Lane Detection and Following System for Autonomous Vehicles

Group 74
Curtis Davies, No. 101146353
Liam Gaudet, No. 101155009
Ian Holmes, No. 101149794
Robert Simionescu, No. 101143542

Carleton University
SYSC4907 - Engineering Capstone Project
Prof. Richard Dansereau, Prof. Chao Shen
30 October, 2023

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Definition
API	Application Programming Interface
CNN	Convolutional Neural Network
EOL	End of Life
FR[X]	Functional Requirement #X
K&C	(Lane) Keeping and Control
ML	Machine Learning
MPC	Model Predictive Control
MVP	Minimum Viable Product
NFR[X]	Non-Functional Requirement #X
PID	Proportional-Integral-Derivative
ROS	Robot Operating System
SAE	Society for Automotive Engineers
SBC	Single-Board Computer

1. OVERVIEW

The primary objective of this project is to create a system for autonomous vehicles that uses computer vision and hardware interfacing to allow the vehicle to monitor the position of lanes on its road and maintain a steady position within its own lane as it drives.

1.1. Identification of the need

Our project addresses a critical concern in autonomous vehicle development: the reliable detection and following of lanes in dynamic and diverse real-world environments. By enhancing the capabilities of autonomous vehicles to maintain their positions within lanes, our system aims to bolster road safety and reduce the need for human interaction for maintaining a steady course on trips in various kinds of roads and environmental conditions.

1.2. Problem Definition

The system is expected to function similarly to lane-following technologies in existing cars, such that they can autonomously maintain the vehicle's position within its lane on the road on which it drives. The system is required to make the required calculations in real time and adjustments need to be smooth enough as to not introduce an unsafe level of jerk to the steering of the vehicle.

1.2.1. Functional requirements

- **FR1: Lane Detection Accuracy**
  The system should accurately detect all lane borders within 10 centimetres of their actual position on the road 75% of the time lane positions are calculated in a given video frame. This can be benchmarked using lane detection datasets that provide ground truth values for lane positions to measure against when testing.
- **FR2: Lane Following Precision**
  The system should maintain a deviation of no more than 0.2 metres from the centre of its lane under normal driving conditions.
- **FR3: Real-time Processing**
  The system should complete the entire processing pipeline from video frame to hardware instruction in 150 milliseconds or less.

1.2.2. Non-Functional Requirements

- **NFR1: Reliability**
  The system should have a minimum uptime of 99.5% to ensure consistent operation during driving.
- **NFR2: Compliance**
  The system should comply with any regulations and standards put in place by the Government of Ontario and the SAE for behaviour of automated driver assistance systems within vehicles.
- **NFR3: Extensibility:**
  The system should provide an API that allows for programmatic control and access to its subsystems to allow any future autonomous vehicle systems to interact with it in tandem.
- **NFR4: Documentation**
  The system should provide adequate documentation of its processes, software and runtime behaviour such that any technically-oriented person not previously affiliated with the system's development can easily understand its functionality and inner workings.

1.2.3. Constraints

Given an allocated time of 8 months to complete the project, the timeline is a significant constraint and needs to be heavily accounted for when determining work to be completed. The prioritization and planning of hardware acquisition, development, testing, debugging and documentation of the system is essential for the completion of the project in such a short timeframe, which may limit the scope of our project based on available time as it progresses.

The team consists of four Software Engineering undergraduate students, each with varying levels of understanding of different variants of software development and mostly minimal experience with electronics and hardware development. This limits the hardware complexity of our system, meaning our team should focus more on the software aspect of development and use off-the-shelf hardware components.

With an allocated budget of C$500, cost-effective solutions must be prioritized for the project. The team aims to use hardware borrowed from the school wherever possible to minimize the cost of building the system.

1.3. Conceptual Solutions

1.3.1. Literary Review

In the landscape of commercial lane detection and following systems, several existing products and technologies exist in the public domain. Many consumer car manufacturers provide lane assistance as part of a general driver assistance system, such as Subaru's EyeSight [1] and Honda's Sensing [2] packages. These systems rely on camera-based vision sensors and quick image processing algorithms to detect lane markings and assist drivers in maintaining their lane position. While their the responsiveness they provide is fast and the systems work in ideal conditions, changes in light levels or unfavourable weather conditions will often render these systems completely unusable as the computer vision algorithms powering their lane detection are not sophisticated enough to detect lines without a stark contrast in light levels between the lines and the road.

Fig. 1. EyeSight System installed behind a vehicle's windshield [1]

On the research front, novel approaches to lane detection have gained prominence over the past 5 years, given the recent increase in capabilities of machine learning for computer vision available to small teams of developers. Approaches such as Row-wise Classification [3] of pixel data or parametric curve modeling [4] enable researchers and individuals to create model lane detection systems on hardware strong enough to run moderately complicated CNNs and other deep learning techniques. While these deep learning approaches are often more robust in varying environmental conditions, the hardware they require to process image data to extract lane position information may be too expensive for an individual researcher or for mass production in consumer-grade vehicles.

Our system needs to be robust enough to accurately and rapidly extract lane positioning information based on a video feed from the car, while avoiding the need to implant a large amount of processing hardware in the vehicle itself. Once a working model of the system is successfully implemented, the challenge will be optimizing the system to minimize its hardware footprint and the coupling between the software system and the hardware on which it is installed.

1.3.2. Concepts

- **Onboard Processing**: This concept for the system would involve having a single piece of hardware perform all aspects of data processing, between lane position extraction, decision making and vehicle control interfacing. The hardware would need to be a small-enough SBC to reasonably fit in a vehicle, while also being powerful enough to handle somewhat complicated machine learning techniques. Something like the NVIDIA Jetson [5] platform may prove suitable for this task, but could be expensive given the project's budget.

- **Distributed Processing**: This concept involves offloading computationally expensive processes, such as deep learning image analysis, to a separate computer that is not included within the system installed in the vehicle. This allows us to use larger, more powerful hardware and have a smaller install base on our physical system. However, this would present the challenge of being able to efficiently transfer data between subsystems while minimizing latency, as this is a real-time system with a need for immediate calculations while travelling at high speeds on a road.

1.4. System Architecture

The overall architecture of our system is best described as a combination between a pipeline architecture and a microservice architecture. The system starts off by receiving an image from the camera mounted on the JetAuto robot car. The image will be processed into the software, and given to the computer vision node where the image will be analysed to determine the location of the lane lines as well as the center of the lane. Afterwards, this information is passed to the lane keeping section where the movement commands are determined which is then sent to the movement node where movement commands are converted into proper machine instructions.

1.4.1. Software Architecture

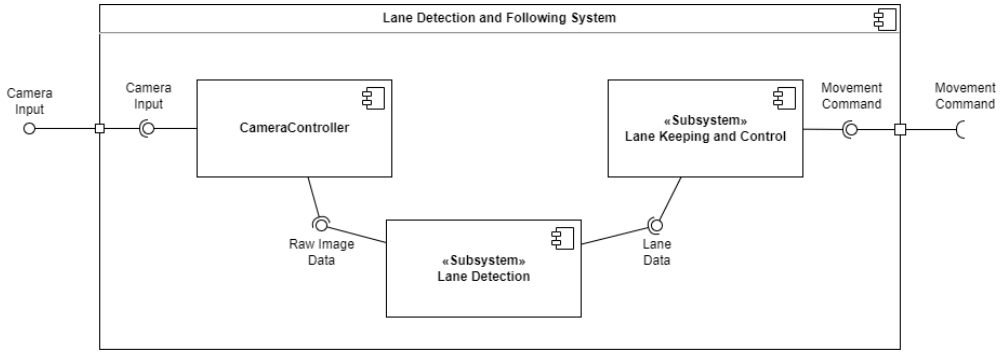

Fig. 2. Component Diagram describing the software architecture of the Lane Detection and Keeping system.

From a software perspective, the system will consist of two main subsystems: Detection and Keeping & Control (K&C), as well as the CameraController component. The CameraController component is an interface between the input sensor hardware and the Detection subsystem to transfer raw image data. The Detection subsystem receives this raw image data and determines the vehicle location within a lane, and sends this data to the Keeping & Control subsystem. The K&C subsystem interprets the lane data and produces movement commands to adjust the course of the vehicle. The two subsystems are discussed further in section 3 of this proposal.

1.4.2. Physical Architecture

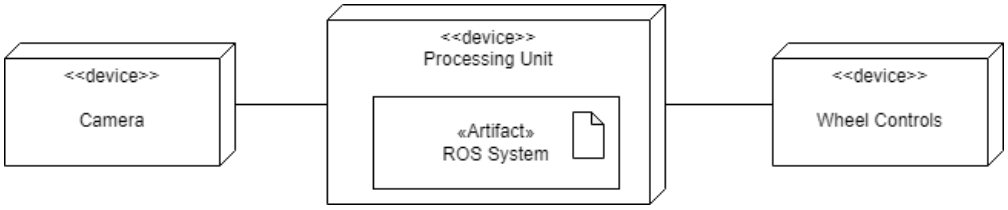

Fig. 3. Deployment diagram describing the hardware-software relationships describing the physical architecture of the Lane Detection and Keeping system.

The physical architecture for the system is fairly simple with only three nodes: the camera, the processing unit with the ROS environment, and the hardware/simulation for controlling the vehicle wheels. First, the camera sensor(s) sends data to the processing unit which contains the software architecture in section 1.4.1 to complete all calculations. The system then outputs any commands to the simulation or robot to perform corrections.

2. WORK PLAN

2.1. Work Breakdown Structure

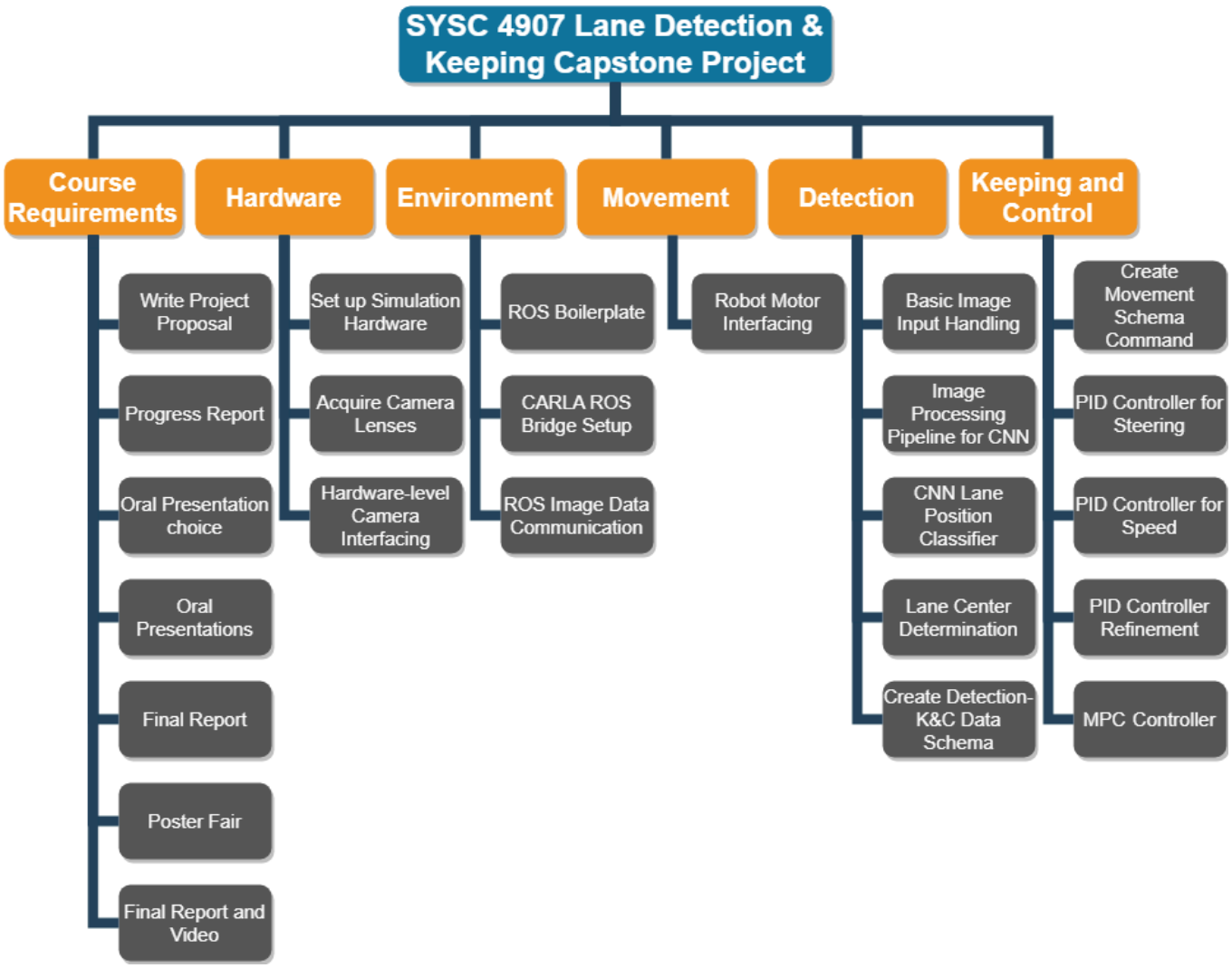

Fig. 4. Work Breakdown Structure of the Lane Detection and Keeping system.

2.2. Responsibility Matrix

Table 1
Responsibility Matrix for the Lane Detection and Keeping System

Step	Project Task	Curtis	Liam	Ian	Robert
1	Write Project Proposal	R	R	R	R
2	Progress Report	R	R	R	R
3	Oral Presentation	R	R	R	R
4	Final Report	R	R	R	R
5	Poster Fair	R	R	R	R
6	Final Report and Video	R	R	R	R
7	Set up Simulation Hardware	R			
8	Acquire Camera Lenses	S		R	
9	Hardware-level Camera Interfacing			R	S
10	ROS Boilerplate				R
11	CARLA ROS Bridge Setup	S			R
12	ROS Image Data Communication	S			R
13	Movement				R
14	Basic Image Input Handling	R			
15	Image Processing Pipeline for CNN	R		S	
16	CNN Lane Position Classifier	R		S	

17	Lane Center Determination	R		S	
18	Create Detection-K&C Data Schema	S		R	
19	Create Movement Schema Command		R		S
20	PID Controller for Steering		R		S
21	PID Controller for Speed		R		S
22	PID Controller refinement		R		S
23	MPC Controller		R		S

R = Responsible; S = Support

2.3. Project Network

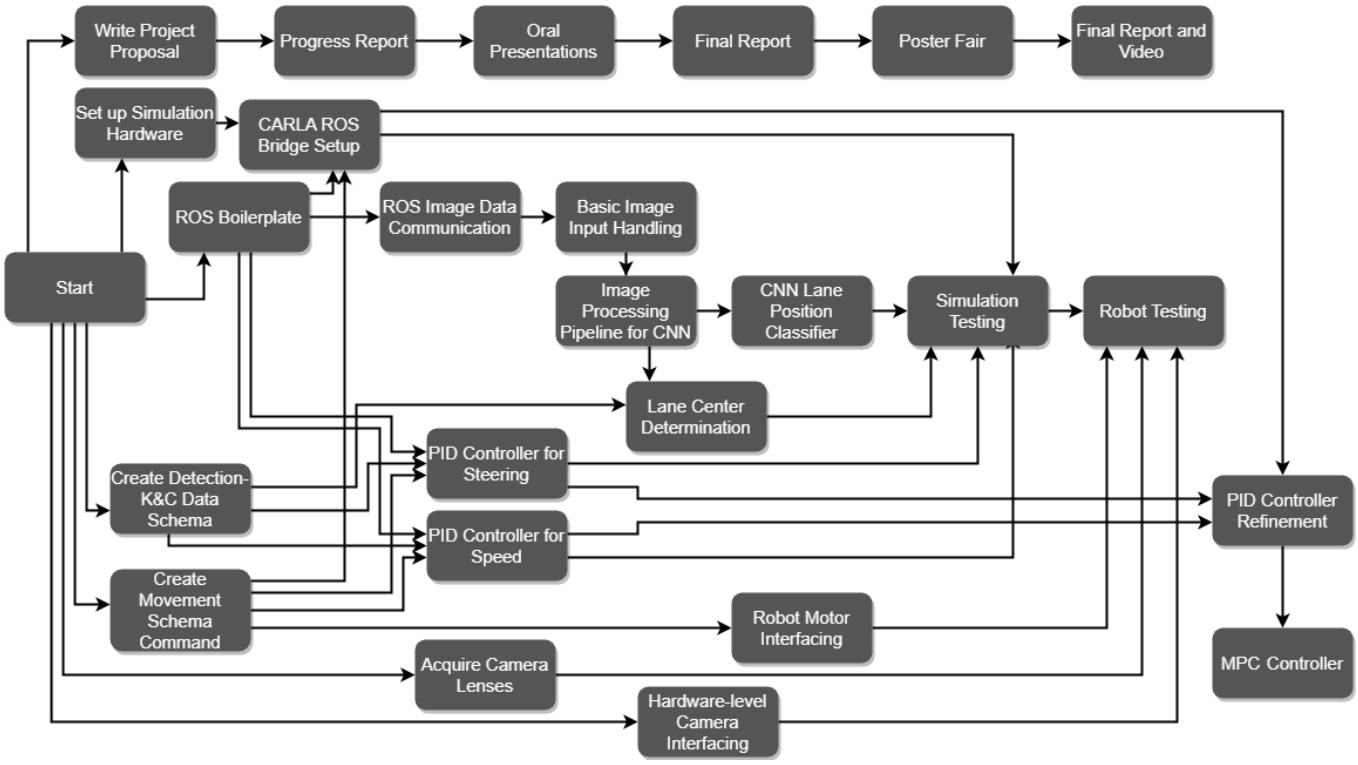

Fig. 5. Project Network of the Lane Detection and Keeping system.

2.4. Gantt Chart

The project's Gantt Chart can be found on our project's ClickUp page found here. Alternatively, it can be viewed below.

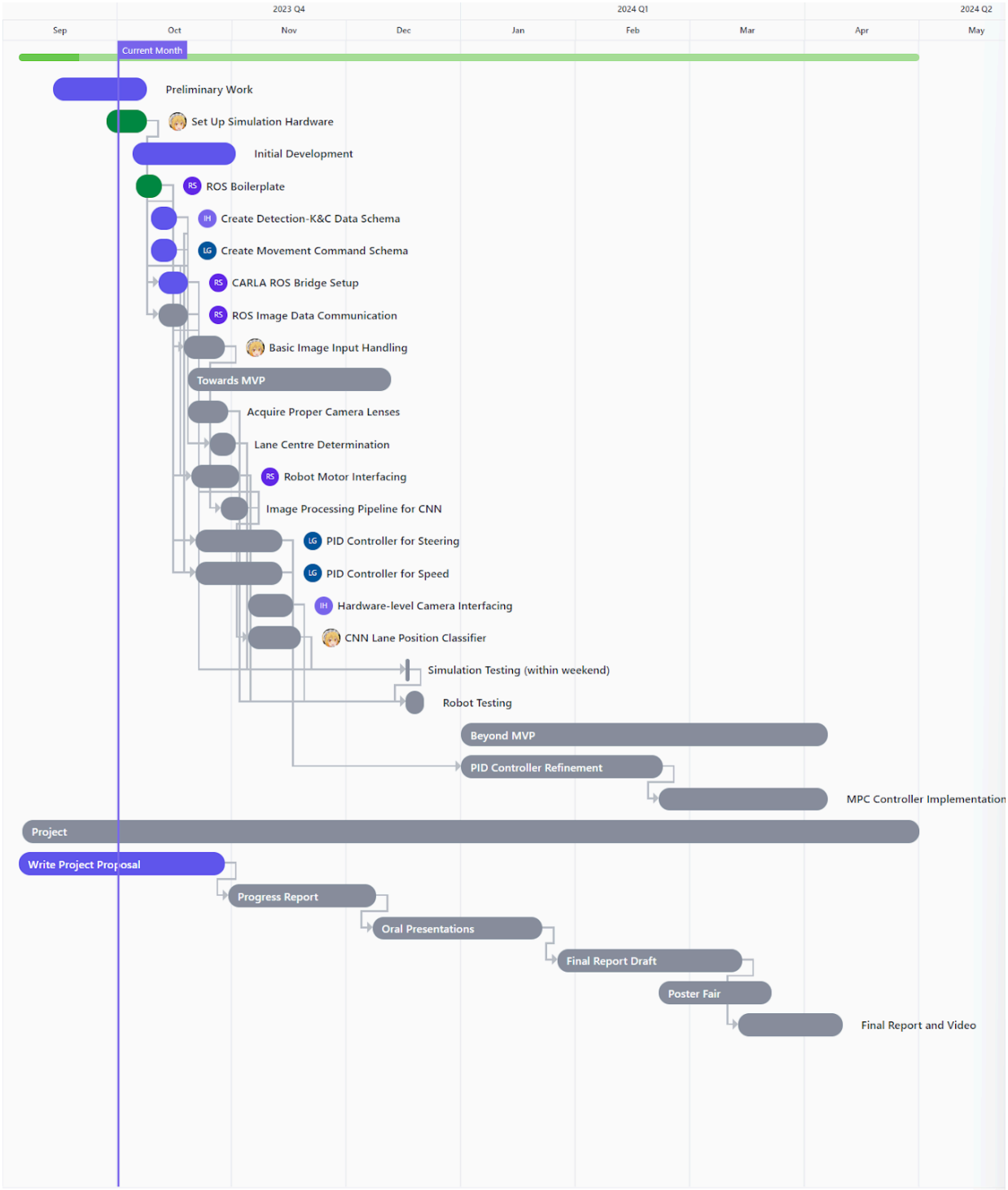

Fig. 6. Gantt Chart of the Lane Detection and Keeping system.

2.5. Costs, Special Components and Facilities

Table 2
Costs and Equipment Breakdown for Lane Detection and Keeping System

Item	Description	Predicted Cost
Hiwonder JetAuto ROS Robot Car	A robot that can have cameras configured onto the frame and respond to movement commands.	0$ Use the existing robot owned by Carleton
x2 daA1440-220uc Basler Dart camera with CS mount	Two cameras that can be used in stereo to provide proper imaging for the lane detection.	0$ Use the existing cameras owned by Prof. Dansereau
x2 Camera Lenses	Two lenses that provide the necessary FOV and FStop values for imaging processes.	~$60
Ubuntu Box	A Ubuntu computer capable of running the Carla Simulation software that can be accessed remotely for testing software.	0$ Use the existing computer owned by Carleton
Total:	-	= $60

2.6. Risk Analysis

Table 3
Risk Levels and Definitions for the Lane Detection and Keeping System

RISK LEVEL		PROBABILITY of the event occurring		
		Low	Moderate	High
SEVERITY of the event on the project's success	Low	VERY LOW	LOW	MEDIUM
	Moderate	LOW	MEDIUM	HIGH
	High	MEDIUM	HIGH	VERY HIGH

Risk Level	Description
Very Low	Very low risk, does not require any significant plans for mitigation.
Low	A preliminary study on a plan of action to recover should be performed and documented.
Moderate	A significant risk. A plan of action to recover should be made in advance.
High	A very significant risk that warrants considering a change in the project to address, or a plan of action should be made.
Very High	An unacceptable risk. The project should be changed to address this risk.

Table 4
Risk Identification and Mitigation for the Lane Detection and Keeping System

Identified Risk	Probability	Severity	Risk Level	Plan of Action
1. Neural Networks / ML is too complicated	**Low** There are lots of previous templates and examples to draw from.	**High** Failure to construct a proper neural network would result in the "lane detection" part of the project being incompletable.	**Moderate**	Since the complexity of the Neural Network represents a significant risk, should this part of the project prove too complex a switch to a more algorithmic approach for lane detection will be used.
2. Hardware not powerful enough	**Moderate** Due to the limited strength of the onboard processor, it is possible that it will not be powerful enough to run the neural network.	**Moderate** The neural network will be a soft real-time system which means if it is slow it will not significantly affect the success of the project	**Moderate**	The onboard processor not being powerful enough represents a significant risk due to breaking soft real-time requirements and being likely to happen due to the smaller size of the onboard processor. Should this situation be encountered, a distributed processing system should be used where neural network processing is done by a remote laptop.
3. Scope Creep	**Low** After the proposal, no un-proposed features will be implemented unless all other performable work has already been accomplished.	**Low** Due to the nature of starting development with an MVP, scope creep is unlikely to affect the first deliverable.	**Very Low**	Scope Creep represents a very low risk due to an early development of an MVP which means any scope creep that is encountered can be disregarded with minimal impact to the overall project.

Identified Risk	Probability	Severity	Risk Level	Plan of Action
4. Future Integration Challenges	**Moderate** While documentation and code readability will be prioritized, the time constraints put on the project could lead to quick solutions to problems that may not be easily read by future teams integrating our system with other autonomous driving systems.	**Low** Integration with future self-driving systems is preferred but not critical to the project. Having software that is less interoperable with other services will not severely impact our system in itself.	**Low**	To mitigate reduced interoperability, we plan to maintain well-organized documentation, on both a code and structural level, and ensure that code is properly annotated with comments so future groups can read it with ease and understand how the system works. In addition, data types passed between subsystems will be documented and made available so other systems within a vehicle can interface directly and easily with ours.
5. Software Version Incompatibility	**Low** ROS 2, Ubuntu, Carla, and the Carla ROS bridge all have their own versions that are only compatible with specific versions of the others, but these are fairly clearly documented and problems should be straightforward to avoid.	**Low** The worst-case scenario is that an incompatibility between chosen versions is found and cannot be worked around, in which case we can migrate to versions that are compatible, which may require changing Ubuntu versions but should not require redoing much work.	**Very Low**	We have checked the version requirements of all the software we are using to ensure they are compatible. The only issue that may arise is with the Carla ROS bridge, which is meant to be used with ROS 2 Foxy, while we are using a newer version, ROS 2 Humble. Foxy is EOL, so we believe the risk of the ROS bridge not working is lower than the risk of encountering issues with Foxy.

3. SUBSYSTEMS

3.1. Lane Detection

The Lane Detection Subsystem is responsible for interpreting first person camera footage of a vehicle traversing a roadway, and detecting/determining the location of the lane markers (lines) as well as the vehicle's position within the lane.

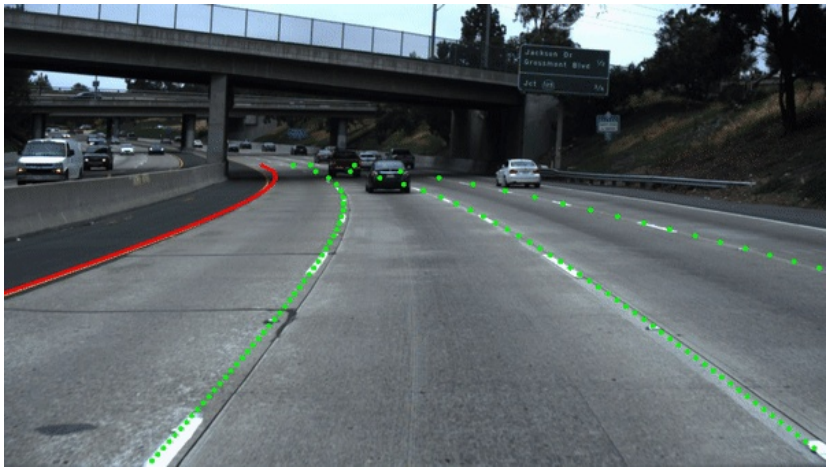

Fig. 7. Road line detection algorithm identifying painted lines on the road. This data is used to calculate the centre of the vehicle's lane. [7]

3.1.1. Requirements

Requirement from system-wide requirements:

- **FR1: Lane Detection Accuracy**
  The system should accurately detect all lane borders within 10 centimetres of their actual position on the road 75% of the time lane positions are calculated in a given video frame. This can be benchmarked using lane detection datasets that provide ground truth values for lane positions to measure against when testing.

Subsystem-specific requirements:

- **FR1.1: Environmental Adaptability**
  The subsystem should accurately detect lanes in diverse environments, including different weather conditions (e.g., rain, fog, sunlight) and on both straight and curved road segments.
- **FR1.2: Real-Time Lane Position Determination**
  The subsystem should efficiently and promptly determine the vehicle's location within the lane and provide this information to the Keeping & Control module at a rate of 20 updates per second for timely decision-making.
- **FR1.3: Machine Learning Integration**
  The subsystem should use machine learning to effectively and reliably determine the positioning of road lines, enhancing accuracy and adaptability across various road and weather scenarios.
- **FR1.4: Lane Position Memory**
  The subsystem should keep a short-term memory of the positions and curvature of lanes based on previous video frames to maintain a high level of accuracy when analyzing new image data.

3.1.2. Technologies and Methods

The Lane Detection Subsystem will be implemented in Python, leveraging the OpenCV library for low-level image data parsing and preprocessing. For deep learning methodologies for road line detection, we will utilize PyTorch, along with Transformers and other pertinent machine learning Python libraries.

This subsystem will demonstrate versatility in accepting video input from a diverse range of potential sources. The subsystem will need to work smoothly with different types of cameras, including those on our simulation platform (CARLA), the robot (JetAuto), and real cars with Basler Cameras. This way, it can handle various sources of image data during development and testing.

During development, methodologies from several courses at Carleton University will be used. Detecting lane markings will involve using computer vision to identify characteristics of the lines to determine where lanes are. This knowledge is covered in the COMP 4102 (Computer Vision) course at Carleton. The detection process

will also use a Convolutional Neural Network (CNN) to classify sections of the image and detect road line markings. We learned how to use convolutional neural networks in our SYSC 4415 (Introduction to Machine Learning) course, along with how to use them for image processing and classification in real-world settings.

3.1.3. Conceptualization

Image Data Preprocessing

We plan to test the efficiency and accuracy of the Road Line Detection model with a variety of preprocessing techniques used on image data before it is passed into the model. We aim to find the optimal set of preprocessing techniques allowing the model to make more use of input data than it would with standard RGB image data.

- **Perspective Transformation:**
  This technique applies a distortion to the source image, simulating a 3D rotation that mimics a view from a different angle. This rotation, as if observing the road from a top-down perspective, can assist in accurately determining horizontal road positions relative to the vehicle.

- **Histogram Equalization:**
  This technique balances out pixel brightness values across the image to boost contrast. This will likely help increase the difference in values between pixels belonging to a darker, paved road and a lighter, painted road line and aid in detection.

- **Colour Space Conversion:**
  This process involves changing how colours and intensity values are represented in the image, rather than RGB values. The Lab colour space separates colour information from brightness, potentially allowing for a more easy distinction between the road and its lines.

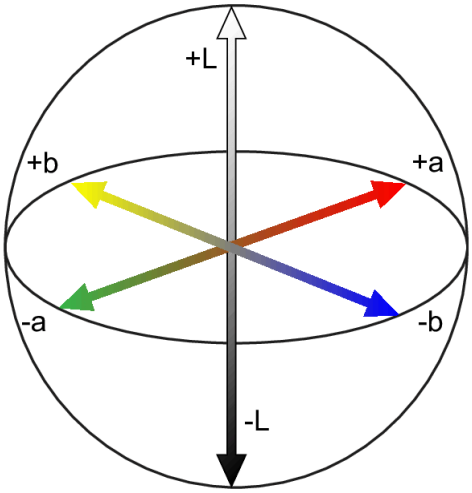

Fig. 8. Visual representation of the Lab colour space. Colours are represented as an L, a and b value, rather than Red, Green and Blue in the RGB colour space. [8]

Road Line Detection

We have looked into multiple approaches for road line detection from image data. Both implementations use a CNN to analyze preprocessed image data to find the locations of road lines within a video frame.

- **Raw Curve Modeling**:
  This approach analyzes every pixel on each video frame and uses edge detection to pick up on strips of different lightnesses on pixels to determine those belonging to road lines. After finding all pixels related to road lines, determine which pixels belong to which lines and draw parametric curves to represent them.

- **Row-Wise Classification:**
  This approach splits each video frame into 1-pixel-tall rows and scans along each row with a classifier model to find the position of each line on each row of pixels. This narrows the scope of analysis for a classifier, potentially allowing for more accurate results.

3.1.4. Software Architecture

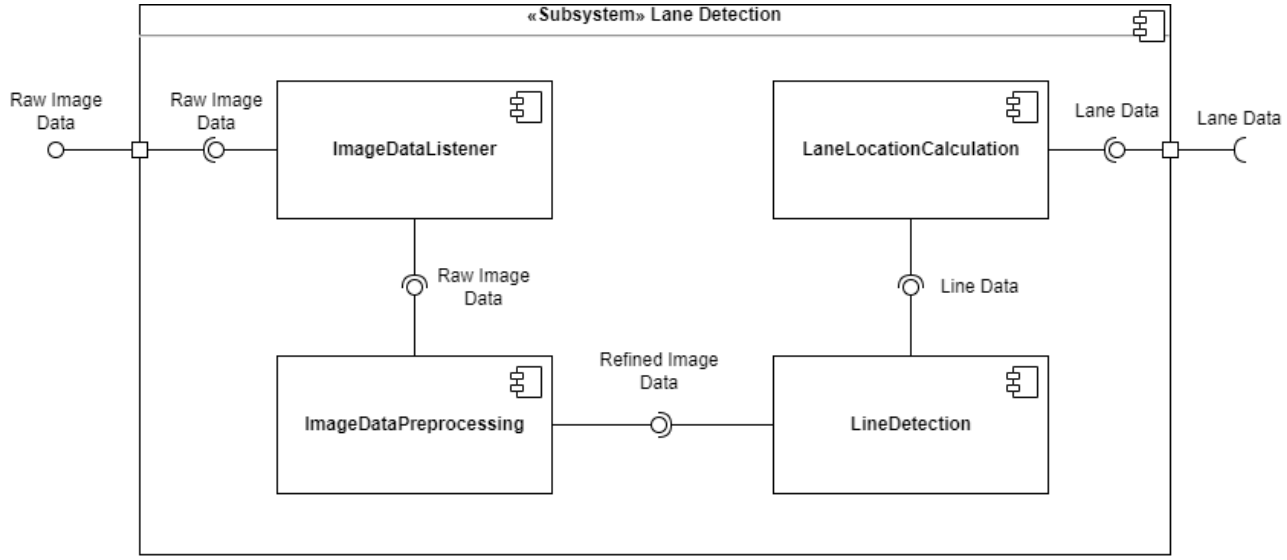

Fig. 9. Component Diagram describing the software architecture of the Lane Detection subsystem.

The Lane Detection subsystem receives raw image data from input sensors by subscribing to a ROS topic. The ImageDataListener takes this data and forwards it to the ImageDataPreprocessing component to format the data for detecting lane markings/lines. The LineDetection component processes the refined image and determines line location data with reference to the vehicle. The line data is then forwarded to the LaneLocationCalculation component which determines where the vehicle is within the detected lane and outputs this data to the K&C subsystem.

3.1.5. Implementation

The implementation of the Detection subsystem will begin by specifying the lane data format for messaging with the K&C subsystem. The messaging format will include the vehicle position relative to the center of the lane which will be used to calculate any adjustments to be executed. The messaging specification must be implemented first to allow for parallel development of the K&C subsystem.

After the specification has been cemented, the hardware interfacing for the camera sensors will be developed. This interface will include developing the CameraController ROS node which will publish raw image data to a ROS topic which the Detection subsystem will subscribe to.

The next step in the implementation will be to create the ImageDataListener which will read new image data from the raw image data ROS topic. This listener will

likely involve using the rclpy python library to read incoming image data to be refined by the ImageDataPreprocessing component.

The image preprocessing will consist of turning the raw image data into more usable data for determining line markings on the roadway. The specifications for how the data will be refined and what the usable data will look like must be determined by considering the chosen model for lane positioning.

Finally, the line detection and lane positioning steps will be implemented to create the lane location data to be used by the K&C subsystem. Detecting the lane markings/lines from the input images will make use of the OpenCV python libraries and give an accurate depiction of where the lane is. The current vehicle location within the lane will be determined by considering the positioning of the camera on the vehicle to calculate the difference in reference frames. The current vehicle location will be published to a ROS topic in the defined messaging specification to allow the K&C subsystem to calculate adjustments.

3.1.6. Evaluation

As autonomous driving is becoming a more popular application for individuals studying deep learning, various online datasets containing test data with dash cam video footage and ground truth pixel location values for road lines within each video frame. Datasets include TuSimple, CULane and Llamas, each with aforementioned ground truth values for road line pixel locations for model training.

These ground truths can be used when implementing supervised learning for our classifier, allowing us to determine our model's precision and recall. Precision is a metric that determines out of each positive guess that a pixel belongs to a line, which is correct. Recall determines out of each pixel belonging to a line, how many were identified by the model.

In our refinement of our model, we want to balance precision and recall, but focus more on precision as our primary metric. If the model fails to detect a line for a few frames due to an issue with recall, the system can remember where lines are from previous frames and continue to drive normally. However, if the system believes additional lines exist on the road that aren't actually there (due to excessive recall and a lack of precision), it could lead to very erratic driving on the part of the movement subsystem.

3.2. Lane Keeping and Control

The Lane Keeping & Control (K&C) subsystem is responsible for taking in lane position data with relation to the vehicle, determining the best corrective action to take to steer the vehicle towards the centre of its lane and execute the appropriate instructions to the hardware/simulation.

3.2.1. Requirements

Requirement from system-wide requirements:

- **FR2: Lane Following Precision**
  The system should maintain a deviation of no more than 0.2 metres from the centre of its lane under normal driving conditions.

Subsystem-specific requirements:

- **FR2.1: Steering Control Stability**
  The control subsystem should limit the amount of movement fluctuations due to overcompensation of steering.
- **FR2.2: Control Frequency**
  The subsystem must produce movement command updates at a frequency of 50Hz.
- **FR2.3: Closed Loop Properties**
  The subsystem must produce movement commands using the most recent information available.
- **FR2.4: Output Attributes**
  The output of the subsystem should be appropriate for the wheel configuration (e.g. Ackermann geometry, mecanum wheels, etc).

3.2.2. Technologies and methods

The development of the K&C subsystem requires the use of three primary technologies and software concepts.

For starters, the K&C system will require the use of multi-threading. Multithreading is the process by which a system has multiple concurrent threads, a program in execution, running concurrently in parallel. There are several advantages of using multi-threading, including better performance due to being able to run tasks

during I/O blocking time, greater scheduling flexibility, and it separates the concern of what a task does and when it does it. With respect to our system, Multithreading will need to be used to have our PID processing run in parallel to receiving new system environments from the lane detection subsystem. Multithreading was instructed as part of the SYSC 3303 Introduction to Real Time Systems course offered at Carleton.

Another important methodology for this project is that of feedback control systems. These systems are control systems that incorporate comparing the measured variable with its target value and then manipulates the system to minimize this error[6]. Feedback control systems are advantageous since the controller takes into account any unforeseen changes present in the system such as friction from the environment or older data such as from long input processing times, both of which are likely to happen with our system[6]. These concepts are taught as part of the SYSC 3600 and SYSC 4505 courses offered at Carleton University. While these courses are not required as part of a Software Engineering degree, they can be taken as electives for students in Software Engineering and the project supervisors have experience with this concept that can be shared with the students.

Cars and car-like vehicles use an arrangement of linkages called an Ackermann steering geometry in order to steer effectively[17]. It allows for the wheels on the inside and outside of a turn to trace circles of different radii, preventing slipping. In systems that use Ackermann steering or something similar, the front wheels turn while the rear wheels remain in place. In contrast, robots like the JetAuto use mecanum wheels, which allow for omnidirectional movement. While this is a useful property for robots, we intend to use the JetAuto to evaluate the system’s capabilities for car-like vehicles. As such, this subsystem should be able to send commands to vehicles with mecanum wheels that result in Ackerman-like movement. While this concept was not explicitly taught as part of a software engineering degree at carleton, the integration of hardware components with embedded systems relates to the practices discussed in SYSC 3310 Introduction to Real-Time Systems.

3.2.3. Conceptualization

There are many different control loops that could be used to handle the lane keeping and control requirements for this sub-system. However, out of all these methods, there are two that have stood out for us as potential conceptual solutions.

PID Controller:

This control loop relies on adjustable parameters and takes in an error signal, which measures the deviation between the vehicle's position and the center of its

driving lane. It then produces a sinusoidal curve with the goal of minimizing this error as rapidly as possible while avoiding excessive overshooting[13]. The main advantage of this method lies in its user-friendliness. Implementing a PID controller is straightforward as it requires only three variables for configuration, making it a preferable control system. However, a PID controller has its limitations. The main limitation with a PID controller is the limitation from only being able to tune using three parameters[14]. We anticipate that it may not be the most optimal control system due to its relatively simple configurability and handling of unexpected changes in the road path.

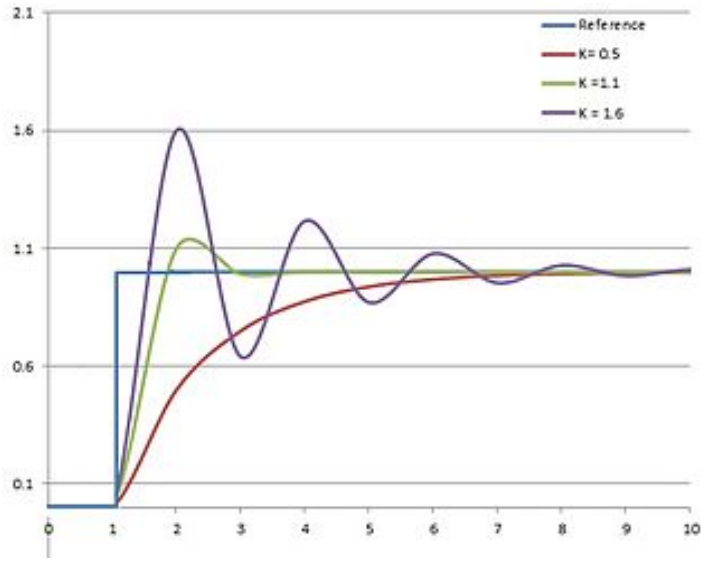

Fig. 10. Example of a PID controller with different values passed as parameters. Parameters need to be tuned to ensure the curve approaches the target value quickly, but with minimal overshoot. [9]

MPC Controller:

The MPC controller is a more sophisticated control system compared to the PID controller, with use in various industries and areas in academia.[16] It utilizes a model of the system, a cost function, and an optimization algorithm to determine the optimal strategy for minimizing error, which in this case is the distance between the car's center and the center of its lane. One significant advantage of this controller is its potential for high accuracy when implemented effectively. The MPC controller takes a wider array of variables and information into account when making decisions, offering greater potential for improved performance metrics. However, it comes with the drawback of increased complexity. Unlike a PID controller, which is relatively straightforward to set up and configure, implementing an MPC controller involves

considering a larger set of variables and a model of the environment surrounding the system, making it more challenging during the initial stages of development.

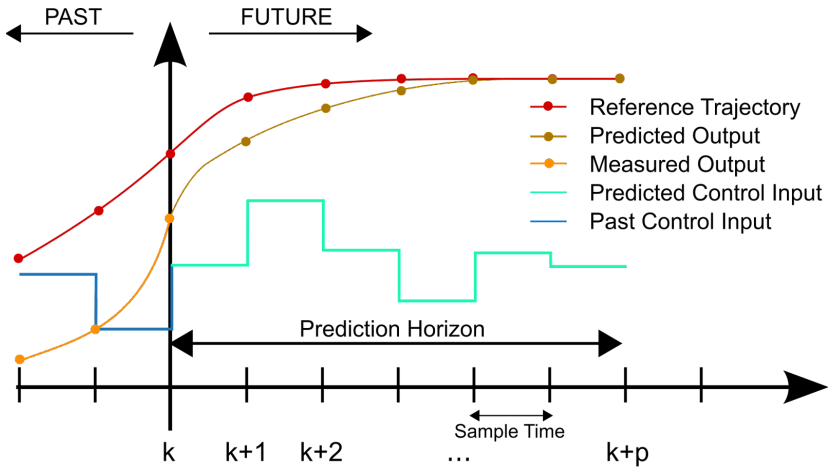

Fig. 11. Example of an MPC controller tuning its parameters using a model of its environment, predicting what values will be needed to correct its trajectory. [10]

Preferred Solution:

The initial approach for the K&C subsystem will involve first implementing a PID controller. Once this controller has been fine-tuned and validated, we will consider transitioning to an MPC controller. This choice is driven by the fact that constructing a PID controller carries significantly lower risk compared to an MPC controller. Given our priority to swiftly achieve an MVP, the PID controller is less likely to introduce delays in MVP development. However, recognizing the MPC controller's potential for a more robust corrective system compared to the PID controller, if sufficient time allows after the successful implementation and refinement of the PID controller, we will explore the development of an MPC controller.

3.2.4. Software Architecture

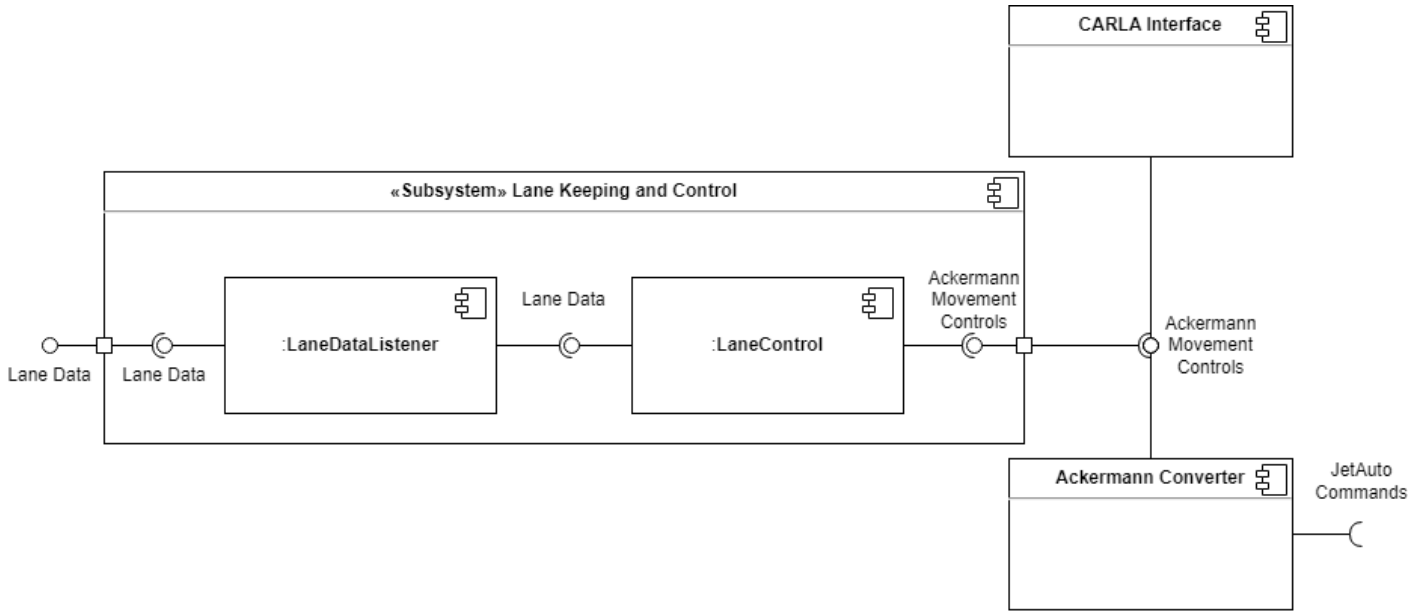

Fig. 12. Component diagram describing the software architecture of the Lane Keeping and Control subsystem.

The Lane Keeping & Control subsystem receives lane data from the Detection subsystem via a ROS topic using a publish-subscribe approach, allowing for asynchronous communication to accommodate different processing rates. The LaneDataListener component captures the lane position data and updates the LaneControl component. This component operates on a 50Hz timer, determining the optimal angle and velocity for the vehicle to stay centered in its lane. The output is formatted as an `ackermann_msgs/AckermannDrive` message, a data type provided by ROS, and then published to a separate ROS topic. This data is subsequently utilized by either a hardware module (simulated or real) capable of responding to Ackermann steering signals, or a software component that interprets the Ackermann steering commands and converts them into a compatible format.

3.2.5. Implementation

The build plan for this subsystem will begin with outlining the exact specification of the lane data that the system will receive and the exact format of the movement controls to export. This is because once these properties are established, every node in the system can be implemented in parallel rather than in series.

Following the decision of the ROS topic formats, the listener for the lane data will be implemented. This will likely involve creating a listener for the published message, and storing it in a shared box that the PID controller can access, but the listener can write to in order to update it with the latest information.

After the listener has been implemented, the PID controller will be configured as a stub. The purpose of this is so that while the controller is being implemented, other systems can still listen to topics published by this node and receive correct data.

Next, the development of the PID controller will be performed. There are a lot of resources available for designing PID controllers, and the course notes on SYSC 3600 will provide a string foundation from which to build the controller. This is expected to take the most time to accomplish due to the high risk involved with learning and implementation.

Once an MVP is complete, a system for using Ackermann steering messages with mecanum-wheeled vehicles to simulate Ackermann steering will be designed and implemented in order to allow control of the Hiwonder JetAuto by this system.

3.2.6. Evaluation

When it comes to evaluating the effectiveness of a closed controller, there are many metrics used for measuring its performance. Historically, the main measurements used for evaluating a PID controller are the settling time, overshoot, and rise time. More complex measurements such as Integral Absolute Error exist, but for the scope of this project we will keep focus on the three listed metrics.

For starters, the settling time of a PID controller is defined as the last moment the step response enters the 5% error strip and never leaves from that point on[15]. Equation (1) represents how to calculate the settling time for a given equation.

$$t_s = \min \{ t > 0 : |y(t') - y(\infty)| / y(\infty) \leq 0.05 \text{ for all } t' > t \} \quad (1)$$

The overshoot of a system is also an important measurement to consider with respect to the controller. Overshoot is defined as the maximum amount by which the response overshoots the steady-s-tate value, and is typically seen as the amplitude of the first peak[15]. Mathematically speaking, when a control system experiences an overshoot, its behaviour can be typically represented as a sinusoidal function oscillating about the point 1 in the form of (2).

$$y(t) = 1 - e^{-\sigma t}(\cos(\omega_d t) + (\sigma/\omega_d)\sin(\omega_d t)) \quad (2)$$

Given the nature of when the first peak would appear with respect to this sinusoidal equation, the time at which the first peak takes place(t_p) can be seen in (3).

$$t_p = \pi/\omega_d \quad (3)$$

Therefore, the overshoot of the sinusoidal equation representing the control system can be evaluated as per (4) with reference to Fig. 13.

$$M_p = y(t_p) - 1 \quad (4)$$

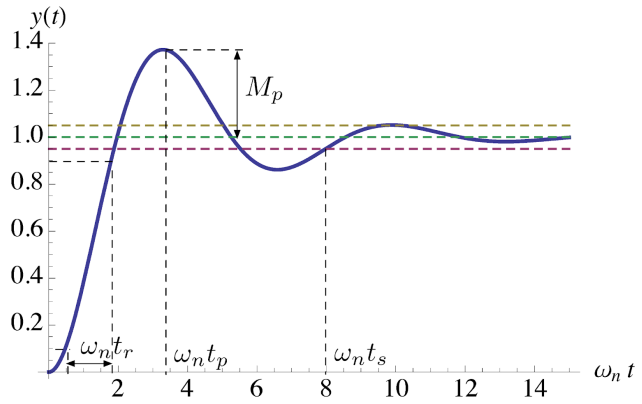

Fig. 13. The calculation of overshoot with respect to a controller experiencing sinusoidal behaviour.

Lastly, the rise time of the controller is also something to consider. The rise time of a controller is the time it takes to get from 10% to 90% of the steady state value[15]. The rise time of a system can be evaluated with (5), with reference to Fig. 14.

$$t_r = t_{0.9} - t_{0.1} \quad (5)$$

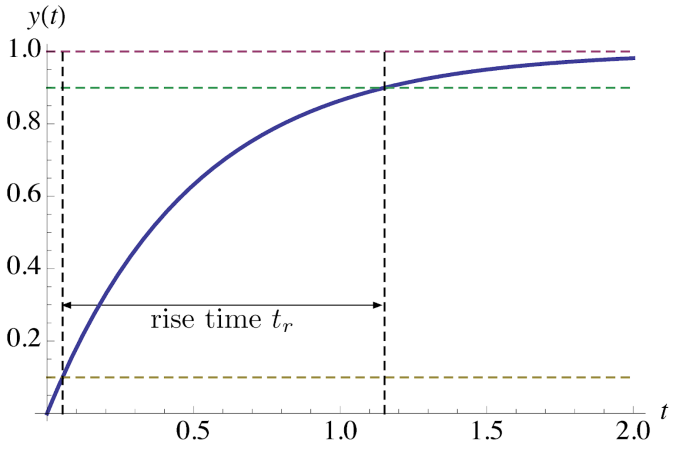

Fig. 14: The calculation of rise time with respect to a controller experiencing sinusoidal behaviour.

As is the case with most PID controllers, it is preferable to have each of the three characteristics M_p, t_r, and t_s be as low as possible. Having a low overshoot is important since we want to ensure that the car doesn't accidentally veer into another lane. Having a low rise time is important since we want to make sure that the vehicle doesn't take too long getting into the lane if it slips out of it. And lastly, having a low settling time is also valuable since a low settling time means that the car reaches the center of the lane and stays there as soon as possible.

With respect to these three characteristics, we intend to primarily focus on minimizing overshoot as we do not want the vehicle to accidentally swing into another lane. Secondly, minimizing the settling time is also important since we want to reduce the amount of time it takes for the car to steadily follow the center of the lane path. Lastly, while we want to still minimize the rise time, we will not put as much focus on it as rise time tends to be inversely proportional to the overshoot, meaning that it is hard to minimize both.

To assess the PID controller's performance, we will rely on external measurements. Although we anticipate continuous feedback from the control system, potential delays in neural network response may necessitate the use of older data. Therefore, we will introduce an external monitoring system to mathematically trace the controller's trajectory, enabling accurate evaluation of overshoot, rise time, and settling time.

4. SYSTEM INTEGRATION AND EVALUATION

4.1. Integration

Recall that the system is made up of two primary subsystems: *Lane Detection* and *Lane Keeping & Control*. The Lane Detection subsystem processes real-time video camera footage from a vehicle as it drives, identifies lane markers and calculates the centre of the vehicle's lane. This information is relayed to the Keeping & Control subsystem, which takes this data and determines the best corrective action to keep the vehicle centered in its lane. The K&C subsystem is responsible for generating steering and acceleration commands based on the received lane position data, which is then sent to the vehicle hardware for execution.

4.1.1. Orchestration with ROS

ROS is a set of software libraries under Linux that allow for the easy orchestration and data transfer between processes running on a robot.[11] The system will use ROS as a framework for orchestrating processes, facilitating data exchange and enabling interaction with the system's surrounding environment. ROS will serve as the backbone that synchronizes the Lane Detection and K&C subsystems, with each subsystem running as a set of "nodes" that are managed by ROS and communicating via ROS message topics. This architecture facilitates efficient communication and coordination between the Lane Detection and K&C subsystems. ROS topics allow the subsystems to communicate asynchronously, meaning the rate at which subsystems send messages are not dependent on one another, as they would be in a synchronous Pipeline architecture. For example, if the Keeping & Control subsystem encounters an issue that delays sending a movement command, the next cycle will still be provided with the most recent lane information from the Detection subsystem. This is critical in a real-time implementation such as an autonomous driving system, since the vehicle will be moving at high speeds and will need to make decisions based on environmental data in a manner that is as real-time as possible to avoid lane drifting or collisions.

By incorporating ROS into our system, we gain significant benefits for its design and operation. ROS enables us to break down different tasks into separate nodes, promoting a modular approach to development. This means we can focus on specific tasks without them getting tangled up. It also helps us keep a clear track of how data moves through our system, making it easier to manage and maintain.

Additionally, using ROS topics for communication allows us to interact with various hardware components for tasks like video input and vehicle control. This means we can adapt our system to different setups without having to make major changes to our codebase. This flexibility streamlines communication between our system and the environment, ensuring smoother operation of the lane detection and keeping process as the vehicle drives.

4.2. Evaluation

The system will be developed and evaluated for two phases of demonstration. The first phase will involve testing using a simulation environment, and the second will involve integration with a physical robot on a test roadway.

For the main course of the project, testing will be performed in a simulation environment using CARLA. CARLA is a driving environment simulation tool used for iterative, test-driven development of autonomous driving systems.[12] This testing method will give freedom of modifying the various parameters to be considered with respect to road conditions. Using the simulation will also allow for quick testing of features throughout development. When running simulations using CARLA, the distance from the center of the lane will be determined by the detection subsystem and adjustments to remain in the lane will be made by the K&C subsystem. The results of the adjustments will be observed and recorded to verify proper functioning of the system.

The secondary testing for the project will include running the system on the Hiwonder JetAuto driving on a test track. The camera sensors will be mounted to the JetAuto and interfacing will be developed for communicating movement commands. Testing will be completed in a similar way to the simulation phase by observing and recording the system's ability to maintain a sufficiently small distance from the center of the lane.

The evaluation of the individual Detection and K&C subsystems will be completed as described in sections 3.1.6 and 3.2.6 respectively. The overall evaluation of the system will involve a more high level look at the system's functionality.

5. SUMMARY

The primary objective of this project is to create a system for autonomous vehicles that uses computer vision and hardware interfacing to allow the vehicle to monitor the position of lanes on its road and maintain a steady position within its own lane as it drives. The work performed will be primarily software based, using a Hiwonder JetAuto robot car that will run three subsystems: lane detection, lane keeping and control, and movement.

Many existing products offer some sort of lane detection and following systems, and many consumer car manufacturers provide some kind of lane assistance as part of their vehicle's systems. Commercial applications of lane detection tend to be very good at recognizing lanes under ideal conditions, but struggle a lot when it comes to unfavourable lighting or weather conditions. Research wise, there have been significant improvements in lane detection over the last 5 years with new approaches such as row-wise classification, or parametric curve modeling that allow lane detection systems to run more complicated CNN's.

To address this problem, the project team will create a lane detection and keeping system. It will be composed of two parts: a lane detection subsystem and a lane keeping & control subsystem. The lane detection subsystem will focus on the implementation of a CNN that will analyse the video from a camera mounted on the robot to determine the center of the lane path to follow. The lane keeping and control subsystem will take the path of the center lane and with the use of a controller mark the path for the vehicle to take and execute it.

This project relates significantly to the degree of the students involved as each student is taking Software Engineering. The development of this system is entirely software based, and focuses on using many important software engineering concepts such as but not limited to project management, real-time systems, embedded systems, and machine learning.

The skills required to undertake this project can be met by all of the group members. For the lane detection, Curtis Davies has extensive experience working with machine learning and neural networks which is required to make such an undertaking. For integrations with the hardware and the underlying ROS operating system, Robert Simonescui has experience working with robotics systems as well as setting up and configuring ROS environments. For the lane keeping and control, while none of the group members have direct experience working with controller systems,

every member of the group is very good at learning and implementing new skills such that they will be able to learn the necessary components with relative ease.

In order to apply these skills, several methods learned during the group's studies at Carleton will be used. For starters, the microservice software architecture will be used as the basis for the architecture of the system. This architecture was taught in SYSC 4120, and separates each part of the program into isolated components that communicate with each other. For the lane detection, skills learned from SYSC 4415(Introduction to Machine Learning) and COMP 4102(Computer Vision) will be used to properly determine the location of the lines as well as evaluate the center of each lane. For dealing with the input and output from the hardware, techniques used from SYSC 3310 will be used. Lastly, the project management skills taught in SYSC 4106 will be used to ensure that the project goes smoothly.

A timeline for the project was developed to manage the flow of the project. This started off with the creation of a Work Structure Breakdown outlining the required tasks to accomplish for each section of the project. Following the breakdown, each task was given responsibility from the appropriate members of the group as primary or secondary responsibility. Then, dependencies between these tasks were created and a Gantt chart was built outlining the project's timeline. These can be found at Figures 4, 5, 6 and Table 1.

Several risks to the project were also identified. The largest risk to the project was identified as the neural network being too complex for what the group is trying to do. To mitigate this, a backup plan has been decided to use an algorithm approach instead. Another large risk is the hardware not being powerful enough on the robot car. A backup plan has been identified to use a distributed architecture where neural network processing is done on a remote computer in case this problem is encountered. Scope creep has also been identified as a risk, but due to how unlikely it is to be encountered and how little the impact will be on the project due to an MVP approach, no additional mitigation strategies were identified. Lastly, integration challenges were identified as being a relatively low risk but in order to limit any impact, a highly organised documentation strategy will be implemented.

In terms of special equipment, four components were identified. For starters, a robot car is required to test the implementation of the project's software. This is available from the university. Cameras are also required as a means of gathering information from the lanes, and have been previously purchased by past year's projects and do not need to be re-bought. A simulation box is also required as a means of testing our implementations before we put it on the hardware, which has also been

previously bought by the university and is currently available for use. Lastly, camera lenses need to be acquired as the cameras from previous years do not have lenses suitable for our project's needs.

REFERENCES

[1] "Subaru EyeSight - Subaru Technology - Subaru Canada," www.subaru.ca. https://www.subaru.ca/WebPage.aspx?WebSiteID=282&WebPageID=18112

[2] Honda Automobiles. "Honda Sensing." Honda Automobiles, 2018, automobiles.honda.com/sensing. https://automobiles.honda.com/sensing

[3] Yoo, Seungwoo, et al. "End-To-End Lane Marker Detection via Row-Wise Classification." ArXiv (Cornell University), 6 May 2020, https://doi.org/10.48550/arxiv.2005.08630. Accessed 1 May 2023.

[4] Feng, Zhengyang, et al. "Rethinking Efficient Lane Detection via Curve Modeling." ArXiv.org, 21 May 2023, arxiv.org/abs/2203.02431. Accessed 5 Oct. 2023.

[5] NVIDIA Corp. "Jetson - Embedded AI Computing Platform." NVIDIA Developer, 30 Oct. 2015, https://developer.nvidia.com/embedded-computing.

[6] https://www.cim.mcgill.ca/~ialab/ev/Intro_control1.pdf.

[7] W. Van Gansbeke, B. De Brabandere, D. Neven, M. Proesmans, and L. Van Gool, "End-to-end lane detection through differentiable least-squares fitting," *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Oct. 2019. doi:10.1109/iccvw.2019.00119

[8] C. Bisulca, P. C. Nascimbene, L. Elkin, and D. A. Grimaldi, "Variation in the deterioration of fossil resins and implications for the conservation of fossils in Amber," *American Museum Novitates*, vol. 3734, no. 3734, pp. 1–19, Feb. 2012. doi:10.1206/3734.2

[9] TimmyK "PID VaryingP" *Wikimedia Commons*, 7 Nov. 2014, https://commons.wikimedia.org/wiki/File:PID_varyingP.jpg

[10] M. Behrendt, "A basic working principle of Model Predictive Control." *Wikimedia Commons*, 2 Oct. 2009, https://commons.wikimedia.org/wiki/File:MPC_scheme_basic.svg

[11] "Robot operating system," ROS, https://ros.org/ (accessed Oct. 30, 2023).

[12] A. Dosovitskiy, et al. "CARLA: An Open Urban Driving Simulator," *arXiv*, 10 Nov. 2017, arXiv:1711.03938v1

[13] P. Woolf, et al. "P, I, D, PI, PD, and PID control" in *Chemical Process Dynamics and Controls*: University of Michigan Engineering Controls Group, 2009, ch.9.2. Accessed: October, 26, 2023. Available: https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Chemical_Process_Dynamics_and_Controls_(Woolf)

[14] P. Woolf, et al. "PID Downsides and Solutions" in *Chemical Process Dynamics and Controls*: University of Michigan Engineering Controls Group, 2009, ch.9.6. Accessed: October, 26, 2023. Available: https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Chemical_Process_Dynamics_and_Controls_(Woolf)/09%3A_Proportional-Integral-Derivative_(PID)_Control/9.06%3A_PID_Downsides_and_Solutions

[15] M. Raginsky, D. Liberzon. *ECE 486 Control Systems*. University of Illinois Urbana-Champaign, Illinois, US, 2019. Accessed: Oct. 26, 2023. Available: https://courses.engr.illinois.edu/ece486/fa2019/handbook/lec06.html

[16] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—A survey," *Automatica*, vol. 25, no. 3, pp. 335–348, May 1989. doi:10.1016/0005-1098(89)90002-2

[17] DataGenetics. "Ackermann Steering," *DataGenetics*. Available: http://datagenetics.com/blog/december12016/index.html