{t+n} = Annual Average Daily Traffic (AADT) value of t year, forecasted n years in the future;E_t = AADT observed in base year t; and

g = AADT annual growth rate.

Given the Atlanta Braves' stadium as a DRI, its traffic impacts were considered as well.

An increase in public transit ridership is well expected with the introduction of the BRT system. This would result in a decrease in network vehicle volumes. Considering the assumptions made in ridership, the decrease in network traffic volumes was estimated and reflected in the future traffic volumes. The future traffic volumes were derived by following the steps below.

1. From the existing traffic volumes, the year 2040 background traffic volumes were estimated by applying a growth rate (Equation 7).
2. The project volumes from the Atlanta Braves Development of Regional Impact traffic study [104] were obtained and adjusted to reflect 2040 project conditions.
3. The traffic volumes from (1) and (2) above were added to obtain total 2040 traffic volumes.
4. A traffic volume reduction by introducing the new public transit alternatives (described in the next subsection) was applied to obtain effective future (2040) network traffic volumes.

5.4.3.3 Ridership Forecast

The introduction of either traditional BRT or SMFe-BRT is expected to increase the existing public transportation ridership, which is 10.6% for commuters [87]. From the cities that have already adopted BRT, a significant increase in ridership has been observed. Based on the *BRT Ridership Analysis* [88], the ridership increase ranges from 27% to 84%. To be realistic and conservative, the lower end of the ridership growth, 27%, was assumed for traditional BRT in this study. Given the more advantageous features of the proposed SMFe-BRTs, an even higher ridership is anticipated. The additional increase in ridership for SMFe-BRTs is estimated by referencing the Ridership Responsiveness subsection in the Review of Literature in Section 5.3. Because the ridership response and the change in wait time are interdependent, an assumption was made that SMFe-BRT would reduce the passenger wait time by 5 minutes compared to the traditional BRT, due to its operational flexibility. As such, the wait times for BRT and SMFe-BRT are estimated to be 30 minutes and 25 minutes, respectively. Based on the variability of wait time on ridership at peak-hours conditions, which was estimated to be −0.27 [89], the 17% reduction in wait time would yield approximately a 5% reduction in ridership. In other words, the SMFe-BRT is expected to have 5% additional ridership over the traditional BRT. The future ridership and reduction in network vehicles are related by Equation 8.

$$Future\ Ridership = Vehicles\ Reduced\ \times Average\ Vehicle\ Occupancy \tag{8}$$

Given that the average vehicle occupancy in Metro Atlanta is very close to 1, it was assumed that future ridership is equal to vehicles removed [90]. By following Step 3 in the Vehicle Volumes Prediction subsection, the network vehicles reduced due to the ridership increase can be estimated by Equation 9.

$$\begin{aligned} Vehicles\ Reduced_{27\%} &= 0.106\, V_{(3)} + (0.106)(1 + 0.27)V_{(3)} \\ &= (0.106)(2.27)V_{(3)} \end{aligned} \tag{9}$$

where

$V_{(3)}$ = the Step 3 results from the Vehicle Volumes Prediction section.

Equation 9 was used for the traditional and SMFe-BRT's 27% increase in ridership. Likewise, with SMFe-BRT's 32% increase in ridership, the network vehicles reduced can be estimated by Equation 10.

$$Vehicles\ Reduced_{32\%} = (0.106)(2.32)V_{(3)} \tag{10}$$

5.4.3.4 Traffic Signal Timing

For Cobb Parkway, adjustments were made on future signal timing as additional bus-dedicated guideways are added. Transit priority was coded in the simulation by using check-in and check-out detectors at signalized intersections where the center dedicated BRT lanes are proposed. When the check-in detector detects the BRT vehicle, the phase for the BRT is activated if the current phase is red or is extended if the current phase is green. For the locations where the signalized intersection is located just ahead of a proposed station, the standard presence detectors are used instead. The typical method of reflecting the BRT signal priority in the existing signal timing involves using additional signal phases (e.g., 9 and 10) for the BRT or SMFe-BRT vehicles with check-in and check-out detectors for the dedicated lanes and adjusting signal timing for other phases accordingly. An example at the Cobb Parkway and South Marietta Parkway intersection is provided in Fig. 37. The assigned movements for each signal group (SG) are as follows: SG 1: Northbound left; SG 2: Southbound through; SG 3: Westbound left; SG 4: Eastbound through; SG 5: Southbound left; SG 6: Northbound through; SG 8: Westbound through; SG 9: Dedicated lane Southbound; SG 10: Dedicated lane Northbound. Additional signal groups 12 and 16 are assigned as overlap signals of SG 2 and SG 6. The primary purpose of the overlapped signals is to allow green time for dedicated lanes when the southbound and northbound phases, SG 2 and SG 6, are green. SG 302 and SG 306 are assigned to the dedicated lanes for transit priority. Check-in detectors 312 and 316 are assigned to SG 302 and 306 each, and so as checkout detectors 322 and 326. Fig. 39 captures the moment when a public transit vehicle enters the intersection. As shown on the signal timing table, the phases are shortened once the vehicle checks in with detector 316, and the green phase is given to SG 10 (SG 306).

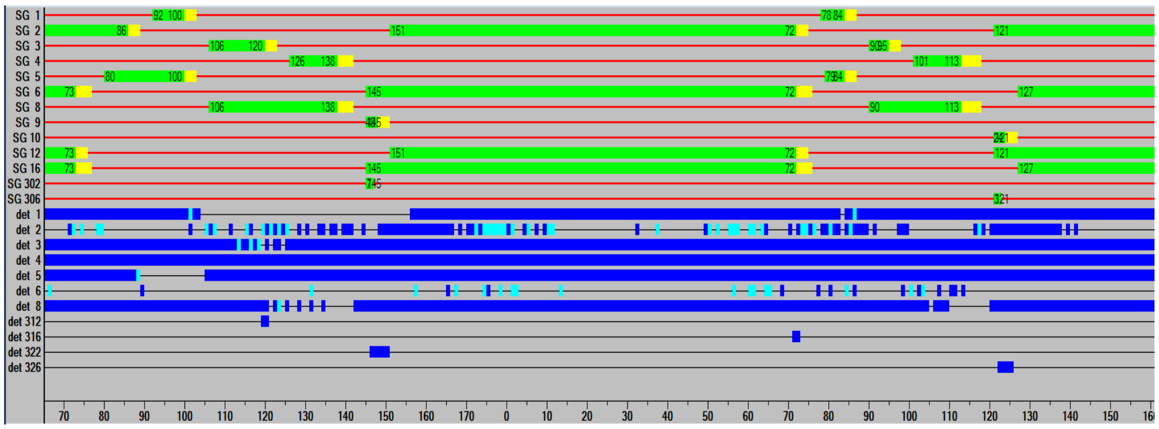


FIGURE 39

Ring Barrier Signal Timing at the Intersection of Cobb Pkwy and S. Marietta Pkwy

5.4.3.5 Emission Estimation

It is essential to know the fuel type of the BRT. As part of MARTA's goal for sustainability, Compressed natural gas (CNG) buses will be introduced [105]. Following the recent trend, it was assumed that the future BRTs will be run on CNG fuel.

The emission calculation based on CNG was computed using Equation 11. The equation is formed based on the emission factors of each pollutant, multiplying the distance. The distance is converted from the speed the vehicle is traveling at the time step. These data are collected from Vissim every 10 seconds.

$$Emission_{pollutant}\ (g) = EF \times v \times \frac{1\ (hr)}{3600\ (sec)} \times 10\ (sec) \qquad (11)$$

where,

EF = emission factor by pollutant (g/mile); and

v = the speed of the vehicle at the time step (mph).

Note that since SMFe-BRT modules will be fully electric, there will be no emissions from the SMFe vehicle.

5.4.4 Multicriteria Evaluation

5.4.4.1 Weights for Decision Criteria

Similar groups of criteria from a previous study [101] were adopted in this study. Some weights for subcriteria were reallocated or adjusted accordingly. The resulting priority profile is shown in Table 8.

TABLE 8
AHP Priority Profile

Criteria	Sub-Criteria	Final Weights	Equivalent Weights
Transport		**0.46**	
	Delay*	0.72	0.33
	Wait time	0.28	0.13
Environment (Emissions)		**0.13**	
	CO	0.14	0.02
	CO_2	0.14	0.02
	PM2.5	0.14	0.02
	NO_X	0.14	0.02
	CH_4	0.14	0.02
	N_2O	0.14	0.02
	VOC	0.14	0.02
Finance		**0.41**	
	Capital Cost	0.22	0.09
	O&M	0.33	0.13
	Revenue	0.45	0.18

* Note for GA-400 Corridor, the delay criteria is removed.
Wait time takes the whole weight of 0.46.

5.4.4.2 Analytic Hierarchy Model

The hierarchical structure of the AHP model developed in this study is shown in Fig. 40.

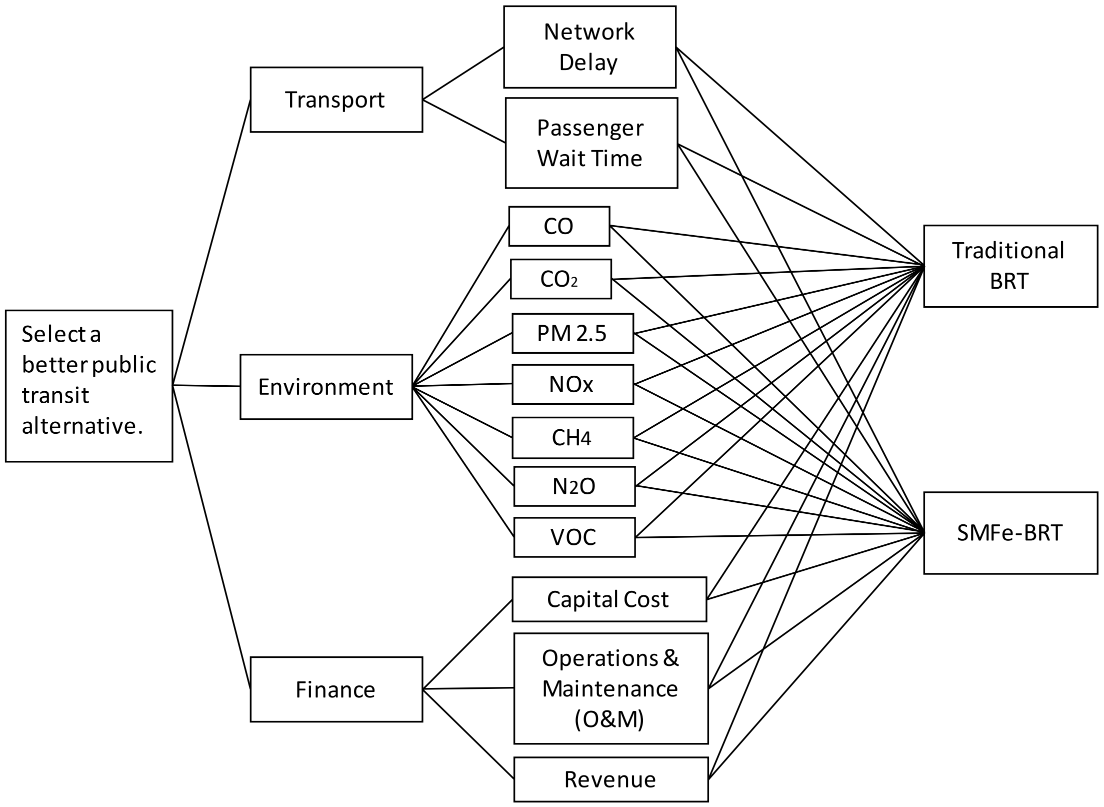


FIGURE 40

Hierarchical Structure of the Model

Note that the performance measures for the first two criteria, i.e., Transport and Environment, were derived from the network simulation runs. For assessing the finance of projects, regional cost data and revenues from the estimated ridership were used, which are discussed in the following subsection.

5.4.5 Finances

5.4.5.1 Capital Cost Estimate [106]

The “Connect Cobb: Northwest Transit Corridor Alternatives Analysis” includes the detailed costs per the design alternatives. While categories such as stations, support facilities, and site work capital costs will likely remain the same for BRT and SMFe alternatives, the guideway is expected to be different. Since the SMFe-BRT will be 25% slimmer in body width than the regular BRT, the construction and material costs for the guideway and track elements would be lower than the traditional BRT system. The unit cost of grade-exclusive right-of-way was $992 per linear foot in 2012. This unit cost was factored up 1.09, based on the historical consumer price indices, to obtain dollar value in the year 2019, which is $1,081.28 per linear foot [107]. The total costs for the traditional BRT were calculated based on the estimated length of the proposed guideway, 11.9 miles for Cobb Parkway and 12 miles for GA 400. Because of the slimmer body of SMFe-BRT, a 25% reduction was applied for the material used for constructing the dedicated lane.

5.4.5.2 Operation and Maintenance Cost

According to the “Georgia Department of Transportation Fact Book,” a total of $14,561,221 was spent on asphalt and concrete roadway pavement maintenance and repair in 2012 [108]. That breaks down to a unit cost of $303.36 per lane mile, provided the Department conducted the annual maintenance of 48,000 lane miles in Georgia. The O&M unit cost also was adjusted to reflect dollar value in 2019, resulting in $330.66 per lane mile [108].

For BRT O&M costs, the research team used data from Seattle, Washington, where BRT had been adopted and operated. According to the Madison Area Transportation Planning Board, vehicle operations and related costs are $75.61 per BRT annual revenue bus hours; vehicle maintenance–related costs are $1.39 per BRT revenue bus miles; articulated bus premiums are $0.35 per BRT revenue bus miles; ticket vending machine maintenance costs are $6,500 per machine unit; and station and stop maintenance costs are $2,000 per directional bus stops [109].

Annual bus hours and bus miles were calculated using the simulated travel times and distances for the BRT and SMFe-BRT lines. For daily bus revenue estimation, only simulated hours were considered. For the Cobb Parkway corridor, the proposed bus stations are located in the center of the roadway. For the O&M calculation purposes, it was assumed that the center-located bus stations are shared for both directions. The number of ticket machines was assumed to be the same as the number of bus stations.

5.4.5.3 Revenues

Annual revenues for the proposed systems were calculated using the simulated results. The total boarding passengers for AM, PM, and off-peak conditions were extracted from the simulation runs, and the annual total number of boarding passengers was estimated for weekday conditions. A MARTA ticket costs $2.50 for a one-way trip. In 2014, approximately 75,500 people commuted to work on MARTA per workday, among those approximately 41,500 people used MARTA because they had no other alternative means of transportation [110]. Instead of purchasing single-way tickets for $2.50 each trip, these people were assumed to purchase a monthly pass, which translates to a lower per-trip cost. For estimation purposes, the monthly pass was assumed to be $80, which is equivalent to $1.82 per trip based on the assumption of 22 weekdays per month and two trips per weekday). The annual revenues were estimated by multiplying the numbers of boarding passengers by the respective trip costs.

This page is intentionally left blank.

6 FINDINGS/RESULTS

6.1 Findings from Task 2

The developed power and propulsion system for each of the lead and follower module prototypes works properly when operated by a remote-control unit, for motor throttling (forward and reverse), steering, regenerative braking, and emergency braking. Outdoor tests indicate that the design of these prototypes yields performance that meets or exceeds the technical objectives (mainly straight-line speed and cornering speed) proposed for their power, propulsion, steering, and braking systems.

6.2 Findings from Task 3

The developed leader–follower controller works properly in an indoor environment. In order to solve the measurement delay problem, a dual-Kalman-filter strategy and a multi-thread programming technique were integrated into the control scheme. The indoor experimental results using two autonomous vehicles validated the effectiveness and robustness of the proposed approach. Meanwhile, the researchers observed some challenges in the outdoor tests. In particular, a regular laser sensor cannot obtain correct measurements in a bright outdoor environment.

6.3 Findings from Task 1

The scenarios studied for both the Cobb Parkway and GA 400 corridors are summarized in Table 9.

TABLE 9
Summary of Study Scenarios

Corridor	Scenario	Study Period
Cobb Parkway	BRT - 27% Ridership Increase	AM Peak, PM Peak, Off-peak
	SMFe - 27% Ridership Increase	AM Peak, PM Peak, Off-peak
	SMFe - 32% Ridership Increase	AM Peak, PM Peak, Off-peak
GA-400	BRT - 5 min headway	AM Peak, PM Peak
	BRT - 15 min headway	Off-peak
	SMFe - 5 min headway	AM Peak, PM Peak
	SMFe - 10 min headway	AM Peak, PM Peak
	SMFe - 15 min headway	Off-peak
	SMFe - 20 min headway	Off-peak

Five simulations were conducted for each study period corresponding to each scenario in Table 9. This resulted in 45 (i.e., 9×5) simulation runs for each corridor.

6.3.1 Cobb Parkway Corridor Results

6.3.1.1 Network Performance

The network results from simulations for the Cobb Parkway corridor are summarized in Table 10. Note that the results are the averages of five simulation runs.

As shown in Table 10, the Average Delay for BRT 27% and SMFe 27% are relatively the same for the AM peak condition. For the PM peak condition, SMFe 27% has a slightly higher Average Delay than the BRT 27%. For the off-peak condition, SMFe 27% shows a slightly lower Average Delay than BRT 27%. The same trend applies to the metrics of Average Stop Delay and Average Stops.

The poorer performance of SMFe during the PM peak condition (which is worse than the AM peak condition) could be attributable to the fact that the slimmer SMFe modules, which are virtually coupled by design, will result in an overall longer body than the traditional BRT vehicle. Given the transit signal priority, the direct implication of this design feature is a longer disruption to the signal operations than its counterpart BRT vehicle because of a longer dwell time for the transit phase. This likely caused more stops or longer delays to other vehicles in the conflicting

TABLE 10
Cobb Parkway – Network Results from Vissim Simulations

Performance Indicator		AM Peak			PM Peak			Off-Peak		
		BRT 27%	**SMFe 27%**	**SMFe 32%**	**BRT 27%**	**SMFe 27%**	**SMFe 32%**	**BRT 27%**	**SMFe 27%**	**SMFe 32%**
Highway Vehicles	Average Delay	205	205	200	212	219	214	98	96	94
	Average Stop Delay	147	147	143	161	167	163	64	62	61
	Average Stops	2.73	2.73	2.65	2.71	2.75	2.69	1.92	1.88	1.85
Transit Vehicles	Average Delay	341	447	445	342	489	492	519	454	454
	Average Stop Delay	178	125	121	179	147	153	353	293	285
	Average Stops	7.05	5.38	5.31	7.26	6.38	6.36	6.97	6.47	6.90
Transit Users	Total Alighting Passengers	1,890	1,979	1,838	2,216	1,957	1,793	2,418	2,020	1,959
	Total Boarding Passengers	2,105	2,280	1,945	2,358	2,192	1,919	2,572	2,278	2,085
	Total Occupancy	3,515	4,047	3,282	4,315	4,098	3,377	4,374	4,052	3,744
	Total Waiting Passengers	12,227	14,926	11,696	15,838	14,474	11,565	15,352	14,754	13,161
	Total Waiting Time(sec.)	107,815	119,313	113,776	121,625	122,961	110,942	128,768	117,520	118,004
	Average Waiting Time (sec./person)	8.82	7.99	9.73	7.68	8.50	9.59	8.39	7.97	8.97

traffic flow. It also depends on the time point in the signal cycle at which the SMFe vehicle approaches each signalized intersection along the arterial. This effect would be nonexistent if the SMFe-bus operates on a freeway. On the other hand, the enhanced performance of the SMFe-BRT during the off-peak condition is due to the reduced number of modules (i.e., two modules for the off-peak condition as compared to three modules for the AM and PM peak conditions) where transit signal priority would cause less disruption to the general traffic flow.

Additional automobile users switching to SMFe-BRT will lead to increased ridership. For the SMFe 32% scenario, this will result in a slightly reduced network vehicle volume, which explains the slight reduction in Average Delay, Average Stop Delay, and Average Stops as compared to the SMFe 27% scenario. It should be noted that the possible latent demand of network traffic was not explicitly accounted for in the simulations.

By comparing SMFe 32% with BRT 27%, all three metrics (i.e., Average Delay, Average Stop Delay, and Average Stops) are lower for SMFe 32%, which signals a better performance of SMFe for the AM peak condition. However, for the PM peak condition, the SMFe 32% still results in a higher Average Delay and Average Stop Delay since the PM peak condition is worse than the AM peak condition. This implies that additional ridership increase is required to outperform the BRT 27%. However, if measured by Average Stops, SMFe 32% outperforms BRT 27%.

The performance summary for the transit vehicle in Table 10 reveals a similar trend as discussed previously, except that SMFe has a lower Average Stops and Average Stop Delay.

Based on the statistics on transit users, for SMFe 27%, the Average Waiting Time is lower for the AM peak and off-peak conditions because more passengers were served during those periods as compared to BRT 27%. However, the Average Waiting Time for SMFe 27% becomes higher than for BRT 27% in the PM peak condition due to a smaller number of users served because of its longer service time. Increasing ridership to 32% results in a longer Average Waiting Time. This is due to the fact that three modules were used with an 8-minute headway for both the

27% SMFe and 32% SMFe scenarios. For the SMFe 32% scenario, all transit users waiting at each station may not be picked up at once, resulting in an overall longer Average Waiting Time.

6.3.1.2 Evaluation based on Multiple Criteria

Multicriteria evaluation was conducted based on three criteria: (1) Transport, (2) Environment, and (3) Finance. Each criterion includes a number of subcriteria. For example, the Transport criterion was evaluated based on two subcriteria: Average Delay and Average Passenger Wait Time for transit. Those performance data were compiled from network simulation runs. The Environmental impact was evaluated based on a list of subcriteria of various types of emissions, which are also obtained from network simulation runs. The Finance criterion considers three subcriteria: Capital Cost, Operations and Maintenance, and Revenue. The values derived for those various subcriteria are summarized in Table 11. As seen, the highest emission was CO_2, followed by CO, NO_x, and CH_4. SMFe had lower capital cost because of its narrower module bodies, which require 25% less material for the dedicated lane. It should be pointed out that right-of-way costs were not considered in this study since it varies dramatically by locations. Significant capital savings can be reaped if the right-of-way costs were considered. The O&M costs are similar. SMFe 32% had a higher revenue because of the additional ridership.

TABLE 11
Cobb Parkway – Summary of Subcriteria Values

		BRT 27%			SMFe 27%			SMFe 32%		
Criteria	**Sub Criteria**	**AM**	**PM**	**OFF-PEAK**	**AM**	**PM**	**OFF-PEAK**	**AM**	**PM**	**OFF-PEAK**
Transport	Avg. Delay (sec./veh)	204.99	204.57	98.28	204.78	215.24	95.91	199.85	219.93	93.54
	Avg. Passenger Wait Time (sec./person)	7.68	8.50	9.59	8.82	7.99	9.73	8.39	7.97	8.97
Environment (Emissions)	CO2 (g)	36926.39	37724.26	36665.99						
	CO (g)	97.32	99.42	96.64						
	PM 2.5 (g)	0.03	0.03	0.03						
	NOx (g)	35.78	36.55	35.53						
	CH4 (g)	32.33	33.03	32.10						
	N2O (g)	2.87	2.93	2.85						
	VOC (g)	0.28	0.29	0.28						
Finance	Capital Cost	$ 152,995,920			$ 114,746,940			$ 114,746,940		
	O&M	$ 849,197			$ 862,428			$ 862,428		
	Revenue	$ 3,568,033			$ 3,469,478			$ 3,804,900		

Based on the subcriteria values in Table 11, the AHP model was applied by comparing two scenarios: (1) BRT 27% vs. SMFe 27%, and (2) BRT 27% vs. SMFe 32%, for three operating periods: AM peak, PM peak, and off-peak. The results are summarized in Table 12.

TABLE 12
Cobb Parkway – Multicriteria Evaluation Results

Operating Hour	Comparison	BRT	SMFe	Preferred
AM Peak	BRT 27% and SMFe 27%	0.435	0.565	SMFe
PM Peak	BRT 27% and SMFe 27%	0.432	0.568	SMFe
Off Peak	BRT 27% and SMFe 27%	0.429	0.571	SMFe
AM Peak	BRT 27% and SMFe 32%	0.430	0.570	SMFe
PM Peak	BRT 27% and SMFe 32%	0.428	0.572	SMFe
Off Peak	BRT 27% and SMFe 32%	0.424	0.576	SMFe

As shown in Table 12, with a higher overall score, SMFe is preferred for all six comparisons.

6.3.2 GA 400 Corridor Results

GA 400 is a freeway corridor, which is different than Cobb Parkway. As such, transit signal priority is not considered as an influential factor to the network performance. The main variable in this setting was the operating time headway of transit vehicles. To compare the two transit alternatives, BRT and SMFe, in a realistic fashion, the headway was varied by the operating time period. A 5-minute headway was used for AM peak and PM peak periods for BRT. Two headways, 5 minutes and 10 minutes, were used for SMFe. For the off-peak period, a 15-minute headway was used for BRT and two headways of 15 minutes and 20 minutes were used for SMFe. The goal of using two headway settings was to examine the operational flexibility of SMFe by varying the number of modules.

6.3.2.1 Network Performance

The network results from simulations for the GA 400 corridor are summarized in Table 13. As shown in Table 13, the Average Waiting Time was significantly lower, nearly in half, if SMFe was operated in 10-minute headway as compared to 5-minute headway. The Total Waiting

Passengers were relatively the same for the AM peak and PM peak periods, but the Total Waiting Time was reduced by almost 50%. By referencing the SMFe operations settings in Table 7, a varying number of modules (2–4) was suggested based on the headway. The waiting time is dramatically reduced due to operating more modules with a longer headway. However, for the off-peak period when the demand is not an issue, increasing headway will in turn increase the Average Waiting Time because passengers have to wait longer times at stations due to the longer headway.

For GA 400, the Average Passenger Wait Time was used as the only subcriterion for the Transport criterion because of the separated operation of transit vehicles from other vehicles. Note the distinction between GA 400 and Cobb Parkway, which is an arterial corridor where the transit vehicles interact with other vehicles at signalized intersections with transit signal priority.

Other subcriteria values are summarized in Table 14. Note that the highest emission was CO_2, followed by CO, NO_x, and CH_4. The amounts of emissions are larger than those of Cobb Parkway because there is a much heavier traffic volume on GA 400. For the off-peak period, the lower O&M cost is due to the longer headway (less frequent bus operations) and the lower revenue results from a lower ridership.

TABLE 13
GA 400 – Transit Results from Vissim Simulations

Performance Indicator	AM Peak			PM Peak			Off Peak		
	BRT hw = 5 min	SMFe hw = 5 min	SMFe hw = 10min	BRT hw = 5 min	SMFe hw = 5 min	SMFe hw = 10min	BRT hw = 15 min	SMFe hw = 15 min	SMFe hw = 20min
Total Alighting Passengers	547	540	542	543	537	554	365	486	475
Total Boarding Passengers	2,798	2,808	2,699	2,682	2,649	2,592	1,248	2,104	2,070
Total Occupancy	5,158	5,114	5,068	5,006	4,936	4,916	2,810	3,921	3,916
Total Waiting Passengers	6,312	6,270	6,362	6,211	6,072	6,092	21,114	15,861	5,125
Total Waiting Time(sec.)	190,441	184,527	90,431	190,460	186,626	91,520	75,364	67,339	48,713
Average Waiting Time (sec./person)	30.17	29.43	14.22	30.66	30.74	15.02	3.57	4.25	9.50

TABLE 14
GA 400 – Summary of Subcriteria Values

Criteria	Sub Criteria	BRT			SMFe					
		5 min headway AM	5 min headway PM	15 min headway OFF-PEAK	5 min headway AM	5 min headway PM	15 min headway OFF-PEAK	10 min headway AM	10 min headway PM	20 min headway OFF-PEAK
Transport	Avg. Passenger Wait Time (sec./person)	30.17	30.66	3.57	29.43	30.74	4.25	14.22	15.02	9.50
Environment (Emissions)	CO2 (g)	137,547.10	137,605.00	45,982.20						
	CO (g)	362.51	362.67	121.19						
	PM 2.5 (g)	0.10	0.10	0.03						
	NOx (g)	133.27	133.32	44.52						
	CH4 (g)	120.43	120.48	40.26						
	N2O (g)	10.70	10.70	3.58						
	VOC (g)	1.04	1.04	0.35						
Finance	CAPITAL COST	$ 152,995,920			$ 114,746,940			$ 114,746,940		
	O&M	$ 188,279	$ 187,784	$ 99,537	$ 197,994	$ 197,868	$ 103,379	$ 128,550	$ 128,511	$ 79,398
	REVENUES	$ 1,493,251	$ 1,557,729	$ 694,675	$ 1,475,210	$ 1,563,297	$ 1,171,638	$ 1,443,027	$ 1,502,828	$ 1,152,484

Based on the subcriteria values in Table 14, the AHP model was applied by comparing BRT and SMFe operations subject to the same and different time headways, which were varied by time periods. The results are summarized in Table 15. All comparisons, except the off-peak period with the longer headways (i.e., 15 minutes for BRT and 20 minutes for SMFe), indicate that SMFe is a better option because of an overall higher evaluation score. In particular, operating SMFe at 10-minute headway during both AM peak and PM peak periods is clearly superior to operating BRT at 5-minute headway during the same peak periods. This is indicated by the nearly doubled evaluation scores of SMFe compared to BRT (highlighted rows in Table 15).

TABLE 15
GA 400 – Multicriteria Evaluation Results

Operating Hour	Comparison	BRT	SMFe	Preferred
AM Peak	5 min headway for both BRT and SMFe	0.428	0.572	SMFe
PM Peak	5 min headway for both BRT and SMFe	0.430	0.570	SMFe
Off Peak	15 min headway for both BRT and SMFe	0.426	0.574	SMFe
AM Peak	5 min headway for BRT and 10 min headway for SMFe	0.334	0.666	SMFe
PM Peak	5 min headway for BRT and 10 min headway for SMFe	0.338	0.662	SMFe
Off Peak	15 min headway for BRT and 20 min headway for SMFe	0.504	0.496	BRT

This page is intentionally left blank.

7 CONCLUSIONS

This research project sought to improve upon conventional BRT via a novel vehicle concept that the research team called the SMFe-bus. The key features of this vehicle are: (1) its narrower width (25–50% slimmer than a regular bus), requiring less right-of-way; (2) that it consists of a lead module with a driver cab, and a few driverless follower modules/cars trailing behind it; (3) that its follower modules can be easily attached and detached from the preceding module by way of "virtual coupling" to meet varying passenger demand by time of day with optimized operations; and (4) given the smaller size of the modules, that each of them are self-propelled by in-wheel electric motors, which will allow the modules to better negotiate turns while being more friendly to the environment than fossil-fuel engines. The significance of this project is that it will lead to a system that costs less than conventional BRT (by reducing right-of-way and construction costs), while providing an equal or better level of service, and is more environmentally friendly.

This initial phase of the planned multi-stage research project was aimed at achieving the following objectives regarding the Slim Modular Flexible Electric Bus Rapid Transit (SMFe-BRT) concept and the Slim Modular Flexible Electric Bus (SMFe-bus) vehicle:

1. Demonstrate a higher benefit-to-cost ratio for the SMFe-BRT approach compared to the existing BRT approach (using Cobb County's BRT proposal and MARTA's GA 400 Transit Initiative's BRT option as case studies), and determine infrastructure design and operational feature requirements
2. Develop two-wheel-drive prototype lead and follower SMFe-bus modules with 3-hp motors and 150-Ah battery pack, capable of speeds greater than 15 mph
3. Demonstrate straight-line following by the two-module prototype SMFe-bus at 15 mph within an 8-ft-wide path, and also proper tracking of 90-degree cornering at 4 mph within the swept path of a 40-ft city transit bus

The following was accomplished:

1. For the feasibility study, the SMFe-BRT was evaluated against the traditional BRT at two projects sites (Cobb Parkway and GA 400) in Georgia, where BRT options are currently considered. This evaluation was conducted in a multicriteria context. Network operations data, including delay, stops, emissions, and revenues, were obtained from multiple simulation runs for different scenarios. Regional cost data were utilized to estimate capital and O&M costs. The key findings of this study are as follows:
    a. The SMFe-BRT service times are generally longer than those of the traditional BRT due to the overall longer vehicle body when operating during peak periods of demand. The increased number of modules during the peak periods provides additional carrying capacity, but also increases the service time for boarding and alighting at busy stations.
    b. When operating along an arterial corridor, because of transit signal priority, the dwelling time for SMFe-BRT phase would be longer than that for the traditional BRT during the peak periods because of the longer body of the SMFe-bus (consisting of virtually coupled multiple modules) than the traditional BRT vehicle. This likely disrupts traffic flow of other vehicles, especially those in conflicting movements during the peak periods. As a result, the overall network delay may be increased. Note that the operational benefits expected from the SMFe-BRT mainly include less waiting time for transit riders (because of demand-responsiveness) and fewer delays or congestion for other vehicles (because of reduced network traffic due to shifted trips to the SMFe-BRT mode). Implementing SMFe-BRT would be desirable if those operations benefits outweigh the adverse effect of traffic signal disruption to other vehicles.
    c. When operating along a freeway corridor, the headway would be a key variable for SMFe-BRT operations. As demonstrated in this study, significant benefits can be

reaped if proper headways are selected in accordance with operating periods of different demands.

   d. Fuel type is one of the main differences between the traditional BRT and the SMFe-BRT. The traditional BRT in this study was assumed to be powered by compressed natural gas, while SMFe-BRT will be fully electric. As such, SMFe-BRT is assumed to have zero emissions. However, this is contingent upon the assumption that the electricity driving the SMFe vehicle is generated from a clean energy source.

2. The developed power and propulsion system for each of the lead and follower module prototypes works properly when operated by a remote-control unit, for motor throttling (forward and reverse), steering, regenerative braking, and emergency braking. Outdoor tests indicate that the design of these prototypes yields performance that meets or exceeds the technical objectives (mainly straight-line speed and cornering speed) proposed for their power, propulsion, steering, and braking systems.
3. The developed leader–follower controller works properly in an indoor environment. In order to solve the measurement delay problem, a dual-Kalman-filter strategy and a multi-thread programming technique were integrated into the control scheme. The indoor experimental results using two autonomous vehicles validated the effectiveness and robustness of the proposed approach, and demonstrated module straight-line tracking up to 4 mph for time intervals of several seconds long. Meanwhile, researchers observed some challenges in the outdoor tests. Specifically, a regular laser sensor cannot obtain correct measurements in a bright outdoor environment.

Hence, this project fully accomplished two of its three objectives, while partially reaching the objective of demonstrating straight-line following by the two-module prototype SMFe-bus at 15 mph, and also proper tracking of 90-degree cornering at 4 mph.

This page is intentionally left blank.

8 RECOMMENDATIONS

Although substantial progress has been made toward a better BRT system, additional research and development work is needed, especially to improve the SMFe-bus' module tracking performance, to make the proposed SMFe-BRT system a reality.

This page is intentionally left blank.

9 REFERENCES

[1] T. Brown and R. Paling, "*Getting More from Our Roads: An Evaluation of Special Vehicle Lanes on Urban Arterials*," NZ Transport Agency, Research Report 557, 2014, 135p.

[2] Bureau of Transportation Statistics, "Georgia transportation by the numbers," January 2016, https://www.bts.gov/sites/bts.dot.gov/files/legacy/georgia.pdf.

[3] "Transportation Remains Top Concern For Metro Atlanta Residents," https://patch.com/georgia/atlanta/transportation-remains-top-concern-metro-atlanta-residents.

[4] C.M. Nichols, "Chicago MPC builds its case for top-tier BRT: Community contributions stretch well beyond faster bus service," *Bus Ride*, Vol. 50, No. 1, 2014, pp. 22–23.

[5] T. Orosz and E. Beaton, "Select Bus Services delivers BRT to New York City," *Bus Ride*, Vol. 50, No. 1, 2014, pp. 24–25.

[6] D.R. Heres, D. Jack, and D. Salon, "Do public transport investments promote urban economic development? Evidence from bus rapid transit in Bogotá, Colombia," *Transportation*, Vol. 41, No. 1, 2014, pp. 57–74.

[7] L. Ma, R. Ye, and H. Titheridge, "Capitalization Effects of Rail Transit and Bus Rapid Transit on Residential Property Values in a Booming Economy: Evidence from Beijing," *Transportation Research Record: Journal of the Transportation Research Board*, No. 2451, 2014, pp. 139–148.

[8] P. May, "vivaNext – Highway 7 East (H3) BRT Dedicated Lanes, Transportation 2014: Past, Present, Future," 2014 Conference and Exhibition of the Transportation Association of Canada.

[9] C.E. Siedler, "Can Bus Rapid Transit be a Sustainable Means of Public Transport in Fast Growing Cities? Empirical Evidence in the Case of Oslo," *Transportation Research Procedia*, Vol. 1, No. 1, 2014, pp. 109–120.

[10] D. Rutherford, "sbX brings BRT to the Inland Empire," *Bus Ride*, Vol. 50, No. 4, 2014, pp. 26–27.

[11] A.L. Dodero, P.M. dos Santos da Rocha, J.J. Hernandez, et al., "Evaluating Improvements in Bus Rapid Transit in Mexico City, Mexico: How Feasible Is It to Improve a Consolidated System?" *Transportation Research Record: Journal of the Transportation Research Board*, No. 2451, 2014, pp. 88–96.

[12] M.A. Ortiz and J.P. Bocarejo, "TRANSMILENIO BRT Capacity Determination Using a Microsimulation Model in VISSIM," Transportation Research Board 93rd Annual Meeting, 2014, 17p.

[13] D. Klepal, "Cobb seeks SPLOST cash for rapid bus system," *The Atlanta Journal-Constitution*, http://www.ajc.com/news/news/cobb-seeks-splost-cash-for-rapid-bus-system/ngYTW/. [Accessed Feb. 29, 2016].

[14] J. Magnusson, "BRT Brings More Bang for the Buck," *Sustainable Transport*, Vol. 25, 2014, pp. 8–9.

[15] D. Wickert, “MARTA: No longer a dirty word in Gwinnett?” *The Atlanta Journal-Constitution*, http://www.myajc.com/news/news/local-govt-politics/marta-no-longer-a-dirty-word-in-gwinnett/nqS4g/. [Accessed Feb. 29, 2016].

[16] X. Wang and Q. Li, “Utilization of the spare capacity of exclusive bus lanes based on a dynamic allocation strategy,” *WIT Transactions on the Built Environment*, Vol. 138, 2014, pp. 173–189.

[17] T. Liu, A. Ceder, J. Ma, and G. Wei, “CBVC-B: A System for Synchronizing Public-Transport Transfers Using Vehicle-to-Vehicle Communication,” *Procedia – Social and Behavioral Sciences*, Vol. 138, 2014, pp. 241–250.

[18] M.E. Mallia and K. Simpson, *Wireless Global Positioning System Fleet Tracking System at the University at Albany*, NYSERDA Report 14-27, 2014, 34p.

[19] G. Ellem, C. Matthews, and N. Tyson, “Fast charge batteries and in route charging: an emerging option for low-cost freight electrification,” CORE 2014, Rail Transport for a Vital Economy, Conference on Railway Engineering, Adelaide, South Australia, 5–7 May 2014.

[20] Y.C. Kim, K-H. Yun, and K-D. Min, “Automatic guidance control of an articulated all-wheel-steered vehicle,” *Vehicle System Dynamics*, Vol. 52, No. 4, 2014, pp. 456–474.

[21] M. Conte, F. Vellucci, M. Ceraolo, et al., “Energy storage system studies for heavy-duty hybrid electric vehicles in the EC HCV project,” Transport Research Arena (TRA) 5th Conference: Transport Solutions from Research to Deployment, 2014, 10p.

[22] K. Amagai, T. Takarada, M. Funatsu, and K. Nezu, “Development of Low-CO_2-emission Vehicles and Utilization of Local Renewable Energy for the Vitalization of Rural Areas in Japan,” IATSS Research, Vol. 37, No. 2, 2014, pp. 81–88.

[23] M.J. Bradley & Associates, *Updated Comparison of Energy Use & CO_2 Emissions from Different Transportation Modes*, American Bus Association, 2014, 17p.

[24] A. Lajunen, “Energy consumption and cost–benefit analysis of hybrid and electric city buses,” *Transportation Research Part C: Emerging Technologies*, Vol. 38, 2014, pp. 1–15.

[25] A. Alam, B. Besselink, V. Turri, J. Martensson, and K.H. Johansson, “Heavy-Duty Vehicle Platooning for Sustainable Freight Transportation: A Cooperative Method to Enhance Safety and Efficiency,” in *IEEE Control Systems*, Vol. 35, No. 6, Dec. 2015, pp. 34–56.

[26] T. Zhao and Y. Wang, “A neural-network based autonomous navigation system using mobile robots,” Proc. Intl. Conf. Control Automation Robotics & Vision (ICARCV), December 2012.

[27] D. Hidalgo and J.C. Muñoz, “A review of technological improvements in bus rapid transit (BRT) and buses with high level of service (BHLS),” *Public Transport*, Vol. 6, No. 3, 2014, pp. 185–213.

[28] Proterra webpage: https://www.proterra.com/products/

[29] C. Curry, “Lithium-ion Battery Costs: Squeezed Margins and New Business Models,” July 10, 2017, https://about.bnef.com/blog/lithium-ion-battery-costs-squeezed-margins-new-business-models/. [Accessed July 26, 2018].

[30] “Proterra DuoPower™: Overview and Technical Specifications,” AxleTech webpage, https://www.axletech.com/en/products/electric-solutions/proterra-duopowertm.

[31] L. de Novellis, A. Sorniotti, P. Gruber, et al., *Torque Vectoring for Electric Vehicles with Individually Controlled Motors: State-of-the-Art and Future Developments* In: 26th International Electric Vehicle Symposium (EVS26), May 2012, Los Angeles, CA.

[32] "Elaphe Technology," Elaphe webpage, http://in-wheel.com/technology/.

[33] "Key Features," Protean webpage, https://www.proteanelectric.com/protean-drive/.

[34] "Elaphe in-wheel electric motors," Elaphe webpage, http://in-wheel.com/.

[35] "Motor Controller | EV Parts – Kelly Controls, LLC," Kelly webpage, http://kellycontroller.com/brushless-hub-motors-c-21_62.html?osCsid=rt8fo3dihss7dka06iva3gvbe5.

[36] "Kelly Controller KLS7222H, 24V-72V, 220A, Sealed Sinusoidal Wave BLDC Motor Controller," Kelly webpage, http://kellycontroller.com/kls7222h24v-72v220asealed-sinusoidal-wave-bldc-motor-controll-p-1448.html. [Accessed Jan. 9, 2017].

[37] "Kelly Controller Hub Motor 48V 2KW (disc-brake)," Kelly webpage, http://kellycontroller.com/hub-motor-48v-2kwdisc-brake-p-164.html. [Accessed Jan. 9, 2017].

[38] "Turnigy 5X 5Ch Mini Transmitter and Receiver (Mode 2)," Hobby King webpage, https://hobbyking.com/en_us/turnigy-5x-5ch-minitransmitter-and-receiver-mode-2.html. [Accessed Jan. 9, 2017].

[39] "Raspberry Pi 3 Model B," Raspberry Pi Foundation webpage, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. [Accessed Jan. 9, 2017].

[40] "MCP4725 12-Bit DAC with Raspberry Pi: Overview," Adafruit webpage, https://learn.adafruit.com/mcp4725-12-bit-dac-with-raspberry-pi/overview. [Accessed Jan. 9, 2017].

[41] U.S. Department of Transportation, "Connect Cobb Corridor Environmental Assessment," Federal Transit Administration, April 2015, https://cobbcounty.org/images/documents/dot/studies/ConnectCobb/EA/Connect_Cobb_Corridor_EA_April_2015_FINAL.pdf. [Accessed Jan. 9, 2017].

[42] Electropaedia, "Electric Vehicle Charging Infrastructure," http://www.mpoweruk.com/infrastructure.htm. [Accessed Jan. 9, 2017].

[43] A. Loria, J. Dasdemir, and N. Jarquin-Alvarez, "Decentralized formation-tracking control of autonomous vehicles on straight paths," *IEEE 53rd Annual Conference on Decision and Control*, December 2014.

[44] R.D. Cruz-Morales, M. Velasco-Villa, R. Castro-Linares, and E.R. Palacios-Hernandez, "Leader–follower formation for nonholonomic mobile robots: Discrete-time approach," *International Journal of Advanced Robotic Systems*, Vol. 13, No. 2, 2016.

[45] C. Paliotta and K.Y. Pettersen, "Leader–follower synchronization with disturbance rejection," *2016 IEEE Conference on Control Applications*, September 2016.

[46] X. Chen, P. Yan, and A. Serrani, "On input-to-state stability-based design for leader/follower formation control with measurement delays," *International Journal of Robust and Nonlinear Control*, Vol. 23, No. 13, 2013, pp. 1433–1455.

[47] S. Mori and T. Namerikawa, "Formation control considering disconnection of network links for a Multi-UAV system: An LMI approach," *Journal of Robotics and Mechatronics*, Vol. 28, No. 3, 2016, pp. 343–350.

[48] C.B. Low, “A flexible leader–follower formation tracking control design for nonholonomic tracked mobile robots with low-level velocities control systems,” *18th IEEE International Conference on Intelligent Transportation Systems*, September 2015.

[49] E. Bicho and S. Monteiro, “Formation control for multiple mobile robots: a non-linear attractor dynamics approach,” *IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2003.

[50] M.A. Dehghani, and M.B. Menhaj, “Communication free leader–follower formation control of unmanned aircraft systems,” *Robotics and Autonomous Systems*, Vol. 80, 2016, pp. 69–75.

[51] K. Maeda and E. Konaka, “Cruise control of a two-wheeled vehicle based on MPC to predict the trajectory of a preceding vehicle,” *53rd Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, September 2014.

[52] A. Saxena, H. Li, D. Goswami, and C.B. Math, “Design and analysis of control strategies for vehicle platooning,” *IEEE 19th International Conference on Intelligent Transportation Systems*, November 2016.

[53] W.B. Dunbar and D.S. Caveney, “Distributed receding horizon control of vehicle platoons: Stability and string stability,” *IEEE Transactions on Automatic Control*, Vol. 57, No. 3, 2012, pp. 620–633.

[54] M. Saska, V. Vonasek, T. Krajnik, and L. Peuil, “Coordination and navigation of heterogeneous MAV-UGV formations localized by a ‘hawkeye’-like approach under a model predictive control scheme,” *International Journal of Robotics Research*, Vol. 33, No. 10, 2014, pp. 1393–1412.

[55] Z. Sun and Y. Xia, “Receding horizon tracking control of unicycle-type robots based on virtual structure,” *International Journal of Robust and Nonlinear Control*, Vol. 26, No. 17, 2016, pp. 3900–3918.

[56] H. Koroglu and P. Falcone, “State feedback synthesis for homogenous platoons under the leader and predecessor following scheme,” *13th European Control Conference*, June 2014.

[57] H. Koroglu and P. Falcone, “Controller synthesis for a homogenous platoon under leader and predecessor following scheme,” *2014 American Control Conference*, June 2014.

[58] C. Chen, Y. Xing, V. Djapic, and W. Dong, “Distributed formation tracking control of multiple mobile robotic systems,” *IEEE 53rd Annual Conference on Decision and Control*, December 2014.

[59] H. Koroglu and P. Falcone, “Joint synthesis of dynamic feed-forward and static state feedback for platoon control,” *IEEE 53rd Annual Conference on Decision and Control*, December 2014.

[60] C. Chen, F.D.L. Torre, and W. Dong, “Distributed exponentially tracking control of multiple wheeled mobile robots,” *2014 American Control Conference*, June 2014.

[61] L. Brinon-Arranz, A. Pascoal, and A.P. Aguiar, “Adaptive leader–follower formation control of autonomous marine vehicles,” *IEEE 53rd Annual Conference on Decision and Control*, December 2014.

[62] T. Yucelen and E.N. Johnson, “Control of multivehicle systems in the presence of uncertain dynamics*,” International Journal of Control*, Vol. 86, No. 9, pp. 1540–1553, 2013.

[63] H. Wu, M. Karkoub, and C. Hwang, “Mixed Fuzzy Sliding-Mode Tracking with Backstepping Formation Control for Multi-Nonholonomic Mobile Robots Subject to Uncertainties: Category (3), (5),” *Journal of Intelligent and Robotic Systems: Theory and Applications*, Vol. 79, No. 1, 2015, pp. 73–86.

[64] S. Hung, S.N. Givigi, and A. Noureldin, “A Dyna-Q (Lambda) approach to flocking with fixed-wing UAVs in a stochastic environment,” *IEEE International Conference on Systems, Man, and Cybernetics*, October 2015.

[65] Z. Peng, D. Wang, Z. Chen, X. Hu, and W. Lan, “Adaptive dynamic surface control for formations of autonomous surface vehicles with uncertain dynamics,” *IEEE Transactions on Control Systems Technology*, Vol. 21, No. 2, 2013, pp. 513–520.

[66] F. Rinaldi, S. Chiesa, and F. Quagliotti, “Linear quadratic control for quadrotors UAVs dynamics and formation flight,” *Journal of Intelligent and Robotic Systems: Theory and Applications*, Vol. 70, No. 1-4, 2013, pp. 203–220.

[67] J. Ghommam, H. Mehrjerdi, and M. Saad, “Robust formation control without velocity measurement of the leader robot,” *Control Engineering Practice*, Vol. 21, No. 8, 2013, pp. 1143–1156.

[68] S. Hung and S.N. Givigi, “A Q-Learning approach to flocking with UAVs in a stochastic environment,” *IEEE Transactions on Cybernetics,* Vol. 47, No. 1, 2017, pp. 186–197.

[69] A.A. Peters and O. Mason, “Leader following with non-homogeneous weights for control of vehicle formations,” *2016 IEEE Conference on Control Applications*, September 2016.

[70] A.A. Peters, R.H. Middleton, and O. Mason, “Leader tracking in homogeneous vehicle platoons with broadcast delays,” *Automatica*, Vol. 50, No. 1, 2014, pp. 64–74.

[71] Y. Liu, H. Gao, B. Xu, G. Liu, and H. Cheng, “Autonomous coordinated control of a platoon of vehicles with multiple disturbances,” *IET Control Theory and Applications,* Vol. 8, No. 18, 2014, pp. 2325–2335.

[72] J. Kwon and D. Chwa, “Adaptive bidirectional platoon control using a coupled sliding mode control method,” *IEEE Transactions on Intelligent Transportation Systems,* Vol. 15, No. 5, 2014, pp. 2040–2048.

[73] Y. Zheng, S.E. Li, K. Li, and L. Wang, “Stability Margin Improvement of Vehicular Platoon Considering Undirected Topology and Asymmetric Control,” *IEEE Transactions on Control Systems Technology*, Vol. 24, No. 4, 2016, pp. 1253–1265.

[74] X. Guo, J. Wang, F. Liao, and R.S.H. Teo, “Distributed Adaptive Integrated-Sliding-Mode Controller Synthesis for String Stability of Vehicle Platoons,” *IEEE Transactions on Intelligent Transportation Systems,* Vol. 17, No. 9, 2016, pp. 2419–2429.

[75] S. Sabau, C. Oara, S. Warnick, and A. Jadbabaie, “Optimal distributed control for platooning via sparse Coprime factorizations,” *IEEE Transactions on Automatic Control*, Vol. 62, No. 1, 2017, pp. 305–320.

[76] K. Ampountolas and M. Kring, “Mitigating bunching with bus-following models and bus-to-bus cooperation,” *IEEE 18th International Conference on Intelligent Transportation Systems: Smart Mobility for Safety and Sustainability*, September 2015.

[77] Q. Li and Z. Jiang, “Pattern preserving path following of unicycle teams with communication delays,” *Robotics and Autonomous Systems*, Vol. 60, No. 9, 2012, pp. 1149–1164.

[78] B. Hu and M.D. Lemmon, "Distributed Switching Control to Achieve Almost Sure Safety for Leader–Follower Vehicular Networked Systems," *IEEE Transactions on Automatic Control*, Vol. 60, No. 12, 2015, pp. 3195–3209.

[79] X. Yu and L. Liu, "Distributed Formation Control of Nonholonomic Vehicles Subject to Velocity Constraints," *IEEE Transactions on Industrial Electronics,* Vol. 63, No. 2, 2016, pp. 1289–1298.

[80] G.C. Karras, K.J. Kyriakopoulos, and G.K. Karavas, "Towards cooperation of underwater vehicles: A leader–follower scheme using vision-based implicit communications," *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015,* September 2015.

[81] E. Alpaydin, *Introduction to Machine Learning*, 3rd Edition, MIT Press, 2014.

[82] Atlanta Regional Commission, *2014 Transportation Fact Book*, 2014.

[83] Cobb County Government, "Route 10," https://cobbcounty.org/index.php?option=com_content&view=article&id=4449:route-10&catid=427&Itemid=2073. [Accessed July 24, 2017].

[84] H. Levinson, S. Zimmerman, J. Clinger, S. Rutherford et al., *Bus Rapid Transit*, Transportation Research Board, Washington, D.C., 2003.

[85] Connect Cobb Northwest Transit Corridor Environmental Assessment, U.S. Department of Transportation and Federal Transit Administration, April 2015.

[86] Connect 400 – GA 400 Transit Initiative Scoping Booklet – Environmental Impact Statement, Federal Transit Administration (FTA) and Metropolitan Atlanta Rapid Transit Authority (MARTA), April 2015.

[87] J.H. Thompson, "United States Census Bureau," April 27, 2016, file:///D:/Research/COBB%20DATA/Surveying%20U.S.%20Census%20Bureau%20Commuting%20Data%20in%20Atlanta.html. [Accessed Feb. 5, 2018].

[88] M. Peak, C. Henke, and L. Wnuk, *Bus Rapid Transit Ridership Analysis*, U.S. Department of Transportation, Federal Transit Administration, Washington, D.C., 2005.

[89] A.M. Lago, P.D. Mayworm, and J.M. McEnroe, "Ridership Response to Changes in Transit Services," *Transportation Research Record,* Vol. 818, pp. 13–19.

[90] R. Guensler, V. Elango, A. Guin, M. Hunter, et al., *Atlanta I-85 HOV-to-HOT Conversion: Analysis of Vehicle and Person Throughput*, Georgia Department of Transportation, Atlanta, 2013.

[91] M. Peak, *Analysis of Fuels and Propulsion System Options for BRT Vehicles*, Federal Transit Administration, Washington D.C., 2004.

[92] A. Faiz, C.S. Weaver, and M.P. Walsh, *Air Pollution from Motor Vehicles: Standards and Technologies for Controlling Emissions*, Washington: The World Bank, 1996.

[93] "Exhaust Emission Rates for Heavy-Duty On-road Vehicles in MOVES2014," United States Environmental Protection Agency, 2015.

[94] "Emission Factors for Greenhouse Gas Inventories," United State Environmental Protection Agency, 2014.

[95] O. Delgado and R. Muncrief, *Assessment of Heavy-Duty Natural Gas Vehicle Emissions: Implications and Policy Recommendations*, The International Council on Clean Transportation, Washington, D.C., 2015.

[96] "Conversion Factors for Hydrocarbon Emission Components," United States Environmental Protection Agency, 2005.

[97] K. Nosal and K. Solecka, "Application of AHP method for multi-criteria evaluation of variants of the integration of urban public transport," in *17th Meeting of the EURO Working Group on Transportation*, Sevilla, 2014.

[98] M.T. Tabucanon and H.-M. Lee, "Multiple Criteria Evaluation of Transportation System Improvement Projects: The Case of Korea," *Journal of Advanced Transportation,* Vol. 29, No. 1, pp. 127–143.

[99] H. Sivilevicius and L. Maskeliunaite, "The Criteria for Identifying the Quality of Passengers' Transportation by Railway and Their Ranking Using AHP Method," *Transport,* Vol. 25, No. 4, 2011, pp. 368–381.

[100] K.L. Poh and B.W. Ang, "Transportation Fuels and Policy for Singapore: an AHP Planning Approach," *Computers & Industrial Engineering,* Vol. 37, 1999, pp. 507–525.

[101] M.P. Schlickmann, L.M. Martinez, and J. Pinho de Sousa, "A tool for supporting the design of BRT and LRT services," in *20th EURO Working Group on Transportation Meeting*, Budapest, Hungary, 2017.

[102] PTV AG, "Wiedemann 74 Model Parameters," Vissim 5.40-01 User Manual, 2011, p. 136.

[103] T. Sliupas, "Annual Average Daily Traffic Forecasting Using Different Techniques," *Transport,* Vol. 21, No. 1, 2006, pp. 38–43.

[104] Atlanta Braves Stadium and Mixed-Use Development – Transportation Analysis, Kimley-Horn and Associates, Inc., May 2014.

[105] "MARTA Sustainability," Metropolitan Atlanta Region Transit Authority, http://www.itsmarta.com/marta-Sustainability.aspx. [Accessed April 6, 2018].

[106] "Connect Cobb: Northwest Transit Corridor Alternatives Analysis," Georgia Department of Transportation, Atlanta, 2012.

[107] T. McMahon, "Historical Consumer Price Index (CPI-U) Data," https://InflationData.com. [Accessed April 13, 2018].

[108] "Georgia Department of Transportation Fact Book," Georgia Department of Transportation, Atlanta, 2013.

[109] "Madison BRT Transit Corridor Study O&M Cost Estimates," Madison Area Transportation Planning Board, http://www.madisonareampo.org/documents/LMadisonBRTOMCostEstimates.pdf.

[110] MARTA Annual Report, July 1, 2013 – June 30, 2014.

This page is intentionally left blank.

10 APPENDICES

10.1 Appendix A—Low-level Python Programs

10.1.1 Raspberry Pi Programs – Summary

The SMFe-Bus prototype is made up of a lead module and a follower module. The follower module will be following/tracking the lead module without a physical connection and will rely on camera image and laser scanner data processing performed by a laptop, which will then command the follower module to make appropriate speed and steering adjustments. There are three Python programs that need to run simultaneously when testing the prototype lead and follower modules to demonstrate ‘virtual coupling and tracking’. The program operating the lead module is “lead_mod_t9.py.” This program will only be processing incoming signals from a radio frequency (RF) remote control and will not rely on data from the other programs. These incoming signals are received into the program as pulse-width-modulated (PWM) signals generated by the RC receiver.

The program operating the follower module is “lmsc_v3_27.py” and will rely on commands sent from the laptop to adjust the hub motor speeds and the steering of the follower module. Finally, the last program that has to be running is “TCP_client_v1_24.py”. This program is run on the main control laptop that will be physically situated on the follower module. The two programs that operate the follower module (the laptop’s tracking program and the RPi’s lmsc_v3_27 program) will be using the TCP_client_v1_24.py program for them to communicate according to the TCP/IP protocol.

10.1.2 Lead_mod_t9.py

The lead module program (although this program is also run on the follower module for maneuvering it from its parking spot in the lab to the test site under RC operator control) starts by extending the brake actuator, which releases pressure on the brake system. The RF transmitter provides functionality for the lead module to go forward, reverse, accelerate, brake, steer, and engage the emergency brake. The direction to accelerate is checked by the program, immediately followed by the status of the emergency brake. If the emergency brake is not engaged, the throttle signal is checked and the voltage being sent to the motor controllers is varied correspondingly. The program will next move to decipher the signal for the steering. Next, the brake signal will be checked and applied if necessary. The program operates in a polling fashion, checking the signals for the emergency brake, then the throttle, then the steering, and finally the brakes. An interrupt can be applied at any time to engage the emergency brake and exit the program.

lead_mod_t9.py

```
#!/usr/bin/env python

import time
from time import sleep
import pigpio
import Adafruit_MCP4725
import sys
import read_PWM                                 # This is an additional file. It 
needs to be in the same folder as this file to run.
import RPi.GPIO as GPIO

PWM_GPIO =4             # LEFT-RIGHT CHANNEL 1 (RIGHT JOYSTICK)
PWM_GPIO4 = 6           # REVERSE #channel 4
EMERGENCY = 12          # Emergency Brake Output
REVERSE = 16
PWM_GPIO2 = 19          # CH5 switch
THROTTLE = 21           # Microswitch
PWM_GPIO3 = 22          # UP-DOWN CHANNEL 3 (LEFT JOYSTICK)
REVERSESIGN = 23

BRK1_GPIO = 18
BRK2_GPIO = 17
BRKHALL_GPIO = 20
STRACT1_GPIO = 25       # Steering H-Bridge signal 1
STRACT2_GPIO = 26       # Steering H-Bridge signal 2
STRHALL_GPIO = 27       # Hall sensor input from steering actuator

k=10                    # Forward turning constant
kr=10                   # Reverse turning constant
SAMPLE_TIME = 1         # Adjust higher to slow down "print" time to console for 
debugging
x=1                     # Used for countdown timer to shutdown program
i=1
var=1                   # For infinite loop
c=3                    # Shutdown timer length

        #PWM Values for throttle and steering
```

```
ThrCut = 1335                                       # Current throttle cut. Values
lower than this will stop the car.
ThrHigh = 1645                                      # Maximum throttle. Used to
determine when to engagne max throttle.

StrLeft = 1080                                      # PWM for maximum left
steering
StrMidL = 1475                                      # Values betwwen StrMidL and
Str Mid H are a straight line. This is a range since the PWM doesn't stay in one
place when centered.
StrCenter = 1480                                    # Used for calcation to
determine the steering. Should be between StrMidL and StrMidH
StrMidH = 1485                                      # Values betwwen StrMidL and
Str Mid H are a straight line. This is a range since the PWM doesn't stay in one
place when centered.
StrRight = 1935                                     # PWM for maximum right
steering

FWDPWM = 1350                                       # Reverse low value. Anything
lower than this will allow it to enter FWD
REVPWM = 1650
flag = False

EBakMid = 1200                                      # Since the ebrake is a switch
this value is the middle value to deteremine on/off.

BRKMIN = 1200                                       # Start value for Braking
Range
BRKHALL = 0
#BRKHALLSTR = BRKMIN                                 # BRKHALL start value
#BRKSTR = 1300
BRKPOS = 0
#SV = False                                          # Start value flag

STRHALL = 500                                       # Hall sensor count center =
500.
STRHALLD = 0                                        # Used for steering algorithm.
Steering Hall Delta is the differnce in the desired position and the current
position.
MAX = 575                                          # Steering right limit
MIN = 425                                           # Steering left limit

class reader:
   """
   A class to read PWM pulses and calculate their frequency
   and duty cycle.  The frequency is how often the pulse
   happens per second.  The duty cycle is the percentage of
   pulse high time per cycle.
   """
   def __init__(self, pi, gpio, weighting=0.00):
      """
      Instantiate with the Pi and gpio of the PWM signal
      to monitor.

      Optionally a weighting may be specified.  This is a number
      between 0 and 1 and indicates how much the old reading
      affects the new reading.  It defaults to 0 which means
      the old reading has no effect.  This may be used to
      smooth the data.
      """
      self.pi = pi
      self.gpio = gpio

      if weighting < 0.0:
         weighting = 0.0
      elif weighting > 0.99:
         weighting = 0.99

      self._new = 1.0 - weighting      # Weighting for new reading.
      self._old = weighting            # Weighting for old reading.

      self._high_tick = None
      self._period = None
      self._high = None

      pi.set_mode(gpio, pigpio.INPUT)

      self._cb = pi.callback(gpio, pigpio.EITHER_EDGE, self._cbf)

   def _cbf(self, gpio, level, tick):

      if level == 1:

         if self._high_tick is not None:
            t = pigpio.tickDiff(self._high_tick, tick)

            if self._period is not None:
               self._period = (self._old * self._period) + (self._new * t)
            else:
               self._period = t

         self._high_tick = tick

      elif level == 0:

         if self._high_tick is not None:
            t = pigpio.tickDiff(self._high_tick, tick)

            if self._high is not None:
               self._high = (self._old * self._high) + (self._new * t)
            else:
               self._high = t


   def pulse_width(self):
      """
      Returns the PWM pulse width in microseconds.
      """
      if self._high is not None:
         return self._high
      else:
         return 0.0



   def cancel(self):
      """
      Cancels the reader and releases resources.
      """
      self._cb.cancel()

                          # Above is for reading analog PWM signal

def STEERING(self):
    global STRHALL
    if GPIO.input(STRACT1_GPIO) == 1 and GPIO.input(STRACT2_GPIO) == 0:
        STRHALL = STRHALL - 1
    elif GPIO.input(STRACT1_GPIO) == 0 and GPIO.input(STRACT2_GPIO) == 1:
        STRHALL = STRHALL + 1

def BRAKING(self1):                                    # function to determine
actuator position
    global BRKHALL
    if GPIO.input(BRK1_GPIO) == 1 and GPIO.input(BRK2_GPIO) == 0:
        BRKHALL = BRKHALL + 1          # increments hall sensor counter when
extending
    elif GPIO.input(BRK1_GPIO) == 0 and GPIO.input(BRK2_GPIO) == 1:
        BRKHALL = BRKHALL - 1          # decrements hall sensor counter when
contracting

GPIO.setmode(GPIO.BCM)
GPIO.setup(EMERGENCY,GPIO.OUT)
GPIO.output(EMERGENCY,GPIO.LOW)
GPIO.setup(REVERSE,GPIO.OUT)
GPIO.output(REVERSE,GPIO.LOW)
GPIO.setup(REVERSESIGN,GPIO.OUT)
GPIO.output(REVERSESIGN,GPIO.LOW)
GPIO.setup(THROTTLE,GPIO.OUT)
GPIO.setup(STRACT1_GPIO,GPIO.OUT)
GPIO.output(STRACT1_GPIO,GPIO.LOW)
GPIO.setup(STRACT2_GPIO,GPIO.OUT)
GPIO.output(STRACT2_GPIO,GPIO.LOW)
GPIO.setup(STRHALL_GPIO, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(BRK1_GPIO, GPIO.OUT)
GPIO.setup(BRK2_GPIO, GPIO.OUT)
GPIO.setup(BRKHALL_GPIO, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.add_event_detect(STRHALL_GPIO, GPIO.RISING, callback=STEERING, bouncetime=20)
GPIO.add_event_detect(BRKHALL_GPIO, GPIO.RISING, callback=BRAKING, bouncetime=20)
GPIO.output(REVERSESIGN,GPIO.LOW)

dac = Adafruit_MCP4725.MCP4725(address = 0x62, busnum= 1)   # i2c address (address
= 0x62) Left motor
dac2 = Adafruit_MCP4725.MCP4725(address = 0x63,busnum= 1)   # Right motor
dac3 = Adafruit_MCP4725.MCP4725(address = 0x60,busnum= 1)   # Variable Regen
Braking
dac.set_voltage(0)                                          # Start DAC's at 0V
and wait 3 seconds
dac2.set_voltage(0)
dac3.set_voltage(0)

if __name__ == "__main__":

   pi = pigpio.pi()
   p = read_PWM.reader(pi, PWM_GPIO)                  # Steering
   p2 = read_PWM.reader(pi, PWM_GPIO2)                 # E-Brake
   p3 = read_PWM.reader(pi, PWM_GPIO3)                 # Throttle
   p4 = read_PWM.reader(pi, PWM_GPIO4)                 # F/R Selection (Yaw
Channel)

   GPIO.output(BRK1_GPIO, GPIO.HIGH)
   GPIO.output(BRK2_GPIO, GPIO.LOW)
   print("Extending Actuator")
   time.sleep(10)
   GPIO.output(THROTTLE,GPIO.HIGH)                              # Starts in
forward
   GPIO.output(BRK1_GPIO, GPIO.LOW)
   GPIO.output(BRK2_GPIO, GPIO.LOW)
   BRKHALL = BRKMIN                                     # Set Hall Counter at 1200

   try:
      while var == 1:                                           # Infinte loop

         #time.sleep(SAMPLE_TIME)

         pw = p.pulse_width()                                 # Steering
         pw2 = p2.pulse_width()                              # E-Brake
         pw3 = p3.pulse_width()                              # Throttle
         pw4 = p4.pulse_width()                              # Forward/Reverse
Selection

         if pw4 > REVPWM and pw3 <= BRKMIN:                 # Reverse selection
           flag = True
           GPIO.output(REVERSE,GPIO.HIGH)
           GPIO.output(REVERSESIGN,GPIO.HIGH)

         elif pw4 < FWDPWM and pw3 <= BRKMIN:                  # Forward selection
           flag = False
           GPIO.output(REVERSE,GPIO.LOW)

         if pw2 > EBakMid:
            GPIO.output(EMERGENCY,GPIO.HIGH)                # E-Brake ON
            #GPIO.output(ABrake,GPIO.HIGH)
            print("The E-Brake is ON!")
            time.sleep(2)

         elif pw2 < EBakMid:
            GPIO.output(EMERGENCY,GPIO.LOW)                 # E-Brake OFF
            #GPIO.output(ABrake,GPIO.LOW)

         if pw2 <= EBakMid and pw3 > BRKMIN:
            if pw3 <= ThrCut and flag is False:                          # In
forward but stopped
               dac.set_voltage(0)
               dac2.set_voltage(0)
               #print("Forward stopped")
            elif pw3 >= ThrHigh and flag is False:                  # Full-speed
foward
               dac.set_voltage(2048)
               dac2.set_voltage(2048)
               #print("Forward full speed")
            elif pw3 > ThrCut and pw3 < ThrHigh and flag is False:   # Forward
speed varies: 1925-1335=590 - These ranges have been tested and are incorrect.
```

```
               dacout = int((pw3 - ThrCut)*(3.471))                  # 2048/590 =
3.47 which converts PWM to 12 bit
               dac.set_voltage(dacout)
               dac2.set_voltage(dacout)
               #print("Forward...")
            elif pw3 <= BRKMIN and flag is True:                     # In reverse
but stopped
               dac.set_voltage(0)
               dac2.set_voltage(0)
               #print("Reverse stopped")
            elif pw3 >= ThrHigh and flag is True:
       # Full-speed reverse which is mutliple of this
               dac.set_voltage(2048)                                 # value
"1966"/4096 and motor controller max.
               dac2.set_voltage(2048)
               #print("Reversed full speed")
            elif pw3 > ThrCut and pw3 < ThrHigh and flag is True:   # Reverse
speed varies: Same as FWD method
               dacout = int((pw3 - ThrCut)*(2.29))                    #
1966/850=2.29
               dac.set_voltage(dacout)
               dac2.set_voltage(dacout)
               #print("Reverse...")

         R=(500+(StrCenter - pw))                        # 500 is value random
counter value. Must match STRHALL start.
         STRHALLD = abs(STRHALL - R)
         if R > MAX:
            R = MAX
         elif R < MIN:
            R = MIN
         if STRHALLD < 5:
            GPIO.output(STRACT1_GPIO,GPIO.LOW)           # H-Bridge inputs are
both set low since steering is in position
            GPIO.output(STRACT2_GPIO,GPIO.LOW)
            #print("STEERING IN POSITION")
         elif STRHALL < R or STRHALL < MIN:
            GPIO.output(STRACT1_GPIO,GPIO.LOW)                       # H-Bright is
move steering RIGHT
            GPIO.output(STRACT2_GPIO,GPIO.HIGH)
            #print("STEERING RIGHT")
         elif STRHALL > R or STRHALL > MAX:
            GPIO.output(STRACT1_GPIO,GPIO.HIGH)                      # H-Bridge is
moving steering LEFT
            GPIO.output(STRACT2_GPIO,GPIO.LOW)
            #print("STEERING LEFT")

         BRKPOS = BRKHALL - pw3                           # BRKPOS compares
current position to desired position
         BRKTHR = BRKHALL                                 # BRKTHR is the Max
Braking Limit
         if BRKTHR < 1175:
          BRKTHR = 1175
         if BRKPOS < 4 and BRKPOS > (-4):
            time.sleep(0.01)
            GPIO.output(BRK1_GPIO, GPIO.LOW)              # Stopped when current
position and desired position are
            GPIO.output(BRK2_GPIO, GPIO.LOW)              # in range of 4 and -4
            print("Not Moving")
         elif pw3 < BRKMIN:
            if BRKPOS >= 4 and BRKHALL >= BRKTHR:
              GPIO.output(BRK1_GPIO, GPIO.LOW)                 # Moving in
              GPIO.output(BRK2_GPIO, GPIO.HIGH)
              print("Moving In/In Breaking Zone")
            elif BRKPOS >= 4 and BRKHALL < BRKTHR:        # Stopped (Max Brake
Position)
              GPIO.output(BRK1_GPIO, GPIO.LOW)
              GPIO.output(BRK2_GPIO, GPIO.LOW)
            elif BRKPOS <= (-4):
                GPIO.output(BRK1_GPIO, GPIO.HIGH)            # Moving out, in
braking range
                GPIO.output(BRK2_GPIO, GPIO.LOW)
                print("Moving Out/In Breaking Zone")
         elif pw3 >= BRKMIN:
            GPIO.output(BRK1_GPIO, GPIO.HIGH)             # Moving out, out of
braking range
            GPIO.output(BRK2_GPIO, GPIO.LOW)
            print("Moving Out/Applying Throttle")
         #elif pw3 >= BRKMIN and BRKHALL > BRKHALLSTR:
          #  GPIO.output(BRK1_GPIO, GPIO.LOW)              # Moving in, incase
actuator overshoots
          # GPIO.output(BRK2_GPIO, GPIO.HIGH)
          #  print("Moving In/Applying Throttle")


         print("BRKHALL = {} ".format(BRKHALL))
         print("BRKPOS = {} ".format(BRKPOS))
         print("BRKHALLSTR = {} ".format(BRKHALLSTR))
         #print("Steeering ={} ".format(int(pw)))        # Steering
         #print("E-Brake ={} ".format(int(pw2)))         # E-Brake
         #print("Rev/Fwd ={} ".format(int(pw4)))         # Rerse
         print("Throttle ={} ".format(pw3))       # UP DOWN
         #print("Steering Count(STRHALL) ={} ".format(STRHALL))
         #print("Steering Position(R) ={} ".format(R))
         #print("Steering PW = {} ".format(pw))
         #print("Steering HALLD = {} ".format(STRHALLD))

   except KeyboardInterrupt:
      dac.set_voltage(0)
      dac2.set_voltage(0)
      GPIO.output(EMERGENCY,GPIO.HIGH)
      GPIO.output(STRACT1_GPIO,GPIO.LOW)
      GPIO.output(STRACT2_GPIO,GPIO.LOW)
      while (x < c):
         print("PiBus is stopping in, {} seconds".format(c - x))
         x = x + 1
         sleep(1)
      GPIO.cleanup()
      sys.exit()

   p.cancel()
   pi.stop()
```

10.1.3 lmsc_v3_27.py

Similar to the lead module program, the follower module program starts by extending the brake actuator, which releases pressure on the brake system. The very next operation of the program is to check the signal of the emergency brake button. An RF remote, similar to the RF remote of the lead module, will send a signal to the follower program if the designated switch on the remote is activated. It is the only function of the follower module RF remote. All other instruction will come directly from the laptop. If the emergency brake is engaged, a series of events will ensue on the follower module. First, the electric brake system will activate via a designated GPIO pin. Then, the physical brake system actuator will start to retract, applying pressure to the brakes. The values for the speed of the two motors will be sent to the laptop during this process. Once this process has finished and the follower module has stopped, the brake system will be disengaged and the emergency brake will be switched off, as long as the actual RF remote switch has been turned off.

As long as the emergency brake switch is not activated, the program will simply wait for the TCP client to connect. When it does connect, the follower module will wait for commands from the "TCP_client_v1_24.py" program on the laptop. Currently, the "TCP_client_v1_24.py" program being run to test the follower module only provides a final speed for the follower module to accelerate to until the user cancels the program. This acceleration is optimized by a proportional–integral–derivative (PID) controller class. The program operates in a polling fashion constantly checking the emergency brake signal and then receiving data from the laptop. An interrupt can be applied at any time to engage the emergency brake and exit the program.

lmsc_v3_27.py

```
#!/usr/bin/env python

import time
from time import sleep
from datetime import datetime
import pigpio
import Adafruit_MCP4725
import Adafruit_ADS1x15
import sys
import socket
import RPi.GPIO as GPIO
import logging
import threading
import struct
import random
import math


####################################################
# RESERVED FOR PWM READER
####################################################


class reader:
    """
    A class to read PWM pulses and calculate their frequency
    and duty cycle.  The frequency is how often the pulse
    happens per second.  The duty cycle is the percentage of
    pulse high time per cycle.
    """
    def __init__(self, pi, gpio, weighting=0.00):
        """
        Instantiate with the Pi and gpio of the PWM signal
        to monitor.

        Optionally a weighting may be specified.  This is a number
        between 0 and 1 and indicates how much the old reading
        affects the new reading.  It defaults to 0 which means
        the old reading has no effect.  This may be used to
        smooth the data.
        """
        self.pi = pi
        self.gpio = gpio

        if weighting < 0.0:
           weighting = 0.0
        elif weighting > 0.99:
           weighting = 0.99

        self._new = 1.0 - weighting      # Weighting for new reading.
        self._old = weighting            # Weighting for old reading.

        self._high_tick = None
        self._period = None
        self._high = None

        pi.set_mode(gpio, pigpio.INPUT)

        self._cb = pi.callback(gpio, pigpio.EITHER_EDGE, self._cbf)

    def _cbf(self, gpio, level, tick):

        if level == 1:

            if self._high_tick is not None:
                t = pigpio.tickDiff(self._high_tick, tick)

                if self._period is not None:
                    self._period = (self._old * self._period) + (self._new * t)
                else:
                    self._period = t

            self._high_tick = tick

        elif level == 0:

            if self._high_tick is not None:
                t = pigpio.tickDiff(self._high_tick, tick)

                if self._high is not None:
                    self._high = (self._old * self._high) + (self._new * t)
                else:
                    self._high = t


    def pulse_width(self):
        """
        Returns the PWM pulse width in microseconds.
        """
        if self._high is not None:
            return self._high
        else:
            return 0.0



    def cancel(self):
        """
        Cancels the reader and releases resources.
        """
        self._cb.cancel()

                         # Above is for reading analog PWM signal

####################################################
# RESERVED FOR MODIFIED PWM READER CLASS
####################################################


class PwmReader(reader):

    def __int__(self, pi, gpio, weighting=0.00):

        reader.__init__(self, pi, gpio, weighting=0.00)

        self.gpio = gpio

    def refresh(self):

        self._high_tick = None
        self._period = None
        self._high = None
        self._cb.cancel()
        self._cb = self.pi.callback(self.gpio, pigpio.EITHER_EDGE, self._cbf)


####################################################
# RESERVED FOR Hall Sensor Handler Class
####################################################


class HsensorHandler:

    def __init__(self, pi, gpio_lft, gpio_rht):

        self.pi = pi
        self.gpio_lft = gpio_lft    # PWM GPIO (Left -> 13)
        self.gpio_rht = gpio_rht    # PWM GPIO (Right -> 23)

        # trig hall sensor reader
        self.pl = PwmReader(self.pi, self.gpio_lft)
        self.pr = PwmReader(self.pi, self.gpio_rht)

    def close(self):

        self.pl.cancel()
        self.pr.cancel()
        print("PWM readers stopped.")
        time.sleep(1)


####################################################
# RESERVED FOR SPEED CONTROLLER CLASS
####################################################


class SpdCtrl(HsensorHandler):

    def __init__(self, pi, gpio_lft, gpio_rht, dac_lft, dac_rht):

        HsensorHandler.__init__(self, pi, gpio_lft, gpio_rht)

        # PID Parameters
        self.Kpwm = 48.0/4096.0
        self.Kp = 0.840     # Initial Kp value is 0.840 (0, 1.8, 0.002)
        self.Ki = 2.02      # Initial Ki value is 2.02  (0, 3.2, 0.002)
        self.Kd = 0.008     # Initial Kd value is 0.008 (0, 0.2, 0.0002)

        # PID Parameters (Left)
        self.op_lft = 0.0    # controller output
        self.sp_lft = 0.0    # Set Point
        self.pv_lft = 0.0    # process variable
        self.e_lft = 0.0     # error
        self.ie_lft = 0.0    # integral of the error
        self.dpv_lft = 0.0   # derivative of the pv
        self.P_lft = 0.0     # proportional
        self.I_lft = 0.0     # integral
        self.D_lft = 0.0     # derivative
        self.sp_lft = 0.0    # set point

        self.pv_p_lft = 0.0    # previous value of pv
        self.ie_p_lft = 0.0    # previous value of ie

        # PID Parameters (Right)
        self.op_rht = 0.0    # controller output
        self.sp_rht = 0.0    # Set Point
        self.pv_rht = 0.0    # process variable
        self.e_rht = 0.0     # error
        self.ie_rht = 0.0    # integral of the error
        self.dpv_rht = 0.0   # derivative of the pv
        self.P_rht = 0.0     # proportional
        self.I_rht = 0.0     # integral
        self.D_rht = 0.0     # derivative
        self.sp_rht = 0.0    # set point

        self.pv_p_rht = 0.0    # previous value of pv
        self.ie_p_rht = 0.0    # previous value of ie

        # Upper and Lower limits on OP
        self.op_hi = 409.6
        self.op_lo = 0

        # PID Sample Time
        self.pid_smp_tm = 0.001  # 1ms

        # PID loop flags
        self.first_pid_lp = True

        # Initialize DAC
        self.dac = dac_lft
        self.dac2 = dac_rht

    def get_pid_parameters(self):

        pid_para = [self.Kp, self.Ki, self.Kd]
        return pid_para

    def read_left_motor_speed(self):

        return self.pv_lft

    def read_right_motor_speed(self):

        return self.pv_rht

    def write_left_motor_speed(self, pv_lft):

        self.pv_lft = pv_lft

    def write_right_motor_speed(self, pv_rht):

        self.pv_rht = pv_rht

    def refresh_left_pwm_reader(self):

        self.pl.refresh()

    def refresh_right_pwm_reader(self):
        self.pr.refresh()

    # define DAC outputer (motor controller input)
    def dac_handler(self, amature_voltage_lft, amature_voltage_rht):

        # amature_voltage
        Va_lft = amature_voltage_lft
        Va_rht = amature_voltage_rht

        if Va_lft > 4.8:
            dacout = 4096
        elif Va_lft > 0 and Va_lft <= 4.8:
            dacout = (4096/4.8)*Va_lft
        elif Va_lft <= 0:
            dacout = 0
        else:
            dacout = 0

        if Va_rht > 4.8:
            dac2out = 4096
        elif Va_rht > 0 and Va_rht <= 4.8:
            dac2out = (4096/4.8)*Va_rht
        elif Va_rht <= 0:
            dac2out = 0
        else:
            dac2out = 0

        self.dac.set_voltage(int(dacout))
        self.dac2.set_voltage(int(dac2out))

    # PID Control Algorithm
    def pid_control(self, set_point):

        # PID Contorl

        delta_t = self.pid_smp_tm
        self.sp_lft = set_point[0]
        self.sp_rht = set_point[1]

        pw_lft = self.pl.pulse_width()  # in us
        pw_rht = self.pr.pulse_width()

        # print("pw_lft = %f\tpw_rht = %f" % (pw_lft, pw_rht))

        if pw_lft > 0:
            freq_lft = 1 / ((pw_lft / 100000.0) * 2)  # in Hz
        else:
            freq_lft = 0.0

        if pw_rht > 0:
            freq_rht = 1 / ((pw_rht / 100000.0) * 2)  # in Hz
        else:
            freq_rht = 0.0

        rpm_lft = 15.0 * freq_lft
        rpm_rht = 15.0 * freq_rht
        self.pv_lft = rpm_lft
        self.pv_rht = rpm_rht
        self.e_lft = self.sp_lft - self.pv_lft
        self.e_rht = self.sp_rht - self.pv_rht

        # print("e_lft = %f\te_rht = %f" % (self.e_lft, self.e_rht))

        # calculate starting on second cycle
        if self.first_pid_lp == False:

            self.dpv_lft = -(self.pv_lft - self.pv_p_lft)/delta_t
            self.dpv_rht = -(self.pv_rht - self.pv_p_rht)/delta_t

            self.ie_lft = self.ie_p_lft + self.e_lft * delta_t
            self.ie_rht = self.ie_p_rht + self.e_rht * delta_t

        else:
            self.first_pid_lp = False

        self.P_lft = self.Kp * self.e_lft
        self.P_rht = self.Kp * self.e_rht

        self.I_lft = self.Ki * self.ie_lft
        self.I_rht = self.Ki * self.ie_rht

        self.D_lft = self.Kd * self.dpv_lft
        self.D_rht = self.Kd * self.dpv_rht

        self.op_lft = self.P_lft + self.I_lft + self.D_lft
        self.op_rht = self.P_rht + self.I_rht + self.D_rht

        # print("op = %f = %f + %f + %f" %(self.op, self.P, self.I, self.D))

        if self.op_lft > self.op_hi:  # check upper limit
            self.op_lft = self.op_hi
            self.ie_lft = self.ie_lft - self.e_lft * delta_t # anti-reset windup
        if self.op_lft < self.op_lo:  # check lower limit
            self.op_lft = self.op_lo
            self.ie_lft = self.ie_lft - self.e_lft * delta_t # anti-reset windup

        if self.op_rht > self.op_hi:  # check upper limit
            self.op_rht = self.op_hi
            self.ie_rht = self.ie_rht - self.e_rht * delta_t # anti-reset windup
        if self.op_rht < self.op_lo:  # check lower limit
            self.op_rht = self.op_lo
            self.ie_rht = self.ie_rht - self.e_rht * delta_t # anti-reset windup

        # calculate amature voltage which relates to the PID output
        Va_lft = self.op_lft * self.Kpwm
        Va_rht = self.op_rht * self.Kpwm

        # send the Va to a DAC so that it generates an output to the motor
controller
        self.dac_handler(Va_lft, Va_rht)

        self.ie_p_lft = self.ie_lft
        self.ie_p_rht = self.ie_rht

        self.pv_p_lft = self.pv_lft
        self.pv_p_rht = self.pv_rht

    # clean up
    def close(self):

        HsensorHandler.close(self)
        self.dac.set_voltage(0)
        self.dac2.set_voltage(0)
        print("\nDACs are cleaned up.\n")
        time.sleep(1)


####################################################
# RESERVED FOR TCP-IP CLASS
####################################################


class Tcpip:

    def __init__(self, ip, port):

        # Data
        self.request = ''
        self.request_p = ''
        self.ask = ''

        self.x = [0.0, 0.0, 0.0]     # Added Array value location 2

        self.TCP_SERVER_PAUSE = 0.5
        self.TCP_SERVER_TRANS_PAUSE = 0.004

        self.first_receive = False

        # define host ip: Rpi's IP
        self.HOST_IP = ip
        self.HOST_PORT = port
        print("Starting socket: TCP...")
        time.sleep(self.TCP_SERVER_PAUSE)

        # create socket object:socket=socket.socket(family,type)
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # self.server.setsockopt(socket.SOL_SOCKET,socket.SO_RCVTIMEO,
struct.pack('LL',0.005,0))
        host_addr = (self.HOST_IP, self.HOST_PORT)
        print("TCP server created @ %s:%d!" %(self.HOST_IP, self.HOST_PORT) )
        time.sleep(self.TCP_SERVER_PAUSE)

        # bind socket to addr:socket.bind(address)
        self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server.bind(host_addr)
        print('Socket bind complete.')
        time.sleep(self.TCP_SERVER_PAUSE)

        # listen connection request:socket.listen(backlog)
        self.server.listen(5)
        print('Socket now listening...\n')
        time.sleep(self.TCP_SERVER_PAUSE)
        # waiting for connection in main loop
        while True:
            print("Connecting to the client...")
            time.sleep(self.TCP_SERVER_PAUSE)
            (self.client), (client_ip, client_port) = (self.server).accept()
            print("Connection accepted from %s:%s.\n" %(client_ip, client_port))
            time.sleep(self.TCP_SERVER_PAUSE)
            self.time_0 = datetime.utcnow()
            break

    def data_trans(self, runtime, spd_ctrl_status, str_ctrl_status):

        act_spd = [spd_ctrl_status[0], spd_ctrl_status[1]]  # actual motor speed
in the sequence of left, right
        des_spd = spd_ctrl_status[2]
        pid_para = [spd_ctrl_status[3], spd_ctrl_status[4], spd_ctrl_status[5]]
        cur_loc = str_ctrl_status[0]
        cur_ang = str_ctrl_status[1]
        des_loc = str_ctrl_status[2]
        des_ang = str_ctrl_status[3]
        idx_end = 0     # looking up index for '#' (ending) from the received
string
        idx_com = 0     # looking up index for ',' (comma) from the received
string
        idx_star = 0    # looking up index for '*' (star) from the received string

        # req = ''        # temporary variable for self.request
        # i = 0           # loop counter
        # 0 - ask = 'S', indicating the server has received the set point speed
values
        # 1 - Non-legal receiving (could either be not including '#' or not
receiving anything)
        # 2 - ask = speeds (at first check if server receive 'Q#')
        # 3 - ask = 'S', indicating the server has received the desired steering
location value
        flag = 0

        # receive & send
        for i in range(1):

            delta_time = (datetime.utcnow() - self.time_0)
            tm = delta_time.seconds + delta_time.microseconds*1.0e-6
            if tm > 10:
                return -1

            self.client.setblocking(0)
            try:
                self.request = self.client.recv(64)
                self.client.setblocking(1)
            except Exception:
                flag = 1
                break
            # Check if the string includes ending symbol '#'
            if self.request.count('#'):
                idx_end = self.request.index('#')
                # flag = 0
            else:
                self.request = ''
                flag = 1
                break
            # Check if the string includes ',' which means set points are received
            if self.request.count(','):
                idx_com = self.request.index(',')
                flag = 0
            elif self.request.count('*'):
                idx_star = self.request.index('*')  # Added
                flag = 3
            else:
                flag = 2
                break

        if flag == 0:
            req = self.request[:idx_end]
            try:
                self.x[0] = float(req[:idx_com])
                self.x[1] = float(req[idx_com+1:])
                self.ask = 'S'
            except Exception:
                self.x[0] = 0.0
                self.x[1] = 0.0
                self.x[2] = 1100
                self.ask = ''
                # print("Server Error: Receiving wrong values!")
            try:
                self.client.send(str(self.ask))
            except Exception:
                self.ask = ''
                # print("Server Error: Sending 'S' in error!")

        elif flag == 3:
            req = self.request[:idx_end]
            try:
                self.x[2] = float(req[idx_star+1:])
                self.ask = 'S'
            except Exception:
                self.x[2] = 0.0
                self.ask = ''
            try:
                self.client.send(str(self.ask))
            except Exception:
                self.ask = ''

        elif flag == 1:
            self.request = ''
            self.ask = ''
        elif flag == 2:
            req = self.request[:idx_end]

            try:
                # Check if the string includes 'Q', which means actual speeds are
requested
                if req == 'QA':
                    self.ask = str(cur_ang)
                    self.client.send(str(self.ask))
                elif req == 'QS':
                    self.ask = str(act_spd[0]) + ',' + str(act_spd[1])
                    self.client.send(str(self.ask))
                elif req.count('Q'):
                    self.ask = str(runtime) + ',' + str(act_spd[0]) + ',' +
str(act_spd[1]) + ',' + \
                               str(des_spd) + '*' + str(cur_loc) + '*' +
str(cur_ang) + '*' + \
                               str(des_loc) + '*' + str(des_ang)
                    self.client.send(str(self.ask))
                # Check if the string includes 'Q', which means pid parameters are
requested
                elif req.count('D'):
                    self.ask = str(pid_para[0]) + ',' + str(pid_para[1]) + ',' +
str(pid_para[2])
                    self.client.send(str(self.ask))
                # If received 'P', exit
                elif req.count('P'):
                    self.ask = ''
                    return 1
                # Check the first time that receives the heart pack
                elif req.count('H'):
                    self.first_receive = True
                    self.ask = 'K'
                    self.time_0 = datetime.utcnow()
                    self.client.send(str(self.ask))
            except Exception:
                pass

        return 1    # Tcpip Works normally

    def get_request(self):

        return self.request

    def get_ask(self):

        return self.ask

    def get_set_point(self):

        return self.x

    def close(self):

        self.client.close()
        self.server.close()
        print("TCP-IP closed.\n")
        time.sleep(1) #Added sections


####################################################
# This class basically fixes the pwm reading problem
####################################################


class NewSpdCtrl(SpdCtrl):

    def __init__(self, pi, gpio_lft, gpio_rht, dac_lft, dac_rht):
        SpdCtrl.__init__(self, pi, gpio_lft, gpio_rht, dac_lft, dac_rht)

        # Variables
        self.spd_check_pt = [0.0, 0.0]       # Left and right
        self.spd_check_pt_p = [0.0, 0.0]     # _p means previous
        self.tm_check_pt = [0.0, 0.0]
        self.tm_check_pt_p = [0.0, 0.0]       # _p means previous
        self.tm_repeat = [0.0, 0.0]
        self.turns = [0, 0]             # 0 for previous, 1 for recent

        # Flags
        self.counting_end = True  # if true, stop counting repeat time
        self.first_pwm_adjustment_loop = True

    def refresh_speed_controller(self, set_point, recent_speed):

        # PID Parameters
        self.Kpwm = 48.0 / 4096.0
        self.Kp = 0.840  # Initial Kp value is 0.840 (0, 1.8, 0.002)
        self.Ki = 2.02  # Initial Ki value is 2.02  (0, 3.2, 0.002)
        self.Kd = 0.008  # Initial Kd value is 0.008 (0, 0.2, 0.0002)

        # PID Parameters (Left)
        # self.op_lft = 0.0  # controller output
        self.sp_lft = set_point[0]  # Set Point

        self.pv_lft = recent_speed[0]  # process variable

        self.e_lft = 0.0  # error
        self.ie_lft = 0.0  # integral of the error
        self.dpv_lft = 0.0  # derivative of the pv
        self.P_lft = 0.0  # proportional
        self.I_lft = 0.0  # integral
        self.D_lft = 0.0  # derivative
        self.sp_lft = 0.0  # set point

        self.pv_p_lft = recent_speed[0]  # previous value of pv
        self.ie_p_lft = 0.0  # previous value of ie

        # PID Parameters (Right)
        # self.op_rht = 0.0  # controller output
        self.sp_rht = set_point[1]  # Set Point

        self.pv_rht = recent_speed[1]  # process variable

        self.e_rht = 0.0  # error
        self.ie_rht = 0.0  # integral of the error
        self.dpv_rht = 0.0  # derivative of the pv
        self.P_rht = 0.0  # proportional
        self.I_rht = 0.0  # integral
        self.D_rht = 0.0  # derivative
        self.sp_rht = 0.0  # set point

        self.pv_p_rht = recent_speed[1]  # previous value of pv
        self.ie_p_rht = 0.0  # previous value of ie

        # Upper and Lower limits on OP
        self.op_hi = 409.6
        self.op_lo = 0

        # PID Sample Time
        self.pid_smp_tm = 0.001  # 1ms

        # PID loop flags
        self.first_pid_lp = True

        # Initialize DAC
        # self.dac.set_voltage(0)
        # self.dac2.set_voltage(0)

        # NewSpdCtrl variables
        self.spd_check_pt = [0.0, 0.0]  # Left and right
        self.spd_check_pt_p = [0.0, 0.0]  # _p means previous
        self.tm_check_pt = [0.0, 0.0]
        self.tm_check_pt_p = [0.0, 0.0]  # _p means previous
        self.tm_repeat = [0.0, 0.0]
        self.turns = [0, 0]  # 0 for previous, 1 for recent

        # NewSpdCtrl flags
        self.counting_end = True  # if true, stop counting repeat time
        self.first_pwm_adjustment_loop = True

    def fix_pwm_reading(self, sp, speed, tm, lft_or_rht):

        if self.turns[lft_or_rht] == 0:
            self.tm_check_pt_p[lft_or_rht] = tm
            self.spd_check_pt_p[lft_or_rht] = speed
            self.turns[lft_or_rht] = 1
        else:
            self.tm_check_pt[lft_or_rht] = tm
            self.spd_check_pt[lft_or_rht] = speed
            self.turns[lft_or_rht] = 0

        if self.first_pwm_adjustment_loop:
            self.first_pwm_adjustment_loop = False
            self.counting_end = True
        else:
            for i in range(1):
                if self.spd_check_pt[lft_or_rht] !=
self.spd_check_pt_p[lft_or_rht]:
                    self.counting_end = True
                else:
                    tm_r = self.tm_check_pt[lft_or_rht]
                    tm_p = self.tm_check_pt_p[lft_or_rht]
                    self.tm_repeat[lft_or_rht] = self.tm_repeat[lft_or_rht] +
abs(tm_r - tm_p)
                    if self.spd_check_pt[lft_or_rht] == 0:
                        self.counting_end = True
                        break

                    self.counting_end = False

                    # if self.tm_repeat[lft_or_rht] > 0.5 * (1 /
(self.spd_check_pt[lft_or_rht] / 60.0)):
                    if self.tm_repeat[lft_or_rht] > 0.35:
                        #if sp >= 30:  # Check if spd command less than 20 RPM
                        #    self.counting_end = True
                        #else:
                        self.counting_end = True

                        self.refresh_left_pwm_reader()
                        self.refresh_right_pwm_reader()
                        self.refresh_speed_controller([0, 0], [0, 0])

        if self.counting_end:
            self.tm_repeat[lft_or_rht] = 0


####################################################
# This class integrates my motor speed control part
# to Corey's steering control code
####################################################


class Integrate:

    def __init__(self, _pi):

        # GPIOs
        self.THROTTLE = 21  # Microswitch
        self.EMERGENCY = 12  # Emergency Brake Output
        self.REVERSE = 16
        self.PWM_GPIO2 = 19  # CH5 switch
        self.PWM_GPIO3 = 22  # UP-DOWN CHANNEL 3 (LEFT JOYSTICK)
        self.REVERSESIGN = 23  # Needs to be changed because it'll be used for
speed control
        self.BRK1_GPIO = 18
        self.BRK2_GPIO = 17
        self.BRKHALL_GPIO = 20
        self.STRACT1_GPIO = 25  # Steering H-Bridge signal 1
        self.STRACT2_GPIO = 26  # Steering H-Bridge signal 2

        self.SPDCONTROL_LFT = 13  # PWM GPIO for speed control (Right: 23; Left:
13)
        self.SPDCONTROL_RHT = 23  # PWM GPIO for speed control (Right: 23; Left:
13)
        self.ABrake = 20  # Testing brake mod

        # Logging and Debugging
        logging.basicConfig(filename='/mnt/bus/logs/log.txt', level=logging.INFO)

        # Import ADC for Steering Slide Pot
        self.adc = Adafruit_ADS1x15.ADS1015()

        # Variables and Constants
        self.k = 10  # Forward turning constant
        self.kr = 10  # Reverse turning constant
        self.SAMPLE_TIME = 0.01  # Adjust higher to slow down "print" time to
console for debugging
        self.x = 0  # Used for countdown timer to shutdown program
        self.var = 1  # For infinite loop
        self.c = 3  # Shutdown timer length
        self.DS = [0.0, 0.0]    # motor speed set points
        self.DS_p = [0.0, 0.0]  # Previous motor speed set points
        self.tm = 0.0       # Timer
        self.tm_0 = None    # timer beginning stamp
        self.tm_p = 0.0    # Previous time
        self.dt = None      # time period
        self.tm_1 = None    # timer ending stamp
        self.i = 0  # Loop counter
        self.clt_st = 0  # Client Status

        # Flags
        self.time_stamp = False
        self.first_sp_cpr = True  # the first comparison about set point speeds

        # Variables and constants for fixing pwm reading problem
        self.lft_spd = [0.0, 0.0]  # left speed check point for zero speed command
([0] is previous, [1] is now)
        self.rht_spd = [0.0, 0.0]
        self.lft_timeout_check = [0.0, 0.0]  # left motor time check point for
zero speed command timeout
        self.rht_timeout_check = [0.0, 0.0]
        self.lft_tm_repeat = 0.0  # left motor time period that the speeds have
become the same
        self.rht_tm_repeat = 0.0

        # Flags for fixing pwm reading program
        self.spd_check = 0  # Reference for previous or recent speed, relating to
sft_spd and rht_spd

        # PWM values
        self.EBakMid = 1200  # Since the ebrake is a switch this value is the
middle value to deteremine on/off.

        self.BRKMIN = 1330
        self.BRKHALL = 0
        self.BRKHALLSTR = self.BRKMIN  # BRKHALL start value
        self.BRKSTR = 1300
        self.BRKPOS = 0

        self.BrkReset = 160
        self.Brkhall = 160
        self.BrkStop = 0

        # Flag for triggering E-Break
        self.e_break_on = False

        # steering constants for angle-bits conversion
        self.A1 = 135606
        self.A2 = 55800
        self.B = 411.72
        self.K = -0.0621
        self.Theta0 = 73.0

        # steering variables
        self.GAIN = 1
        self.d = 0  # Used as a distance from current to desired
        self.dv = 0  # Distance direction
        self.dpast = 0
        self.PWM = 0
        self.DA = 0  # desired angle
        self.DL = self.__angle_to_bits(self.DA)  # Desired location
        self.PPWM = 0  # Past PWM used for optimization by not having to setting
same PWM value every loop
        self.looping = True
        self.MAX = 1550
        self.MIN = 350
        self.BOOSTCOUNTER = 100
        self.PWMBOOST = 130  # Used to get stuck actuator moving BUT IS ALSO
MINIMUM USABLE PWM

        # TCP Socket ip and port
        self.ip = "192.168.1.2"
        self.port = 4869

        # PI
        self.pi = _pi

        # DAC config
        self.dac = Adafruit_MCP4725.MCP4725(address=0x62, busnum=1)  # i2c address
(address = 0x62) left motor
        self.dac2 = Adafruit_MCP4725.MCP4725(address=0x63, busnum=1)  # right
motor
        self.dac3 = Adafruit_MCP4725.MCP4725(address=0x60, busnum=1)  # Variable
Regen Braking

        # PWM Readers
        self.p2 = reader(self.pi, self.PWM_GPIO2)  # E-Brake
        # p3 = reader(pi, PWM_GPIO3)  # Throttle

        # GPIO initialize
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(self.EMERGENCY, GPIO.OUT)
        GPIO.output(self.EMERGENCY, GPIO.LOW)
        GPIO.setup(self.REVERSE, GPIO.OUT)
        GPIO.output(self.REVERSE, GPIO.LOW)
        GPIO.setup(self.REVERSESIGN, GPIO.OUT)
        GPIO.output(self.REVERSESIGN, GPIO.LOW)
        GPIO.setup(self.THROTTLE, GPIO.OUT)
        GPIO.setup(self.ABrake, GPIO.OUT)
        GPIO.output(self.ABrake, GPIO.LOW)

        GPIO.setup(self.BRK1_GPIO, GPIO.OUT)
        GPIO.setup(self.BRK2_GPIO, GPIO.OUT)

        GPIO.setup(self.STRACT1_GPIO, GPIO.OUT)  # Sets up pin 25
        GPIO.output(self.STRACT1_GPIO, GPIO.LOW)  # Sets pin 25 to LOW
        GPIO.setup(self.STRACT2_GPIO, GPIO.OUT)  # Sets up pin 26
        GPIO.output(self.STRACT2_GPIO, GPIO.LOW)  # Sets pin 26 to LOW

        # DAC initialize
        self.dac.set_voltage(0)  # Start DAC's at 0V and wait 3 seconds
        self.dac2.set_voltage(0)
        self.dac3.set_voltage(0)
        GPIO.output(self.REVERSESIGN, GPIO.LOW)
        sleep(2)

        # Add TRIM center code.
        GPIO.output(self.THROTTLE, GPIO.HIGH)  # Starts in forward
        GPIO.output(self.REVERSE, GPIO.LOW)

        # steering initialize
        self.pi.set_PWM_dutycycle(self.STRACT1_GPIO, 0)
        self.pi.set_PWM_dutycycle(self.STRACT2_GPIO, 0)
        self.adc.start_adc(0, gain=self.GAIN)
        self.CurrentLocation = self.adc.get_last_result()
        self.CurrentAngle = self.__bits_to_angle(self.CurrentLocation)
        print("Steering location starting at {}".format(self.CurrentLocation))
        print("Front wheel angle starting at {}".format(self.CurrentAngle))
        logging.info("PROGRAM RESTARTED! Starting Location =
{}".format(self.CurrentLocation))

        # thread list
        self.threads = []
        self.thread_steering_controller = None
        self.thread_motor_speed_controller = None
        self.thread_tcp_communication = None

        # flag for stopping all threads
        self.stop_all_threads = False

        # others
        self.tcp = None
        self.spd = None
        self.pid_para = None

    def init_controller(self):

        GPIO.output(self.BRK1_GPIO, GPIO.HIGH)
        GPIO.output(self.BRK2_GPIO, GPIO.LOW)
        print("Extending Actuator")
        time.sleep(3)
        GPIO.output(self.BRK1_GPIO, GPIO.LOW)
        GPIO.output(self.BRK2_GPIO, GPIO.LOW)

        print("Centralizing steering actuator")
        self.__steering_control_unit()
        self.CurrentLocation = self.adc.get_last_result()
        self.CurrentAngle = self.__bits_to_angle(self.CurrentLocation)
        print("Now steering location is at {}".format(self.CurrentLocation))
        print("Now well angle is at {}".format(self.CurrentAngle))

        self.tcp = Tcpip(self.ip, self.port)
        self.spd = NewSpdCtrl(self.pi, self.SPDCONTROL_LFT, self.SPDCONTROL_RHT,
self.dac, self.dac2)
        self.pid_para = self.spd.get_pid_parameters()
        print("\n")
        print('Kp = {}'.format(self.pid_para[0]))
        print('Ki = {}'.format(self.pid_para[1]))
        print('Kd = {}'.format(self.pid_para[2]))

    def start_timer(self):

        self.tm = 0
        self.tm_0 = datetime.utcnow()
    def __steering_control_unit(self):

        self.CurrentLocation = self.adc.get_last_result()
        self.CurrentAngle = self.__bits_to_angle(self.CurrentLocation)
        # self.DL = self.set_point[2]
        # print("DL={}".format(self.DL))
        self.looping = True
        self.PWMBOOST = 130  # This reset the PWMBOOST value everytime so they our
minimums stay consistent
        # print("Currently at {}".format(self.CurrentLocation))
        logging.info("Starting Location = {}".format(self.CurrentLocation))
        if self.DL > self.MAX:
            self.DL = self.MAX
            # print("Maximum = {}".format(self.MAX))
        if self.DL < self.MIN:
            self.DL = self.MIN
            # print("Minimum = {}".format(self.MIN))
        logging.info("Desired Location = {}".format(self.DL))

        while self.looping == True:

            try:
                self.CurrentLocation = self.adc.get_last_result()
                self.CurrentAngle = self.__bits_to_angle(self.CurrentLocation)
            except Exception:
                pass
            self.dv = (self.CurrentLocation - self.DL)
            self.d = abs(self.dv)
            if self.d > 150:
                self.PWM = 250
            if 150 > self.d > 50:
                self.PWM = int((6 / 5) * self.d + 70)
            if self.d < 50:
                self.PWM = self.PWMBOOST
                if abs(self.d - self.dpast) < 5:
                    self.BOOSTCOUNTER = self.BOOSTCOUNTER - 1
                    if self.BOOSTCOUNTER == 0:
                        self.PWMBOOST = self.PWMBOOST + 5
                        self.BOOSTCOUNTER = 100
                        # print("Rasing PWM since actuator isn't moving!")
            if self.PWM != self.PPWM:
                if self.dv < 0:
                    self.pi.set_PWM_dutycycle(self.STRACT1_GPIO, self.PWM)
                    self.pi.set_PWM_dutycycle(self.STRACT2_GPIO, 0)
                if self.dv > 0:
                    self.pi.set_PWM_dutycycle(self.STRACT2_GPIO, self.PWM)
                    self.pi.set_PWM_dutycycle(self.STRACT1_GPIO, 0)
                self.PPWM = self.PWM
            if self.d < 1:
                self.pi.set_PWM_dutycycle(self.STRACT1_GPIO, 0)
                self.pi.set_PWM_dutycycle(self.STRACT2_GPIO, 0)
                self.looping = False
            self.dpast = self.d
        # print("d={}".format(self.d))
        # print("Ending at {}".format(self.CurrentLocation))
        logging.info("Ending Location = {}".format(self.CurrentLocation))

    def steering_control(self):
        # Y = -.0621x+421.72
        # Min = 400bits = 15 5/8in * 25.4 mm
        # Max = 1550 = 12 13/16in

        while True:

            self.__steering_control_unit()

            if self.stop_all_threads:
                break

            sleep(0.005)

    def speed_control(self):

        while True:

            while self.e_break_on:
                sleep(0.001)

            if self.first_sp_cpr:
                self.first_sp_cpr = False
            else:
                if (self.DS[0] != self.DS_p[0]) or (self.DS[1] != self.DS_p[1]):
                    self.spd.refresh_speed_controller(
                        self.DS,
                        [
                            self.spd.read_left_motor_speed(),
                            self.spd.read_right_motor_speed()
                        ]
                    )
                    # self.first_sp_cpr = True

            self.spd.pid_control([self.DS[0], self.DS[1]])

            self.spd.fix_pwm_reading(self.DS[0], self.spd.read_left_motor_speed(),
self.tm, 0)
            self.spd.fix_pwm_reading(self.DS[1],
self.spd.read_right_motor_speed(), self.tm, 1)

            self.DS_p[0] = self.DS[0]
            self.DS_p[1] = self.DS[1]

            if self.stop_all_threads:
                break

            sleep(0.001)

    def start_e_brake(self):

        while True:

            while self.p2.pulse_width() > self.EBakMid:

                self.e_break_on = True

                # Electrical Breaking
                GPIO.output(self.EMERGENCY, GPIO.HIGH)  # E-Brake ON

                # Physical Braking
                while self.Brkhall > self.BrkStop:
                    GPIO.output(self.BRK1_GPIO, GPIO.LOW)
                    GPIO.output(self.BRK2_GPIO, GPIO.HIGH)
                    self.Brkhall = self.Brkhall - 1
                    sleep(0.01)
                    print("HALL {}".format(self.Brkhall))
                    self.time_stamp = True

                GPIO.output(self.BRK1_GPIO, GPIO.LOW)
                GPIO.output(self.BRK2_GPIO, GPIO.LOW)

                # self.DA = 0
                # self.DL = self.__angle_to_bits(self.DA)
                self.spd.pid_control([0, 0])

            while self.Brkhall < self.BrkReset:
                GPIO.output(self.BRK1_GPIO, GPIO.HIGH)
                GPIO.output(self.BRK2_GPIO, GPIO.LOW)
                self.Brkhall = self.Brkhall + 1
                sleep(0.01)
                print("hall{}".format(self.Brkhall))

            GPIO.output(self.BRK1_GPIO, GPIO.LOW)
            GPIO.output(self.BRK2_GPIO, GPIO.LOW)

            GPIO.output(self.EMERGENCY, GPIO.LOW)  # E-Brake OFF

            self.e_break_on = False

            if self.stop_all_threads:
                break

    def tcp_communication(self):

        while True:

            self.clt_st = self.tcp.data_trans(
                float(self.tm_p),
                [
                    self.spd.read_left_motor_speed(),
                    self.spd.read_right_motor_speed(),
                    self.DS[0],
                    self.spd.get_pid_parameters()[0],
                    self.spd.get_pid_parameters()[1],
                    self.spd.get_pid_parameters()[2]
                ],
                [
                    self.CurrentLocation,
                    self.CurrentAngle,
                    self.DL,
                    self.DA
                ]
            )

            # DS, DA are desired values obtained from laptop
            self.DS = [self.tcp.get_set_point()[0], self.tcp.get_set_point()[1]]
            self.DA = self.tcp.get_set_point()[2]

            # Dl is converted form DA
            self.DL = self.__angle_to_bits(self.DA)

            if self.clt_st == -1 or self.stop_all_threads:
                self.stop_all_threads = True
                break
            sleep(0.01)

    def __angle_to_bits(self, angle):

        # -411.72
        # bits = (math.sqrt(135606 - 55800 * math.cos(math.pi * (angle + 73.0) / 
180.0)) - 411.72) / -0.0621
        bits = (math.sqrt(self.A1 - self.A2 * math.cos(math.pi * (-angle + 
self.Theta0) / 180.0)) - self.B) / self.K
        return bits

    def __bits_to_angle(self, bits):

        # angle = int((math.acos((135606 - math.pow((411.72 - 0.0621 * bits), 2)) 
/ 55800) * (180.0/math.pi)) - 73.0)
        angle = -1*int(
            (math.acos((self.A1 - math.pow((self.B + self.K * bits), 2)) / 
self.A2) * (180.0 / math.pi)) - self.Theta0
        )
        return angle

    def run_timer(self):

        print(
                "%f\t%f\t%f\t%d\t%f\t%f\t%f\t%f\t%d\t%f\t%d" %
                (
                    self.tm,
                    self.spd.read_left_motor_speed(),
                    self.spd.read_right_motor_speed(),
                    self.DS[0],
                    self.spd.get_pid_parameters()[0],
                    self.spd.get_pid_parameters()[1],
                    self.spd.get_pid_parameters()[2],
                    self.CurrentLocation,
                    self.CurrentAngle,
                    self.DL,
                    self.DA
                )
        )
        self.tm_1 = datetime.utcnow()
        self.dt = self.tm_1 - self.tm_0
        self.tm_p = self.tm
        self.tm = self.dt.seconds + self.dt.microseconds * 1.0e-6

    def get_flag_stop_all_threads(self):

        return self.stop_all_threads

    def close(self):
        self.stop_all_threads = True
        self.dac.set_voltage(0)  # Start DAC's at 0V and wait 3 seconds
        self.dac2.set_voltage(0)
        self.pi.set_PWM_dutycycle(25, 0)
        self.pi.set_PWM_dutycycle(26, 0)
        self.spd.close()
        self.tcp.close()
        GPIO.output(self.EMERGENCY, GPIO.HIGH)
        GPIO.output(self.ABrake, GPIO.HIGH)
        while self.x < self.c:
            print("PiBus is stopping in, {} seconds".format(self.c - self.x))
            self.x = self.x + 1
            sleep(1)
        GPIO.cleanup()


####################################################
# Main subroutine
####################################################


if __name__ == "__main__":

    pi = pigpio.pi()

    while True:

        integ = Integrate(pi)

        try:

            integ.init_controller()

            threads = []

            thread_motor_speed_controller =
threading.Thread(target=integ.speed_control)
            threads.append(thread_motor_speed_controller)

            thread_steering_controller =
threading.Thread(target=integ.steering_control)
            threads.append(thread_steering_controller)

            thread_tcp_communication =
threading.Thread(target=integ.tcp_communication)
            threads.append(thread_tcp_communication)

            thread_e_break = threading.Thread(target=integ.start_e_brake)
            threads.append(thread_e_break)

            for t in threads:
                t.setDaemon(True)
                t.start()

            integ.start_timer()

            while True:                     # Infinite loop
```

```
##########################################################################
                integ.run_timer()
                if integ.get_flag_stop_all_threads():
                    break
                sleep(0.02)
##########################################################################
            integ.close()
            del integ.tcp
            del integ.spd
            del integ

        except KeyboardInterrupt:

            integ.close()
            pi.stop()
            sys.exit()

            # p.cancel()
```

10.1.4 TCP_client_v1_24.py

This program is run on the laptop that will be connected to the follower module. As of this moment, the program connects to the follower module Raspberry Pi and sends a final speed for the follower module's motors to accelerate or decelerate to. An interrupt can be applied at any time to disconnect from the follower's Raspberry Pi.

TCP_client_v1_24.py

```
"""
TCP_client_v1_2.py
by Chixiang Zhang
9/22/17
"""

import socket
import time
import sys

"""
IMPORTANT:
Do NOT put this program into the RPi. It should be implemented directly on PC/MAC.
Excecute this program right after TCP_server.py starts running.

This program is a TEST CODE for the client part of the TCP/IP communication, which
realizes the half-duplex communication between RPi (Server) and PC/MAC (Client).
In the test code, firstly a counting number which starts from 1000 (increments by
1)
is sent from client to server. The server recieves that number, then responds a
```

```
counting number starting from 0 (also increments by 1) to the client, which
recieves
that number correspondingly.

Sources/Sample Code (NOT written in English, but it's useful to take a look at the
sample code):
[1] general way to do TCP/IP  -
http://blog.sina.com.cn/s/blog_864b79ca0102w795.html
[2] half-duplex communication - http://www.itnose.net/detail/6359275.html
"""


TCP_CLIENT_PAUSE = 0.5
TCP_CLIENT_TRANS_PAUSE = 0.01
#SERVER_OFF = '0'
#SERVER_ON = '1'

#server_status = SERVER_OFF

#RPi's IP
#SERVER_IP = "10.100.51.227"
SERVER_IP = "192.168.1.2"
SERVER_PORT = 4869
#SERVER_IP = '10.0.0.111'
#SERVER_PORT = 5555
#print("Starting socket: TCP...")
time.sleep(TCP_CLIENT_PAUSE)
server_addr = (SERVER_IP, SERVER_PORT)

#creat socket object for client
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# building connection
while True:
    try:
        print("Connecting to server @ %s:%d..." %(SERVER_IP, SERVER_PORT))
        time.sleep(TCP_CLIENT_PAUSE)
        client.connect(server_addr)
        print("Successfully connected!\n")
        break
    except Exception:
        print("Can't connect to server,try it latter!")
        time.sleep(TCP_CLIENT_PAUSE)
        continue

# send
count = '100,100#'
#while True:
for i in range(0,100):
    ask = "100" + ",100#"
    client.send(str(ask))
    #print("Sending %s to server" %str(ask))
    #count = count + 1
    # recieve
    response = ''
    while len(response) == 0:
        response = client.recv(128)
    print("%s\n" %str(response))
    ask = "H#"
    client.send(str(ask))
    #print("Sending %s to server" %str(ask))
    #count = count + 1
    # recieve
    print('a')
    response = ''
    while len(response) == 0:
        response = client.recv(128)
    #print("%s\n" %str(response))
    print('d')

    for j in range(1):
        time.sleep(0.1)
        ask = 'Q#'
        client.send(str(ask))
        response = ''
        while len(response) == 0:
            response = client.recv(128)
        print("%s\n" %str(response))

        print('c')

        ask = 'H#'
        client.send(str(ask))
        print('b')
        response = ''
        while len(response) == 0:
            response = client.recv(128)
        #print("%s\n" %str(response))

for k in range(0,500):
    time.sleep(0.1)
    ask = '-10,-10#'
    client.send(str(ask))

    response = ''
    while len(response) == 0:
        response = client.recv(128)
    print("%s\n" %str(response))

    ask = "H#"
    client.send(str(ask))

    response = ''
    while len(response) == 0:
        response = client.recv(128)

#for i in range(0,200):
#    client.send('P#')
client.close()

'''
    try:
        for i in range(0,10):
            ask = str(i*10+10) + ",80#"
            client.send(str(ask))
            #print("Sending %s to server" %str(ask))
            #count = count + 1
            # recieve
            response = client.recv(128)
            print("%s\n" %str(response))
            time.sleep(3)
            client.send('Q#')
            response = client.recv(128)
            print("%s\n" %str(response))
    except Exception:
        client.close()
        print("Exception: client closed.")
        break
'''
```

10.2 Appendix B—High-Level C++ Program

10.2.1 Summary

The high-level C++ program was developed to implement vehicle autonomous tracking. It includes a computer vision module, a laser sensor module, a BP neural network model and a communication module. The program acquires the visual and distance information from the camera and laser sensor, then employs a trained neural network controller to calculate the desired translational and rotational velocities of the follower vehicle. Then, the control commands are sent to the low-level PID controllers via TCP/IP communication. Meanwhile, the current velocities of the vehicle could be sent back to the high-level C++ program via TCP/IP communication.

10.2.1.1 Main.cpp

This is the main program and the beginning point of the whole program.

10.2.1.2 Robot_vision.cpp and robot_vision.h

The computer vision module.

10.2.1.3 Robot_laser.cpp and robot_laser.h

The laser measurement module.

10.2.1.4 Robot_bp.cpp and robot_bp.h

The neural network module

10.2.2 Main.cpp

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <cmath>
#include "robot_util.h"
#include "robot_laser.h"
#include "robot_vision.h"
//#include "robot_bp.h"
```

```
#include "stdafx.h"
#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <sstream>
#include <csignal>
// Need to link with Ws2_32.lib, Mswsock.lib, and Advapi32.lib
#pragma comment (lib, "Ws2_32.lib")
#pragma comment (lib, "Mswsock.lib")
#pragma comment (lib, "AdvApi32.lib")

using namespace std;

#define DEFAULT_BUFLEN 512


double vTrans = 0; // The vehicle's translational velocity, unit: cm/s
double vRot = 0;   // The vehicles' rotational velocity, unit: degree/s

const int TOTAL_EXECUTION_TIMES = 50000;

const int TOO_CLOSE = 300; // unit:mm
const int SUPER_CLOSE = 200;
const int LASER_FAR = 2000;
const int LASER_NEAR = 1000;

const int MAX_ROT_VEL = 15;  // deg/s
const int MIN_ROT_VEL = -15;
const int MAX_VEL = 100;  // cm/s
const int MIN_VEL = -100;


// gains
const double K_FORWARD_VEL = 0.05;
const double K_BACKWARD_VEL = 40000; // Change to larger if wanting to back up
faster (40000 for line)
const double K_LEFT_ROT_VEL = -0.02;
const double K_RIGHT_ROT_VEL = -0.02;
const double K_ADJUST_ROT_VEL = 0.98;

const int VISION_WIDTH = RobotVision::VISION_WIDTH;
const int VISION_HEIGHT = RobotVision::VISION_HEIGHT;
const int VISION_LOW = 0.27 * VISION_WIDTH; // 346;
const int VISION_HIGH = 0.72 * VISION_WIDTH; // 921;



const int HARD_SLEEP_TIME = 1;  // ms or s?

// Tcp/IP communication setting
SOCKET ConnectSocket=INVALID_SOCKET;
char* IPaddress = "192.168.1.2";
```

```
char* port = "4869";

// function delaration
void readMatrix(char *fileName, double * myMatrix, int rows, int cols);
void neuralNetwork(double distance, double imageX, int
left_or_right_by_laser_angle, int * output);
SOCKET createSocket(char* address, char* port); //creat a socket
int send(SOCKET ConnectSocket, string stringbuf); // send a string through the
socket
string queryData(SOCKET ConnectSocket); // receive a string through the socket
void sendDesiredVehicleSpeed(double transVel, double rotVel); // transVel: cm/s,
rotVel: deg/s
void getCurrentVehicleSpeed(double *pTrans, double *pRot);
double getVel();
double getRotVel();
void getWheelSpeed(double *pw1, double *pw2);// Unit: deg/s
void setVel(double v);
void setRotVel(double w);
void stopVehicle();
void sig_handler(int sig);
void testVehicleMotion();
void testWheelSpeed();

// NN Architecture
const int INPUT_LAYER_NUM = 3;
const int HIDDEN_LAYER_NUM = 50;
const int OUTPUT_LAYER_NUM = 5;
double w[INPUT_LAYER_NUM][HIDDEN_LAYER_NUM];
double v[HIDDEN_LAYER_NUM][OUTPUT_LAYER_NUM];

int main(int argc, char **argv) {


    // create the communication channel
    ConnectSocket = createSocket(IPaddress, port);

   // Test communication with the low-level controller.
    //testVehicleMotion();
  //  testWheelSpeed();
  //  return 1;

    //////////////////////////////////////////////////////////////////////////
    cv::ocl::setUseOpenCL(false);

    //load the weights of the neural network
    readMatrix("w_n.txt", &w[0][0], INPUT_LAYER_NUM, HIDDEN_LAYER_NUM);
    readMatrix("v_n.txt", &v[0][0], HIDDEN_LAYER_NUM, OUTPUT_LAYER_NUM);

       ////////////////////////////
       RobotLaser laser;
       RobotVision vision;

    // Initialize laser before depth camera, otherwise it won't be able to
initialize
    laser.init();

    // Initialize vision including camera and cascade detector
    if (!vision.init("cascade_stop_sign.xml")) {
        cout << "error: there was an error when initializing vision component." << 
endl;
        return 1;
    }

       // Initilize the vision kalman filter
       vision.init_kalman_filter();

       // Laser, Vision, BP variables
       double laser_distance = 0;
       double laser_angle_in_degree = 0;
       double laser_start_angle = -15;
       double laser_end_angle = 15;
       double vision_center_x = 0;
       double rot_vel_increment = 0;
       double vel_increment = 0;
       int bp_output[OUTPUT_LAYER_NUM];
       bool is_predicted = false;
       int predicted_counter = 0;
       int predicted_max = 10;
       double temp_vision_area = 0;
       double temp_vision_center_x = 0;
       double temp_vision_center_y = 0;

       // Initialize laser kalman filter
       laser.init_kalman_filter(laser_start_angle, laser_end_angle, 
&laser_angle_in_degree);

       // Accessory variables
       int i = 0;
       double tmp = 0;

       // Initial 0 speed
       vTrans=0;
    vRot = 0;

    cout << "\n\n\n ---------------------------------------\n";
    cout << "Start the loop now !!!!!!!!\n\n" << endl;

       while (i < TOTAL_EXECUTION_TIMES) {
        signal(SIGINT, sig_handler);
              i++;

       if (false == laser.is_open()) {
                     laser.init();
              }

        //get the distance
              laser_distance = laser.get_distance(laser_start_angle, 
laser_end_angle, &laser_angle_in_degree);
        cout << "Laser distance= " << laser_distance << " at angle of"<< 
laser_angle_in_degree <<endl;

        //laser_distance = laser.get_distance_on_thread(laser_start_angle, 
laser_end_angle, &laser_angle_in_degree);
              if (laser_distance <0) {
            Sleep(HARD_SLEEP_TIME);
                     continue;
              }

              if (laser_distance < TOO_CLOSE && laser_distance > SUPER_CLOSE) {
                     stopVehicle();
                     cout << "The distance is too short, the vehicle is stopped! "
<< endl;
                     continue;
              }
              else if (laser_distance < SUPER_CLOSE)
        {
                     if (getVel() >= 0) {
                            //setVel(-100);  // the vehcile moves backward
                stopVehicle();
                     }
                     cout << "The distance is super close, need to stop the
vehicle." << endl;
            continue;
              }

              // get the vision position

              vision_center_x =
vision.detect_with_cascade_on_thread(laser_angle_in_degree, &is_predicted);
           cout << "vision_center=" << vision_center_x << " is_predicted= " <<
is_predicted << endl;

       /*     if (is_predicted) {
                     predicted_counter++;
              }

        // if the visual position is predicted for too many times
              int look_for_rot_vel = 5; // the rot speed used to search the leader
vehicle
              if (predicted_counter > predicted_max) { // the visual object is
lost, so need to rotate to find it.
            cout << "Stop sign is lost, rotate the vehicle to find it" << endl;
            stopVehicle();
                     while
(!vision.detect_wtih_cascade_inline(&temp_vision_center_x, &temp_vision_area,
&temp_vision_center_y)) {
                            if (laser_angle_in_degree > 0) {
                                   // laser senses the object is on the left
                                   setRotVel(look_for_rot_vel);
                            }
                            else
                {
                                   // laser senses the object is on the right
                                   setRotVel(-look_for_rot_vel);
                            }
                            Sleep(50);
                cout << "rotating to find the visual target. \n";
                     }
            cout << "Found the visual target. \n";

            // once find the visual target, stop rotating and update the Kalman
filter
                     vision.correct_kalman_filter(temp_vision_center_x,
temp_vision_center_y);
```

```
                    predicted_counter = 0;
                    continue;
              } */

        // New factor, laser_angle_in_degree should be within -120 to 120, make it
range into 0 to 1
        int left_or_right_by_laser_angle = laser_angle_in_degree > 0 ? 0 : 1; //
laser_angle_in_degree / 240 + 0.5;

              // OUTPUT (neuralNetwork): bp_output
        cout << "NN INPUTs (d,x, theta):" << laser_distance << ", " <<
vision_center_x << ", " << left_or_right_by_laser_angle << endl;
              neuralNetwork(laser_distance, vision_center_x,
left_or_right_by_laser_angle, bp_output);
        cout << "NN outputs: " << "  [0]=" << bp_output[0] << "  [1]=" <<
bp_output[1] << "  [2]=" << bp_output[2] << "  [3]=" << bp_output[3] << "  [4]="
<< bp_output[4] << endl;

        // determine rot_vel_increment
              if (vision_center_x <= VISION_LOW) {
                    rot_vel_increment = (vision_center_x - VISION_LOW) *
K_LEFT_ROT_VEL;
              }
              else if (vision_center_x > VISION_LOW && vision_center_x <=
VISION_HIGH) {
                    rot_vel_increment = 0;
              }
              else {
                    rot_vel_increment = -1.0 * (vision_center_x - VISION_HIGH) *
K_RIGHT_ROT_VEL;
              }

        // determine the vel_increment
              double cur_vel = getVel();
              if (cur_vel > 0) {
                    vel_increment = abs(laser_distance) * K_FORWARD_VEL;
              }
              else {
                    vel_increment = K_BACKWARD_VEL / abs(laser_distance);
              }



        bool is_rotating = false;

              //execute the actions
              if (bp_output[0] == 1) {

            // Turn left immediately instead of slowly
        /*    tmp = getRotVel() + rot_vel_increment;
                    if (tmp > MAX_ROT_VEL) {
                          tmp = MAX_ROT_VEL;
                    }

            setRotVel(tmp); // turn left
                    cout << "NN OUTPUT(Action): Turn Left: " << " rotVel= " << tmp
<< endl;
            is_rotating = true; */
```

```
              }

              if (bp_output[1] == 1) {

            // Turn right immediately instead of slowly
       /*     tmp = getRotVel() - rot_vel_increment;
                     if (tmp < MIN_ROT_VEL) {
                            tmp = MIN_ROT_VEL;
                     }

                     setRotVel(tmp); //turn right
                     cout << "NN OUTPUT(Action): Turn Right: " << " rotVel= " << 
tmp << endl;
            is_rotating = true; */
              }

  /*      if (bp_output[4] == 1) {
            cout << "bp_output[4] == 1" << endl;
            if (is_rotating) {
                // reduce rotVel by factor K
                tmp = tmp * K_ADJUST_ROT_VEL;
                setRotVel(tmp); // adjust rotation
                cout << "NN OUTPUT(Action): Adjust Rotation Speed by " << 
K_ADJUST_ROT_VEL << ": " << " rotVel= " << tmp << endl;
            }
        } */

              if (bp_output[0] == 0 && bp_output[1] == 0) {
                     setRotVel(0);
                     cout << "NN OUTPUT(Action): moveing linearly, no rotation! 
Vel: " << getVel() << endl;
              }

              if (bp_output[2] == 1) {

                     tmp = getVel();
                     if (tmp > 0) {
                            tmp += vel_increment;
                     } else 
            {
                            tmp = vel_increment;   // ???????????
                     }

                     if (tmp > MAX_VEL) {
                            tmp = MAX_VEL;
                     }

                     setVel(tmp); //increase speed
            Sleep(100);
                     cout << "NN OUTPUT(Action): Increase linear speed. Old Vel=" 
<< getVel() << ", New Vel="<< tmp <<endl;
              }

              if (bp_output[3] == 1) {

                     tmp = getVel();
                     if (tmp < 0) {
                            tmp -= vel_increment;
```

```
                    }
                    else {
                           tmp = -vel_increment;
                    }

                    if (tmp < MIN_VEL) {
                           tmp = MIN_VEL;
                    }

                    setVel(tmp); //decrease speed
             Sleep(1000);
                    cout << "NN OUTPUT(Action): Decrease linear speed. Old Vel="
<< getVel() << ", New Vel=" << tmp << endl;
              }

        Sleep(HARD_SLEEP_TIME);
        cout << "---------------------------------------\n\n\n";

       }


       // Close Laser
       laser.close();

    //close the communication and delete the socket
    string end = "P#";
    send(ConnectSocket, end);
    shutdown(ConnectSocket, SD_SEND);
    closesocket(ConnectSocket);

       return 0;
}

void readMatrix(char *fileName, double * myMatrix, int height, int width) {
       std::ifstream file(fileName);
       int count = 0;
       for (int i = 0; i < height; i++) {
              for (int j = 0; j < width; j++) {
                     file >> *(myMatrix + count);
                     count = count + 1;
              }
       }
       file.close();
}

void neuralNetwork(double distance, double imageX, int
left_or_right_by_laser_angle, int *output) {
       double x[INPUT_LAYER_NUM];  //input units;
       double H[HIDDEN_LAYER_NUM]; //hidden units

       //data pre-processing
       if (distance > LASER_FAR) {
        x[0] = 2; // +(distance - LASER_FAR) / distance - 0.5;
       }
       else if (distance <= LASER_FAR && distance > LASER_NEAR) {
        x[0] = 1; // +(distance - LASER_NEAR) / (LASER_FAR - LASER_NEAR) - 0.5;
       }
       else {
        x[0] = 0; // +(distance - 0) / LASER_NEAR - 0.5;
       }

       if (imageX >= VISION_HIGH) {
        x[1] = 2; // +(imageX - VISION_HIGH) / imageX - 0.5;
       }
       else if (imageX < VISION_HIGH && imageX > VISION_LOW) {
        x[1] = 1; // +(imageX - VISION_LOW) / (VISION_HIGH - VISION_LOW) - 0.5;
       }
       else {
        x[1] = 0; // +(imageX - 0) / VISION_LOW - 0.5;
       }

    // left or right by laser_angle, already make it between 0 and 1
    x[2] = left_or_right_by_laser_angle;

       //display status
       cout << "Processed INPUT(NN): distance=" << x[0] << "   " << ", image
location=" << x[1] << ", left_or_right_by_laser_angle=" << x[2] <<  "\n";

       // calculate the outputs of the hidden units
       for (int i = 0; i < HIDDEN_LAYER_NUM; i++) {
              double sum = 0;
              for (int j = 0; j < INPUT_LAYER_NUM; j++) {
                     sum = sum + x[j] * w[j][i];
              }
              H[i] = 1 / (1 + exp(0 - sum));
       }

       //caculate the outputs of the output layer
       for (int i = 0; i < OUTPUT_LAYER_NUM; i++) {
              double sum = 0;
              for (int j = 0; j < HIDDEN_LAYER_NUM; j++) {
                     sum = sum + H[j] * v[j][i];
              }
              *(output + i) = (int)round(sum);
       }
}



SOCKET createSocket(char* address, char* port)
{
    SOCKET ConnectSocket = INVALID_SOCKET;
    WSADATA wsaData;
    struct addrinfo *result = NULL,
        *ptr = NULL,
        hints;
    // Initialize Winsock
    int iResult;
    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        printf("WSAStartup failed with error: %d\n", iResult);
        return INVALID_SOCKET;
    }

    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;

    // Resolve the server address and port
    iResult = getaddrinfo(address, port, &hints, &result);
    if (iResult != 0) {
        printf("getaddrinfo failed with error: %d\n", iResult);
        WSACleanup();
        return INVALID_SOCKET;
    }

    // Attempt to connect to an address until one succeeds
    for (ptr = result; ptr != NULL; ptr = ptr->ai_next) {

        // Create a SOCKET for connecting to server
        ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype,
            ptr->ai_protocol);
        if (ConnectSocket == INVALID_SOCKET) {
            printf("socket failed with error: %ld\n", WSAGetLastError());
            WSACleanup();
            return INVALID_SOCKET;
        }

        // Connect to server.
        iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);
        if (iResult == SOCKET_ERROR) {
            closesocket(ConnectSocket);
            ConnectSocket = INVALID_SOCKET;
            continue;
        }
        break;
    }
    return ConnectSocket;
}

int send(SOCKET ConnectSocket, string stringbuf)  // send a string
{
    int iResult;
    char * sendbuf = new char[stringbuf.length() + 1];
    strcpy_s(sendbuf, stringbuf.length() + 1, stringbuf.c_str());
    // Send an initial buffer
    char r[1] = "";
    iResult = send(ConnectSocket, sendbuf, (int)strlen(sendbuf), 0);
    if (iResult == SOCKET_ERROR) {
        printf("send failed with error: %d\n", WSAGetLastError());
        closesocket(ConnectSocket);
        WSACleanup();
        return 0;
    }
    //printf("Bytes Sent: %ld\n", iResult);
    //send by send will cause failing to the second send,
    //so have to insert a "receive" between 2 "send" in clinet
    iResult = 0;
    //while (1)
    //{
    // if (r[0] == '1')
    //       break;
    // recv(ConnectSocket, r, 1, 0);
    //}
    //
    recv(ConnectSocket, r, 1, 0);
    if (r[0] == 'S')
    {
        iResult = 1;
       // printf("send successfully\n");
    }
    else if(r[0] == 'K')
        iResult = 1;
    else {
        printf("server don't receive\n");
    }

    return iResult;
}

string queryData(SOCKET ConnectSocket)  // receive a string
{

    int iResult;
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;
    char *s = "Q#";
    string str;
    iResult = send(ConnectSocket, s, 2, 0);
    //cout << " query iResult " << iResult << endl;
    if (iResult == SOCKET_ERROR) {
        printf("send failed with error: %d\n", WSAGetLastError());
        closesocket(ConnectSocket);
        WSACleanup();
        return str;
    }
    // Receive until the peer closes the connection
    do {
        //cout << "waiting receive queryData  " << endl;
        iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);
        //cout << iResult << endl;
        if (iResult > 0) {
            //printf("Bytes received: %d\n", iResult);
            str = recvbuf;
            break;
        }
        else if (iResult == 0)
            printf("Connection closed\n");
        else
            printf("recv failed with error: %d\n", WSAGetLastError());

    } while (iResult > 0);
    return str;
}

void setWheelSpeed(double w1, double w2) {  //deg/s

  //  cout << "Dessied W1(deg/s)=" << w1 << "  Desired W2=" << w2 << endl;
 //   cout << "----------------------------------------------------------------\n"
<< endl;
```

```
    //convert from deg/s to RPM
    w1 = w1 / 6;
    w2 = w2 / 6;

    // begin to send w1 and w2
    string sendbuf;
    sendbuf = to_string(w1) + "," + to_string(w2) + "#";
    send(ConnectSocket, sendbuf);

    string heartBag = "H#";
    send(ConnectSocket, heartBag);
    return;
}

void sendDesiredVehicleSpeed(double transVel, double rotVel)  // units: cm/s,
deg/s
{
    //the receiving in utf8 will case messed up when show on console
    //however, it is only a display issue.
    //system("chcp 65001");

    cout << "\n\n------------------------------------------------------------" <<
endl;
    cout << "Desired V=" << transVel << " desired W=" << rotVel << endl;

    //convert v and w into w1 and w2
    // unit of v: cm/s
    //unit of w: deg/s

    rotVel = rotVel / 180 * 3.14159; //convert deg/s to radian/s
    double L = 37.7;  // the distance between two wheels, unit: cm
    double D = 45.72; // the diameter of the wheels, unit: cm

    double w1 = 1 / D*transVel + L / (2 * D)*rotVel;
    double w2= 1 / D*transVel - L / (2 * D)*rotVel;

    // convert unit from rad/s to deg/s

    w1 = w1 / 3.14159 * 180;
    w2 = w2 / 3.14159 * 180;

    //send th desired speeds of two wheels
    setWheelSpeed(w1, w2);

    return;
}


void getCurrentVehicleSpeed(double *pTrans, double *pRot)  //Units: cm/s, deg/s
{
    double w1, w2;
    getWheelSpeed(&w1, &w2);

    // convert from deg/s to radian/s
    w1 = w1 / 180 * 3.14159;
    w2 = w2 / 180 * 3.14159;

    // get v and w
    double L = 37.7;  // the distance between two wheels, unit: cm
    double D = 45.72; // the diameter of the wheels, unit: cm
    double v = D / 2 * w1 + D / 2 * w2;
    double w = D / L*w1 - D / L*w2;

    // convert from radian/s to deg
    w = w / 3.14159 * 180;

    //cout << "v=" << v << "  w=" << w << endl;
    (*pTrans) = v;
    (*pRot) = w;



    return;
}

void getWheelSpeed(double *pw1, double *pw2) {   // Unit: deg/s

    string recvBuff = "";
    double w1 = 0, w2 = 0; // the velocities of the wheels

    recvBuff = queryData(ConnectSocket);
    // parse the recvBuff to get w1 and w2
    // insert code here .....
    // .....
    // ..................
    // ......................
    int separ = recvBuff.find(',');
    istringstream istr(recvBuff.substr(0, separ));
    istr >> w1;
    istringstream istr2(recvBuff.substr(separ + 1, recvBuff.length() - separ));
    istr2 >> w2;

   // cout << "w1=" << w1 << "  w2=" << w2 << " Unit: RPM" << endl;

    // convert from RPM to deg/s
    w1 = w1 * 6;
    w2 = w2 * 6;

    (*pw1) = w1;
    (*pw2) = w2;

    string heartBag = "H#";
    send(ConnectSocket, heartBag);

    return;
}

double getVel() {   // Unit: cm/s
    double v, w;
    getCurrentVehicleSpeed(&v, &w);
    return v;
}

double getRotVel() {   // unit: deg/s
    double v, w;
    getCurrentVehicleSpeed(&v, &w);
    return w;
}

void setVel(double v) {   // unit: cm/s
    double tmpV, tmpW;
 //   getCurrentVehicleSpeed(&tmpV, &tmpW);
    sendDesiredVehicleSpeed(v, 0);
    string heartBag = "H#";
    send(ConnectSocket, heartBag);
}

void setRotVel(double w) {  // unit: deg/s
    double tmpV, tmpW;
    getCurrentVehicleSpeed(&tmpV, &tmpW);
    sendDesiredVehicleSpeed(tmpV, w);
    string heartBag = "H#";
    send(ConnectSocket, heartBag);
}

void stopVehicle() {
    sendDesiredVehicleSpeed(0, 0);
    string heartBag = "H#";
    send(ConnectSocket, heartBag);
}

//for keyboard interrupt
void sig_handler(int sig)
{
    if (sig == SIGINT)
    {
        string s = "P#P#P#";
        cout << " cccccccccccccccccccccccccccccccccccccccccc" << endl;
        send(ConnectSocket,s);
    }
}


void testVehicleMotion() {       ///// Test the communication with the Rasverry PI

    double desiredV = 0, desiredW = 0; // desired v and w
    double curV = 0, curW = 0;  //current v and w
    int i = 1;


    double w1, w2;


    int timeDelay = 10;  // unit: ms
    int tMax = 10000; // unit:ms


    // set a linear speed
    cout << "\n\n Set new V=60 \n\n";
    setVel(60);
    int t = 0;

    while (t <=tMax ) {
        t = t+timeDelay;

        Sleep(timeDelay);
        getWheelSpeed(&w1, &w2);
        getCurrentVehicleSpeed(&curV, &curW);
        cout << t << " w1=" << w1 << " w2=" << w2 << " V=" << curV << " W=" <<
curW<<endl;
    }
 //   speed1.close();


    cout << "----------------------------------------" << endl;
    // stop the vehicle
    cout << "\n\n Stop the vehicle now \n\n";
    stopVehicle();

    t = 0;

    while (t <= 10000) {

        t = t + timeDelay;

        Sleep(timeDelay);
        getWheelSpeed(&w1, &w2);
        getCurrentVehicleSpeed(&curV, &curW);
        cout << t << " " << "w1=" << w1 << " w2=" << w2 << " v=" << curV << " w="
<< curW << endl;
    }



   cout << "----------------------------------------" << endl;

   //ofstream speed2("speed2.txt", std::ofstream::trunc);
   cout << "\n\n Set new V=30 deg/s \n\n";
    setVel(30);
    t = 0;

    while (t<=tMax) {

        t = t+timeDelay;

        Sleep(timeDelay);
        getWheelSpeed(&w1, &w2);
        getCurrentVehicleSpeed(&curV, &curW);
        cout << t << " " <<"w1="<< w1 << " w2=" << w2 << " v=" << curV << " w=" <<
curW << endl;
    }
   // speed2.close();*/
     cout << "----------------------------------------" << endl;

     //ofstream speed2("speed2.txt", std::ofstream::trunc);
     cout << "\n\n Set new V=60  \n";
   //  setVel(60);
     t = 0;

     while (t <= tMax) {
         t = t + timeDelay;

         Sleep(timeDelay);
         getWheelSpeed(&w1, &w2);
         getCurrentVehicleSpeed(&curV, &curW);
         cout << t << " w1=" << w1 << " w2=" << w2 << " v=" << curV << " w=" <<
curW << endl;
     }
     // speed2.close();*/
     cout << "----------------------------------------" << endl;

     // stop the vehicle
     cout << "\n\n stop the vehicle and close communication !!!! \n\n";
     stopVehicle();
     Sleep(timeDelay);

    //close the communication
    string end = "P#";
    send(ConnectSocket, end);
    shutdown(ConnectSocket, SD_SEND);
    closesocket(ConnectSocket);
    return;
}

void testWheelSpeed(){
    double desiredW[12] = {10,20,30,40,0, 10, 20,30,60,0,100,150}; //RPM

    double t;
    double tMax =10000; //ms
    double timeDelay = 10; //ms

    double w1, w2;

    for (int i = 0; i < 12; i++) {

        cout << "\n\n-----------------------------------------------------" <<
endl;
        cout << "Desired w1 (RPM)=" << desiredW[i] << "  Desired w2 (RPM)=" <<
desiredW[i] << endl;
        cout << "\n\n-----------------------------------------------------\n" <<
endl;

        w1 = desiredW[i] * 6;  //RPM --> deg/s
        w2 = desiredW[i] * 6;  //RPM --> deg/s
        setWheelSpeed(w1, w2);

        t = 0;
        while (t <= tMax) {
            t = t + timeDelay;
            Sleep(timeDelay);
            getWheelSpeed(&w1, &w2);
           // cout << t << " w1 (RPM)=" << w1/6 << " w2 (RPM)=" << w2/6 << endl;
            cout << t <<" "<< w1 / 6 << " " << w2 / 6 << endl;
        }


    }
    // stop the vehicle
    cout << "\n\n stop the vehicle and close communication !!!! \n\n";
    stopVehicle();
    Sleep(timeDelay);

    //close the communication
    string end = "P#";
    send(ConnectSocket, end);
    shutdown(ConnectSocket, SD_SEND);
    closesocket(ConnectSocket);
    return;
}
```

10.2.3 Robot_vision.h

```
#ifndef ROBOT_VISION_H
#define ROBOT_VISION_H

#include <thread>
#include <atomic>
#include "opencv2/objdetect.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "opencv2/video/tracking.hpp"
#include "opencv2/core/ocl.hpp"
#include <ctime>
#include <iostream>
#include <fstream>
#include "robot_util.h"
#include "Aria.h"
#include "ArRobot.h"
#include "robot_util.h"
#include <zed/Camera.hpp>
using namespace std;

class RobotVision {
private:
       // MultiThread
    std::atomic<bool> is_data_available;
    std::atomic<bool> is_started;
    std::thread *current_thread;

       // Cascade
       cv::CascadeClassifier *cascade_classifier;
       cv::VideoCapture *video_capture;
       std::atomic<double> center_x;
       std::atomic<double> center_y;
       std::atomic<double> area;

       // Kalman Filter
       static const int NUM_KF_STATE = 5;
       static const int NUM_KF_MEASUREMENT = 2;
       cv::KalmanFilter *kalman_filter;
       cv::Mat_<float> measurement;
```

```
        // Debug Log
        bool is_debug;

        // Experiment Data Output
        ofstream *output_real;
        ofstream *output_predicted;

    // Depth Camera
    static const bool USE_DEPTH_CAMERA = true;
    sl::zed::Camera *zed;
public:
        ofstream *output_vision_time;
    static const int VISION_WIDTH = 1280;
    static const int VISION_HEIGHT = 720;
    /*static const int VISION_WIDTH = 640;
    static const int VISION_HEIGHT = 480;*/

        RobotVision();
        bool init(cv::String cascadeFilePath);
        bool init_kalman_filter();
    double detect_with_cascade_on_thread(double laser_angle_in_degree, bool
*is_predicted);
        bool detect_wtih_cascade_inline(double *center_x, double *area, double
*center_y);
        void correct_kalman_filter(double center_x, double center_y);
};
#endif
```

10.2.4 Robot_vision.cpp

```
#include "robot_vision.h"

RobotVision::RobotVision() {
        is_data_available = false;
        is_started = false;
        is_debug = false;

        output_real = new ofstream("robot_vision_cascade_real.txt");
        output_predicted = new
ofstream("robot_vision_kalman_filter_predicted.txt");
        output_vision_time = new ofstream("output_vision_time.txt");
}

bool RobotVision::init(cv::String cascadeFilePath) {
        cascade_classifier = new cv::CascadeClassifier();
        if (!cascade_classifier->load(cascadeFilePath)) {
                std::cout << "Error: There was an error when loading cascade xml
file." << endl;
                return false;
        }

    if (USE_DEPTH_CAMERA) {
        zed = new sl::zed::Camera(sl::zed::HD720);
        sl::zed::ERRCODE err = zed->init(sl::zed::MODE::PERFORMANCE, 0, true);

        // Quit if an error occurred
        if (err != sl::zed::SUCCESS) {
            std::cout << "Error: Unable to init the ZED:" << errcode2str(err) <<
std::endl;
            delete zed;
            return false;
        }
    }
    else {
        video_capture = new cv::VideoCapture();
        if (!video_capture->open(0)) {
            std::cout << "Error: There was an error when opening camera." << endl;
            return false;
        }
    }

       kalman_filter = new cv::KalmanFilter(RobotVision::NUM_KF_STATE,
RobotVision::NUM_KF_MEASUREMENT);
       kalman_filter->transitionMatrix = (
              cv::Mat_<float>(5, 5) <<
              1, 0, 1, 0, 0, // x
              0, 1, 0, 1, 0, // y
              0, 0, 1, 0, 1, // Vx
              0, 0, 0, 1, 0, // Vy
              0, 0, 0, 0, 1); // Ax

       measurement = cv::Mat_<float>(2, 1);
       measurement.setTo(cv::Scalar(0));
       return true;
}

void detect_with_cascade_raw(cv::VideoCapture *video_capture, sl::zed::Camera
*zed, bool use_depth_camera, cv::CascadeClassifier *cascade_classifier,
       std::atomic<bool> *is_data_available, std::atomic<double> *return_center_x,
std::atomic<double> *return_center_y,
       std::atomic<double> *return_area, bool is_debug) {
       cv::Mat captureFrame, grayscaleFrame;
       double area = 0, center = 0;
       std::vector<cv::Rect> stops;

    int NUM_OF_SPLITS = 3;
    int *splits = new int[NUM_OF_SPLITS];
    int CLOSE_THRESHOLD = 245;
    int area_left = 0, area_right = 0;

    // Initialize all to 0
    for (int i = 0; i < NUM_OF_SPLITS; i++) {
        splits[i] = 0;
    }

       cv::namedWindow("outputCapture", 1);
    if (use_depth_camera) {
        int width = zed->getImageSize().width;
        int height = zed->getImageSize().height;
        cv::Mat image(height, width, CV_8UC4, 1);
        cv::Mat depth(height, width, CV_8UC4, 1);

        if (!zed->grab(sl::zed::SENSING_MODE::FULL))
        {
            // Retrieve left color image
            sl::zed::Mat left = zed->retrieveImage(sl::zed::SIDE::LEFT);
            memcpy(image.data, left.data, width * height * 4 * sizeof(uchar));

            // Retrieve depth map
            sl::zed::Mat depthmap = zed-
>normalizeMeasure(sl::zed::MEASURE::DEPTH);
            memcpy(depth.data, depthmap.data, width * height * 4 * sizeof(uchar));

            /*for (int i = 1; i < RobotVision::VISION_WIDTH; i++) {
                for (int j = 1; j < RobotVision::VISION_HEIGHT; j++) {
                    sl::uchar3 depth_value = depthmap.getValue(i, j);
                    if (depth_value.c1 >= CLOSE_THRESHOLD && depth_value.c2 >=
CLOSE_THRESHOLD && depth_value.c3 >= CLOSE_THRESHOLD) {
                        for (int k = 1; k <= NUM_OF_SPLITS; k++) {
                            if (i >= RobotVision::VISION_WIDTH / NUM_OF_SPLITS *
(k - 1) && i < RobotVision::VISION_WIDTH / NUM_OF_SPLITS * k) {
                                splits[k - 1] += 1;
                            }
                        }
                    }
                }
            }

            int max_split = -1;
            for (int k = 1; k <= NUM_OF_SPLITS; k++) {
                if (splits[k - 1] > max_split) {
                    area_left = RobotVision::VISION_WIDTH / NUM_OF_SPLITS * (k -
1);
                    area_right = RobotVision::VISION_WIDTH / NUM_OF_SPLITS * k;
                    max_split = splits[k - 1];
                }
            }*/

            captureFrame = image;
        }
    }
    else {
        video_capture->read(captureFrame);
    }

       // No need to convert
       //cvtColor(captureFrame, grayscaleFrame, CV_BGR2GRAY);
       //equalizeHist(grayscaleFrame, grayscaleFrame);
    /*std::cout << "Vision Image Size: " << captureFrame.rows << ", " <<
captureFrame.cols << std::endl;*/

    if (use_depth_camera) {
        //cv::Rect temp(area_left, 1, 1280 / NUM_OF_SPLITS, 719);
        //grayscaleFrame = captureFrame(temp);
        grayscaleFrame = captureFrame.clone();
    }
    else {
        grayscaleFrame = captureFrame.clone();
    }

       cascade_classifier->detectMultiScale(grayscaleFrame, stops, 1.05, 3,
CV_HAAR_FIND_BIGGEST_OBJECT | CV_HAAR_SCALE_IMAGE, cv::Size(60, 60));
```

```
	if (stops.empty()) {
		//std::cout << "Info: Did not detect the object!" << endl;
		(*return_center_x) = -1.0;
		(*return_center_y) = -1.0;
		*is_data_available = true;
	//	cv::imshow("outputCapture", captureFrame);
		cv::waitKey(1);
		return;
	}

	double tmpArea = 0;
	double temp_center_x = 9999;
	for (int i = 0; i < stops.size(); i++) {
		cv::Point pt1(area_left + stops[i].x + stops[i].width, stops[i].y + 
stops[i].height);
		cv::Point pt2(area_left + stops[i].x, stops[i].y);
		rectangle(captureFrame, pt1, pt2, cvScalar(0, 255, 0, 0), 1, 8, 0);
		if (stops[i].x < temp_center_x) {
			temp_center_x = area_left + stops[i].x;
			// This is very important, need to calculate the center of the 
detected image, instead of left-up corner
			(*return_center_x) = area_left + stops[i].x + stops[i].width / 
2;
			(*return_center_y) = area_left + stops[i].y + stops[i].height 
/ 2;
		}
	}

	cv::imshow("outputCapture", captureFrame);
	cv::waitKey(1);
	(*return_area) = tmpArea;
	if (is_debug) {
		std::cout << "Center: " << center << "; Area:" << area << std::endl;
	}

	*is_data_available = true;
}

bool RobotVision::detect_wtih_cascade_inline(double *return_center_x, double 
*return_area, double *return_center_y) {
	std::atomic<double> init_center_x = -1.0;
	std::atomic<double> init_center_y = -1.0;
	std::atomic<double> init_return_area = -1.0;
	std::atomic<bool> init_is_data_available = false;
	clock_t timer_start, timer_end;

	timer_start = clock();
	detect_with_cascade_raw(video_capture, zed, USE_DEPTH_CAMERA, 
cascade_classifier, &init_is_data_available,
		&init_center_x, &init_center_y, &init_return_area, is_debug);

	if (is_debug) {
		timer_end = clock();
		//std::cout << "It takes " << RobotUtil::get_diff_in_ms(timer_start, 
timer_end) << " ms for vision to execute once." << std::endl;
	}
```

```
        (*return_center_x) = init_center_x;
        (*return_center_y) = init_center_y;
        (*return_area) = init_return_area;

        if (init_center_x < 0) {
                return false;
        }
        return true;
}

void detect_with_cascade_raw_loop(RobotVision *vision, std::atomic<double>
*return_center_x, std::atomic<double> *return_center_y,
        std::atomic<double> *return_area, std::atomic<bool> *is_data_available,
double laser_angle_in_degree) {

        double vision_area = 0;
        double vision_center_x = 0;
        double vision_center_y = 0;
        long temp_vision_time_start = RobotUtil::get_current_time_in_ms();
        boolean result = false;

        while (true) {
                temp_vision_time_start = RobotUtil::get_current_time_in_ms();
                result = vision->detect_wtih_cascade_inline(&vision_center_x,
&vision_area, &vision_center_y);
                (*vision->output_vision_time) << RobotUtil::get_current_time_in_ms()
<< " " << (RobotUtil::get_current_time_in_ms() - temp_vision_time_start) << endl;
                if (result) {
                        break;
                }
        }

        (*return_center_x) = vision_center_x;
        (*return_center_y) = vision_center_y;
        (*return_area) = vision_area;
        (*is_data_available) = true;
}

bool RobotVision::init_kalman_filter() {

        std::atomic<double> init_center_x = -1.0;
        std::atomic<double> init_center_y = -1.0;
        std::atomic<double> init_return_area = -1.0;
        std::atomic<bool> init_is_data_available = false;

        while (true) {
                detect_with_cascade_raw(video_capture, zed, USE_DEPTH_CAMERA,
cascade_classifier, &init_is_data_available,
                        &init_center_x, &init_center_y, &init_return_area, is_debug);
                if (init_center_x > 0 && init_center_y > 0) {
                        (*output_real) << RobotUtil::get_current_time_in_ms() << " "
<< init_center_x << endl;

                        kalman_filter->statePost.at<float>(0) = init_center_x;
                        kalman_filter->statePost.at<float>(1) = init_center_y;
                        kalman_filter->statePost.at<float>(2) = 0;
                        kalman_filter->statePost.at<float>(3) = 0;
                        kalman_filter->statePost.at<float>(4) = 0;
```

```
                    setIdentity(kalman_filter->measurementMatrix);
                    setIdentity(kalman_filter->processNoiseCov,
cv::Scalar::all(1e-4));
                    setIdentity(kalman_filter->measurementNoiseCov,
cv::Scalar::all(10));
                    setIdentity(kalman_filter->errorCovPost, cv::Scalar::all(.1));

                    //if (is_debug) {
                           cout << "Initializing vision kalman filter done: " <<
init_center_y << ", " << init_center_y << endl;
                    //}

                    return true;
             }
       }

       return false;
}

void RobotVision::correct_kalman_filter(double center_x, double center_y) {
       if (center_x > 0 && center_y > 0) {
             measurement(0) = center_x;
             measurement(1) = center_y;
             kalman_filter->correct(measurement);
       }
}

double RobotVision::detect_with_cascade_on_thread(double laser_angle_in_degree,
bool *is_predicted) {
    // If the image data is available
    if (is_data_available) {
        is_data_available = false;
        is_started = false;
        current_thread->join();
             (*is_predicted) = false;

             (*output_real) << RobotUtil::get_current_time_in_ms() << " " <<
center_x << endl;
             //cout << "Real" << endl;

             if (center_x > 0 && center_y > 0) {
                    measurement(0) = center_x;
                    measurement(1) = center_y;
                    kalman_filter->correct(measurement);

                    if (is_debug) {
                    //     cout << "Real Found, Correct KF: " << center_x << ", "
<< center_y << endl;
                    }

                    return center_x;
             }
             else {
                    cv::Mat prediction = kalman_filter->predict();

                    if (is_debug) {
                     //     cout << "Real Not Found, Use Predicted: " <<
prediction.at<float>(0) << ", " << prediction.at<float>(1) << endl;
                     }

                     return prediction.at<float>(0);
              }
    }
    // If the image data is not available
    else {
        // If the thread of getting the data is not started
        if (false == is_started) {
            //current_thread = new std::thread(&detect_with_cascade_raw,
video_capture, cascade_classifier, &is_data_available, &center_x, &center_y,
&area, is_debug);
                     current_thread = new
std::thread(&detect_with_cascade_raw_loop, this, &center_x, &center_y, &area,
&is_data_available, laser_angle_in_degree);
                     is_started = true;
        }

              cv::Mat prediction = kalman_filter->predict();
              (*output_predicted) << RobotUtil::get_current_time_in_ms() << " " <<
prediction.at<float>(0) << endl;
       //     cout << "Predicted" << endl;

              if (is_debug) {
              //     cout << "Predicted: " << prediction.at<float>(0) << ", " <<
prediction.at<float>(1) << endl;
              }

              (*is_predicted) = true;

        return prediction.at<float>(0);
    }
}
```

10.2.5 Robot_laser.h

```
#ifndef ROBOT_LASER_H
#define ROBOT_LASER_H

#include <iostream>
#include <thread>
#include <atomic>
#include <math.h>
#include <fstream>
#include "Urg_driver.h"
#include "robot_util.h"
#include "opencv2/video/tracking.hpp"
using namespace qrk;
using namespace std;

class RobotLaser {
private:
       // MultiThread
       std::atomic<bool> is_data_available;
```

```
        std::atomic<bool> is_started;
        std::thread *current_thread;

        // Laser Urg
        std::atomic<double> _distance;
        std::atomic<double> _laser_angle_in_degree;

        // Kalman Filter
        static const bool USE_KALMAN_FILTER = true;
        static const int NUM_KF_STATE = 5;
        static const int NUM_KF_MEASUREMENT = 2;
        cv::KalmanFilter *kalman_filter;
        cv::Mat_<float> measurement;
    // If we put laser on thread, then we don't need to initialize the laser any
more
        bool is_kalman_filter_initialized = false;

        // Experiment Data Output
        ofstream *output_real;
        ofstream *output_predicted;
        ofstream *output_real_angle;
        ofstream *output_predicted_angle;
public:
        Urg_driver *urg;

        // Debug Log
        bool is_debug;

        static const Urg_driver::connection_type_t SERIAL_TYPE =
Urg_driver::Serial;
        static const Urg_driver::connection_type_t ETHERNET_TYPE =
Urg_driver::Ethernet;
        static const int URG_MIN_DEGREE = -120;
        static const int URG_MAX_DEGREE = 120;
        static const int URG_TOTAL_DEGREE = 240;
        static const int URG_NUM_OF_DATA_POINTS_RETURNED = 682;

        RobotLaser();
        bool init();
        bool init(Urg_driver::connection_type_t type);
        bool init_kalman_filter(int start_angle, int end_angle, double
*laser_angle_in_degree);
        bool is_open();
        void close();
    double get_distance(int start_angle, int end_angle, double
*laser_angle_in_degree);
        double get_distance_on_thread(int start_angle, int end_angle, double
*laser_angle_in_degree);
};
#endif
```

10.2.6 Robot_laser.cpp

```
#include "robot_laser.h"

RobotLaser::RobotLaser() {
		is_data_available = false;
		is_started = false;
		is_debug = true;

		if (USE_KALMAN_FILTER) {
			output_real = new ofstream("robot_laser_real.txt");
			output_predicted = new
ofstream("robot_laser_kalman_filter_predicted.txt");
			output_real_angle = new ofstream("robot_laser_real_angle.txt");
			output_predicted_angle = new
ofstream("robot_laser_kalman_filter_predicted_angle.txt");
		}
}

bool RobotLaser::init() {
		if (USE_KALMAN_FILTER) {
			kalman_filter = new cv::KalmanFilter(RobotLaser::NUM_KF_STATE,
RobotLaser::NUM_KF_MEASUREMENT);
			kalman_filter->transitionMatrix = (
				cv::Mat_<float>(5, 5) <<
				1, 1, 0, 0, 0,  // distance
				0, 1, 1, 0, 0,  // Vx
				0, 0, 1, 0, 0,  // Ax
				0, 0, 0, 1, 1,  // laser angle
				0, 0, 0, 0, 1); // laser angle change speed

			measurement = cv::Mat_<float>(2, 1);
			measurement.setTo(cv::Scalar(0));
		}

		return init(RobotLaser::SERIAL_TYPE);
}

bool RobotLaser::init(Urg_driver::connection_type_t type) {
		urg = new Urg_driver();
		const char *device_name;
		int baudrate_or_port;

		if (type == Urg_driver::Serial) {
			device_name = "COM3";
			baudrate_or_port = 115200;
		}
		else if (type == Urg_driver::Ethernet) {
			device_name = "192.168.0.10";
			baudrate_or_port = 10940;
		}
		else {
			cout << "Error: Wrong Urg_driver connection type." << endl;
			return false;
		}

		if (urg->is_open()) {
			urg->close();
		}

		if (!urg->open(device_name, baudrate_or_port, type)) {
			cout << "Error: There was an error when executing
Urg_driver::open(): " << urg->what() << endl;
```

```
		return false;
	}

	return true;
}

bool RobotLaser::is_open() {
	return urg->is_open();
}

void RobotLaser::close() {
	urg->close();
}

double RobotLaser::get_distance(int start_angle, int end_angle, double
*laser_angle_in_degree) {
	// Urg laser device will return 682 data points
	// The scan range of Urg laser device is -120 degree to 120 degree

	if (start_angle > RobotLaser::URG_MAX_DEGREE
		|| start_angle < RobotLaser::URG_MIN_DEGREE
		|| end_angle > RobotLaser::URG_MAX_DEGREE
		|| end_angle < RobotLaser::URG_MIN_DEGREE
		|| start_angle > end_angle) {
		cout << "Error: Invalid input range. Range should be within -120
degree to 120 degree." << endl;
		return -1.0;
	}

	clock_t timer_start, timer_end;
	timer_start = clock();

	urg->start_measurement(Urg_driver::Distance, 1, 0);

	vector<long> data;
	long time_stamp = 0;
	if (!urg->get_distance(data, &time_stamp)) {
		cout << "Error: There was an error when executing
Urg_driver::get_distance(): " << urg->what() << endl;
		urg->close();
		return -1.0;
	}

	double degree_per_points = RobotLaser::URG_TOTAL_DEGREE * 1.0 /
RobotLaser::URG_NUM_OF_DATA_POINTS_RETURNED;
	double degree_counter = -120.0;
	double min_distance = 99999.0;
	int min_distance_data_point_index = -1;
	double min_distance_degree = 0.0;
	for (int i = 0; i < data.size(); i++) {
		if (degree_counter > start_angle && degree_counter < end_angle) {
			if (data[i] > 50 && data[i] < min_distance) {
				min_distance_data_point_index = i;
				min_distance_degree = degree_counter;
				min_distance = data[i];
			}
		}
		degree_counter += degree_per_points;
	}

	if (min_distance_data_point_index == -1) {
		cout << "Error: Cannot find the closest distance." << endl;
		return -1.0;
	}

	if (is_debug) {
		//cout << "Found the closest distance: " << min_distance << " at " 
<< min_distance_degree
		//<< " degree with data point index [" << 
min_distance_data_point_index << "]." << endl;
		timer_end = clock();
	//	cout << "It takes " << RobotUtil::get_diff_in_ms(timer_start, 
timer_end) << " ms for laser measurement only." <<endl;
	}

	if (USE_KALMAN_FILTER) {
		// Use Kalman Filter to correct distance
		if (is_kalman_filter_initialized) {
			measurement(0) = min_distance;
			measurement(1) = min_distance_degree;
			kalman_filter->correct(measurement);
			cv::Mat prediction = kalman_filter->predict();

		//	cout << "KalmanFilter revised distance: " << 
prediction.at<float>(0) << ", " << prediction.at<float>(1) << endl;
	//		cout << "Original measured distance: " << min_distance << ", " 
<< min_distance_degree << endl;

			(*output_predicted) << RobotUtil::get_current_time_in_ms() << 
" " << prediction.at<float>(0) << endl;
			(*output_real) << RobotUtil::get_current_time_in_ms() << " " 
<< min_distance << endl;
			(*output_predicted_angle) << 
RobotUtil::get_current_time_in_ms() << " " << prediction.at<float>(1) << endl;
			(*output_real_angle) << RobotUtil::get_current_time_in_ms() << 
" " << min_distance_degree << endl;

			min_distance = prediction.at<float>(0);
			min_distance_degree = prediction.at<float>(1);
		}
	}

	(*laser_angle_in_degree) = min_distance_degree;

	return min_distance;
}

bool RobotLaser::init_kalman_filter(int start_angle, int end_angle, double 
*laser_angle_in_degree) {

	if (USE_KALMAN_FILTER) {
	 if (false == is_open()) {
	     init();
	 }
		double min_distance = get_distance(start_angle, end_angle,
laser_angle_in_degree);

		kalman_filter->statePost.at<float>(0) = min_distance;
		kalman_filter->statePost.at<float>(1) = (*laser_angle_in_degree);
		kalman_filter->statePost.at<float>(2) = 0;
		kalman_filter->statePost.at<float>(3) = 0;
		kalman_filter->statePost.at<float>(4) = 0;

		setIdentity(kalman_filter->measurementMatrix);
		setIdentity(kalman_filter->processNoiseCov, cv::Scalar::all(1e-4));
		setIdentity(kalman_filter->measurementNoiseCov,
cv::Scalar::all(10));
		setIdentity(kalman_filter->errorCovPost, cv::Scalar::all(.1));

		cout << "Initializing laser kalman filter done: " << min_distance <<
", " << (*laser_angle_in_degree) << endl;
		(*output_real) << RobotUtil::get_current_time_in_ms() << " " <<
min_distance << endl;
		(*output_real_angle) << RobotUtil::get_current_time_in_ms() << " "
<< (*laser_angle_in_degree) << endl;

		is_kalman_filter_initialized = true;
	}

	return true;
}

void get_distance_raw(RobotLaser *laser, int start_angle, int end_angle,
std::atomic<bool> *is_data_available,
	std::atomic<double> *distance, std::atomic<double> *laser_angle_in_degree)
{

	if (start_angle > RobotLaser::URG_MAX_DEGREE
		|| start_angle < RobotLaser::URG_MIN_DEGREE
		|| end_angle > RobotLaser::URG_MAX_DEGREE
		|| end_angle < RobotLaser::URG_MIN_DEGREE
		|| start_angle > end_angle) {
		cout << "Error: Invalid input range. Range should be within -120
degree to 120 degree." << endl;
		(*is_data_available) = true;
		(*distance) = -1.0;
		(*laser_angle_in_degree) = -1.0;
		return;
	}

	clock_t timer_start, timer_end;
	timer_start = clock();

	laser->urg->start_measurement(Urg_driver::Distance, 1, 0);

	vector<long> data;
	long time_stamp = 0;
	if (!laser->urg->get_distance(data, &time_stamp)) {
		cout << "Error: There was an error when executing
Urg_driver::get_distance(): " << laser->urg->what() << endl;
		(*is_data_available) = true;
		(*distance) = -1.0;
```

```
                (*laser_angle_in_degree) = -1.0;
                laser->urg->close();
                return;
        }

        double degree_per_points = RobotLaser::URG_TOTAL_DEGREE * 1.0 /
RobotLaser::URG_NUM_OF_DATA_POINTS_RETURNED;
        double degree_counter = -120.0;
        double min_distance = 99999.0;
        int min_distance_data_point_index = -1;
        double min_distance_degree = 0.0;
        for (int i = 0; i < data.size(); i++) {
                if (degree_counter > start_angle && degree_counter < end_angle) {
                        if (data[i] > 50 && data[i] < min_distance) {
                                min_distance_data_point_index = i;
                                min_distance_degree = degree_counter;
                                min_distance = data[i];
                        }
                }
                degree_counter += degree_per_points;
        }

        if (min_distance_data_point_index == -1) {
                cout << "Error: Cannot find the closest distance." << endl;
                (*is_data_available) = true;
                (*distance) = -1.0;
                (*laser_angle_in_degree) = -1.0;
                return;
        }

        if (laser->is_debug) {
        //      cout << "Found the closest distance: " << min_distance << " at " <<
min_distance_degree
        //              << " degree with data point index [" <<
min_distance_data_point_index << "]." << endl;
                timer_end = clock();
        //      cout << "It takes " << RobotUtil::get_diff_in_ms(timer_start,
timer_end) << " ms for laser measurement only." << endl;
        }

    (*is_data_available) = true;
        (*distance) = min_distance;
        (*laser_angle_in_degree) = min_distance_degree;
}

double RobotLaser::get_distance_on_thread(int start_angle, int end_angle, double
*laser_angle_in_degree) {
        // If the laser data is available
        if (is_data_available) {
                is_data_available = false;
                is_started = false;
                current_thread->join();

                if (_distance > 0 && _laser_angle_in_degree > 0) {
              if (USE_KALMAN_FILTER) {
                 measurement(0) = _distance;
                 measurement(1) = _laser_angle_in_degree;
                 kalman_filter->correct(measurement);
                if (is_debug) {
        //            cout << "Real Found, Correct KF: " << _distance << ", " <<
_laser_angle_in_degree << endl;
                }
            }

                        (*output_real) << RobotUtil::get_current_time_in_ms() << " "
<< _distance << endl;
                        (*output_real_angle) << RobotUtil::get_current_time_in_ms() <<
" " << _laser_angle_in_degree << endl;

                        (*laser_angle_in_degree) = _laser_angle_in_degree;
                        return _distance;
              }
              else {
            if (USE_KALMAN_FILTER) {
                cv::Mat prediction = kalman_filter->predict();

                if (is_debug) {
            //        cout << "Real Not Found, Use Predicted: " <<
prediction.at<float>(0) << ", " << prediction.at<float>(1) << endl;
                }

                (*output_predicted) << RobotUtil::get_current_time_in_ms() << " "
<< prediction.at<float>(0) << endl;
                (*output_predicted_angle) << RobotUtil::get_current_time_in_ms()
<< " " << prediction.at<float>(1) << endl;

                (*laser_angle_in_degree) = prediction.at<float>(1);
                return prediction.at<float>(0);
            }
            else {
                (*laser_angle_in_degree) = _laser_angle_in_degree;
                return _distance;
            }
              }

              return _distance;
       }
       // If the laser data is not available
       else {
              // If the thread of getting the data is not started
              if (false == is_started) {
                     current_thread = new std::thread(&get_distance_raw, this,
start_angle, end_angle, &is_data_available, &_distance, &_laser_angle_in_degree);
                     is_started = true;
              }

              cv::Mat prediction = kalman_filter->predict();
              (*output_predicted) << RobotUtil::get_current_time_in_ms() << " " <<
prediction.at<float>(0) << endl;
              (*output_predicted_angle) << RobotUtil::get_current_time_in_ms() <<
" " << prediction.at<float>(1) << endl;
       //     cout << "Predicted" << endl;

              if (is_debug) {
```

```
		//	cout << "Predicted: " << prediction.at<float>(0) << ", " <<
prediction.at<float>(1) << endl;
		}

		(*laser_angle_in_degree) = prediction.at<float>(1);
		return prediction.at<float>(0);
	}
}
```

10.2.7 Robot_bp.h

```
#ifndef ROBOT_BP_H
#define ROBOT_BP_H

#include <iostream>
using namespace std;

class RobotBP {
private:
	int in_num;
	int mid_num;
	int out_num;

	double **w;
	double **v;
	double *output;
public:
	RobotBP(int in_num, int mid_num, int out_num);
	void predict(double distance, double imageX);
	double* get_output();
};
#endif
```

10.2.8 Robot_bp.cpp

```
#include "robot_bp.h"

RobotBP::RobotBP(int _in_num, int _mid_num, int _out_num) {
	in_num = _in_num;
	mid_num = _mid_num;
	out_num = _out_num;
}

double* RobotBP::get_output() {
	return output;
}

void RobotBP::predict(double distance, double imageX) {
	double x[2];  //input layers
	double H[100]; //hidden layers

				//data pre-processing
				//too close: 0-200; close: 200-400; center: 400-800;
far: 800-1200; too far: 1200-5000
```

```
        if (distance <= 200 && distance >= 0)
                x[0] = 4;
        else if (distance <= 400 && distance > 200)
                x[0] = 3;
        else if (distance <= 800 && distance > 400)
                x[0] = 2;
        else if (distance <= 1200 && distance > 800)
                x[0] = 1;
        else if (distance <= 5000 && distance > 1000)
                x[0] = 0;

        //too left: 0-140; left: 140-220; center: 220-420; right: 420-500; too
right: 500-640
        if (imageX <= 140 && imageX >= 0)
                x[1] = 0;
        else if (imageX <= 220 && imageX > 140)
                x[1] = 1;
        else if (imageX <= 420 && imageX > 220)
                x[1] = 2;
        else if (imageX <= 500 && imageX > 420)
                x[1] = 3;
        else if (imageX <= 640 && imageX > 500)
                x[1] = 4;


        //display status
        cout << "distance=" << x[0] << "    " << "image location=" << x[1] << "\n";

        // calculate the outputs of the hidden units
        for (int i = 0; i < 100; i++)
        {
                double sum = 0;
                for (int j = 0; j < 2; j++)
                {
                        sum = sum + x[j] * w[j][i];
                }
                H[i] = 1 / (1 + exp(-sum));
        }

        //caculate the outputs of the output layer
        for (int i = 0; i < 10; i++)
        {
                double sum = 0;
                for (int j = 0; j < 100; j++)
                {
                        sum = sum + H[j] * v[j][i];
                }
                output[i] = sum;
                cout << "number " << i << " is " << output[i] << endl;
        }
}
```