HUD Avoidance Car System

Charles Doodnauth, Minh Pham, Tien Nguyen, Steven Swaby

Contents

[Acknowledgements & Mentions 3](#_Toc26489137)

[Abstract 4](#_Toc26489138)

[Introduction 5](#_Toc26489139)

[Background Information 6](#_Toc26489140)

[Idea / Design Process 6](#_Toc26489141)

[Results 10](#_Toc26489142)

[Discussion 12](#_Toc26489143)

[Conclusion 14](#_Toc26489144)

[Lessons Learned 14](#_Toc26489145)

[Future Work 15](#_Toc26489146)

[References 16](#_Toc26489147)

# Acknowledgements & Mentions

Dr. Peter J. Kootsookos

* Assistance throughout the entire project, from design to testing.

Dr David Broderick

* Assistance with various questions and coding.

Jonathan Braverman

* Assistance with efficiency issues through multithreading and selective scan areas for tracking.

Caleb Bostic-Gardner

* Assistance with lane tracking program and every coding related question anyone could have.

Jordan “MVP” Graves

* Helped secure replacement equipment after camera failure.

# Abstract

Studies have shown that pedestrian deaths from motor vehicle incidents have been increasing not only within the US, but across the world. This could be caused by several reasons however we believe that this rising number of incidents can be reduced with a seemingly simple solution, a second set of eyes. By taking advantage of progress made within the computer vision field, an in car pedestrian detection system could alert drivers of unnoticed pedestrians that emerge within the vehicles path. Acting as an automated human visual system allowing the driver to notice what may have been missed and react before it’s too late. We must also acknowledge that other dangers exit and could be also solved with the same methods, to prove this, the system will also monitor the road lanes, alerting the driver if their vehicle begins to drift out of their lane. It will be built with cost as a factor as we wish to make it available to all, whether it’s a citizen in a world power or an emerging nation.

# Introduction

*Why?*

Based on a study from ghsa.com from 2007 to 2016, the percentage of pedestrian deaths from

total motor vehicle deaths continue to increase. From 2015 to 2016, pedestrian fatalities increased 28%

from the ten largest cities in America. The roads are extremely dangerous as drivers and pedestrians are

becoming less cautious and more careless. This trend shows to be true across the globe as pedestrian

deaths occur at alarming rates, the estimated road traffic death rate (per 100 000 population).

According to the World Health Organization, Algeria 23.8%, Angola 26.9%, Belize 24.4%, Benin 27.7%,

Thailand 36.2%, and Venezuela 45.1% are just a few of the nations that fall above the 20% mark.3

*What?*

Goal is to alert the driver of potential danger caused by unseen pedestrians and lane drift. The

system will contain a multiple component, working together in order to supply the driver with as much

accurate information as possible. It can be seen as two different detection systems running together to

give the driver warnings and alerts of potential situations that may have gone unnoticed by the driver

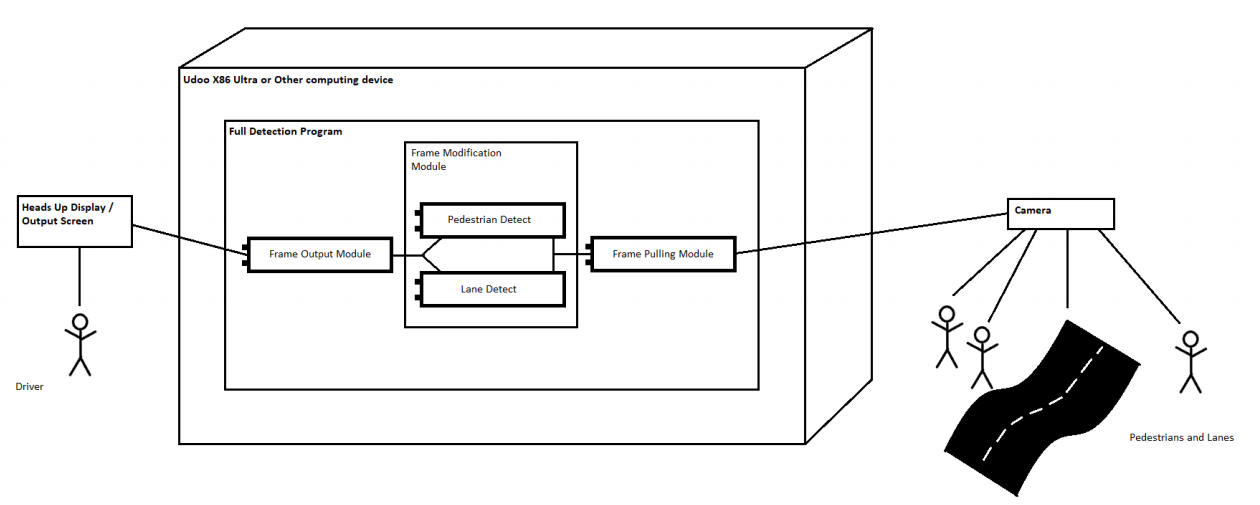
and aims for compatibility among vehicles manufactured between years 1985 to present.

# Background Information

The field of Computer Vision is constantly improving. The technology has reached a point where high precision and accuracy algorithms are available to the public through open source libraries such as OpenCV. Research into new and improved algorithms are ever changing and can use various hardware with varying levels of identification accuracy. This combined with the current market of high power and efficiency SoC computing devices, allows the creation of very powerful mobile equipment. With the dropping cost of performance hardware, the device proposed here would be a valid solution to our problem.

# Idea / Design Process

To begin the design of the system, a block diagram was drawn out to give a general high-level overview of what we were aiming to accomplish. Knowing that id was desired to be a simple plug and play design it made sense to take advantage of SoC. We decided to challenge ourselves by programing in Python, we do acknowledge that OpenCV in python is shown to run slower than C++, and that with C++ options such as boosting performance with a GPU is possible. However, the budget would not allow us to pursue this option, and the experience of working with python would outweigh using a language we were familiar with. Once these two major decisions were agreed upon, we drew up the block diagram. The following block diagram (fig.1) was the most recent update, reflecting the final scope of the project.

Fig. 1

Our external peripherals included a camera with a 150-degree field of view, and a heads-up display. The original planned heads up display was to be an overlay on the windshield, this way the driver was not distracted to check a screen on the dash. However, locating something along those lines proved to be somewhat impossible, so for our purpose we will be using an external monitor. The camera we decided to use was a GoPro, because of its high FPS capture rate, field of vision, and high-resolution video. The main computing unit chosen was a Udoo X86, this device at the time we believed would be more than capable to handle the video processing we required. As for the program itself, it was broken into three separate pieces in order to take advantage of multithreading capabilities. One module focuses on pulling the frame from the camera, the second processes the frame scanning for pedestrians and lanes and making modifications upon detections, lastly the third finally module passes the modified frame to the output display.

# Implementation

After sourcing and ordering components (list and pricing can be found below in fig 2), work on researching various OpenCV detection methods and a python crash course was completed. Based on the hardware acquired and lack of GPU processing ability (budget issues), a detection method called “Histogram of Oriented Gradients” was selected as the main detection system. This method, created by Navneet Dalal and Bill Triggs, relies mainly on the CPUs processing abilities, and is a well-known OpenCV library with great documentation. Our original code implementation, which can be found on GitHub, had tremendous issues. The lag between frame input and output would render the system useless in a moving vehicle. This caused more research into making the code more efficient, modifying the scanning area and frames scanned, and taking full advantage of the processor. It was also discovered that the GoPro, while having a USB port, does not interface with a PC as older GoPro’s had. There was no way to treat the camera as an external Web Camera therefore no way of viewing the feed on the PC directly. This required once again deeper research into solutions and was solved with the use of a capture card device.

For lane detection, a separate code (created by Caleb and Steve), made use of more OpenCV functions in order to detect lane markings. The code uses canny edge detection to detect the edge between the lane marking and the roadway pavement. Once detected, a line is drawn on the image highlighting the lane markings on the ground. In order to maximize the accuracy of the detection, the scanned area, or region of interest, is marked out from the horizon (street to sky) and the beginning of the hood of the vehicle. To further filter the false positive a filter was created to ignore lines that would be impossible to have been a road marking.

With the pedestrian detection, the lag encountered was reduced by implementing a multithreading solution. This allowed us to merge the lane tracking with the pedestrian detection, with a no major issues.

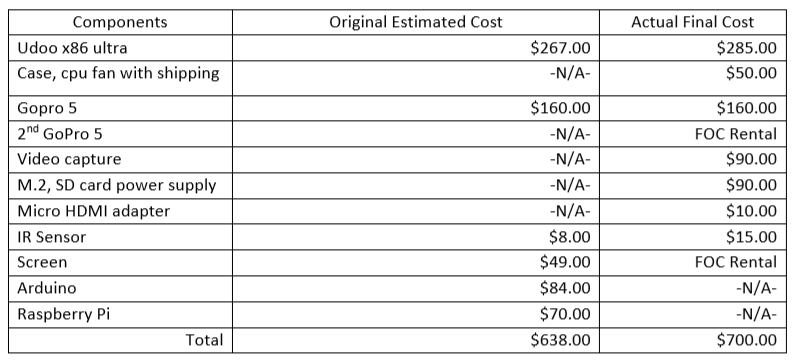


Fig. 2

# Results

Unfortunately, due to time, the project is not in the final stages as we had planned originally, and no in-vehicle testing was completed. However, we were able to run a few tests of prerecorded video footage, and we can run a live feed to detect and mark out pedestrians in real time. These tests were completed to check frame rates across different settings, devices and core counts. When testing on the Udoo, most videos would cause the entire system to become unstable. The Udoo was oddly enough able to run the real time detection in a somewhat stable state, no crashing, but at a major hit to frame rate, roughly 10.4 FPS at 720p, while the CPU usage was pinned at 100% maxed out speed at 2.56 GHz. The only video file that ran semi stable until ultimately crashing was the video titled “Fin.mp4”, the video ran at 7 FPS @ 720p, and once again the CPU was at 100% usage and at its max clock speed of 2.56 GHz. The Udoo is equipped with an Intel Pentium N3710, a low voltage processor with a base clock of 1.60 GHz and max boost to 2.56 GHz.

In order to see what results would have been on a much more modern, yet when compared very power-hungry CPU, the same testing was completed on an ASUS Laptop. The laptop is powered by an Intel i7 9750H, it has a base clock of 2.60 GHz and a max turbo of 4.50 GHz. All test videos were able to be completed, and the data can be found below (Figures 3 & 4). The focus with this processor testing was to see the how low of a core usage could be used before it became similar to the Udoo or crash. Live video testing resulted in 29.8 FPS @720p, with core usage at 100% and max clock speed of 3.62 GHz with all 12 cores (6 physical and 6 logical) permitted. The minimum before breaking was at 2 cores, at a max clock speed of 4.16 GHz we were able to achieve a modest but not very useful 15.3 FPS @ 720p.

Fig. 3



Fig. 4

The actual pedestrian locating, and lane finding/tracking success rates were inconclusive, much time was spent attempting to get the code up and running resulting in a loss of time for full testing. A quick script allowed us to record the total number of frames drawn and pushed onto the screen, this data was compared with the total amount of frames shown in order to get a percent of frames found in each clip. Two videos that were tested contained no actual pedestrians but resulted in 18% and 21% of the total frames containing boxes. Based on this data we can say that roughly 19% of the boxes found in all videos tested could be possibly false positives. We would like to have done further, more precise testing by count the physical amount of people in the videos and compare that to the total that are found through the program, but time has prevented this. As for the lane detections we planned to time the total amount of time the lane highlight marks appeared on screen and compare it to the total time that physical lanes were shown. Again, due to time we were unable to test this.

As for the actual testing procedure, the laptop and Udoo were loaded up with Windows 10, Python 3.7.3, OpenCV 4.1.1, our full code and the testing videos. In order to give the Udoo a best possible scenario, all explorer related processes were killed prior to testing except for PyCharm, which would be used to collect data as well as for setting core usage for the laptop testing. The laptop, explorer was left alone, but no other programs or applications were running, only essential windows processes.

# Discussion

Throughout the course of the project, as we expected we faced many setbacks, but many of them were not expected issues that arose from component issues. Those when combined with the issues we expected to face resulted in larger than though delays within the set schedule. The original timeline did contain a buffer for these issues, but they were mainly expected to be coding and testing delays caused by environmental issues like weather and cloudy fall/winter days. Because of that assumption, our buffer time was left mainly towards the end of the project timeline. Major issues we did encounter fell along the lines of time management / working around the teams scheduling resulting in project setbacks, hardware component issues, and financing.

On the hardware front, a lot went wrong. The original components were chosen based on cost, performance, form factor and reliability, but it appears we did encounter faulty hardware in the case of our first camera, and we underestimated the processing power needed. As for the reliability and faulty hardware, these things can happen and are unavoidable risks with any project. Thankfully we discovered that the school offers free rental equipment and by pure luck they carried the GoPro Hero 5, the same model we were initially using. This allowed us to diagnose the issue encountered and pinpoint that it originated within the cameras micro HDMI cable, and not any of the adapters, wiring or the capture card. The rental thanks to Jordan was able to be extended for most of the semester without need to return and rent again. A major hardware issue we faced was the limits of the Udoo board. Unfortunately, this issue was one we could have avoided with a bit more research and planning. It is clear now that the code should have been completed on a higher end PC prior to choosing and purchasing the board. If we had done it in this order, we would have seen that at the minimum requirements for life feed scanning at 30 FPS is a 4 core processor capable of sustaining a 4.05GHz clock speed, roughly the equivalent of an Intel i7 4790k (4 cores with turbo @4.40GHz). A final hardware setback I would like to mention, because it was a direct result of human error, was the loss of a solid state hard drive. At some point between moving the components between school and home, the drive was misplaced. I wanted to mention this because it is something that none of us would have expected to be an issue in the early planning stages.

Regarding the camera link between the Udoo and the GoPro, this again was an unexpected issue that could have been avoided with a bit more research. But it seems that the result we ended up with may have been the best-case scenario regardless. Thanks to the CamLink 4K device we were able to use this third party device to treat the GoPro as a USB webcam. This allowed us to not sacrifice quality by going with an actual web camera. The device is part of a family of video equipment used heavily by YouTube and twitch streamers. The final device used is the second one we tried. The original was a capture card made by the same company, typically used to pull feed from an external game console system for streamers to use. However, that device had a heavy delay which for streamers is not an issue as it can be fixed in software by delaying their other video and audio feeds, but that would not work for us as we needed to as close to real time as possible.

As for scheduling among the team, as most of us are about to graduate or in our second to last semester and have jobs, organizing time for the project became more and more difficult as the fall semester began. While we expected some issues, it was a bit more difficult than expected to balance work, home and school life with many other technical classes requiring work on various projects and labs throughout the semester.

Financially we technically did great and managed to stay underbudget (can be found in figure 2). But if we were to have waited and found the minimum requirements we needed for the processor prior to purchase, the price would have increased by a minimum of 300 dollars plus personal component donations (motherboard, ram, PSU). There is no affordable SoC that contains the baseline we needed. An Intel NUC would have cost more than the total we budgeted for the entire project, at 900 dollars for a i7 processor. This means a custom build would have been necessary to get the best performance per dollar we could.

Further research shows that we were on the correct track by ditching the lane sensor idea. Currently lane tracking systems found in modern production vehicles are capable of distinguishing lane markings, including dashed markings, with the use of a monochrome cameras or stereo camera setups. The most common found is monochrome cameras and they are implemented by Toyota and various other manufacturers. In the case of Toyota, their system “Toyota Safety Sense 2.0” is capable of both tasks that we set out to accomplish. Diving deeper into their systems and other manufacturers systems, a trend followed that led us to a company called MobilEye, which is owned by Intel. They are the current dominant name in car computer vision for safety and Autonomous Vehicles controlling nearly 70% of the market. MobilEye produces custom hardware for these systems, running their own software, but have promised in November 2019 that their next generation hardware will be open, called “Open EyeQ5”. However, there is questioning behind how open is open, as Intel doesn’t like to share. Other notable competitors in the market are Nvidia with their “NVIDIA DRIVE”, and ITD Lab who have gone all in with the stereo camera setup, which can be found in Subaru’s “EyeSight” safety features.

Toyota offers its system as standard safety across all vehicles and Subaru offers on all vehicles as well but at an additional cost for certain base models. While researching, it was noticed that, in Toyotas case, the system was offered on all but two cars, the 86GT and the GR Supra. Further digging found the system as an optional extra on the Supra models, while the 86GT has no mention of the system. The only difference between the 86GT and the rest of Toyotas lineup is that it is currently the only vehicle that has a manual transmission option. It appears that the system has some fallbacks when used in cars offered with manual transmissions, and it holds true across the board with Subaru listing that the option was only available on the WRX line if combined with the their CVT transmission.

# Conclusion

With the equipment limitations, unexpected challenges arose with the program and operating system. Ultimately, after further researching and seeking advice from other sources, we were able to improve on most of the issues with the code itself and get it efficient to be run on different hardware with minimal lag. However, these changes resulted in the inability to perform the in vehicle testing and didn’t meet some aspects of the original success criteria we set in the beginning of the project. The system was not able to be integrated within an automobile as the hardware could not support the required processing power needed for a reasonable detection with minimal lag. In addition, a higher budget would have been necessary to meet those requirements. As stated in the discussion, either an Intel NUC or a custom-built PC would have been necessary to meet the in-vehicle requirements.

## Lessons Learned

Learning Python, and how to utilize the OpenCV libraries is what we expected to be the most challenging, however python ended up being a simple and nice to use language. The library for OpenCV is huge, and is difficult to navigate at first, but once figuring out the general direction we wish to go in, it became a bit simpler to use. We have barely scratched the surface of OpenCV and while we would like to dive deeper into it, time is not on our side.

The issues that were brought on from the hardware limitations caused an unexpected need to understand, learn and attempt to implement a multithreaded and multiprocessing solutions and unfortunately due to time constraints we were unable to attempt a multiprocessing version. The interesting ways that different operating systems seem to handle the program, for example the lightweight Linux we tried used less resources but the lag in the detection was somehow worse than Windows 10, which we still cannot explain. Resizing and modifying scanning area in single frame, something that we did not expect to need to do but ended up being an absolute necessity. Realization that new high-end camera systems like a GoPro do not have a simple plug and play interface to view the output on a PC and figuring out a way to get them to interface with 3d party hardware.

## Future Work

Currently, we have no plans to continue working on this specific project together. However, certain group members have expressed interest within the area of computer vision technologies and we have seen some interest from other students that may want to continue work on this specific project and this area generally, and we greatly encourage it and are more than willing to help. There are many uses that the same theory and methods used here can be applied and some of us are interested in pursuing the area of study further. With consumer level devices able to do so much with computer vision, the possibilities of this being implemented on commercial grade hardware or even on server hardware would be amazing, almost like something you would expect to see in a Bond film. To those who wish to continue the project, we advise you to follow up on MobilEye’s Open EyeQ5 hardware. Keep in mind that the specifications that are listed for the device would make it seem to be highly expensive.

# References

1."Pedestrian Traffic Fatalities by State". Creative by Tony Fry Design, page 3-6. Published February 28, 2018.

2."Pedestrian Traffic Fatalities by State". Creative by Tony Fry Design, page 4-6. Published February 28, 2018.

3.“GHO | By Category | Road Traffic Deaths -Data by Country.” World Health Organization, World Health Organization, apps.who.int/gho/data/node.main.A997.

4.GoPro HERO5 BLACK. Retrieved from https://www.amazon.com/GoPro-Hero5-Black-E-Commerce-Packaging/dp/B01MT0TJOM/

5.UDOO X86 Ultra embeds two processors. Retrieved from estimate found from https://www.seco.com/misk/UDOO\_X86\_datasheet.pdf

6. GeeekPi 7 inch 1024 x 600 HDMI Screen LCD Display with Driver Board Monitor for Raspberry Pi. Retrieved from <https://www.amazon.com/GeeekPi-Screen-DisplayMonitorRaspberry/dp/B01L6YXZLY>

GitHub URL : <https://github.com/HUD-Car-System-CCSU>