Queen's University

Kingston, Ontario, Canada

Mounted Machine Learning Camera System for Object Detection and Track Tracing During Hyperloop Operations

Design Team Research Paper (European Hyperloop Week 2023)

Authors:
Kevin Liang, Harsh Kalyani, Noah Chung, Ryan Silverberg, Vince Song, Thomas Zamin

Contributors:
James Kilpatrick, Nicolas Odorico, Julia O'Neil, Noah Warren, Erin Zhang, Saad Alam, Kevin Gao, Arjun Guliya, Minxuan Luo, Aarav Shah, Owen Billones, Kishore Karan, Thomas Priftakis, Robert Zielinski, Charlie Bolt, Phoebe Chalmers, Luke Gaylor, Madison O'Brien, Ty Cymbalista

Publication Date:
June 2023

Publication Location:
https://queenshyperloop.ca

Word Count:
5938 words

Award:
Full-Scale Awards (Technical Aspects of Hyperloop Systems)

Statement of Originality and Contribution

Following professional engineering practice, we bear the burden of proof for original work. We confirm that this work is original, and sources are cited appropriately whenever used.

Members of this paper are from the Research & Development (AI & ML) team within Queen's Hyperloop: a group of undergraduate and graduate students dedicated to solving advanced Hyperloop problems with cutting-edge technology and custom equipment.

Name Full name of author/contributor	**Organization**Team or company of individual	**Content** Content contributed by author/contributor	**Write-up** Sections written by author/contributor	**Editing** Sections edited by author/contributor
Kevin Liang	Queen's Hyperloop Design Team	Description, Environment & Objectives, Abstract, Results and Discussion, Bibliography	Description, Environment & Objectives, Abstract, Results and Discussion, Bibliography	All
Noah Chung	Queen's Hyperloop Design Team	Environment & Objectives, Approach 1: Edge Impulse, Approach 2: Python Application with Dedicated Hardware, Approach 3: Python Application with Dedicated Hardware	Environment & Objectives, Approach 1: Edge Impulse, Approach 2: Python Application with Dedicated Hardware, Approach 3: Python Application with Dedicated Hardware	All
Harsh Kalyani	Queen's Hyperloop Design Team	Introduction, Approach 1: Edge Impulse, Approach 2: Python Application with Dedicated Hardware, Approach 3: Python Application with Dedicated Hardware, Results and Discussion	Introduction, Approach 1: Edge Impulse, Approach 2: Python Application with Dedicated Hardware, Approach 3: Python Application with Dedicated Hardware, Results and Discussion	All

Ryan Silverberg	Queen's Hyperloop Design Team	Approach 2: Python Application with Dedicated Hardware	Approach 2: Python Application with Dedicated Hardware	Approach 2: Python Application with Dedicated Hardware
Vince Song	Queen's Hyperloop Design Team	Results and Discussion	Results and Discussion	All
Thomas Zamin	Queen's Hyperloop Design Team	Results and Discussion	Results and Discussion	All

Abstract

Research Question

There are many safety factors to consider when designing a Hyperloop system. The main factors include debris on the track, cracks, and defects in the tunnel. To ensure that these factors are kept at bay, various sensors and equipment must be added. The additions of these sensors and equipment can be costly and occupy considerable space. The research question is: how can the main safety factors be covered without being too expensive or volume consuming?

Overview of Motivation

The motivation for this research stems from two key factors. To develop a more modern safety system for the Hyperloop pod and to ensure quality assurance throughout transit due to cases such as faults in systems, obstacles, and faulty sensors. These elements were vital to the research motivation, as these cases pose risks and potentially destruction of the pod in transit if not handled correctly. The camera developed from this research aids to prevent these issues.

Presentation of Results

The final product generated by research and execution through different attempts is a camera system capable of identifying objects and tracing Hyperloop tracks. The product includes mechanical and hardware designs that allow the system to be mounted on experimental Hyperloop pods, as well as software capable of working with physical components to meet MVP requirements. The product uses the YOLO algorithm and Python libraries, such as TensorFlow, to have the desired functionalities as outlined in the motivation. Several datasets were also generated and downloaded to train the machine-learning model, which allowed the final product to identify several common items as proof-of-concept.

The camera system has been designed to accommodate 3D printing as part of the manufacturing process and mostly uses off-the-shelf hardware components. In terms of software, the Python programming language was used to develop the various algorithms necessary to train and use machine-learning models.

Table of Contents

General

Description

Queen’s Hyperloop Design Team (QHDT) is a dynamic group of over 150 passionate students from Queen’s University, Kingston. Collaborating across diverse disciplines, the team shares a vision to change the future of transportation. Leveraging the immense talent within Queen’s University, our multi-disciplinary team is committed to pushing the boundaries of innovation and engineering excellence. This paper serves as a testament to the collective dedication and progress made towards making Hyperloop a reality. QHDT is also a partner and competitor of the Canadian Hyperloop Conference (now Hyperloop Global), another annual competition.

With the advancement of technologies in the software and computing industries, there is now an opportunity for the implementation of advanced computer vision systems to provide numerous advantages to student-designed Hyperloop pods. In particular, the ability to detect objects and trace rails have been enhanced with breakthroughs in machine learning models. While camera systems may not replace sensors completely at the current stage, the ability to visualize the surroundings of a Hyperloop pod in operation can benefit Hyperloop infrastructure and improve safety factors of pods. In general, such technologies can reduce accidents, provide developers with more data to work with, and enhance various systems of a Hyperloop pod. Applications and benefits of such systems can also be seen on existing train station cameras and rail sensors, as seen in Figure 1.

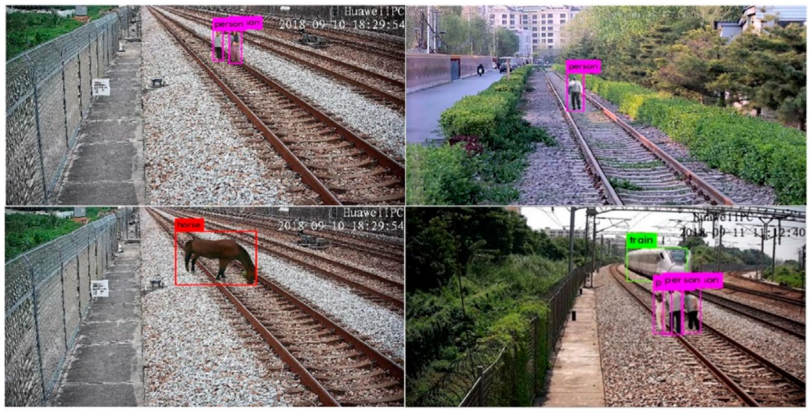


Figure 1: Image detection as seen on a camera pointed at train tracks.

As a team, Queen’s Hyperloop has been investigating the numerous applications of machine learning and machine vision models in Hyperloop systems. Specifically, the team has built and applied object detection and line tracking algorithms to improve Hyperloop pods, which are used as part of a physical mounted camera system that can be attached to Hyperloop pods rapidly [1]. This research project included both physical and software components that function as a system when attached to a chassis. While limited, the system also provides data and outputs that can be used to control other subsystems on an Hyperloop pod.

Environment & Objectives

Machine learning and software development requires a digital environment and relevant tools that enable the development of software applications. Hence, most of the work done for research in such areas was carried out on computers and software tools were used to complement the process. To create an efficient environment to reach goals and complete tasks, all code was written using Python-compatible Integrated Development Environments (IDEs) and shared using GitHub. This created a seamless and synchronous development process.

The MVP objective was to create a camera system powered by advanced ML/AI models to enhance the performance and safety of Hyperloop systems. The camera system was to incorporate an object detection AI model to accurately identify and localize objects within the video frame. The model aimed to analyze the visual content and identify defects, debris, and the future positions of the pod. The model was trained on a dataset that included various object types to allow for accurate object detection. Monitoring the condition of the track through using the model would help in detecting any structural damage, cracks, or other abnormalities that would affect the operation and safety of the system.

The long-term goal was to have an accurate and affordable functioning camera system that can pass all test cases to ensure a secure and reliable transportation system. The test cases will include uploading video feeds to make sure the object detection system can identify defects, debris, and the future position of the pod. The AI model was trained to detect relevant objects so that appropriate responses can be generated by the model for the Hyperloop control system. The future positions of the pod will be determined by detecting and identifying the tracks and their turns, as well as other information provided by various onboard sensors. The system must output the future positions in a way that can be used to adjust the pod.

Research

Introduction

Topic & Motivation

The advancement of transportation technology has always been at the forefront of human progress. With the emergence of the Hyperloop concept, a new era of high-speed, efficient, and sustainable transportation is on the horizon. However, as with any innovative system, ensuring safety and reliability are paramount concerns.

Motivated by the need for an advanced safety infrastructure and a robust quality assurance process, the team's research aims to explore solutions that mitigate risks and enhance both passenger and pod safety during Hyperloop operations. Utilizing a camera-based system will help to overcome the limitations of traditional sensor and equipment installations, which can be costly and space-consuming.

The camera system would be attached to individual pods and provide data through machine-learning tools. The features of this system will allow for prediction of turning angles, identifying weaknesses or damages of the infrastructure, and detecting objects or humans that may cause harm to human lives and Hyperloop pods. These features aim to enhance the safety of passengers on Hyperloop pods while improving operational efficiency of onboard equipment.

Background Information

In this research paper, QHDT addresses the challenge of developing a cost-effective and space-efficient safety system for the Hyperloop Pod. The objective is to tackle key safety factors, including debris on the track, cracks, defects in the tunnel, and predict the future position of the pod for necessary adjustments. By focusing on these factors, QHDT aims to create a modern safety system that enhances the overall quality assurance of Hyperloop transit.

The core question guiding this research is: How can the main safety factors associated with Hyperloop systems be effectively addressed without incurring exorbitant costs or compromising space within the Hyperloop pod?

To ensure efficient collaboration and maintain consistency among team members, all code development was built using Python-compatible IDEs and shared through GitHub. This approach facilitates seamless code readability, ease of modification, and synchronization among team members. OpenCV and Python code were used to test and train the AI model. OpenCV is a computer vision and machine learning software library that provides various tools and algorithms for video and image processing [2]. Video processing is mainly used for filtering, thresholding, and transformation. However, the main video processing feature used in the project is feature detection and extraction. OpenCV includes functions for detecting edges corners and blob in a video as well as being able to track the detected item through multiple frames of video. Next, this project took advantage of the machine learning algorithms offered by OpenCV. OpenCV includes algorithms for clustering, dimensionality reduction, and regression analysis. These algorithms can be used to group similar objects together, reduce the number of features in a dataset, or predict the value of a continuous variable based on other factors and patterns.

Testing of the AI model entails using procedurally generated environments and existing videos to act as proof-of-concept test environments. The procedurally generated environments create a video where desired objects for detection will pop up in the video at random. Once the AI is trained to detect these desired objects, the generated environments will be swapped with existing videos to act as a real-world environment. The existing videos will fine tune the training of the AI model as factors that may be non-existent in the generated environment. These factors include false identification (identifying objects that look like the desired objects but are not), lighting changes, and cluttered or textured backgrounds.

Methodology

The final solution was generated using three different approaches to the problem, each using different software and hardware designs to attempt as many methods as possible. Four groups of students were tasked to handle the approaches and try for a final design solution. The methodologies and results of these groups were classified into the three approaches as shown below. Note that all approaches occurred roughly around the same period, where limited communication exists between the groups.

Once complete, the approaches were then evaluated relative to the MVP goals to generate a final solution from their design choices. This process allowed for multiple software practices and hardware components to be put through the trial, which improved the overall confidence of the team when moving forward to the final solution. Moreover, the diverse approaches provided insights that would otherwise be lost for the problem at hand, which could be explored later in other research projects. The methodology used in this research project is like forming study groups to investigate certain topics, where multiple groups were tasked to solve an open-ended problem.

Approach 1: Edge Impulse

The first approach for the system implementation consisted of three subsystems: the sensor infrastructure, the embedded system, and the user interface. The sensor infrastructure served as a protective case for the embedded system that housed the camera module, ensuring the safety and durability of the system's components. Meanwhile, the embedded system comprised of the BalenaFin, a Raspberry Pi camera module, and a Raspberry Pi microcontroller. Note that the case can be 3D printed to ease the manufacturing process. All parts of this design are listed below in Table 1 and are displayed as an exploded view in Figure 2 and as a complete system in Figure 3.

Table 1: All materials listed for this embedded system.

Item No.	**Part**	**Description**	**Quantity**
1	BalenaFin	Carrier board for the Raspberry Pi microcontroller (9 x 9 x 1.5cm)	1
2	BalenaFin Case	Comes with BalenaFin to protect components (11.5 x 9.5 x 4cm)	1
3	Power Supply	12V 2A Power Supply Adapter	1
4	USB to USB-C Cable	To connect BalenaFin to display (32cm long)	1
5	RPI Camera Module	Raspberry Pi Camera Module V2 (2.4cm x 2.5cm)	1
6	Camera Case Front	Protective case for camera module	1
7	Camera Case Back	Protective case for camera module	1

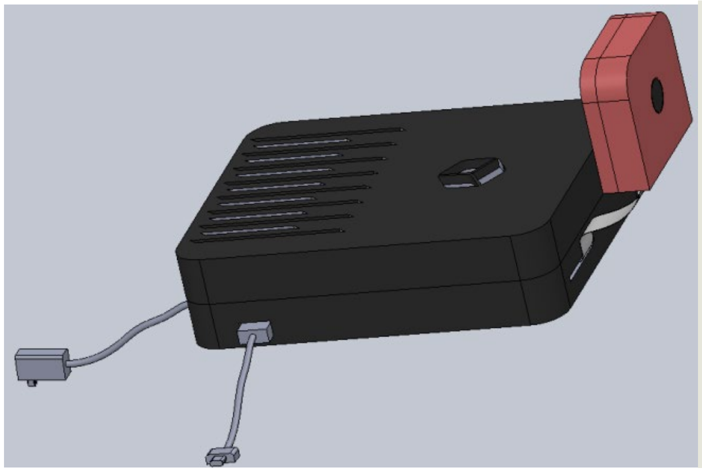


Figure 2: Isometric View of the Embedded System.

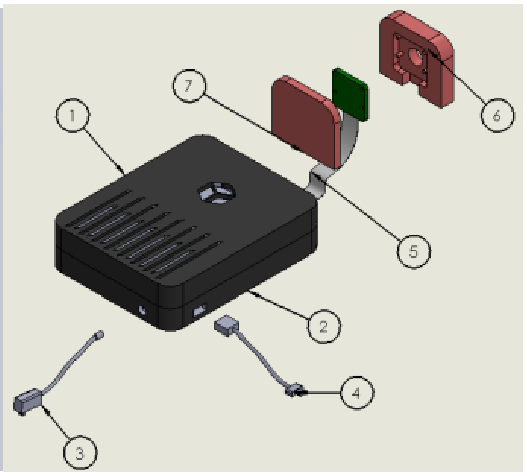


Figure 3: Exploded View of the Embedded System

The microcontroller was programmed with an object and lane detection machine learning model obtained from Edge Impulse, an open-source collection of pre-trained models [3]. The chosen model, called "Faster Objects, More Objects" (FOMO), is an "algorithm that enables microcontrollers to perform real-time object detection, tracking and counting" [4]. It utilized algorithms from OpenCV, Keras, and TensorFlow and was implemented in Python to identify and detect any potential hazards in real-time. OpenCV is an open-source library that includes several hundred computer vision algorithms and was used for performing different tasks such as object detection and video analysis [2]. The Keras library provides a Python interface for artificial neural networks as well as acts as an interface for the TensorFlow library [5]. TensorFlow is a machine learning platform that was crucial in the creation of the algorithm as it processes and loads data, trains models, and implements algorithms into a real system [6]. This algorithm was trained using the supervised learning method which uses a training set of images to teach the algorithm. Figure 4 below outlines the process for training the algorithm.

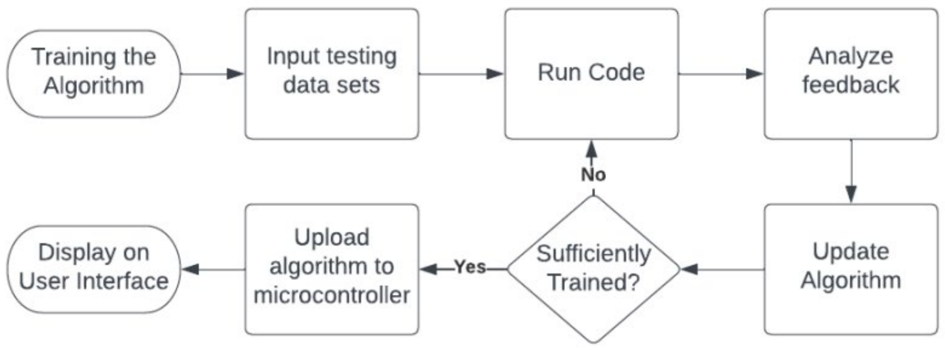


Figure 4: Training Process for the Algorithm

The data from the embedded system was transmitted to a user interface, which was as a platform to present the data to stakeholders. The platform was a website that was developed using HTML, CSS, and Flask, a Python web-development framework [7]. Wireframes and mockups of the UI/UX were created using Figma, a collaborative design space for creating high-fidelity designs of user interfaces. Once the UI/UX design was created using Figma, Tkinter-Designer was used to convert the Figma design into Python code with the Figma API. Figure 5 below outlines the methodology to obtain and display the data.

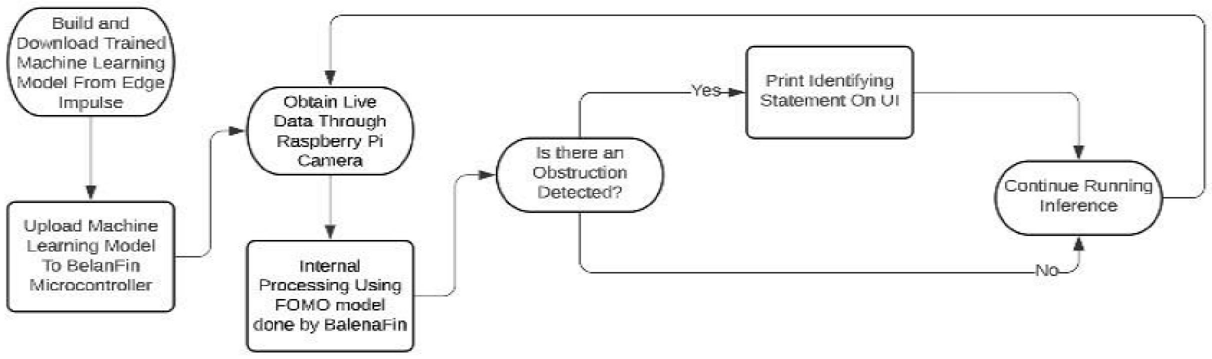

Build and Download Trained Machine Learning Model From Edge Impulse
Upload Machine Learning Model To BelanFin Microcontroller
Obtain Live Data Through Raspberry Pi Camera
Internal Processing Using FOMO model done by BalenaFin
Is there an Obstruction Detected?
Yes
Print Identifying Statement On UI
No
Continue Running Inference

Figure 5: Flow Chart for obtaining and displaying data.

However, despite these efforts, this approach faced several issues. The BalenaFin, which was initially selected as part of the embedded system, lacked the necessary modularity and was subsequently replaced with an alternative Raspberry Pi board. The Raspberry Pi boards also lacked sufficient computational power, leading to low frame rates and frequent screen freezes. They could not handle the required tasks of both computing and displaying information. Moreover, the pretrained FOMO model was incompatible with the BalenaFin board, further hindering implementation and undermining the system's feasibility. Given these limitations and issues with the selected components, an alternative approach had to be pursued to achieve the desired functionality and reliability in the system.

Approach 2: Python Application with Dedicated Hardware

The second approach utilized three subsystems: sensor infrastructure, machine learning model, and user interface.

Instead of using a camera as the main source of detection, this approach was attempted using LiDAR sensors. LiDAR is an acronym for light detection and ranging as it uses light pulses to gather three-dimensional information about the tunnel the hyperloop pod will travel in [8]. LiDAR will have a longer range than a camera and will have an easier time keeping up with 1000km/h speeds. However, LiDAR is not able to sense colour which may cause issues as small cracks and debris may not be sensed by lidar and would need to rely on color to be sensed. Figure 6 is a visual representation of a LiDAR scan provided below.

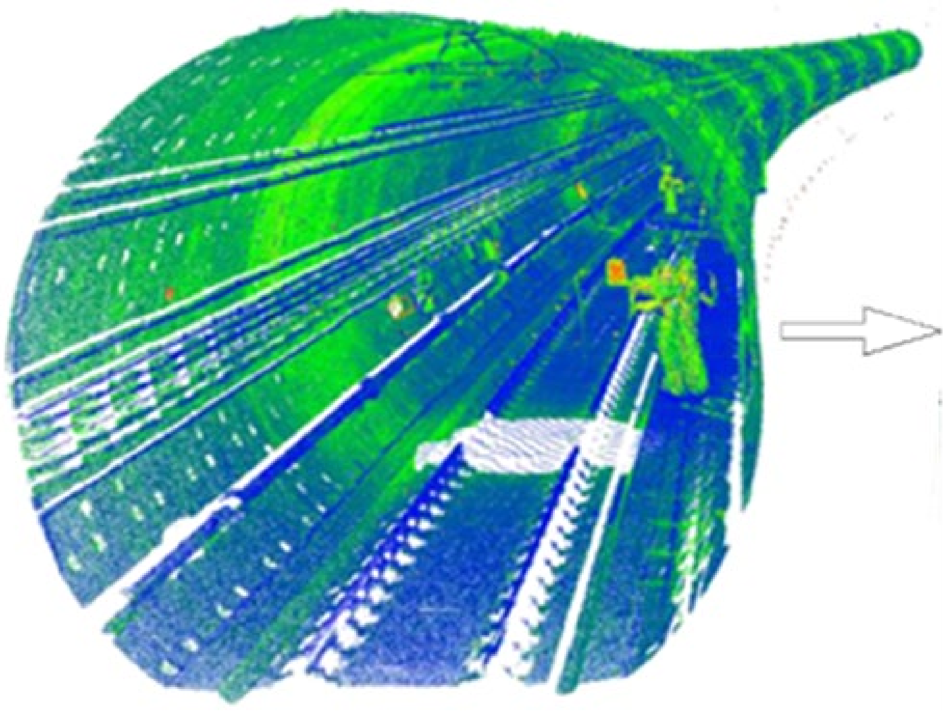


Figure 6: Visualization of the LiDAR scan

The machine learning model utilizes a Python library known as Scikit-learn, which provides access to a type of commonly used type of algorithm called support vector machines. They sort given data into classes, which are the categories given in the labels of the training dataset. For example. If the support vector machine is given images of fruit labelled either “apple” or “orange,” the support vector machine will sort a new, unlabeled picture, into these two categories. Then, the team used TensorFlow, a Python library developed by Google which simplifies the implementation of machine learning [6]. It has pre-existing machine learning models in the form of neural networks to help get projects started and is overall easier to implement compared to other similar frameworks. The neural network itself is a more complex version of a classification algorithm. It is made up of layers which “weight” features/attributes of the data (edges, size, etc.) and compute information from sensors in matrices. A loss function is also defined, which compares the network’s predictions on test data to reality and updates it by changing the weights and computations accordingly. These two parts together form the machine learning model.

For this approach, a custom-made dataset was implemented, and the pre-existing model was trained with it. When training a machine learning model to a new dataset, it is important to note that the more images there are the better the dataset is. Also, there should also be a balanced number of images for each object that a model is trying to classify. As the specificity of the model increases, the more specific the labels will be, but also the images as the model must be able to distinguish between two similar objects what this means is that as you want to distinguish between two similar objects the inputs need to reflect how they are different.

The machine learning model is integrated with the LiDAR sensors using a fully embedded system. An Orange Pi 3 LTS is sufficient for image processing and is used as the main CPU for the system. Once the image is processed and should an object be detected, it is sent first to the Hyperloop controller to minimize latency, then is shown to the user on a screen mounted to the Hyperloop pod. This entire system is powered by the Orange Pi since the power draw will be so low that it will not rely on power from any of the other Hyperloop subsystems. A flow chart depicting the pattern of the vision sensor system process described above can be seen below in Figure 7.

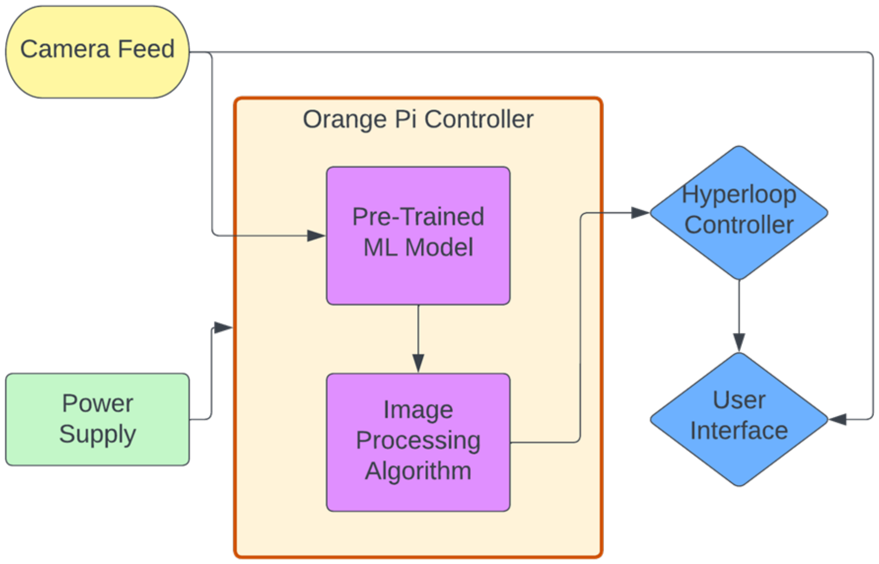
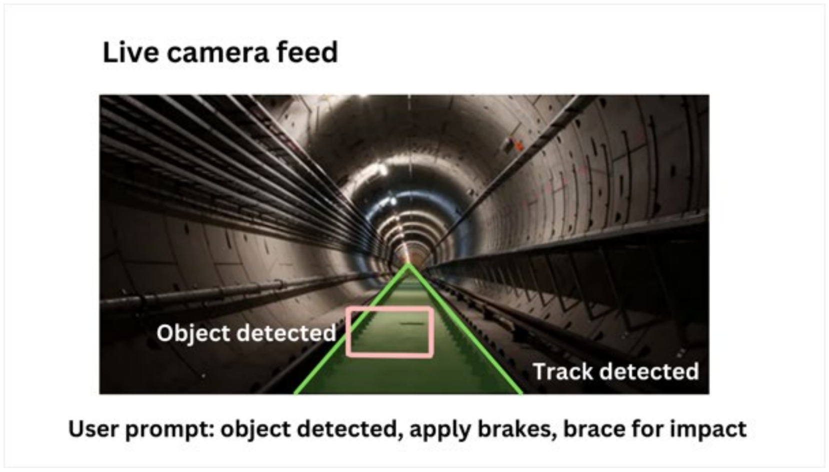


Figure 7: Flow Chart for this approach.

The user interface was created using Flask, a lightweight web framework that can be used to easily create web applications. The user interface uses green lines to follow and detect the tracks. Then, a pink rectangle is used to detect irregularities or damage that the pod may come across. Finally, if an irregularity is detected, the user will be notified and prompted with the appropriate action. Figure 8 shows the proposed user interface.



User prompt: object detected, apply brakes, brace for impact

Figure 8: The User Interface for this approach.

In conclusion, the second approach utilizes LiDAR sensor instead of cameras to detect tracks and irregularities in the hyperloop tunnel. The LiDAR sensors relay their imaging to the Orange Pi 3 LTS where the images will be processed, and the machine learning model will scan for tracks and objects. Finally, the outputs from the machine learning model will be displayed on the User Interface with a visual and verbal representation of the detected objects.

Approach 3: Python Application Without Dedicated Hardware

The last approach was a system composed of three subsystems: the hardware system, back-end software, and user interface. The hardware system was comprised of a chassis adjusted to a protective case for the camera. This was done to reduce the amount of 3D printing required as the battery pack and Raspberry Pi both already have exterior casings. This change also allowed for an ease of access to the camera for repairability. Figure 9 shows the different views of the camera chassis.

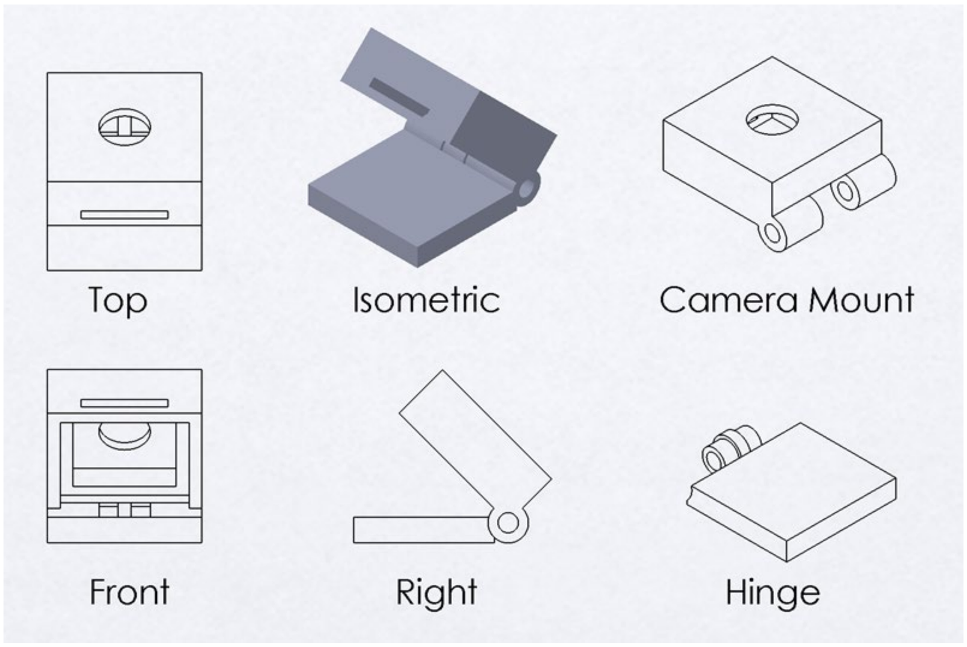


Figure 9: All view of the proposed chassis.

The backend software consists of two complementary algorithms, namely an object detection algorithm and a lane detection algorithm. To achieve real-time object detection, the team implemented the renowned "YOLO (You Only Look Once)" algorithm. YOLO excels in swiftly detecting objects within its field of view and accurately determining their relative positions, making it well-suited for deployment in a fast-moving pod [9]. Conversely, the lane detection algorithm incorporates a fusion of multiple algorithms to accomplish its intended purpose. These algorithms used Canny Edge detection and probabilistic Hough Transformations, which effectively converted detected edges into lines [10]. By employing these techniques, the algorithm proficiently identifies and tracks lane boundaries, contributing to enhanced navigation and path planning for the pod. After developing a machine learning model, it is imperative that it is trained using an appropriate dataset. Recognizing the significance of this process, the team adopted a pragmatic approach by utilizing readily available datasets from platforms like Open Images Dataset [11]. This platform offers a vast array of pre-existing datasets encompassing hundreds of objects, streamlining the training phase of the model. As a proof-of-concept, the model has been trained to identify specific items. (See Figure 10).

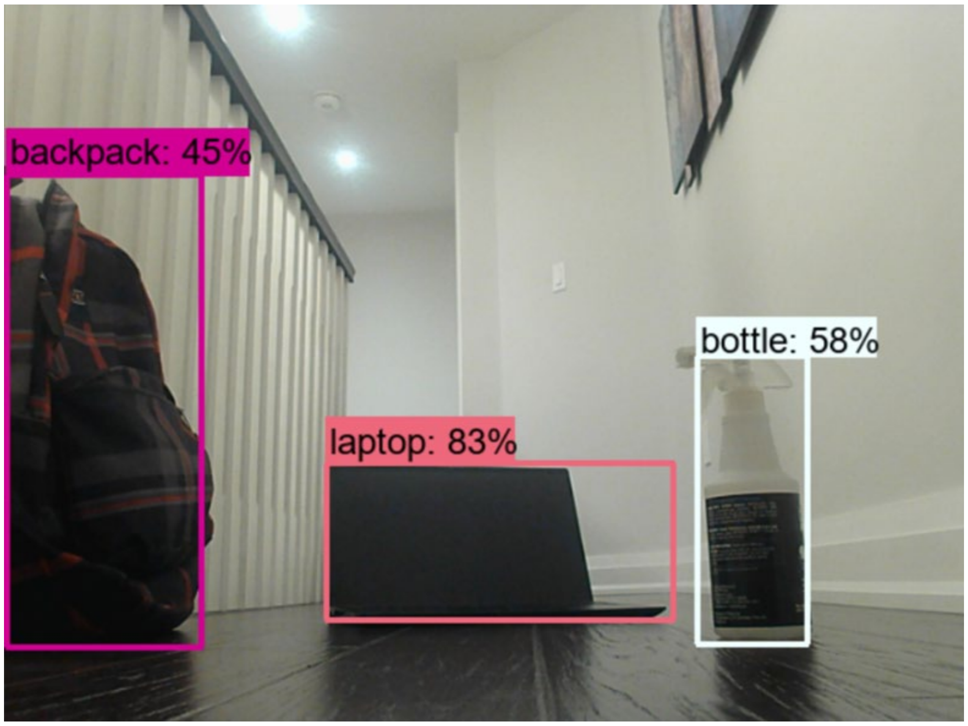


Figure 10: Trained model being able to detect objects.

The lane detection algorithm operates on a continuous stream of real-time video input, processing each frame individually to detect and track lane boundaries. The process involves a series of steps that enhance the image quality and isolate the lane lines for accurate identification. Initially, the oncoming video is divided into discrete frames. Each frame undergoes pre-processing, wherein the colours are transformed into grayscale and a blurring technique is applied to reduce noise and enhance edge detection accuracy. The Canny Edge detection algorithm is then employed to extract the edges from the pre-processed grayscale image. This conversion results in an image that highlights the prominent edges and lanes within the frame. To further refine the edges, a mask is applied to exclude regions that are less likely to contain lane markings. Subsequently, probabilistic Hough Transformations are used to identify line segments that constitute the lane, which are then averaged out into two separate lines representing the left and right lane boundaries. To provide visual feedback to the user, the identified lane lines are superimposed onto the original frame and are outputted to the user at the same frame rate as the original video, ensuring real-time visualization of the lane detection process. Figure 11 below outlines a flowchart for the lane detection algorithm.

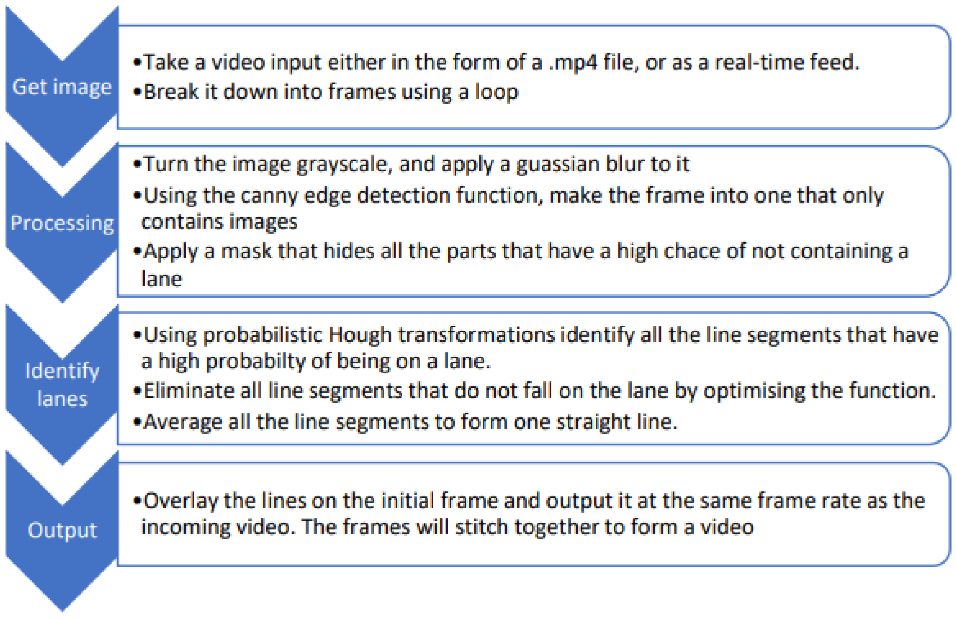

Get image
•Take a video input either in the form of a .mp4 file, or as a real-time feed.
•Break it down into frames using a loop
Processing
•Turn the image grayscale, and apply a guassian blur to it
•Using the canny edge detection function, make the frame into one that only contains images
•Apply a mask that hides all the parts that have a high chace of not containing a lane
Identify lanes
•Using probabilistic Hough transformations identify all the line segments that have a high probabilty of being on a lane.
•Eliminate all line segments that do not fall on the lane by optimising the function.
•Average all the line segments to form one straight line.
Output
•Overlay the lines on the initial frame and output it at the same frame rate as the incoming video. The frames will stitch together to form a video

Figure 11: Flowchart for lane detection code highlighting all the major steps taken by the code to detect a lane.

One of the primary advantages of this lane detection algorithm lies in its simplicity and versatility. It offers a straightforward implementation that can be easily understood and customized by users. Furthermore, its versatility enables optimization for various use cases, such as low-light conditions, high-speed scenarios, or complex tracks with numerous turns. Figure 12 below shows images that demonstrate the lane detection algorithm in action.

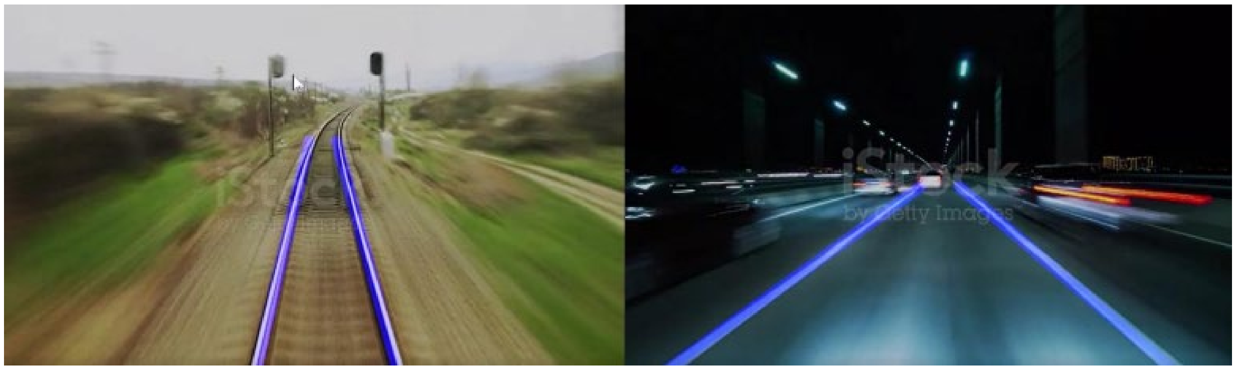


Figure 12: Images to demonstrate lane detection.

The front-end of the software is a user-friendly interface designed to provide real-time updates during the operation of the Hyperloop pod. It consists of two prominent panels, each serving a distinct purpose. The first panel displays a live video feed captured by the pod's camera, while the second panel presents updates on any objects detected within the camera feed. Below these panels, a status bar is located, which relays alert and warning messages if any abnormalities are detected during the pod's runtime. When the object detection algorithm identifies an object within the video feed, in addition to displaying the relevant information in its designated panel, the status bar undergoes a visual change from a neutral state to display an alert message throughout the duration of the object detection process. This alert message serves as an indicator to the user that an object has been detected and requires attention. If the object being detected is located on the track and poses a potential collision risk with the pod, the status bar displays a waring message specifically indicating that the object is obstructing the pod’s path. In scenarios where the lane detection algorithm detects that the pod is deviating from the track, the status bar presents a warning message stating that the pod is deviating from its intended path. This warning message serves as a prompt for the user to take corrective measures to ensure the pod remains on the designated track. Additionally, if the lane detection algorithm fails to detect the lane lines altogether, the status bar switches to display a warning message indicating that the pod's path is blocked. This message signals that the algorithm is unable to accurately identify the lane boundaries, potentially resulting in unsafe navigation conditions. In cases where both the object detection and lane detection algorithms trigger warning messages simultaneously, priority is given to the lane detection warning message. This decision assumes that the object detection panel will already highlight the presence of any significant objects that the pod may encounter, emphasizing the importance of displaying the lane detection warning message to draw attention to potential deviations from the designated track. Overall, this user-friendly interface offers a comprehensive and intuitive means of monitoring the pod's operation, providing real-time video feedback, object detection updates, and crucial warning messages to ensure the safety and efficient functioning of the Hyperloop pod. Figure 13 shows the full display of the UI for lane detection. Figure 14 shows the display of notifications for the user. Figure 15 is a flowchart for the lane detection code.

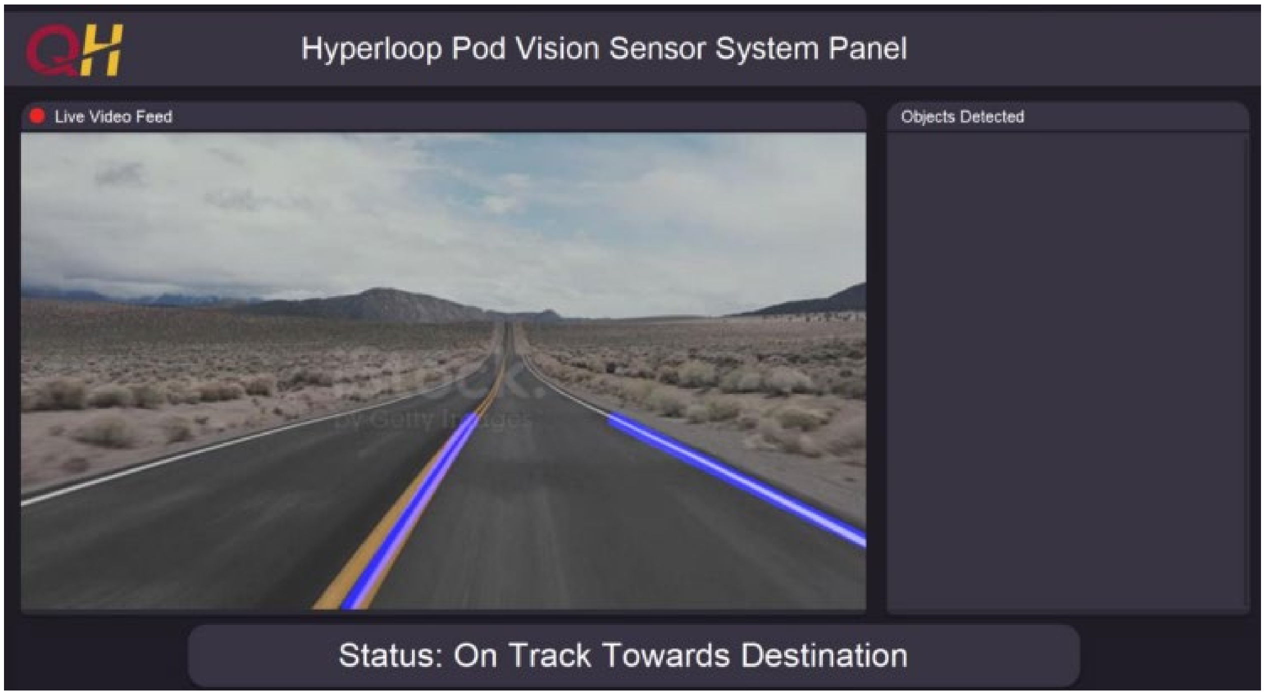
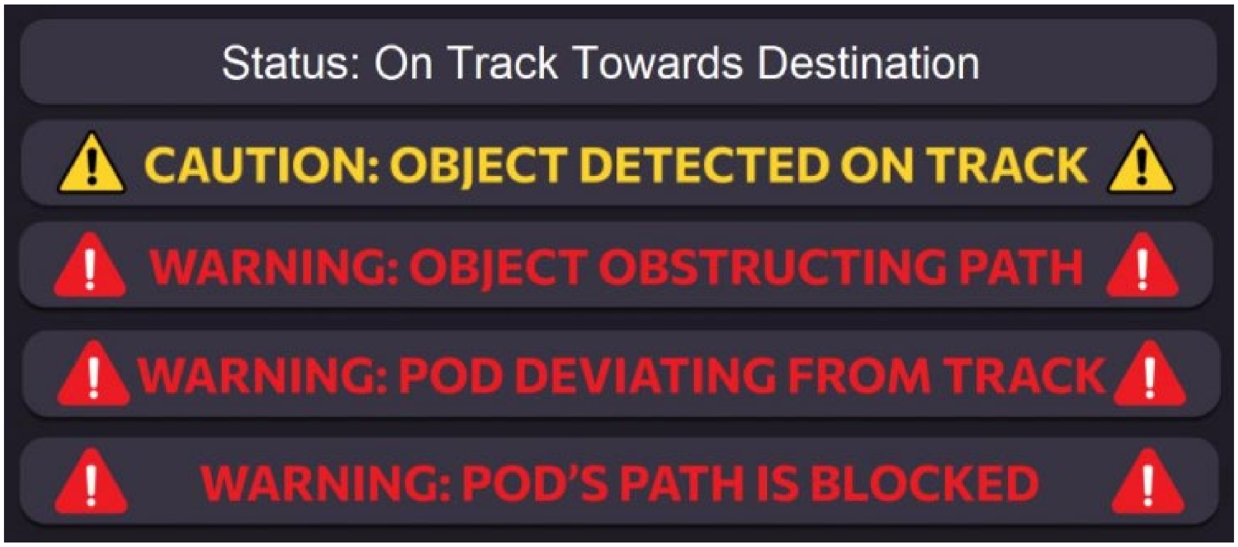

Hyperloop Pod Vision Sensor System Panel
Live Video Feed
Objects Detected
Status: On Track Towards Destination

Figure 13: Fullscreen display of the UI running a lane detection test video.


Status: On Track Towards Destination

CAUTION: OBJECT DETECTED ON TRACK

WARNING: OBJECT OBSTRUCTING PATH

WARNING: POD DEVIATING FROM TRACK

WARNING: POD'S PATH IS BLOCKED

Figure 14: Display of neutral status message (Top), object detection alert (Second Top), object detection warning message (Middle), pod path deviation warning message (Second Bottom), pod pathway being blocked warning message (Bottom).

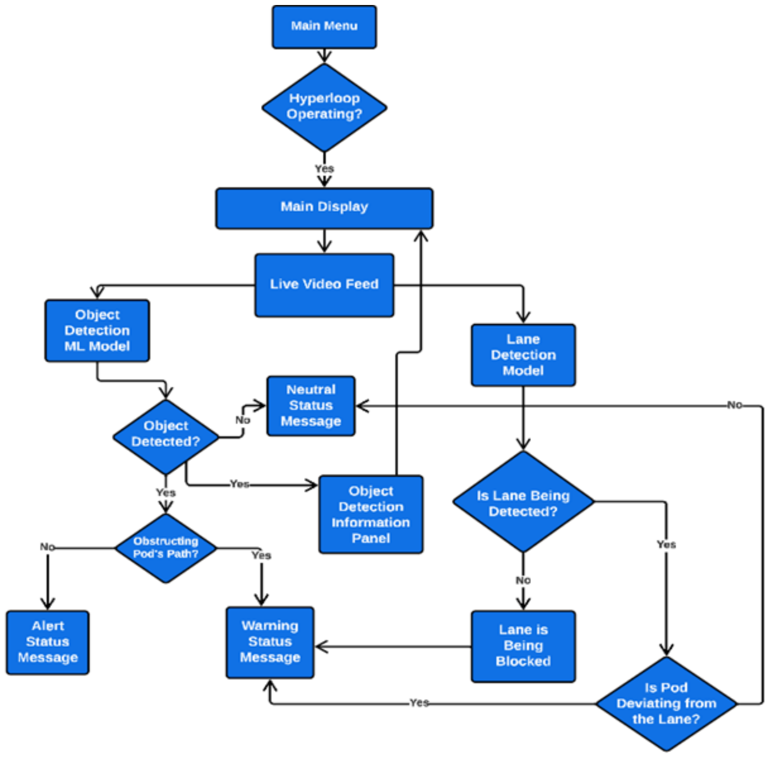


Figure 15: Flow chart of the lane detection algorithm.

Finally, the integration between the camera and the computer running the UI software is linked using a P2P (Peer-to-Peer) Connection. This UI software, in the form of an executable, connects to the vision sensors computer and send all relevant information to the UI. This means that the

UI runs independently from the machine. With this design the UI can also be stored on a USB key and transported to any relevant operator with ease. The P2P connection will be done via an ethernet cable connecting to the machine's onboard computer. The flowchart below shows the process of the system in Figure 16.

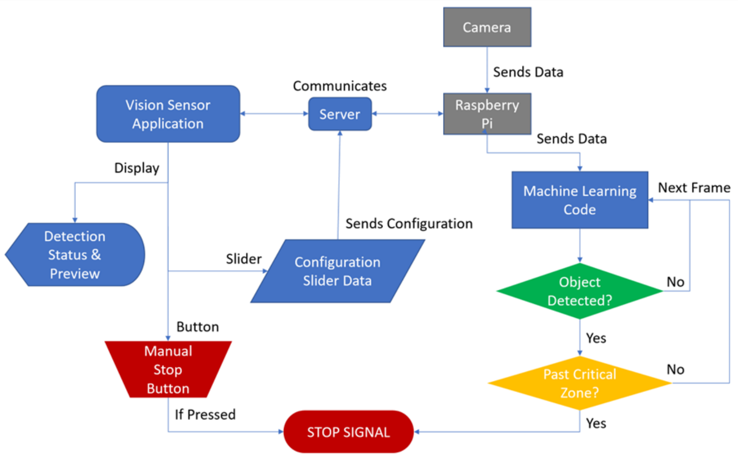


Figure 16: Flowchart of the P2P connection approach.

Results and Discussion

Evaluation of Approaches

Valuable lessons were learned from the three approaches to the problem, especially regarding the choices of components and software practices. It is apparent that certain methods worked effectively individually but had issues when integrating with other components. Conversely, certain design choices worked well when combined as a package but are ineffective in terms of performance or simply cannot meet the minimum viable product. To create the final product and ensure that the solution meets all functional requirements, all three approaches were first given a performance score using an evaluation matrix. These scores would then be used to select certain design choices that hold high influence for higher scores in relevant categories, which would then be used to piece together the final product.

The effectiveness of each approach was evaluated using the following evaluation with criteria listed and explained in Table 2 to justify the given scores.

Table 2: Criteria to evaluate the solutions.

Score	**1**	**2**	**3**	**4**	**5**
Safety	Physical design is	Physical design is	Physical design is	Physical design is safe;	Physical design is safe;

	not safe. users' safety is in danger.	somewhat safe; however, precautions need to be taken while using the device.	relatively safe, with a slight risk to user safety.	however, improvements can be made to decrease the user's risk of danger.	there is no risk to user safety
Appearance	Physical design does not exist. Missing components / no protective casing. UI/UX is very difficult to use and very unaesthetic.	Physical design is dull and bulky. None of the components are enclosed in a protective casing. Design is not aesthetic nor uniform. UI/UX is complicated and not aesthetic	Physical design is bulky and rough. Most components are not enclosed in a protective casing. Design is not very aesthetic and uniform. UI/UX is somewhat complicated but aesthetic	Physical design is somewhat sleek. Most of the components are enclosed in a protective casing. Design is somewhat aesthetic and uniform. UI/UX is somewhat simple and aesthetic	Physical design is sleek. Each component is enclosed in a protective casing. Design is aesthetic and uniform. UI/UX is simple and aesthetic
Functionally	UI/UX does not function / exist. Hardware component is not safe. Design is not compact or easy to apply	UI/UX is difficult to use. Hardware component is not safe. Design is not compact and is difficult to apply.	UI/UX is somewhat difficult to use. Hardware component is mostly safe. Design is compact but difficult to apply.	UI/UX is somewhat easy to use. Hardware component is safe. Design is somewhat compact and easy to apply.	UI/UX is easy to use. Hardware component is safe. Design is compact and easy to apply

Requirements	Does not detect objects. No UI/UX component. The model does not work/exist	Detects objects up to 10m ahead or less and 1m to the side or less with some errors. UI/UX does not produce warning when object is approaching. Machine Learning is not used at all	Detects objects up to 10m ahead or less and 1m or less to the side without errors. UI/UX produces warning when object is approaching with lots of errors. Minimal Machine Learning aspects	Detects objects up to 20m ahead and 2m to the side with some errors. UI/UX produces warning when object is approaching with some errors. Aspects of Machine Learning is implemented.	Detects objects up to 20m ahead and 2m to the side without errors. UI/UX produces warning when object is approaching without errors. Full Machine Learning Model is used
Compatibility	None of the subsystems are compatible with the QHDT's design.	Only one subsystems of the prototype are compatible with the QHDT's design.	Two subsystems of the prototype are compatible with the QHDT's design.	All three subsystems of the prototype are almost compatible with the QHDT's design.	All three subsystems of the prototype are compatible with the QHDT's design.

Approach 1: Edge Impulse

For the first approach, the following scores were given for each category based on the prototype's performance, as shown in Table 3.

Table 3: The evaluation score of the first approach.

Criteria	**Safety**	**Appearance**	**Functionality**	**Requirements**	**Compatibility**
Score	4/5	5/5	3/5	3/5	3/5

The first approach saw the creation of comprehensive mechanical and hardware designs that would be easily manufactured using 3D printers. However, the FOMO model proved to be inferior to YOLO and Edge Impulse was problematic during integration. The prototype provided great insights into the requirements of a successful and robust machine-learning model, which allowed for better understanding of training and testing processes of machine vision. Hence, the model was scored high in terms of physical design but low on other categories, resulting in its mechanical and hardware designs being implemented in the final product.

Approach 2: Python Application with Dedicated Hardware

For the second approach, certain information was successfully passed from the first approach, leading to a better prototype. With knowledge of the first model, a better model was created from scratch with each subsystem being much more compatible with each other. Although this led to better performance on the microcontroller, it came with hindered accuracy causing lower appearance and requirement scores on the evaluation matrix shown in Table 4. This model could have been implemented into the design, however, it was ultimately abandoned and reiterated for a much more accurate and effective model.

Table 4: Evaluation rubric for the second approach.

Criteria	**Safety**	**Appearance**	**Functionality**	**Requirements**	**Compatibility**
Score	4/5	3/5	4/5	2/5	5/5

Approach 3: Python Application without Dedicated Hardware

For the third approach, an effective YOLO algorithm was implemented to detect objects in real time with minimum delays. In addition to object detection, lane detection was made possible using Canny Edge and Hough Transformation. A proper UI was also designed and successfully implemented, making this prototype the strongest in terms of software performance. Beyond the machine-learning models and UI, the software provides data storage capabilities and enables the Hyperloop system to react to different events. The implementation of this advanced response framework allows for easier integration of the camera system with existing control systems.

Table 5: Evaluation rubric of the third approach.

Criteria	**Safety**	**Appearance**	**Functionality**	**Requirements**	**Compatibility**
Score	4/5	3/5	5/5	5/5	5/5

However, the prototype does not have a physical design that is better than the approach 1 prototype. It does have superior software capabilities and would theoretically be easier to integrate with most existing hardware components. In the end, the software components of this prototype were selected to move forward to be integrated into the final product.

Final Product

The integration of the third approach for its software-based object and lane detection along with the first approach for its mechanical and hardware components resulted in the development of the final product for real-time monitoring and safety enhancement in Hyperloop pods.

Approach three focused on the software aspect of the system, incorporating algorithms for object and lane detection. The object detection algorithm utilized the YOLO algorithm, renowned for its ability to swiftly detect objects within its field of view. This algorithm was trained using datasets from platforms like Open Images Dataset, streamlining the training phase and enabling the identification of specific objects. The lane detection algorithm incorporated a fusion of algorithms, including Canny Edge detection and probabilistic Hough Transformations. This algorithm effectively detected and tracked lane boundaries, contributing to enhanced navigation and path planning for the Hyperloop pod.

Approach one focused on the mechanical and hardware elements of the system. It included the sensor infrastructure and the embedded system. The sensor infrastructure served as a protective case for the embedded system, ensuring the safety and durability of the camera module. The embedded system consisted of a BalenaFin microcontroller, a Raspberry Pi camera module. Although the BalenaFin had issues with implementation in the first approach, it was still considered in the final product due to its higher processing power and extra memory storage over a conventional Raspberry Pi.

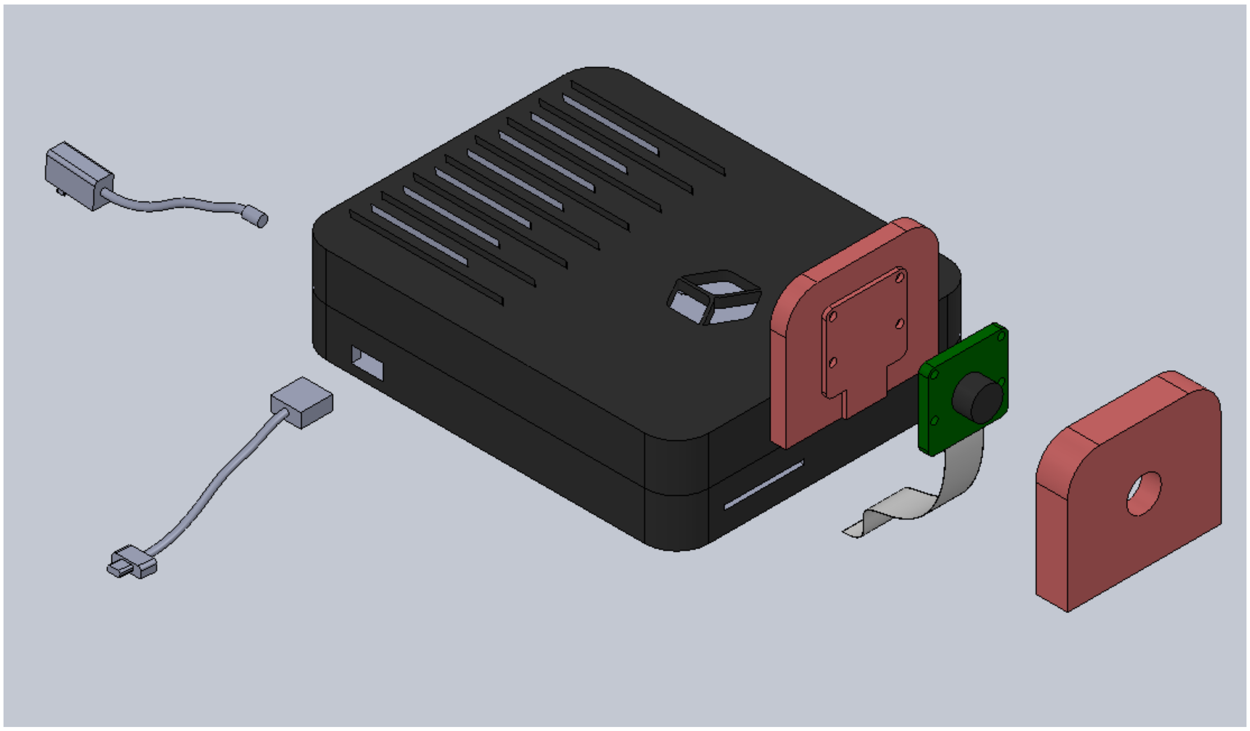
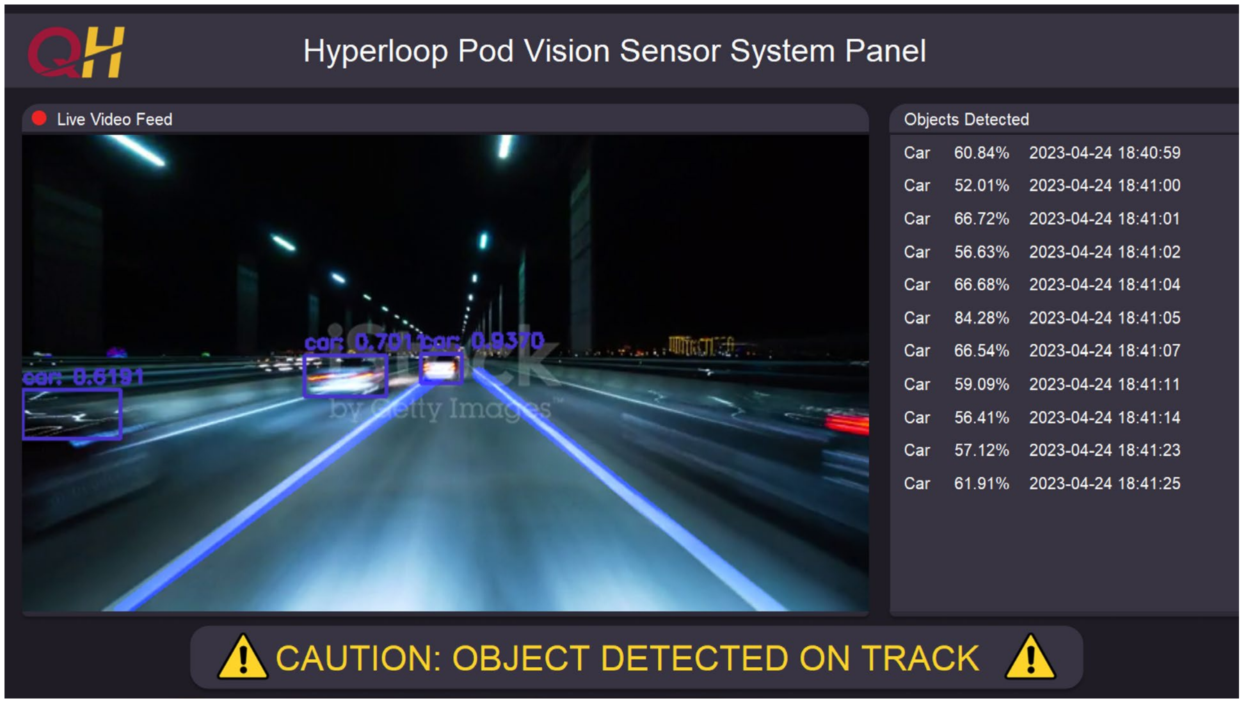


Figure 17: Exploded view of the final system.

The integration of these approaches resulted in a powerful system that combines the capabilities of real-time object and lane detection with potentially robust mechanical and hardware components. The final product provides a user-friendly interface for monitoring the operation of Hyperloop pods in real-time. The interface displays a live video feed captured by the pod's camera and provides updates on any objects detected in the camera feed. It also includes a status bar that relays alert and warning messages, indicating potential obstacles or deviation from the designated track. The combination of these approaches provided an effective solution for object detection, lane detection, and user interface, ultimately improving the safety and efficiency of Hyperloop transportation.

To use the final product on real Hyperloop pods, users can apply the same techniques to train the model to identify new desirable objects and tracks. The physical system can then be mounted on a pod and interfaced with existing control systems, which will work with the machine-learning software to operate during transit. Ideally, the model would be trained to identify common objects and issues with Hyperloop systems, so that the pod can pick up their presence during operation and act immediately if necessary.


Hyperloop Pod Vision Sensor System Panel
Live Video Feed
car: 0.6191
car: 0.701
car: 0.9370
Objects Detected
CAUTION: OBJECT DETECTED ON TRACK

Car	60.84%	2023-04-24 18:40:59
Car	52.01%	2023-04-24 18:41:00
Car	66.72%	2023-04-24 18:41:01
Car	56.63%	2023-04-24 18:41:02
Car	66.68%	2023-04-24 18:41:04
Car	84.28%	2023-04-24 18:41:05
Car	66.54%	2023-04-24 18:41:07
Car	59.09%	2023-04-24 18:41:11
Car	56.41%	2023-04-24 18:41:14
Car	57.12%	2023-04-24 18:41:23
Car	61.91%	2023-04-24 18:41:25

Figure 18: UI for the final system showing the live video feed, objects detected, and any notifications.

Results Discussion

By implementing a fully functional object and lane detection algorithm into an Hyperloop pod, a safer environment is created for the user as well as a more reliable system. For further improvement of the current design, it would be ideal to have focused on integration from the start as it has caused problems with multiple approaches from the start as well as ensuring a smooth process between subsystems. There should also be a stronger focus on the type of hardware that is used as there have been some framerate and processing issues with previous approaches. Lastly, it is recommended to retrain the model on better datasets to reflect new environments, as different Hyperloop systems may include different features that would require the model to adapt and conquer.

As stated earlier, the final design also does not have a reliable hardware system that allows for the software functions to run optimally. The core problem is due to hardware integration issues and performance, where chipboards cannot run the software to display information efficiently. However, the software on its own can still perform on a computer using its webcam, which provides proof of its functionality and effectiveness if an ideal hardware system is used. In the future, a logical solution would be to combine hardware systems together to strengthen their processing power, or simply find more powerful boards on the market. These solutions would still sacrifice other aspects of the design, such as increasing space usage or costs. Lastly, a critical assumption was made that the camera system and algorithm would be powerful enough to function normally during high-speed conditions. This assumption remains to be tested as there are no ways for the team to conduct a test with a fast vehicle easily.

The benefits of the systems are also to be seen. The main idea behind the various features of the MVP is to allow more safety factors to be added to the pod instead of the infrastructure, which typically would be the most expensive part of an Hyperloop project. By having the pod as a more autonomous and effective maintenance or error forecasting tool, companies can save valuable money on construction as failure detection tools are localized within the pods. Other functions like lane detection can also help onboard control systems to better navigate through turns, which can allow pods to run more autonomously and fare better against uncertainty.

When investigating similar topics, it is important to dive deeper into applications of machine-learning in Hyperloop, as the technology behind it has been improving drastically over the past few years. The costs of implementation have also decreased and could be widely used in many ways beyond data science, which is critical for costly projects like Hyperloop systems.

Bibliography

[1] Queen's Hyperloop Design Team, "769 - Hyperloop Machine Vision Sensor System Project Proposal," [Online]. Available: https://onq.queensu.ca/d2l/le/content/675364/viewContent/4440713/View. [Accessed 2023 January 10].

[2] OpenCV, "Introduction - Open Source Computer Vision," [Online]. Available: https://docs.opencv.org/4.x/d1/dfb/intro.html. [Accessed February 2023].

[3] "Edge Impulse," [Online]. Available: https://www.edgeimpulse.com/. [Accessed 3 March 2023].

[4] J. Davis, "Edge Impulse offers up FOMO algorithm, enabling object detection for microcontrollers," 18 April 2022. [Online]. Available: https://www.edgeir.com/edge-impulse-offers-up-fomo-algorithm-enabling-object-detection-for-microcontrollers-20220418. [Accessed May 2023].

[5] "Keras - Get Started," [Online]. Available: https://keras.io/. [Accessed February 2023].

[6] "TensorFlow," [Online]. Available: https://www.tensorflow.org/. [Accessed February 2023].

[7] A. Naskar, "Add HTML and CSS in Flask Web Application," Think Infi, [Online]. Available: https://thinkinfi.com/flask-adding-html-and-css/. [Accessed January 2023].

[8] NOAA, "What is lidar?," NOAA, [Online]. Available: https://oceanservice.noaa.gov/facts/lidar.html#:~:text=Lidar%2C%20which%20stands%20for%20Light,variable%20distances)%20to%20the%20Earth.. [Accessed June 2023].

[9] V. Sichkar, "Train YOLO for Object Detection with Custom Data," Udemy, June 2022. [Online]. Available: https://www.udemy.com/course/training-yolo-v3-for-objects-detection-with-custom-data/. [Accessed March 2023].

[10] M. Virgo, "Lane Detection with Deep Learning (Part 1)," Medium, 12 May 2017. [Online]. Available: https://towardsdatascience.com/lane-detection-with-deep-learning-part-1-9e096f3320b7. [Accessed March 2023].

[11] "Open images dataset V7," Google, [Online]. Available: https://storage.googleapis.com/openimages/web/visualizer/index.html. [Accessed March 2023].