8-2019

Embedded System Design of Robot Control Architectures for Unmanned Agricultural Ground Vehicles

Ryan Humphrey
University of Nebraska - Lincoln, ryan.humphrey@huskers.unl.edu

EMBEDDED SYSTEM DESIGN OF ROBOT CONTROL ARCHITECTURES FOR UNMANNED AGRICULTURAL GROUND VEHICLES

by

Ryan Patrick Humphrey

A THESIS

Presented to the Faculty of

The Graduate College of the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Electrical Engineering

Under the Supervision of Professors A. John Boye and Santosh Pitla

Lincoln, Nebraska

August, 2019

EMBEDDED SYSTEM DESIGN OF ROBOT CONTROL ARCHITECTURES FOR UNMANNED AGRICULTURAL GROUND VEHICLES

Ryan P. Humphrey, M.S.

University of Nebraska, 2019

Advisors: A. John Boye and Santosh Pitla

Engineering technology has matured to the extent where accompanying methods for unmanned field management is now becoming a technologically achievable and economically viable solution to agricultural tasks that have been traditionally performed by humans or human operated machines. Additionally, the rapidly increasing world population and the daunting burden it places on farmers in regards to the food production and crop yield demands, only makes such advancements in the agriculture industry all the more imperative. Consequently, the sector is beginning to observe a noticeable shift, where there exist a number of scalable infrastructural changes that are in the process of slowly being implemented onto the modular machinery design of agricultural equipment. This work is being pursued in effort to provide firmware descriptions and hardware architectures that integrate cutting edge technology onto the embedded control architectures of agricultural machinery designs to assist in achieving the end goal of complete and reliable unmanned agricultural automation.

In this thesis, various types of autonomous control algorithms integrated with obstacle avoidance or guidance schemes, were implemented onto controller area network (CAN) based distributive real-time systems (DRTSs) in form of the two unmanned agricultural ground vehicles (UAGVs). Both vehicles are tailored to different applications in the agriculture domain as they both leverage state-of-the-art sensors and modules to attain the end objective of complete autonomy to allow for the automation of various types of agricultural related tasks. The further development of the embedded system design of these machines called for the developed firmware and hardware to be implemented onto both an event triggered and time triggered CAN bus control architecture as each robot employed its own separate embedded control scheme.

For the first UAGV, a multiple GPS waypoint navigation scheme is derived, developed, and evaluated to yield a fully controllable GPS-driven vehicle. Additionally, obstacle detection and avoidance capabilities were also implemented onto the vehicle to serve as a safety layer for the robot control architecture, giving the ground vehicle the ability to reliability detect and navigate around any obstacles that may happen to be in the vicinity of the assigned path. The second UAGV was a smaller robot designed for field navigation applications. For this robot, a fully autonomous sensor based algorithm was proposed and implemented onto the machine. It is demonstrated that the utilization and implementation of laser, LIDAR, and IMU sensors onto a mobile robot platform allowed for the realization of a fully autonomous non-GPS sensor based algorithm to be employed for field navigation. The developed algorithm can serve as a viable solution for the application of microclimate sensing in a field.

Acknowledgements

First, I would like to thank by two co-advisors, Dr. John Boye and Dr. Santosh Pitla for all of the invaluable guidance and insight they had to offer throughout this entire process. Had of it not been by Dr. Boye, I would have never known about all the exciting engineering robotics projects that were happening inside of Splintor's laboratory, and I would have never gotten to benefit from getting to work on such an exhilarating, hands on project for my thesis that allowed me to dive deeper into the engineering areas I'm most passionate about – robotics and electronics design. Additionally, Dr. Pitla gave me the freedom to select the available the projects to build upon as well as offered flexibility for areas of further development from the projects that I chose. He equipped me with the tools necessary to succeed while also allowed me to wrestle first hand with the engineering design problems that I would inevitably face along the way throughout the course of my research. Their leadership coupled with the encouragement I received from the two of them played an integral role in allowing me to successfully complete this project.

I'd also like to extend gratitude to my third committee member, Dr. Yufeng Ge. Thanks to his research and suggestions for my project, I got to reap the reward of being able to see first-hand, an application for my project, which added motivation and value to my work.

Additionally, there are also several others I'd like to express gratitude towards who contributed, either directly or indirectly, in some shape or form to this project. To my mom, for her never ending support and encouragement. To my dad, for giving me inspiration to become an engineer and assuring me that this was an endeavor I was capable of pursuing. To John Wurm, for lending a helping hand to assist in troubleshooting during a critical stage of project testing. To my lab mate, CheeTown Liew, for all the insight, wisdom, and assistance he offered throughout the entire process of formulating, developing, and testing this project. Finally and most importantly, I'd like to thank my Lord and Savior Jesus Christ, for it was through His enabling power that this was all possible.

Table of Contents

List of Figures

List of Tables

List of Equations

1 Introduction

The motivation for the research undertaken in this thesis will be presented in this chapter. The chapter will also introduce and discuss important technical concepts that were foundational in the design of the two agricultural ground robots that were developed in this thesis. Finally, an outline with a set of objectives for this thesis will be given. The goal of this chapter is to first inform the reader of the imminent demand for incorporating robotics and automation into the agriculture industry. Subsequently, this discussion will be followed by introducing and elucidating the general system design principles that form the backbone of the conventional design architecture employed in modern field machinery equipment. Finally, the chapter will conclude by laying out the roadmap for the remainder of the document, where proposed solutions that address portions of this impending problem are presented via the development and implementation of two unmanned agricultural ground robots.

1.1 Motivation

The rapidly increasing world population presents a challenging problem for the agriculture industry. Developing an efficient method for food and fuel production to meet this ever increasing demand is easily the most pressing issue that scientists and engineers in this sector are facing today. To further compound the problem, the number of agricultural labors over the past decade has dipped substantially, in both fully developed and developing countries [1]. The shortage of people working on farms today is becoming a chronic worldwide issue. Consequently, to help combat this dilemma, the areas of robotics and automation are starting to gain traction in the agriculture industry.

While the concept of automation, is not a foreign idea to the agriculture sector, as this technology has been utilized in this field for years in the form of harvesting (albeit, still cab operated), it has long been bottlenecked by the current state of technology itself. To put it simply, the engineering was just simply not sufficiently developed or adequately refined to the point of being able to offer any sound solutions for the automation or *unmanned* automation of any of the other more complex agricultural tasks. These more

complex agricultural tasks, in order to be successfully completed, would often require technological feats and achievements such as localization, increased precision, improved dexterity, and/or more advanced vision processing systems. These more advanced agricultural jobs are the tasks that fall under the umbrella of what is often referred to as precision agriculture or smarting farming – agricultural related tasks that necessitate precision in guidance and measurements which in turn, requires an appropriate response to the variability in the field in regards to crop growth and raising livestock [2]. For decades, this labor was performed manually by humans, and there was very little incentive for change as the supply demands of the time were not as daunting and burdensome as they are today. Thankfully, as this impending problem has become more imminent, it has hence generated an increasing amount of interest and research towards merging the developmental progress made in robotics, mobile vehicles, and automation over the recent years, with the agricultural machinery in use today.

Over the last couple of decades, the development of technologies such as global positioning systems (GPSs), machine vision, laser-based sensors, and inertial devices have allowed for the concept of autonomous, mobile vehicles to go from being a wishful, nonsensical idea to now being a technologically and economically viable possibility. In the agricultural sector, the implications of these technological advancements are pronounced as this ushered the beginning of developing cab-less field machinery designs to begin automating the execution of precision agricultural related tasks.

The benefits to having such a system are two-fold. First, autonomous ground vehicles remedy the labor concerns as was mentioned earlier. Secondly, they can significantly increase production rates as they are often appreciably more efficient than human laborers and can also yield higher quality products [3]. Consequently, precision agriculture is beginning to undergo a vast number of scalable infrastructural changes, where the engineering advancements being made in robotics and automation are starting to get leveraged and integrated into the agricultural realm.

This thesis is an exploration into the further development of agricultural machinery, formulating and further developing the embedded system design and control architecture of these robots and evaluating the practical implementation and viability of various state of the art sensors and modules in serving as navigational aids in unmanned automation. The work offered in this thesis is an extension of the current technological trends being observed in the field today, where this restructuring of agricultural equipment is necessary in order for the industry to keep pace with modern technology to be able to provide innovative field machinery that can plausibly fulfill their rapidly expanding roles.

1.2 Background

This thesis focuses on the further development of two agricultural ground robots – the unmanned agricultural ground vehicle (UAGV) and the Inter-row robot. In order to understand the work accomplished in this thesis, it is incumbent to first address the basic operating principles and concepts that served as the building blocks to designing and further developing these robots. The goal of this section is to introduce and discuss important technical concepts that were foundational in the design of these two agricultural ground machines. To alleviate any chance of confusion, this discussion must take place before the actual design of the robots themselves can be addressed. In this section, there are two basic important principles to be introduced and explained: distributive real-time systems (DRTSs) and controller area networks (CAN). These new concepts, vital to design of both robots, will now be elucidated in the following sections.

1.2.1 Distributive Real-Time Systems (DRTSs)

The agricultural ground robots further developed in this thesis are classic examples of what are often referred to as distributive real-time systems (DRTSs). Thus, this first section will be dedicated to giving the reader a brief but meaningful overview of distributive real-time systems and the consequential implications that arise from such systems.

First, before distributive real-time systems can be discussed, it is important to address the notion of a real-time system and its distinction to that of just a simple (time-unrelated) system. In both cases, the systems will perform some type of operation on the given input to produce an output. However, in a real-time system, the correctness of the system depends not only on the logical computational result, but also on if the result was computed within a certain specified time frame [4]. Thus, a real-time system is evaluated on the timeliness of the computed result and not just the result itself. This introduces a wide array of design concerns that need to be taken into account when designing a real-time system as there are many different factors that can affect the timeliness of such a system.

First and foremost, at the lowest system-level abstraction layer, the timeliness of a real-time system is constrained by the physical speed limitations of the selected components in use that comprise the system. Capacitors and transistors are the central components being alluded to here, as these are the circuit elements that possess many time and speed specifications that stem from the component's physical design and construction [5]. Additionally, for digital electronics, the implemented hardware design governing the combinational logic and thus the signals' propagation paths from the input to the output has obvious direct ramifications on the overall speed of the system as well. The hardware circuity could be a physically designed circuit implemented onto a PCB. This would encompass both integrated circuitry, which is self-contained in the selected integrated circuit (IC) chips being used in the electronic design, and discrete circuitry (individual electronic components), which is commonly used for the external supporting circuitry needed for the selected ICs being used on the designed and developed PCB. Alternatively, the combinational hardware could also be synthesized from a written VHDL program implemented onto a field programmable gate array (FPGA) board. Finally, in electronic designs where microcontrollers are employed, the overall speed of the system is also dictated by the developed algorithms in the software which are implemented onto the microcontrollers and perform all of the necessary computations to produce the desired result for the application. Thus, there are many design concerns that the engineer must consider and account for when designing a real-time system in order to

ensure that all the computational results are completed in a timely manner, before the assigned deadlines. The time deadlines are assigned according the real-time system's specific application. Meeting these allocated target times ensures safe operation of the entire system [4].

Now that the reader has been briefed with the notion of a real-time system, the next step is to examine a distributive real-time system (DRTS). If a real-time system is said to be distributive, the system includes multiple subsystems which are all inner-connected together [6]. A DRTS thus, links together a number of different computing entities, each of which, are working on a particular subtask that contributes to accomplishing the broad overarching goal of the entire system. DRTSs are observed today in a vast array of sectors and have a wide range of applications. Aviation, consumer electronics, medical equipment, nuclear power plants, telecommunications, and obviously agriculture machinery are just a few of the many examples where a DRTS is employed and makes up the backbone of the system design architecture [6].

In a DRTS, there is a complex problem that the system as a whole is designed to solve. The problem often involves many separate computational tasks and will demand the need for many different types of input modules (joysticks, receiver modules, various types of measurement sensors, etc.) and output components (motors, actuators, amplifiers, LEDs, LCDs, etc.). For this reason, the system will get broken down into several subsystems, thus allowing the complex problem to get subdivided into several smaller computational tasks, where each subdivided task can then get assigned to a particular subsystem. Usually a microcontroller or a FPGA is selected to be used for each subsystem and can handle all of the processing and computational requirements, specific to the subsystem's allocated task. Each subsystem is then assigned to its appropriate input and output modules, determined by the task that was issued to it. A central communication bus, which inner-connects all the subsystems together, then allows for all the subsystems to be in complete communication with one another. In this way, a DRTS can concurrently handle a set of sub-processes in real time, where each sub-process constitutes to a small portion of the

overall complex computational task corresponding to the problem or job the DRTS was designed to handle. Figure 1.1 presents an illustrative example of a simple distributive real-time system.

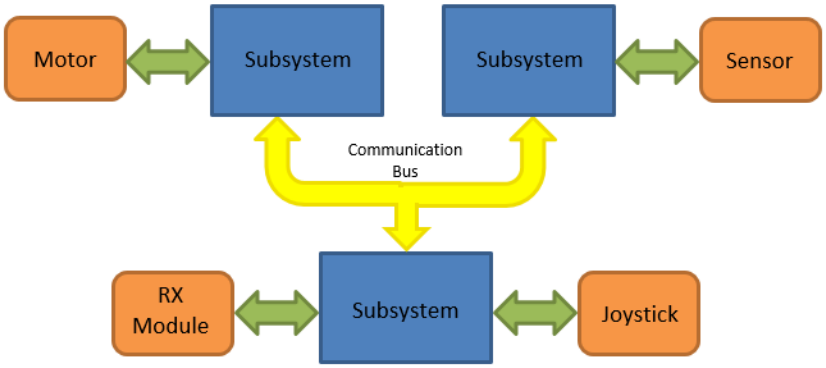


Figure 1.1 - Example of a Distributive Real-Time System

The next topic of discussion involves examining the communication medium in use for the DRTS. Since CAN was selected as the communication protocol for these agricultural robots, the next section gives a light introduction into CAN where the major, most important concepts are addressed and explicated.

1.2.2 Controller Area Network (CAN)

The pervasive use of CAN in agriculture machinery warrants a formal discussion of this network technology. The CAN bus is a robust communication protocol standard that allows microcontrollers and other electronic control units (ECUs) on the bus to communicate back and forth with one another [7]. It was originally designed with the intention that its main application would be in the automotive industry, but it quickly found its way over to the aviation and agriculture sectors as well due to its low cost, flexibility, portability, and rugged design structure, to highlight just a few of the many advantages it possesses in embedded control machinery applications.

The CAN bus is to an automobile what the nervous system is to a person. The nervous system enables all parts of the body to communicate with each other which is analogous to how the CAN bus allows all of the ECUs on an automobile to talk back and forth with one another. Thus, in distributive real-time

systems such as the automobile example, the CAN bus forms the backbone of the entire embedded system.

For a CAN-based DRTS, all of the components or ECUs (referred to as "subsystems" in the previous section) that comprise the DRTS are physically connected to the CAN bus. These components are commonly referred to as *nodes* [5]. For a basic illustration of a DRTS where the CAN bus is utilized as the communication medium, refer to Figure 1.2.

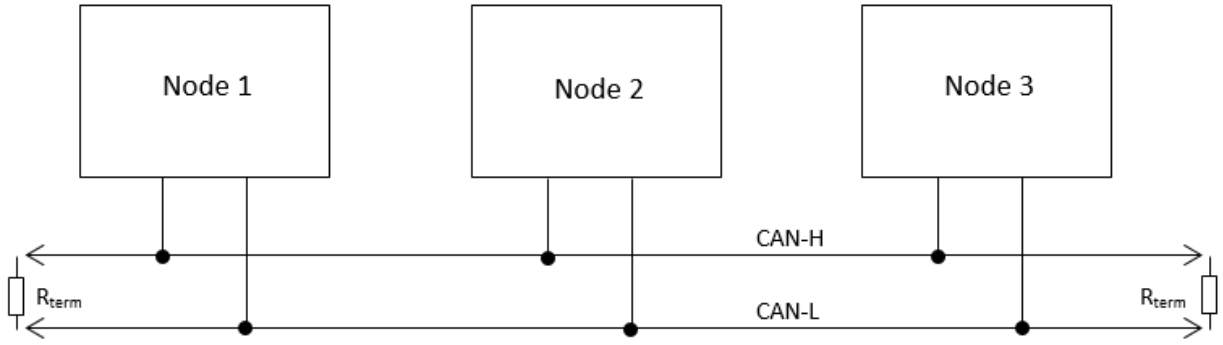


Figure 1.2 - Distributive Real-Time System using the CAN protocol

As implicated from the figure above, the CAN bus employs a differential signaling scheme. When the bus is idle, both the low and high CAN lines carry 2.5 V with a differential voltage of 0 V [8]. When data bits are being transmitted, the CAN high line goes up to 3.75 V while the CAN low line drops down to 1.25V, thus establishing a 2.5 V differential voltage between the two lines [8]. As is common for differential signaling schemes, these two CAN cables are typically assembled in a twisted pair configuration to minimize the effects of crosstalk and electromagnetic induction between the two wires. Additionally, this ensures that any noise that happens to be present on either of the cables will be equally imposed onto both wires, thus effectively making the noise common-mode where it will then be subtracted out when obtaining the differential voltage. Finally, both ends of the CAN line are terminated with a terminating resistor [8]. The resistor value is chosen such that its impedance will match the characteristic impedance of the transmission bus which will help mitigate signal reflections and optimize signal integrity.

Thus, in a CAN-based DRTS, the CAN bus becomes the sole medium that allows the message or CAN frames from one node to be sent out onto the bus and then processed and received by another node. Thus, this introduces the issue of bus contention among the nodes, where multiple nodes may be simultaneously trying to transmit a message across the bus. In order to understand how this problem is handled, a closer examination of a single CAN message packet is required.

1.2.2.1 *CAN Message Frame*

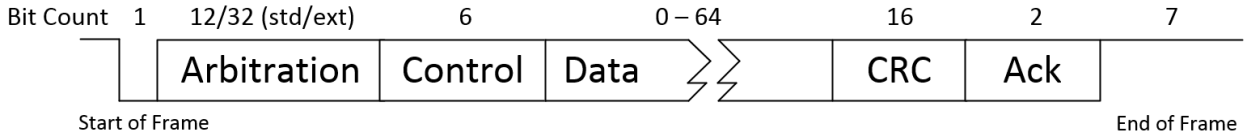


Figure 1.3 - CAN data frame (Troyer, 2017)

A single CAN message frame can be broken into the specified fields seen above in Figure 1.3. The CAN packet structure is very similar to that of other serial protocols. The packet contains a header, payload, and the actual data content of the message. A description of each field as well as its function is offered below.

- *Start of Frame* – Denotes the start of a CAN frame transmission
- *Arbitration* – A unique identifier for the message that determines its priority
- *Control* – Denotes of the byte length of the data to be transmitted
- *Data* – The actual data to be transmitted (length determined by control field)
- *CRC* – Cyclic redundancy check used for error detecting
- *Ack* – Transmitter sends a recessive (1) to which a receiver can assert a dominant (0)
- *End of Frame* – Denotes the end of the CAN frame

1.2.2.2 *CAN Bus Contention Problem*

The one field of particular interest in regards to solving the bus contention problem is the arbitration field where the CAN message is given a unique ID specific to its CAN contents. The ID will inform all the

other system nodes about the contents of the CAN message [5]. Thus, the ID will allow each node to know whether or not the associated CAN message is of any of use to that particular node. If it is, the node will know how to handle the message and what to do with the data. This idea is illustrated below in Figure 1.4.

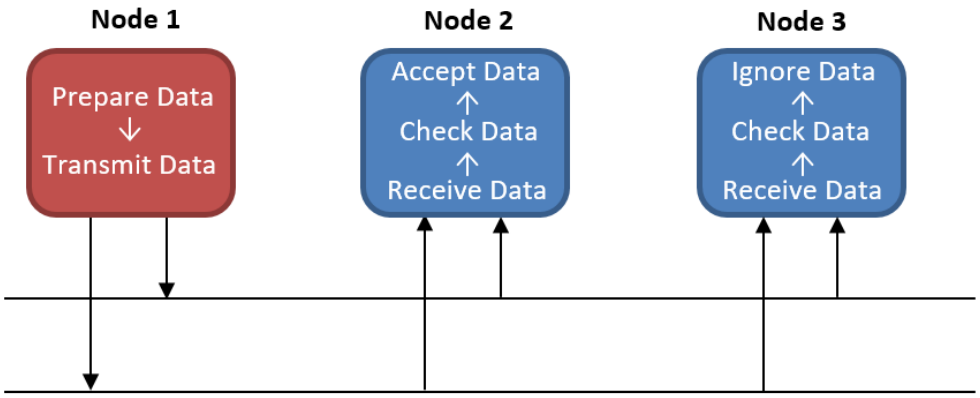


Figure 1.4 – System nodes accepting or ignoring transmitted CAN data based on the ID field content

Moreover, the ID can also be used to determine the priority of the message [7]. If two or more nodes transmit their start of frame bit simultaneously (i.e. multiple nodes attempt to take hold of the bus), a node contending for the bus will give up the arbitration process when it detects a dominant level (0) when it is attempting to send a recessive (1) [5]. The nodes will continue to relinquish hold of the bus through this process until one node is left. The one remaining node then seizes hold of the bus and is able to transmit its message [5]. The process then repeats as before for the remaining nodes that still have messages to send [5]. Since the arbitration field is thus transmitted in order of the most significant bit (MSB) to the least significant bit (LSB), we can say that the ID field with smallest numerical value is given the highest priority while the ID field containing the largest value is given the lowest priority. Thus, a systems engineer can assign the appropriate ID fields to each unique CAN message in the firmware according to this established hierarchy.

1.2.2.3 *CAN System Design*

As it turns out, there is a more systems conscious approach to CAN bus management that does not involve reliance on the data frame ID fields for access to the bus. There are two common approaches to use to handle CAN bus communication when designing and implementing an embedded system onto a DRTS – event triggered systems and time triggered systems. As each system design technique possesses its own unique set of challenges, drawbacks, and advantages, being able to distinguish the difference between the two is essential as this knowledge will allow the engineer to pick the appropriate, most suitable system for the given problem at hand.

1.2.2.3.1 Event Triggered Systems

An event triggered system, as the name implies, is driven by events. The system is designed such that a certain specified set of conditions will trigger the transmission and reception of certain messages which in turn, will invoke the execution of a particular task [5]. An example of such a system could be a very simple obstacle avoiding robot. Say the robot is equipped with two separate microcontrollers, motors, and a simple vision sensor placed at the front of the machine. The first microcontroller is responsible for processing the vision sensor data, while the second microcontroller commands the motors that drive the robot forward. For the sake of discussion, assume the system was designed such that if the vision sensor detects an obstacle within six inches of the vehicle, the microcontroller on the sensor node will send a "stop" message to the motor node to inform it that an obstacle has been detected and that the robot should stop moving. This example illustrates an event triggered system where an obstacle being within six inches of the robot serves as the event that triggers the transmission of a message to the microcontroller controlling the motors, which in turn, causes the robot to stop moving. Otherwise, no transmission will occur and the robot will continue moving forward. Thus, event triggered systems unfold dynamically during program runtime and are directed by the occurrence of asynchronous events outside the system that transpire over time.

The major advantage with event triggered systems is their ability to react promptly to asynchronous external events, since tasks are only executed when the appropriate event has occurred [9]. However, this comes at the expense of system solidity, since correct node operation in a DRTS is then contingent on external dependencies [5]. That is to say, reliable CAN bus operation in event triggered systems is entirely dependent on the successfulness of the transmission and reception of messages by the associated system nodes that constitute the DRTS. Moreover, it assumes that each node is always operating correctly. In a more complicated, large-scale DRTS, where there exist multiple microcontrollers, each concurrently executing an array of tasks and communicating back and forth among one another, there are often many events that dictate what message is be transmitted and then what tasks need to be executed. If any one of those external dependencies is not met (i.e., a task is not correctly executed by any one of the nodes or a message is not properly transmitted or received), the system will suffer critical system breakdown where the DRTS will no longer be able to respond appropriately to its dynamic environment. Thus, event driven systems are generally not very robust and are more susceptible to system failure. These systems might be adequate for relatively simple systems (where system failure would be less likely) or smaller scale solutions (where system failure is relatively inconsequential). However, these systems are understandably never used in the automobile or aviation industry where system failure would have catastrophic implications.

1.2.2.3.2 Time Triggered Systems

A time triggered system is a more robust alternate method and is thus, generally the more preferred approach when reliable CAN bus communication and proper system operation is crucial for the machine in operation. In a time triggered CAN (TTCAN) system, a defined schedule is assigned for when each CAN message is allowed to be transmitted out onto the bus [5]. Each message is given an allocated time slot or *time window* for transmission where the assigned length of time is typically the same for each message [5]. In this way, a time triggered system is able to make aperiodic or sporadic signals, periodic in nature, since each signal, whether periodic, aperiodic, or sporadic, has an assigned time window within

the cyclic messaging sequence for when that message is allowed to be transmitted out onto the bus. Thus, time-triggered systems are inherently deterministic in nature which makes for a more rugged system design architecture.

If for whatever reason, a node misses the time window for the transmission of its associated message, or say an event triggers the transmission of a message whose time window in the message sequence has already passed, it will have to wait till the next cycle for message transmission. In most cases, this does not present any problems or cause need for concern since the transmission frequency is usually sufficiently fast to where experiencing such a delay would be negligible. This transmission cycle, which encapsulates the time it takes for the transmission of all the messages in the system, is commonly referred to as the *basic period* [5]. The basic period thus, dictates the transmission frequency of the DRTS, where its period needs to be at least as long as the number of time widows needed to account for all of the system messages but it could also be longer (BP >= $N_{messages}$*TW). Figure 1.5 gives an illustration of what has just been described.

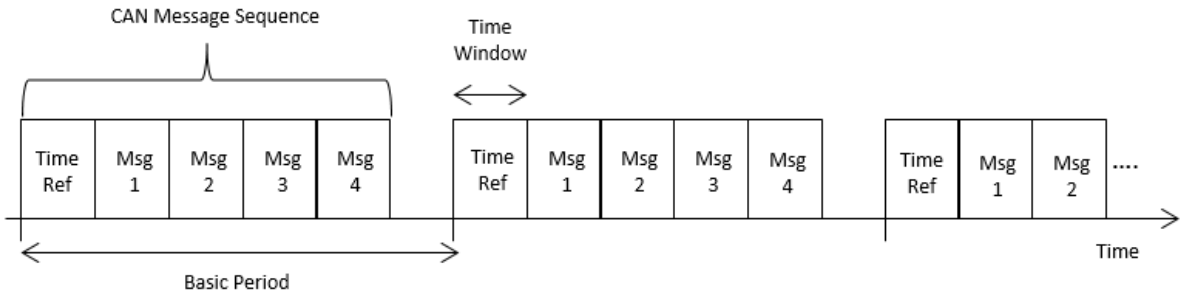


Figure 1.5 - Example of a TTCAN messaging sequence in the time domain

This sense of node redundancy or *replica determinism*, characteristic of the time triggered system design approach, eliminates the node contingency dilemma that debilitates event triggered systems, since proper node operation is now determined by the statically defined cyclic schedule, which is completely independent of the individual operations being executed by each of the existing system nodes. Thus, this results in enhanced fault tolerance performance and thereby offers a more dependable, robust solution to

ensuring proper operation of the DRTS [10]. This improvement in the system's reliability however, comes at the expense of slower system response times and reduced flexibility and scalability when looking to modify or expand the DRTS [10]. Thus, the systems engineer must consider the system complexity and the intended application when deciding on which design architecture to employ. The integration and implementation of event triggered messages onto a time triggered system will be exemplified and thus, better understood, in Chapter 4, where an obstacle avoidance scheme (event triggered in nature) will be implemented onto a time triggered system.

1.2.2.4 *Time Synchronization*

Another important design concern, specifically for TTCAN systems, that should be briefly addressed is the notion of time synchronization. Every microcontroller is governed by an oscillator which provides clock pulses which are required for the synchronization of internal events within the microcontroller [5]. The implication here is that since the entire DRTS (with is likely comprised of multiple microcontrollers) operates on time-triggered schedule for collision-free communication, each node in the system needs to be continuously updated on the "correct" time to ensure proper operation and to avoid clock drift among each of the system nodes. Clock drift is a well-known phenomenon in electronics where there exists slight deviances in the oscillator's frequency due to internal disturbance (such as aging), external disturbances (such as temperature changes), and imperfections and impurities in the crystal [4]. These deviances in the oscillator's clock frequency become more apparent over longer periods of time if left unaccounted for. To compensate for this phenomenon, one node will get designated as the "master clock" where the microcontroller's associated oscillator will serve as the centralized clock source for the entire DRTS. The master clock node will then transmit a time reference message at the beginning of every basic period. The time reference message will be processed by all the remaining nodes in the system so that they are each continuously updated or "corrected" on the global time (i.e., the master node's oscillator clock) at the beginning of every transmission cycle, and will therefore know the correct point in time within the configured bus schedule to transmit their own associated messages. This principle was hinted at in Figure

1.5 which illustrated that the time reference message served as the lead off message in the transmission of the CAN messaging sequence.

1.3 Agenda

In this section, the agenda of this thesis will be presented. It should be noted that this thesis is a continuation of a former graduate student's work. The graduate student had developed and worked on two separate agricultural robots. Both of which, were tailored to different applications in the agriculture realm. This thesis focuses on the further development of these two agricultural ground machines. To distinguish between the two, the first robot will be referred to as an unmanned agricultural ground vehicle (UAGV) from this point on in the document. Chapters 2-4 pertain to the past and present work that has been completed on this ground vehicle. The second robot that was further developed in this thesis is a robot that will be denoted as the Inter-row or Row Follower. All relevant information pertaining to this robot in regards to its system design and functionality can be found in Chapter 5. An outline for the thesis agenda is given below.

1) Chapter 2 – Summarize and give a top level overview of the unmanned agricultural ground vehicle (UAGV) that was developed. This includes discussing its general system architecture, hardware architecture, firmware design, and general functionality.
2) Chapter 3 – Discuss and demonstrate the development of GPS waypoint navigation and its implementation onto the UAGV.
3) Chapter 4 – Discuss the principles of obstacle detection and avoidance and how this technology was integrated onto the UAGV.
4) Chapter 5 – Introduce the Inter-row robot which employs a sensor based autonomous navigation approach for field navigation. Its design, functionality, further development, and intended application will all be discussed in detail.
5) Chapter 6 – Conclusion & Outlook
6) Chapter 7 – References

2 UAGV Overview

In this chapter, a high level overview of the unmanned agricultural ground vehicle (UAGV) will be given. The goal of this chapter is to give the reader a basic understanding of the UAGV's general design and operation. Since this thesis is an extension of a former graduate student's work, a summary and basic outline of his work will be presented, supplemented with a brief introduction to the additional work and development that was implemented onto the vehicle. This will allow the reader to walk away with a technical understanding of the robot's basic design methodology so that in the subsequent chapters where newly designed nodes or nodes that were further developed will be explained in greater detail, the reader will be already well-rehearsed and acquainted with the robot's general system design and operating principles and should therefore be able to quickly understand and follow these new developments.

2.1 System Architecture

The platform this UAGV used was manufactured at the Agricultural Machinery and Research Laboratory on the campus of the University of Kentucky [5]. A 24 V DC motor (Model NPC 41250, NPC Robotics Mound, MN) was selected as the motor to be fitted onto the chassis. The motor was then coupled to a differential rear axis, which provided minimal tire skid [5]. The platform used Ackerman steering pinions, where a 24 V linear actuator (Model 85915 Motion System Corporations, NJ) was used for electronic control over the steering angle of the front wheels [5]. Additionally, the actuator had a built in potentiometer which allowed for positional feedback. Two 24 V lead acid batteries were used to provide power to the robot. Power was distributed via 80 A circuit breakers and a screw terminal array which was located underneath the batteries. The chassis for the UAGV, installed with the basic hardware components just described, is depicted below in Figure 2.1. The chassis served as the development platform for embedded system architecture prototyping for agricultural machinery and allowed for the evaluation of implemented feedback controllers for autonomous robot navigation.

Figure 2.1 - Robot chassis with batteries, motors, and E-Stop switches (Troyer, 2017)

2.2 Distributive Real-Time System Overview

As alluded to earlier, the UAGV employs a distributive real-time system architecture. Such systems allow for scalability, maintainability, system modularity, and firmware simplification where software tasks can be broken down and allocated appropriately to each of the existing system nodes [5]. In the DRTS, there exist multiple nodes (or subsystems), each having their own assigned tasks, which constitutes to accomplishing the overall goal of the entire system. Additionally, each node or subsystem has their own associated set of modules, which is specific to the type of task each node was designed to perform and execute. This particular DRTS predictably uses CAN as the communication medium, which inner-connects all of the subsystems together, thus allowing for complete communication among all the system nodes. In the UAGV's current state of development, there exist five nodes, each with their own set of accompanying modules necessary for accomplishing their assigned tasks. Figure 2.2 below offers an illustration of the CAN based DRTS for the UAGV.

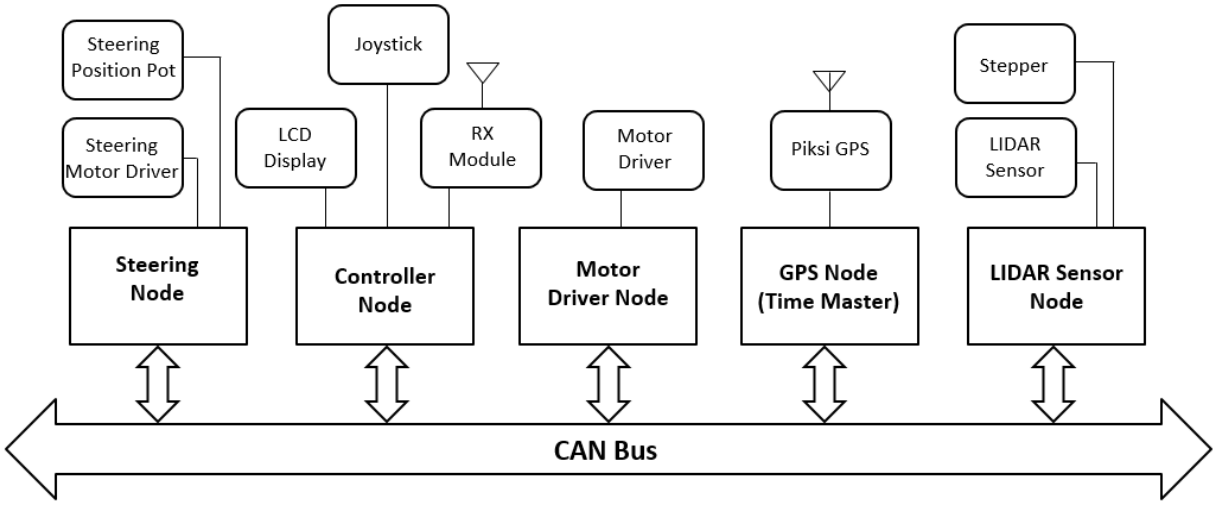


Figure 2.2 - Distributive Real-Time System of the UAGV

1) *Controller Node* – Acts as the CPU of the entire system. Facilitates the state machine of the UAGV, receives key fob commands (via the RX module) for state machine switching, and monitors and processes the joystick commands (when operating in manual mode). Furthermore, the node is responsible for receiving and processing all of the LIDAR, steering angle position, and GPS CAN data needed to run the control algorithm to perform all the appropriate computations for navigation (when operating in autonomous navigation mode). The calculated controller outputs are then transmitted as a CAN control feedback message to both the motor driver and steering nodes.
2) *GPS Node* – Provides the robot's GPS position data needed to derive the control state signals.
3) *Motor Driver Node* – Receives and executes speed commands transmitted from the controller node.
4) *Steering Node* – Receives steering angle commands from the controller node and provides positional steering angle feedback data to the controller node via the built in potentiometer on the linear actuator.

5) *LIDAR Sensor Node* – Processes the LIDAR data to scan for and detect any obstacles or hazards in the robot's proximity and computes and transmits the required steering angle to the controller node to avoid the detected obstacle(s).

It should also be noted that, while not shown in the figure, the robot was designed with a d'sub connector attached at the end of the CAN bus such that a CAN data logger could be easily added to the system for debugging, troubleshooting, and system data logging and collection purposes. The data logger (Kvaser Memorator Pro) would essentially then act as the sixth node on the bus and allowed for all of the processed CAN data to be logged and saved. The data logger is depicted below in Figure 2.3.

Figure 2.3 - Kvaser Memorator CAN data logger (Retrieved from https://www.amazon.com/Kvaser-Memorator-Pro-2xHS-v2/dp/B07N1W32L8)

Figure 2.4 depicts the completed UAGV, where all of the system nodes discussed are shown integrated onto the robot chassis. As indicated in the figure, small, compact enclosure boxes were selected to be used to house all of the associated hardware for each of the developed system nodes. This would include the print circuit board (PCB), the microcontroller(s), and the external sensors and modules that each node encompasses.

Figure 2.4 - Fully assembled UAGV with all of the system nodes that constitute the DRTS

The CAN based DRTS for the UAGV will be discussed in greater detail in the subsequent sections and chapters, as this section was intended to just serve as a light, high level introduction to the UAGV's DRTS.

2.3 CAN Hardware Implementation Overview

In this section, an overview of the hardware architecture for the UAGV will be given. The selected microcontrollers and the PCBs designed by the previous graduate student will be briefly discussed. In this section, it will become clear from a hardware perspective, how all of the system nodes were designed and implemented onto the CAN bus.

2.3.1 Selected Microcontrollers

The former graduate student decided to use microcontrollers from the ST microelectronics family for the system node development of the associated DRTS which allowed for CAN bus implementation. Thus, these inexpensive STM32 microcontrollers come with all of the CAN features necessary for complete communication across a CAN bus. STM32s are essentially development boards, complete with a debugger/flash downloader device that interfaces the target microcontroller to an integrated development environment (IDE) on a PC, allowing for rapid firmware development, where the algorithms and implemented software could be easily and efficiently tested and evaluated.

Two different development boards were chosen to serve as the system nodes for the CAN based DRTS. It was decided that the STM32f308k8 microcontroller would be used for both the motor driver and steering nodes. The student selected this microcontroller for these two nodes due to its small form factor, which allowed it to fit inside the small enclosure box where both nodes also required a motor driver board [5]. One driver module was used to drive the DC motor on the UAGV for translational movement, and the other was utilized to drive the linear actuator which controlled the steering angle of the front two wheels. The selected motor driver module is a 2x12 32 A Sabertooth and can be seen below in Figure 2.5.

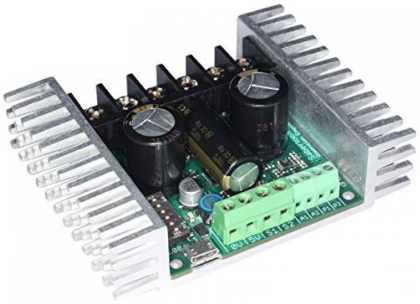


Figure 2.5 – Sabertooth dual 32 A motor driver (Retrieved from https://www.amazon.com/Sabertooth-Dual-32A-Motor-Driver/dp/B00O1722NG)

The STM32f308k8, interfaced to the driver boards, included a 72 kHz, 32 bit ARM Cortex M4 processor, and contains all the basic set of peripherals as standard for ST Micros [5]. The other development board used for the development of the UAGV was the STM32f446RE microcontroller. The student selected this development board for the controller and GPS node due to its fast processor clock, as it contains a 180 MHz ARM Cortex M4 processor [5]. Additionally, the STM32f446RE development board was selected to be used for the newly implemented node to the system - the LIDAR sensor node. This development board was tailored for high performance applications and was thus, delegated to handle the more mathematically intensive and computationally demanding tasks. Pictures of both development boards can be seen below in Figure 2.6 and Figure 2.7.

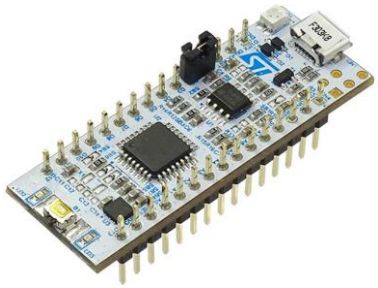


Figure 2.6 - ST Microelectronics microcontroller development board (F303k8) (Retrieved from https://www.iot-jungle.com/shop/carte-cpu/nucleo/nucleo32/nucleo-f303k8/)

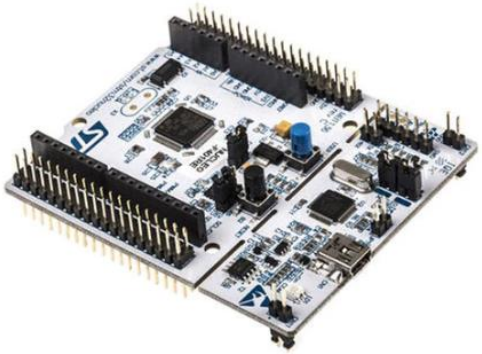


Figure 2.7 – ST Microelectronics microcontroller development board (F446E) (Retrieved from https://www.digikey.ca/product-detail/en/stmicroelectronics/NUCLEO-F401RE/497-14360-ND/4695525)

2.3.2 Board Designs

In order to integrate all of the developed nodes onto a common CAN bus as was shown in Figure 2.2, PCB boards needed to be designed and configured in a way that would allow for the CAN and power lines to inner-connect all of the microcontrollers (or nodes) onto the DRTS. Figure 2.8 and Figure 2.9 show the developed CAN PCBs for both of the two development boards that allowed for all the nodes to be daisy chained together via the connection of their CAN and power lines. Both boards were designed by the previous graduate student. The PCB shown in Figure 2.8 was designed for the STM32f446RE while the board seen in Figure 2.9 was designed for the STM32f308k8. As can be gathered from the figures, the student designed the boards in a way that allowed the microcontrollers to be mounted onto the top of the PCBs so as to give access to all of their functional pins.

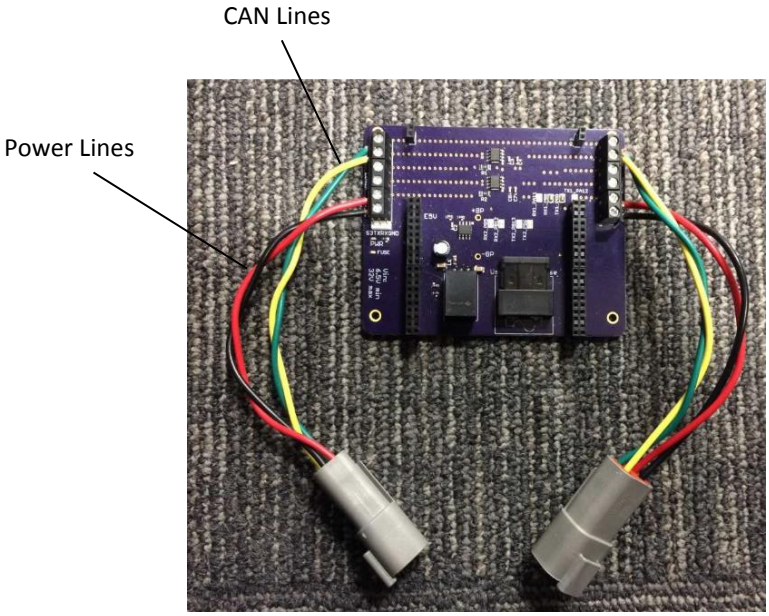


Figure 2.8 – CAN PCB for F446E board

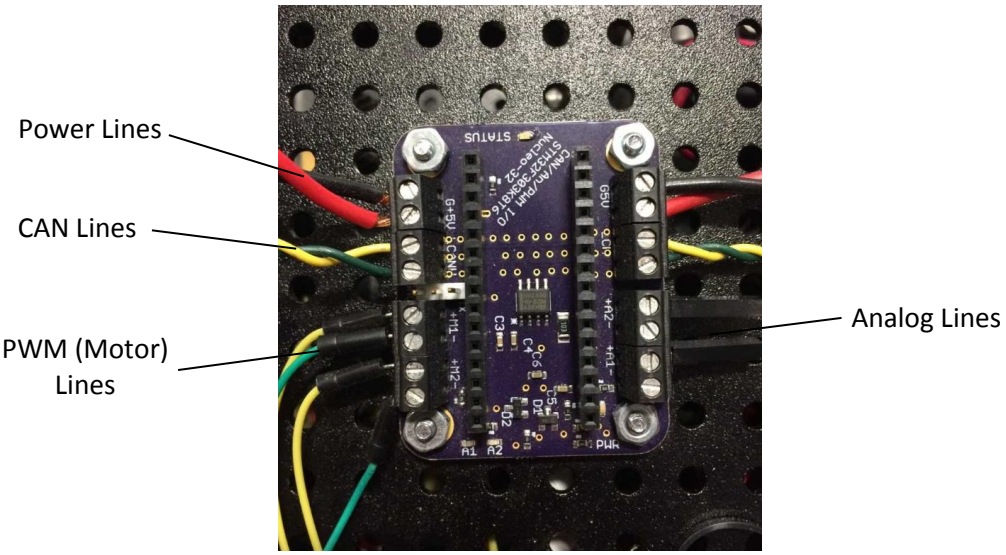


Figure 2.9 – CAN PCB for F303k8 Board

The 446 board included screw terminal blocks that connect to the CAN bus signaling pins as well as the power and ground lines, where the power, ground, and CAN line traces all went directly across the board to the terminal blocks on both sides of the PCB thereby allowing for the external wiring (or daisy

chaining) of these lines across all of the nodes that comprise the system. Vias were also added between these traces to improve the noise immunity of those respective traces on the board [5]. The main components for this PCB include a SN65HVD250 CAN bus transceiver, a 1 amp ATO fuse mount, and a switching power supply module. The fuse was integrated into the power regulator circuitry to provide protection from back EMF from the DC motors used in the steering and motor driver nodes. Some additional status indicator LEDs were also implemented onto to the board to provide visual feedback, thus aiding in debugging and troubleshooting.

For the 308k8 PCB, the board design was similar but with some distinct differences. Like the first board, this PCB also included screw terminals for the CAN bus signal pins in addition to the power and ground lines which again, allowed for the daisy chain connection of all the nodes onto the CAN bus. Additionally, screw terminals were used for providing access to analog input channels and pulse width modulation (PWM) output channels. For the analog channels (which were traced to ADC peripheral pins on the STM32), the student implemented a clamping filter for transient surge protection purposes [5]. The analog and PWM lines enabled the boards to be interfaced to the appropriate sensors, motors, and/or modules according to what was called for in the initial robot design. As the intent of this section was to just give the reader a brief overview of the implemented hardware, a more detailed, in-depth explanation of these boards can be found in pages 22-24 and pages 91-96 of [5].

2.4 CAN Firmware Implementation Overview

In this section, a top-level overview of the UAGV's firmware will be presented. This involves observing the data packing scheme for the CAN messages to be transmitted, addressing the time triggered CAN (TTCAN) bus schedule setup, discussing the CAN peripheral which enables communication among all the nodes on the bus, and finally, examining all the other utilized peripherals in the firmware that enabled the microcontrollers to be interfaced to all of the implemented hardware introduced in section 2.2.

2.4.1 CAN System Messages

The first piece of information to address is the data encoding or message packaging scheme for the CAN messages to be transmitted over the bus. The messages associated with the newly developed LIDAR sensor node are outlined in Figure 2.10 and Figure 2.11, while the messages associated with the work completed by the previous graduate student can be referenced in pages 105-109 in [5].

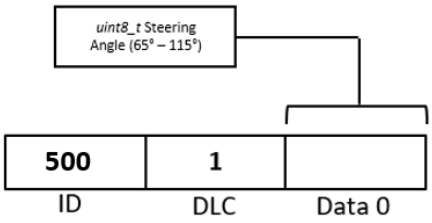


Figure 2.10 - Message packaging scheme for steering message based on LIDAR data

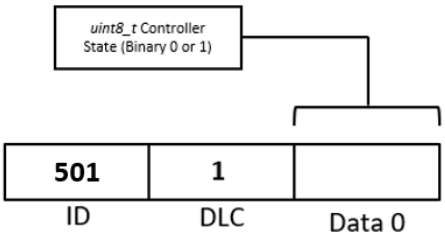


Figure 2.11 - Message packaging scheme for controller state message used to enable/disable the sensor node

Both of the above messages only consist of a single byte of data. Hence, packaging these messages was extremely simple as no bit shifting was required for transmission and reception. Descriptions of each of the system message IDs (not just the newly developed ones shown above) are given below.

- 100 – *Tim Ref Message:* Time reference message transmitted from the GPS node (time master) to keep all the system nodes synchronized and properly updated on the global time so that all messages are transmitted during their allocated time slot.
- 401 – *Latitude Message:* Latitude coordinate transmitted from the GPS node

- 402 – *Longitude Message:* Longitude coordinate transmitted from the GPS node
- 200 – *Position Message:* Northing-easting position transmitted from the GPS node to the controller node.
- 201 – *Velocity Message:* Northing-easting velocity transmitted from the GPS node to the controller node.
- 202 – *Heading Message:* Heading angle of the UAGV transmitted from the GPS node to the controller node
- 206 – *Steering Message:* Steering angle position (of the linear actuator) transmitted from the steering node to the controller node.
- 300 – *Feedback Command Message:* Steering and motor commands transmitted from the controller node to the steering and motor driver nodes.
- 450 – *Data Message 0:* UAGV's X and Y coordinates transmitted from the controller node to the CAN data logger
- 451 – *Data Message 1:* UAGV's heading angle, PSI, and speed transmitted from the controller node to the CAN data logger.
- 452 – *Data Message 2:* UAGV's cross track error, steering angle, and steering command transmitted from the controller node to the CAN data logger.
- 453 – *Data Message 3:* Controller input signal and steering pot ADC transmitted from the controller node to the CAN data logger.
- 500 – *Steering Message:* Steering Angle transmitted from the LIDAR sensor node to the controller node, informing the controller of the required steering angle needed in order to avoid and navigate around the detected obstacle(s).
- 501 – *Controller State Message:* The current operating state of the controller is transmitted from the controller node to the LIDAR sensor node to inform the sensor node on whether or not it should look for obstacles. The sensor node will only scan for obstacles if it is autonomous mode.

2.4.2 TTCAN Architecture

The UAGV employs a time triggered CAN (TTCAN) approach where all the messages are each assigned to a particular time slot in the deterministic periodic schedule. The transmission slot or time window for each system message was chosen to be one 1 mS wide. This time window was selected to provide a 480 ms delay time for the worst case CAN message transmission time of 520 ms [5]. The overall transmission frequency of the system messages shared across all of the nodes is 10 Hz, thereby implying that the basic period of the entire system is 100 ms. The system could therefore accommodate for up to 100 separate system messages. A timer interrupt handler was used to establish the transmission time of the time reference message for time tracking and synchronization across all of the nodes in the system. Naturally, the timer was configured to operate at a 1 kHz frequency to accommodate for the transmission of all the system messages within the basic period. Figure 2.12 illustrates this time triggered setup for the UAGV.

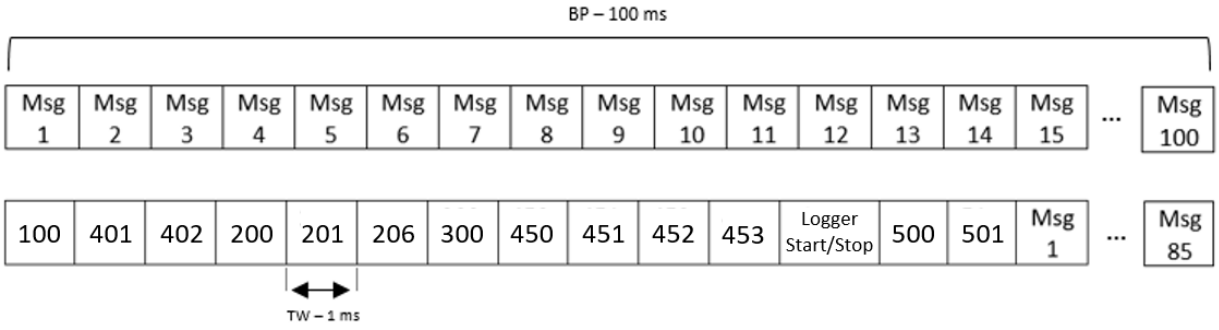


Figure 2.12 - UAGV Time Triggered CAN system message sequence design

The CAN bx peripheral, found in the STM32 microcontroller family, was leveraged to develop the TTCAN infrastructure for the UAGV. The peripheral provides a large number of options allowing for optimal communication for a wide range of different applications. Since this work was done by the former graduate student and is fairly involved, this information will not be covered here since it goes beyond the scope of this thesis. For a complete explanation of all the details regarding how this peripheral was set up and configured, refer to pages 111-116 of [5].

The CAN bx peripheral allows for the implementation of interrupts. An interrupt, when activated, will jump from its current section of program memory to run a short segment of code, where that segment of code is specific to that particular interrupt. An interrupt is usually triggered by some external event outside of the processor. Having this ability from the peripheral was critical as it formed the groundwork for developing the TTCAN infrastructure for the UAGV. Interrupt handlers were used to manage all of the TTCAN related tasks as it pertained to time tracking and synchronization and message reception and transmission. TX and RX ring buffers coupled with interrupt handlers were used for the processing and transmission of system messages across the TTCAN bus. All of the aforementioned CAN messages that comprise the TTCAN bus for the UAGV are each assigned to either a TX or RX ring buffer based on whether the node is receiving or transmitting that message. When a TX or RX ring buffer contains a message for transmission or reception, the appropriate interrupt for the associated microcontroller would fire to transmit or process that CAN message accordingly during the allocated time slot that was pre-delegated to that particular message.

2.4.3 Peripheral Firmware Integration

Interrupts not only facilitated the operation of CAN message reception and transmission, they also governed the operation of peripherals that were implemented to interface with all of the sensors and modules that were illustrated in Figure 2.2. This includes analog to digital converters (ADCs) and direct memory access (DMA) controllers for the joystick and RC remote, and a DMA controller and universal synchronous/asynchronous receiver/transmitter (USART) for the LCD. Figure 2.13 presents an in-depth, top level overview of the developed firmware on the UAGV, where all of the utilized modules and peripherals are shown integrated together in the embedded system design.

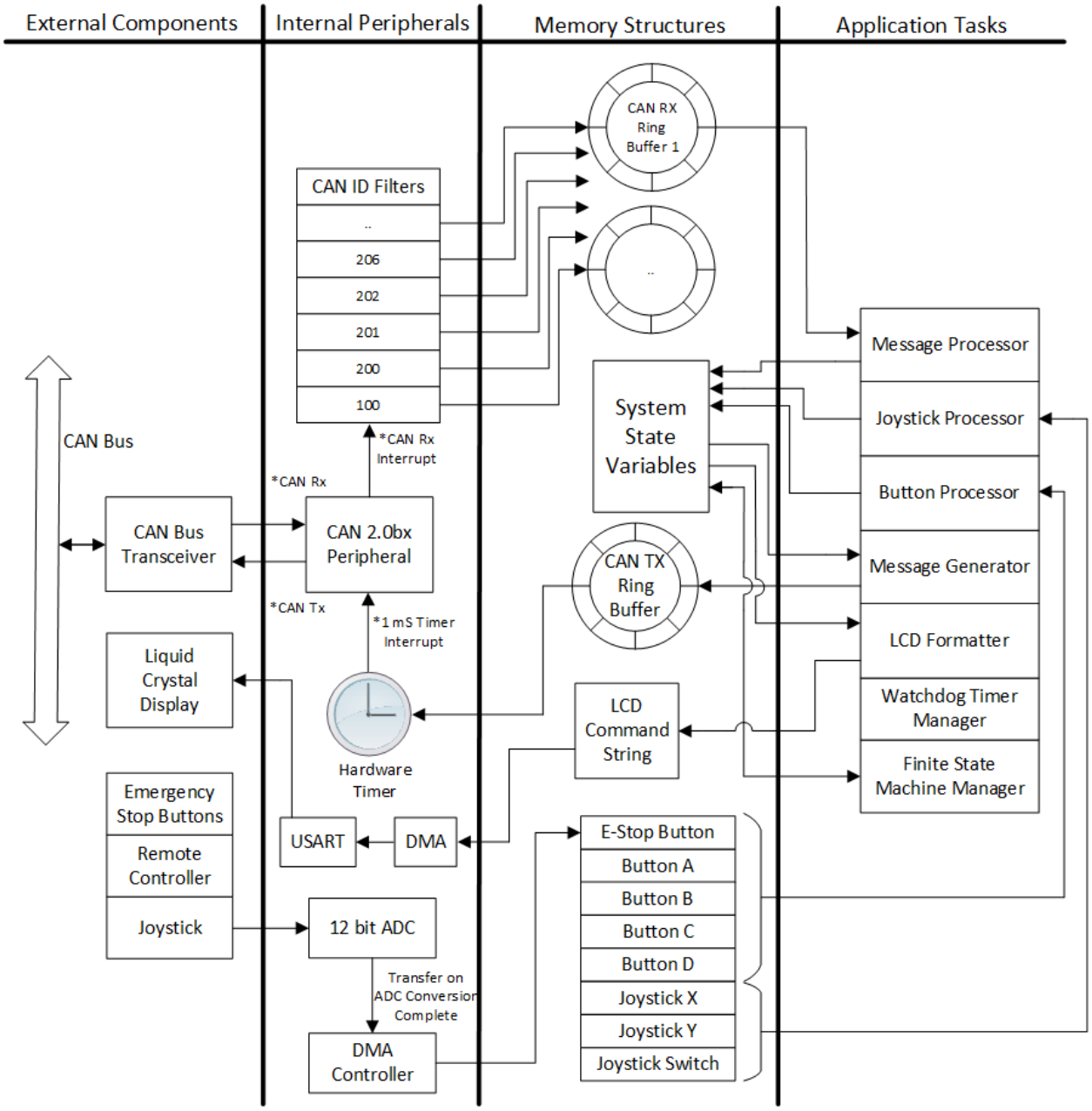


Figure 2.13 - Firmware architecture of the UAGV (Troyer, 2017)

The majority of the information presented in this section was a simply a reiteration of the material addressed in Troyer's thesis, conveyed here in a more concise form. The goal of this section was to give the reader the necessary background information regarding UAGV's firmware infrastructure, highlighting the key concepts and principles, so that in the subsequent chapters, where the firmware is further

developed, the reader will have been equipped with an adequate amount knowledge regarding the UAGV's firmware design to be able to comprehend the supplementary work.

2.5 Finite State Machine System Modes

Now that the reader is briefed on the both hardware and firmware architecture of the UAGV, it is necessary to now address the operational modes of the UAGV, its associated finite state machine (FSM), and the obligatory pieces of hardware necessary for each functioning mode.

A four button key fob remote and a 350 MHz RX module were selected to give the user access and control over the UAGV's functioning modes. Pushing a button on the key fob remote would activate the corresponding pin on the RX module, giving the user remote control over the UAGV. The RX module was naturally implemented on the controller node (as was implicated in Figure 2.2). In this way, the user was able to have control over the FSM layer of the UAGV. Figure 2.14 below shows the key fob and the associated RX module.

Figure 2.14 – Key fob and receiver module for operating mode switching (Troyer, 2017)

Additionally, one of the functioning modes of the UAGV was manual control. To achieve this, a simple analog joystick was selected to be used which allowed the user to have control over the UAGV's movement while in this mode. This enabled the user to manually steer the robot to and from testing sites. A picture of the selected joystick is shown in Figure 2.15. As with the RX module, the joystick was also implemented onto the controller node.

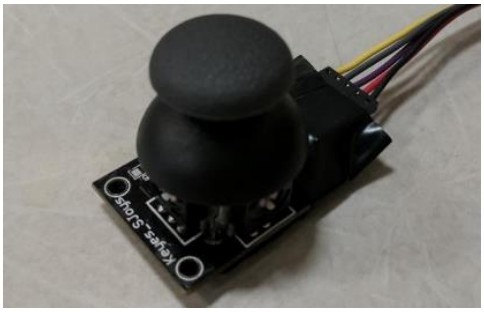


Figure 2.15 – Analog joystick for manual control mode (Troyer, 2017)

Finally, for feedback display, a thin film transistor liquid crystal display (LCD) was installed onto the controller node as well. The LCD informs the user of helpful, relevant information such as the robot's current mode, the CAN bus status, and the GPS state parameters obtained from the GPS node. The LCD can be observed below in Figure 2.16.

Figure 2.16 - Liquid Crystal Display (LCD) for system state visualization (Troyer, 2017)

As already stated, the operating modes of the UAGV are governed by a designed FSM to which the user has control over via the key fob remote. The various states of the UAGV's FSM are given below along with a brief description of each state.

- *Init Mode* – Robot enters this mode when initially turned on. No signals are routed to the steering actuator or the motor drivers in this mode. Waits to receive commands from the user via the key fob switch.
- *Manual Mode Ready* – No signals are routed to the steering or motor drivers yet. Will enter *Manuel Run Mode* upon appropriate user input.

- *Autonomous (GPS) Mode Ready* – No signals are routed to the steering or motor drivers yet. Will enter *GPS Run Mode* upon appropriate user input.
- *Manual Run Mode* – Signals are routed to the steering and motor driver nodes based on the joystick input from the user.
- *Autonomous (GPS) Run Mode (Line Navigation)* – Signal are routed to the steering and motor driver nodes based on the entered GPS waypoint set and the line control algorithm commissioned which drives the UAGV in a straight line towards the second waypoint.
- *Autonomous (GPS) Run Mode (Curvature Navigation)* – Signal are routed to the steering and motor driver nodes based on the entered GPS waypoint set and the curvature control algorithm commissioned which drives the UAGV in a curvature, circular arc towards the second waypoint.
- *Avoid Obstacle Mode* – The obstacle avoidance algorithm computes the necessary steering angle to be routed to the steering node to avoid and navigate around the detected obstacle(s).
- *Emergency Stop Mode* – Halts operation of the robot where signals are no longer routed to the steering and motor driver nodes. Will enter this mode when the corresponding E-stop buttons on the machine are pressed.

A visual representation of the UAGV's FSM can be observed below in Figure 2.17. The figure shows the specific buttons on the key fob remote that will trigger state transitions. Moreover, the diagram also illustrates the different environmental factors that can also dictate whether or not a state transition will occur.

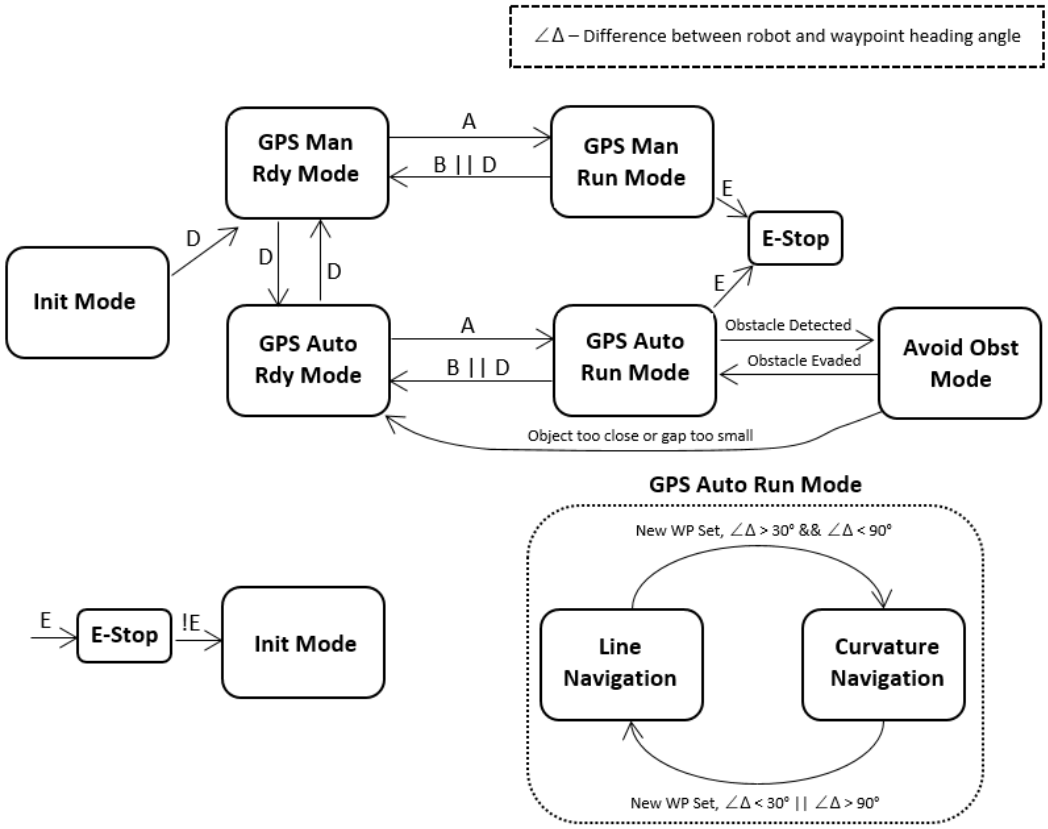


Figure 2.17 - Finite State Machine of operating modes for the UAGV

The two navigation algorithms depicted in the above figure (line and curvature) and the state transitioning that occurs between the two schemes while in autonomous mode, will be discussed further and thus, better understood in the proceeding chapter.

3 GPS Waypoint Navigation

The invention of global positioning systems (GPS) completely transformed the landscape of navigation. This revolutionary innovation allowed for the extraction of the precise geographical location, velocity and associated time of any GPS receiver in the world, provided that there are at least four satellites with an unobstructed line of sight to the GPS receiver [11]. This technology thus allowed for the guidance, control, and navigation of mobile robots [11]. When initially introduced in the 70s, it was only a matter of time before this technology was eventually leveraged and utilized in the military sector, ultimately becoming the core navigation system for military aircraft, vessels, tanks, and personal [12]. However, since then, GPS navigation has found a vast array of other applications on a much broader scope which includes (but is not limited to) aviation, marine, telecommunications, surveillance, financial services, road transportation, and as of recently, agriculture. We even use this technology on a daily basis via the use of our smart phones as navigational aids to help us get to our desired destination points. The development of the GPS has reached a point to where it is now playing an integral role in the life of the everyday consumer.

This technology also has pronounced applications on the agriculture industry. Since robots can now be guided via a GPS, this concept can then be applied to agricultural tasks such as planting, watering, weeding and harvesting, all of which were tasks that were either previously performed by human laborers or by machines being manually operated by humans. As a result, this technology has played a pivotal role in spurring the practical realization and implementation of autonomous navigation practices in the realm of agriculture via the emergence of unmanned agricultural ground vehicles (UAGVs) that can autonomously perform agricultural tasks without requiring any human aid or intervention. Hence, this technology is serving as one of the primary catalysts in regards to the many scalable infrastructural changes the industry is beginning to experience, where GPSs and other innovative technologies are in the

process of gradually being implemented onto field machinery in effort to develop automated and improved crop production methods and field management techniques [13].

In this chapter, the reader will first be equipped with the necessary background knowledge which will provide the framework for the development of the GPS waypoint navigation work that has been accomplished in this project. This information involves addressing the initial state of the UAGV project as it pertains specifically to the GPS waypoint navigation development. It also includes having a brief discussion of the selected GPS module that is being used on the UAGV. This will be followed by a discussion of several of the well-known path following algorithms or control schemes that will be implemented onto the UAGV for GPS waypoint navigation control. The chapter will then offer an in depth discussion and examination of the final implemented embedded system design for the GPS waypoint navigation scheme. Finally, the chapter will conclude with the data obtained from several conducted test runs where the performance of the developed control algorithms will be evaluated.

3.1 Background

As was briefly addressed in Chapter 2, a Swift Navigation Piksi unit was selected to be used as the GPS receiver on the UAGV for the purpose of precision navigation. The unit is a high precision GPS receiver platform with open source firmware and hardware, allowing for the implementation of real-time kinematic (RTK) technology. RTK GPSs are able to achieve a much higher positioning resolution (up to 2-3 cm precision), as the configuration allows for the mitigation of atmospheric errors that the standard GPS units suffer from [14]. For this reason, there has been an increasing amount of interest in employing RTK-GPSs onto farm vehicles to perform precision agricultural related tasks [15]. Figure 3.1 below shows a picture of the current GPS device in use on the UAGV.

Figure 3.1 - Swift Navigation RTK CD-GPS module

This unit is classified as a carrier-differential or pseudo-differential GPS (CD-GPS) receiver. These types of receivers do not give absolute position readings as compared to the standard, traditional GPS units. They require a base station to be set up where the GPS base station then serves as the reference point for all of the GPS readings on the receiver module [16]. Thus, all of the GPS position readings are relative to the location of the base station. This unit was selected to be used as the GPS module for the UAGV due to its high precision capabilities, which was particularly advantageous for the application of precision agriculture, involving navigation-focused tasks.

The initial state of development the originally inherited UAGV was in (with regards to GPS navigation) was that the UAGV would work for simple GPS line navigation which involves using just two GPS waypoints. The control system or path following algorithm on the controller node would process the GPS data transmitted from the GPS node and would then send the appropriate commands to the motor driver and steering nodes to pilot the UAGV from the first waypoint to the second waypoint. The important caveat to note here is that the system currently only works for an input of strictly two GPS waypoints. Moreover, the user must manually stop the UAGV when the second waypoint has been reached as the UAGV currently has no awareness of its position relative to the target waypoint it is approaching. These primary concerns are addressed in section 3.3 where the implemented solution to these problems will be

discussed in detail. But before that discussion can take place, it is important to first overview the various path following algorithms that were implemented, tested, and analyzed during the development of the GPS waypoint navigation scheme

3.2 Path Following Algorithms

There are numerous path following algorithms or control schemes that have been studied and developed in literature over the years for land based navigation problems. In GPS applications, these algorithms determine the route the robot travels to go from one waypoint over to the next. In order for the ground vehicle to honor the calculated path, which is specific to the selected algorithm in use, the model the ground vehicle resembles must be that of a closed-loop system as shown in Figure 3.2.

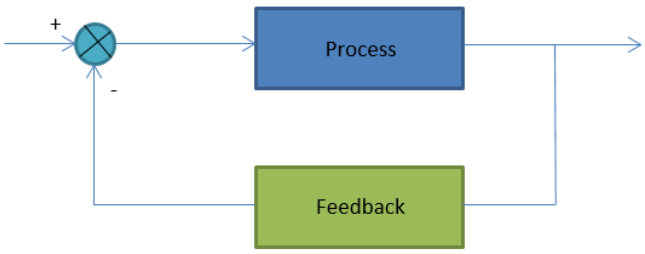


Figure 3.2 - Closed loop system diagram

Closed loop systems form the backbone of the equipment used in industrial control applications as they allow for the regulation of important parameters of interest (temperature, pressure, speed, etc.) [17]. There are a couple of key items that comprise that of a closed-loop system that should briefly be addressed. First, the system consists of the main component which is commonly referred to as the *process plant* [17]. The process plant is simply the system that we are attempting to control [17]. The plant could be, say, an automobile in cruise control or an aquarium with a heater that monitors the water temperature. In this project, the UAGV would be considered the process plant. The second vital element in any closed-loop system is the feedback loop [17]. The feedback loop is the part of the system where at least a portion of the output signal gets fed back to the input of the system [17]. These are the important parameters that need to be regulated. Various types of sensors can be used to obtain readings of these output signals. The

measured output can then get compared with the desired set point value on the input side to obtain an error signal. A controller can then be designed to monitor the error, calculate a control signal to adjust the measured outputs appropriately, and drive the measured error down to zero [17]. Figure 3.3 illustrates a closed loop system with a controller incorporated into the model.

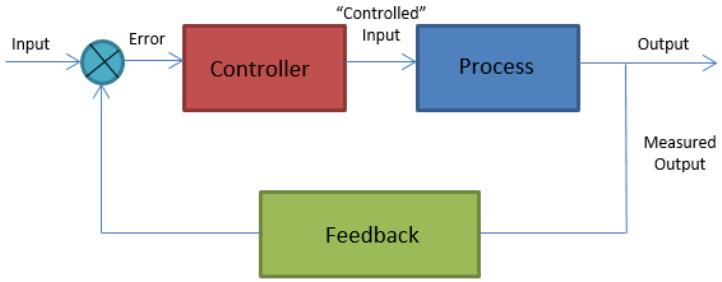


Figure 3.3 - Closed loop system with an added controller

In this way, a closed loop system allows for the regulation of any measured parameter. To relate this specifically to the problem in this thesis, implementing a path following algorithm on the UAGV equates to making the UAGV resemble the closed-loop system model where the output (or feedback signal) is equivalent to the steering angle of the UAGV. The path following algorithm which contains all the necessary computations to calculate the required steering angle can be equated to the controller of the system. In the subsequent section, some of the more prevalent path following algorithms or control schemes used today to solve land-based navigation problems will be discussed.

3.2.1 P Controller

The P controller is probably the simplest, most intuitive control scheme to employ for regulating a system. The controller is a descendant of the preeminent PID (proportional-integral-derivative) controller. The operating principle of the PID controller is that it performs mathematical operations (in the form of proportional multiplications, integrals, and derivatives) on the measured error signal where the error again, is the measured difference between the estimated value and the desired value. The individual results from each of those operations are then multiplied by the appropriately weighted constants and summed together to produce the output control signal that should assist in driving the system's error

down to zero, if the coefficients for each error parameter are properly weighted and tuned. Equation 3.1 below shows the fundamental equation for the PID controller that was just described.

$$u = K_P e + K_I \int_0^t e dt + K_D \frac{d}{dt} e \qquad 3.1$$

These controllers are commonly used in industrial control applications to regulate some important parameter of interest. Each of the three terms plays their own unique role that aids in controlling and optimizing the response of the system. [17] documents various occasions over the past couple of decades where a PID controller served as the guidance system of a mobile robot or tractor in agricultural environments. However, for many applications, one often does not need to leverage all three terms in order to suitably control the system and acquire a reasonable controller response. For this particular UAGV project, the previous graduate student implemented a hybrid P controller model which achieved satisfactory results.

In the P controller scheme, the error is just multiplied by some proportional constant [18]. For the P controller implemented onto the UAGV, there are three proportional error parameters of interest: cross track error, heading error, and steering angle error [5]. The cross error (XTE) is the distance between the robot and the straight line between the two current waypoints, where the error distance is measured normal to the line that connects the two waypoints together. The heading error (ψ) is the measured difference between the robot's current heading angle and the heading angle of the formed line that inner-connects the two waypoints. The heading angle of the waypoint line is often referred to as the bearing angle. Finally, the steering angle error (δ) is the measured difference between the heading angle of the robot and the steering angle of the front two wheels. The cross track error and heading angle were acquired through the GPS module, while the steering angle was obtained via the built in potentiometer on the actuator that controlled the steering angle of the front two wheels. Figure 3.4 presents a visual illustration of each of these error parameters.

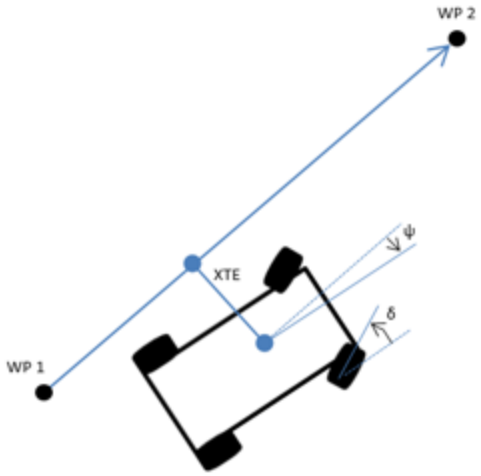


Figure 3.4 - States or error parameters used for the P controller based GPS navigation scheme

These three error parameters collected together, contain all the necessary information pertaining to the robot's current position relative to the desired position. Essentially, it yields all the information regarding how well the UAGV is travelling in the desired straight line between the two waypoints. Thus, the P controller processes all three of the error state signals to generate a control signal which then adjusts the steering angle accordingly to mitigate the measured error parameters, minimizing the overall path error thus, driving the system towards the established line between the two waypoints. Each of these error parameters are assigned a proportional constant which weights each term accordingly, and the individual terms are then all summed together to yield the control signal, which is the signal that gets routed to the linear actuator to appropriately control the steering angle of the front two wheels. The equation governing the P controller can be observed below in Equation 3.2.

$$U = (K1 * XTE) + (K2 * \psi) + (K3 * \delta) \qquad 3.2$$

In a properly tuned control scheme, these three error parameters will be driven down to zero. Thus, if the P controller worked perfectly, the robot would travel in a perfectly straight line between each waypoint. Thus, in this GPS waypoint navigation application, the robot will instantly turn once it reaches its current target waypoint so that robot's new heading angle will align with the newly updated target waypoint. This turns out to be one of the drawbacks of this path following algorithm as the optimal path for the vehicle to

travel to reach each waypoint is often not a straight line [18]. Additionally, the algorithm can be prone to oscillation (i.e. swerving about the desired straight line path) [18], though this phenomena can usually be mostly alleviated with properly tuned proportional weights. Despite the mentioned shortcomings, the algorithm presents a relatively simple solution that can serve well for land based navigation applications where smooth path turning abilities is not a primary or fundamental concern.

3.2.2 Pure Pursuit

Another common path following algorithm that has been thoroughly researched and explored is the Pure Pursuit method. The algorithm essentially calculates a curvature route (or circular arc) between the robot's current position and the current target waypoint it is approaching [18]. The operating principle of the algorithm is actually quite trivial as understanding the inner-workings of the algorithm only requires a general conception of some basic level geometry. The straight line formed between the two waypoints forms the base leg of an isosceles triangle. This is also commonly referred to as the *chord length* [19]. The two other equilateral legs of the triangle form the radius of a circle. The arc or the curvature path that the Pure Pursuit algorithm calculates is simply just a slice or portion of that circle, as the circle contains both GPS coordinates (of the waypoint pair) along its circumference. Figure 3.5 below illustrates this idea below where all of the relevant parameters are labeled accordingly.

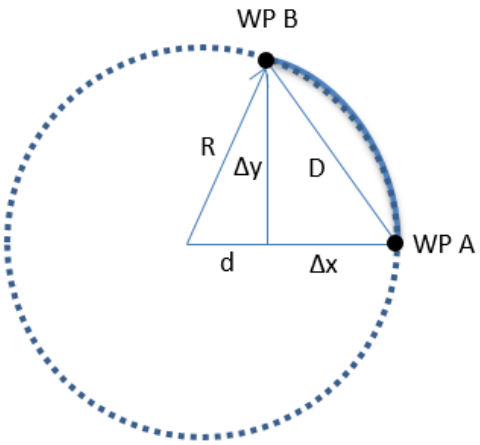


Figure 3.5 - Visualization of the geometry for the Pure Pursuit algorithm with all the important parameters labeled

The D parameter (chord length) shown in the figure can be calculated by utilizing Pythagorean's theorem. Employing some trivial geometrical triangle equations and identities and performing some basic algebraic manipulation, the radius corresponding to the circular curvature of the waypoint set can be acquired from Equation 3.3. If the reader is interested in all the mathematical details regarding how the equation below was obtained, a complete step by step derivation of the entire process is offered in [19].

$$R = \frac{2\Delta x}{D^2} \tag{3.3}$$

Once the radius has been obtained, the steering angle can then be computed which will allow the UAGV to follow the calculated curvature route. Figure 3.6 illustrates this idea while Equation 3.4 represents the mathematical expression for this computation. The equation corresponds to the initially calculated steering angle value that should in theory, enable the UAGV to track the calculated arc that will drive the vehicle towards the second waypoint in the active waypoint pair.

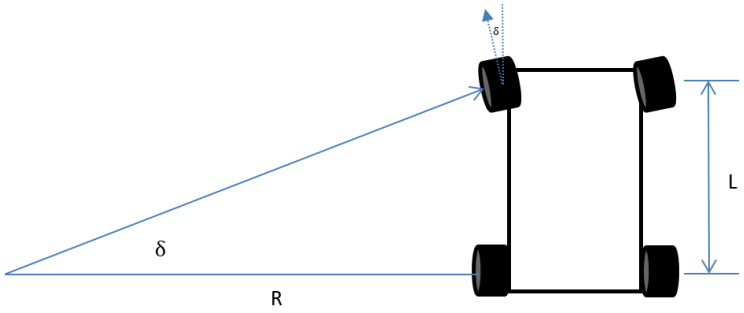


Figure 3.6 - Trigonometric identity revealing the computed steering angle used in the Pure Pursuit algorithm

$$\delta = \tan(\frac{L}{R}) \qquad 3.4$$

Hence, obtaining the steering angle for the robot just requires knowledge of the tangent trigonometric identity once the corresponding radius for the curvature has been acquired. It should be noted that the radius parameter, *R*, shown in Figure 3.6, corresponds to the radius for the circle shown in Figure 3.5. The steering angle shown in Figure 3.6 corresponds to what the steering angle of the robot needs to be in order for the UAGV to follow the curvature arc of the circle shown Figure 3.5. Thus, this steering angle can be transmitted to the steering node as a steering angle set point.

In most Pure Pursuit control schemes, the look-ahead distance determines the severity of the turn performed to reach the desired destination point. This parameter is analogous to the distance to a spot in front of a car that a human driver might look toward to track the roadway. In this design setup, the look-ahead distance parameter will have both static and dynamic properties. The parameter can be described as static in the sense that the parameter's value will simply correspond to the distance between the two GPS coordinates in the active waypoint set, a value that will remain unaltered as the vehicle approaches the second waypoint. However, the parameter is also dynamic in the respect that the look-ahead distance will change each time the vehicle reaches the target waypoint, where the waypoint set will then get updated.

An important note to reiterate here is that the computed steering angle message is not transmitted as a steering command (or control signal) but rather, as a steering set point. However, the linear actuator that controls the steering angle is driven via PWM, thereby implying that we can only directly control the steering rate and direction of the front wheels and not the actual steering angle itself. Thankfully, as was noted in chapter 2 when the UAGV was first introduced, the linear actuator comes with a potentiometer in which its angular position can be abstracted via the deployment of an ADC. All of the detailed information regarding the ADC and PWM configuration and calibration for steering control, will not be addressed here as that goes beyond the scope of this thesis, but it can be found in pages 99-102 in [5] if the reader is interested. This steering angle measurement is performed at the steering node and is periodically transmitted as a CAN message (an ID of 206 as was stated in section 2.4) onto the bus during its allocated time slot within the time triggered schedule. Thus, this positional feedback allows for the steering angle of the wheels to be continuously monitored and compared with the computed set point angle. The comparison of that result then determines the PWM commands to generate to either turn the wheels left or right accordingly to drive the steering angle towards the calculated set point. This thus allowed for steering angle set point control, which formed the foundation for the development of the implemented curvature navigation scheme.

3.2.3 A Pure Pursuit Path Planner Integrated with a P Controller

Due to external disturbances (terrain variations, slippage, wind, lateral slopes, etc.) and imperfections in the initially computed steering angle, the UAGV may begin to deviate from the assigned curvature route at any point in time. Thus, to compensate for this and ensure the vehicle stays on course, a control algorithm must be implemented to access how accurately the vehicle is following the assigned curvature path and increase or decrease the steering angle accordingly if significant deviations are being observed.

3.2.3.1 *Controller Design Setup*

To realize a curvature controller, a new radius center point coordinate can be obtained for every time the UAGV reaches its assigned waypoint and moves on to the next subsequent coordinate, which thereby

yields a new waypoint set. The radius center point corresponds to the center of the newly calculated circular arc (via the Pure Pursuit algorithm) for the new active waypoint set. This center coordinate serves as a reference point allowing for the actual radius (the distance between UAGV's position and the circle center point) to be calculated and compared with the target set point radius (actual radius of the circular arc). The mathematical calculation for obtaining the center coordinates from the derived circle is dependent upon the computed difference between the x and y components of the waypoint pair. Figure 3.7 depicts this idea below where the computations for acquiring the radius center point for all four of the possible quadrants are shown.

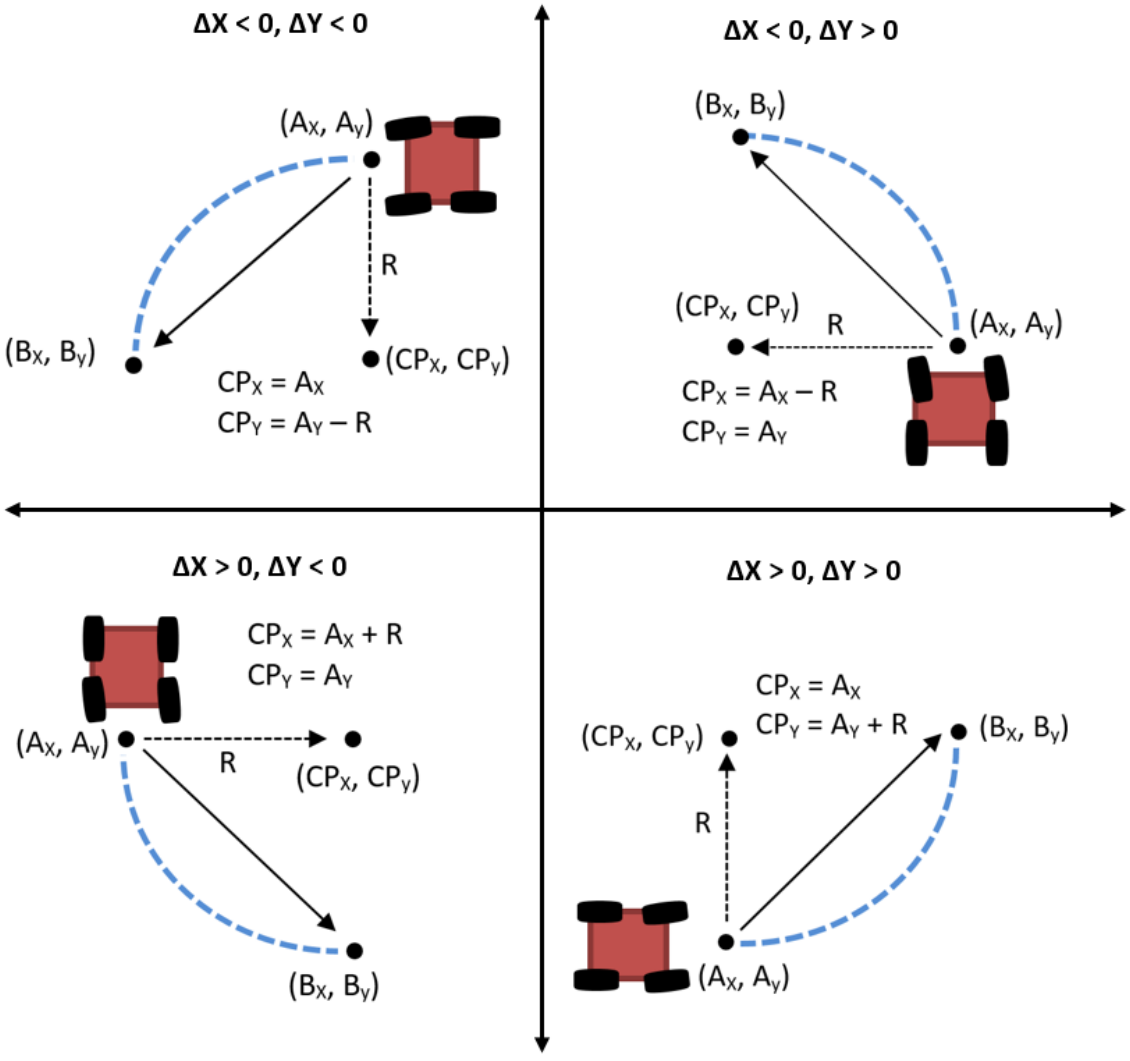


Figure 3.7 - Radius center point calculation of waypoint set for all four of the possible quadrants.

The figure above illustrates the mathematical equations for abstracting the radius center point from the calculated curvature arc for a counter-clockwise turn. Taking the difference between the x and y components of the two waypoints to obtain the delta values then allows for the waypoint pair to be placed in one of four quadrants, thereby revealing the particular formula to be leveraged for acquiring the circle center point coordinate of the calculated curvature arc. Though not explicitly shown here, the reader can infer from the same figure to visualize how the mathematical equations can be altered accordingly to attain the circle center point coordinate for a clockwise turn. Obtaining the radius center point of the circle then allows for precise control over the steering angle where it can then be adjusted accordingly to help keep the UAGV on course and counteract and compensate for when the vehicle begins to diverge from the assigned route. Figure 3.8 illustrates this idea below.

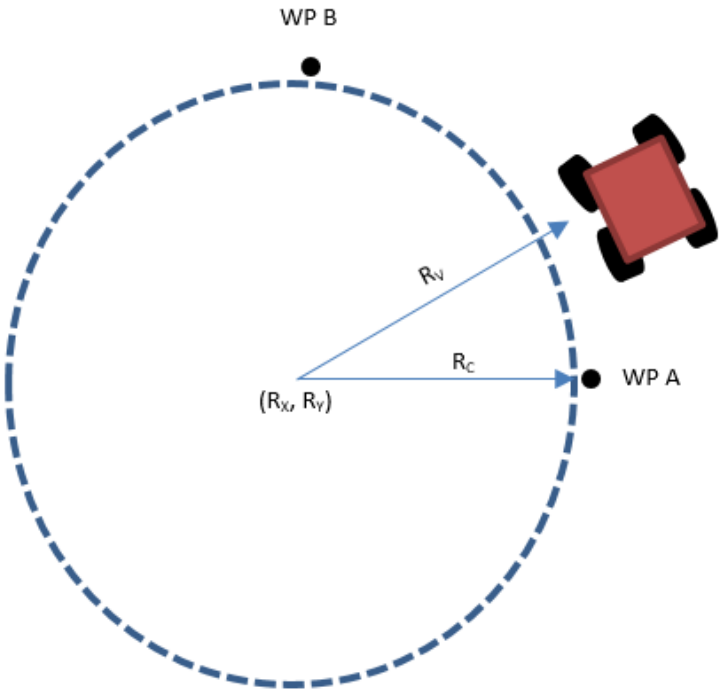


Figure 3.8 – Tracking route accuracy via comparing the measurements of the radius of the calculated arc of the circle (R_C) corresponding to the desired path of travel with the actual radial distance between the circle center point and the robot's current position (R_V)

Thus in essence, the Pure Pursuit algorithm handles the path planning for the UAGV as it maps out the curvature route, and the P controller aids in keeping the UAGV on the assigned course. Consequently, the final implemented curvature navigation algorithm design is a hybrid controller that blends the Pure Pursuit scheme with a P controller and is tailored to accommodate for the speed limitations of the controller node in order to obtain optimal path following performance results. The Pure Pursuit algorithm utilizes trigonometry to determine the steering angle set point for the vehicle to track the calculated the curvature route which establishes the reference arc that ties the two active waypoints together. A P controller is then employed to monitor the accuracy of the UAGV's traveled course via comparing the robot's current radius with the desired radius as was depicted above. The controller will bias the steering angle set point accordingly if the UAGV begins to deviate from the calculated curvature route. For the P controller design, a set of boundary conditions were defined which dictated the controller's response when path deviations occurred. Figure 3.9 illustrates this concept in depth, visually detailing how the boundary conditions determined the controller response.

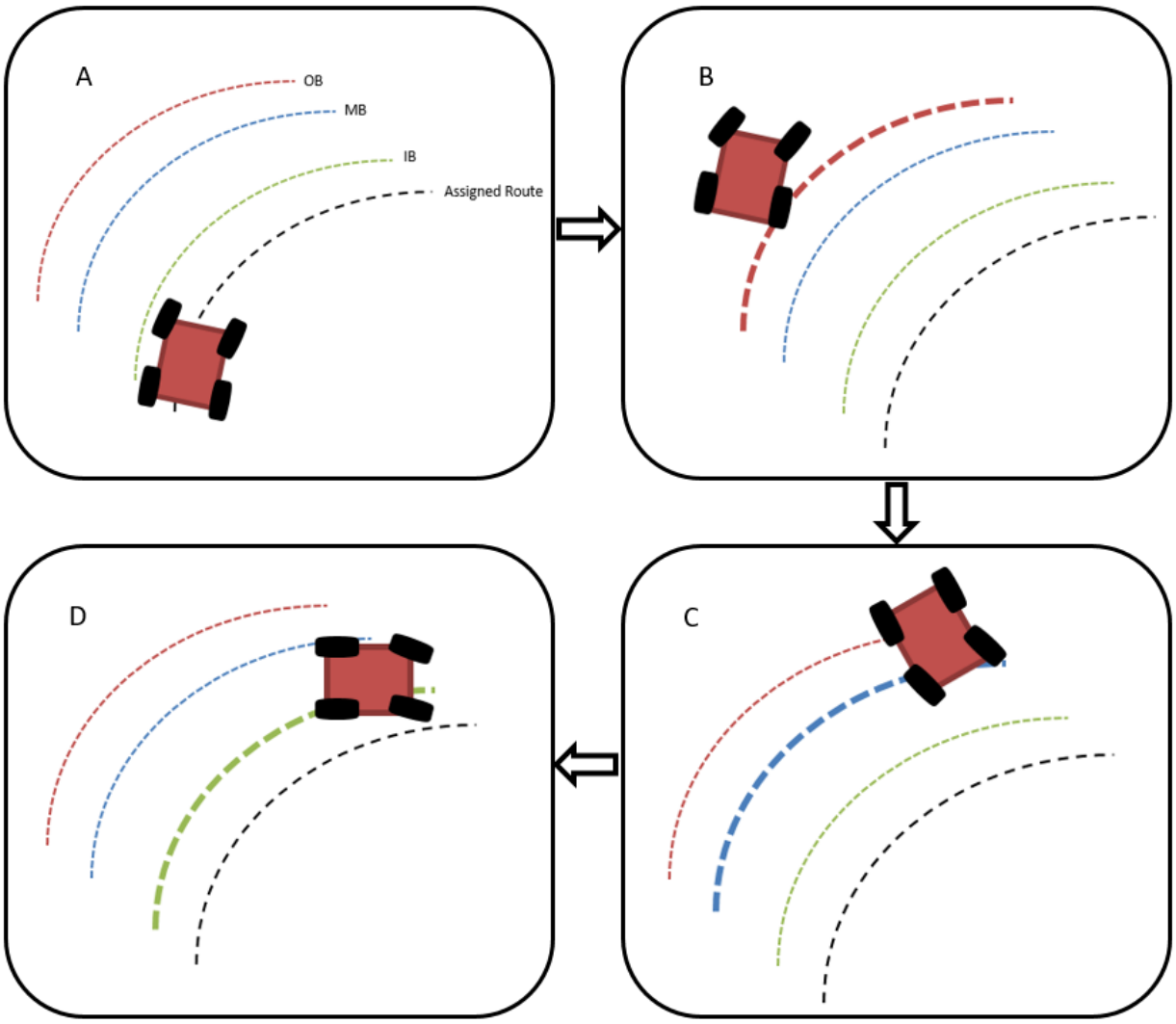


Figure 3.9 – Designed P controller for accessing and correcting for measured path deviations.

As portrayed in the figure above, the designed controller can be broken down into four stages. In stage A, the UAGV is on the assigned route. In stage B, the UAGV is outside of the outer boundary. The UAGV will adjust its steering set point to a magnitude proportional to the vehicle's distance from the assigned route. In stage C, the UAGV is back within the middle boundary. The steering direction is then reversed in effort to have the UAGV slowly approach desired route and avoid from over compensating. Finally, in stage D, the UAGV has arrived back within the inner boundary and is thus, close enough to the desired arc where the steering angle can then be set back to the original set point.

The implication drawn from Figure 3.9 is that the distance between the outer boundary and the target path can be interpreted as the dead band zone since the controller won't be activated unless the UAGV meanders outside of this allotted region. Additionally, the boundary conditions illustrated above, which determine the controller's behavior, only apply when the UAGV is outside of the assigned route. When the UAGV is inside the arc, the UAGV will reverse its steering angle direction, where the magnitude of the new steering angle will be proportional to the UAGV's distance from the desired curvature. The steering angle will then be set back to the original set point when the UAGV is no longer inside the curvature route

3.2.3.2 ***Mathematical Description of Controller***

The derived mathematical equations for the controller that determines the exact value of the steering angle set point can be observed below in Equation 3.5. Table 3.1 offers a description of each of the parameters with their associated hard-coded values as applicable.

$$f(r) = \begin{cases} \delta_{SP} + 3, & iff\ r > (R_C + OB)\ and\ \delta_{SP} > 0 \\ -5, & iff\ r < (R_C + MB)\ and\ \delta_{SP} > 0\ and\ r_{past} > (R_C + OB) \\ \delta_{SP}, & iff\ r < (R_C + IB)\ and\ \delta_{SP} > 0\ and\ r_{past} > (R_C + OB) \\ \delta_{SP} - \dfrac{R_C - r}{30}, & iff\ r < R_C \\ \delta_{SP}, & otherwise \end{cases} \quad (3.5)$$

Variable	Description	Value (initialized in the firmware)
$f(r)$	*Steering angle output as a function of r*	--
$\boldsymbol{\delta_{SP}}$	*Steering angle set point*	--
$\boldsymbol{r}$	*Measured radius*	--
$\boldsymbol{R_C}$	*Actual radius of curvature arc*	--
$\boldsymbol{r_{past}}$	*Previously measured radius*	--
$\boldsymbol{OB}$	*Outer boundary*	*1000 (mm)*
$\boldsymbol{MB}$	*Middle boundary*	*750 (mm)*
$\boldsymbol{IB}$	*Inner boundary*	*250 (mm)*

Table 3.1 - Variables from piecewise function with their associated description and hard coded values (as applicable)

The piecewise equation shown above strictly applies for a clockwise turn (hence, the $\delta > 0$ comparison). The piecewise equation for a counter clockwise turn is mathematically equivalent where the steering angle output is simply symmetrical to the outputs observed for the clockwise turn case. Thus, for the sake of avoiding redundancy, the piecewise function for the counterclockwise turn will not be illustrated here. The values for the parameters observed in the equation and table provided above were obtained pragmatically, where an extensive amount of field testing was performed. It was determined that the values displayed in Table 3.1 realized a reasonable controller response, where controller overcompensation (stemming from an overly aggressive controller that is restricted or limited by a slow response time) was mostly eliminated or at least minimized.

As mentioned earlier, the controller scheme is unique in the regard that it is actually designed to generate a new steering angle set point to compensate for observed path divergences. Controllers are customarily designed to produce a control signal which appropriately administers the *speed* of an actuator or a motor to drive the system towards the desired set point, where the proposed control strategy just presented, actually continuously generates and adjusts the set point control parameter to keep the vehicle on the mapped out circular route. Since the linear actuator that controls the steering angle of the front two wheels

is driven via PWM, the PWM values were saturated in both steering directions when adjusting the steering angle to quickly drive the wheels towards the newly calculated set point. This was done to make the UAGV as responsive to the controller output as possible.

3.2.3.3 *Floating Point Math Inefficiency*

It is also worth mentioning that in most Pure Pursuit derivations, the control scheme is typically contained within the algorithm itself. However, these control schemes usually require the use of trigonometric computations which are inefficient, computationally exhaustive operations to implement on a low-level microcontroller. Thus, the Pure Pursuit algorithm was integrated with a P controller to avoid the excessive use of these trigonometric calculations which will considerably slow down the runtime speed of the control algorithm. Thus, the most prominent challenge with implementing curvature navigation algorithms onto embedded system platforms is the ubiquitous use of computationally expensive calculations involving trigonometric and/or square root computations. The controller was designed in a way to accommodate for this concern and therefore reduce the heavy use of these cumbersome floating point math operations.

3.2.3.4 *Controller Node Computational Load*

To further complicate matters, the controller node already has an exhaustive number of additional tasks that it is responsible for executing. The controller node is accountable for receiving and processing all of the GPS CAN messages and periodically updating all the GPS position information on the LCD screen. Additionally, it governs the operation of the entire state machine via monitoring and receiving the key fob button commands and processing the joystick commands when operating in manual mode. Furthermore, it transmits the CAN steering and motor commands over to the steering and motor driver nodes in addition to transmitting the supplementary CAN messages (IDs of 450-453) over to the data logger for system data collection and analysis and controller response evaluation. Hence, the control algorithm was just one small piece of what all was being processed and executed by the controller node. Thus, for this reason, the controller node was somewhat bottlenecked by the limited processing speed at which the control

algorithm could be executed, as the calculated outputs from traditional curvature navigation schemes that would keep the UAGV on course, would inevitably lag severely in real-time performance testing. Thus, devising a controller to minimize the use of floating point math as well generating a steering angle set point output instead of a control signal, helped remedy this dilemma to an extent and allowed for the conceived and developed Pure Pursuit / P controller model to feasibly be implemented onto the embedded system development platform.

3.2.3.5 *Benefits of Curvature Navigation for Agricultural Machinery*

An important distinction to make here between the Pure Pursuit method that was integrated with a P controller and the unique P controller scheme that was addressed in the first section, is that the Pure Pursuit approach allows the robot to take a curvature-like route when travelling to and from GPS waypoints, while the PID control scheme simply forces the robot to attempt to drive in a straight line between each of the two successive coordinates. Thus, as was mentioned earlier, vehicles that utilize a PID control scheme will often have to make sharp turns when travelling from waypoint to waypoint. These sharp turns will generally be avoided in the Pure Pursuit method thanks to the smooth circular route the algorithm will have the vehicle take. For this reason, the Pure Pursuit method is generally the more preferred approach for most agricultural machinery in use today. Moreover, the curvature turning algorithm is better suited for the Ackerman steering configuration, which is ubiquitous among the machinery seen today within the agriculture industry. Thus, while the described P controller is a suitable navigational solution for waypoint navigation, it is expected that the proposed hybrid Pure Pursuit / P controller model just described, will yield more desirable results when turning is required.

3.3 Implemented Embedded System Design

In this section, the implemented embedded system design for the multiple GPS waypoint navigation algorithm will be discussed in detail. Additionally, a feature was added to the UAGV to allow it to heuristically determine the appropriate control algorithm to commission for each GPS waypoint pair. This added attribute will also be addressed in this section.

3.3.1 Multiple GPS Waypoint Navigation Implementation

The overall end goal for this portion of the project was to obtain a fully controllable GPS driven UAGV, where the user could define an assigned route for the UAGV to travel via feeding the UAGV an arrangement of GPS waypoints. In order to achieve this objective, the UAGV must have some way of knowing when it is time to update and move on to the next waypoint. The implication here is that the UAGV must be able to intermittently check its current position relative to the waypoint it is heading towards. Since the X and Y coordinates of the current target GPS waypoint are obviously already known, Pythagorean's theorem can then be applied for the two coordinates to compute the hypotenuse of the formed triangle once the updated X and Y coordinates corresponding to the UAGV's current position are received over the CAN bus from the GPS node (which occurs at a base period frequency of 10 Hz). The hypotenuse of the triangle corresponds to the measured distance between the UAGV's current position and the waypoint that the UAGV is currently approaching. This distance value will be referred to as the look-ahead distance (LD). Additionally, a radius defining a circle around the GPS waypoints can then be assigned to serve as the threshold for determining whether or not the UAGV is close enough to the waypoint that it can update and move on to the next GPS coordinate. This is the basic operating principle governing how the embedded system for the GPS navigation scheme was designed. This concept is illustrated below in Figure 3.10.

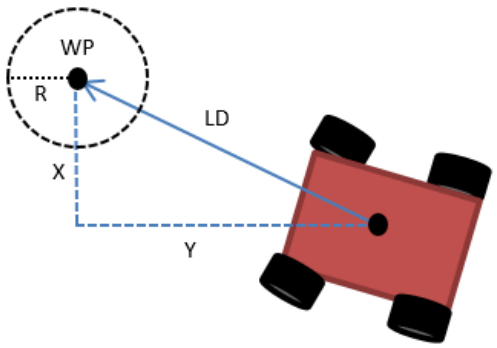


Figure 3.10 - Look-ahead Distance computation

A simple timer peripheral was selected to be used in this design. All the timers on STM32 boards have the same general architecture as they are all designed to accommodate for both hardware and software related tasks. Software tasks mainly involve giving time bases, timeout event generation, and time triggers while hardware tasks primarily consist of generating waveforms on output pins (for PWM applications). For this particular application, a timer needs to be configured for the sole purpose of simply controlling the frequency at which an interrupt handler would execute. Naturally, the interrupt handler would contain the code for checking the robot's position and then computing the corresponding distance between UAGV's position and the current target waypoint. Thus, the UAGV's position check will be governed by a timer driven interrupt. Figure 3.11 illustrates the logical functionality of the interrupt handler in block diagram form.

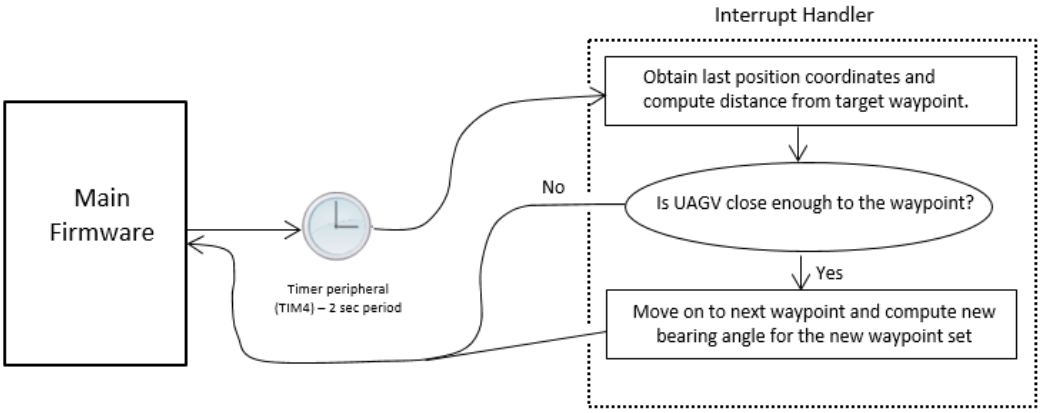


Figure 3.11 - Block diagram of the implemented interrupt handler used for the GPS navigation scheme

The timer peripheral needed to be configured such that it would count at an appropriate clock frequency and also have a suitable period. For this design, as noted in the figure above, it was decided that the UAGV should check its current position once every two seconds. Additionally, the radius defining the circle associated with each GPS waypoint was selected to be two meters, thus setting the error margin to be within plus or minus two meters of the goal GPS waypoint. Since the UAGV's default speed was configured to be at around 1 meter per second, this setup should provide the UAGV with an ample

number of checks for the computed distance of the UAGV's position to be within the range of the error tolerance radius.

Setting the interrupt frequency to 2 seconds required proper configuration of the associated timer peripheral. Thankfully, this was a relatively straight forward procedure as there were just a couple of design parameters that needed to be set accordingly. First, the timer4 peripheral was selected to be used which has a clock rate of 90 MHz. Selecting a pre-scaler of 45,000 divides tick frequency down to 2 kHz. Additionally, the top count value of the timer was selected to be 4,000 thereby yielding the desired period of 2 seconds. The top value determines the highest tick value the timer will count to before triggering the interrupt handler event and resetting the counter. A visualization of this setup can be observed below in Figure 3.12.

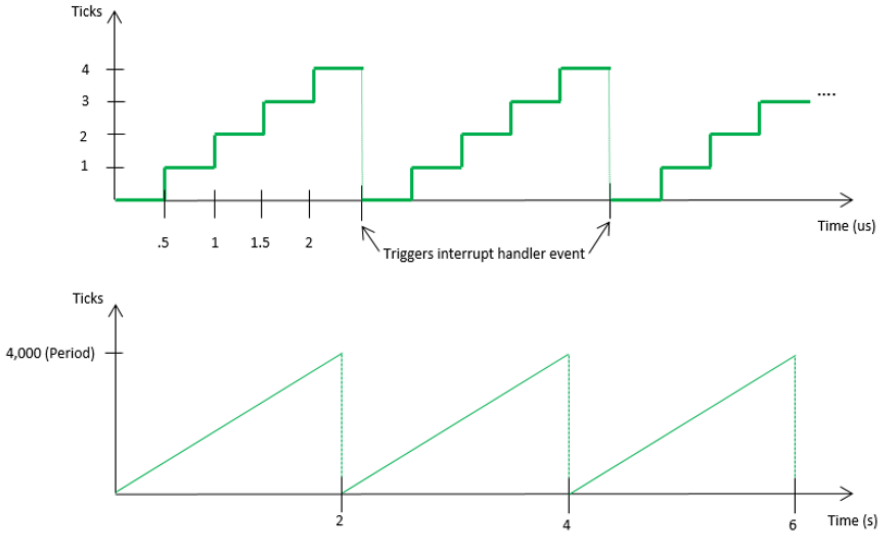


Figure 3.12 - Timing diagram for interrupt handler execution in microscale (top) and macroscale (bottom) form

As already implicated from Figure 3.11, the interrupt handler contains the code to compute the UAGV's current distance relative to the waypoint it is currently approaching, using the latest position coordinates received from the GPS node. Once the distance is calculated, the system would then check to see if that distance was less than the defined radius being used for all of the GPS waypoints. If it is, the UAGV will

know that it is close enough to the current target waypoint and can therefore update and move on to the next waypoint in the assigned route. In the firmware, there was a statically declared one dimensional array which contained all of the GPS waypoints for the UAGV. Hence, the waypoint set gets updated via incrementing the index to that array. Additionally, whenever the UAGV moves on to the next waypoint, the new bearing angle is computed for the new waypoint set. Figure 3.13 gives an illustration that visually summarizes this entire procedure.

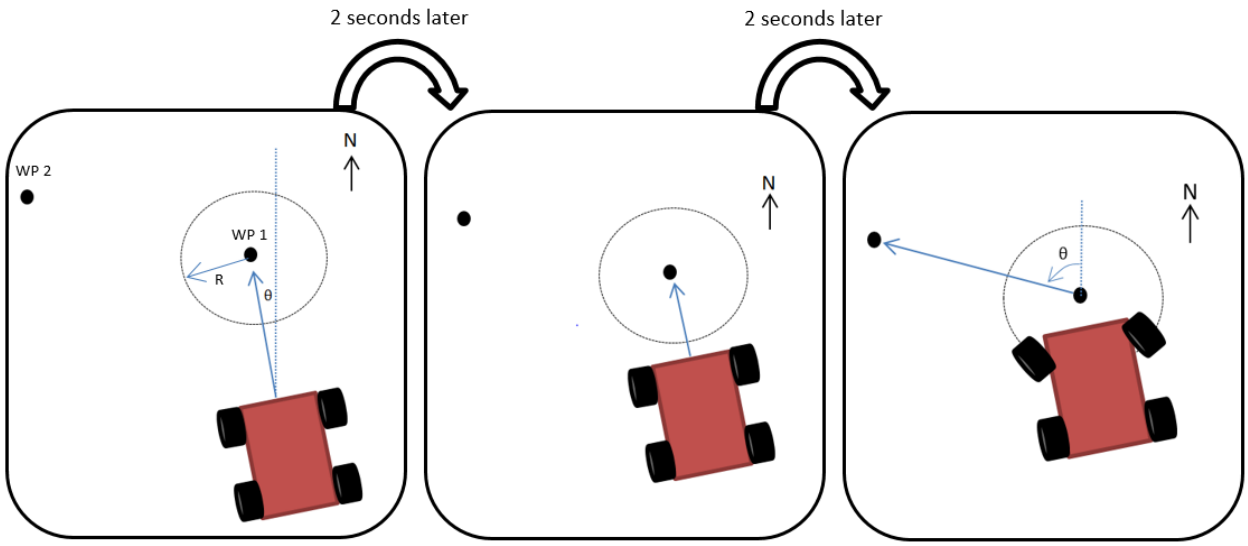


Figure 3.13 - GPS waypoint navigation process

Finally, when the last waypoint is reached (i.e., the index comes to the last coordinate in the waypoint array), all commands being routed from the controller node to the steering and motor driver nodes will cease as the robot has completed its assigned route and will therefore stop and exit out of GPS navigation mode.

3.3.2 Dynamic Selection of Navigation Scheme

Another feature that was added and implemented onto the UAGV was the ability to dynamically select the appropriate navigation algorithm to execute each time the waypoint set was updated. In essence, the UAGV would be able to decide if the P controller or Pure Pursuit algorithm should be utilized for the new

waypoint set. The decision making process depends on two variables: the current heading angle of the robot and the bearing angle of the new waypoint set. Based on the measured discrepancy of these two angles, the UAGV will then be able to determine if a turn needs to be performed (Pure Pursuit) or if the line navigation algorithm would be best suited for the vehicle. Figure 3.14 illustrates this idea below.

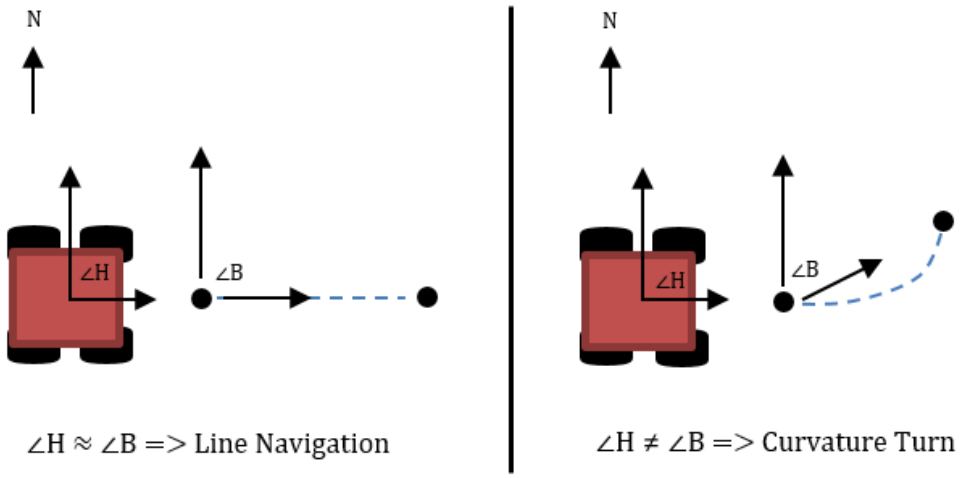


Figure 3.14 - Dynamic selection of path following algorithm based on the comparison of the heading and bearing angles

In the firmware, North is designated as the 0° degree mark when performing the heading and bearing angle computations. The angle value then increases from 0° to 360° in the clockwise direction such that the East direction would be at 90°. For determining whether to perform a left or right turn, the heading angle and bearing angle are compared. A left turn is executed when the bearing angle is less than the heading angle and a right turn is performed for the reverse case. Figure 3.15 below illustrates the exact allotted regions for determining which navigation algorithm to utilize each time the waypoint set gets updated.

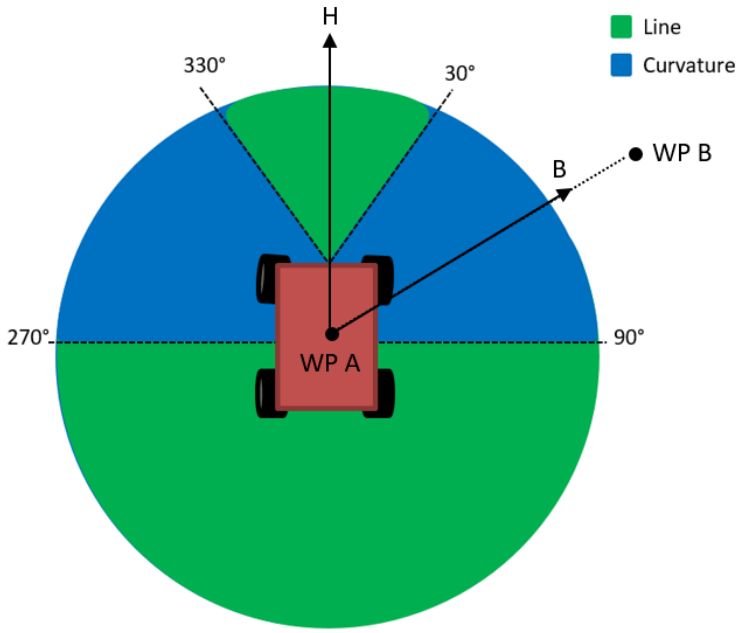


Figure 3.15 - Allotted regions demonstrating the selection of the navigation algorithm to commission for each individual waypoint set

As depicted in the figure above, the curvature algorithm will only get called upon when the heading angle and bearing angle disparity are within either the 30° to 90° or 270° to 330° range. Otherwise, the line navigation algorithm will be employed. The line navigation algorithm is robust in the sense that it can handle all the possible transitions to any four of the existing quadrants which is why the line navigation algorithm was configured to be executed for the 90° to 270° range. If the user desires the UAGV to utilize the curvature navigation algorithm to complete a 180° turn, a total of three waypoints (two waypoint pairs) would need to be used. This requirement stems from the nature of how the curvature algorithm was derived. The mathematical computations for calculating the circle center point will only work correctly if the waypoint pair falls into one of the four available quadrants based on the X and Y delta values (as was illustrated in Figure 3.7) where the heading and bearing angle delta is less than 90°. In general, the line navigation should be used to handle tight turns while the curvature navigation performance is optimal for wider turns. The user should be mindful of this when assigning a GPS waypoint path for the vehicle to take.

Since the end goal is to obtain a fully controllable GPS driven UAGV, this feature enables the UAGV to be fed an array of waypoints where the UAGV can dynamically determine if line navigation (P controller) or curvature turning (hybrid Pure Pursuit / P controller) should be performed for each waypoint set. This decision is made each time the UAGV reaches its current target waypoint and moves on to the next subsequent coordinate. This sense of intelligence allows the UAGV to optimize its travelled path for each individual waypoint set by determining the most fitting navigation algorithm to commission. Figure 3.16 offers a depiction of this concept where the UAGV is fed a batch of waypoints and must determine the optimal control algorithm to execute for each waypoint pair.

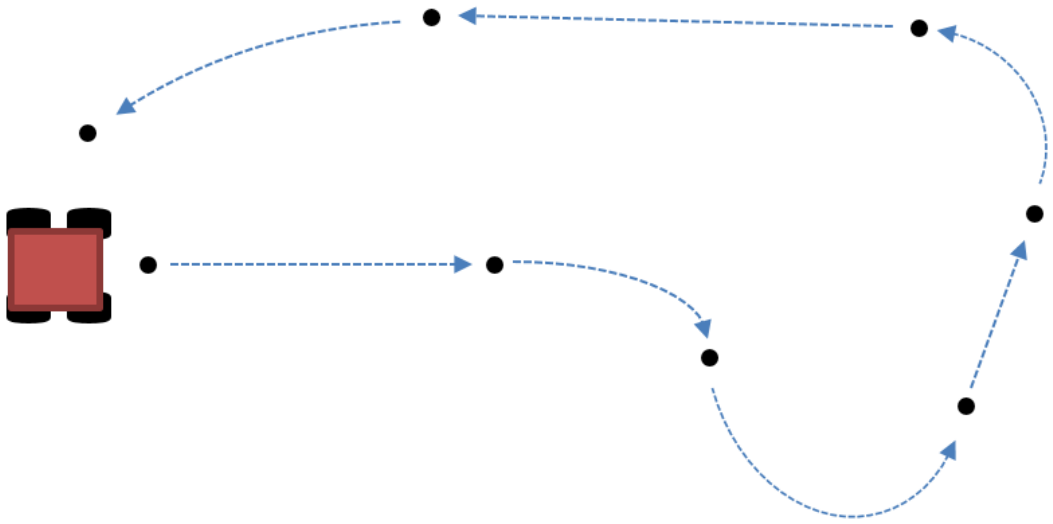


Figure 3.16 - Dynamic selection of the path following algorithm to execute for each individual waypoint pair from a multi-coordinate GPS assigned route

3.4 Results & Discussion

The developed multiple GPS waypoint navigation scheme implementation on the UAGV was thoroughly tested to access its overall performance. The data acquired from these experiments will now be presented and discussed.

A series of tests were performed on the UAGV to verify the developed GPS navigation algorithm was operating as intended. Figure 3.17 below depicts the tractor test track which is the location of where all of the GPS navigation testing was performed for the refinement of the developed algorithms. Also indicated in the figure is the location of the base station for all of the navigation tests which served as the reference point for all the obtained GPS position readings.

Figure 3.17 - Nebraska Tractor Test Track for autonomous navigation tests

For the first verification test, the UAGV was fed an assortment of waypoints that would require the vehicle to update its waypoint set each time the current target waypoint was reached. Additionally, line navigation was the only control algorithm that was employed for this test. This allowed for the conceived and developed multiple GPS waypoint navigation scheme outlined in section 3.3.1 to be tested and evaluated to ensure that the UAGV would properly update its waypoint set at the appropriate time and would opt out of autonomous navigation mode when the last assigned waypoint has been reached. Figure 3.18 shows the position map for the UAGV for an assortment of waypoints demonstrating the working multiple GPS waypoint navigation scheme.

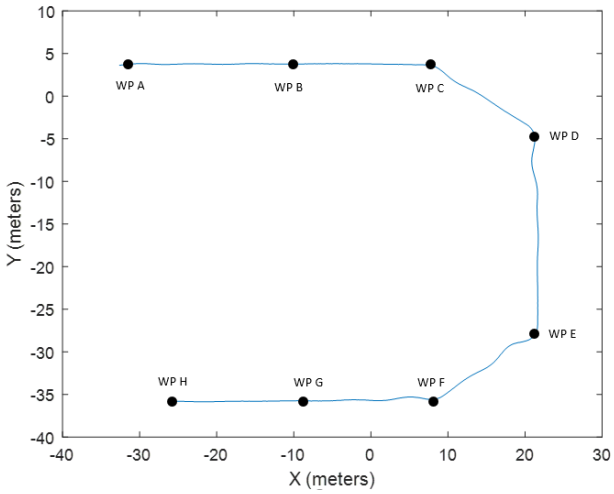


Figure 3.18 - Position map for the UAGV for an assortment of waypoints where just the line navigation algorithm is utilized

The second verification test conducted involved attempting to have the vehicle perform a smooth, curvature turn via the utilization of just the line navigation algorithm. In this approach, the semi-circular route was partitioned into a set of equally spaced waypoints to approximate the curvature path. The position map depicting the performance results for this test is shown below in Figure 3.19.

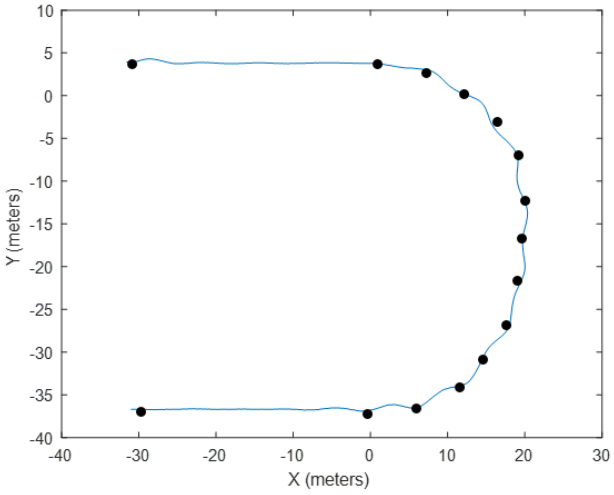


Figure 3.19 - Position map where an approximate clockwise curvature turn was performed utilizing just the line navigation algorithm and a set of equally spaced waypoints

For the above test, the frequency at which the periodic position check would occur needed to be increased to half a second to account for the finer waypoint resolution that the assigned route required. Fortunately, increasing the frequency of the position check by this amount did not noticeably slow down the execution of the firmware or cause the control algorithm to lag.

For the third test iteration, the UAGV was fed a set of waypoints where the same clockwise turn would be performed. However, in this experiment, the curvature navigation algorithm was enabled. Additionally, the waypoints were strategically assigned such that both navigation algorithms would need to be utilized. This allowed for the accuracy and reliability of the UAGV's ability to dynamically determine the most appropriate navigation algorithm to execute for each waypoint pair to be accessed. Additionally, this allowed for the accuracy and stability of the curvature algorithm to be inspected and evaluated. Figure 3.20 depicts the third verification test where a position map is shown, illustrating the GPS waypoint assigned route and the vehicle's traveled path.

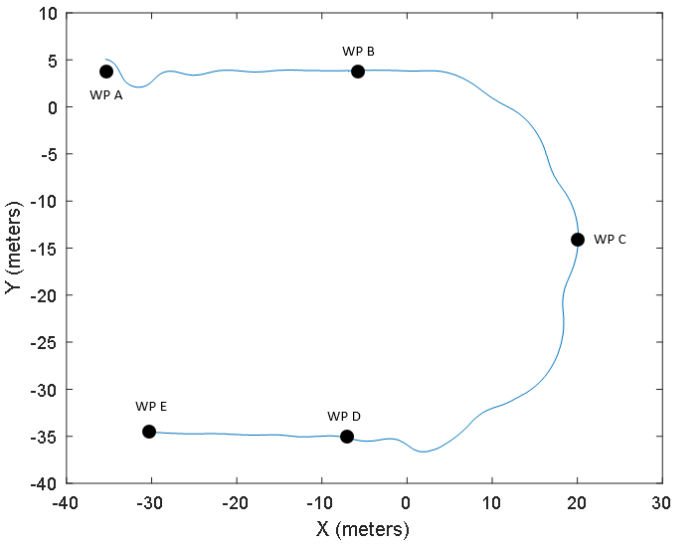


Figure 3.20 - Position map for an assortment of waypoints illustrating the integration of both the line and curvature navigation algorithms in which a clockwise turn is performed

As can be observed in the figure above, the UAGV performed a clockwise turn and was successfully able to navigate to each waypoint and stopped when the last waypoint was reached. Moreover, the UAGV correctly selected the appropriate navigational algorithm to execute for each waypoint pair. This strategy yielded similar performance results to the second verification test (Figure 3.19) where the same curvature turn was performed, but this configuration required far fewer waypoints. The figures that follow offer a visually more detailed description of the path tracking accuracy results (from Figure 3.20) of the executed algorithms for each individual waypoint set.

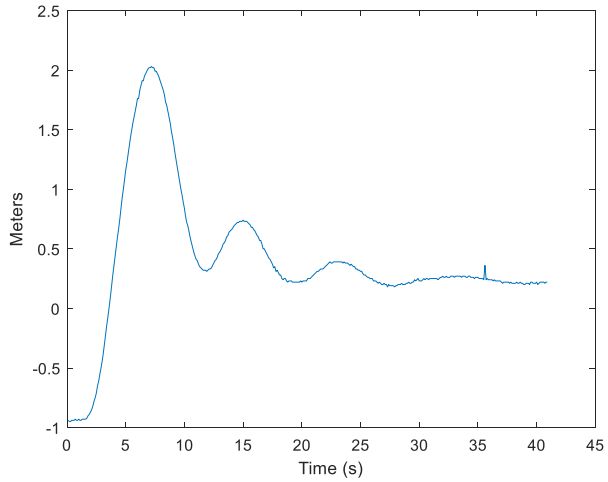


Figure 3.21 - Cross track error for A-B waypoint set shown as a function of time

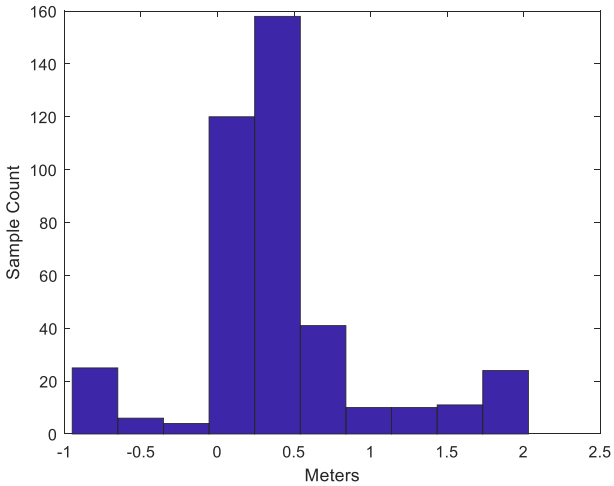


Figure 3.22 - Cross track error distribution for A-B waypoint set

Figure 3.21 displays the cross track error as a function of time while Figure 3.22 depicts the cross track error distribution of the A-B waypoint set via a histogram. The UAGV had a settling time of around 25 seconds with a mean steady state error of 2.32 cm.

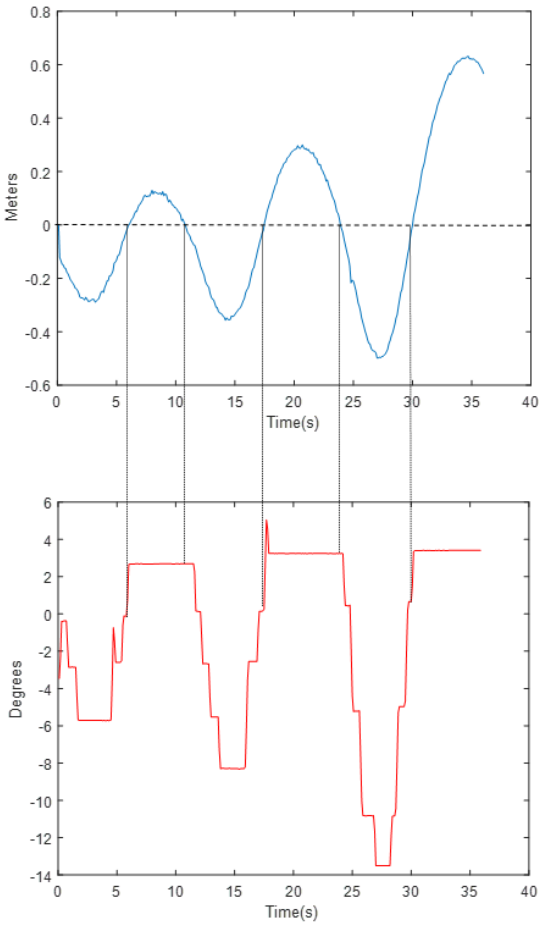


Figure 3.23 - Radius error (top) and steering angle control set point (bottom) for B-C waypoint set shown together as a function of time

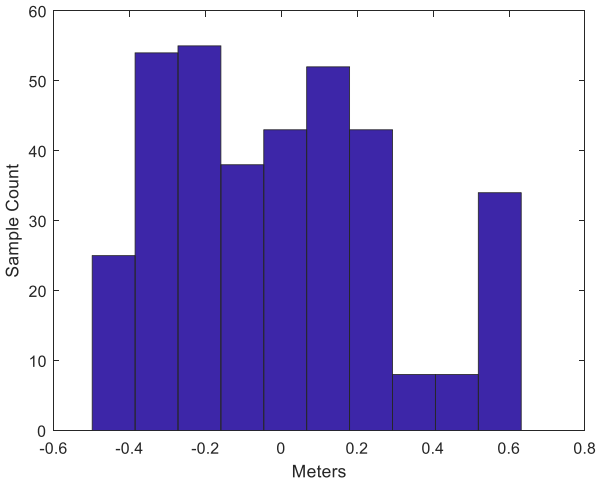


Figure 3.24 - Radius error distribution of B-C waypoint set

Figure 3.23 displays both the radius error and the resulting steering angle set point (adjusted by the controller) together as a function of time while Figure 3.24 depicts the radius error distribution of the B-C waypoint set via a histogram. The histogram indicates a relatively even distribution from about -.5 m to .6 m. Figure 3.23 visually illustrates and reveals the controller latency where the parameter could thus be graphically obtained. The plot displayed the controller's response to the vehicle's measured position (relative to the curvature arc) where the steering angle set point is adjusted accordingly when the vehicle begins to deviate too far from the calculated curvature route. Additionally, a closer examination of the curvature navigation algorithm provided in Figure 3.23 reveals a noticeable amount of undesirable oscillation about the target curvature arc (i.e., the X axis corresponding to the black dashed line). Since the steering angle will only be adjusted in the event that the UAGV has moved outside of the pre-determined boundary thresholds, oscillating, to some extent, is unavoidable, since vehicle will inevitably begin to drift from the curvature route as time progresses. The controller was designed with the primary goal of keeping the UAGV within a reasonably close vicinity to the curvature arc. Eliminating oscillations, while desirable, was a secondary objective.

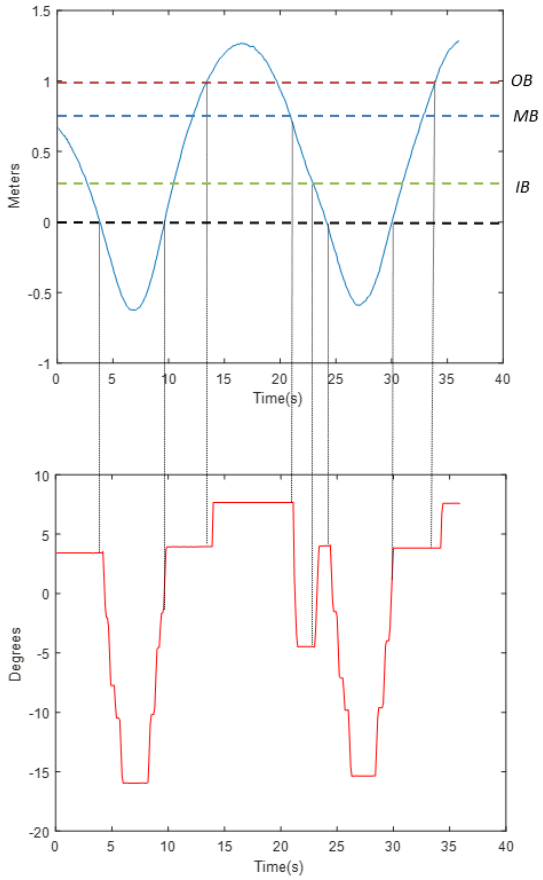


Figure 3.25 - Radius error (top) and steering angle control set point (bottom) for C-D waypoint set shown together as a function of time

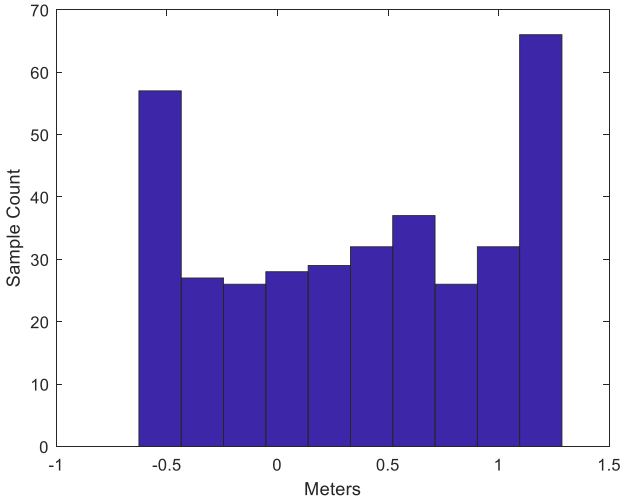


Figure 3.26 - Radius error distribution of C-D waypoint set

Figure 3.25 displays the both radius error and the resulting steering angle set point (adjusted by the controller) together as a function of time, while Figure 3.26 depicts the radius error distribution of the C-D waypoint set via a histogram. Figure 3.25 indicates a noticeable increase in the magnitude of the oscillations around the desired curvature when compared with the prior B-C arc. This therefore prompts the activation of the boundary conditions for the P controller to keep the UAGV on the curvature path (as was explained in section 3.2.3). When the error radius increases beyond the extent of the outer boundary (1 meter), it can be observed that the controller incrementally increases the steering angle to a value proportional to the vehicle's distance from the desired arc. Additionally, when the vehicle re-enters the middle boundary, the steering angle was reversed in attempt to have the vehicle slowly approach the desired curvature. Finally, when the vehicle's position was back within the inner boundary, the steering angle was set back to the original steering angle set point. The controller lag or latency depicted in Figure 3.23 and Figure 3.25 was observed to be negligible since the implemented control technique minimized the use of computationally expensive floating point calculations. Again, the observed oscillations stem from the nature of the controller's design as the proposed and developed scheme was primarily implemented as a means to prevent the vehicle from venturing too far off of the calculated course.

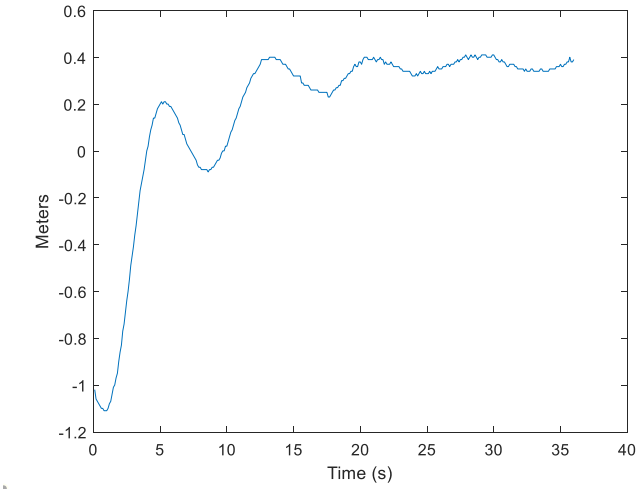


Figure 3.27 - Cross track error for D-E waypoint set shown as a function of time

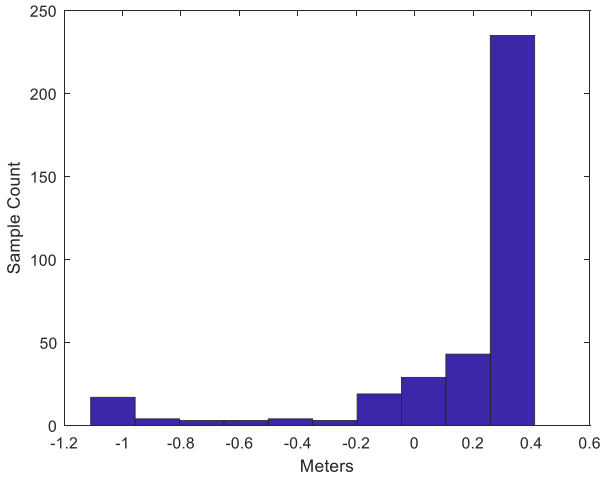


Figure 3.28 - Cross track error distribution for D-E waypoint set

Figure 3.27 displays the cross track error in the time domain while Figure 3.28 depicts the cross track error distribution of the D-E waypoint set via a histogram. The UAGV had a settling time of around 20 seconds with a mean steady state error of 3.52 cm. This was the last waypoint set of the GPS assigned route. The vehicle, hence, stopped when it reached waypoint E.

The autonomous multiple GPS waypoint algorithm was demonstrated to be fully operational where the UAGV could be completely controlled via the GPS module. A curvature navigation scheme was proposed in effort to further optimize the vehicle's traveled path. The UAGV could intelligently select the most fitting control algorithm to execute for each waypoint pair based on the comparison of the bearing angle of the active waypoint set and the current heading angle of the vehicle itself. The most noticeable room for improvement would be further refinement of the curvature navigation algorithm to further extenuate the observed oscillations about the calculated target arc.

4 Obstacle Avoidance

The topic of obstacle detection and avoidance will be discussed in this chapter. With the trends in agricultural machinery over the recent years and the motivation and rationale for autonomous navigation implementation, the area of obstacle detection and avoidance is a pressing area of research in the agriculture sector. In order to accomplish the goal of complete autonomy, the designed robot must be developed to closely and effectively mimic and perform all the functions of that of a human. In a typical robot control architecture configuration, a microprocessor serves as the robot's brain and the decision making is determined by the developed algorithms and programs. Actuators and motors serve as the robot's "feet", allowing the robot to move, break, and steer accordingly. However, the most difficult function to emulate is that of the human eye. The act of being able to take in all the information regarding one's surroundings and process the information quickly to respond appropriately is a feat effortlessly performed by a human. However, implementing this onto a robot can be quite the arduous quandary for an engineer to tackle. This is the problem that will be addressed in this chapter. Various vision technologies used for obstacle detection will first be discussed. Several obstacle avoidance algorithms will then be introduced and explained. Subsequently, the implemented sensor node with the selected vision sensor and obstacle avoidance algorithm will be addressed in which the hardware architecture, embedded firmware design and overall implementation will be explained in detail. Finally, the chapter will conclude with a discussion and analysis of the obtained data from the conducted test runs.

4.1 Vision Sensor Technology

An analysis of the vision sensor technology realm for autonomous robots reveals a handful of obstacle detection sensor possibilities. Each has their own unique set of advantages and drawbacks that makes each sensor technology suitable for different applications. In this particular project, the main concern is developing a reliable object *detection* scheme for obstacle avoidance in agriculture applications where the location (relative to the robot) and size of the obstacles can be extracted. Following will be a discussion of

two different vision sensor technologies used today that are especially applicable to this project. Their advantages and drawbacks will be examined and their use in unmanned agriculture ground vehicles (UAGVs) will be considered.

4.1.1 Ultrasonic

One common technology used for obstacle detection and avoidance is the ultrasonic sensor, also referred to as sonar. Ultrasonic sensors operate on the basis of sound waves. Bats and whales use this principle for navigation and obstacle avoidance. Ultrasonic sensors can generally be placed into two categories: active or passive. Active ultrasonic sensors generate sound waves that propagate through the environment where their echo returns can be measured to detect the presence and location of obstacles. Passive ultrasonic sensors merely listen for sound echoes. Thus, an active ultrasonic sensor has both a transmitter and receiver while a passive sensor has only a receiver.

One application of sonar technology is that it can be leveraged to aid in navigation and obstacle avoidance. This was first realized in the 80s when a sonar ring was developed onto a robot for the purpose of obstacle detection [20]. Additionally, in the same year, a sonar based mapping and navigation system was developed onto an autonomous mobile robot as a means for obtaining a multilevel two dimensional description of the robot's environment to assist with path planning and navigation [21].

Due to the sound wave's ability to penetrate water, ultrasonic technology has found many applications in underwater or marine navigation as well. In [22], chirp sonar was utilized to analyze and classify marine sediments. In [23], multi-beam forward looking sonar was employed for underwater obstacle detection, avoidance, and tracking. Sonar can also be applied to autonomous surface vehicles (ASVs). In [24], a scanning strategy for sonar was presented in which ASVs would be able to navigate and avoid obstacles in lake and harbor environments, relying entirely on the received and processed sonar data.

Though sonar's main area of application has remained in the marine domain, sonar technology, on a smaller scale, has been explored and harnessed in different areas of the agriculture sector as well for the

purpose of aiding mobile robots in navigation and automation. In the early 90s, an ultrasonic based sensing scheme was developed onto a robot to allow for furrow guidance [25]. In [26], an ultrasonic based weed detection system was contrived to be used on cereal crops. In [27], a comprehensive study was conducted to evaluate the accuracy and effectiveness of the use of ultrasonic sensors for the application of object detection in agriculture environments where several objects commonly encountered in agricultural settings were used in the experiment to assess the ultrasonic sensor's ability to reliably identify them. In another study, ultrasonic sensing technology was investigated as an approach for corn plant canopy characterization [28].

One of the defining characteristics and unfortunate drawback of the ultrasonic sensor is the necessity to use a myriad of sensors together if a detailed picture of the environment is desired [29]. An additional requirement for proper object detection is that the object must be roughly perpendicular to the sensor for successful detection since the sensor operates on the basis of measuring echo returns [29]. Moreover, sound waves travel at a relatively slow rate when compared with the many other vision sensors on the market today that utilize waves in the electromagnetic spectrum. Hence, this makes their reaction time a concern. Furthermore, this makes sonar less capable of tracking moving objects due to the sensors' slower signal speeds and sampling rates. Many of these mentioned limitations among other constraints have prevented sonar from being widely used as the primary sensor for obstacle detection and avoidance on an autonomous agriculture machine. Still, sonar provides a low-cost obstacle detection scheme that serves well for applications where simple object detection is required and could potentially have applications in serving as a supplementary sensor to aid in autonomous navigation.

4.1.2 LIDAR

Another relevant vision detection technology scheme is light detection and ranging, or more commonly referred to as LIDAR. The operating principle of LIDAR is actually quite simple: illuminate a designated area (which consequently emits pulse of electromagnet light) and measure the time it takes for the light waves to come back to the source [30]. Most commonly this is accomplished through high frequency

lasers where the LIDAR instrument will fire rapid pulses of laser light in a designated area, and the scattered photons will come back and get collected at the receiver [30]. The photons are then measured as a function of a time, thus allowing for obstacles and their placement relative to the sensor to be extracted [30].

LIDAR possesses several distinct advantages that make it a technology of particular interest, specifically in the area of autonomous vehicle research. Most notably, since LIDAR technology employs laser scanners for data collection, it thus has high resolution capabilities, which consequently allows for very accurate and precise measurements of depth, spanning a large field of view, to be performed [30]. Hence, LIDAR technology enables vision instruments to procure descriptive pictures of the environment, with high granularity. Additionally, the amount of light present in the environment has little to no effect on the performance of these sensors [30]. Moreover, since these sensors emit laser pulses of light, their response time is exceptionally fast [30]. Furthermore, lasers can travel at far distances thus giving these sensors a broad detection range [30]. LIDAR is a newer form of obstacle detecting technology when compared to sonar so it is still in the early stages of development, but it has been starting to make a name for itself over the recent years with all of these compelling advantages being brought to light.

Consequently, this technology has quickly started to gain traction in the research industry of autonomous navigation and obstacle avoidance. Most notably, researchers are looking to incorporate this technology into future autonomous, self-driving automobiles to aid in obstacle detection and avoidance. There is an extensive amount of research being done today in the sector of self-driving automobiles, and one of the key instrumental sensors at the forefront of the entire operation is the utilization of a LIDAR sensor. It is quickly being realized in the industry as the one indispensable ingredient to one day possibly attaining the end goal of having a fully self-drivable car. Research has been done to utilize this technology to aid GPSs, implemented onto commercial vehicles, to achieve autonomous navigation in urban environments [31]. Additionally, the detection of traversable road terrains for positive and negative obstacle detection was accomplished via the implementation of a LIDAR histogram [32]. In [33], 2D-LIDAR was leveraged to

run in parallel with a precise localization algorithm for vehicles in 3D urban environments. 2D-LIDAR was also utilized for obstacle detection and avoidance for ground mobile robots in [34]-[35]. In [36], LIDAR was instrumented onto a GPS guided mini-bus to achieve obstacle detection and avoidance.

Though LIDAR sensing technology has shown an immense amount of promise in the area of self-driving automobiles, over just the past decade, it has quickly started to gain momentum in the agriculture industry as well. In [37], a 3D LIDAR sensor was utilized on an agricultural robot for plant detection and feature extraction. As [38] notes, LIDAR is becoming widely used to measure the structural and geometrical characteristics of trees and crops which reveals important scientific, biological information such as photosynthesis, growth, CO2 sequestration, and evapotranspiration. Though the geometrical abstraction and biological analyzation of plants is one of the most eminent applications of LIDAR technology in the agriculture sector, LIDAR can also be used for simple object detection to aid in the navigation of mobile agricultural ground robots. As the technology is being researched, further developed, and better understood, LIDAR is becoming the most popular utilized sensor for obstacle detection and avoidance in autonomous field applications. In numerous studies, it was used as the primary obstacle detecting device for the testing and development of UAGVs [39]-[41].

LIDAR is the undeniable future when it comes to obstacle detection and identification technologies. Its many attractive advantages have allowed it to find a wide variety of possible applications which has consequently, motivated engineers and scientists to research and investigate the technology further. The implications the technology has, specifically on the agriculture industry is apparent. However, the most substantial roadblock for LIDAR remains that it is still a relatively young, rudimentary technology and there is much yet still to be developed and refined.

4.2 Obstacle Avoidance Algorithms

An investigation into obstacle avoidance algorithms discloses an innumerable assortment of options, due to the knowledge accumulated from the extensive amount of research that has been performed in this area

over the past half century. Obstacle avoidance has been a pressing area of research for the past several decades due to possessing a very diversified scope of applications that span a wide variety of industries. This would include (but is not limited to) construction, manufacturing, waste management, space exploration, military transportation, and obviously agriculture. This consequently has opened the gateway to the development and commercialization of a countless number of obstacle avoidance algorithms that are being utilized on robots today for a widespread range of applications. In this section, some of the more established, well-known reactive obstacle avoidance techniques will be examined, culminating with the selection of one of the described algorithms to implement onto the UAGV.

4.2.1 Bug Algorithm

The simplest obstacle avoidance algorithm ever developed is an avoidance scheme called "the bug algorithm". The basic idea of the algorithm is that once an obstacle is encountered, the robot will fully circle the object to find the point with the shortest distance to the goal location [42]. The robot proceeds towards the target location once a complete circle of the encountered object has been made and the closest point to the goal location has been established. An illustration of this principle can be seen below in Figure 4.1.

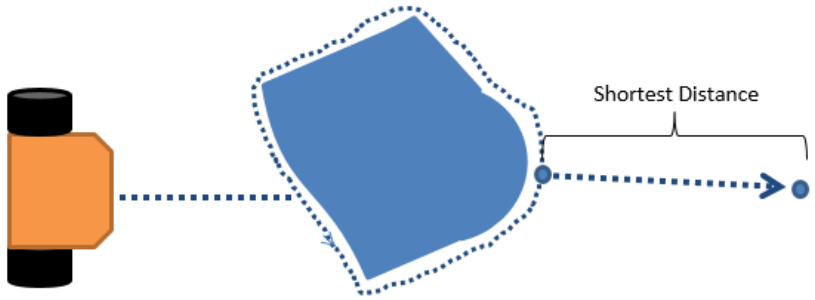


Figure 4.1 - Bug algorithm

As the reader might have conjectured, the algorithm has long been crippled by its obvious inefficiency, which has prevented the algorithm from gaining much traction as a viable, adequate solution for real-world obstacle avoidance problems. Moreover, since the algorithm relies only on recent sensor

measurements, the system is very susceptible to sensor noise [42]. Despite these conspicuous shortcomings, there have been some modifications and improvements made to the algorithm over the years to help decimate the obvious efficiency issue [42]. However, the algorithm still remains as a very inefficient avoidance scheme, especially when compared with all of the other modern algorithms that have been developed and are in use today. The main advantage with the bug algorithm is its simplicity. However in this case, simplicity fails to provide a sound, elegant solution to the problem at hand.

4.2.2 Vector Field Histogram

Another pretty common obstacle avoidance scheme is the vector field histogram (VFH) algorithm. This algorithm maps all of the sensor readings into a histogram where the histogram corresponds to a probability distribution function (PDF) – the probability of obstacles being present at individual position segments (i.e., bins of a histogram) spanning the robot's field of view [42]. Figure 4.2 gives an illustration of this concept.

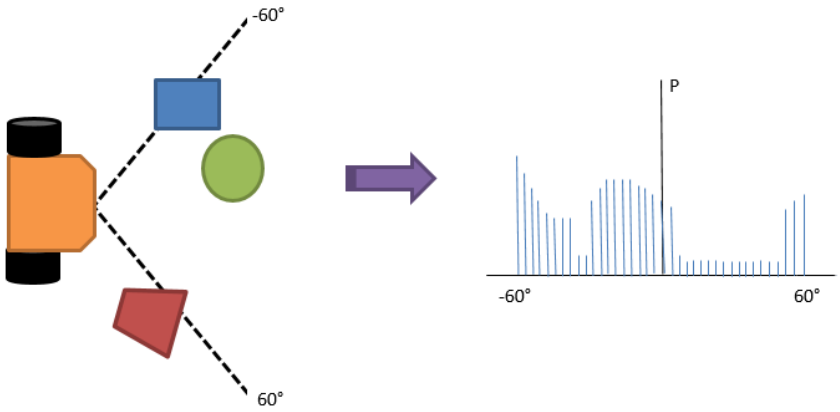


Figure 4.2 - Vector Field Histogram algorithm

Although it is not too difficult to obtain a high level understanding of how this algorithm works, the algorithm itself is actually rather mathematically rigorous. There are a lot of advance and rather complex computations that will be glossed over in this short presentation of the scheme. The algorithm is described in full detail in [43] if the reader is interested in a more detailed explanation of this avoidance approach.

Since this algorithm employs a histogram to obtain a map of its surrounding, this algorithm is much more immune to sensor noise than that of the bug algorithm [42]. Sensor noise will only slightly raise the bins but it will not be by a sufficient amount for it to erroneously register as an object. Thus, sensor noise will not have any influence on the robot's overall peripheral field of vision. The histogram is thus designed to filter out and reject noise fluctuations as outliers. Hence, this scheme offers a more robust, noise resistant solution to the problem compared to the former bug algorithm approach.

The most prominent drawback with the VFH, however, is the considerable computation load that this algorithm requires [42]. This presents a difficult challenge for many real-world problems where one would have to develop a practical implementation of this algorithm onto an embedded system platform. Though still viable as it can still most certainly be accomplished, this disadvantage prevents the VFH algorithm from being a satisfactory solution for this thesis problem from a complexity standpoint, since this particular application requires an embedded system implementation.

4.2.3 Attractive Forces

Another common avoidance algorithm is one that is called the potential fields approach. This algorithm assumes the robot is driven by virtual force where the goal location is an attractive force and obstacles and obstructions serve as repulsive forces [42]. The final path the robot will take to head towards the goal location is the combination of all the contributing forces. A visual illustration of this algorithm can be observed below in Figure 4.3.

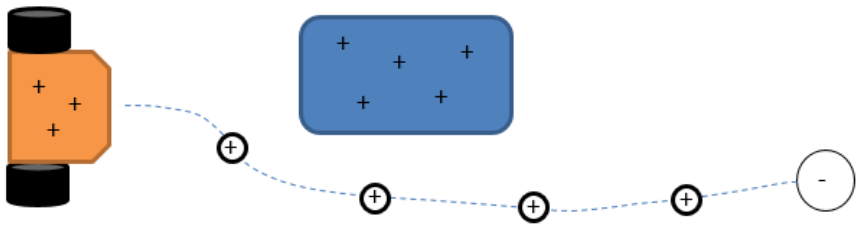


Figure 4.3 - Attractive forces algorithm

Essentially, this algorithms equates the robot's surrounding to a vector field where obstacles are given an opposing electric charge while the destination point is assigned an opposite, attractive charge. The most prominent disadvantage with this algorithm is the immense amount of math involved in computing these vector fields which obviously can present a challenge for implementing such a scheme on an embedded system application [42]. Additional drawbacks include poor performance in narrow passages, trap situations that can easily occur, possible fluctuations in the travelled path when obstacles are present, and sometimes avoiding the shortest path if the passageway is narrow [44]. All of these drawbacks stem from the nature of the algorithm which operates on the principle of vector fields. While the algorithm is definitely suitable for some applications, these drawbacks prevented this algorithm from ultimately being selected as the obstacle avoidance scheme to implement onto the UAGV.

4.2.4 Follow the Gap

The last and most important obstacle avoidance scheme to be addressed in this section is an algorithm known as the "Follow the Gap" method. The algorithm essentially detects all obstacles in front of the vehicle, finds the associated gaps, determines which gap is the largest, and then guides the robot down the center of that gap [45]. An illustration of this idea can be seen below in Figure 4.4.

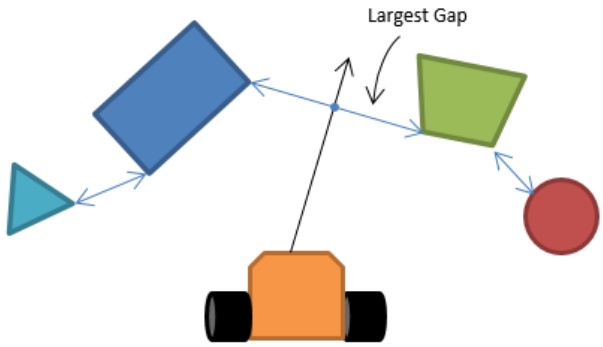


Figure 4.4 – Follow the gap algorithm

This algorithm presented several compelling advantages. First, the algorithm was pretty intuitive as implementing the algorithm onto a microcontroller that controls a robot would be relatively

straightforward. Secondly, the computations required were comparatively simple and thus could be easily handled by a microcontroller on an embedded system platform. Furthermore, the avoidance scheme would serve well as a reactive obstacle avoidance algorithm for a UAGV which, in this case, would cause the vehicle to temporarily override the GPS navigation control scheme until the detected obstacle (s) are no longer in sight and have been safely eluded. For these reasons, it was decided that this obstacle avoidance algorithm would be implemented onto the UAGV. Since this avoidance scheme was selected, the overall workings of the algorithm will be explained in a little more detail than the priors ones were.

The algorithm essentially detects all of the obstacles present and uses the edges of the obstacles to compute the lengths of all the corresponding gaps [45]. In order to understand how the algorithm accomplishes this, the algorithm can be broken down into several steps.

The first step involves using the detected obstacle edges and their measured distances relative to the sensor to obtain the associated X and Y coordinates for each detected edge. These coordinates can be obtained by employing Pythagorean's theorem. An illustration of this concept can be observed below in Figure 4.5. The associated equations for this step are shown in Equations 4.1-4.4.

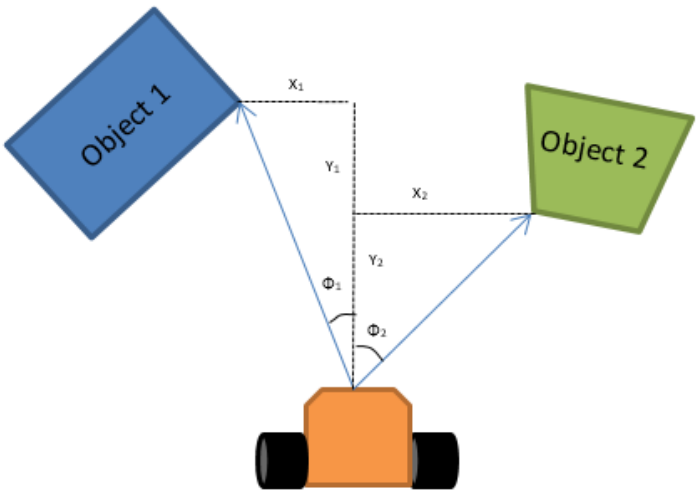


Figure 4.5 - Triangles formed from detected edges

$$Y_1 = d_1 cos\phi_1 \quad 4.1$$

$$X_1 = d_1 sin\phi_1 \quad 4.2$$

$$Y_2 = d_2 cos\phi_1 \quad 4.3$$

$$X_2 = d_2 sin\phi_2 \quad 4.4$$

Now that the coordinates for the detected edges have been obtained, one can then compute the gap distance by again, employing Pythagorean's theorem. This step is illustrated below in Figure 4.6. The corresponding equations for this calculation are Equations 4.5-4.7.

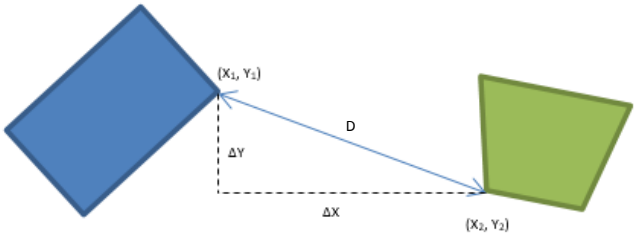


Figure 4.6 - Pythagorean's theorem implementation for gap calculation

$$\Delta X = X_1 - X_2 \quad 4.5$$

$$\Delta Y = Y_1 - Y_2 \quad 4.6$$

$$D = \Delta X^2 + \Delta Y^2 \quad 4.7$$

Finally, once the gap length has been determined, the algorithm then computes the next successive gap via the same process just described (if another gap exists). Once all of the gaps have been computed, all the gaps are compared and the largest gap gets selected. The largest computed gap is then used for calculating the necessary steering angle to drive the vehicle down the center of that gap. This is accomplished by employing the law of cosines, utilizing some additional fundamental trigonometric identities, and applying some basic algebraic manipulation. An illustration of this idea with all the

relevant parameters labeled can be observed below in Figure 4.7. The derived equation for the steering angle computation is shown in Equation 4.8. If the reader interested in a complete explanation of this derivation, the reader can reference [45] where the entire step by step derivation is outlined and discussed in detail.

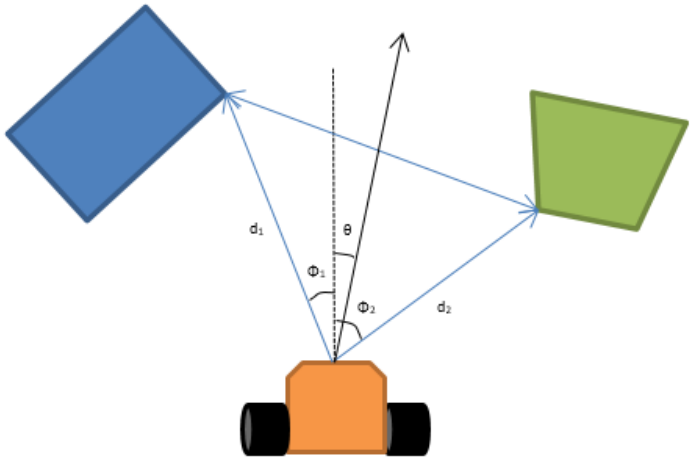


Figure 4.7 - Final stage of the follow the gap algorithm

$$\theta = \arccos\left(\frac{d_1 + d_2\, cos(\phi_1 + \phi_2)}{{d_1}^2 + {d_2}^2 + 2d_1 d_2\, cos(\phi_1 + \phi_2)}\right) - \phi_1 \qquad 4.8$$

4.3 Implemented Embedded System Design

The overarching objective for this portion of the project was to implement a real-time reactive obstacle avoidance algorithm onto the UAGV where the UAGV would be able to detect obstacles and then navigate around those detected obstructions. This obstacle avoidance scheme would operate in line with the GPS waypoint navigation program, thus, enabling the UAGV to safely complete its assigned route that was mapped out by the respective GPS waypoints that it was given via the user. Since the curvature navigation algorithm is not currently designed to be able to recover from extreme path divergences, the UAGV will be configured to only attempt to avoid detected obstacles if the line navigation algorithm is in operation. Otherwise, the UAGV will simply stop when an obstacle is detected. The first piece of material

that needs to be addressed is the hardware components that were chosen to be used on the implemented sensor node of the UAGV.

4.3.1 Hardware Components

The Garmin 1D LIDAR-lite V3HP was selected to be used as the vision sensor for the UAGV. A LIDAR sensor was chosen due to its many intriguing advantages that were noted in the earlier section. Moreover, this particular sensor was compact, light-weight, reasonably priced, and came with pre-developed low-level driver firmware that allowed for the sensor to quickly and easily be interfaced to a microcontroller. The pre-developed firmware was an especially attractive feature since it significantly sped up the development time. This LIDAR sensor will essentially act as the "eyes" for the vehicle. A picture of the sensor is offered below in Figure 4.8.

Figure 4.8 - Garmin LIDAR sensor (Retrieved from https://buy.garmin.com/en-US/US/p/557294)

Additionally, since the sensor is only 1D, the device needs to be rotated in order for the UAGV to obtain a complete field of view of its surroundings. To accomplish this, a stepper motor (model: SY35ST28-0504A) was used for the LIDAR sensor to be mounted on. The selected stepper was a bipolar stepper with a step size of 1.8° (200 steps/revolution) and consisted of two motor coils. Figure 4.9 below shows a picture of the selected stepper.

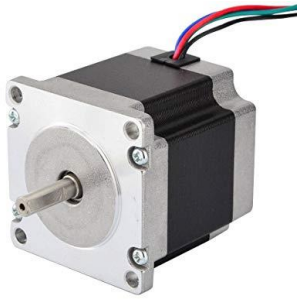


Figure 4.9 – Stepper motor (SY35ST28-0504A) used for rotating LIDAR sensor (Retrieved from https://www.amazon.com/Stepper-Motor-178-5oz-1-26Nm-Stepping/dp/B00PNEPF5I)

While a pair of H-bridges can serve as suitable driver for stepper motors, such a configuration puts a lot of additional requirements on the microcontroller. In such a setup, the microcontroller has to figure out the step sequence signals to send to the H-bridges which will in turn, control the stepper motor response. This method can thus take up a lot of the processing power of a microcontroller and hence, from a processor standpoint, is seen as an inefficient approach to controlling a stepper. Thankfully, there exist stepper driver modules on the market today that are self-contained, where the modules themselves perform all the necessary step sequence computations for controlling the stepper motor based on the digital pules received from the microcontroller. The A4988 module was selected to serve as the stepper motor driver board and can be seen below in Figure 4.10.

Figure 4.10 - Stepper motor driver module (A4988) (Retrieved from https://www.newegg.com/p/298-005A-00010)

Additionally, the stepper motor and LIDAR sensor were directly interfaced to an Arduino microcontroller due to the available pre-written low-level driver libraries that existed for the LIDAR sensor which was exclusively compatible with the Arduino. The Arduino Uno is depicted below in Figure 4.11. Additionally, a picture illustrating how all of the discussed hardware was assembled and mounted onto the front of the UAGV platform is provided below in Figure 4.12.

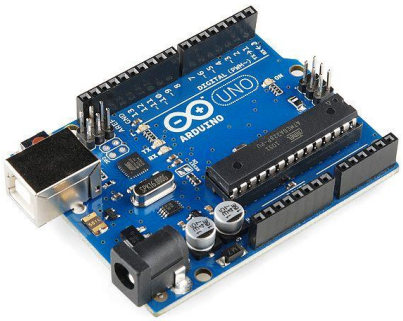


Figure 4.11 - Arduino Uno (ATMEGA328P) microcontroller (Retrieved from https://www.crazypi.com/arduino-uno-r3-microcontroller)

Figure 4.12 - Installation of stepper motor and LIDAR sensor onto the front cross frame of the UAGV

4.3.2 Embedded System Architecture

Now that the selected pieces of hardware have all been discussed, we can now begin discussing the general hardware architecture of the developed sensor node.

As previously noted, an Arduino was used to interface with the LIDAR sensor. It was also used to control the stepper motor that the LIDAR sensor was to be mounted on. Since the available low-level module driver libraries were only compatible with the Arduino platform, it was decided that the Arduino would be integrated into the sensor node for purpose of avoiding the need of having to write our own custom register-level driver libraries from scratch. The Arduino was configured to communicate with the LIDAR sensor over the I2C communication protocol. Additionally, the Arduino was linked to a STM32f446RE, which served as the central microcontroller for the LIDAR sensor node. The Arduino and STM32 communicated with each other serially over the USART peripheral. This step was essential since the STM32 has the discussed CAN peripheral necessary for communicating with all of the other nodes in the DRTS. Thus, the Arduino acts as the "middle man" between the components and the STM32. A visual illustration of the setup just described can be observed below in Figure 4.13.

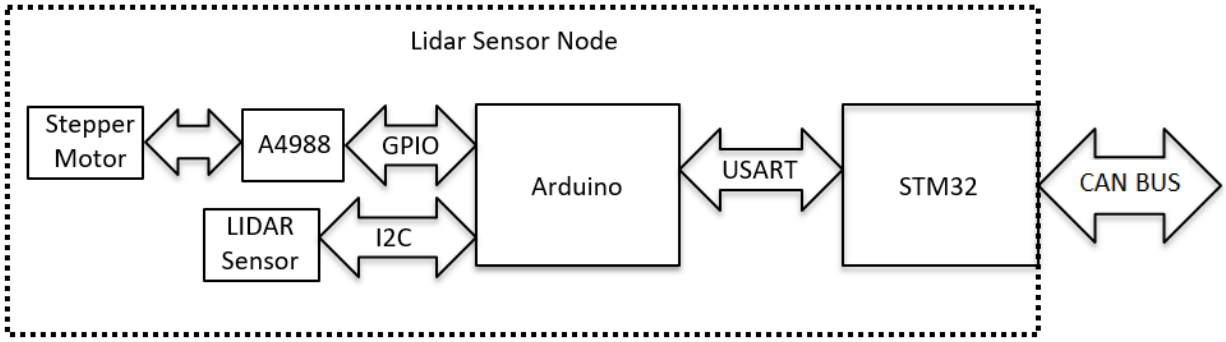


Figure 4.13 - Embedded hardware architecture of the sensor node

4.3.3 Firmware Design

With the embedded system design architecture for the sensor node now fully explained, the framework for the implemented firmware on the sensor node and the controller node can now be discussed. For the

sensor node, the discussion will be broken down according to the two microcontrollers and their assigned tasks allocated in the firmware.

4.3.3.1 *LIDAR Sensor Node - Microcontroller Task Allocation*

The Arduino has three primary tasks: 1) Control the position of the stepper motor which rotates the sensor giving the UAGV a full range field of view. 2) Process the LIDAR data to scan for possible obstructions. 3) Run the obstacle avoidance algorithm in the event an obstacle is detected to compute the necessary steering angle to avoid the detected obstacle(s).

In the firmware, the Arduino was configured to rotate the sensor from 30° to 150° in increments of .9° (the micro step pins on the stepper driver chip can be utilized to further increase the resolution of the stepper position). At each positional increment, the Arduino would process the LIDAR reading and store the result in a "LIDAR array". The LIDAR array contains all the LIDAR readings at each degree point spanning the 30°-150° range. For this setup, a defined radius for the LIDAR readings was decided to be 4.5 meters. This threshold radius determined the detection range for the UAGV's line of sight. Any reading that fell below this radius threshold would register as an obstacle while any reading that was greater than this threshold would be interpreted as a gap. A visual illustration of what has just been described is offered below in Figure 4.14.

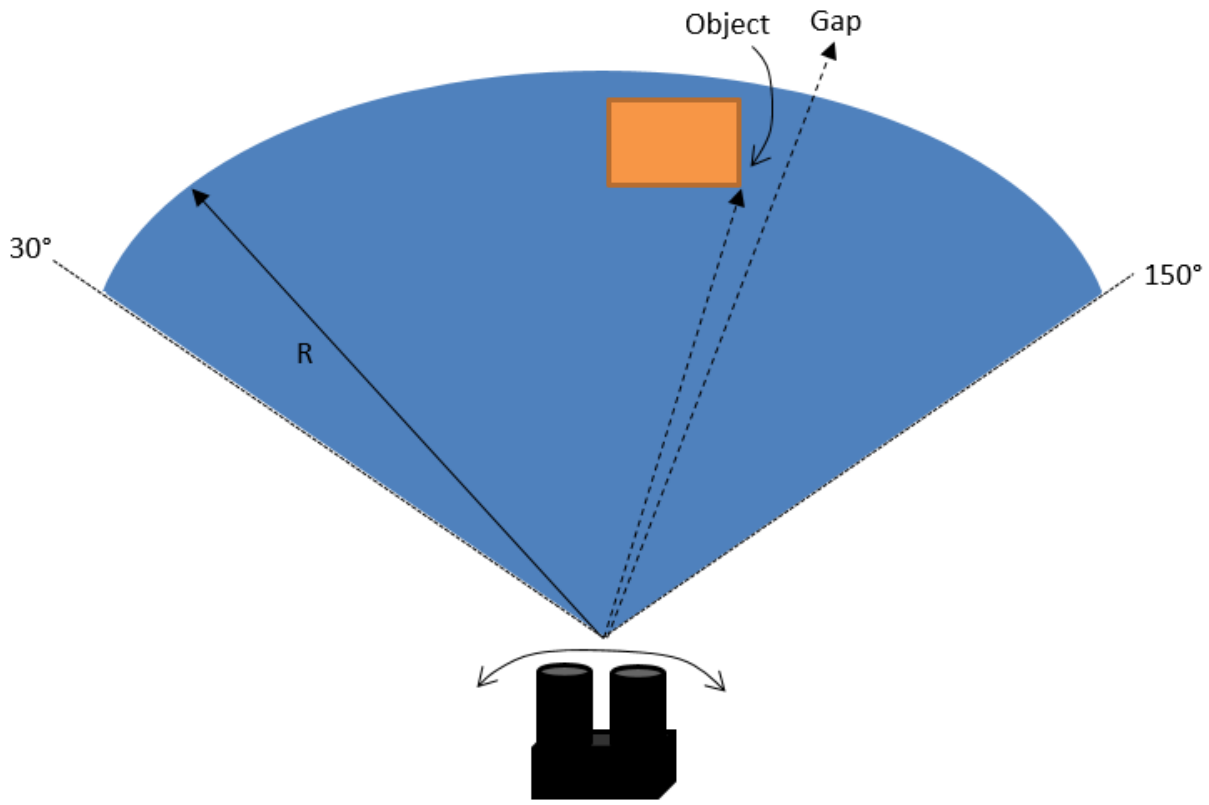


Figure 4.14 - UAGV's field of vision and object/gap identification

Performing a complete sweep spanning 30°-150° would take approximately two seconds for the stepper to complete. Thus, the UAGV would receive an updated vision map of its surroundings approximately every two seconds. Since this design scheme is limited by the speed at which the LIDAR sensor can be rotated, the UAGV has a somewhat slow reaction time to a detected obstacle. To compensate for this, the threshold radius was defined to be sufficiently large to provide the UAGV with an ample amount of time to detect and navigate around obstacles. Once a complete sweep has been performed (i.e. the LIDAR array obtains all readings spanning the 30° to 150° range), the Arduino will then check to see if there was a LIDAR reading that fell below the radius threshold. If there was, the Arduino will then run the "follow the gap" obstacle avoidance algorithm using the data obtained in the LIDAR array to compute what the steering angle needs to be in order to safely maneuver the vehicle around the detected obstruction(s). Otherwise, no steering angle will be calculated.

The STM32's central responsibility is to receive and transmit data across the CAN bus. The controller node will repeatedly send out a CAN message informing the sensor node of the UAGV's current operating mode in the FSM (ID of 501 as noted in section 2.4). Thus, this CAN message gets processed by the STM32 at the sensor node. The STM32 will then convey that information to the Arduino over the USART. Thus, if the UAGV is in autonomous run mode, the STM32 will serially transmit a "b" character over the USART to the Arduino which will notify it to begin rotating the stepper and utilizing the sensor to start scanning for obstacles. Otherwise, the Arduino will be idle, waiting to receive a "b" from the STM32, which will only occur if the UAGV's current operating state is autonomous run mode.

Additionally, the STM32 is also responsible for sending the steering angle computed by the Arduino onto the CAN bus to be received and processed by the controller node. As was discussed in section 2.4, the steering message has an allocated time slot for transmission and reception. Thus, if the STM32 receives a steering angle from the Arduino, the STM32 will convert the serial character data to an unsigned integer, which can then be stored as data in a CAN packet (ID of 500) matching the message packaging scheme that was shown in Figure 1.3. Additionally, the STM32 is configured to wait for the appropriate time slot to transmit the steering message during its assigned time window within the TTCAN bus schedule (Figure 2.12). The sensor node will know the correct point in time to transmit the message by utilizing the time reference message it will periodically receive from the GPS node, as this message serves as the baseline time reference point and helps keep all the nodes in the system synchronized. If no obstacle is detected or the UAGV is not in autonomous run mode, the STM32 on the sensor node will simply just be idle and no message will be transmitted during the allocated time frame for the steering angle message. In this way, an event triggered message was integrated onto a deterministic, time triggered structured system.

4.3.3.2 ***Controller Node - State Machine Revision***

The other piece of firmware that should be addressed briefly is the implemented firmware on the controller node. The firmware design of the controller node had to be modified to resemble that of the updated FSM depicted in Figure 2.17. Essentially, in autonomous navigation mode, the controller node

would process the CAN messages from the GPS node to obtain its current position in order to compute the appropriate steering and speed commands to be sent over to the steering and motor driver nodes to drive the UAGV towards the target GPS waypoint. However, if the controller node received a steering angle message from the LIDAR sensor node, the controller would then enter the Obstacle Avoidance state. In this mode, the controller node would temporarily ignore the GPS data and would instead, process the steering angle message received from the LIDAR sensor node. A steering set point message would then periodically get transmitted by the sensor node over to the controller node each time a complete sweep from the stepper was performed, provided that there is still a detected object within the UAGV's defined range of sight.

There is still an unresolved issue with the current design setup however that needs to be addressed. The UAGV still has no way of transitioning out of the obstacle avoidance state and going back into autonomous navigation mode when an obstacle has successfully been evaded. This was remedied by having the Arduino transmit a unique serial "path is clear" notification to the STM32 if there are no longer any obstacles detected after a complete LIDAR sweep has been performed. The STM32 on the sensor node will then respond by transmitting the "path is clear" CAN message over to the controller node. This message will effectively need to be transmitted twice in order for the UAGV to transition out of the obstacle avoidance state and to go back into GPS navigation mode. The first transmission will cause the controller node to straighten out its wheels. The second transmission will actually trigger the state transition within the FSM. Thus, two consecutive obstacle-free scans need to be performed in order for UAGV to transition back into the GPS navigation state. In this way, the UAGV will appropriately transition out of the obstacle avoidance state in the FSM when all of the obstructions have been successfully evaded. If the detected obstacle is too close to the vehicle to be safely avoided, a "stop" command will be transmitted as the steering message to halt the UAGV. Table 4.1 specifies how the data contained within the steering message is mapped to a unique, respective driving command.

Message Description	Character	Unsigned Integer (CAN data)	Integer	Action
Steering CMD	‘A’ – ‘s’	65 - 115	-25° - 25°	Steering Angle of UAGV to avoid obstacle
“Stop”	‘t’	116	26	Stops UAGV
“Path is Clear”	‘u’	117	27	UAGV transitions back into Auto Run Mode

Table 4.1 - The content of the steering command message transmitted to either avoid obstacles, stop the robot, or transition out of the obstacle avoidance state if the detected objects have been successfully evaded

4.3.3.3 ***Signal Flow of the CAN Transmitted Steering Message***

Figure 4.15 illustrates the signal flow chain and the data type conversion process for the transmission of the LIDAR obstacle avoiding steering message throughout the relevant nodes within the DRTS over the shared CAN bus.

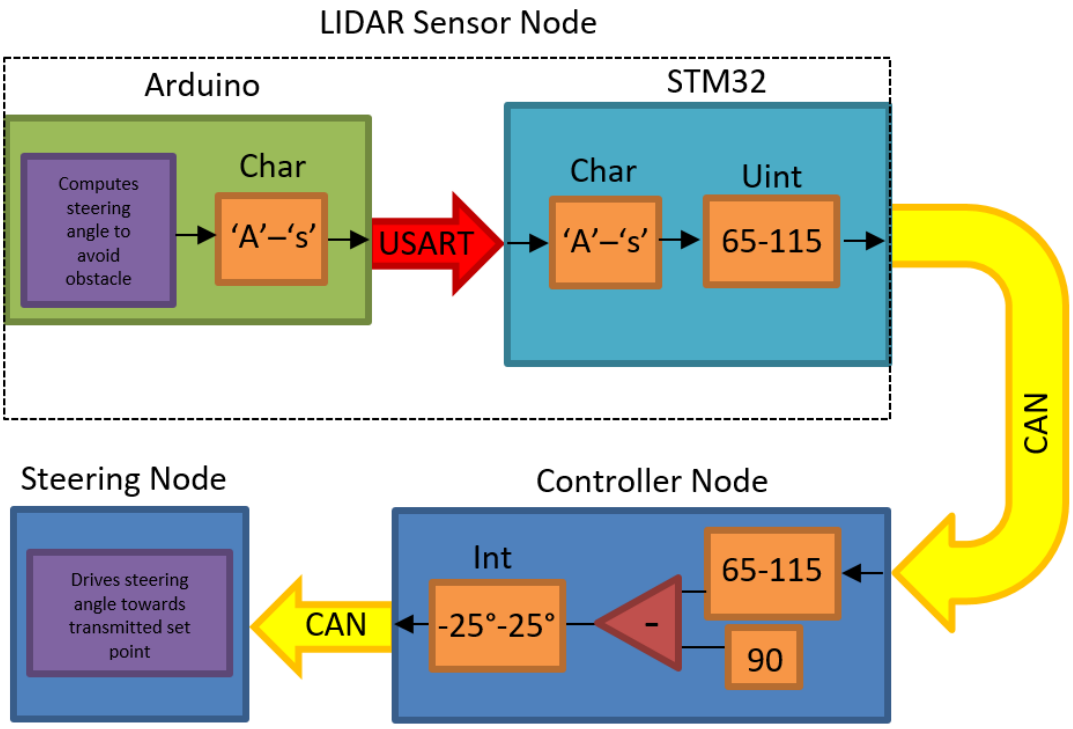
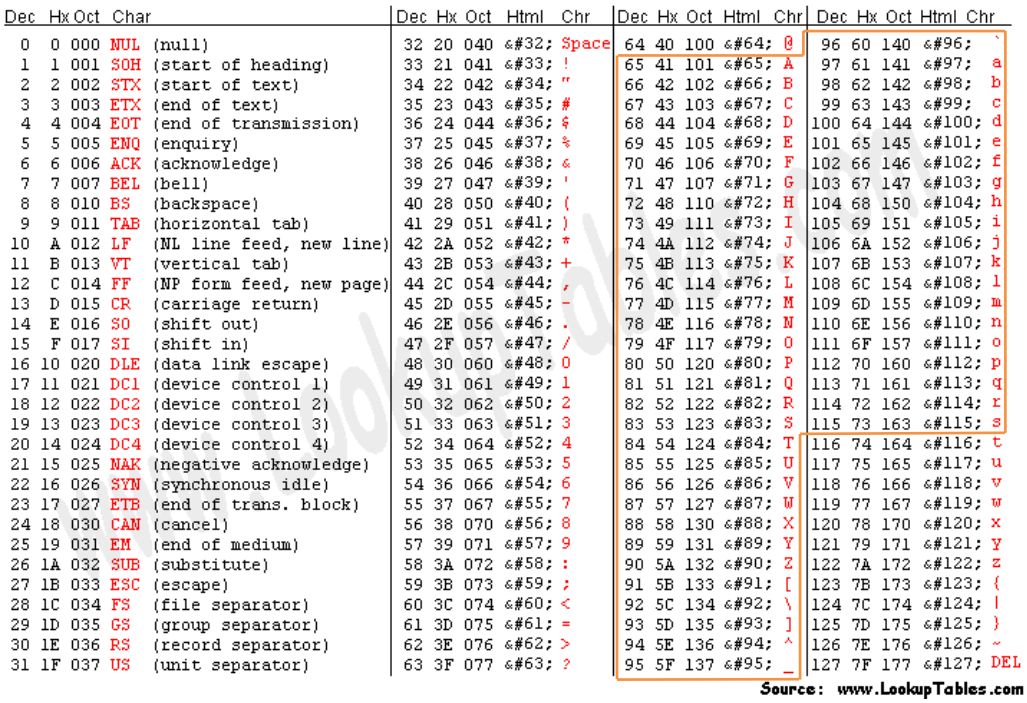


Figure 4.15 – Flowchart illustrating the data type conversion process of the transmitted steering angle message

If an obstacle is detected, the Arduino generates an unsigned integer value (65-115) corresponding to the steering angle that is needed in order to avoid the detected obstacle(s). However, the unsigned integer must be converted into a character in order to be serially transmitted over the USART to the STM32. Referencing the ASCII table provided below in Figure 4.16 depicts the one to one mapping between the serial characters and the unsigned integer values. Once serially transmitted, the character is then converted back to its original unsigned integer value. The controller node will receive this value from the sensor node over the CAN bus. The unsigned integer can then be subtracted by 90 and packaged into a CAN message again which, when transmitted, can then be interpreted by the steering node as a steering angle set point for the front two wheels (in the -25°-25° range). Thus, when an obstacle is detected, the steering node will receive the new steering command CAN message in the span of two transmission cycles (200 ms).



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Figure 4.16 - ASCII character chart (Retrieved from: http://www.jerrypeng.org/tutorials/ascii.html)

4.4 Results & Discussion

The UAGV underwent a series of verification tests to evaluate and validate the performance of the proposed obstacle avoidance scheme. Since the obstacle detection and avoidance algorithm was designed to operate in conjunction with the developed GPS waypoint navigation algorithm, the tests were performed where obstacles or barriers were placed at varying positions along a GPS assigned route. The data acquired from these test runs will be presented and discussed in this section. Figure 4.17 below depicts the selected physical barriers that were used in the obstacle avoidance test runs.

Figure 4.17 - Selected objects to serve as obstructions in the conducted obstacle avoidance tests

It was decided that the same GPS route will be issued for all of the test runs that would be performed. For the sake of simplicity, the assigned route will simply be a straight line containing just two GPS waypoints. The variables to alter in the experiment will be the number of obstructions and the placement of each obstruction. All test runs were conducted on the Nebraska Tractor Test Track that was depicted in Figure 3.17. The base station was kept in the same location as before for the conducted obstacle avoidance test runs. Thus, all distance measurements shown in the position map that depict the vehicle's X-Y position are location readings relative to the base station location depicted in Figure 3.17.

The first set of experiments called for a very basic test set up where only a single obstacle was placed at either the right or left side of the UAGV and was positioned somewhere in between the two GPS assigned coordinates. The two figures below illustrate the results, revealing the performance of the implemented avoidance scheme.

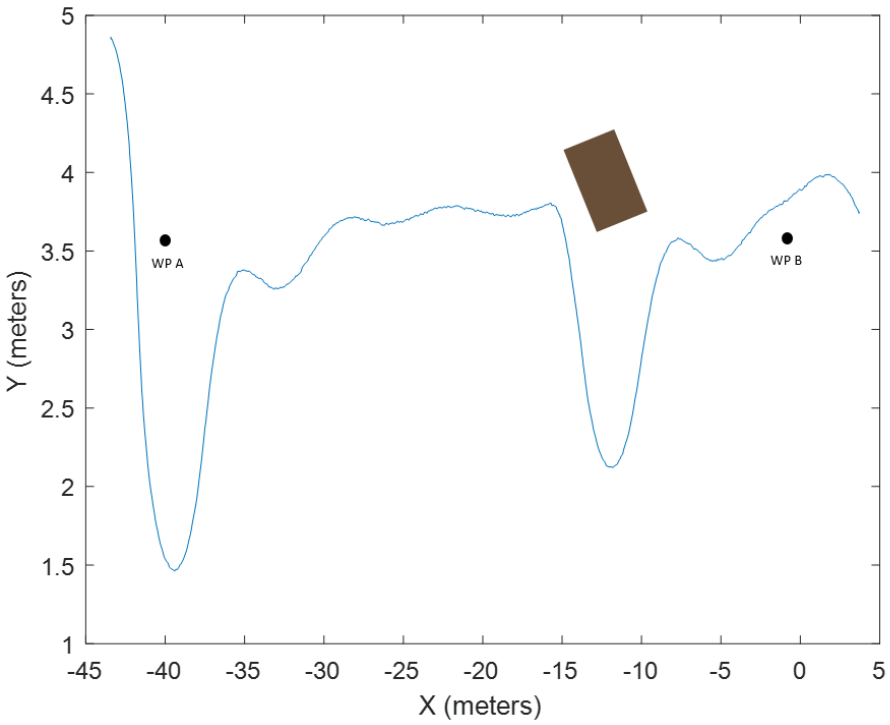


Figure 4.18 - Position map for test run where an obstacle was positioned to the left of the UAGV

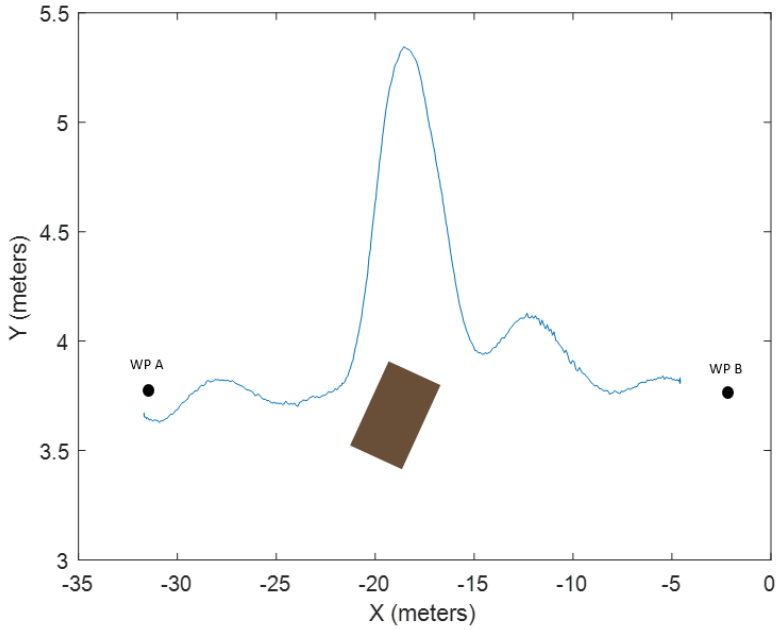


Figure 4.19 - Position map for test run where an obstacle was positioned to the right of the UAGV

The two figures above demonstrate the UAGV temporarily diverging off course in attempt to avoid the detected obstacle on the left (Figure 4.18) and the right (Figure 4.19). When the vehicle has determined that it has safely maneuvered around the identified obstruction, the UAGV will revert back to autonomous (GPS) mode where it will return to driving down or tracking the preplanned course. Figure 4.20 illustrates a slightly more challenging experiment where the UAGV is presented with an assortment of objects that it must avoid and navigate around. This more advanced test run allowed for the reliability and robustness of the implemented avoidance scheme to be examined.

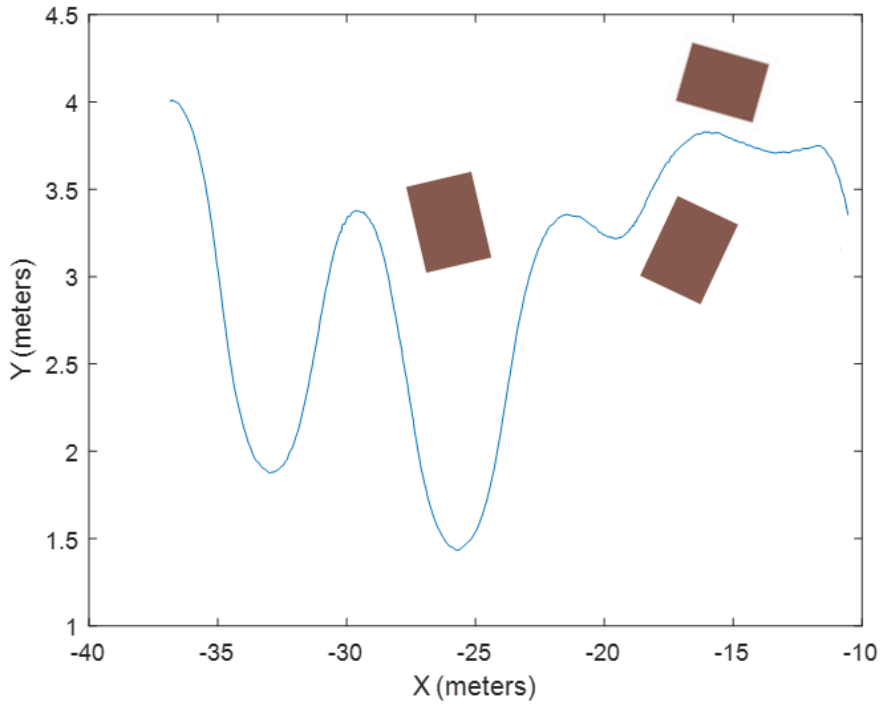


Figure 4.20 – Position map for test run where an assortment of obstacles were placed in the UAGV's path

Figure 4.21 shows the results from a test run where the two large objects were then placed closer together at a distance that happened to be less than the width of the vehicle. This test simulated the scenario for when obstacles are unavoidable due to the kinematic constraints and physical dimensions of the UAGV, thus verifying that the vehicle would promptly halt its autonomous operation if a detected obstacle was

measured to be too close to be avoided or the largest available pathway was calculated to be too small for the vehicle to safely pass through.

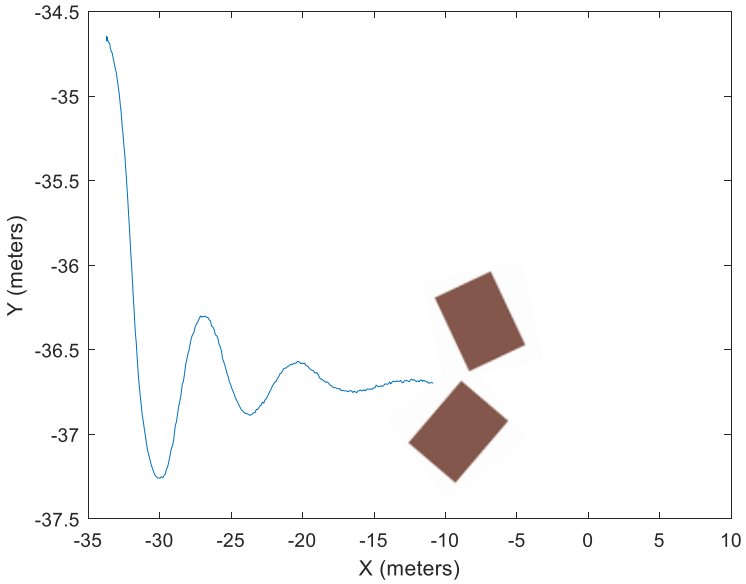


Figure 4.21 – Position map for test run where the UAGV was faced with obstructions that were unavoidable

The conducted test trials provided empirical evidence that the implemented obstacle detection and avoidance scheme on the UAGV functioned as intended. The selected LIDAR sensor would reliably detect all of the selected obstructions (shown in Figure 4.17), and the selected threshold range of 4.5 meters provided enough time for the vehicle to react and maneuver around the detected obstacle(s). The scheme does leave some room for improvement however. During the experimental tests, it was noticed that the sensor would occasionally erroneously detect obstacles when the path in front of the vehicle was in fact, obstruction-free. Applying some additional filtering onto the sensor readings would likely improve the sensor's performance and eradicate those erroneous readings. Additionally, it was discovered that the obstacle avoidance algorithm would perform less reliably when obstacles were placed on both sides of the vehicle, a functional performance flaw that originates from the nature of how the obstacle detection scheme was designed. Since the utilized vision sensor is a 1D device and requires the

deployment of a stepper to effectively make the sensor 2D, the sensor is unable to instantaneously obtain a vision map of what is in front of it. Rather, the vision map is obtained incrementally in the span of about a second by rotating the stepper from left to right. Thus, since these measurements are obtained while the vehicle is moving, objects located on the left side of the vehicle will appear slightly further away than objects positioned on the right, which can easily frustrate the computed steering angle result. If this current vision sensor design topology remains in place for future operation, it is recommended that the vehicle be programmed to just stop when obstructions are detected on both sides of the UAGV. Scaling the LIDAR readings to offset these distance measurement deviations between the left and right LIDAR scans, is also a potentially viable solution that would be worth exploring.

5 Sensor Based Navigation

The second agricultural ground robot that was further developed in this thesis is the Inter-row follower. In this chapter, all of the accomplished work related to this ground vehicle will be discussed. In the first section, the necessary background information will be given in which the reader will be given a general top level overview of all the design aspects of the ground vehicle. The reader will be briefed on the laser based under the canopy navigation scheme that was designed and developed by the former graduate student. Additionally, the first section will also be supplemented with a light introduction to the additional work that was performed on the machine, where design changes made to the hardware and firmware architecture to accommodate for the added functionality, will be noted and introduced here. In the second section, the additional work accomplished on this robot will be discussed in detail where a fully autonomous non-GPS sensor based algorithm was developed and implemented onto the machine to yield a robot capable of autonomously navigating its way through a field via the utilization of just three different types of sensors. All of the auxiliary functionality associated with this algorithm along with the implemented embedded system design will be diligently analyzed and explained. Finally, the chapter will conclude with an examination and discussion of the tests results obtained from the navigation experiments performed both in the laboratory and in the field.

5.1 Background

Since the additional work performed on this robot further expounded on the work completed by the precedent graduate student, it is necessary to first offer a brief, but informative recapitulation of the project that was inherited, addressing robot's initial stage of development. That way, the reader will be adequately informed on the initial project state and design configuration, and can therefore follow the supplementary work that was performed. A light, top level overview of the additional work performed on the robot will also be offered in this section. The first important piece of information to address is the motivation behind the development of this robot.

5.1.1 Intended Application

The discovery of the GPS is responsible for many of the drastic changes being observed in the agriculture industry today. Developing GPS based navigation schemes has allowed for the unmanned automation of agricultural related tasks to really gain momentum in the agriculture sector over the past decade as it is starting to become a feasible alternative to manual labor. However, there are also numerous incidents where relying on a GPS for navigation is simply not practical.

In the realm of agriculture, there are certain situations where the GPS reception is unreliable and cannot be depended on for navigation. The most obvious example of such a circumstance is crop row navigation. Overhang of crop canopies can easily be enough to interfere with a GPS signal, thereby rendering any GPS based navigation scheme ineffective [5]. The most prominent application for under-canopy navigation and thus, the motivation for the development of this robot, is microclimate sensing. Microclimate sensing is the utilization of various types of sensors to perform a diversified set of crop measurements that are useful for studying and analyzing the stalks' overall health. The collected information will be especially valuable for accessing and predicting crop yield.

Under canopy navigation in particular, can reveal crop information that is difficult to achieve from the alternative method of deploying unmanned aerial vehicles (UAVs). UAVs are limited in the sense that they are mostly only able to measure and collect data from the top portion of crops, where under canopy navigation allows for the crop measurements to be taken anywhere along the height of the stalk. Thus, under-canopy navigation allows for more informative, comprehensive crop data to be collected which will aid in being able to more accurately predict and access crop yield in a field. Hence, developing reliable autonomous navigation solutions in cluttered crop canopies, where GPS signals are capricious, is a challenging yet prevalent problem to address in the agriculture industry today. As noted in [46], developing a means for the reliable navigation of autonomous machines in the absence of a GPS is crucial to the development and design of low-cost, light, compact, agricultural ground robots. The attention thus, turns to adding supplementary navigation instruments to aid and improve navigation reliability.

Developing an autonomous machine which utilizes sub-canopy sensors and instruments to successfully navigate through crop rows, in the presence of unreliable GPS reception, is the exact problem that the Inter-row robot was designed to solve.

5.1.2 System Architecture Overview

The robot platform that housed all of the developed electronics was a six wheeled aluminum prototype frame, denoted as the "Dagu Wild Thumper", which served as the robot's chassis. Additionally, each motor-wheel had a clamped spring for added suspension. This helped compensate for unpredictable external disturbances in the environment such as dirt clods, puddles, and pivot tracks [5]. Additionally, a front buffer, leaf guard ring, and mounting brackets for the two distance sensors were installed onto the platform.

The top surface of the aluminum frame contained a grid of equally spaced mounting holes which allowed for rapid prototyping where all of the hardware could be easily mounted, tested, modified, and/or removed accordingly during the design and development stages. All of the hardware that was mounted onto the prototype frame will be addressed in section 7.1.4. The robot chassis was driven by a 2x12 12 A Sabertooth motor driver, where a differential steering approach was employed. A depiction of robot prototype platform with all of the installed hardware is offered below in Figure 5.1.

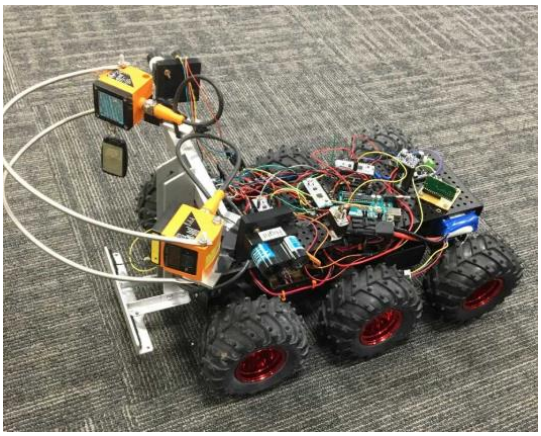


Figure 5.1 - Inter-row robot with all of the leveraged sensors and developed electronics fully mounted and installed onto the mechanical chassis

5.1.3 Distributive Real-Time System Overview

As was the case with the UAGV, the Inter-row robot exhibits a DRTS architecture. The DRTS for the Inter-row follower consists of four nodes. Like the UAGV, the agricultural ground vehicle employs CAN as the communication protocol for multi-subsystem interfacing and data transportation among all the developed nodes. Figure 5.2 illustrates the CAN based DRTS with all of the associated modules that were interfaced to each node.

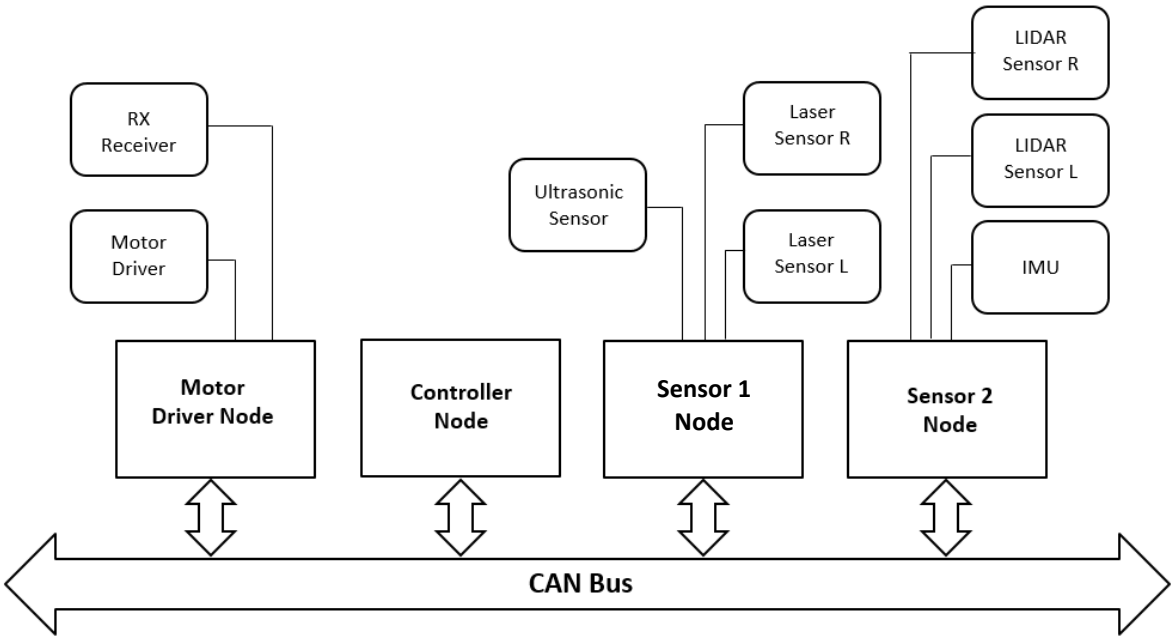


Figure 5.2 - Distributive Real-Time System of the Inter-row follower

Motor Driver Node: Sends the appropriate serial commands to the motor driver on the Inter-row to skid steer the robot accordingly.

Controller Node: Can be thought of as the CPU of the entire embedded system. Governs the FSM which controls the system operating mode of the robot. Receives data from the sensor 1 and sensor 2 nodes, processes them according to robot's current operating state, and then transmits the appropriate motor command signals over to the motor driver node. Additionally, the controller node will transmit state machine messages out to the sensor 1 and 2 nodes to keep them updated on the robot's current operating

mode to ensure that each node is executing the appropriate task, specific to the robot's active operating state.

Sensor 1 Node: Utilizes the two laser distance sensors for row following navigation and the ultrasonic sensor for emergency stopping. Receives state machine control signals from the controller node which notifies the node on whether or not to leverage the laser sensors for row following navigation.

Sensor 2 Node: An IMU is utilized to obtain the yaw or heading angle of the machine which is essential for turning in the headland. Additionally, LIDAR sensors are used for crop row detection. Like the sensor 1 node, the sensor 2 node receives state machine control signals from the controller node which informs the node on the application task to execute (i.e., which sensor modules to utilize for navigation).

5.1.4 CAN Hardware Implementation Overview

In this section, an overview of the implemented hardware for the Inter-row robot will be given. This involves having a discussion of the selected microcontrollers and the corresponding PCB that was interfaced to it. It also entails of giving the reader a brief overview of the system components that comprised the Inter-row robot. Finally, it also requires addressing the modified wiring topology that the robot employed.

5.1.4.1 *Board Design*

As noted in the previous section, the Inter-row robot is comprised of four nodes. In this hardware setup, the STM32f308 microcontroller shown in Figure 2.6 was used for all four of the system nodes. Thus, the corresponding CAN PCB (as was shown in Figure 2.9) was originally implemented onto each of the microcontrollers in the initial design. However, as the embedded firmware continued to evolve as a result of integrating additional functionally onto the machine, it became necessary to revise the board design to accommodate for the various software and hardware modifications that had been made to the originally inherited design. The final board design, depicted below in Figure 5.3, is specifically designed to be compatible with the latest row follower design configuration, which contains all of the recently added

sensors along with the upgraded firmware which includes all of the developed supplementary functionality. The board contains additional sets of terminal blocks that are routed on the PCB to the appropriate peripherals on the microcontroller, which are to be interfaced to all of the appropriate external modules that constitute the system (motor drivers, IMUs, vision sensors, etc.). Each terminal block is labeled accordingly in the figure.

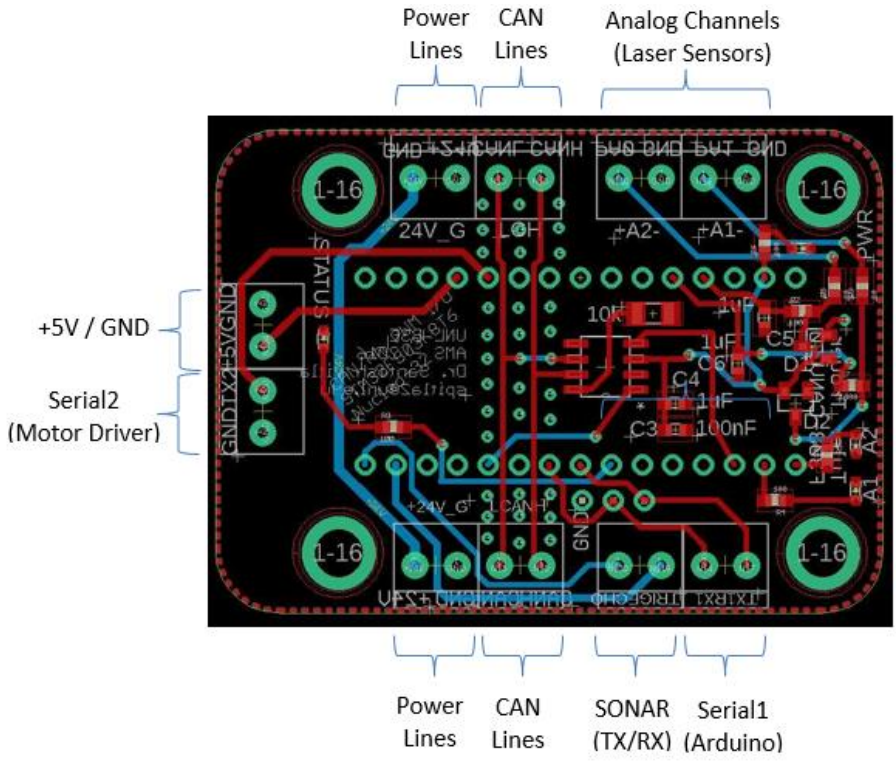


Figure 5.3 - Modified CAN F303k8 PCB tailored to the latest Row Follower system design

5.1.4.2 ***Hardware Components***

To power the robot, a 13 V 3-cell lithium battery (5000mAh 3S, Turnigy, Hong Kong, CN) was used. A power unit module (DFR0205) supplied regulated power to the Kvaser CAN data logger (Figure 2.3) which would be added to the CAN bus for system data collection purposes during field navigation tests. The power module was tied to a d'sub connector attached at the rear of the robot which allowed for the Kvaser to easily be connected onto the CAN bus. The battery and power module are shown below in Figure 5.4 and Figure 5.5.

Figure 5.4 - 13 V 3 cell lithium battery (Retrieved from https://hobbyking.com/en_us/turnigy-battery-heavy-duty-5000mah-6s-60c-lipo-pack-xt-90.html)

Figure 5.5 - Power module unit (adjustable buck converter) (Retrieved from https://core-electronics.com.au/dc-dc-power-module-25w.html)

The selected distance sensors to be mounted at the front of the robot were O1D100 laser sensors from IFM-efector. As was shown in Figure 5.1, they were placed at the front of the robot, each positioned inwards at 45° in opposite directions to provide distance measurements on both sides of the crop row. They operated at 24 V DC and provided an analog output voltage (0-10 V DC) proportional to the detected distance of the closest object within range of the sensor. Consequently, a boost converter was leveraged to step the approximate input voltage of 13 V coming from the battery to 24 V to power the laser sensors. Additionally, a voltage divider was implemented on the distance sensors to drop the maximum output voltage of 10 V down to 3.33 V to ensure safe operating levels for the pins on the microcontroller of the sensor 1 node. These laser sensors were leveraged for under-canopy crop row navigation as will be discussed in the subsequent section. Ultrasonic (HC-SR04) and LIDAR sensors (HC-SR04) were also implemented onto the two sensor nodes. The LIDAR sensors were utilized to detect

the starting and ending points of crop rows, while the ultrasonic sensor was used for emergency stopping situations. Additionally, an inertial measurement unit (IMU) was instrumented onto the system to allow for precision steering in the headland for navigating over to the next subsequent row in the field. Furthermore, the same 433 MHz receiver module that was used on the UAGV was also implemented onto the motor driver node of the Inter-row-robot to allow the user to easily start and stop the robot via the remote. The same power module shown in Figure 5.5 that was used for powering the Kvaser, was also used to supply regulated power to the entire CAN bus system, where the module was placed in between the battery power source and all of the system nodes connected onto the CAN bus. The selected DC motor driver was a dual channel 2x12 12 A Sabertooth module, which is depicted below in Figure 5.6.

Figure 5.6 -2x12 Sabertooth 12 A motor driver module (Retrieved from https://www.cooking-hacks.com/sabertooth-dual-12a-motor-driver)

The motor driver could either be controlled serially (over the USART) via 7 bit characters or by an analog voltage that can be controlled via PWM. This is an important design change to take note of as the prior configuration used Victor SP motor drivers which could only be driven by PWM (hence the PWM channels that can be observed on the original 308k8 PCB in Figure 2.9). The Sabertooth motor driver presented the compelling advantage of being able to control the polarity (and thus the direction) of the motors. This is an imperative feature for the goal of being able to have the robot turn in the headland once

the end of the row has been reached. Additionally, PWM control bottlenecked the capability of the original motor driver as the STM32 microcontrollers generated a maximum output voltage of only 3.3 V, while the DC motors on the mechanical chassis had a 7.5 V rating. Thus, the serial control capabilities that the new motor driver module provided, remedied this dilemma as the driver was no longer constrained by the limited voltage available from the microcontroller pins, thus allowing for the motors on the chassis to be fully utilized if necessary.

5.1.4.3 ***Wiring Topology***

An additional important modification that was made to the design architecture of the Inter-row that is worth drawing attention to is the implemented wiring topology. As mentioned earlier, a daisy chain wiring scheme was originally employed, and while this topology proved suitable for the operation of the original three system nodes, the configuration was rendered ineffective when the second sensor node was added on to the system. The daisy chain wiring topology inherently connects all of the nodes in series which therefore equally divides the voltage of the power source across each node. While this wiring scheme worked suitably for the original three nodes in use, adding an additional node to the system consequently dropped the supplied voltage across each node below the voltage threshold required to effectively power and turn on each microcontroller. Thus, the daisy chain wiring scheme was replaced with a star configuration, which effectively connects all of the nodes in parallel. This rectified the voltage power up dilemma as this wiring setup allows all of the system nodes to receive the same amount of voltage, equivalent to the voltage of the power device (i.e., 13 V battery), regardless of the number of nodes in the system. An illustration of the two wiring topologies is shown below in Figure 5.7.

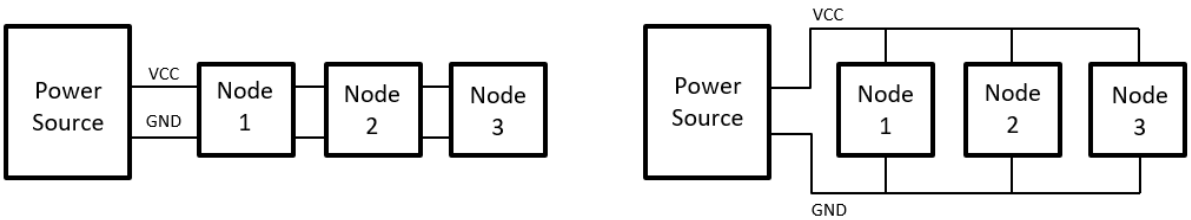


Figure 5.7 - Daisy chain (left) vs star configuration (right)

5.1.5 CAN Firmware Implementation Overview

In this section, an overview of the implemented firmware for the Inter-row robot will be presented. This involves observing the data packing scheme for the CAN messages to be transmitted, addressing the evented triggered system architecture setup, and finally, examining all the implemented peripherals in the firmware that enabled all the nodes to perform their designated tasks.

5.1.5.1 *CAN System Messages*

There are several CAN messages that needed to be configured to allow for complete communication among all of the four nodes to ensure proper system operation. These include CAN messages for the various measurement sensors, flags or event trigger messages that notify the controller node of the occurrence of a significant event, and state machine control messages that notify the sensor nodes of a state transition within the operating FSM. The message packaging scheme for the newly developed system messages can be seen below in Figures 5.8 - 5.11.

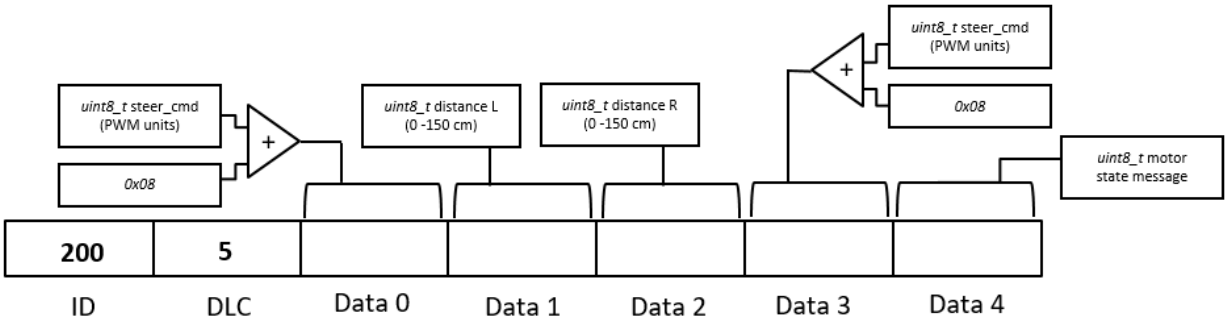


Figure 5.8 - Message packaging scheme for motor command message

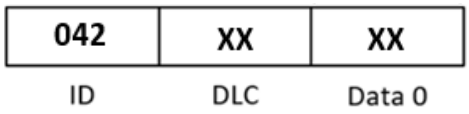


Figure 5.9 – Message packaging scheme for IMU control signal

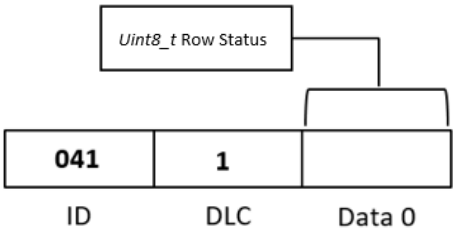


Figure 5.10 - Message packaging scheme for row status control signal

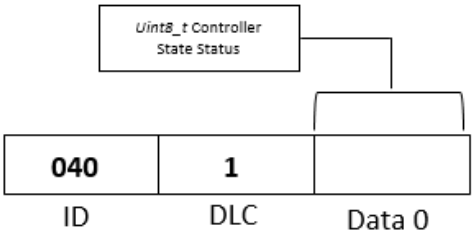


Figure 5.11 - Message packaging scheme for controller state (FSM) message

Descriptions of each of the CAN system message IDs (not just the newly developed ones shown above) are given below.

- 042 – *IMU Message*: Flag or event triggered message that gets transmitted by the sensor 2 node to notify the controller node when the robot has completed a 90° turn.
- 041 – *Row Status Message*: Flag or event triggered message that gets transmitted by the sensor 2 node and is used to notify the controller node of start/end of crop row detections.
- 040 – *FSM Controller State Message*: Message contains the information regarding the controller's current operating state in the FSM. Message gets transmitted by the controller node to the two sensor nodes after a state transition occurs to synchronize all the nodes with the central FSM being managed by the controller. Consequently, each node is therefore able to respond accordingly by executing the appropriate task according to the new operating state.
- 201 – *Distance Sensor Measurements (not shown here)*: Distance sensor measurements obtained from the two laser scanners. The measurements are produced by the sensor 1 node and transmitted over the CAN bus to the controller node.

- 200 – *Motor Message*: Messages transmitted by the controller node over to the motor driver node. The messages are processed by the motor driver node and are converted into serial commands to be sent to the motor driver module to steer and drive the robot accordingly. The information contained in the data [4] byte informs the motor node of the controller node's current operating state in the FSM, which dictates the resulting transmitted serial motors commands.

The FSM control message (040) is the CAN message that gets transmitted by the controller node after a state transition as it contains the content that informs all of the other system nodes on the bus of the controller's new state or operating mode within the FSM. Table 5.1 below outlines the data content to operating state mapping where each operating mode within the FSM gets assigned to a particular number in the data [0] byte of the CAN FSM message. The table effectively illustrates all of the developed operating states of the Inter-row. The various states of the Inter-row's FSM will be explained in detail in section 5.2.5.

CAN FSM Message (040) – Data[0] Content	State (Operating Mode) in FSM
0x00	*Enter Row Mode*
0x01	*Follow Row Mode*
0x02	*Leave Row Mode*
0x03	*Turn Left (90°) Mode*
0x04	*Turn Right (90°) Mode*
0x05	*Move to Left Row*
0x06	*Move to Right Row*
0x07	*Stop Mode*

Table 5.1 - CAN FSM message for operating modes of the Inter-row

As already indicated in the CAN message descriptions, the row status message (041) and IMU message (042) can be thought of as flags or event trigger messages that are transmitted by the sensor 2 node. They will effectively trigger a state transition within the FSM at the controller node end. These messages are essentially control signals that notify the controller node of the occurrence of an important event. The message will therefore be processed by the controller node for state transitioning within the FSM. The descriptions detailing these two messages' contents are outlined below in Table 5.2.

CAN ID	Data[0] Content	Message
041	0x03	End of row has been detected
041	0xFF	Start of row has been detected
041	0xAA	Next row over has been detected
042	--	Robot has turned 90° (L/R)

Table 5.2 - CAN control message descriptions

Additionally, as mentioned earlier in the motor CAN message description, the motor node gets continuously updated on the controller node's current state via the content contained in the data [4] byte of the motor message (200). The information contained in data [4] notifies the motor node of the current operating state, thus informing the node on whether the robot is to turn on a dime (*Turn Left/Right Modes)*, drive forward at a constant speed (*Move to Next Left/Right Row Mode* or *Enter Row Mode*), or use the data received in data [0] to steer the robot accordingly (*Follow Row Mode*). The content contained in data [4] of the motor state message that determine the controller state and thus, the serial motor command to transmit to the motor driver module, is outlined and decrypted below in Table 5.3.

CAN Message – Data[4] Content	State (Operating Mode) of the Controller Node
0	*Follow Row Mode*
1	*Move to Left/Right Row*
2	*Turn Left (90°) Mode*
3	*Turn Right (90°) Mode*
4	*Enter Row Mode*
5	*Stop Mode*

Table 5.3 - CAN motor message for operating modes of the Inter-row

The content contained in the data [4] byte of the CAN motor message can essentially be equated to the select lines of a multiplexer, where the data contained in the byte determines which motor command will get routed (multiplexed) to the motor driver module. An illustration of this concept is offered below in Figure 5.12.

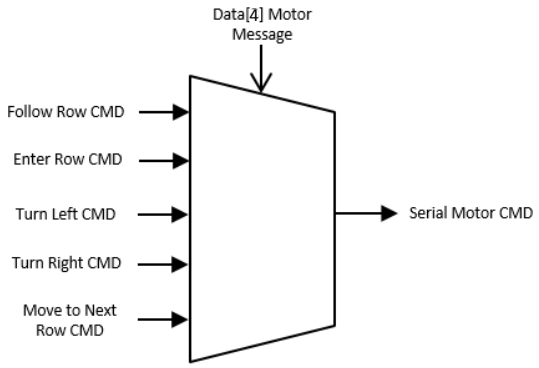


Figure 5.12 – Motor driver node processing the CAN motor message via a multiplexer

5.1.5.2 ***Embedded System Design Implementation***

It is now time to address the original development stage the Inter-row robot was in when the project was initially inherited. More specifically, we will examine the embedded system design architecture that was in place. This will then allow the reader to follow the supplementary work that was accomplished which will be discussed in detail in section 5.2.

At the sensor 1 node, the two distance sensors were sampled at a frequency of 250 Hz, a process controlled by a hardware timer. The sampled analog signals were converted to digital values via a 12 bit analog to digital converter (ADC) coupled with a direct memory access (DMA) controller. The digital voltages could then be mapped to their corresponding distance values which could then be packaged into a CAN message to be transmitted out onto the CAN bus for the controller node to process.

At the controller node, the received distance values from the sensor 1 node could then be processed and handled accordingly. Since the primary application for the Inter-row robot is under-canopy crop row navigation without the aid of a GPS, the problem demanded for a robust sensor based navigation scheme. The proposed solution was to employ a Kalman filter. A Kalman filter is a computationally recursive algorithm that achieves an optimal averaging factor via a predictor-corrector type estimator scheme [47]. The Kalman filter takes into account the correlation between the different types of measurements being performed, the uncertainty of external disturbances, and the existing sensor noise to obtain an increasingly

accurate estimate of the measured state as time progresses, which results in the minimization of the overall error covariance [47]. [48] offers a complete, in-depth analysis of the Kalman filter where all of the necessary mathematical concepts and principles that are foundational to the Kalman filter design are introduced and explained in detail. Additionally, all of the complex, rigorous equations and matrices that comprise the filter are outlined and thoroughly discussed. Though the filter has a wide range of applications, the application pertinent to this thesis is the filter's usefulness in aiding with guidance navigation. [49]-[51] offer just a few of the many examples where a Kalman filter served as the primary navigation and localization algorithm for a mobile vehicle in an agricultural setting. In this application, the Kalman filter works to accommodate for the variability in the crop rows, filtering out outliers such as overhanging leaves and inner-row gaps between each stalk to more accurately estimate and predict the robot's lateral position within the field row.

Thus, the individual distance measurements received from the two laser distance sensors were each sent through a Kalman filter. Once filtered, the difference between the two distance measurements was taken to obtain an error signal. If the robot was perfectly centered in the row, the distance measurements for the two laser sensors should both be equivalent, and the error signal would thus be zero. A PI control scheme was then implemented to determine and generate the appropriate motor command based on the calculated error signal. The motors would then be actuated accordingly, where the measured error would be driven down to zero, therefore maintaining a centered position for the robot as it navigates through the crop row. The controller node would then transmit the motor commands over the CAN bus to the motor driver node. The motor driver node would then process these motor command messages and generate the corresponding serial commands to send to the motor driver module. An in depth explanation of this navigation setup, supplemented with visual aids, is offered in pages 17-21 and 30-33 of [5].

This proposed sensor based navigation scheme was thoroughly tested in the fields and yielded satisfactory results. The data from all the tests can be found in pages 33-38 of [5]. Thus, when this project was

inherited, the embedded system design framework for a reliable laser-based navigation scheme for under-canopy applications was fully in place and implemented onto the Inter-row machine.

An important note to make here is that, implicit in the embedded system design description, the Inter-row robot employs an event triggered system design approach, contrary to the UAGV which demonstrates a time triggered architecture. The whole system is governed (or "triggered") by the various measurement sensor readings which serve as the "event" that drives the operation of the entire system. The distance readings for the sensors (or the "event") triggers the controller to process the filtered data and to run the control algorithm which determines the appropriate motor commands to be transmitted to the motor controller node. Since the distance sensors are sampled at a rate of 250 Hz, the actual event is inherently driven by time. Thus, the system can be thought of as an event triggered system where the event itself is innately time triggered.

5.2 The Non-GPS Based Fully Autonomous Navigation Algorithm

Although the Inter-row robot was developed to the point where it could reliably navigate through crop rows using just the two installed laser sensors, the robot still had no way knowing when the end of a row (or start of a row) was reached. Moreover, it was also unable to navigate around to the next successive row in the field. This limited autonomy prevents the Inter-row robot from serving as a practical, useful solution for the vast majority of agricultural related tasks. Thus, the next important stage of further development on the Inter-row robot was to further expound upon this design to make the robot fully autonomous. This involves developing a mechanism where the robot could successfully detect the starting and ending points of crop rows. Additionally, it requires deriving a navigational method to accomplish the headland turning required to properly reposition the robot over to the next adjacent row. The end goal is to have a fully autonomous robot that is able to navigate through an entire field without the need for any human intervention. This demand thus inherently calls for the development of a robust, non-GPS sensor based fully autonomous navigational algorithm that is specifically tailored to the smaller UAGVs that are utilized and designed for field row navigation in under canopy sensing applications. The proposed and

developed fully autonomous navigational scheme that was designed in this thesis as a feasible solution to this problem will now be presented. The algorithmic concepts will first be introduced and explained. This will be followed by an in depth examination of the additional node (with the supplementary sensors) that was developed and instrumented onto the Inter-row to realize the algorithm. Finally, the embedded system design and the implemented FSM for the algorithm will be discussed and described in detail.

5.2.1 End of Row Detection

In order for the robot to be able to navigate around rows at the end of a field, the robot first has to be given the ability to recognize when it has reached the end of the current row it is driving through. The same LIDAR sensor that was used on the UAGV for obstacle detection and avoidance was also selected to be used on the Inter-row to handle this task.

Two LIDAR sensors were thus implemented onto the newly developed sensor 2 node. As shown in Figure 5.1, the sensors were placed on the left and right front sides of the robot platform, facing directly outwards. This location enabled the sensors to be able to detect the beginning and ending points of field rows. In this new configuration, the sensor 2 node would now be able to receive lateral distance readings on both sides of the robot from the newly added LIDAR sensors.

For detecting the end of crop rows, gaps between the stalk rows could potentially cause the robot to erroneously perceive that it has reached the end of the crop row. To account for this concern, the sensor 2 node was designed so that it would only initiate an “end of row” detection event after a predetermined sufficient number of consecutive distance readings measured above a certain threshold defined in the firmware. The defined threshold can be equated to a set distance value that is greater than the crop row width in the field. Additionally, when the robot is located at the very beginning of a field row, the robot needs to able to reliably identify when it has successfully entered that particular crop row. The same threshold used to trigger the “end of row” detection could also be employed to enact a “start of row” notification. Thus, this design configuration allowed for the sensor 2 node to utilize the LIDAR sensor to

search for and reliably identify the beginning and ending point of crop rows in the field, while the sensor 1 node would simultaneously read and use the laser distance measurements for navigating the robot through the crop row. Figure 5.13 depicts the "end of row" detection process just described.

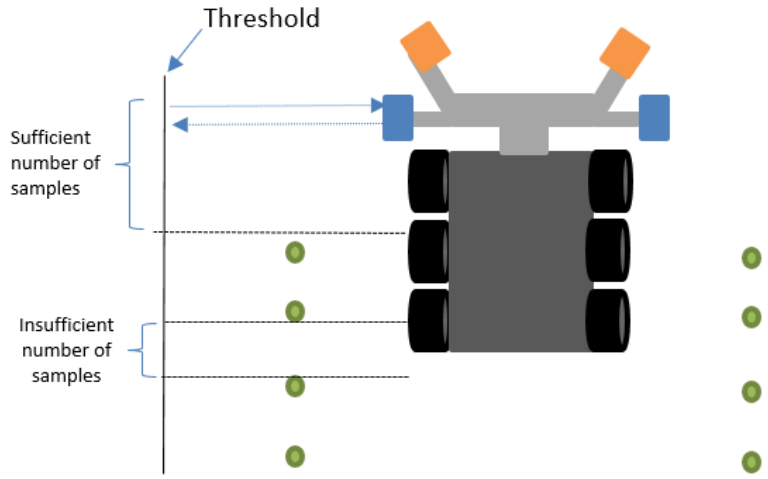


Figure 5.13 - Row detection scheme illustration

5.2.2 Headland Turning and Repositioning

Once the end of the crop row has been detected, the robot must have an implemented navigation scheme to be able to maneuver over to the next successive row in the field. There are a couple of different approaches one can take to solve this problem. In [52], a variable field of view (FOV) camera was used to guide the robot over to the next row. A similar FOV method was proposed in [53] for navigation between rows in a field. In [54], a predefined path in the form of a Bezier curve was implemented onto a robot to handle the headland turning requirements in an agricultural field. In [55], odometers were employed to allow the agricultural ground vehicle to maneuver its way over to the next row in the field. Another alternative to these proposed schemes is to employ a GPS driven approach. Since the robot is no longer operating under the canopy, it should theoretically be easier to establish a reliable GPS signal, implying that GPS waypoint navigation could serve as a viable approach to handling the headland steering requirements. However, the successfulness of this approach is still contingent on there always being a reliable GPS signal at the end of every row, which is a volatile assumption to hold. Another possible

solution would be to implement a sensor guided turning scheme where the LIDAR sensors can be used to guide the robot over to the next row and an IMU could be utilized to handle the precise steering requirements, somewhat similar to the design implementation for the agricultural robot described in [56].

In the final design approach that was ultimately selected, the same LIDAR sensors that were used to detect the beginning and ending points in the field rows was also used to guide the robot over to the next row. Meanwhile, an IMU can handle the precise turning requirements needed to successfully navigate the Inter-row over to the next adjoining row in the field for the robot to then navigate through. The selected IMU was a BNO055. The IMU is a 9 degree of freedom (DOF) sensor and contained an accelerometer, gyroscope, and magnetometer that were all implemented onto an ARM Cortex-M0 based processor. The IMU was chosen due to the available developed low-level driver firmware that allows the sensor to easily be interfaced with an Arduino which expedited the development time. The IMU is depicted below in Figure 5.14.

Figure 5.14 - BNO055 9 DOF IMU sensor (Retrieved from https://www.amazon.com/Adafruit-Absolute-Orientation-Fusion-Breakout/dp/B017PEIGIG)

Utilizing an IMU to aid in the precise steering control of mobile vehicles is a concept that has long been vigorously researched and developed and is thus well documented in literature. [57] discusses the implementation of well-established data fusion algorithms (in form of a complimentary filter and a Kalman filter) onto a real time embedded system platform. The developed algorithms optimally combine the data received from the three sensors that comprise the IMU to achieve accurate position and

orientation estimation. In [58], a Kalman filter is developed that effectively fuses the data received from a gyroscope and magnetometer to obtain an accurate estimate of the yaw angle of the robot. In [59], a real-time turning angle acquisition method is developed on an agricultural ground robot, utilizing a tri-axial accelerometer sensor.

It is also very common for IMUs to be utilized as secondary sensors to compliment the primary sensing system, which is usually a GPS. This notion holds true for many of the autonomous agricultural ground vehicles in development today. Such examples can found in [60]-[63]. In this particular design, the IMU will operate in conjunction with the LIDAR sensors to guide and steer the robot over to the next row.

The proposed algorithm for headland navigation can be broken down into several distinct stages. In the first stage, the LIDAR sensors detect when the robot has reached the end of the crop row which results in the transmission of the "end of row" message (041) from the sensor 2 node to the controller node. The controller node will then transmit the appropriate motor command message to the motor driver node to have the robot perform either a right or left turn. Meanwhile, the sensor 2 node will leverage the IMU to measure the yaw angle of the robot and will wait to observe either a positive or negative 90° delta heading angle, based on whether the robot is completing a left or right turn. This is accomplished via the pre-written low-level driver firmware for the IMU that integrates and fuses the data received from the gyroscope and magnetometer to obtain an optimal heading angle measurement. Once the 90° turn has been accomplished, the first stage of the algorithm has successfully been completed.

In the subsequent stage, the robot will then slowly start to drive forward while the sensor 2 node either utilizes the left or right LIDAR sensor to notify the robot when it has reached the next adjacent crop row. The LIDAR sensor will first be used to scan for a distance reading that falls within a pre-determined proximity threshold that was determined experimentally and is defined in the firmware. This notifies the robot that it has just detected the next subsequent crop row in the field. When this event has occurred, the sensor 2 node will then wait for a successive number of LIDAR readings (where the selected number was

again determined empirically) to fall outside of this proximity threshold. This event serves as the indicator that the robot has successfully reached the next gap corresponding to the next contiguous stalk row in the field. At which point, the Inter-row will then perform another 90° turn just as before and will then be properly aligned and positioned and ready to enter the next row in the field. Figure 5.15 outlines this entire procedure.

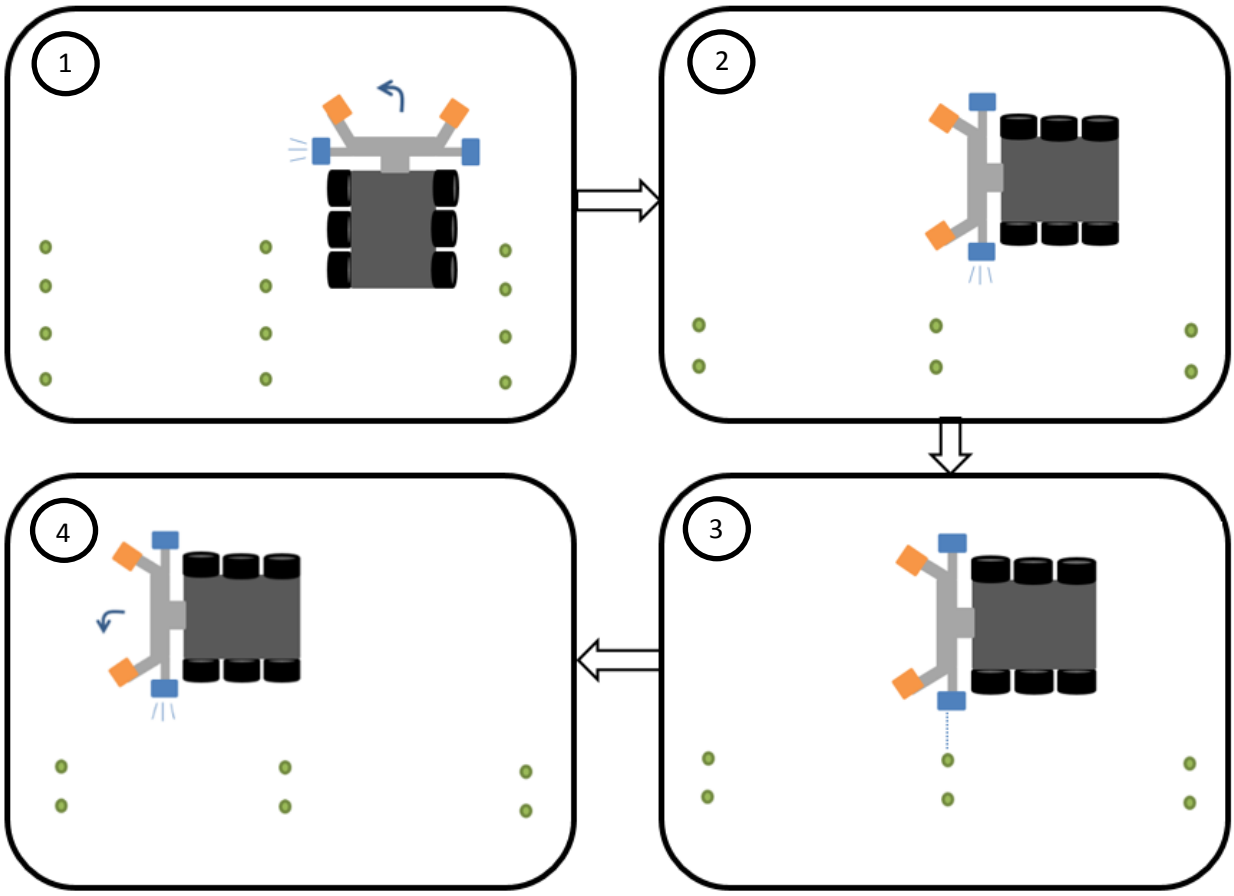


Figure 5.15 - Row repositioning (headland navigation) procedure

In stage 1 of Figure 5.15, the robot has detected the end of the row and will perform a 90° left turn. In stages 2-4, the LIDAR sensor is used to guide the Inter-row over to the next successive row. The sensor will first be used to look for LIDAR readings that fall within the defined proximity threshold which would indicate that the Inter-row has reached the next adjacent stalk row (stages 2-3). The LIDAR sensor will then be utilized to look for a successive number of LIDAR readings that fall outside of the proximity

threshold indicating the Inter-row has successfully made it over to the next neighboring field row (stage 4); at which point, the Inter-row will perform another 90° left turn and will be properly aligned and positioned to be able to enter and drive through the next succeeding row in the field.

5.2.3 Emergency Stopping

Since the robot is fully autonomous and driven by a multitude of sensors, erroneous readings from any of the sensors could cause the robot to go off path and/or potentially run into crops or any other obstructions that may happen to be present in the field. Thus, a simple ultrasonic sensor was selected to serve as an additional safety measure to the robot's operation and was implemented onto the sensor 1 node. Thus, the sensor 1 node would read the laser scanner sensor measurements for row guidance and navigation while simultaneously also taking ultrasonic readings. The sensor was placed directly in front of the robot as its purpose was to simply identify if there were any obstacles directly in front of the robot's path. The particular sensor that was chosen was a HC-SR04 as shown in Figure 5.16.

Figure 5.16 - Ultrasonic sensor (HC-SR04) (Retrieved from amazon.com/HC-SR04-Ultrasonic-Distance-Measuring-Sensor/dp/B00F167T2A)

Sonar technology, as was noted in section 4.1, served as a competent vision sensor technology in this simple application since the problem strictly demanded for observing vertical obstructions in the field that stand directly in front of the robot's path. Moreover, while it is certainly not a viable standalone sensor for autonomous navigation, it served as a nice complementary sensor to the central navigational laser/LIDAR and inertial sensors and acted as an additional safety layer to the robot's operation.

5.2.4 Sensor 2 Node Development

The developed sensor 2 node that was integrated into the Inter-row system design has a very similar design architecture to that of the LIDAR sensor node that was developed on the UAGV as was discussed in section 4.3. The selected IMU device came with pre-developed low-level driver libraries that handled all of the low-level sensor fusion and I2C communication between the IMU IC and the Arduino microcontroller. Thus, since the driver firmware was only compatible with the Arduino, an Arduino was chosen to interface directly with the IMU to accelerate the development time. As was the case with the developed LIDAR node, the STM32 microcontroller was also selected to be used on the node to handle the CAN communication with all of the other nodes in the system. Additionally, the two microcontrollers communicated with each other serially over the USART peripheral. The Arduino was interfaced to the IMU and LIDAR sensors over the I2C communication protocol. Since the node utilized two separate LIDAR sensors, each sensor needed to be given its own unique address in order for the Arduino to be able to talk to each sensor individually over the I2C bus. The embedded system architecture for the sensor 2 node just described can be observed below in Figure 5.17.

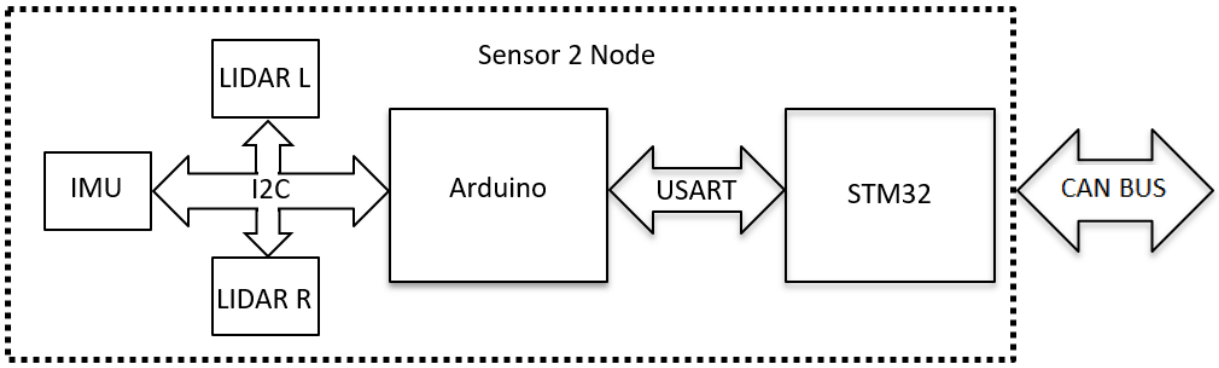


Figure 5.17 - Embedded hardware architecture of the sensor 2 node

The Arduino is responsible for processing the sensor data from the IMU and LIDAR sensors, while the STM32 is in charge of transmitting and receiving CAN messages to and from the controller node. The controller node will transmit a CAN FSM state message after every state transition to update the two

sensor nodes on the Inter-row's new active operating state. This FSM message will be received and processed by the STM32 microcontroller of the sensor 2 node. The STM32 will then notify the Arduino of the new operating state over the USART. The Arduino will then respond by reading from the appropriate sensor to aid in the autonomous navigation procedure. When the task has been completed, the Arduino will serially notify the STM32, thereby triggering the transmission of the appropriate CAN message out to the controller node over the CAN bus. This will all become clearer in the next section to follow, where the implemented FSM along with all of the associated system modes will be explained and examined in detail.

5.2.5 Implemented Embedded System Design

Now that the reader is briefed on the row detection and headland turning scheme and the added sensor 2 node implementation, it is now time to address how this additional work was integrated into the overall embedded system design architecture of the Inter-row robot.

5.2.5.1 *CAN Signal Flow among the DRTS*

In order to implement the row detection and headland turning schemes onto the Inter-row robot, an FSM needed to be developed and designed to control the operating modes of the Inter-row robot. As was noted earlier, the Inter-row employed an event driven embedded system architecture. Therefore, an event triggered FSM will be developed. The developed FSM will be implemented onto the controller node. Thus, from a top level perspective, the controller node would manage the operating mode in the FSM of the robot. The controller node will be configured to receive and process the data and signals transmitted by the sensor 1 and sensor 2 nodes so that it can respond accordingly by sending the appropriate motor command out to the motor driver node to put or keep the robot in the appropriate state within the FSM. The controller node would then send out an FSM CAN message every time it performed a state transition, to inform the other nodes of the new operating mode the robot is now in. Therefore, the two sensor nodes on the receiving end would be able to properly respond and execute the appropriate tasks based on the

robot's new operating mode. Figure 5.18 depicts a visual illustration of this top level design where the CAN signal flow among all of the system nodes within the DRTS is depicted.

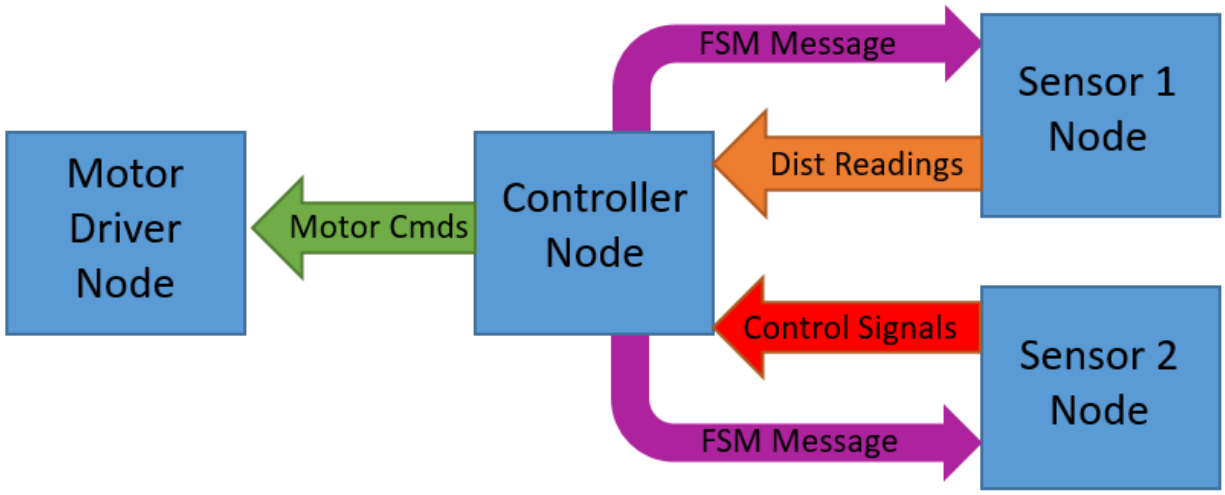


Figure 5.18 - CAN signal flow among all the system nodes in the DRTS architecture

As the Inter-row is a CAN based DRTS, all the messages being transmitted back and forth across the CAN bus among all the system nodes (as depicted above) are all unique CAN messages, where each message was specified and described in section 5.1.5.1. As the figure above suggests, an important distinction to make between the two sensor nodes is the type of messages that are being transmitted by each one. The sensor 1 node transmits the laser distance readings over the CAN bus for the controller node to process. The sensor 2, however, does not actually transmit any sensor measurement data, but instead, transmits control signals or event trigger messages that notify the controller to update its state within the FSM. Thus, all of the data acquisition for the LIDAR and IMU sensors happens strictly at the sensor 2 node end. Thus, the important design implication being driven at here is that the sensor 2 node is effectively responsible for detecting the events that will trigger a state transition. Additionally, the motor driver node does not receive or process the FSM message being transmitted by the controller node. This is because the motor driver node is already properly informed on the active operating state of the controller node via the data [4] byte contained in the CAN motor command message.

5.2.5.2 ***System Mode Descriptions***

The various states of the implemented FSM are given below with a brief description of each state.

- *Enter Row Mode:* The LIDAR sensor is continuously sampled and read to search for the start of the row in the field using the method described in section 5.2.1. Once the robot has entered the row, a control signal will be transmitted by the sensor 2 node to notify the controller node to transition to the *Follow Row Mode* state in the FSM.
- *Follow Row Mode*: The Inter-row will navigate through the crop row using the laser based method described in section 5.1.5.2. Meanwhile, the sensor 2 node will simultaneously be sampling the LIDAR sensor to search for the end of the row as was described in detail in section 5.2.1. When the end of the row is detected, the sensor 2 node will transmit a control signal to the controller node. The controller node will respond by transitioning the robot into the *Turn Left Mode* state or the *Turn Right Mode* state.
- *Turn Left Mode:* The Inter-row will complete a 90° left turn using the process outlined in section 5.2.2. When the turn is complete, the sensor 2 node will transmit an IMU control signal message to notify the controller node, to which the controller will then make the appropriate state transition. If the Inter-row was previously in the *Follow Row Mode* state, the robot will enter the *Move to Left Row* state. If the Inter-row was previously in the *Move to Left Row* state, the robot will enter the *Follow Row Mode* state.
- *Turn Right Mode:* The Inter-row will complete a 90° right turn using the process outlined in section 5.2.2. When the turn is complete, the sensor 2 node will transmit an IMU control signal message to notify the controller node, to which the controller will then make the appropriate state transition. If the Inter-row was previously in the *Follow Row Mode* state, the robot will enter the *Move to Right Row* state. If the Inter-row was previously in the *Move to Right Row* state, the robot will enter the *Follow Row Mode* state.

- *Move to Left Row:* The Inter-row will use the left LIDAR sensor when transitioning between rows using the procedure outlined in stages 2-4 of Figure 5.15. When the robot has successfully reached the next adjacent row over, the sensor 2 node will transmit the appropriate control message to the controller node to enable robot to then enter the *Turn Left Mode* state.
- *Move to Right Row:* The Inter-row will use the right LIDAR sensor when transitioning between rows using the procedure outlined in stages 2-4 of Figure 5.15. When the robot has successfully reached the next adjacent row over, the sensor 2 node will transmit the appropriate control message to the controller node to enable robot to then enter the *Turn Right Mode* state.
- *Stop Mode*: The robot will enter this mode when the Inter-row has gone through the last row in the field. The controller will transmit a control signal that informs that other nodes to halt their operation. The controller node will refrain from sending CAN motor commands to the motor driver node at this time as it has completed its assigned route and will therefore terminate its operation.

5.2.5.3 *Finite State Machine Implementation*

A visual illustration of the implemented event triggered FSM for the Inter-row is shown below in Figure 5.19 where the FSM is depicted in block diagram form. The developed state machine is an event driven FSM, where state transitions are dependent on certain conditions being met (i.e., the occurrence of a specific event detected by one of the utilized sensors). For example, in *Row Follow Mode*, a state transition will only occur in the "event" that the LIDAR sensor from the sensor 2 node detects the end of the crop row. Additionally, the FSM messages are utilized by the controller node so that the other nodes in the system are properly informed on the robot's current operating state. This transaction is cardinal for proper system operation as the robot's active operating state will directly dictate the associated tasks that both of the two separate sensor nodes are to execute. For example, in *Row Follow Mode*, the sensor 1 node will utilize the two laser sensors for crop row navigation while the sensor 2 node will leverage the LIDAR sensors for end of row detection. Meanwhile, the IMU will not be utilized as there is no use for

the unit in this particular operating state. However, in *Turn Left Mode*, the sensor 2 node will leverage the IMU to obtain the yaw angle of the robot, while the laser and LIDAR sensors will temporarily not be utilized as there is no need for sampling those sensors in this operating mode. Thus, the robot's mode of operation directly dictates the specific set of tasks that the two sensor nodes are to carry out, and the FSM message being transmitted out by the controller node to notify the two nodes of the robot's new operating state is crucial in order for those tasks to all be executed at the appropriate time.

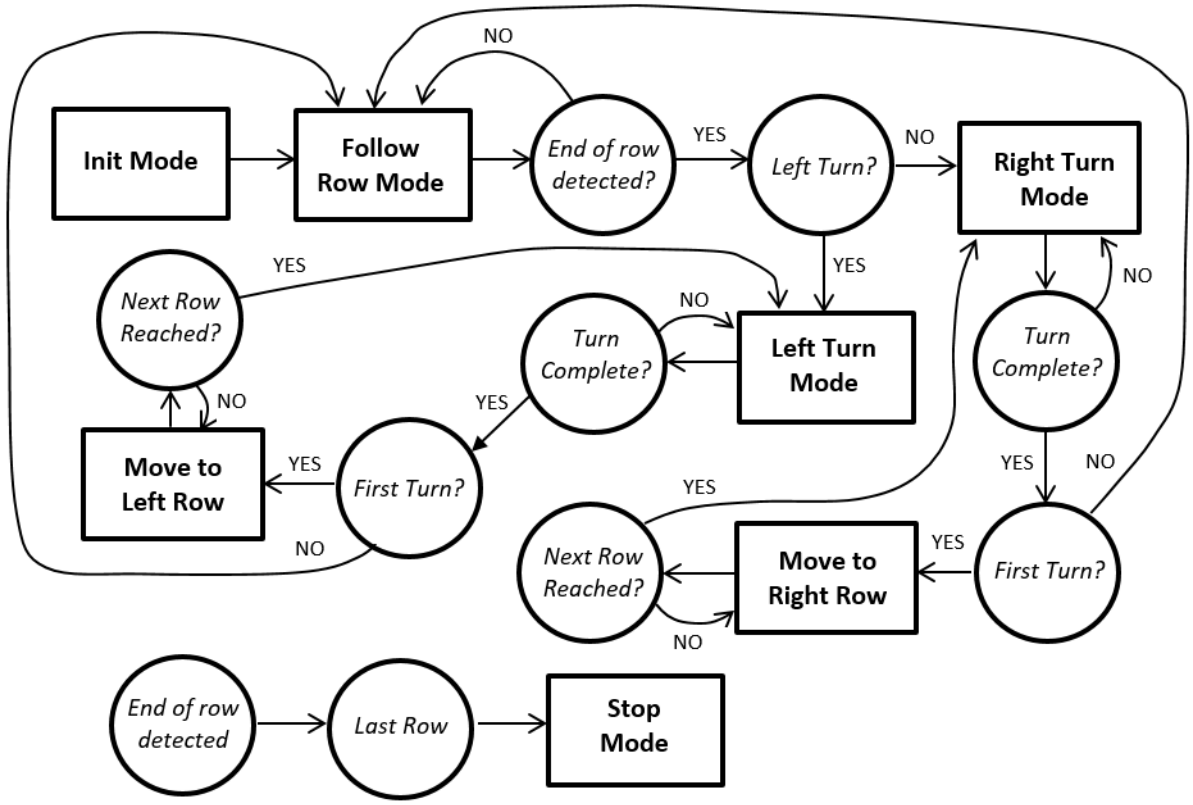


Figure 5.19 - Event-triggered Finite State Machine of Inter-row robot

5.2.5.4 ***Synchronization of Arduino with FSM***

Unlike the motor driver and sensor 1 node, whose STM32s are directly interfaced to their associated modules, the STM32 microcontroller on the sensor 2 node is indirectly linked to its associated sensors via the Arduino which performs all of the actual measurements on the sensors. Consequently, there are an additional number of steps that need to be executed by the sensor 2 node in response to a state transition

in order for both the STM32 and the Arduino to remain synchronized with the central FSM being managed by the controller node. Figure 5.20 depicts a flowchart detailing the corresponding sequence of events that occur for each time a state machine transition takes place at the controller node. The flowchart depicts all the mandatory tasks that are executed by each of the system nodes, which includes the additional tasks that are enacted at the sensor 2 node to ensure FSM synchronization with the Arduino.

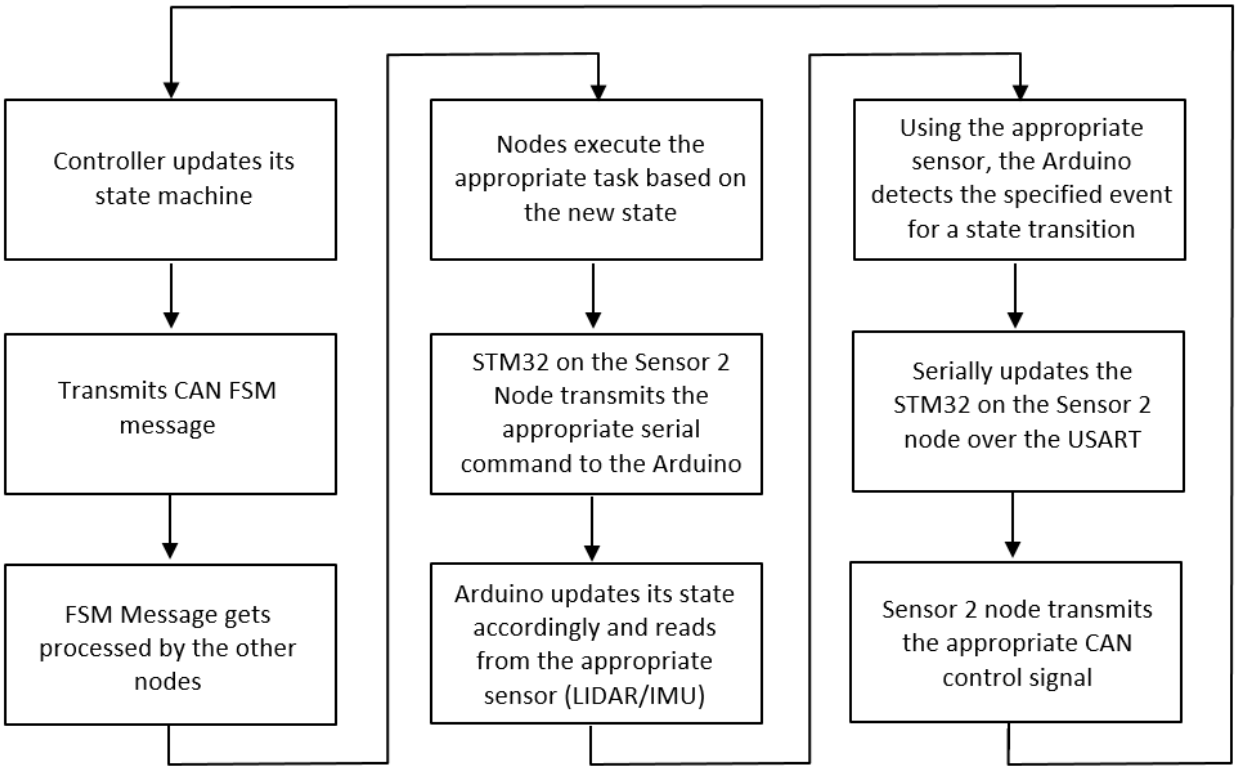


Figure 5.20 - Flowchart illustrating the sequences of events that gets triggered for every state transition that occurs inside the controller's FSM

Table 5.4 depicts the serial commands that govern the synchronization process between the STM32 and the Arduino at the sensor 2 node which ensures that both microcontrollers remain coincident with the current operating state of the FSM at the controller node. Thus, the Arduino is able to execute the appropriate task via the serial state machine updates it receives from the STM32 over the USART. The Arduino will then serially notify the STM32 when the task is complete. The STM32 will respond by

transmitting the appropriate control message over the CAN bus which will then alert the controller node to update its state in the FSM as the current task to perform has been fulfilled.

Serial Command	Signal Flow	State Transition	Application Task
'E'	*STM32 -> Arduino*	*Arduino enters **Enter Row** state*	*Utilizes LIDAR sensor for detecting the start of the crop row*
'F'	*STM32 -> Arduino*	*Arduino enters **Follow Row** state*	*Utilizes laser sensors for row following and LIDAR sensors for detecting the end of the crop row*
'R'	*STM32 -> Arduino*	*Arduino enters **Turn Right** state*	*Utilizes IMU sensor to perform a 90° right turn*
'L'	*STM32 -> Arduino*	*Arduino enters **Turn Left** state*	*Utilizes IMU sensor to perform a 90° left turn*
'H'	*STM32 -> Arduino*	*Arduino enters **Get Heading** state*	*Utilizes IMU to obtain reference heading angle*
'r'	*STM32 -> Arduino*	*Arduino enters **Move to Right Row** state*	*Utilize right LIDAR sensor to navigate to the next right row*
'l'	*STM32 -> Arduino*	*Arduino enters **Move to Left Row** state*	*Utilize left LIDAR sensor to navigate to the next left row*
'C'	*Arduino -> STM32*	*Arduino enters **Idle Mode***	*Task has been completed. Arduino will now wait for a new serial command from the STM32.*

Table 5.4 - Serial synchronization of Arduino with the controller's FSM

5.2.5.5 ***Peripheral Firmware Integration***

Figure 5.21 visually illustrates the top-level firmware architecture of the Inter-row Follower that formed the framework for the developed FSM. The figure illustrates the embedded firmware infrastructure from the controller node's perspective, where all of the outside sensors, modules, microcontrollers, and peripherals (coming from the additional nodes) are shown integrated into the entire system. Additionally, all the peripherals and modules depicted in the figure that are not linked to the Arduino, belong to one of the STM32s from one of the four respective nodes, which are not shown explicitly.

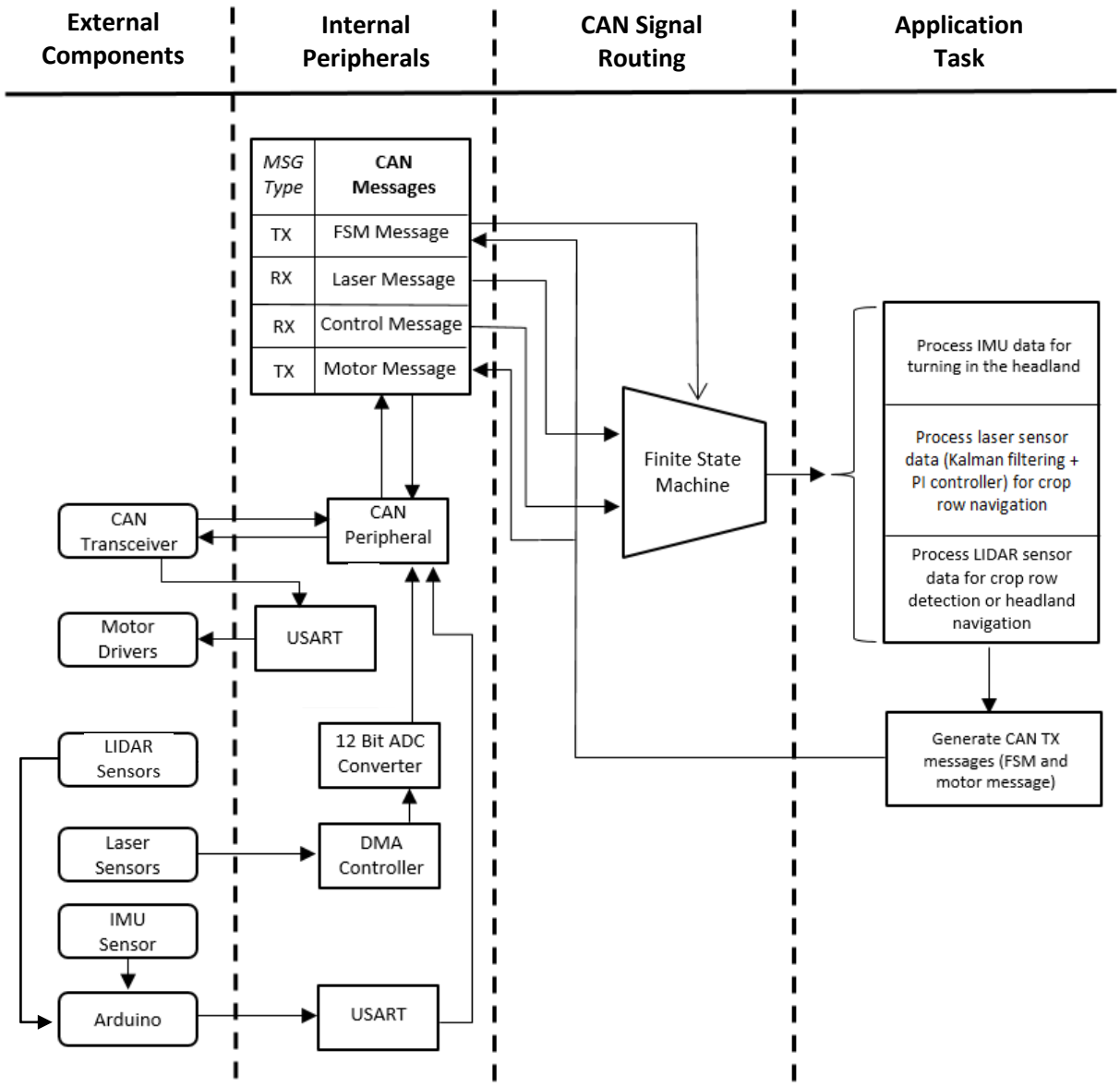


Figure 5.21 - Firmware architecture of Inter-row Follower

As implicated in the figure, the state machine CAN control message can be represented as a select line to a multiplexer which controls the particular CAN input messages (coming from the sensor 1 and 2 nodes) that will be processed and used by the controller node. The state machine control signal, transmitted by the controller node, updates all the other system nodes on the robot's current operating state which in turn, governs the applications tasks to be executed by each node in the system and the essential CAN input messages that need to be monitored and processed. Thus in essence, the figurative multiplexer at the

controller node essentially determines via the FSM message (i.e., the select line) which application tasks need to be executed which then results in the appropriate CAN input messages getting multiplexed from their respective nodes to the controller node over the CAN bus.

5.3 Results & Discussion

Verification tests that demonstrate and substantiate the working sensor based algorithm for the Row Follower were performed and will be presented and discussed in this section.

Data was collected and logged for the three sensors that were utilized to achieve autonomous navigation. Since the LIDAR and IMU sensors were interfaced with the Arduino (and not a STM32), both of the channels on the Kvaser (that was shown in Figure 2.3) were deployed in which one channel was used to log the laser sensor data (from the STM32) and the other was used to simultaneously collect the LIDAR and IMU data (from the Arduino). Since the data was being logged as CAN messages via the Kvaser, a CAN shield needed to be interfaced with the Arduino in order to feasibly log and collect the LIDAR and IMU CAN data. The shield was able to convert the IMU/LIDAR data into CAN messages that can then be processed and logged by the Kvaser for system data collection and analysis. The selected CAN shield that directly mounts onto the Arduino, is depicted below in Figure 5.22.

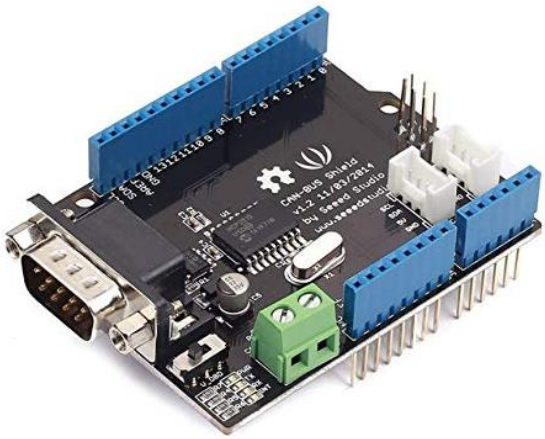


Figure 5.22 - CAN shield for logging LIDAR and IMU data as CAN messages via the Kvaser (Retrieved from https://www.amazon.com/CAN-BUS-Shield-Compatible-Arduino-Seeeduino/dp/B00NQVH666)

The three graphs shown below (Figures 5.23-5.25) exhibit the logged data and visually demonstrate the working algorithm for a simulated test, where the data collected for each of the three sensors are each shown simultaneously as a function of time. The logged data was sliced into three separate time segments. Thus, the three graphs displayed below, each depict a fifteen second time segment of operation, thus encapsulating an entirety of forty five seconds worth of time, which proved to be a sufficient amount to effectively demonstrate all the functioning operating states of the developed sensor based algorithm. Additionally, the operating states within the FSM are depicted and labelled appropriately throughout each plot as the operating mode of the robot can be inferred from the collected sensor data.

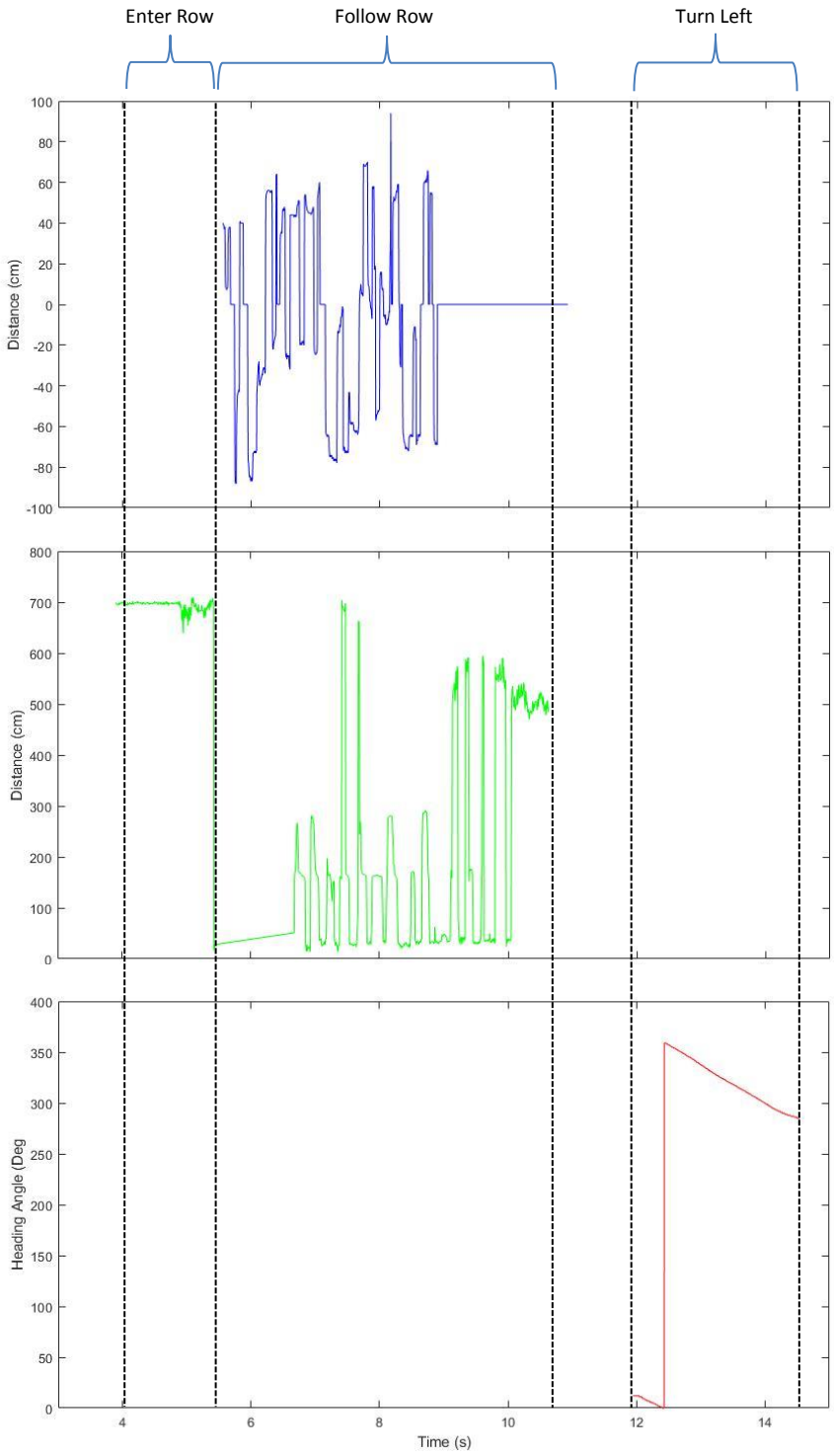


Figure 5.23 – Simulated test where the logged laser data (top), LIDAR data (middle), and IMU data (bottom) demonstrate the working non-GPS sensor based algorithm for a time segment of 0-15 sec.

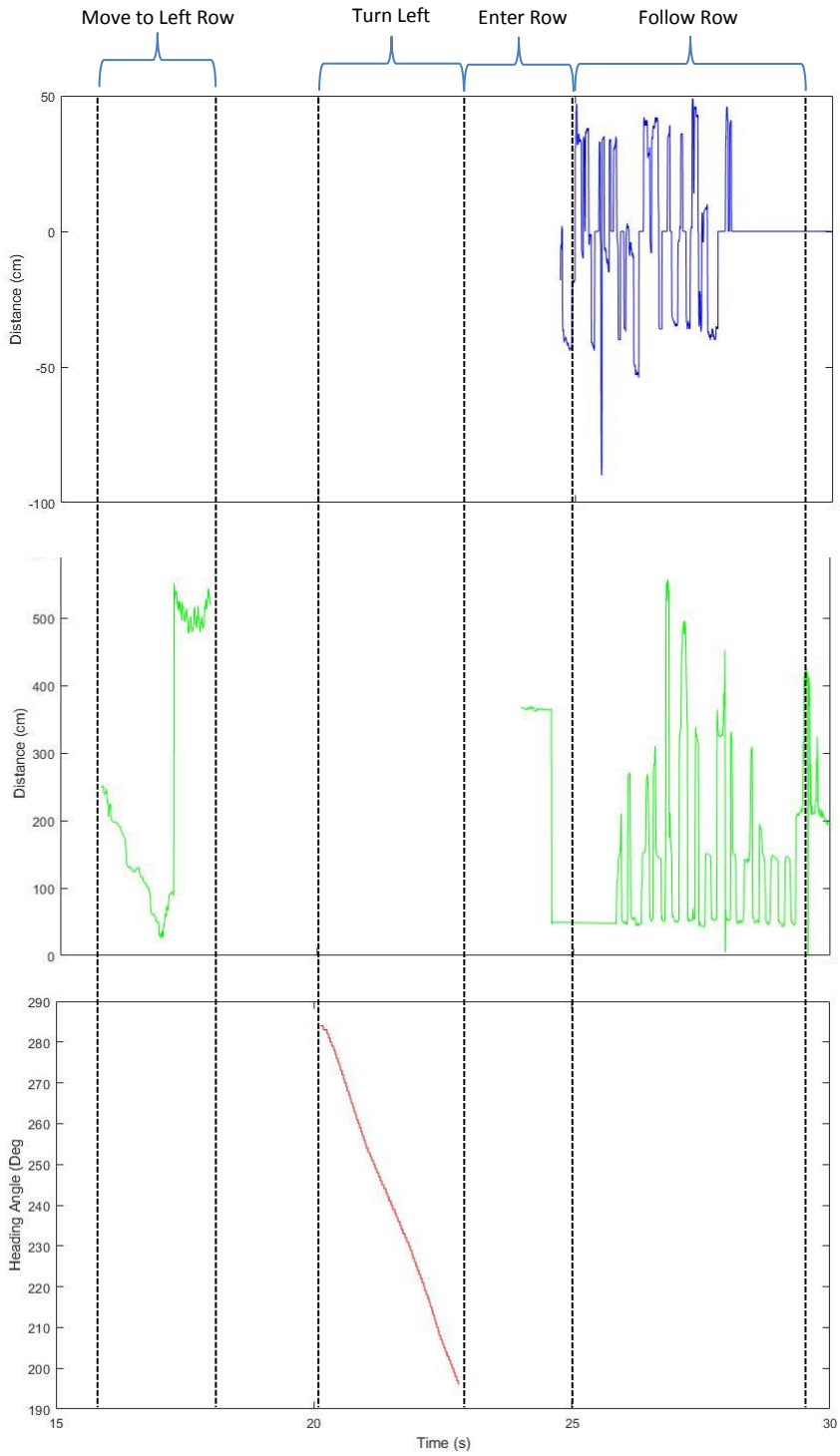


Figure 5.24 – Simulated test where the logged laser data (top), LIDAR data (middle), and IMU data (bottom) demonstrate the working non-GPS sensor based algorithm for a time segment of 15-30 sec.

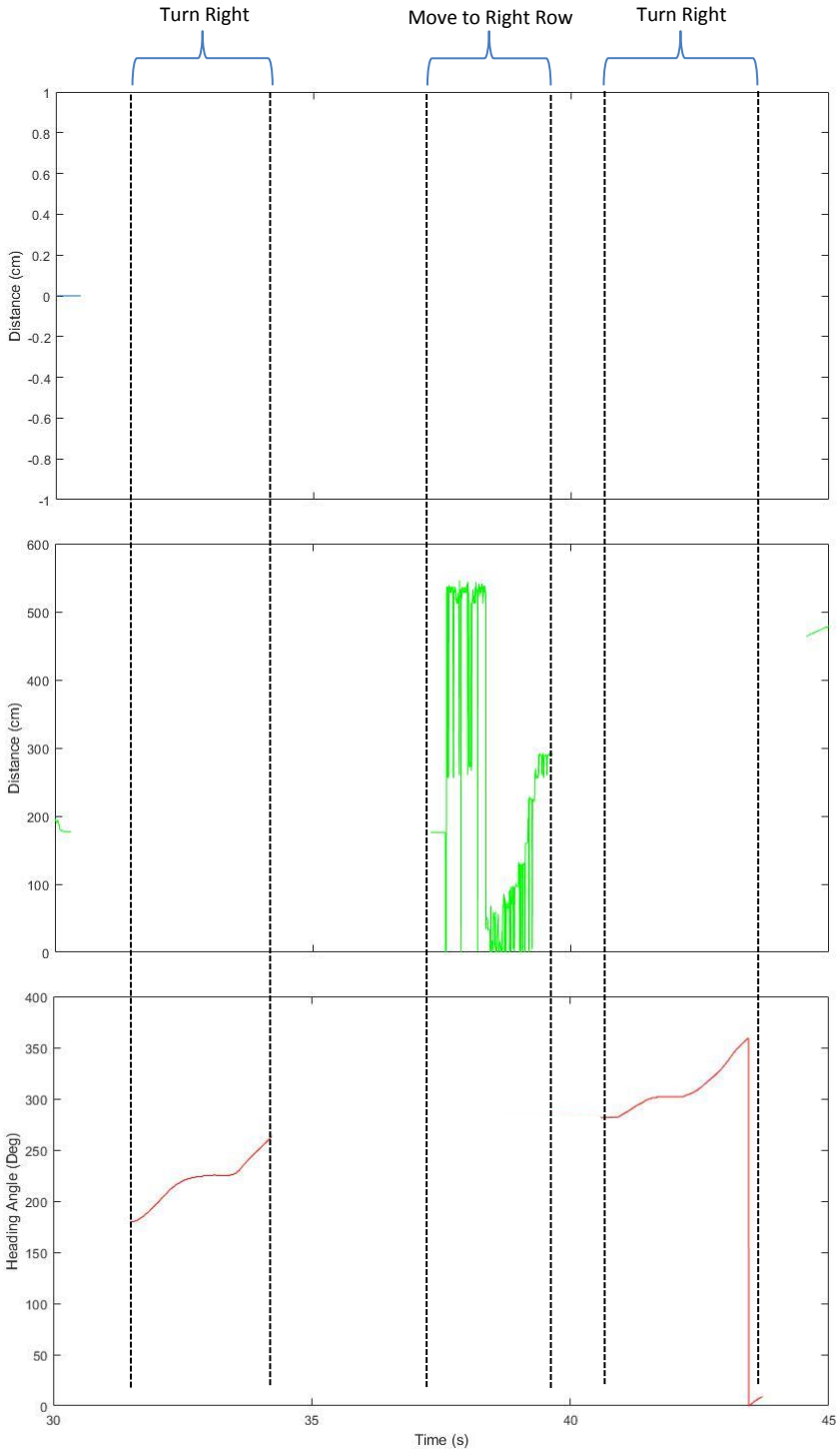


Figure 5.25 – Simulated test where the logged laser data (top), LIDAR data (middle), and IMU data (bottom) demonstrate the working non-GPS sensor based algorithm for a time segment of 30-45 sec.

It is important to note that the logged laser data was the calculated delta value between the individual distance measurements obtained from the left and right laser sensor (i.e., the difference between the data 1 and data 2 byte from the ID 200 CAN message – as was portrayed in Figure 5.8). Thus, the depicted laser sensor data observed in the three plots effectively indirectly illustrate how accurately or successfully the Inter-row is maintaining a centered position within the row of the field. Additionally, the reader might have also noticed the moments of time in the graphs where the laser difference measurements appeared to be zero, seeming to indicate that the row follower was perfectly centered within the row. This phenomenon could be specifically observed in the 9-11 second time segment of Figure 5.23 and the 28-30 second time segment of Figure 5.24. This was actually the inherent result of the row follower nearing the end of the row where a threshold of a 150 cm on the distance sensor measurements was applied in the firmware. When the row follower approaches the end of a row, the fired laser pulses will begin to extend beyond the crops. The distance measurements would thus saturate at 150 cm as the laser distance sensors will no longer be detecting the crop rows, hence resulting in the observed differential measurement of zero between the two laser sensors. This is consistent with what the graphs illustrate above as this event would occur right before the headland turning portion of the developed navigational algorithm would be observed.

There are also a couple of other features in the plots that are worth drawing attention to as some of the other plot characteristics in the figures may not be explicitly clear or explained via the provided state machine descriptions. Detecting the end of crop rows would require approximately about a half second's worth of time for the sampled LIDAR data to fall outside of the pre-determined threshold defined in the firmware. This can be observed at around the 10 second mark of Figure 5.23 and around the 39 second mark of Figure 5.25. The variability in the LIDAR measurements stem from the high frequency at which the LIDAR sensors are being sampled. Thus, the high sampling rate results in the LIDAR sensors detecting the stalks, the leaves, and the gaps in between the crops. Furthermore, for *Moving to the Left/Right Row* states, the next subsequent crop row is first detected via the LIDAR sensor when readings

fall below the pre-determined threshold (17 second mark of Figure 5.24 and 38 second mark of Figure 5.25) which will be followed by a consecutive number of LIDAR readings that will then inevitably fall outside the threshold, implying that the Inter-row has successfully reached the next adjoining row. Finally, the IMU data corresponds to a 0-360° value where values less than 0 will wrap back up to 360 and values greater than 360 will wrap back around to 0, as can observed in Figure 5.23 and Figure 5.25. The designed and developed state machine allowed the DRTS to intelligently and strategically utilize the pertinent sensors at the appropriate point in time based on the Inter-row's current location in the field. Lastly, all of the testing, development, and refining of the algorithm was performed by simulating a corn field in the machine shop via utilizing some prototype field rows illustrated below in Figure 5.26.

Figure 5.26 - Prototype corn rows used for the testing and development of the sensor based algorithm.

Test runs for the headland navigation algorithm were also conducted in an actual corn field to evaluate how the Inter-Row would perform in an actual field environment. The two figures that follow depict the logged data from all three of the sensors which are shown together for the same time segment. The graphs illustrate the working algorithm where the Inter-row was able to successfully navigate over to the next left row (Figure 5.27) and the next right row (Figure 5.28). Figure 5.29 shows the Inter-row operating in the corn field.

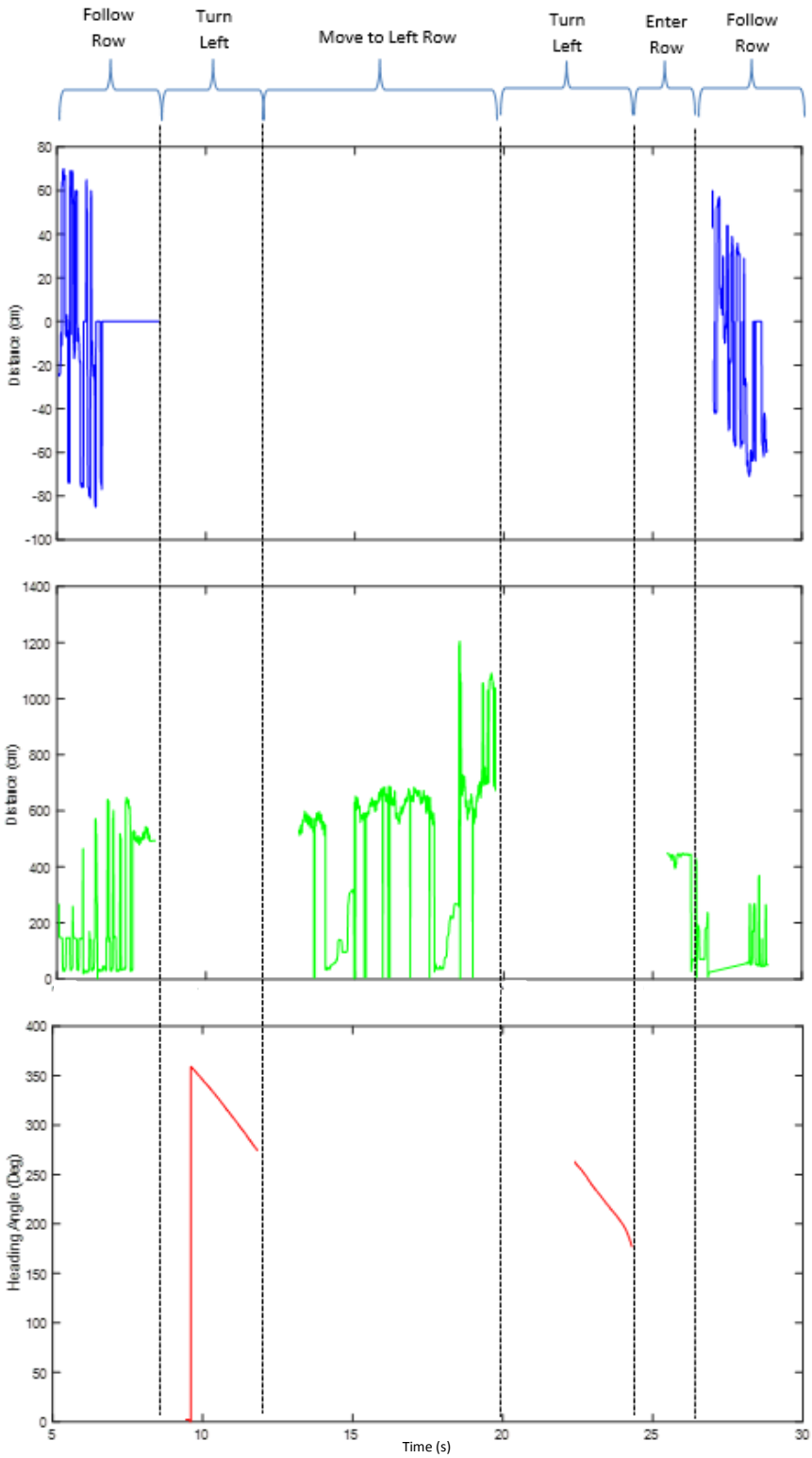


Figure 5.27 - Field test where the laser data (top), LIDAR data (middle), and IMU data (bottom) demonstrate the working headland navigation algorithm where the Inter-row moves to the next left row

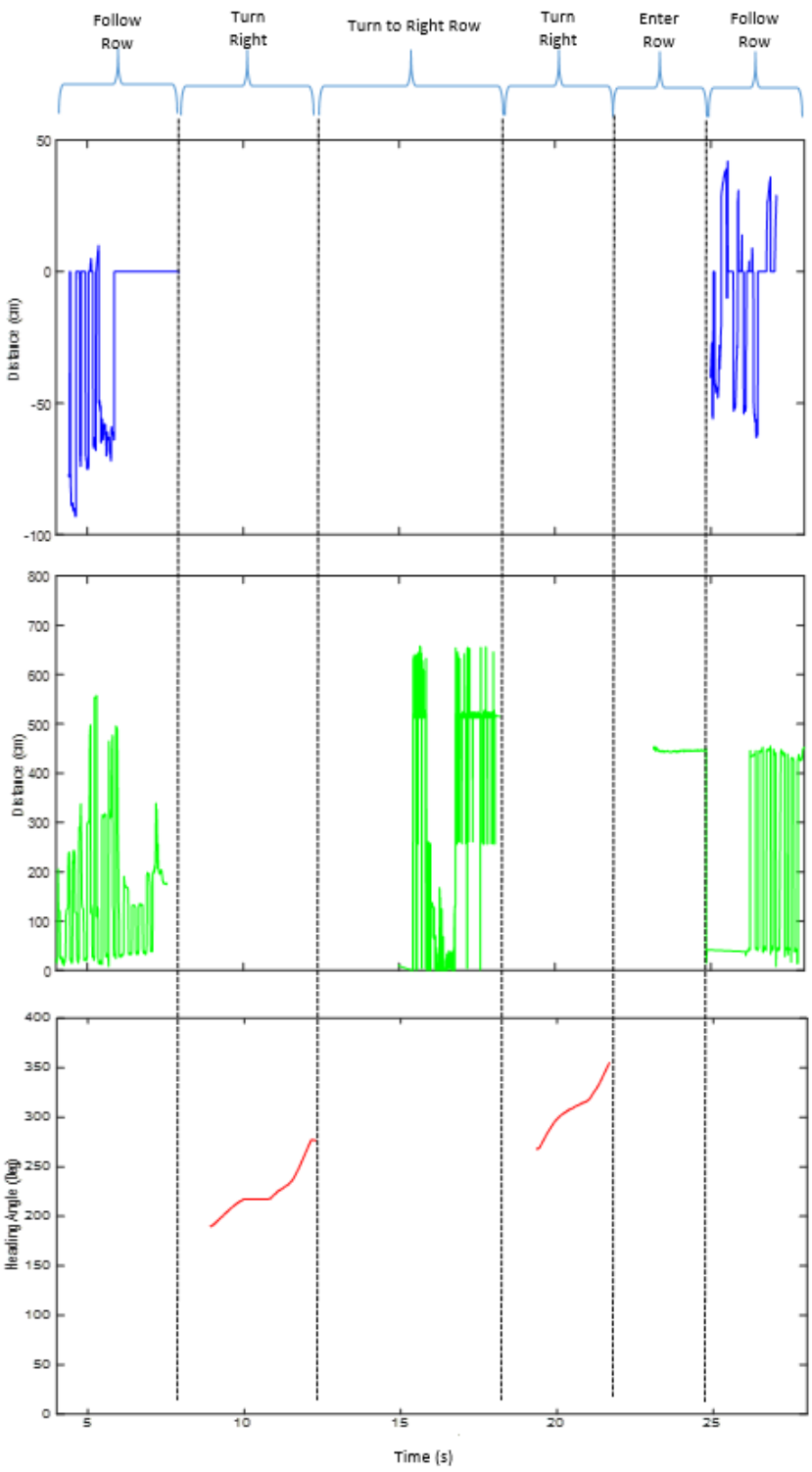


Figure 5.28 - Field test where the laser data (top), LIDAR data (middle), and IMU data (bottom) demonstrate the working headland navigation algorithm where the Inter-row moves to the next right row

Figure 5.29 - Inter-row operating in the field

Additionally, a second chassis was designed to house the additional electronics and modules required for the end goal of utilizing the Inter-row and the developed autonomous navigation algorithm in the field for the application of microclimate data collection. An Arduino was used to log the sensor data and a SD card module was used to save the logged data. A PVC pipe was selected to allow for the placement of the deployed sensors to be adjusted accordingly. The microclimate sensor utilized for the preliminary field test was an EE181 relative humidity sensor probe which came with a solar radiation shield. A depiction of the completed second chassis which housed this sensor (positioned 50 cm above the ground) and would be pulled by the Inter-row is provided below in Figure 5.30. Additionally, Figure 5.31 displays the final assembly where the Inter-row is shown operating in the field with the second chassis attached at the rear of the robot via a hitch. Finally, Figure 5.32 depicts the collected relative humidity data from the conducted autonomous navigation field test where Inter-row autonomously navigated through two rows in the field using the experimental setup shown in Figure 5.31.

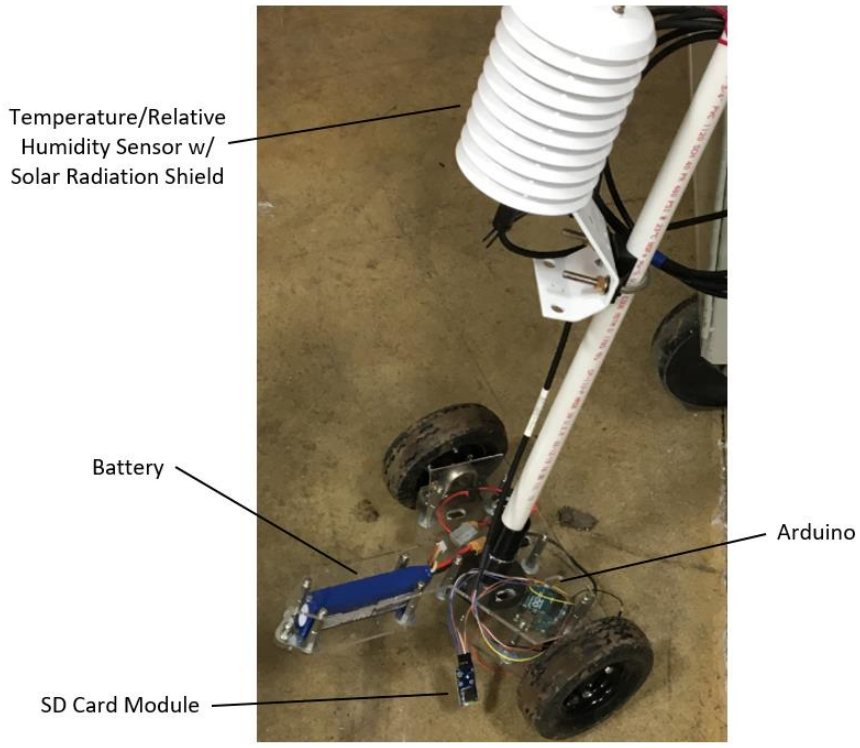


Figure 5.30 – Completed and fully assembled second chassis with all of the required electronics and components for logging the microclimate sensing data

Figure 5.31 - Completed assembly where the second chassis is connected to the rear of the Inter-row for microclimate data collecting in fields

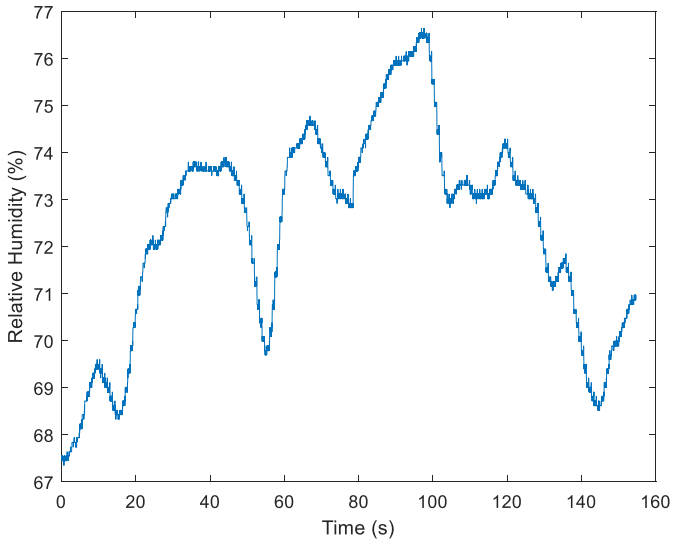


Figure 5.32 - Relative humidity data collected from autonomous navigation test in the field

The application for this conducted experiment was motivated by the plant high throughput field phenotyping research where high resolution measurements on plant traits (such as height or biomass) and environmental variables (such as solar radiation and wind speed) are needed [64]. This form of crop data is traditionally collected by sensors deployed on field vehicles or UAVs that are driven by human operators, where the data is usually collected from above the crop canopy [65]-[66]. The proposed autonomous navigation strategy executed and assessed in this experiment is therefore novel in two aspects. First, the Inter-row robot realizes full autonomy for field data collection. Second, the relative humidity data was collected from under the maize canopy, which has not been addressed by other plant phenotyping platforms.

During the outdoor field tests, the most prominent point of failure would occur at the end of the rows where the robot exhibited unpredictable behavior in regards to the row detection success rate. Further scrutiny revealed that this erratic behavior stemmed from inaccurate readings in the LIDAR sensor measurements – a phenomenon that only manifested in sunny, outdoor environments. It was empirically discovered that exposure to direct sunlight hindered the sensor's reliability and overall performance as the likelihood of receiving occasional false measurements from the sensor increased substantially in bright,

sunlit environments. Since the row detection scheme depended on receiving a consecutive number of samples for proper row distinguishing, just a single inaccurate reading would therefore be enough to skew the row detection mechanism, thereby crippling the Inter-row's ability to reliably identify the end of field rows. This issue was resolved by implementing a running average filter on the LIDAR sensor measurements which effectively minimized the impact of the faulty outlier readings. Additionally, extremely small erroneous measurements (less than 5 cm) that were also commonly triggered by direct exposure to sunlight were filtered out as well.

Since successful navigation over to the next adjacent row was predicated on the LIDAR sensor's ability to reliably detect the next subsequent row in the field, the Inter-row would occasionally miss the next row during the headland navigation process. Adding supplementary encoders or odometers onto the machine to further aid in guiding the robot over to the next row could nicely compliment the LIDAR sensors as these sensors could take additional distance measurements during the row repositioning procedure and could potentially improve the performance and reliability of the unmanned headland navigation algorithm. However, this would also come at the expense of having to increase the cost of the machine to account for the additional hardware required. The design strategy presented in this thesis provided a minimalist, cost sensitive approach, demonstrating that full autonomy in a field could feasibly be achieved via the utilization of just three different kinds of sensors.

Another formal observation made from the conducted experiments was the Inter-row's susceptibility to getting stuck during the test runs due to the turbulent, unpredictable terrain that characterized the corn field. This was particularly apparent when the Inter-row was attempting to perform a left or right turn in the headland as the robot would often require user assistance in order to complete the turn. Moreover, this was also especially evident when Inter-row was pulling the second chassis for microclimate data collection. This particular shortcoming mostly originates from the mechanical limitations of the chassis in use as transitioning over to the larger platform with larger wheels and more powerful motors would mostly likely alleviate this issue. Though the design does leave some room for further improvement and

optimization, the work provided in this chapter offers a novel, sensor-based fully autonomous algorithm for field row navigation, which could serve as a viable alternative method to the deployment of UAVs for microclimate field data collection.

6 Conclusion & Outlook

A complete summary of the accomplished work as well as its implications on the agriculture industry will be reviewed in this chapter. Additionally, ideas for future work and further development will also be discussed.

6.1 Summary

This thesis was an exploration into the further development of the embedded system design architecture of UAGVs and served as a tangible snapshot into the slow moving yet indisputable scalable infrastructural shift in this sector that is being observed today, where the industry is beginning to undergo an arduous period of research and development in effort to provide innovative unmanned automation methods than can practically be implemented onto field machinery. This evolution of agricultural machinery is a necessary alteration to effectuate in order to engineer and streamline improved food production operation techniques and practices for the rapidly increasing world population. The technological advancement of these machines will be instrumental in allowing farmers to be able to meet these impending demands. Hence, continuing to upgrade the modular machinery design of agricultural equipment by adopting the modern, cutting edge technology available today, is an essential endeavor that must be diligently sought after in order to attain the end goal of being able to dispatch agricultural automation machines in the field to carry out and accomplish agricultural related tasks. Moreover, the embedded system design architectures and modern control techniques that these autonomous machines employ, must be rigorously examined, extensively tested and evaluated, and continually refined to ensure vehicle safety and system design reliability and robustness. The continued technological progression in this sector has obvious and pronounced implications on the rest of the world, as the ingenuity and further refinement of this equipment will be pivotal to being able to provide farmers with competent unmanned agricultural machinery that can conceivably keep pace with their rising quotas.

The work accomplished in this thesis offered new algorithms that integrate state of the art technology onto field machinery prototype test platforms to demonstrate that the end goal of complete unmanned autonomy for agricultural automation can plausibly be attained. The developed algorithms were implemented onto both event-triggered (Inter-row) and time triggered (UAGV) CAN bus control architectures. The time triggered system design technique in particular, offers many intriguing advantages for autonomous machine design due to its deterministic nature and the resulting intrinsic safety implications it lends itself to, where the benefits to employing such a system remain mostly untapped by the agriculture sector. The firmware descriptions presented in this thesis for both autonomous machines, serve as a developmental starting point to establishing a new, modern, robust, industry standard system design infrastructure for agricultural machinery that is tailored to attain the recently imminent and pressing demand for complete and reliable autonomous machine navigation.

6.2 Future Work

The next step of further development for the first UAGV discussed in this thesis involves incorporating a more reliable GPS module onto the machine. The current RTK GPS in use suffers from somewhat unreliable and unpredictable behavior. During the conducted test runs for GPS navigation and obstacle avoidance appraisal, the module in operation proved to be notorious for arbitrarily losing its GPS reception where the diagnosis for this spurious and recurrent behavior was ambiguous. Perplexingly, there were no overt causes or discernible grounds to explain the reason for this defect since, during these observed failures, the vehicle was being operated at the Nebraska Tractor Test Track, where it was isolated from tall buildings, power lines, and other possible sources of signal interference. Additionally, it was discovered that the GPS module's performance was also greatly hindered when it was being operated within the vicinity of power lines which significantly limits the places where the UAGV can effectively be used. As a possible remedy to these concerns, a new promising, more robust RTK GPS module was purchased earlier this year (Duro Inertial RTK); however it has yet to be thoroughly tested. The next step would be to interchange the old GPS module with this new unit and to evaluate its performance.

Additionally, as was noted in Chapter 3, the curvature control algorithm needed to be uniquely designed to accommodate for the slow or lagging processing time of the controller node. Having the curvature algorithm utilize fixed point math in comparison to floating point operations would be a good place to start for further improving this algorithm. Thus, other areas worth exploring would be looking into a means to make the curvature turning algorithm more computationally efficient and to investigate the feasibility and possible advantages of potentially breaking the controller node into two separate nodes to reduce the processing demands on the microcontroller. The later suggestion should only be pursued if the designer is confident that his proposed solution will be an improvement upon the current design (from a program efficiency standpoint), as it will likely entail a redesign of the controller node which will thus likely require a considerable amount of development time.

For the inter-row robot, there are several changes or modifications that are strongly encouraged to be made in order for this ground vehicle to be realized as a robust, autonomous navigation solution in actual outdoor field applications. Firstly, it is recommended that the chassis platform gets upgraded to a wider, heavier contraption with larger wheels which would allow the robot to be more capable of handling the rougher terrain. This would reduce the likelihood of the robot getting stuck during outdoor field tests, especially during autonomous navigation operations when the second trailer is also being pulled for microclimate data collection and analysis.

Also, adding a second motor driver onto the chassis is a critical design change that needs to be made. The row follower would struggle to perform skid turns in the headland during outdoor field tests due to the rough terrain which presented a higher coefficient of friction compared to the smooth flat floor the robot operated on in the simulated tests. Two dual channel motor drivers would allow for a left motor and a right motor to be individually driven by one of the modules, while the four remaining motors would be driven by the second dual channel driver board. Thus in this design arrangement, only two motors on both the left and the right side of the chassis would then be connected in parallel. This would be a significant improvement from the present-day configuration where all three motors on both the left and right side of

the chassis are currently connected in parallel (i.e., the first channel is connected to the three motors on left and the second channel is interfaced to the three motors on the right). The inherent design drawback in this currently deployed topology stems from the fact that the parallel connection will cause the motors that encounter the most amount of friction from the wheels being in firm contact with the ground, to receive the least amount of current from the driver. This is because current will always take the path of least resistance. Hence, adding a second motor driver onto the system will significantly improve the robot's performance in outdoor field environments and help alleviate this issue.

Additionally, examining a new power distribution model would also be a worthwhile venture, as the robot's current runtime is capped at around thirty minutes of operation. The solution could be as simple as just connecting two or three of the currently used batteries in parallel. Finally, incorporating an additional node onto the robot via implementing a GPS module onto the machine would also be a fruitful endeavor. GPS capabilities would nicely complement the dead reckoning algorithm and would make for a more robust, versatile robot, essentially adding another protective layer of autonomous navigation as well as offering more flexibility to the system, enabling another means by which unmanned navigation can be achieved.

7 References

[1] Hertz, T. (January 2013). Is There A Farm Labor Shortage? In *American Journal of Agricultural Economics. 95*(2), 476-184. https://doi.org/10.1093/ajae/aas090

[2] Zhang, N., Wang, M., Wang, N. (November 2002). Precision Agriculture – A Worldwide Overview. In *Computers and Electronics in Agriculture. 36*(2-3), 113-132. https://doi.org/10.1016/S0168-1699(02)00096-0

[3] Gebbers, R. and Adamchuk, V. (February 2010). Precision Agriculture and Food Security. In *Science. 327*(5967), 828-831. doi: 10.1126/science.1183899

[4] Lee, I. (2010). Introduction to the Distributive Real-Time System [Powerpoint slides]. University of Pennsylvania. Retrieved from https://www.seas.upenn.edu/~lee/10cis541/lecs/kopetz_chap_1and2-ilx2.pdf

[5] Troyer, T. (Octoboer 2017). Event and Time Triggered Control Module Layers for Individual Robot Control Architecture of Unmanned Agriculutural Ground Vehicles (Master's Thesis). University of Nebraska-Lincoln, Lincoln, NE. Retrieved from https://digitalcommons.unl.edu/biosysengdiss/73/

[6] John, D. (2013). Distributive Real-Time System [Powerpoint slides]. Saint Joseph's College of Engineering and Technology. Retrieved from https://www.slideshare.net/iamdeepakjohn/real-time-system-37486039

[7] Tindell, K. W., Hansson, H. and Wellings, A. J. (1994). *Analyzing Real-Time Communications: Controller Area Network (CAN*). Paper presented at 1994 IEEE Proceedings Real-Time Systems Symposium, San Juan, Puerto Rico, USA. doi: 10.1109/REAL.1994.342710

[8] Corrigan, S. (2016). Introduction to Controller Area Network (CAN). Texas Instruments Incorporated. Retrieved from ti.com/lit/an/sloa101b/sloa101b.pdf

[9] Scheler, F. and Schroeder-Preikschat, W. (2006). Time-Triggered vs Event-Triggered: A Matter of Configuration? In *MMB Workshop Proceedings (GI/ITG Workshop on Non-Functional Properties*

of Embedded Systems Nuremberg. Retrieved from https://pdfs.semanticscholar.org/bbdc/27b0f57fd55b0f9c2d7b5b84fdf3cf19facf.pdf

[10] Albert, A., Gmbh, R. B. (2003). Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. In *Embedded World*, Vol 2004. pp 235-252. Retrieved from https://pdfs.semanticscholar.org/bbdc/27b0f57fd55b0f9c2d7b5b84fdf3cf19facf.pdf

[11] N. A. (2008) An Introduction to GPS Technology: GPS Basics [Powerpoint slides]. University of New Hampshire Cooperative Extension Program. Retrieved from http://webcache.googleusercontent.com/search?q=cache:YvpCk8e7XK0J:www.edc.uri.edu/personal/erica/GPS_Basics/GPS%2520Basics.pptx+&cd=1&hl=en&ct=clnk&gl=us

[12] Mintsis, G., Basbas, S., Papaioannou, P., Taxiltaris, C., and Tziavos, I. N. (2004). Application of GPS technology in the land transportation system. In *European Journal of Operation Research. 152*(2). 399-409. https://doi.org/10.1016/S0377-2217(03)00032-8

[13] Wilson, J. N. (2000). Guidance of agricultural vehicles – a historical perspective. In *Computers and Electronics in Agriculture. 25*(1-2). 3-9. doi: 10.1016/S0168-1699(99)00052-6

[14] N. A. (2014). GPS Correction Comparisons – RTK vs DGPS. Kairos Autonomi. Retrieved from http://kairosautonomi.com/uploads/files/129/Bulletin---RTK-vs-DGPS-010400.pdf

[15] Reid, J. F. Zhang, Q., Noguchi, N., Dickson, M. (2000). Agriculture automatic guidance research in North America. In *Computers and Electronics in Agriculture. 25*(1-2). 155-167. doi: 10.1016/S0168-1699(99)00061-7

[16] Shala, N., Low, T., McCarthy, C., and HanCock, N. (2013). A Review of Autonomous Navigation Systems in Agricultural Environments. In 2013 Society for Engineering in Agriculture Conference: Innovative Agricultural Technologies for a Sustainable Future. Barton, Western Australia. pp 22-25. Retrieved from https://search.informit.com.au/documentSummary;dn=874429263725282;res=IELENG

[17] Corrigan, D. (2011). An Introduction to Control Systems [pdf]. University of Southern Illinois. Retrieved from mee.tcd.ie/~sigmedia/pmwiki/uploads/Teaching.3C1/control_systems.pdf

[18] Lundgren, M. (2003). Path Tracking for a Miniature Robot. Excerpt from Master's Thesis. Umea University. Retrieved from http://www8.cs.umu.se/kurser/TDBD17/VT06/utdelat/Assignment%20Papers/Path%20Tracking%20for%20a%20Miniature%20Robot.pdf

[19] Coulter, C. R. (1992). Implementation of the Pure Pursuit Tracking Algorithm. Carnegie Mellon University. Retrieved from https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf

[20] Walter, Scott A. (1987). The Sonar Ring: Obstacle Detection for a Mobile Robot. Paper presented at 1987 International Conference on Robotics and Automation. Raleigh, NC, USA. doi: 10.1109/ROBOT.1087.1087902

[21] Elfes, A. (1987). Sonar-Based Real-World Mapping and Navigation. In *IEEE Journal of Robotics and Automation. RA-3*(2). 249-265. doi: 10.1109/JRA.1987.1087096

[22] Lablanc, L., Mayer, L., Rufino, M., Schock, S. G., and King, J. (1991). Marine Sediment Classification using the Chirp Sonar. In *The Journal of Acoustical Society of America. 91*(1). 107-115. doi: 10.1121/1.402758

[23] Petillot, Y., Ruiz, I. T., and Lane, D. M. (1999). Underwater Vehicle Obstacle Avoidance and Path Planning Using a Multi-Beam Forward Looking Sonar. In *IEEE Journal of Oceanic Engineering. 26*(2). 240-51. doi: 10.1109/48.922790

[24] Heidarsson, H. K. and Sukhatme, G. S. (May 2011). Obstacle Detection and Avoidance for an Autonomous Surface Vehicle (ASV) using a Profiling SONAR. Paper presented at 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, May 9-13, 2011. Shanghai International Conference Center. doi: 10.1109/ICRA.2011.5980509

[25] Coker, W. B. (1995). *United States Patent No. 5,410,479.* Retrieved from https://patents.google.com/patent/US5410479A/en

[26] Andujar, D, Weis, M., and Gerhards, R. (2012). An Ultrasonic System for Weed Detection in Cereal Crops. In *Sensors I2*. 17343-17357. doi: 10.3390/s121217343

[27] Dvorak, J. S., Stone, M. L., Self, K. P. (2016). Object Detection for Agricultural and Construction Environments Using an Ultrasonic Sensor. In *Biosystems and Agricultural Engineering Faculty Publications. 22*(2). 107-119. https://doi.org/10.13031/jash.22.11260

[28] Colaco, A. F., Molin, J. P., Rosell-Polo, J. R., Escola, A. (2018). Application of light detection and ranging and ultrasonic sensors to high-throughput phenotyping and precision horticulture: current status and challenges. In *Horticulture Research. 5*(35). doi: 10.1038/s41438-018-0043-0

[29] Gray, K. (2000). Obstacle Detection and Avoidance for an Autonomous Farm Tractor (Master's Thesis). Utah State University. Logan, UT. Retrieved from http://www8.cs.umu.se/research/ifor/dl/OBSTACLE%20DETECTION-AVOIDANCE/OBSTACLE%20DETECTION%20AND%20AVOIDANCE%20FOR%20AN%20AUTONOMOUS%20FARM%20TRACTOR.pdf

[30] Discant, A., Rogozan, A., Rusu, C., Bensrhair, A. (2007). Sensors for Obstacle Detection – A Survey. In *30th International Spring Seminar on Electronics Technology (ISSE).* pp 100-105. doi: 10.1109/ISSE.2007.4432828

[31] Fernandez, L. C., Dias, M. A., Osorio, F., Wolf, D. (November 2010). *A Driving Assistance System for Navigation in Urban Environments.* In *Advances in Artificial Intelligence – IBERAMIA 2010.* pp 542-551. doi:10.1007/978-3-642-16959-6_55

[32] Chen, L., Yang, J., and Kong H. (June 2017). Lidar histogram for fast road and obstacle detection. IEEE International Conference on Robotics and Automation (ICRA), Singapore, Singapore. May 29 – June 3, 2017. IEEE. doi: 10.1109./ICRA.2017.7989159

[33] Chong, Z. J., Qin, B., Bandyopadhyay, T., Ang Jr., M. H., Frazzoli, E., and Rus, D. (2013). Synthetic 2D-LIDAR for Precise Vehicle Localization in 3D Urban Environment. IEEE International Conference on Robotics and Automation (ICRA). Karlsruhe, Germany, May 6-10, 2013. IEEE. doi: 10.1109/ICRA.20136630777

[34] Peng, Y., Qu, D., Zhong, Y., Xie, S., and Luo, J. (2015). The Obstacle Avoidance Algorithm based on 2D-LIDAR. IEEE International Conference on Information and Automation. Lijiang, China. August 2015. IEEE. doi: 10.1109/IC/InfA.2015.7279550

[35] Catapang, A. N. and Ramos Jr., M. (2016). Obstacle Detection using a 2D LIDAR System for an Autonomous Vehicle. IEEE International Conference on Control Systems, Computing, and Engineering, Penang, Malaysia, Nov 25-27, 2016. IEEE. doi: 10.1109/ICCSCE.2016.7893614

[36] Fernandez, C., Dominguez, R., Fernandez-Llorca, D., Alonso, J., and Sotelo, M. A. (February 2013). Autonomous Navigation and Obstacle Avoidance on a Micro-bus. In *International Journal of Advanced Robotic Systems.* doi.10.5772.56175

[37] Weiss, U. and Biber, P. (2011). Plant detection and mapping for agricultural robots using a 3D LIDAR sensor. In *Robotics and Automation Systems. 59* (2011). 265-273. https://doi.org/10.1016/j.robot.2011.02.011

[38] Rosell, J. R., Llorens, J., Sanz, R., Arno, J., Ribes-Dasi, Manel. Masip, J., Escola, A., Camp, F., Sonalles, F, Gracia, F., Gil, E., Val, L, Planas, S., and Palacin, J. Obtaining the three dimensional structure of tree orchards from remote 2D terrestrial LIDAR scanning. In *Agricultural and Forest Meteorology. 149* (2009). 1505-1515. https://doi.org/10.1016/j.agrformet.2009.04.008

[39] Biber, P., Weiss, U., Dorna, M., and Albert, A. (2012). Navigation System of the Autonomous Agricultural Robot "Bonirob". In *Workshop on Agricultural Robotics: Enabling, Safe, Efficient, and Affordable Robots for Food Production (Callocated IROS).* Vilamoura, Portugal. Retrieved from cs.cmu.edu/~mbergerm/agrobotics2012/01Biber.pdf

[40] Emmi, L., Gonzales-de-Sota, Pajares, G., Gonzales-de-Santos, P. (2014). New Trends in Robotics for Agriculture: Integration and Assessment of a Reel Fleet of Robots. In *The Scientific World Journal. 1*(21). doi: 10.1155/2014/404059

[41] Malavazi, F. B. P., Guyonneau, R. Fasquel, J. B., Lagrange, S., and Mercier, F. (2018). LIDAR-based only navigation algorithm for an autonomous agricultural robot. In *Computers and Electronics in Agriculture.* Vol. 154. pp 71-79. https://doi.org/10.1016/j.compag.2018.08.034.

[42] Susnea, I., Minzu, V., and Vasiliu, G. (2009). Simple Real-Time Obstacle Avoidance Algorithm for Mobile Robots. In *Proceedings of the 8th WSEAS International Conference on Computational Intelligence, Man-Machine Systems, and Cybernetics.* pp 24-29. Retrieved from https://www.researchgate.net/publication/228955195_Simple_real-time_obstacle_avoidance_algorithm_for_mobile_robots

[43] Borenstein, J. and Koren, Y. (1991). The Vector Field Histogram – Fast Obstacle Avoidance for Mobile Robots. In *IEEE Journal of Robotics and Automation. 7*(3). 278-288. Retrieved from http://www-personal.umich.edu/~johannb/Papers/paper16.pdf

[44] Koren, Y. and Borenstein, J. (1991). Potential Fields Methods and Their Inherent Limitations for Mobile Robot Navigation. International Conference on Robotics and Automation, Sacramento, California, April, 1991. IEEE. doi: 10.1109/ROBOT.1991.131810

[45] Sezer, V. and Gokasan, M. (September 2012). A Novel Obstacle Avoidance Algorithm: "Follow the Gap Method". In *Robotics and Autonomous Systems* pp 1123-1144. doi: 10.1016/j.robot.2012.05.021

[46] Higuti, V. A. H., Velaquez, A. E. B., Magalhaes, D. v., Becker, M., and Chowdhary, G. (2017). Under canopy light detection and ranging – based autonomous navigation. In *Journal of Field Robotics.* doi: 10.1002/rob.21852

[47] Welch, G. and Bishop, G. (2006). An Introduction to the Kalman Filter. TR-95-041. Retrieved from https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf

[48] Kelly, A. (1994). A 3D State Space Formulation of a Navigation Kalman Filter for Autonomous Vehicles. CMU-RI-TR-9419. Retrieved from https://www.ri.cmu.edu/pub_files/pub1/kelly_alonzo_1994_6/kelly_alonzo_1994_6.pdf

[49] Southall, B., Hague, T., Marchant, J. A., and Buxton, B. F. (1999). Vision aided Outdoor Navigation of an Autonomous Horticultural Vehicle. pp. 37-50. doi: 10.1007/3-540-49256-9_3

[50] Southall, B., Hague, T., Marchant, J. A., and Buxton, B. F. (2002). An Autonomous Crop Treatment Robot: Part 1. A Kalman Filter Model for Localization and Crop/Weed Classification. In

International Journal of Robotics Research. 21 (1). 61-74. https://doi.org/10.1177/027836402320556485

[51] Hansen, S., Bayramoglu, E., Andersen, J. C., Raven, O., Andersen, N., and Poulsen, N. K. (2011). Orchard navigation using derivative free kalman filtering. American Control Conference, San Francesco, CA, 2011. doi: 10.1109/ACC.2011.5991403

[52] Xue, J. and Grift, T. E. (2011). Agricultural Robot Turning in the Headland of Corn Fields. In *Appliced Mechanics and Materials.* pp 63-64. doi: 10.4028/www.scientific.net/AMM.63-64.780

[53] Xue, J., Grift, T. E., and Zhang, L. (June 2012). Variable Field of View Machine Vision based Row Guidance of an Agricultural Robot. In *Computers and Electronics in Agriculture.* Vol. 84. 85-91. https://doi.org/10.1016/j.compag.2012.02.009

[54] Thamrin, N. M., Arshad, N. H. M., Adnan, R., Sam, R., Razak, N. A., Misnan, M. F., Mahmud, S. F. (2016). A Turning Scheme in the Headland of Agriculture Fields for Autonomous Robot. In *ARPN Journal of Engineering and Applied Science. 11*(5). 3265-3269. Retrieved from http://www.arpnjournals.org/jeas/research_papers/rp_2016/jeas_0316_3794.pdf

[55] Jakobsen, A. (2015). Crop Row Navigation for Autonomous Field Robot. (Master's Thesis). Technical University of Denmark. Kongens Lyngby, Denmark. Retrieved from http://aut.elektro.dtu.dk/mobotware/doc/robotti/Robotti_row_follow_Aske_Bay_Jakobsen.pdf

[56] Argarwal, N. and Thakur, R. (2016). Design of an Agricultural Robot to Move between the Rows. In *International Journal of Innovative Research Science, Engineering, and Technology. 5*(8). 15734-15739. doi: 10.2991/earne-15.2015.97

[57] Kok, M., Hol, J. D., Schon, T. B. (2017). Using Inertial Sensors for Position and Orientation Estimation. In *Foundations and Trends in Signal Processing. 11*(1-2). 1-153. doi: 10.1561/2000000094

[58] Neto, P., Mendes, N., and Moreria, A. P. (2015). Kalman filter-based yaw angle estimation by fusing inertial and magnetic sensing: a cast study using lost cost sensors. In *Sensor Review 35*(3). 44-50. doi: 10.1108/SR-2014-0723

[59] Zhao, S., Zhang, Z., Xiao, D., and Xiao, K. (2016). A Turning Model of Agricultural Robot on Acceleration Sensor. In *International Federation of Automatic Control. 49*(16). 45-50. https://doi.org/10.1016/j.ifacol.2016.10.081

[60] Wang, H. and Noguchi, N. (2016). Autonomous Maneuvers of a Robotic Tractor for Farming. Paper presented at 2016 IEEE/SICE International Symposium on System Integration. Sapporo, Japan, 2016. IEEE. doi: 10.1109/sii.2016.7844063

[61] Elisson, V. and Gassler, G. (2014). Low Cost Relative GNSS Positioning with IMU Integration (Master's Thesis). Chalmers University of Technology. Retrieved from http://publications.lib.chalmers.se/records/fulltext/200466/200466.pdf

[62] Maklouf, O., Ghila, A., Abdulla, A., and Yousef, A. (2013). Low Cost IMU \ GPS Integration Using Kalman Filtering for Land Vehicle Navigation Application. In *International Journal of Electronics and Communication Engineering 7* (2). 184-190. Retrieved from https://waset.org/publications/9996768/low-cost-imu-gps-integration-using-kalman-filtering-for-land-vehicle-navigation-application

[63] Jasinsky, M., Maczak, J., Radkowski, S., Korczak, S., Rogacki, R., Mac, J., and Szczepaniak. (2016). Autonomous Agricultural Robot: Conception of Inertial Navigation System. In *Robot scheduling subject to multi-project environment constraints.* pp 669-679. doi: 10.1007/978-3-319-2935-8_58.

[64] Andrade-Sanchez, P. Gore, M.A., Heun, J.T., K.R., Carmo-Silva, A.E., French, A.N., Salvucci, M.E., White, J.W. (2014). Development and Evaluation of a field-based high-throughput field phenotyping in soybean and wheat. In *Functional Plant Biology 4*(1), 68-79. doi:10.1071/FP13126.

[65] Bai, G., Ge, Y., Hussain, W., Baenziger, PS., Graef, G. (2016). A multi-sensor system for high throughput field phenotyping in soybean and wheat. In *Computers and Electronics in Agriculture. 128*(1), 181-192. https://doi.org/10.1016/j.compag.2016.08.021

[66] Bai, G. Ge, Y., Scoby, D., Leavitt, B., Stoerger, V., Kirchgessner, N., Irmak, S., Graef, G., Schnable, J., Awada, T. (2019). NU-Spidercam: A large-scale, cable-driven, integrated sensing and

robotic system for advanced phenotyping, remote sensing, and agronomic research. In *Computers and Electronics in Agriculture*. *160*(1), 71-81. https://doi.org/10.1016/j.compag.2019.03.009