2025 Science Fair Project Logbook - Aryan Sharma

September - November 2024

This is the start of my project!

I love playing ice hockey, and one of my favourite players is Johnny Gaudreau. I remember the day when the news about his death spread like wildfire. When I got the heartbreaking information, I was utterly shocked. After doing more research into the cause of his death, I learned that he and his brother Matthew died from a drunk driver. This really impacted me, and I wondered why there couldn't have been some type of camera similar to a speed camera that could detect these drunk drivers on the road. I wished there was a camera that could alert the Police about dangerous drivers on the road, so they didn't have to stay in areas and hope for one to come across them. This is where the inspiration for my project started.

My project question was: how can I design, program, and construct a scalable, autonomous, and efficient AI-powered device that can externally detect and prosecute drunk drivers in real-time?

This month was all about researching and creating objectives for my project. I first researched what was drunk driving to get a full in-depth understanding of the topic. Here are some of my findings:

Drunk driving, also called driving under the influence (DUI) or driving while intoxicated (DWI), is a criminal offense that involves operating a vehicle while one's senses are impaired by alcohol or drugs. This impairment effects the driver's reasoning and muscle coordination, affecting their ability to operate the vehicle safely.

- "On average in Alberta, one in five drivers involved in fatal collisions have been drinking before the collision. This compares to an average of about one in 20 drivers involved in injury collisions. As the severity of the collision increases, so does the likelihood the collision will involve a drinking driver." (MADD Alberta)
- "Countries like South Africa and Canada report relatively high percentages of fatal accidents involving alcohol with South Africa having 58% of its road fatalities stemming from the abuse of alcohol." (Sand Law North Dakota)

I also researched heavily into how drunk driving affects the human body. I had to look into many sources to find this type of information. I learned how to properly read and extract information from research papers, YouTube videos, blogs, and even police reports I found

online! This research process taught me a lot and gave me an insight into how scientists approach their work when they are starting too. Here are some of my findings:

DUI is a crime in which an individual goes behind the wheel after drinking an excessive amount of alcohol. Their consumption amount can be measured by their BAC. This information benefits law enforcement officers and is a leading factor in how the driver behaves.

Alcohol impacts various systems in the body including:

- The Brain: Alcohol slows down brain communication channels, which can reduce judgement, reaction time, and motor function.
- Vision: Peripheral vision and visual acuity are diminished due to alcohol. This prevents one from detecting the movement of external objects on the road and focusing.
- Muscle Coordination: Muscle coordination is affected due to excessive intake of alcohol, which causes a lack of control over steering, braking, and monitoring lane position.
- Reaction Time: Reaction time is seriously slowed down by alcohol, which causes drivers to react slowly in emergencies.

Effects on Behavior:

- Impaired Judgement: Intoxicated drivers may overestimate the degree of impairment and may engage in dangerous actions like speeding or running red lights.
- Erratic Vehicle Movement: Swerving, failure to maintain lane positions, and unstable vehicle speed are frequent indicators of alcohol-impaired driving.
- Decreased Hazard Detection: Alcohol reduces an individual's capability to detect hazards on the road, and they are more likely to have an accident.
- Overconfidence: Drinkers overestimate their skills and take dangerous maneuvers such as tailgating or cutting corners.

Studies indicate an exponential rise in the relative risk of a crash with a linear rise of BAC. The National Highway Traffic Safety Administration (NHTSA) informs us that the following blood alcohol concentrations (BAC) in a driver will have the following predictable impact on his or her ability to drive safely.

- A BAC of .02 will result in a “deterioration in visual functions (rapid tracking of a moving target), a deterioration in the capability to perform two tasks simultaneously (divided attention)”
- A BAC of .05 will result in “reduced coordination, reduced capability to track moving objects, steering difficulty, reduced response to emergency driving conditions”
- A BAC of .08 will result in “concentration, short-term memory loss, speed control, reduced information processing capacity (e.g., signal detection, visual search), impaired perception”

- A BAC of .10 will result in “reduced capability to maintain lane position and brake appropriately”
- A BAC of .15 will result in “substantial impairment in vehicle control, attention to driving task, and in required visual and auditory information processing”

After doing this research, I looked into the current work on how drunk driving is being detected. It came to my shock that almost all of these technologies are still in their beta, which means that they are being developed and haven't been implemented or rigorously tested. This issue has been going on for decades and I was confused as to why solutions weren't being made as fast. But after doing more research I found a couple of key reasons to this issue, which include the cost, the effectiveness, and much more. Here are some findings for current work:

Currently, most drunk driving detection technologies are still under construction. One promising solution lies in the car, where companies are testing breath, facial, and touch-sensing technologies.

a. Breath Based
    i. The systems are in-car integrated breath detectors, which are normally mounted on the steering column.
    ii. The system monitors BAC levels via a scan of the driver's breath using the police-style breathalyzer-looking device.
b. Facial recognition
    i. The in-car system with an in-car camera scans the driver's face and line of sight by facial recognition with the use of AI.
    ii. The system searches for the following indicators of impairment:
        1. Drooping eyelids or half-shut eyelids (indicative of drunkenness or drowsiness).
        2. Sluggish blink and sluggish response (since alcohol impacts the nervous system).
        3. Abnormal head movement (which shows a lack of focus or loss of control).
c. Touch Sensing
    i. Uses sensors that are installed on the steering wheel, gear shift lever, or ignition button to detect alcohol by skin contact.
    ii. Uses infrared spectroscopy or an electrochemical sensor to detect BAC by contact with sweat on the skin.

One main problem with these detection systems is that they can be highly inaccurate and invasive, as items such as cologne and mouthwash, which contain alcohol, can trigger a false positive warning from the detection system (in breath-based technologies). Coupled with the fact that this driver data can easily be illegally leaked and used for malicious purposes, it hasn't

been largely implemented because of the intense bureaucratic concerns involved. Moreover, the high costs of these technologies pose a monetary barrier to implementation. Finally, in some devices, such as car breathalyzers, constant human interaction is required, which can be overwhelming and unnecessary.

1. What do current traffic cameras do?

Currently, traffic cameras are installed around the world. There are 2 main types of these devices.

- Law enforcement and violation detection
    - Detect red-light runners
    - Detect speeders
    - Detect vehicles failing to break at stop signs
- Traffic Management
    - Track vehicle density and speed to understand if there are any clogs
    - Accident detection
    - Provide environmental condition alerts

These current systems work towards catching simple roadside offenders, however, they fail to analyze driving patterns in vehicles. This information alone isn't enough to detect drunk drivers, and my project can aid in detection efforts.

Companies, such as MADD Canada, are working to advocate and support work to stop drunk and drug-affected driving. However, it does not seem like they are making any significant progress towards external technologies to stop this problem. There are many posters and outreach, all of which try to let others know how much of a problem drunk driving is. However, relying solely on awareness and education is insufficient. People will only start to stop driving under the influence if they know cameras are watching their movements.

Finally, after doing all this research, I made three objectives to guide my innovation. here they are:

1. Privacy

Current drunk driving detection systems are extremely invasive and require lots of human interaction. Moreover, since many of these systems involve cloud-based data storage, there is a high chance that user data can be leaked to malicious third-party sources for illegal activities. To address these issues, my first objective focuses on making sure that my camera is privacy-preserving. By focusing on external vehicle movements instead of personal driver data, this solution eliminates the need for intrusive in-car monitoring. Additionally, the system

processes all data locally on an edge computer to mitigate any further privacy concerns. This approach respects the driver's privacy while still effectively identifying dangerous behaviour.

2. Cost-effectiveness and Scalability

Current drunk driving detection methods have not been implemented because of the intense bureaucratic concerns behind them. Many governments, politicians, and vehicle companies have made few efforts to implement existing impaired driving detection technology on a large scale as it could cost them exorbitant amounts of money to install these devices in every car. To solve this problem, my second objective focuses on making my device cost-effective and easy to install in different parts of the city. My system aims to solve this issue by requiring only one processing unit to monitor multiple vehicles simultaneously. This drastically reduces the cost per vehicle compared to in-car systems (which will cost hundreds of millions), making it far more practical and scalable. By eliminating the need for individual systems in every car, my solution ensures that cities can implement this at a fraction of the cost while still accurately identifying drivers under the influence.

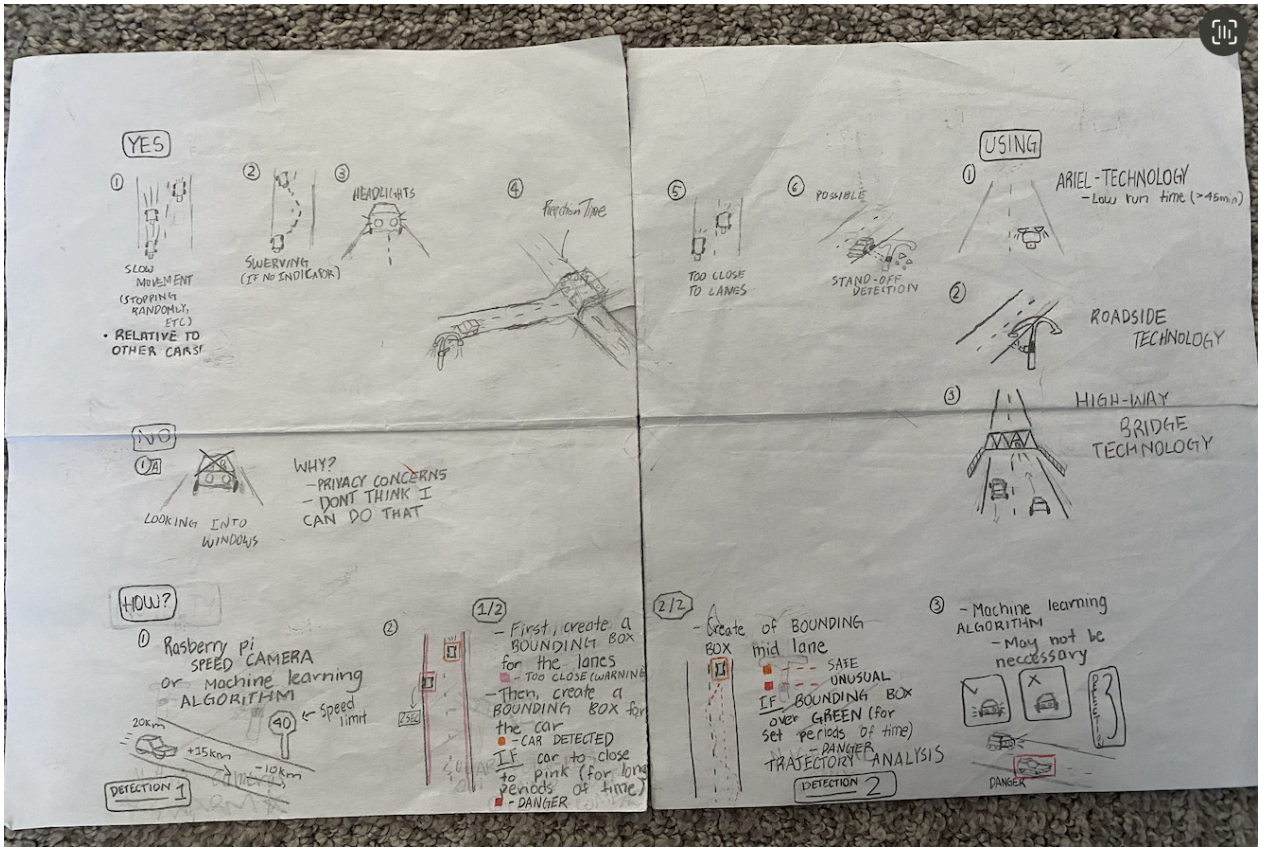
3. Autonomy

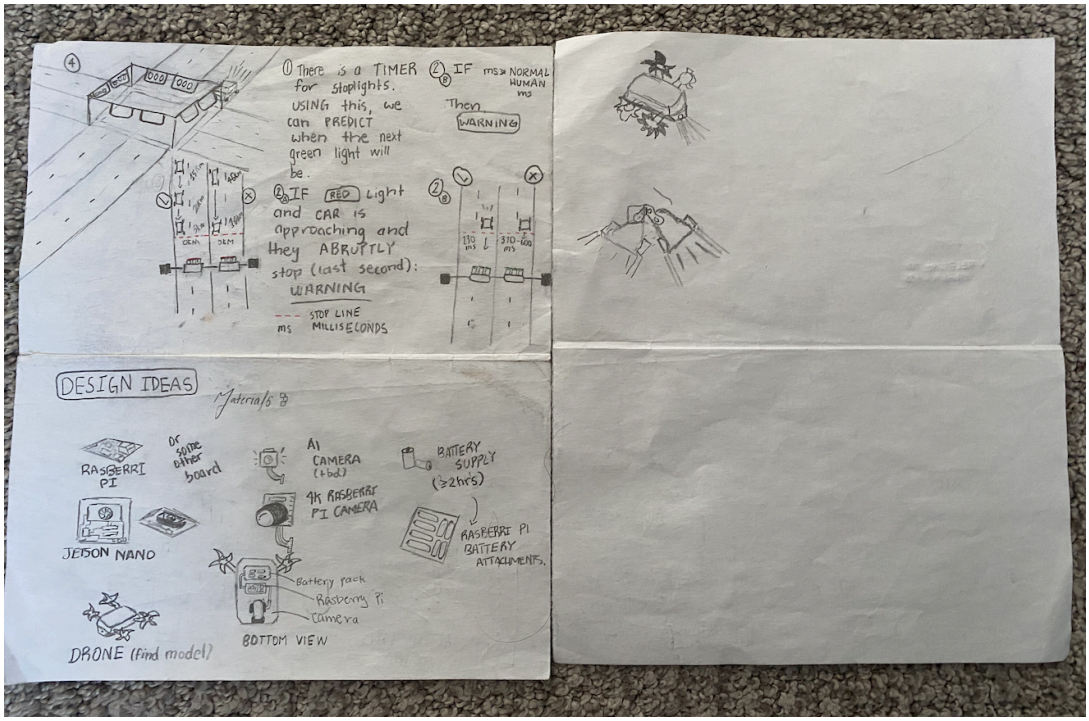
Current drunk driving detection systems are extremely invasive and require lots of human interaction. To address these issues, my third objective focuses on making sure that my camera is non-invasive. My device uses an external, standalone camera that is mounted on a bridge overlooking cars to monitor vehicle behaviour rather than individual drivers. This solution saves both time and money for drivers and companies alike.

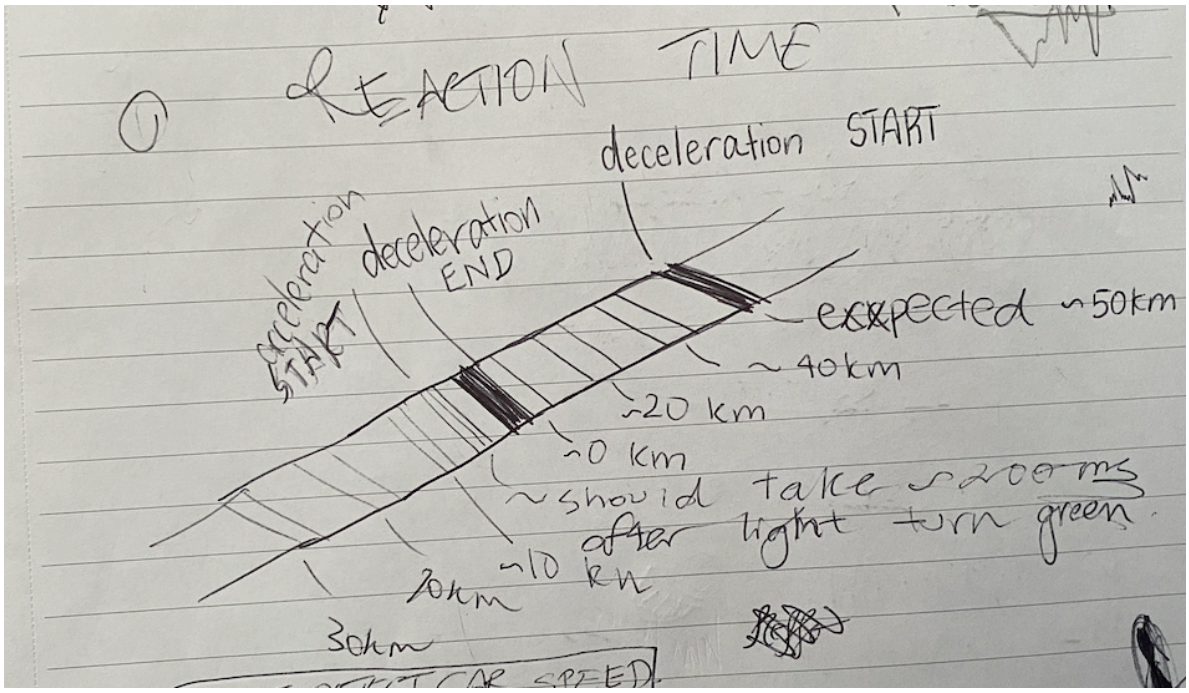
I also Had two other smaller objectives in mind which were durability and public safety. I hope to still work on those but I will keep my main objectives as a priority.

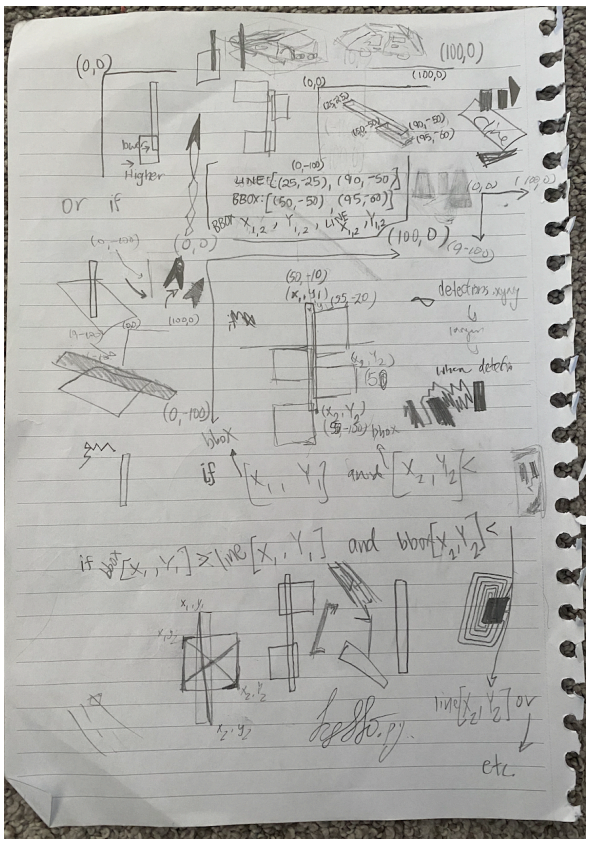
<u>November - December 2024</u>

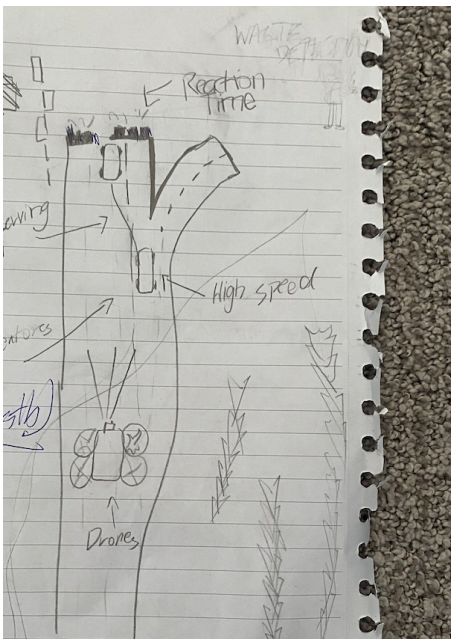
This month was filled with lots of prototyping, drawing in my notebook, and learning difficult concepts. I love drawing blueprints of possible designs, and I filled my notebook with pages and pages of possible iterations of my innovation. In these drawings, I highlighted what I wanted to detect, and what I wanted not to do to keep my device not only novel but effective. I brainstormed different ways to implement my device, and I even hand-wrote the logic for each detection mechanism in simple English terms. Finally, I drew some design ideas for some embedded computers I could be using to pursue this device. Now, of course, a lot of what I drew didn't work out, but they were the jump start for my ideas to make my innovation great in the end. Here are a few examples of drawings I made:


YES
SLOW MOVEMENT (STOPPING RANDOMLY, ETC)
• RELATIVE TO OTHER CARS
SWERVING (IF NO INDICATOR)
HEADLIGHTS
Reaction Time
NO
LOOKING INTO WINDOWS
WHY?
- PRIVACY CONCERNS
- DONT THINK I CAN DO THAT
HOW?
Rasberry pi SPEED CAMERA or Machine learning ALGORITHM
Speed limit
20km
+15km
-10km
DETECTION 1
1/2
- First, create a BOUNDING BOX for the lanes
- TOO CLOSE (WARNING)
- Then, create a BOUNDING BOX for the car
- CAR DETECTED
IF car to close to pink (for long periods of time)
- DANGER
2/2
- Create of BOUNDING BOX mid lane
- SAFE
- UNUSUAL
IF BOUNDING BOX over GREEN (for set periods of time)
- DANGER
TRAJECTORY ANALYSIS
DETECTION 2
TOO CLOSE TO LANES
POSSIBLE
STAND-OFF DETECTION
USING
ARIEL-TECHNOLOGY
- Low run time (>45min)
ROADSIDE TECHNOLOGY
HIGH-WAY BRIDGE TECHNOLOGY
- Machine learning ALGORITHM
- May not be neccessary
DANGER


There is a TIMER for stoplights. USING this, we can PREDICT when the next green light will be.
IF RED light and CAR is approaching and they ABRUPTLY stop (last second): WARNING
STOP LINE
ms MILLISECONDS
IF ms > NORMAL HUMAN ms
Then WARNING
DESIGN IDEAS
Materials
RASPBERRI PI
OR SOME OTHER board
JETSON NANO
AI CAMERA (+bd)
4K RASBERRI PI CAMERA
BATTERY SUPPLY (≥2hrs)
RASBERRI PI BATTERY ATTACHMENTS
Battery pack
Rasberry Pi
Camera
BOTTOM VIEW
DRONE (find model)







This month I also learned about what skills I needed to acquire to complete this project. It was very daunting at first when I realized I had to learn a whole new programming language, CAD design, and data analysis, but my eagerness to complete this project and will to help others

pushed me through! I learned that Python was the most fundamental programming language when one wants to complete a machine/deep learning project, and so I first had to learn the basics of that. I watched many tutorials on YouTube, and I also used a Duolingo-type app called Mimo. This process took me around 2 months to fully master, as I also built small projects to become a better Python programmer. In school, I was taking a Communications technology class, and I had the chance to learn a CAD designing on a platform called Fusion 360. This was a great experience and I got to see how engineers in the real world build simulations and models to solve problems.

December 2024 - January 2025

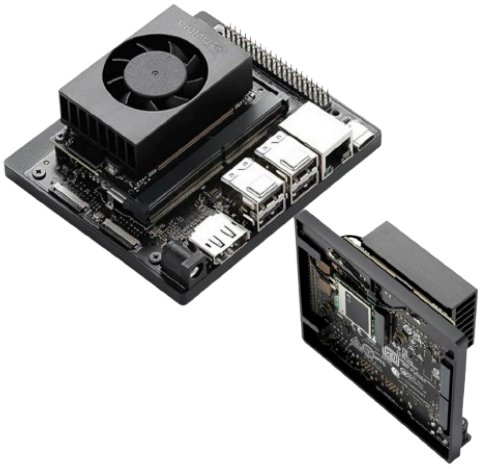
This month was all about learning the nooks and crannies of image classification, computer vision, and deep learning. there were hundreds of YouTube tutorials online, but I liked a channel named Nicholas Renott. His tutorials helped me understand the basics of machine learning and how I can make my own computer vision project. I also took a deep learning specialization course on Coursera taught by Andrew Ng. This was a very intensive course, but I learned a lot about how computers see the world mathematically. To get a full grasp on this topic, I finally went on to a website called Data camp and took three courses. The first course was a general Scikit learn crash course, and that built the fundamentals of my knowledge. Then I moved on to deep learning with pytorch specialization and finally an advanced computer vision course. This took a lot of grit and willpower but in the end, I finished it and I was very proud! After this, I planned what materials I needed for my project. My initial Instinct was to go with something like the Arduino, but I later learned that they have almost zero computation power when it comes to deep learning/computer vision tasks. Then looked on to its more powerful brother, the Raspberry Pi, but even the Pi couldn't withstand as much as I needed. After a lot of research, I finally settled on Jetson Orin Nano. For the camera however, I did use the Raspberry Pi camera module V3, because it is very reputable and works well with any embed computer with CSI ports. Here are some info about my materials list:

My project started with programming and training on my personal computer. This process involved downloading videos, sending them to my collab notebook, and then getting it processed. The entire process can take up to 10 minutes, therefore proving inefficient in the long run. If I took a video of a road and then had to send in my video through the cloud, the upload process would also take far too long and dangerous drivers under the influence would be long gone by the time I had processed the video! This meant everything needed to be processed in real time. Real-time refers to the ability of the computer to execute functions in the present, with no delays that can cause problems. I couldn’t just bring my laptop outside and leave it there for multiple nights, so I researched the issue and found my solution: an edge computer. These tools are small programmable boards (small “computers”) that boast

extremely high computing power. They process data closer to the source of the data, rather than in a centralized data center, to reduce latency and improve performance. In my case, I needed an AI-specific edge computer. After lots of research, I settled on the NVIDIA Jetson Orin Nano. The cost of this device is $250.

Here are some specifications of the NVIDIA Jetson Orin Nano:

- Up to 67 TOPS (trillion operations per second) of AI performance
- Processor Thermal Design Power (W) between 7W and 15W
- GPU: 1024-core NVIDIA Ampere architecture GPU with 32 Tensor Cores
- Memory: 8GB LPDDR5 memory at 68 GB/s bandwidth (for rapid data processing and high-resolution video feeds)
- Supports up to 4 cameras simultaneously
- Versatile connectivity with different port options
- Compact design which allows it to be placed almost anywhere



Here are some of the other boards and cameras I contemplated using:
Embedded computers:

1. BeagleBone AI-64: High-Performance AI & Machine Learning SBC - 102110646
2. Pro Portenta Max Carrier - ABX00043
3. Pro Portenta X8 - ABX00049
4. Raspberry Pi 5 - 64-bit Quad-Core 2.4GHz Single Board Computer, 8GB RAM and Official Raspberry Pi AI HAT+, Build-in Hailo-8 AI Accelerator To Quickly Build A Wide Range Of AI-powered Applications, Options for 13 TOPS / 26 TOPS
5. Nvidia Jetson Orin Nano Developer Kit - 102110839
6. UDOO Bolt V3 - Advanced SBC with AMD Ryzen & Radeon Graphics

7. [Udoo Bolt V8 - SC40-2000-0000-C0]

Camera Modules:

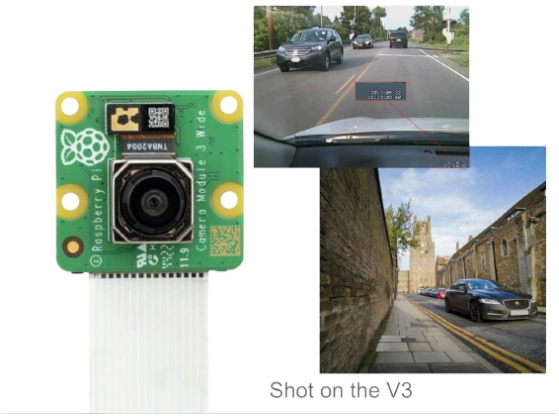
1. [Official Raspberry Pi Camera Module 3 (Made in The UK) : Amazon.ca]
2. [Arducam 64MP Hawkeye Ultra High-Resolution Autofocus Camera Module for Raspberry Pi, Compatible with Raspberry Pi 5/4B/3B+/3B/2B/A+/Zero/W/Zero WH : Amazon.ca]

1. Raspberry Pi Camera Module V3

The main item of this whole project - a camera! This camera essentially acts as the “eye” of the project, providing a high fps playback of the environment. I needed a reputable and powerful camera, so I chose the Raspberry Pi Camera Module V3. The cost of this camera is $25.

Here are some specifications of the Raspberry Pi Camera Module V3:

- Comes with an improved 12MP IMX708 Quad Bayer sensor and features a High Dynamic Range mode
- Features an ultra-wide 120-degree angle of view
- Features ultra-fast autofocus as standard



2. 3D printed case and Solar Panel

To make my device more durable and efficient, I am including additions so that it does not get damaged by rain and rough weather. Currently, I am modelling and printing the case and hood using 3D printing technology to be lightweight yet strong. Apart from making it more efficient, I have purchased a solar panel, which I will connect with a sun-tracking system for maximum energy yield. This will enable the device to work for extended hours, even in isolated locations.

Secondly, I am researching how to waterproof electronic components and test various materials for added durability and longevity.

January - February 2025

This month, I started programming and working on my project. I had a lot of experience now that I took extensive tutorials, so I was prepared. I started by training the model on a car dataset. Here is some information:

Detecting Objects

The very first part of this project, essentially the backbone, is object detection. When a computer processes a video, it cannot tell right away what is what. To help it learn, we can use a technique called deep learning.

Step 1: Acquiring the dataset
A computer needs data to learn. The first step in training a deep learning algorithm is finding a dataset to feed to the algorithm. A computer can read this data and extract conclusions using a Convolutional Neural Network. In my case, a dataset would be a large file consisting of thousands of images of different types of vehicles (cars, motorcycles, trucks, vans, etc). While there wasn't a single dataset for all these types of vehicles, I managed to find a dataset tailored to each specific one. It is very important to detect all types of vehicles because their type affects their trajectory, speed, acceleration, etc. My algorithm needs to take these factors into account when detecting drunk drivers to avoid any false positives. I got my datasets from a website called Kaggel.

Figure 9. An example of a dataset (these are just a few images - typically datasets contain tens of thousands of images)

Step 2: Annotating the dataset

Annotating the dataset is an important step before training the model. Every image/video has to be labelled with information about the objects in it. I simplified this process using Labellmg, an open-source graphical image annotation tool. It defines the spatial extent of each object in an image and draws a simple box around it (called a “bounding box”). I had to go back and manually label each image because some images show scenes of things other than vehicles. The computer reads images in pixels, and therefore I need to make sure that it only understands vehicles in each image. Without labelling, the final output can be very inaccurate and messy. The datasets I found include vehicles in many types of conditions and angles, which is important when you want the model to be generalizable to any given scene.

Step 3: Training the dataset

After obtaining and labelling the data, we have to train the algorithm. When you train on a dataset, it is best practice to use an open-source library to avoid any errors. In my case, I used the YOLOv8x library. It uses convolutional neural networks to process each image. Convolutional Neural networks (CNN) are a specialized type of deep learning algorithms that are designed to process all types of visual data like images and videos. A CNN contains 3 main layers: convolutional layers, pooling layers, and fully connected layers. All of these work together to process and analyze images.

- Convolutional Layer:

        - This is like the main building block of a CNN. It applies filters (also called kernels) to the images inputted and extracts a few main features from the image like edges, texture, or shape. Each filter then slides over the image in increments. This process is called convolution. Then, a dot product is calculated between the input pixels and the filter value. This generates something called a feature map, which highlights areas in the image where the specific features are detected. For example, one of the filters can detect vertical edges while another filter can detect horizontal edges. When more convolution layers are added, the model learns from more complex patterns like shapes and objects.
        - Activation Function: After the process of convolution, an activation function like ReLU (Rectified Linear Unit) is applied to help the CNN learn non-linearity. This allows the network to learn complex patterns by focusing on the positive values in the feature map.
- Pooling Layer:
    - The pooling layer reduces the spatial dimensions of the feature map. The two most commonly used pooling methods are max pooling, which selects only the maximum value in a region, and average pooling, which calculates the average value in a region. Pooling makes the network more efficient. It also leaves room for introducing variation in object positioning and size.
- Fully Connected Layer:
    - After many convolutional and pooling layers are made, the output is “flattened” into a 1D vector. Then, it is passed through fully connected layers. These layers combine the individual features to make predictions, like classifying an object or detecting its location.

This is essentially what is happening during the model training period. Training can take many hours to even days to complete! Once training was complete, I had a model that could predict all 4 types of vehicles (cars, trucks, vans, motorcycles) in any given video! The best part about machine learning is that you don't need to constantly be changing your code when you go into a different environment. If you train your model well (ex. giving it a lot of different types of vehicles and many different types of conditions), then it will easily be able to adapt to new situations.

After getting this finished, I looked into open source program that detected drivable area segmentation. One of my first ideas was actually to use an open source model and tweak that to my liking. I researched two main models called YOLOP and YOLOPv2. That stands for “you only look once for panoptic driving perception”. After getting it running however, it did not work in my favor. The problem lied in the perspective. YOLOP was designed to have a camera placed in the inside of a car, but my model needed an external approach. For this reason, I steered away from using this and had to build everything from scratch. I programmed two of

my main detection modules, which was Lane detection and Speed detection. Here is how that went:

Lane Detection:

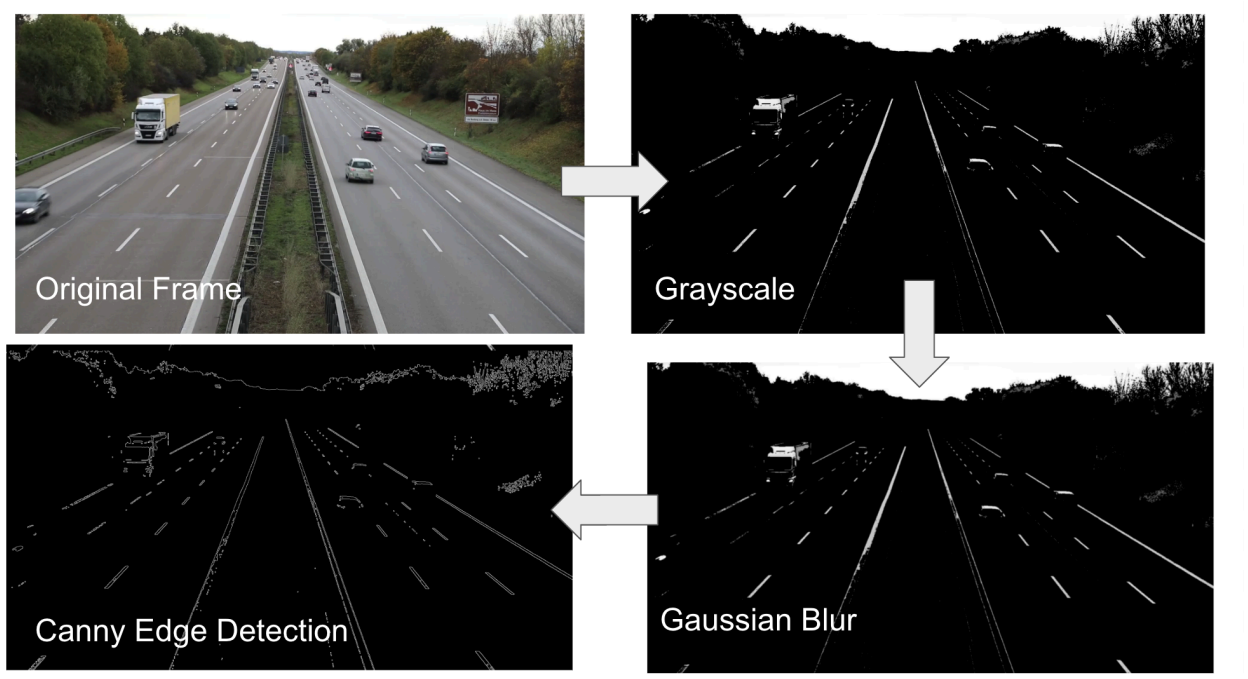
When a driver is drunk, they find it tough to identify the “drivable areas” (which are in-between the lanes), and that could cause the vehicle to stay in a lane path for a prolonged, unnecessary period. As mentioned before, computers can’t pinpoint objects in an image without learning. I can help the computer understand lanes by putting each file through a series of colour changes and then applying a function called Hough Transform. Before understanding the colour changes, we must first understand Hough Transform, a computer vision attribute that detects distinct lines in an image. Detecting lines would be simple if using the “y = mx+b” function, but this equation is not ideal for vertical lines because the slope “m” becomes infinite. Instead, we use the polar form of the line equation:

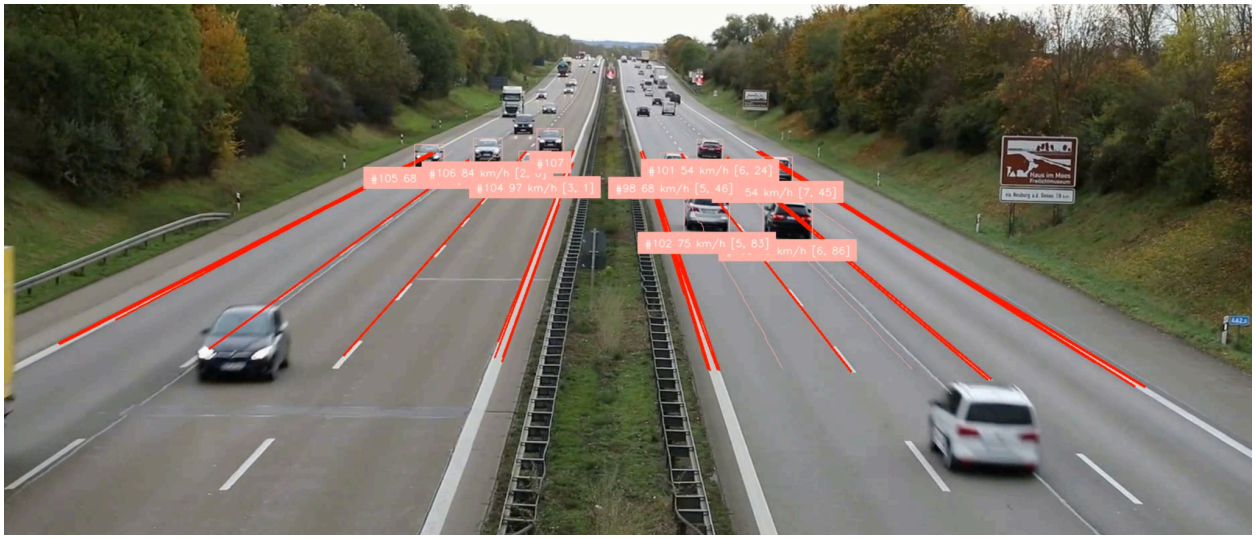
$r = x * \cos(\theta) + y * \sin(\theta)$

where:

- r is the perpendicular distance from the origin to the line.
- θ(theta) is the angle between the normal to the line and the x-axis.
- (x,y) are the coordinates of an edge point.

However, detecting lines in an RGB image is difficult and would produce many false positives (creating lines everywhere). To solve this issue, the program first puts the image, or each frame for a video, into a grayscale. The gray-scale image’s pixels each have an intensity between 0 and 1, compared to the 3 layers between 0 and 255 in a colour image. This not only makes it easier for the Hough transform function but also less computationally intensive on the computer. Next, while it is important to detect as many edges as possible in an image, any possible noise must be filtered out. Noise essentially refers to any distractions in an image that can deter the algorithm from the ground truth. Noise can be reduced by a function called Gaussian Blur. It blurs the image so any small pieces of noise would seem like “blobs” so they don't attract too much attention. Next, the algorithm runs each frame through the “canny edge detector”, which finds every edge. Working together, all of these functions are pivotal to making sure that the final Hough transform logic executes completely. Figure 11. Illustrates the final results of the hough transformation.





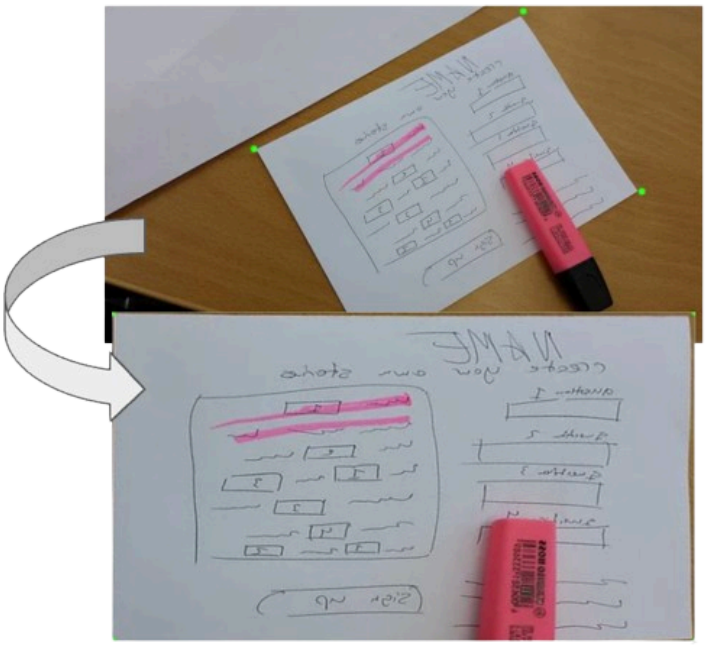
Speed:

In the status quo, law enforcement uses radar guns to perform speed detection of a moving vehicle. A radar speed gun works by emitting radio waves, which bounce off moving vehicles and return to the gun. Changes in the frequency of the reflected waves(due to the Doppler effect) are analyzed to calculate the vehicle's speed. While this method is generally effective, it can get costly and would make my device bulky and impractical if attached. All detection mechanisms occur through a computer's eye (computer vision), and so I needed to figure out how I could detect the speed of a vehicle in a video playback. When I first started to program this, I thought it would be simple: use the speed formula (s = d/t) and the video frames per

second. However, after many tries and fails, I realized that many factors were limiting the accuracy of this detection. The first issue was the perspective of the image. The perspective of the video affects how speed is calculated. In Figure 12., the posts on the right side appear further apart when they are close to the image, and closer when they are farther from the image. The normal equation for calculating speed is distance divided by time, but if we don't have the distance right, then the final calculation will be skewed. The only way to accurately get the distance is to have a bird's eye view of the road. Initially, I bought a drone and tried testing it out, but there were two main reasons why I had to discontinue this idea:

1. Drone regulations in Calgary (where you can fly, how heavy drones need to be, etc) are limiting
2. The drone performance itself was not the best – it was wobbly, had a low runtime of 15 minutes, and could pose a safety risk to drivers as it can be visually distracting

However, after many hours of research, I found a complex Python function called perspective transform. It essentially turns an image from a 3D to a 2D plane. It uses matrix multiplication to make every object in the image the same distance apart, which helps with speed detection.



After this, speed was calculated by computing the distance between two video frames and dividing the result by the inverse of the frame rate. However, when implementing this, another problem arose.

Bounding boxes have a tendency to flicker in the frame - very slightly. This flicker is very hard to see from the naked eye, but it skews the speed reading heavily. I noticed that the speeds I was getting could be 20-30km/h off, and so, to solve this problem, I took the average of the values acquired over a one-second period. In this manner, the speed readings are more

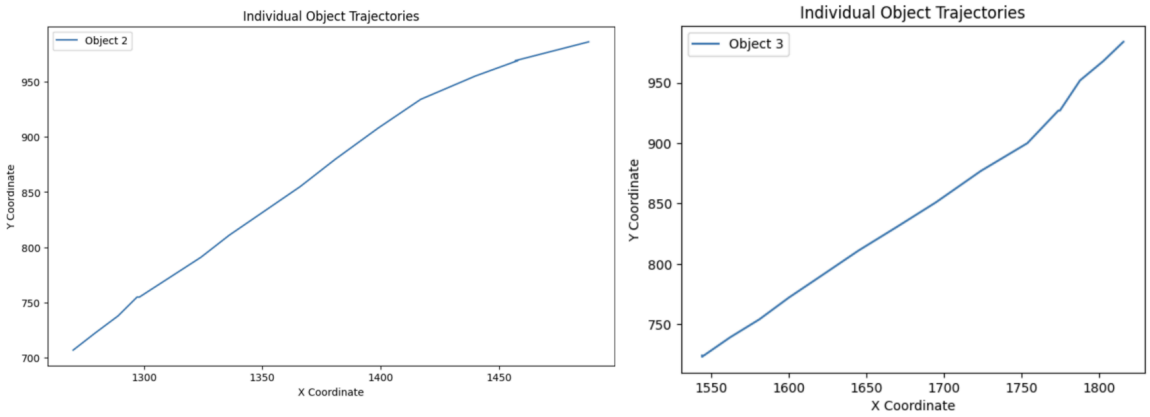
accurate and the car's travel distance is much greater than the flickering-induced little box movement.

February - March 2025

In this month, I finished my next two detection modules, which were trajectory and XY coordinate detection. Here is how that went:

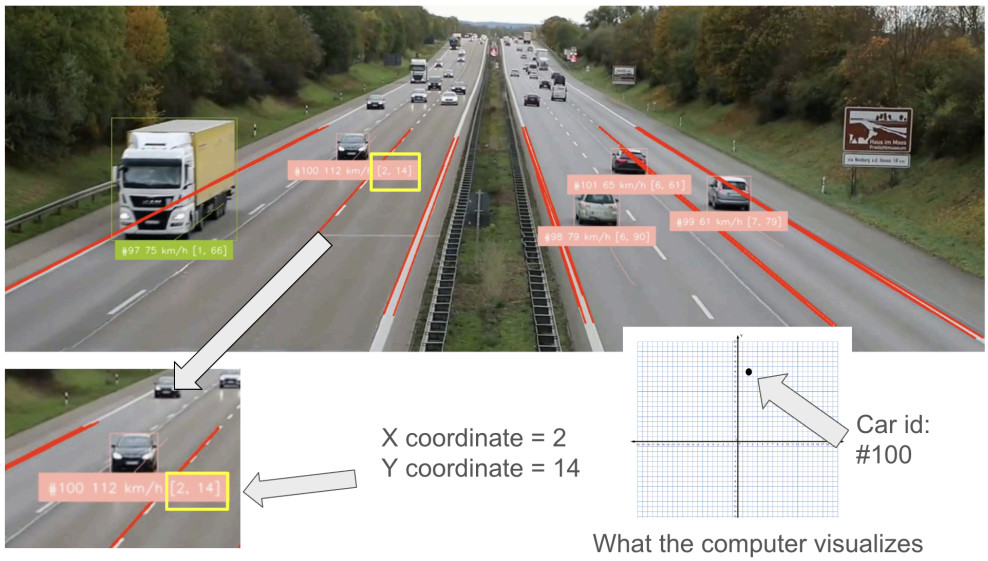
Detecting Trajectory:

A very important factor in detecting whether a driver is drunk or not is its trajectory on the road. When the human eye looks at cars moving, it struggles to remember the exact movements of vehicles over a long period; this is where AI and computer vision can help out! A library named *BYTETrack* tracks objects in a video. Following object identification, the bounding boxes on each vehicle are connected using tracklets via ByteTrack's data association module. In essence, a tracklet is a brief series of frames in which an item has been consistently detected and tracked. ByteTrack ensures that objects are precisely tracked over time, even as they move through the video, by linking detection boxes to tracklets. Additionally, the program filters out redundant detections via a gating mechanism. Once the cars' trajectories are tracked, I need a way to visualize them on a graph. I used a library called MatplotLib to put each individual vehicle's trajectory. This is so the machine learning model that analyzes each trajectory doesn't have to understand complicated images. I am also planning to use perspective transform on plotting trajectories because otherwise they could look diagonal and that skews how the machine learning model sees the image.



XY Coordinate Detection

One of the most significant issues with drunk drivers is that they get too close to other cars, roads, or environments; this is called tailgating. To be able to detect these proximity changes, my algorithm uses XY coordinate detection. This works by placing a virtual Cartesian plane on the video input and projecting the location of every car onto proper X and Y coordinates. It can be achieved using homography transformation, which maps real-world points to a top-down perspective - similar to perspective transformation. Each car becomes a point on a cartesian plane. This enables the system to monitor movement in real-time, detecting dangerous maneuvers like tailgating, and hitting infrastructure. By analyzing these coordinates in real-time, the algorithm can identify impaired driving behaviour and avoid potential accidents to improve road safety.



I also looked into Grant's I could apply for to help me fund the materials in this project as they were slightly expensive. I found a couple, with the most worthy being the microgrant from Youth Central and the Ingenious+ grant from the Rideau Hall Foundation. I applied for both and heard back from the microgrant the first. To my surprise, I got the $200 of funding! I was very happy with this result and it motivated me to work even harder in my project. The results of the Ingenious+ application will come in May.

When I finished all my detection mechanisms I began to analyze each and every one, I first figured I had to know where to test my device, and so I found two main methods, which was real-world and simulation testing. Here's how that went:

Real-world testing:

Once I finished programming and deploying the device, I had to figure out how I would test it. The best two methods would be real-world and simulation. I have done extensive testing on both and have achieved significant results!

Real-world testing:
In the real world, there are many different types of roads, from residential streets to the highway. While my device could be adapted to be used on residential roads (ex: roads that are close to homes, 2-way roads, etc), I decided to prototype and test mainly in highway conditions for two reasons:

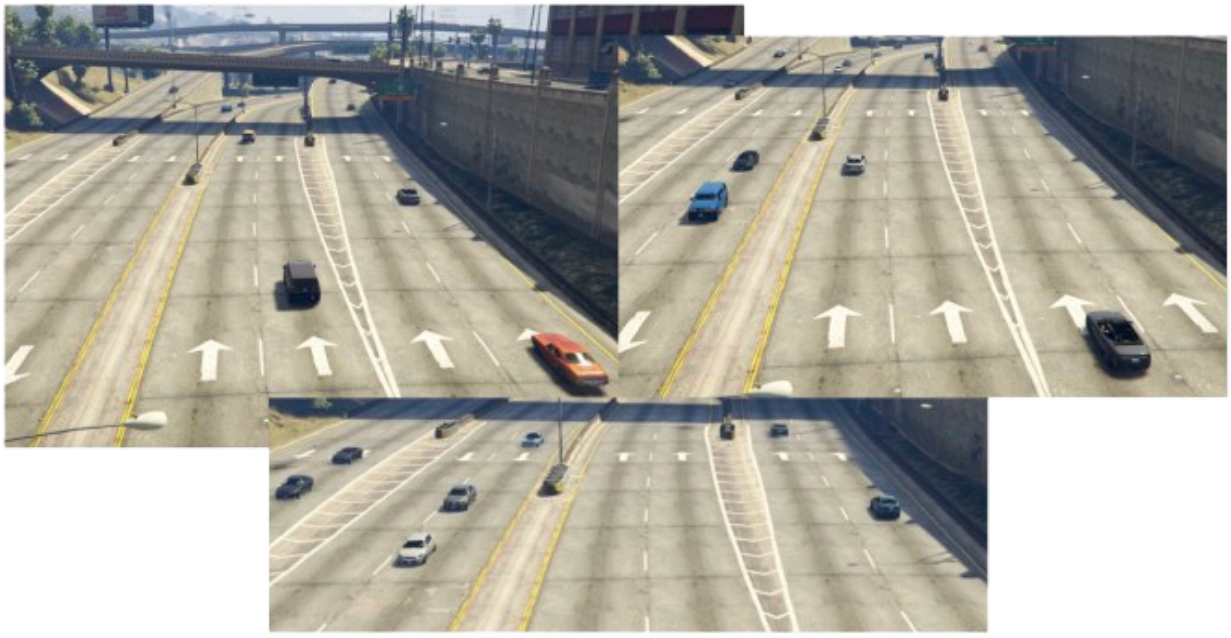
- There is lots of constant data - there will always be more than one car on the highway. On smaller roads you will still get traffic but a low amount of data
- Since there is a lot of data - there is a higher chance of an intoxicated driver being on the road

To execute this testing strategy, I first used videos of cars driving on the internet. Once those tests passed, I surveyed Calgary's highways on Google Maps and looked for areas where there was an overhead bridge. This way, I could record traffic without posing any distractions to the drivers. Once I found an ideal location, I went out and set my device for approximately 1 hour. I stayed in close contact for this time, mainly to make sure there was no theft. To solve this issue in the future, I have made plans to contact the City of Calgary and have my device put on and tested on lamposts. Not only are these structures very high (which would allow for a long stretch of land to be recorded - hence making the system more accurate), but they are immune to any threat of stealing. I could also put my device where speed cameras lie - on top of street lights. Currently, my device can record approximately 500 meters of land.

Simulation testing:
Simulations act just like the real world. Even better: I can stimulate my own, unlimited, and accurate data. In my extensive research to find a great simulator, I had two goals in mind:

- It should be able to stimulate all types of environments (sunny, dark, rainy, etc)
- It should have preloaded AI drivers, so everything looks like normal traffic
- It should be multiplayer (as I need one other person to help me out with this type of testing)

After lots of research and testing different simulators, I settled on using the game GTA V. This game checked all my boxes and I also had it installed from a long time ago, so I didn't need to pay a hefty amount of money for a new game. To make this work, I had my friend use his driving setup and join my online game. I would stand over a bridge - similar to real life - and he would get a car. We replicated all 4 different detection flags (swerving, touching lanes for too long, slow/high speed, too close to other vehicles and infrastructure) in nighttime, rainy, and sunny conditions. Finally, we replicated a vehicle committing all 4 flags in one clip. We made sure the portion of the road covered was similar to the real world (around 500 meters). When this was finished, I uploaded all the clips to my collab notebook to be processed.



I also began to analyze my detection mechanisms, to see if they were accurate and worked. Here are my results:

Detecting Objects

As this was the backbone of my whole algorithm, I needed to make sure that the accuracy was spot on. My overall accuracy was 96.7 percent!

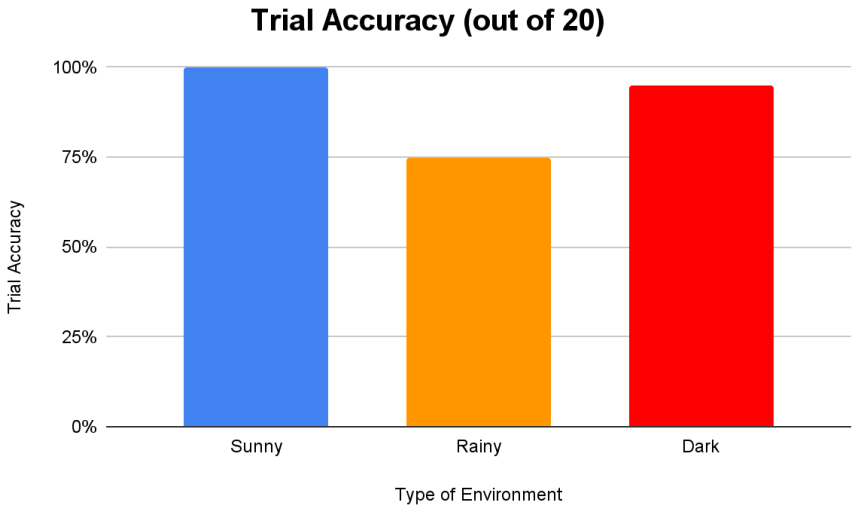
Detecting Lanes

When an individual is under the influence, their eye muscle function is restrained. This causes blurry and crossed vision, making it very difficult for drivers to perceive their surroundings, especially at night. It would be tough to identify the “drivable areas” (which is in-between the lanes), and that could cause the vehicle to stay in a lane path for a prolonged, unnecessary period.

This behaviour can pose a severe risk to other drivers and pedestrians, and so I programmed my algorithm with the following logic to make sure that this demeanour is flagged:

- If the car's bounding box is in contact with any one of the hough lines for more than 3 seconds, then 25% is added to the counter.
- It is for 3 seconds or more because a simple lane turn would cause the vehicle to be in contact with the hough lines for approximately 1-2 seconds.

One of my core objectives is to make my device implementable and autonomous. This means that my lane detection algorithm cannot just work on 1 driving video - it has to work for virtually any scenario. Making a model generalizable is important because it is much more convenient to place anywhere as one does not need to be constantly programming it to adapt to new conditions. To test the ability of generalization, I tested my Hough transform algorithm on sunny, rainy, and nighttime driving conditions. If 90% or more of the hough lines were detected in each video, I counted that as a pass. Otherwise, it would be counted as a “fail”. Below is a graph that details my results in this area:



As you can see, my model works great in sunny and dark conditions but does not do that well in rainy conditions. I am currently working to program a workaround for this and hope to present it at the fair.

Detecting Speed

Drunk drivers can misjudge their vehicle's speed because alcohol slows down cognitive processing, motor skills, and reaction time. This can make the vehicle go over or under the speed limit, or abruptly pick up or slow down speeds. Slower reflexes mean drunk drivers react late to traffic signals, pedestrians, and other vehicles. They may brake too late or too suddenly, causing erratic speed changes.

My algorithm detects this by constantly checking on vehicle speed. I have to factor in the vehicle type because vehicles like trucks brake slower than a car for example. To be even more accurate, my algorithm can also see the speed of a vehicle going into a stop light zone (intersection), and analyze how the braking patterns were there. There is no time limit to flag this since going too fast or slow in the first place is dangerous driving and needs to be stopped.

Since my device uses computer vision to detect the speed of a moving vehicle, I wanted to test it with traditional methods of detecting speed. I asked my dad if he could get into his car, and drive over a highway while I stood atop of bridge recording. Using the car's speedometer as an accurate gauge of the car's speed, he would go in the 100-105 km/h range and I would see if my algorithm can detect an estimation similar. I tested this 3 times to make sure it was accurate and here were my results:

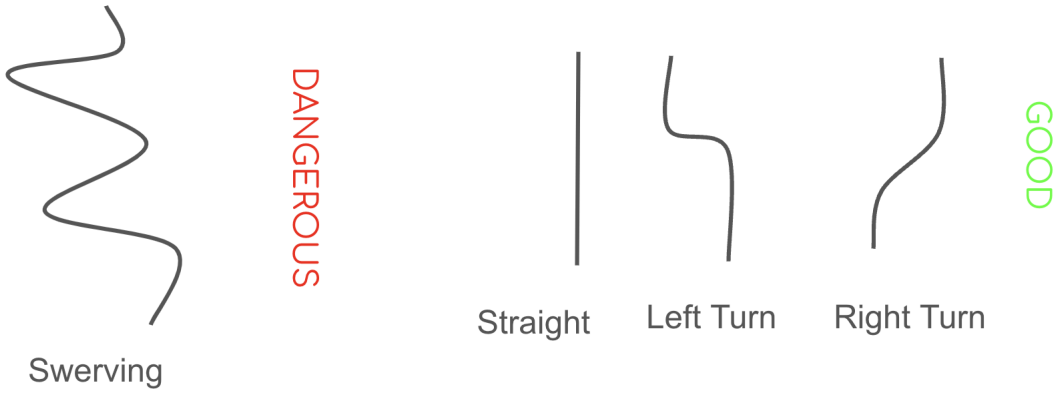
Test 1:	Actual Speed: ~103km/h	Detected Speed: ~106 km/h	Difference: ~3km/h
Test 2:	Actual Speed: ~100 km/h	Detected Speed: ~102 km/h	Difference: ~2km/h-
Test 3:	Actual Speed: ~104 km/h	Detected Speed: ~106km/h	Difference: ~2km/h

As shown by the data, the difference is very minimal, which proves that speed detection is very accurate.

Trajectory

Alcohol affects a specific part of the brain called the cerebellum. It is responsible for human balance and movement. When the cerebellum isn't at full capacity, it makes it harder for the driver to steer smoothly, which causes jerky and inconsistent vehicle movements. Moreover, as mentioned before, a drunk driver takes longer to process visual and spatial information. They most likely do not even notice when their vehicle is drifting or swerving in the amount of time it would take for a normal driver to figure this out.

After I detected the trajectories of cars in the ByteTrack library, I had to decipher which types of trajectories were considered drunk and which were considered sober. My first instinct was that a straight line was sober, which is correct, but I failed to realize that cars would still be making left and right turns. When a car takes these types of turns to switch lanes the trajectory isn't straight but slightly curved and ends up forming a cubic looking function. With these three sober trajectories in mind, I took many images of each and trained my machine-learning model to understand those trajectories as normal. when it comes to a drunk driver's trajectory, it looks like a squiggly line.



Once the machine learning model identifies the trajectory of the vehicle, it adds 25% to the counter if detected as dangerous. The trajectory is only analyzed when the car is just leaving the frame because it works best when analyzing a large period of driving time.

XY Coordinate Detection

Drunk driving disrupts the brain's ability to judge distance. This makes it harder to gauge how close one is to another driver or infrastructure. a drunk driver would think they have more space than they actually do, which could lead to tailgating or near collisions.

With the XY detection mechanism, I can detect how close one car is to another. The logic behind this states that if the X coordinate of one vehicle minus the X coordinate of the other

vehicle and the Y coordinate of one vehicle minus the Y coordinate of the other vehicle is in a range between -10 and 10, then the car would be flagged.

To explain this better, take two cars for an example:

- Car A: (200, 206)
- Car B: (207, 198)

These two cars are very close together, and we know that because the coordinates match up very close together. My initial idea was just to compare if the X and Y coordinates matched up exactly, but that wouldn't work because a car cannot be inside another car, but rather at least very close to it. If you minus the x and y coordinates by each other, then you will get a number close to 0 if they are close together, and that is how the logic works.

If this tailgating continues for over 3 seconds, then the car would be flagged with a 25% added to the counter. Piecing together all the information so far, if all flags were detected, then the counter would reach 100%. This is when the vehicle would be flagged as a drunk or dangerous driver. Even if one detection was flagged, however, it wouldn't appear as a drunk driver as it could have been a spontaneous mistake made by the driver. To test the effectiveness of the detection flag, I used the GTA simulation and had two cars get very close to each other. I wanted to make sure my program flagged this dangerous behavior and so I ran through this scenario 20 times. The overall accuracy was 95%, which means that 19 out of 20 trials conducted successfully worked.

<u>March - April</u>

During this month, I finalized my device and analyzed my objectives. I made my conclusions and started working on some future directions I was thinking about. Here is how that went:

Privacy

- My project makes sure the privacy of the drivers is not compromised for malicious purposes. It blurs out the front view of the hood and makes sure that no information is leaked as all the processing happens on the edge.

Implementation and Scalability

- This device is much more effective than current drunk driver detecting systems and surveillance cameras. Not only is it cheaper in the long run, it can detect many more flags. It can run 24/7 and detect multiple cars at once. Its compactness allows it to be implemented on a large scale, in any environment, and the machine learning model behind it makes it “generalizable” or adaptable for any type of situation given to it. This means there is no need to tweak any parameters before putting it into a new situation.

Autonomy

- This device is fully autonomous and requires no human intervention at all. In fact, the Jetson Orin Nano is so powerful that it can train itself through a process called unsupervised learning. This means that it can slowly adapt to new data that is getting in the real world and make better predictions without any human training needed. I believe this is a great addition as it not only makes it easier for humans but also better for the safety of vehicles.

General Conclusion:
The purpose of this project was to design and program an autonomous, efficient, and implementable robot that can detect and prosecute drunk drivers in real-time. After significant testing and variations throughout each of the 5 detection methods (Speed, XY, Lane, Trajectory, Objects), I have successfully created a successful device that is $300, fully scalable, and performs better than traditional drunk driver detection systems.

Practical Applications:
The practical applications for my drunk driver detection system cross law enforcement, traffic safety, and autonomous vehicle technology. As part of real-time traffic monitoring, the system can be embedded in CCTV cameras or roadside sensors to detect drunk driving and alert the authorities before accidents happen. Police officials can utilize it to select high-risk drivers without using random stops and breath tests, hence facilitating easier implementation of road safety. Insurance companies can also use the system to assess the driving behavior risk, hence reducing accidents and improving road safety policy. I believe this system has the potential to save lives, avoid accidents, and bring a revolution in traffic monitoring across the globe.

Future Directions:
1. Alerting the authorities
After my device detects drunk drivers, I want it to alert the authorities so they can arrive at the scene and prosecute the driver. I have been working on this and I am finishing up training for the model. I will be presenting this at the CYSF 2025.

2. Environmental Context Awareness
Erratic driving is not always drunkenness—it can be a reaction to road surface, weather, or surprise litter. Subsequent releases would include live road conditions, e.g., hairpins, road holes, or heavy rain, so that innocent behaviour is not taken as drunkenness.

I worked on my trifold for my school science fair, and write up for the Calgary Youth Science Fair. I regularly went to the Calgary Public Library to print out sheets of paper to be cut for my school's tri-fold, so I have to thank them for their amenities! Now, I am working on making my project even better by not only raising awareness about drunk driving but also adding a

mechanism to alert the authorities via messages when a driver who is under the influence is detected.

Thank you for reading my logbook!