Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Erlend Einmo

Automatic License Plate Recognition Performance on Low-Powered Mobile Devices

Master's thesis in Information Security
Supervisor: Lasse Øverlier
June 2022

NTNU
Kunnskap for en bedre verden

Erlend Einmo

Automatic License Plate Recognition Performance on Low-Powered Mobile Devices

Master's thesis in Information Security
Supervisor: Lasse Øverlier
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Automatic License Plate Recognition Performance on Low-Powered Mobile Devices

Erlend Einmo

CC-BY-NC 2022/06/01

Abstract

Automatic License Plate Recognition (ALPR) is a technology that might play a bigger part in our everyday lives than we realize. It is the reason we can now pay for parking using an app and no longer need a parking receipt when parking at the local shopping center. The reason for all this is the fact that mobile devices have become increasingly powerful, and our technologies more efficient. These technological advances make ALPR more accessible than ever. One of the questions answered in this paper is how accessible has this become? In this research project, a series of experiments are performed using open-source ALPR implementations on mobile and less mobile devices in order to find out how well they perform. The results show that mobile devices costing $50 USD perform ALPR well enough to be a viable option for a small-scale ALPR system. As part of our experiments, we look at the efficiency of these devices and show that mobile devices are both more cost-effective and power-efficient for ALPR than their more powerful counterparts. Another question we answer is why ALPR sometimes fails.

We create our own dataset, adhering to privacy regulations, as well as testing that the ALPR implementations used in this project have the potential for being used in a privacy-friendly manner.

Sammendrag

Automatisk Bilskiltgjennkjenning ALPR er en teknologi som spiller en større del i hverdagen vår enn vi kanskje tror. Det er grunnen til av vi kan betale for parkering med en app, slipper å ta parkeringsbillett når vi parkerer ved det lokale kjøpesenteret. Grunnen til dette er at mobile enheter har blitt stadig kraftigere, og teknologiene mer effektive. Disse teknologiske fremskrittene har gjort ALPR mer tilgjengelig enn noen sinne. En av spørsmålene stilt i denne oppgaven er hvor tilgjengelig har det blitt? I dette forskningsprosjektet utfører vi en series med eksperimenter med bruk av tre ALPR implementasjoner på mobile- og mindre mobile enheter for å finne ut hvor bra disse yter. Resultatene viser at mobile enheter som koster 50 amerikanske dollar utfører ALPR godt nok til å være en god kandidat for små-skala ALPR systemer. Som del av eksperimentene våre ser vi på effektiviteten til disse enhetene, og viser at mobile enheter både er mer kostnadseffektive og mer energieffektive enn større enheter. Et annet spørsmål vi svarer på er hvorfor ALPR innimellom feiler.

Vi har også laget vårt eget datasett, som følger personvernsreglement, i tillegg til å teste om ALPR implementasjonene brukt i prosjektet har potensiale for å bli brukt på en personvernsvennlig måte.

Contents

Figures

Tables

Code Listings

Acronyms

ALPR Automatic License Plate Recognition. iii, v, 1–5, 7–11, 13–17, 23, 24, 26, 27, 31, 35, 37, 38, 40, 45, 46, 50–53, 55–57, 59, 60, 62, 63, 65–67

CPU Central Processing Unit. 20

CSV Comma Separated Values. 31, 33

GDPR General Data Protection Regulation. 3

GPU Graphics Processing Unit. 20

JPEG Joint Photographic Experts Group. 30

NSD Norwegian Centre for Research Data. 25, 26, 57

OCR Optical Character Recognition. 7, 9, 39, 59, 66

SSH Secure Shell Protocol. 19

USB Universal Serial Bus. 22

VNC Virtual Network Computing. 31

WSL Windows Subsystem for Linux. 19

Chapter 1

Introduction

Automated reading of license plate contents from images is a technology that has been around since 1976 [1], and ever since, the technology has gotten better, faster, and more portable. This project regards the topic of Automatic License Plate Recognition using low-power, portable devices. The background for this topic is the feasibility and performance of mobile low-power devices for such a task while also maintaining compliance with Norway's privacy laws. Portable mini-computers and mobile devices are becoming cheap and available to most people, allowing hobbyists and enthusiasts to develop solutions for their projects. ALPR implementations are commonly used by businesses and city services. Some typical

Figure 1.1: ALPR detection example. Anonymized.

uses for mobile ALPR systems are toll systems on private roads, vehicle scanning by parking attendants, or police searching for specific vehicles or looking up information on one. In Norway, ALPR has also become more prevalent on ferries, and the Norwegian Public Roads Administration utilize ALPR in section speed con-

trol [2]. One of the features that make tiny computing devices like the Raspberry Pi the most interesting is the possibility for all sorts of add-ons and modifications supported by the devices. For example, powering the device off solar power and a battery in the middle of the woods where no power infrastructure is available. Such devices can also be fitted with mobile-data connections to provide network connectivity where this is needed.

ALPR solutions often incorporate machine learning algorithms to identify objects, such as vehicles and license plates, and read the actual characters of the license plate. How quickly the plate needs to be recognized depends heavily on the use case of the implementation. Implementations where police want on-the-fly information, a toll gate where the plate needs processing before opening, or any implementation where the volume of vehicles is high, are examples where rapid processing will be of higher importance. While for example, a system at a toll road that sends invoices after the fact allows for slower processing times, as the invoice is not sent until the next day. Precision is another metric that will have varying importance. However, reading the correct plate is essential in many implementations, as reading the actual plate in an automated fashion is the entire point of having an ALPR system. Correct detection is crucial in cases where money is involved. These two parameters, speed and precision are influenced by a few factors. The first one is hardware. The device used to perform the needed calculations is a significant factor in how long the process will take. A more powerful device will naturally be able to process more data quicker and therefore perform the task quicker. Another factor is the algorithms and methods being used to perform each of the actions of the ALPR. More efficient algorithms will perform quicker and more precisely, which is especially important for low-power devices.

A faster method will typically have lower precision and vice versa; therefore, a trade-off between speed and accuracy is the reality. Having powerful hardware and efficient algorithms and methods will allow better flexibility regarding the trade-off between speed and precision.

1.1 Keywords

Automatic License Plate Recognition, ALPR, Automatic Number Plate Recognition, ANPR, License Plate Recognition, LPR, Raspberry Pi, NVIDIA Jetson, Computer Vision, Machine Learning, Portable Devices, Optical Character Recognition, OCR, Privacy, License Plate, Image Resolution, Image Resolution Performance, ALPR Performance, Mobile Device Performance, Power Efficiency, Price to Performance.

1.2 Problem Description

Complete automatic license plate recognition systems are too expensive for certain implementations. These are the systems one can find at entries to parking lots and similar commercial implementations. Such implementations often come with

multiple cameras utilizing ALPR, for example, to bill parked vehicles automatically. In situations where you want a remote implementation, having a power-efficient device might be more important. Nevertheless, with computing technology becoming smaller and more efficient, mobile platforms like the Raspberry Pi, NVIDIA Jetson, and smartphones are becoming powerful. This increase in computing performance for relatively cheap and portable devices means that they become capable of increasingly computationally complex tasks. The question, then, is whether these devices have reached a point where they can effectively be used to perform machine learning and other relevant processing well enough to perform ALPR?

As ALPR implementations require imaging and computer vision to function, cameras and image parameters/attributes can affect performance. Creating a custom dataset will allow for more images to test on. A bonus is control over cameras used, image parameters, and attributes.

1.3 Privacy Considerations

With the introduction of GDPR, Norwegian and EU laws are strict regarding personal information and, by extension, the processing of license plates [3][4], ALPR in these regions needs to be performed in a privacy-friendly manner. Using online ALPR solutions introduces a variable in the form of sending information to a third party. Therefore, the security and privacy of the data are outside of the control of the person using the solution. A serious company responsible for privacy might also be regarded as a positive; however, in cases where one wants to ensure that privacy and security are upheld, it is an uncontrollable variable. Having an implementation running locally on a device can allow for the creation of privacy-friendly solutions around that ALPR software.

With the processing of personal information being so strict, this also applies to the research in this project. Therefore, all processing and data collection is performed in accordance with Norwegian Centre for Research Data (NSD) guidelines. Any example images used in this thesis are included with consent from the vehicle owner.

1.4 Justification, Motivation, and Benefits

With the various device types that can be used for ALPR implementations, finding the one that performs within the required performance parameters for implementation can be difficult. Having comparable performance metrics for different hardware as well as open-source implementations can aid in finding the most cost-effective solution for the task at hand. Mobile devices are low-powered and can easily be powered using batteries, making them even more portable. Power consumption and efficiency are also factors for these kinds of applications.

With this information, people looking to create an ALPR system can utilize this resource to help decide what suits their needs.

1.5 Research Questions

To concretize the project's main goals, the research questions and accompanying sub-questions are listed below.

- **1.** How well does mobile hardware like the Raspberry Pi and NVIDIA Jetson perform with an ALPR implementation?
  1a) How well do they compare to more powerful devices like laptops and desktops?
  1b) What platform delivers the best price to performance ratio?
  1c) How well do the open-source implementations perform?
  1c.1) Are the open-source implementations privacy-friendly?
- **2.** What physical and technical factors stand out with regards to affecting performance?
  2a) How much do image parameters like image resolution affect ALPR performance?

1.6 Contributions

This section will highlight a list of this project and the thesis' contributions.

Comparative Performance

Directly comparable performance metrics for mobile, low-power devices, showing the capabilities of such devices. With benchmark performance metrics from more common, relatively high-powered devices. In addition to comparable performance of different ALPR software implementations, and information about implementation of these on various platforms.

Factors Affecting Performance

An investigation into various technical and physical factors that affect the performance of ALPR. Helping put forward what factors should be considered and mitigated when setting up an ALPR system.

Dataset

A custom dataset is created, with detailed information about the contents and annotation.

Privacy Consideration

Privacy considerations in multiple contexts. First, directly for ALPR implementations and their handling of data. And secondly, considering European and Norwegian regulations surrounding license plates as personal data, and the obstacles this entails regarding research projects.

Efficient Solutions

Matching the performance of devices to price and energy consumption, tying results regarding price-to-performance. This information can help choose economically responsible and sustainable choices when planning an ALPR system.

1.7 Thesis Structure

The thesis follows a slightly expanded IMRaD structure, with an introduction, background, methodology, experiment, results, discussion, and conclusion chapters. The modifications give a more natural segmentation to the project's presentation and prevent chapters from dragging on forever. Chapter 1 contains the introduction, which introduces the topic as well as the reason for and goals of the project. Chapter 2 presents background information on the topic. The processes and technologies used in ALPR are introduced, alongside some existing work related to the topic of ALPR on Mobile Devices. Chapter 3 presents the broad methodology utilized to answer the research questions for the project. Chapter 4 continues similarly with the experimentation and experimental setup used in the project. This section presents the technical details behind the technologies and devices used in the project and the implementation process of the ALPR software solutions. Also presented in this section is the experimentation process and result analysis. We created this process to gather scientifically sound results, metrics, and datasets.

Chapter 5 relates to the results gathered during the project, and the analysis and discussion of these. The chapter starts with broad and more general results, presented using graphs, tables and figures to present and compare data in an easily understandable and visual manner. After this introduction, the chapter focuses on narrower topics, with more specific comparisons and analysis of data obtained from the experiments. This is where one can find common reasons for failed detections and further investigation of the impact of image resolution on ALPR performance.

Chapter 6 is the discussion chapter. Here, the process of the project as a whole and the various parts of the project are discussed. Finally, chapter 7, where conclusions for the thesis and project are drawn, and suggestions for future work are presented.

Extra content and resources are appended to the end of the thesis, in the appendix.

Chapter 2

Background/Technology

2.1 Automatic License Plate Recognition

Automatic License Plate Recognition (ALPR) is the process of utilizing computer vision to read the contents of a license plate. ALPR appears under a few different names; LPR (License Plate Recognition) and ANPR (Automatic Number Plate Recognition) are a few commonly encountered. This chapter presents the typical ALPR process and information about the steps involved in turning an image of a license plate into characters in a file.

2.1.1 ALPR Process

ALPR implementations are not all precisely the same; there are some variations to the process depending on the implementation. Some steps appear in most, if not all, implementations. Naturally, some steps are required to finish the task at hand. The two most prominent parts of the process are *license plate detection* and *Optical Character Recognition*. These two processes detect where the license plate is in the image and read the characters off it. Accompanying these can be various stages of pre-processing, character segmentation, and post-processing steps. As the ALPR process requires multiple steps, the process typically follows a pipeline containing each step of the process. A general ALPR pipeline can be seen in figure 2.1.

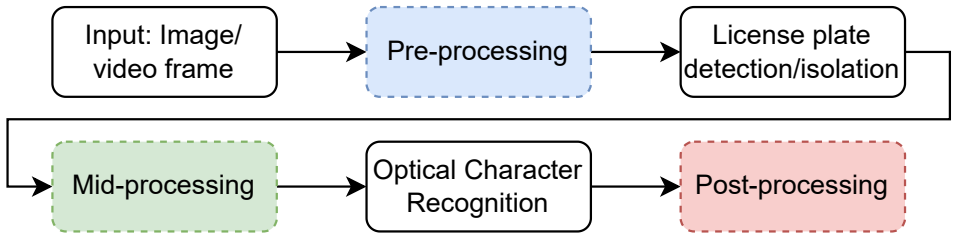


Figure 2.1: ALPR Pipeline visualized. Dashed boxes are optional steps.

2.1.2 Pre-processing

The pre-processing serves to simplify the rest of the steps of the pipeline. Image modifications can be part of the pre-processing. For example, downscaling the input image in order to speed up processing. Another example is vehicle detection and segmentation, which ALPR-Unconstrained utilizes. This process detects all supported vehicles in the image, segments the part of the image that contains the vehicle, and sends just that portion of the image to the next step of the process.

2.1.3 License Plate Detection

License plate detection is one of the fundamental steps in any successful ALPR implementation. While it, in theory, is possible to apply OCR to an image without any pre-processing, this is quite likely to encounter a whole host of false detections.

There are a few various approaches to finding license plates in an image, but typically an algorithm or neural network models are utilized to look for the distinct shape of license plates. OpenALPR utilizes the Local Binary Patterns (LBP) algorithm to locate potential license plates [5]. ALPR-Unconstrained utilizes machine learning in the form of a Convolutional Neural Network (CNN). More specifically, a network called Warped Planar Object Detection Network, designed to detect license plates on warped surfaces [6].

2.1.4 Mid-Processing

Some implementations apply processing between the detection of the license plate and OCR. These steps typically aim to improve OCR performance by processing the license plate area to make it as easy as possible for the OCR.

False-Positive Detection

One of the mid-processing steps that can be applied is a check to see whether the detected license plate area actually is a license plate. OpenALPR does this by performing binarization on the image, which turns the color data in the affected area to binary, or in other words, black and white. The next step applied is a character analysis, where the binarized image is scanned for potential license plate characters [5]. If none are found, the plate is discarded, which saves processing time by not processing false detections later in the process.

Plate Realignment

One of these steps, utilized in one way or another by all three implementations in this project, is "deskew" or plate realignment. Plate realignment is the process of aligning license plates with the image plane, removing skew if the license plate is at an angle. The CNN used for license plate detection in ALPR-Unconstrained incorporates this step in the detection process, as the WPOD-NET model applies a transformation to the detected license plate just after detection [6].

OpenALPR performs edge detection to find the edges of the license plate before re-aligning the license plate [5].

Character Segmentation

Character segmentation is the process of detecting and isolating each individual character. This process can also be implemented as part of the OCR, as it improves the performance of OCR. The character segmentation utilized in ALPR-Unconstrained is part of the Optical Character Recognition, using a modified You Only Look Once (YOLO) object detection system [6].

A common way of performing character segmentation is utilizing vertical histograms along the detected plate area. OpenALPR utilizes this method for character segmentation [5]. This projection shows which portions of the area contain the most features/objects. Prior to utilizing histogram object detection, applying preprocessing to the license plate can increase performance, especially in challenging scenarios like noisy images [7].

2.1.5 Optical Character Recognition

Optical Character Recognition (OCR) is the process of utilizing computer vision to read characters from an image. In the case of ALPR, it is typically done utilizing machine learning algorithms to determine what characters are most likely matches for the characters on the given image. A rudimentary approach to OCR is template matching [8]. This approach uses character segmentation and then attempts OCR on these segments by comparing the segment area with a set of character templates. The character template with the highest likeness gets chosen as the likely correct character.

Another method for OCR is using Extracted Features [9]. This method is typically faster, as it only focuses on the pixels of features within a segment. According to research by Lubna et al., implementations utilize Support Vector Machines for character detections using this method [9].

A common OCR method for open-source software is to use the open-source library Tesseract-OCR [10]. Tesseract was created by Hewlett-Packard Laboratories and has since been developed by various companies, including Google from 2006 to 2018. Different versions of Tesseract utilize various methods for OCR. Tesseract 4 utilizes Long Short-Term Memory neural net and a focus on line recognition for OCR [10].

The combination of the characters with the highest confidence are combined and presented as the result. The exception is if there is post-processing applied.

2.1.6 Post-Processing

Applying post-processing to results can help choose the correct plate by analyzing multiple alternatives for the correct solution. Post-processing in ALPR is essentially the sorting and choosing of the correctly detected license plate.

Confidence Checking

A simple form of post-processing is comparing the confidence of each of the presented results and choosing the result provided with the highest confidence. For solutions where a single license plate is required, like for a parking attendant, applying confidence checks for vehicle detection as well can increase the chances the right vehicle gets chosen, as vehicles further back in the image will typically return smaller confidence. While suggestions with high confidence are chosen, suggestions with low confidence levels can be discarded. This sort of post-processing is implemented by OpenALPR, where the desired number of detections with the highest confidence can be returned to the user [5].

Pattern Matching

Pattern matching is a post-processing step where license plate formats are utilized to choose likely correct detections. OpenALPR has this type of post-processing as an optional step [11]. Pattern matching utilizes knowledge about the character pattern of license plates that are likely to be encountered. For example, an implementation utilizing this in Norway would look for the pattern for standard Norwegian license plates of two letters followed by five numbers. If any of the proposed detections match this pattern, they can be prioritized. Personalized plates could potentially break with this pattern and lead to problems with this post-processing method. This type of post-processing was applied by S. T. H. Rizvi et al., using the format of Italian plates, resulting in a full plate accuracy increase of 13 percent [12].

2.1.7 Factors Affecting ALPR

P. Mukhija et al. have surveyed the topic of factors affecting license plate recognition. They provide a list of challenges relating to ALPR from an Indian standpoint [13]. P. Mukhija et al.'s research provide a list of external and internal challenges to the ALPR process. External factors like damaged license plates, obstructed license plates, and lighting conditions. And internal factors like hardware and image resolution.

2.1.8 ALPR Pipeline for OpenALPR

The pipeline for one of the software implementations used in this project in its default state can be seen in figure 2.2. In the figure, we can see how there are five steps of mid-processing applied and a single post-processing step.

2.2 ALPR Performance on Mobile Devices

Previous research exists on the topic of Automatic License Plate Recognition on mobile devices. Publications relevant to the topic of this project are presented

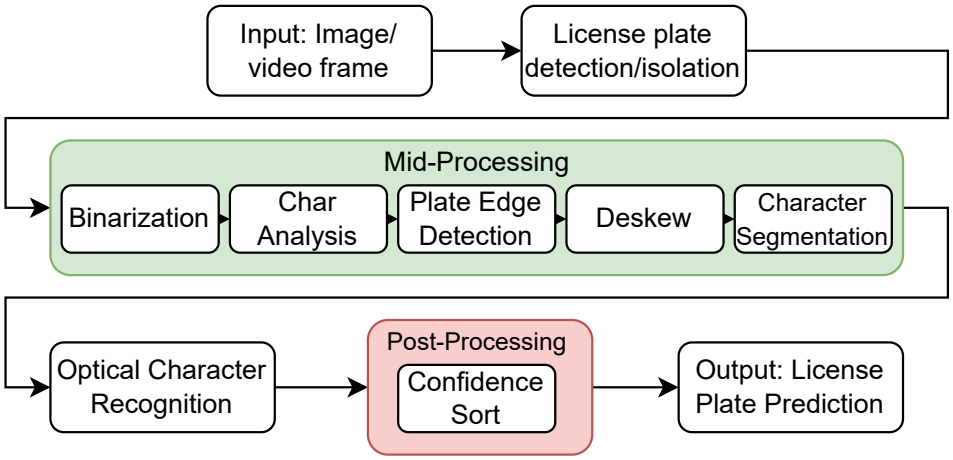


Figure 2.2: ALPR Process pipeline utilized by OpenALPR.

here.

A master thesis written at NTNU related to the topic was published in 2017. In that thesis, the researcher Peter K. Ringset created an ALPR implementation for an iPhone 5 [14]. That project achieved an accuracy of up to 87%, with a processing time of only 50ms.

Another implementation on the Android platform from 2016 achieved 96-98% detection accuracy, with processing times of around half a second [15].

S. Fakhar A G et al. created an implementation for the previous generation of the Raspberry Pi, the 3B. Their implementation resulted in an 85% successful recognition rate, with image-processing times of up to three seconds [16].

S. T. H. Rizvi, et al. performed tests on GPU-equipped mobile platforms with a custom ALPR implementation on Italian license plates [12]. They applied model-simplification for the mobile platforms, resulting in quick ALPR detections of 492 ms for the NVIDIA Jetson TX1, and approximately 550 ms for the Nvidia Shield K1 tablet.

Chapter 3

Methodology

3.1 Literary/Resource Research

While the brunt of the literary research for the project was conducted during the pre-project planning stage, there is always the potential for more useful information to be found. The main portion of the research conducted in the pre-project was searching for potential ALPR-implementations and learning about various methods utilized to perform ALPR. During the project, we will gain more knowledge on the topic and can therefore use this knowledge to perform better searches for relevant information. A secondary objective for this part of the methodology is the potential of finding more potential open-source ALPR software that can be implemented and experimented on during this project. Literary research in itself is a minor part of the project.

The research process will be ongoing as we learn about new technologies, techniques, and other information encountered during the project. This research will be conducted by doing keyword searches for specific topics and technologies to obtain more information.

3.2 Experimentation

In order to answer the research questions for this project, a series of practical experiments will be performed. These tests will attempt to answer the questions by performing ALPR using mobile devices and measuring their quantitative performance. The same tests will also be run on more powerful devices that are less mobile, where performance results will be used as a comparative benchmark. Different software solutions will be implemented and tested on as many devices as possible to test their performance and versatility and to confirm whether such open-source software can be run in a privacy-friendly manner. Three software implementations, all available on GitHub, were chosen for the project. UltimateALPR [17], OpenALPR [18], and ALPR-Unconstrained [19]. Implementing all three software implementations on as many devices as possible allows us to gather a large

number of data points from many experiment variations.

Experiments are conducted according to a pre-defined process. This process ensures that the data collected is as similar as possible across all devices and software implementations and eliminates potential third variables that can influence the results. Detailed information about the experimentation and setup used can be found in chapter 4.

3.2.1 Privacy / Offline Capability

In order to test whether the various implementations can operate in a privacy-friendly manner, the capabilities for offline computation will be tested. This test will be conducted by performing ALPR on devices with their internet connectivity disabled. These tests will ensure that the devices can perform the entire process on the device without sending data elsewhere.

3.2.2 Metrics

Two categories of metrics are measured to evaluate the performance of each of the implementations. These categories are accuracy and speed. The most important metrics will be the accuracy metric of correctly identified plates, as well as the time taken to get that result. Each measurement is made for each dataset on each device, using each implementation.

Accuracy

For this project, the measurement of accuracy is done in multiple ways, as there are several different stages of the ALPR process that can be interesting to look at with regard to accuracy performance. Each of the metrics used to measure accuracy can be found listed below.

- Plate Recognition Rate
  Plates recognized entirely correctly.
  Also referred to as "Correct Plates" (Corr. Plates)
- Plate Detection Rate
  Images where a plate was found at all.
- Character Recognition Rate
  Characters correctly recognized, where plates were recognized.

Calculation The calculation of plate recognition rate can be seen in formula 3.1.

$$\frac{|CorrectPlates|}{|DS|} * 100 \tag{3.1}$$

Calculating license plate detection rate uses the count of images where one or more license plates were detected and dividing that by the total number of images across all the datasets. The equation used can be seen in equation 3.2.

$$\frac{|Detections|}{|DS|} * 100 \tag{3.2}$$

Calculation of character recognition rate uses equation shown in equation 3.3. The number of correct characters is divided by the total number of characters on detected plates.

$$\frac{|CorrectCharacters|}{|DetectedCharacters|} * 100 \tag{3.3}$$

Speed

Speed is measured using timing. This metric looks at the time elapsed between the time an action has started and ended. The actions measured for this project were the time taken for the entire loading and detection of each dataset, initiation of the ALPR engine, and time taken for each detection.

Engine initiation is measured to see the time taken for the engine to load on the devices. Timing of the entire dataset includes the initiation of the ALPR implementation, loading and detection of each image, and writing the resulting plates to the output file. For each individual plate, the time includes the time from a plate is presented to the program, and a result is written to a variable.

- Init Time
  Time taken for the engine to initialize (where applicable).
- Detection Time
  Time taken for a plate to be detected and read.
- Total Time
  Total time taken for detection of an entire dataset.

Information on how the timing is implemented in code is located in section 4.3.4.

Calculation

Detection Time is the average of all detections across a dataset. Average detection time is calculated differently, depending on the implementation. Calculations are either made using a python function or a spreadsheet; more information about this in section 4.6.1 in the next chapter.

For UltimateALPR and OpenALPR, we have access to the time for each plate detection and gather a list of these. We can calculate the average value with formula 3.4, where T is the set (list) of all detection times t.

$$\frac{\sum_{i=1}^{|T|} t_i}{|DS|} \tag{3.4}$$

We do not have access to the detection time for each detection made with ALPR-Unconstrained, as it processes the entire dataset for each step of the process before moving to the next. However, we can calculate an average detection time using the total time taken for detecting all images in the entire dataset. The calculation can be seen in formula 3.5

$$\frac{TotalTime}{|DS|} \tag{3.5}$$

3.2.3 Secondary Experiment

A secondary experiment was conducted using resized versions of the custom dataset. The reason for this was to help answer the research question regarding resolution and its impact on ALPR performance. While the custom dataset and Olavsplates datasets contain images of different resolutions, there are also several other factors different between these images. Performing secondary tests using resized datasets allows us to control these third variables. More information about how this experiment was conducted can be found in section 4.5.2.

3.3 Data Collection

Data collection, in this case, is the collection of datasets of images containing vehicles with license plates visible. Despite all implementations used in the project coming with pre-trained models, a dataset is still needed to benchmark the implementations. Datasets consisting of vehicles with visible license plates in various conditions are desirable. The focus of this project is the use of ALPR in Norway, which makes Norwegian license plates the focus. EU plates are also relevant, as encountering vehicles from other EU and Schengen countries are not unlikely. Plates from most countries in the EU/Schengen area are pretty similar, likely making the models transferable.

While there exist datasets of EU plates online, they are not typically easy to acquire. We contacted the owners/holders of several datasets that were seemingly available. However, they were not allowed to share these due to privacy concerns. License plate numbers in the EU are regarded as personal information under GDPR, making sharing such information troublesome [3][4]. Thus, we will create our own dataset to test the implementations further. Creating a custom dataset will also allow us to create more edge-case scenarios to compare the implementations on more grounds than simply accuracy and speed. Further, we can control the images to be taken in a particular manner, at certain times of day, and with the potential for different weather conditions.

3.4 Result Analysis

After conducting the experiments and gathering the results, these results need to be analyzed. The results are to be collected in a spreadsheet to allow for calculations, comparisons, and other analyses to be conducted visually and practically. The spreadsheet will be standardized across devices, at least as far as practically possible. An example collection of runs from the spreadsheet can be found in appendix A.1.3 Having this standardization allows for consistent comparisons between the experiments with all the various combinations.

The data analysis will compare quantitative results in the form of time measurements for speed and various rates of correct detections for accuracy. We can look at common factors in incorrect or failed detections by examining the images where ALPR did not give a correct result. This examination of images can help uncover which license plate types fail most commonly, characters being misinterpreted. The analysis is the part of the project where most of the research questions will be answered. Here we will find answers to which physical and technical factors affect ALPR performance and all performance comparisons between devices. Further information about how the analysis is performed can be seen in section 4.6.

Chapter 4

Experimentation and Experimental Setup

In order to answer the research questions, complex experiments using various devices, cameras, software implementations, and datasets are conducted. These experiments and the setup behind these will be presented in this chapter.

4.1 Environments

All implementations are run on various Linux interactions. The two mobile devices run native Linux distributions, while the desktop and laptop utilize Windows Subsystem for Linux (WSL) to run Ubuntu 18.04 virtualized on top of Windows 10.

The scripts used to test the implementations for this project are programmed using Python. While consisting of python scripts, ALPR-Unconstrained is run by calling a bash-script. This script calls each of the processes' scripts and sends data along to the next step using this bash script. Python is not the fastest programming language, but it makes the implementation process the quickest for the sake of the project and researcher. As all implementations are implemented using Python, the results are still comparable with one another. Faster results could likely be achieved using more low-level languages like C.

The scripts are called either directly through the terminal or using an SSH connection in order to limit unnecessary resource usage.

4.2 Devices

Choosing devices for performance testing was done according to a few criteria. The project's focus is mobile devices, which is a rather extensive category of devices. Varying performance and price points are desirable. Cost and accessibility are also factors, especially due to the large semiconductor shortage plaguing the world for the last year [20].

With the impressive performance of modern mobile devices, even running machine learning tasks has become quite feasible. This goes for both devices that only have a CPU present, but perhaps even more prominently GPU-enabled devices. This section presents the devices used in this project.

4.2.1 Mobile Devices

Some of the large appeal of these mobile devices is their low power consumption and the large number of possibilities for add-ons and modifications. There is a whole host of add-ons by third parties and even the Raspberry Pi Foundation. On their site, a wide variety of add-ons are available for purchase, like Raspberry Pi Camera modules, a Touch display, and Power-over-Ethernet add-ons, to mention a few [21]. These and other third-party products and a wide variety of sensors and other devices can be connected to the Raspberry Pi and NVIDIA Jetson via the General-Purpose Input/Output interface. The GPIO supports a variety of interface options [22][23]. In addition to GPIO, the pinouts on the boards include power delivery for both 3.3V and 5V, making powering external add-ons and devices easily accessible.

Raspberry Pi 4B

The Raspberry Pi is a tiny, USB-powered, single-board computer with a footprint approximately the size of a credit card. Featured on the Raspberry Pi is a set of pins, the GPIO (General-Purpose Input/Output) interface, allowing for a large variety of physical modifications and attachments to be added to the device. Further, the Pi has a 2-lane MIPI CSI camera port, allowing for the usage of Raspberry's own Pi Camera modules. Full-size USB 2.0 and 3.0 ports will also enable the use of a wide variety of USB cameras with Linux driver support.

NVIDIA Jetson Nano

The NVIDIA Jetson Nano Developer Kit is a tiny computer similar to the Raspberry Pi but has the added benefit of a GPU attached. It is slightly larger along all dimensions yet still plenty portable. Another benefit of the Jetson Nano is the ability to run in a low-power mode, drawing only 5 watts, which could be helpful in particularly power-constrained scenarios. Onboard the Jetson Nano is the same set of pins as on the Raspberry Pi, enabling compatibility with many of the same add-ons [23]. The presence of full-size USB ports and video outputs in the form of HDMI and DisplayPort is also convenient.

4.2.2 Computers

Desktop Computer

The researcher's personal desktop computer was utilized as a benchmark. It is a custom built computer with a powerful processor and graphics card. The perform-

Figure 4.1: NVIDIA Jetson Nano used in this project.

ance could be comparable to running the service on a GPU-enabled server.

GPU-Equipped Laptop

An MSI GL62M gaming laptop was used as a less-portable but still mobile alternative. This laptop is equipped with a modest NVIDIA GTX 1050 graphics processor, allowing for hardware acceleration using the GPU where applicable. It is worth noting that this laptop is approaching five years old at the time of this project.

4.2.3 Device Specifications

Table 4.1 contains a list of key device specifications for the devices used in this project.

4.2.4 Cameras

The cameras used to capture images for the Custom Dataset were the Raspberry Pi v2 camera and a Huawei Mate 20 Pro smartphone. Further camera specifications can be found in table 4.2.

Huawei Mate 20 Pro

The camera setup for the Huawei Mate 20 Pro is quite straightforward. Images were taken using the smartphone's default camera application and the phone's main camera. Both horizontal and vertical orientation was used.

Raspberry Pi Camera Setup

The camera setup for the Raspberry Pi included the Raspberry Pi 4B, with the camera and the Raspberry Pi Touch Display. These add-ons and the device were all encompassed within a 3D-printed case to allow for easy handling and portability.

Table 4.1: Key device specifications.

Spec	Desktop	Laptop	Jetson Nano	Raspberry Pi
CPU	AMD 5900x	Intel i5-7300HQ	ARM A57	ARM A72
CPU Cores	12, 24thread	4	4	4
CPU Speed	4.8 GHz	3.5 GHz	1.43 Ghz	1.5 GHz
GPU	NVIDIA RTX 2080 Ti	NVIDIA GTX 1050	NVIDIA 128-core Maxwell	N/A
CUDA Cores	4352	640	128	N/A
GPU FLOPS (FP32)	13400 GFLOPS [24][25]	1862 GFLOPS [26]	235.8 GFLOPS [27]	N/A
RAM	32GB DDR4	8GB DDR4	4GB LPDDR4	4GB LPDDR4
Power*	550 W	150 W	5/10 W**	15 W

* Maximum system power draw. ** Depending on power mode.

Table 4.2: Camera specifications.

Spec	Pi Camera V2	Mate 20 Pro
Still Resolution	8 Megapixel	40 Megapixel
Image Resolution*	3280 x 2464	3648 x 2736
Image Density	72 dpi	96 dpi
Optical Size	1/4"	1/1.7"

*Used during data collection

The setup can be seen in figure 4.2, notice the cutout on the back of the device for the camera. To power the setup, we used a power bank providing power across a high-quality USB cable at 2.4 amps, 5 volts, which was enough power for this setup.

In order to capture the images, a short script was created. The script is located in appendix A.2.1. This script first counts the number of images (files) in the destination folder. Next, the `raspistill` command that comes pre-installed with Raspbian [28] is called using `os.system()`, with the previously counted number iterated once as the filename. This approach was chosen as the pre-existing `raspistill` command functions well enough for the purpose. The image is stored in the desired folder as a standard JPEG image at the default resolution. A desktop-shortcut was created to easily call the script when outside collecting data. The script opens a terminal and calls the script using Python.

Figure 4.2: Image of Raspberry Pi Camera setup.

4.3 Implementation

Setup and programming were conducted during this project in order to set up devices, equipment, and integrations correctly to collect the results needed to answer the research questions. Further programming was done in order to parse results and generate statistics. The ALPR implementations for this project are all python-based. Some are written in different languages, but all have a python-interface.

4.3.1 Requirements

The software used in this project should be available as an open-source ALPR solution. The software needs to be able to be used in a privacy-friendly manner. In other words, it needs to be able to run locally on the device, without having to send images or license plate information off the device to perform ALPR. A python interface is also a big preference, as we are most familiar with this programming language. Compatibility with multiple platforms is needed, as the software needs to be tested across platforms in order to be able to compare results.

4.3.2 ALPR Software Implementations

The three ALPR open-source implementations chosen for this project are ultimateALPR [17], OpenALPR [18], and a Python3-port[29] of ALPR-Unconstrained [19][6].

Both UltimateALPR and OpenALPR have paid alternatives, where a solution more tailored to different scenarios is available. Rekor, the creators of openalpr, even has online services, like their CarCheck Vehicle Recognition API [30]. But as one of the requirements for this project is to run the software locally on the

devices, this is not relevant to the project.

Implementations were relatively straightforward for the first two, as their respective GitHub releases are built like SDKs. Scripts were written for each of them that first initializes the relevant ALPR engine, then feeds it images from a given directory one by one. The results from these are returned as JSON, which is then parsed for the first license plate and presented to the user, as well as being written to a CSV file. Each CSV file has a unique filename, which follows a standardized format: `device_software_dataset_run#.csv`. This CSV file contains the engine initiation time, followed by one line for each of the supplied images. Each line contains the original filename, the result given by the ALPR engine with the highest confidence, and the time taken for the image to be processed. The timing is done outside the ALPR engine, in the script itself—more information on timing in section 4.3.4.

The third software, ALPR-Unconstrained, works slightly differently, as the software is based around entire directories rather than single images. Entire directories are loaded, and each step is executed for the entire directory before going to the next step. This process is done through python scripts being called in a bash script. Therefore, timing each plate detection is complex, and an average of the entire detection time is used as a substitute.

The code for the experiment implementations can be found at GitHub, note that these are subject to change as time moves on and maintenance and improvements are added. UltimateALPR, alongside installation instructions and required files at [1]. OpenALPR just contains the test script, and requires OpenALPR installation alongside [2]. Installation instructions (For Ubuntu-Linux) can be found at OpenALPRs own GitHub [3]. ALPR-Unconstrained ended up bewing timed using a bash feature (see section 4.3.4), and the Python3-port used can be found at GitHub [4].

4.3.3 Resolution and Image Statistics

In order to calculate statistics about image sizes for entire datasets, a python script was written. `imgSize.py` can be found in appendix A.2.2, listing A.2 The script uses the OS package to walk through entire datasets, and Pillow to gather various information about each of the images in the dataset. Information gathered consists of image resolution (width and height) and file size. After gathering information about every image in the dataset, various metrics are calculated. The minimum and maximum values for width and height are presented to the user. Counts and the proportion of images in landscape vs. portrait orientation are calculated by looking at which axis has the highest pixel count. Further, the average resolution for each axis is calculated and the average pixel count for the images in the dataset.

[1] `https://github.com/Erlein/ultimateALPR-SDK`
[2] `https://github.com/Erlein/OALPR-TestScript`
[3] `https://github.com/openalpr/openalpr/wiki/Compilation-instructions-(Ubuntu-Linux)`
[4] `https://github.com/xeonqq/alpr-unconstrained`

Finally, the average file size is calculated.

4.3.4 Timing programming

The timing is done in two different ways, depending on the software used in the implementation. Mainly, the timing is done by implementing counters in strategic points in the python scripts. The other way is utilized for ALPR-Unconstrained, where the program is run by calling a bash-script that further calls python scripts. Here, the python scrips have timing implemented in each of the scripts in order to time steps, and the whole program is timed by calling the bash script using `time <script>` in the terminal. This command returns the time elapsed between the start and end of the script or command [31].

In order to measure time in python scripts, variables are created using clocks from the python time library [32]. The variable is created at the timing point's start and then subtracted from the clock at the end of the timing window.

The clock used for this project is `perf_counter()` [33]. It has a much higher resolution than the standard `time()` clock [34]. `process_time()` [35] was also considered, as it has the same resolution as `perf_counter()`, however `process_time()` counts CPU time rather than system-wide timing. CPU time is problematic in multi-threaded processes, as we want time elapsed in the real world.

4.4 Datasets

In addition to the custom dataset created for this project specifically, we acquired access to a second dataset. There exists a collection of plates publicly available on the internet called Olavsplates [36]. Olavsplates is a collection of images of vehicles with all different variations of license plates from many different countries. After contacting the owner, we were allowed to perform testing on these images. The images were scraped from the website hosting the images and are therefore quite low-resolution; however, the size and variation in the dataset make it quite applicable for the testing in this project.

Scraping the images was done with a python script, which collects the target page using curl[37] and parses the HTML before sorting the images in the desired output directory. The scraper is not included nor appended in the report, as we do not want to supply tools for scraping images without permission.

4.4.1 Data Collection

The dataset creation will be outlined in this subsection. In order to create a dataset, we first needed permission from NSD. After that, the data collection can be conducted. Data collection takes place in the form of photographing vehicles. These photos will be categorized based on difficulty and various other criteria, and this process is outlined in subsection 4.4.2.

NSD Application & Privacy

As license plates are regarded as personal information in Norway, an application to the Norwegian Centre for Research Data (NSD) is required in order to create a dataset with images of Norwegian vehicles. This is to ensure that the acquisition, storage, and processing of the personal data of Norwegian citizens happens safely and legally.

The application resulted in permission to create a dataset of images of vehicles in Norway and the processing of the license plates on these vehicles in accordance with the required guidelines. Participants in the dataset, a.k.a. the owners of the vehicles in question, will need to be informed. An information pamphlet was created together with NSD. This pamphlet is required to be provided to the owner of the vehicle or attached to the vehicle photographed for the project to inform the participant about the project and their rights regarding participation. Unfortunately for the project, images from the dataset can not be used as examples in this thesis as a consequence of these requirements. The pamphlet is included in appendix A.4, written in Norwegian.

Photographing Vehicles

The dataset was collected by taking photos of vehicles around the local area. Various vehicles are photographed; however, the natural distribution of these was limited by having to photograph parked vehicles. Therefore, the dataset is quite heavy on cars and vans, with a few additions of trailers and lorries. Motorcycles and mopeds are not present in the dataset, as none were present when collecting data. This lack of two-wheeled vehicles is likely due to data collection occurring in spring.

Different cameras were used to photograph vehicles. These are listed in the "cameras" section of the devices chapter, at 4.2.4. Differences in images produced by the two cameras can be seen in figure 4.3, sub-figures 4.3a and 4.3c. The Pi Camera has a wider angle lens and a taller aspect ratio. Having varied cameras allows the possibility to see whether cameras with different specifications and performance impact the ALPR performance.

The images are taken to mimic typical ALPR system implementations; either mounted cameras at a toll booth, gate, or similar, or that of a parking attendant taking photos of vehicles and scanning their plates. Four sample images from the dataset can be seen in figure 4.3. A portion of the images has some more considerable variation, like more acute angles and such, to allow for testing of how well the implementations deal with edge cases. Examples of this can be vehicles parked near objects, making it difficult to photograph their plates head-on.

4.4.2 Metadata/Categorization

We manually analyzed the images in the dataset to extract and categorize various qualitative and quantitative factors about the image. This "metadata" was cre-

(a) Sample image. Mate 20 Pro

(b) Snowy car. Vehicle close by.

(c) Sample image. Pi Camera

(d) Dirty Car.

Figure 4.3: Example images from custom dataset.

ated to allow for faster and more detailed analysis of what factors affect ALPR performance. Simultaneously the images are categorized based on the perceived "difficulty" of plate detection. The difficulty is a qualitative categorization based on how difficult reading the plate is for a human. For example, a dark image of a car with a license plate full of snow will be much more challenging to read than that of a properly lit image of a clean car.

The categories contained groupings of data to keep them somewhat simple to compare. For example, the angle was estimated based on a template with four zones, depending on the highest angle from the plane of the license plate. An illustration of this template is available in appendix A.4,

A list of some of the entries in each category can be seen below. The ones marked "grouped" have a set of possible answers, while the others are less strict.

- Difficulty
  Qualitative, grouped - Easy, Moderate, Difficult
- Vehicle Orientation
  Front/rear
- Lighting Conditions
  Qualitative, grouped - Bright, Light, Neutral, Dim, Dark
- Plate Obstruction
  Qualitative, grouped - None, Slight, Some, Severe
- Obstruction Type
  None, Dirt, Snow, Shade, Pole ...

Table 4.3: Dataset categorization.

Category	Description
Difficulty	Perceived difficulty of reading plate (for a human)
Vehicle Orientation	Front or rear of vehicle
Lighting Conditions	How light is it outside?
Plate Obstruction	Is there anything obstructing the plate?
Obstruction Type	What is obstructing the plate? Snow/dirt/item?
Camera	Which camera was the image taken with?
Portion of Vehicle	Is the whole vehicle in the image?
Angle(Plane)	How severe is the angle compared to plate?
Vehicle Type	Which type of vehicle is it?

- Camera
  Pi Camera V2, Mate 20 Pro
- Portion of Vehicle
  Whole, Lower half, ...
- Angle(Plane)
  grouped - Straight, Slight, Some, Severe
- Vehicle Type
  Wagon, SUV, Trailer, ATV, MC, Hatchback, ...

4.4.3 Dataset Information

There are two primary datasets used for this project, the custom dataset and the dataset from Olavsplates.

Custom Dataset

The dataset created for this project, called the "Custom Dataset" in this report, contains images of parked vehicles from around the local area. Images for this dataset were taken using two different cameras, the Raspberry Pi V2 camera and a Huawei Mate 20 Pro smartphone. The images are of high resolution. The custom dataset contains 118 images.

Olavsplates

The images collected from Olavsplates.com are split into four for this project. On the website, the images of Norwegian vehicles are categorized depending on what type of plates are in the collection. For example, "Currently Issued License Plates." The images are split into the following categories:

- Currently Issued - 118 images
  License plates currently issued for vehicles in Norway.
  Contains a relatively high proportion of personalized plates.
- Former Issues Still Valid - 90 images
  Older license plates that are still valid. Contains old formats like plates with dashes in them, agricultural license plates and plates with registration stickers.
- Duplicates - 72 images
  Same types of images as that of "Currently Issued" plates.
- Duplicates Former Issues - 201 images
  Same types of images as that of "Former Issues Still Valid."

4.4.4 Dataset Statistics

This section presents statistics about the datasets used in this project. All images used in this project are in JPEG format.

Table 4.4: Dataset resolutions.

Stat \DS	Avg. Resolution	Avg. Pixel Count	Avg. Filesize
Custom DS	3110 x 2924	8 896 082	3.397 MB
Currently Issued	600 x 421	252 808	0.125 MB
Former Issues	573 x 415	240 602	0.105 MB
Duplicates	600 x 431	258 392	0.124 MB
Dup. FormerIssues	493 x 353	179 889	0.065 MB

Table 4.4 shows the average resolution and pixel count for the images in each dataset. The images from Olavsplates have a maximum of 600 pixels in their largest dimension. These images are of a lower resolution than those in the Custom Dataset that contains images taken with a modern smartphone and the Raspberry Pi camera v2. Looking at the resolution of the Olavsplates datasets, the DuplicatesFormerIssues dataset has the lowest resolution of all the images. This is because these images are of older vehicles and therefore contain more images that are older and lower resolution.

Table 4.5 shows what proportion of images are taken in landscape and portrait format. The images taken for the Custom Dataset were taken in the manner that was most convenient, and to provide a more varied dataset. Image taken in portrait mode can be helpful in scenarios where there are more vehicles nearby to avoid these getting in the image.

Table 4.5: Dataset image format.

Stat \DS	Landscape	Portrait	Landscape %
Custom DS	68	50	57.6 %
Currently Issued	113	5	95.8 %
Former Issues	84	6	93.3 %
Duplicates	68	4	94.4 %
Dup. FormerIssues	185	16	92.0 %

4.4.5 Dataset Modification

A python script was written to automate resizing entire datasets for the low- vs. high-resolution image tests. The script uses the os python package in order to traverse directories, and the Pillow Python package to open, resize and save the images. Images are loaded from one directory, resized using `.resize()` [38] by a given factor along each axis before being saved to an output directory. Standard JPEG compression is applied, as images are saved using `.save()` with default parameters for .jpg images [38].

4.5 Experimentation

A process to ensure consistent and comparable experimentation was created. Due to a large number of devices and other variables, this sort of rigorous process was necessary to avoid problems in the experimentation phase.

4.5.1 Process

The experimentation is run according to a set checklist to ensure that the process is as similar across all devices, repeatable, and eliminates as many third variables as possible. Following is the checklist followed for each of the tests.

- Prepare
  Make sure all needed resources are loaded onto the device.
- Connection
  Reboot device.
  Connect using SSH (where applicable).
  Wait 1 minute for any background processes to finish.
- Experimentation
  Navigate to the appropriate directory.
  Perform experiments (3 runs)
    Perform run.
    Wait 20 seconds to allow the device to cool.

Iterate output filename, repeat.

- Reset

  Offload files with results.

  Reboot device to prepare for the next set of runs.

This checklist is created to keep the experiments as real-world as possible, i.e., limiting background tasks by rebooting the system just before the tests are run and running several sequential tests to allow any potential use of cache to take effect. A waiting period after the device has booted is also enforced in order to avoid any potential startup tasks taking up resources. Everything needed for the specific test is installed onto the device before the test, including the datasets. The connection to the device is then established using ssh, in order to avoid any unnecessary resources being spent on a VNC (virtual desktop) connection. The exception for this is the desktop and laptop environments, which are running traditional desktop OSes (Windows) and are in daily use. For these, the tests will be conducted in a terminal with minimal background tasks running. When everything is ready, the device has been rebooted, and a connection is established, three consecutive tests are conducted back-to-back, with results gathered for each run. The results from each run are written to CSV files, which can be extracted for analysis after the tests are run.

After the tests are conducted, and results are written to a standardized Comma Separated Values (CSV) format across all the implementations, it is offloaded from the device for further analysis. When the CSV files are offloaded, they are then parsed with an automated script that calculates various statistics (more about this in 4.6), which are put in a spreadsheet for further comparison. In addition to the parsing script, the results get a manual overview to spot any anomalies or patterns that can help answer the research question regarding performance-altering factors.

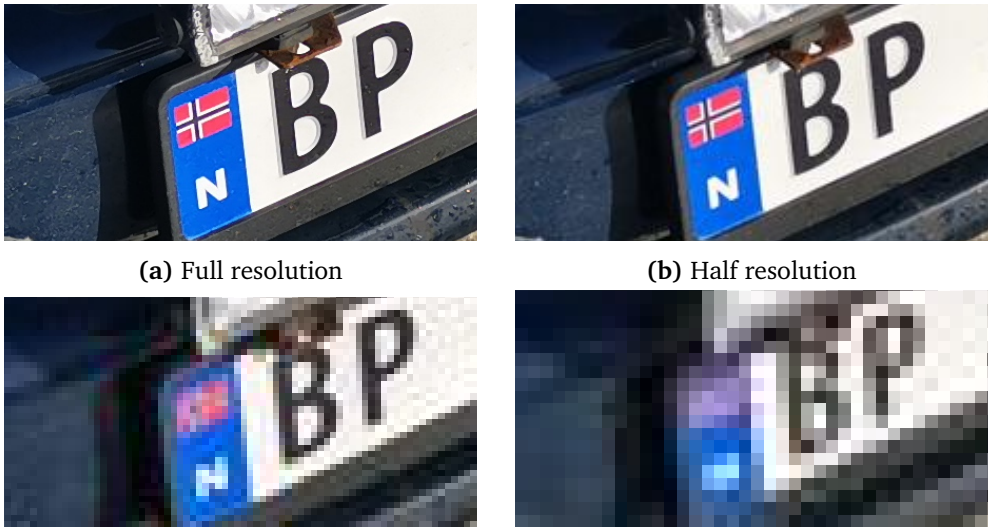
4.5.2 High- vs. Low-Resolution Image Experiments

Initial Analysis

In order to determine whether the resolution of images has a significant impact on ALPR performance for mobile devices, we analyzed the results. A comparison between the results for the Custom Dataset and the CurrentlyIssued dataset from Olavsplates were compared. These two were chosen for this comparison because the datasets are of same size, and with similar contents. With the Custom Dataset being recently created, a large portion of the dataset contains newer license plates, which is also the case in the CurrentlyIssued dataset, where the contents are of currently issued license plate types.

Secondary Experiment

A secondary experiment was conducted after analyzing the results from the primary experimentation. This experiment was performed by resizing the Custom Dataset



(a) Full resolution **(b)** Half resolution

(c) One-sixth resolution **(d)** One-twelfth resolution

Figure 4.4: Portion of sample image showing effects of dataset resizing.

to eliminate third-variables between the different datasets of varied sizes. More information about this experiment is located in section 4.5.2. The resizing was done using the python script described in section 4.4.5, where the dataset was resized to half, 1/6th, and 1/12th size. The reasoning behind the 1/6th size was that it resulted with similar size to that of the Olavsplates datasets, while the half and 1/12th sizes were halves of the original and sixth sizes to have a larger sample size. Figure 4.4 shows a portion of a sample image taken from each of the resized datasets. The samples show the data lost during each of the resizing steps. For reference, the portion of the image shown in sub-figure 4.4a has a higher resolution than the entire source image for sub-figure 4.4d. As the images are saved with a high quality of 90, the characteristic blocking from JPEG compression is not too prevalent.

Table 4.6: Resized dataset resolutions.

Stat \DS	Avg. Resolution	Avg. Pixel Count	Avg. Filesize
Full Size	3110 x 2924	8 896 082	3.397 MB
Half Size	1555 x 1462	2 224 020	0.279 MB
1/6th Size	518 x 487	246 885	0.039 MB
1/12th Size	259 x 244	61 721	0.012 MB

This secondary experiment was a last-minute endeavor in order to gather a little more information. Therefore, it was only run on a single device due to time constraints, using only two software implementations.

4.5.3 Special Characters

To test special characters, some are already present in the datasets the tests were conducted with. Not all three of the special letters in the Norwegian alphabet are present in these datasets. A one-off test where a manipulated image of a modified plate with the three letters "ÆØÅ" (plus some other dummy letters) was presented on a vehicle to the models. The image used was known to provide ample detection from.

Figure 4.5: Custom plate used for manipulation in special character test.

Figure 4.5 shows the license plate that was manipulated onto the vehicle in the image used. The image can not be shown, due to inability to gather consent. The character-combination used is not claimed as a personalized plate as of the writing of this project.

4.6 Results and Analysis

After collecting all the data from the experiments, we are left with in excess of 200 CSV files containing the detection results from all the runs. These need to be analyzed, and the results gathered in a spreadsheet for further comparison.

4.6.1 Statistics Gathering/Metric Calculation

A Python script was written in order to parse the results from the CSV files and calculate metrics from these. The script can be found in appendix A.2.3. `statistics.py` takes a CSV file as an input argument and parses each of the lines of this file. The script takes into account which software was used, as there is some variation of what information is accessible in the CSV file based on which software was run. It checks the filename for which software the run is for. For example, the result files for ALPR-Unconstrained do not contain the detection time for each run, and for UALPR, we need to ignore the last characters.

Parsing through the lines in the CSV file, we start at the top. The first line of the CSV file contains the init time. The last line contains the total time for the detection of the entire dataset, with the exception of ALPR-U, where this is read by timing the bash-script, which runs each step of the program, as explained in section 4.3.4.

Each of the lines in between contains the result from a unique image in the dataset. The format of this result contains the true plate and the detected plate. For UALPR and OALPR, these lines also contain a detection time in seconds.

Accuracy

Each line undergoes a comparison between the genuine and predicted plate. The first step of the process is to check whether the predicted plate contains the string "`NoPlateFound`", which is the case where no plate was detected. A counter for failed detections is iterated if the string is found, and the script moves on to the next line.

If a plate was detected, the first step is directly comparing the true plate and the predicted plate, string to string. If it is a match, it is counted as an "identical" match, and the number of characters is added to the counters for total and correct characters.

If the two are not an exact match, the true plate and predicted plate lengths are compared. In the cases where they are the same length, each character of the genuine plate is compared with its character-counterpart in the predicted plate. Correct characters iterate the counter for correct characters, and the number of characters in the plate is added to the total.

If the plates are of different lengths, a complex shift-checking algorithm is applied to the comparison. In short, characters can be compared with other characters than characters of the same index. The shift-checking allows for checking each character of the shorter string with n places further along the longer string in cases where the characters do not match. n is the difference in length between the two strings. The shift length will be applied to the subsequent characters whenever a shift is applied and results in characters matching. If the shift is still smaller than n, more can still be applied again in cases where the characters do not match. For anyone wanting to delve into how this functions properly, the code is available in appendix A.3, in the `findAcc()` function of the `statistics.py` script. Matching characters are counted, and the total number of characters is also counted.

At the end of the script, the number of plates detected are calculated using the total number of plates and subtracting the failed plates. Next, the correct plates are presented. Following this, the character statistics are presented, using the counters for correct and total characters.

Timing

Where the detection time is present, this is stored in a list until the end of the program, where calculations will be made. For UALPR and OALPR, average, median, minimum, and maximum prediction times are calculated using the list of collected times. These are then presented to the user at the end of the script to be transferred into a spreadsheet.

For ALPR-Unconstrained, the run is timed as the experiments are being performed, and the timings are stored in the spreadsheet directly. In the spreadsheet, the average detection time is calculated using the total time taken divided by the number of images in the dataset.

4.7 Difficulties/Problems

UltimateALPR blocks detection of the last character without a product license. An application for an evaluation license was submitted; however, we never heard anything back. To circumvent problems when calculating metrics, the last character was ignored when evaluating performance results for UALPR. This is unfortunate, as it might affect results, however only for the very last character.

An oversight while programming made the results worse than they potentially could have been. There was no post-processing implemented, which led to results being worse than they should have been, as it led to some background detections. UltimateALPR was affected the worst from this, as OpenALPR has some post-processing built in. However, this does show the performance of the ALPR solutions in their default state, but not their best state.

Chapter 5

Results and Analysis

During this project, a total of 218 test runs were made across five datasets, four devices, and three software implementations, with three runs for each combination (plus a few re-runs). This, combined with various timing and accuracy statistics, resulted in more than 2500 data points. The numbers being presented and analyzed for each of the 75 software-hardware-dataset combinations is the average of the three runs made for each. Short-hand naming for the software implementations is used for the tables and graphs in this section. The legend for these is listed below.

- UALPR = UltimateALPR
- OALPR = OpenALPR
- ALPR-U = ALPR-Unconstrained

The results section is split into parts, each with a different focus area. The first two sections show a broad overview of the accuracy and timing results, where we will start to answer the primary research question of ALPR performance on mobile devices. After these first two sections, the following sections look more in-depth at answering the research questions by comparing the performance with a focus on software, hardware, and datasets. Following these sections, the focus is on price-per-performance and factors affecting ALPR performance. Finally, the results regarding privacy requirements are mentioned.

5.1 Accuracy

In this section, the overall accuracy results from the project will be presented. These show proportions across all datasets combined. The NVIDIA Jetson only has one entry per table here, as the power modes did not affect the detection accuracy.

5.1.1 Correct Plates

Table 5.1 shows the proportion of plates that were exactly correctly identified. These numbers are the percentage of correct detections of the total number of images across all datasets.

Table 5.1: Proportion of correctly identified plates across all datasets.

Device \Software	UALPR	OALPR	ALPR-U
Jetson	82.3%	28.7%	45.6%
Raspberry Pi	82.8%	28.1%	N/A
Laptop	84.1%	30.1%	45.2%
Desktop	84.5%	28.1%	45.6%

The results show that the accuracy for each software implementation has a pretty consistent accuracy across all the devices, with a variation of just about two percent. However, there is a considerable variation in performance between the software implementations, with a difference of as much as 56,4%!

5.1.2 Identified Plates

Table 5.2 shows the proportion of images where at least one license plate was detected. It is worth mentioning that this count includes *any* detection of what the implementation thinks is a license plate. Therefore, there is the possibility of false detections (false positives) present in this data.

Table 5.2: Proportion of images where a license plate was detected.

Device \Software	UALPR	OALPR	ALPR-U
Jetson	96.8%	61.3%	77.1%
Raspberry Pi	96.5%	61.3%	N/A
Laptop	96.0%	60.1%	65.4%
Desktop	96.3%	61.3%	77.1%

The results of the detection rate are similarly consistent across all devices for each implementation, with one exception for the laptop running ALPR-Unconstrained.

5.1.3 Character Recognition

Table 5.3 shows the proportion of characters that were correctly read. This result is not calculated using *all* characters from the true plates in the dataset, but the characters from images where the ALPR implementation attempts to read characters off what it has detected as a license plate. Potential false plate detections

are also included in this, where the software might try to perform OCR on either the incorrect license plate or something it perceives as a license plate but is really not. Therefore, the results presented here need to be taken with a grain of salt.

Table 5.3: Proportion of confirmed correctly read characters (OCR).

Device \Software	UALPR	OALPR	ALPR-U
Jetson	91.4%	82.6%	82.1%
Raspberry Pi	92.1%	82.6%	N/A
Laptop	92.5%	83.6%	82.1%
Desktop	92.8%	82.6%	82.1%

Results from OCR show consistently good results across all devices and implementations. The results are consistent across devices and depend more on the software.

5.2 Speed

In this section, we present the broad timing results from the project. The tables presented in this section contain averages of data for runs across all datasets. The reason for this is the sheer amount of runs. The NVIDIA Jetson has two entries for each of the tables in this section, as the different power modes change the power consumption, resources available to the device, and in turn the timing.

Table 5.4 shows the average of the total time across all datasets for each device-software pair. The total time is the time from the script is called, until it is finished, including initialization, loading of images and detection.

Table 5.4: Average Total time for all datasets and devices. (In seconds)

Device \Software	UALPR	OALPR	ALPR-U
Jetson 5W	33.303	90.125	526.494
Jetson 10W	23.413	57.485	260.959
Raspberry Pi	34.689	67.319	N/A
Laptop	16.417	20.445	264.782
Desktop	5.939	11.250	48.863

Table 5.5 shows the average of the average detection times across all datasets for each device-software pair. The average detection time is the average time taken for an implementation to return a result after being given an image.

Table 5.5: Average Detection time for all datasets and devices. (In seconds)

Device \Software	UALPR	OALPR	ALPR-U
Jetson 5W	0.371	0.756	5.103
Jetson 10W	0.192	0.481	2.495
Raspberry Pi	0.284	0.558	N/A
Laptop	0.107	0.155	2.251
Desktop	0.039	0.129	0.431

5.3 Software Performance

In this section, results will be presented, focusing on the software implementations and their performance at detecting and reading the license plates in each of the datasets. The graphs marked with "no Pi" in them do not feature the data produced on the Raspberry Pi, because the Raspberry Pi failed to produce results for the ALPR-Unconstrained implementation and is therefore omitted to have directly comparable results.

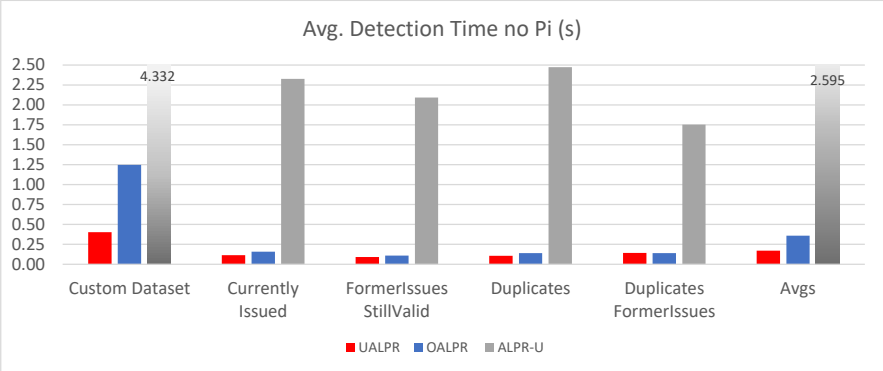


Figure 5.1: Average plate detection time across all devices.

Figure 5.1 shows the average amount of time, in seconds, for a license plate to be read by each of the ALPR implementations. The number used is an average across all devices. OpenALPR and UltimateALPR are similar in performance, with the most significant variation present when detecting the images in the Custom Dataset. ALPR-Unconstrained, however, is quite a bit slower across all datasets. Similar results appear in the graph shown in figure 5.2, which shows the average total time taken for detection of entire datasets.

For both graphs showing time used for detection, ALPR-Unconstrained comes out slower than the rest. UltimateALPR and OpenALPR have more similar performance. However, UALPR is slightly faster for most datasets, and approximately three times faster for the Custom Dataset. UALPR has the lowest time of 22.5

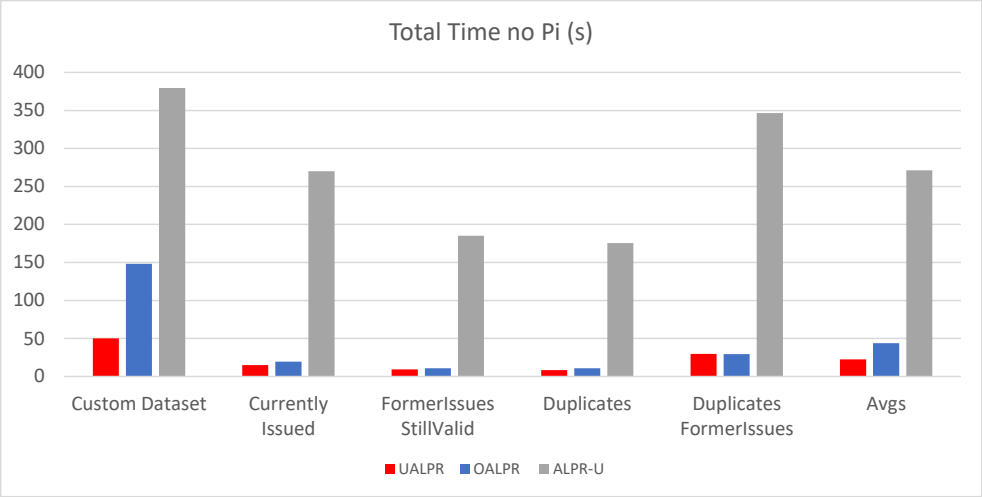


Figure 5.2: Average total time across all devices.

seconds for the averages, with OALPR using almost twice as long at 43.7 seconds. ALPR-U comes in at 271,4 seconds, 12 times slower than UALPR.

Taking a look at the accuracy, the graph in figure 5.3 shows the proportion of correctly identified license plates across all of the datasets. The numbers used are averages of the results across all devices.

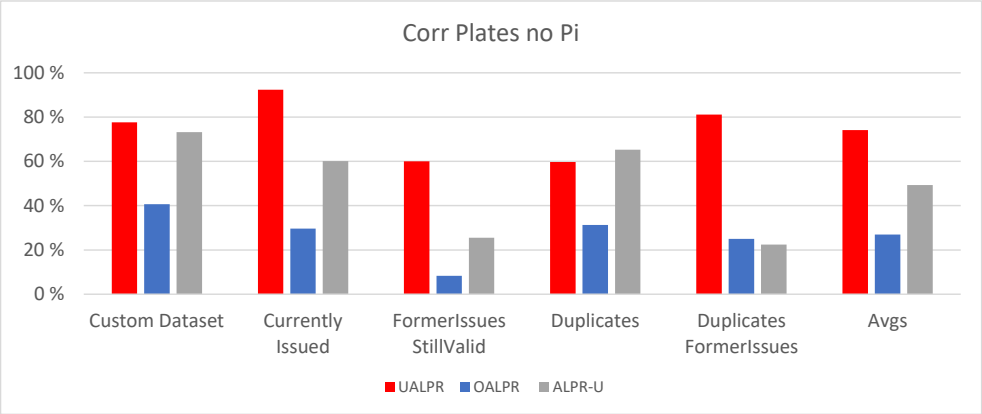


Figure 5.3: Average proportion of correct license plates across all devices.

UltimateALPR returns the highest accuracy regarding correctly identified license plates for all except one dataset. ALPR-Unconstrained has similar performance for the Custom Dataset and the Duplicates set. OpenALPR fares worse when it comes to accuracy, with the best result from detecting the Custom Dataset, with approximately 40% of the images correctly identified. On average, UALPR fares the best at 74% correctly identified license plates, followed by ALPR-Unconstrained with a 49% correct plate rate, and OpenALPR with 26%.

5.3.1 Dataset Performance

With the multiple datasets used for this project, looking at performance variations can give some insight into some factors that affect performance. We can see the differences in the performance of the datasets in the graphs presented earlier in this chapter, namely figures 5.1, 5.2, and 5.3. In figure 5.1 we see the average detection time for each individual image. This shows the time taken for the Custom Dataset takes significantly longer than the four Olavsplates datasets. Detection of images in the Custom Dataset takes 2.5 times longer than the average of the Olavsplates datasets. For the four Olavsplates datasets, the difference between the highest and lowest average detection time is 25%.

Table 5.6: Average correctly identified plate accuracy for each dataset and software.

SW \DS	Custom Dataset	Currently Issued	FormerIssues StillValid	Duplicates	Duplicates FormerIss	Avg Olavsplates
UALPR	77.7%	92.4%	60.0%	59.7%	81.1%	73.3%
OALPR	40.7%	29.7%	8.3%	31.3%	25.0%	23.6%
ALPR-U	73.2%	60.2%	25.6%	65.3%	22.4%	43.3%
Avg	63.8%	50.8%	31.3%	52.1%	42.8%	46.7%

Table 5.6 shows the average rate of correctly identified license plates across all devices for each software implementation. They essentially show the performance of each software implementation on each of the datasets. Looking at the averages of all the Olavsplates sets and comparing it to the Custom Dataset, UALPR har only slightly better accuracy when identifying the Custom Dataset. OALPR and ALPR-U, on the other hand, get their performance almost halved, at approximately 40% worse performance for Olavsplates compared to the Custom Dataset.

On average, the dataset with the highest success rate regarding correctly identified license plates is the Custom Datasets. It is the best performing dataset for two of the three software implementations. The worst performing dataset on average is the FormerIssuesStillValid dataset. It gathered the worst results for two of the implementations and just barely not the worst for the last of the three, with less than 0.3% difference.

The worst combination of software implementations and dataset was OpenALPR identifying FormerIssuesStillValid, only successfully identifying 8.3% of the plates in the supplied dataset. The best combination was UALPR identifying the CurrentlyIssued dataset, with 92.4% of the plates correctly identified.

5.4 Device Performance

This section contains comparisons of the performance of the individual devices. Comparisons will be made between the same dataset(s) and implementations directly. By extension, this will mean that the Raspberry Pi only has two implementations of data, rather than the three for the other devices.

5.4.1 Accuracy

Figure 5.4 shows the accuracy achieved for each of the software implementations. As mentioned in section 5.3, there is a considerable variation in the accuracy achieved for each software implementation. However, the variation between the devices is much smaller, with the most significant variation being 2.17% for UALPR. The

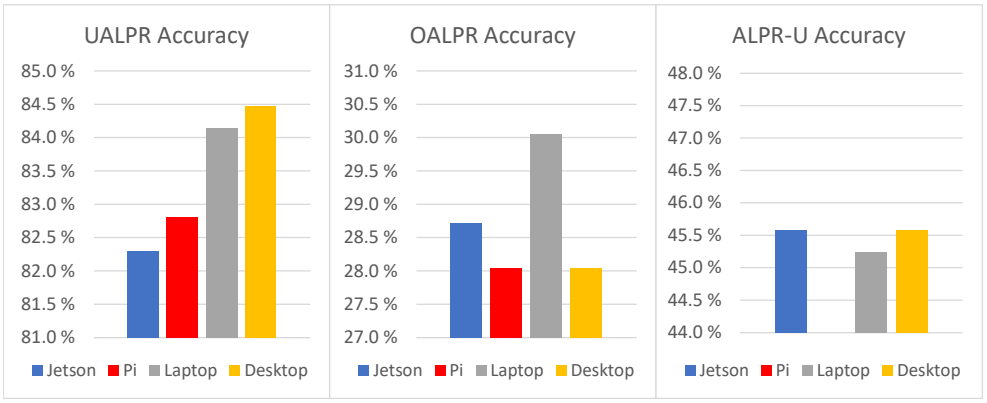


Figure 5.4: Hardware Average Device Correct Plates Accuracy

Table 5.7: Delta of device accuracy.

	UALPR	OALPR	ALPR-U
Max Δ	2.17%	2.00%	0.33%

UALPR accuracy results look like a natural progression, with accuracy performance increasing as the device gets increasingly powerful. However, this is not the case for the two other implementations, where there is no apparent direct connection between hardware performance and accuracy.

5.4.2 Speed

The graph in figure 5.5 shows the average detection time, sorted for each device in a stacked column. The average detection time used for this graph is the average for all datasets. It is worth noting that ALPR-U is not present on the Raspberry

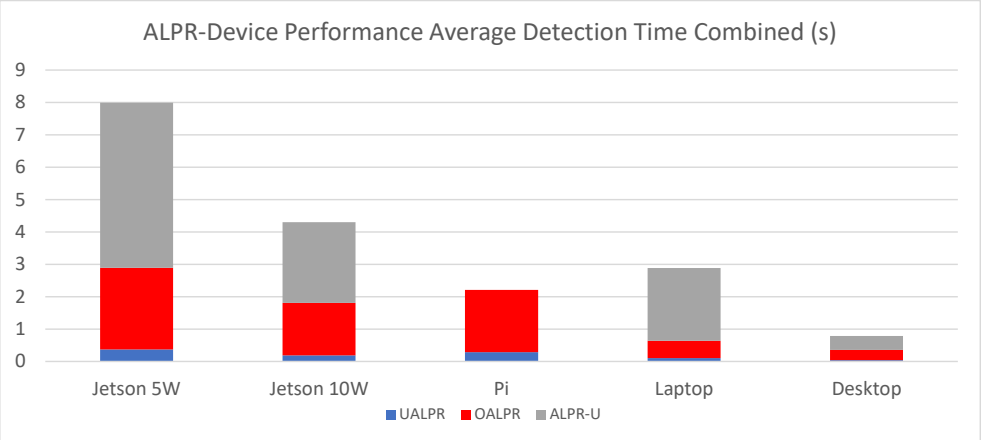


Figure 5.5: Average detection time for devices across all datasets.

Pi. For the average detection time, a lower number is quicker. There is variation between both the devices and the software.

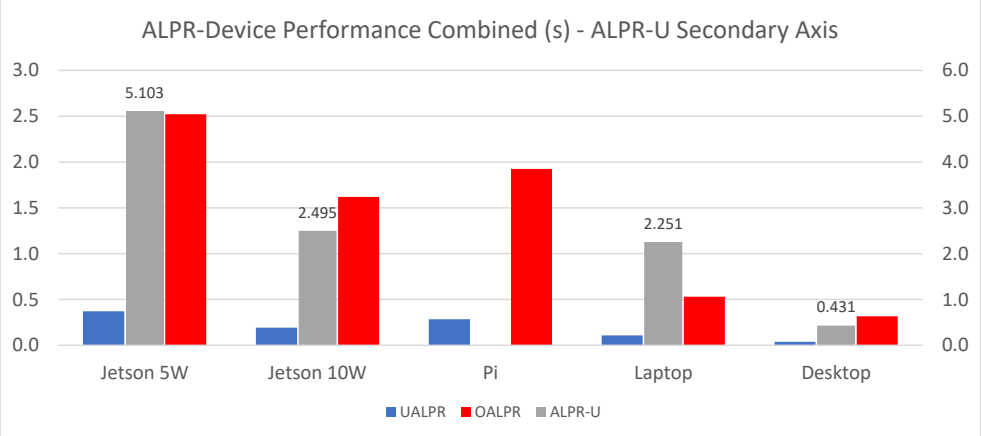


Figure 5.6: Average detection time for devices across all datasets. ALPR-U on the secondary axis.

Looking at the graph in figure 5.6, with the results from ALPR-U plotted on a secondary axis, we can see that the times seem consistent across software implementation, except for ALPR-U on the Laptop.

The results indicate that the slowest performing device is the NVIDIA Jetson in LowPower mode, with its power limited to 5 watts and only two of the four CPU cores. (NVIDIA removed the documentation for this, alternate documentation is provided in appendix A.3.1.) Next is the Raspberry Pi, which the Jetson tightly follows in FullPower mode, where it gets to utilize all four CPU cores and the GPU. The Laptop and Desktop have even quicker performance, with the laptop using 22% of the 5W Jetson, and Desktop at 12%.

Table 5.8: Average Plate Detection Time with Deltas

SW \Device	Jetson 5W	Pi	Jetson 10W	Laptop	Desktop
UALPR	0.371	0.284	0.192	0.107	0.039
OALPR	2.522	1.926	1.618	0.530	0.317
Average	1.447	1.105	0.905	0.319	0.178
Δ	100%	76%	63%	22%	12%

5.5 Hi-Res vs. Low-Res Images

Results from comparisons between high- and low- resolution images are presented in this section. This regards both results from the primary and secondary experiments.

5.5.1 Accuracy

Smaller images contain less data, which sounds like it might be detrimental to the accuracy performance in computer vision situations. The graph seen in figure 5.7 shows the correct plate accuracy results from the Custom Dataset, containing high-resolution images, and the CurrentlyIssued portion of the Olavsplates dataset, which containins lower resolution images.

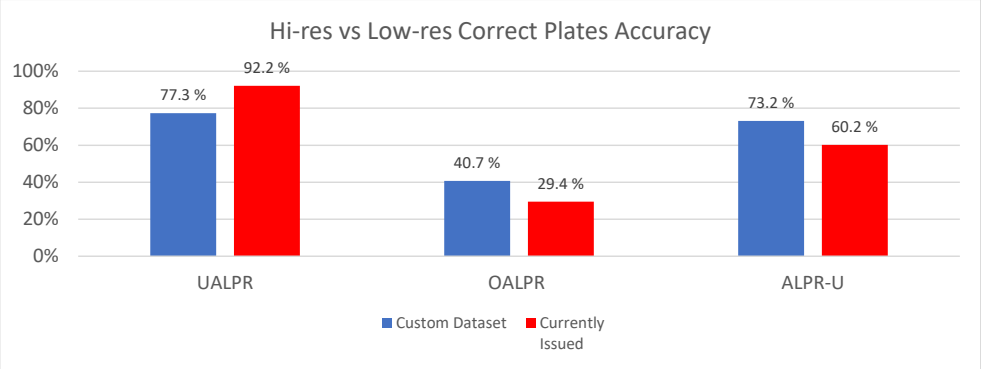


Figure 5.7: High-res vs Low-res dataset correct plates accuracy.

Looking at the results in figure 5.7, OALPR and ALPR-U seem to perform better on the larger images, while UALPR has worse performance, by almost 15%. One reason for this is the prevalence of background detections; in other words, vehicles in the background are selected as the primary target rather than the intended vehicle. Background detections are particularly prevalent for UALPR on the Custom Dataset, with 15 occurrences of this issue. The higher resolution can make it easier for the ALPR to detect vehicles in the background and process these as

well. Background detections can be mitigated by implementing post-processing, using the detections with the highest confidence values as the primary target.

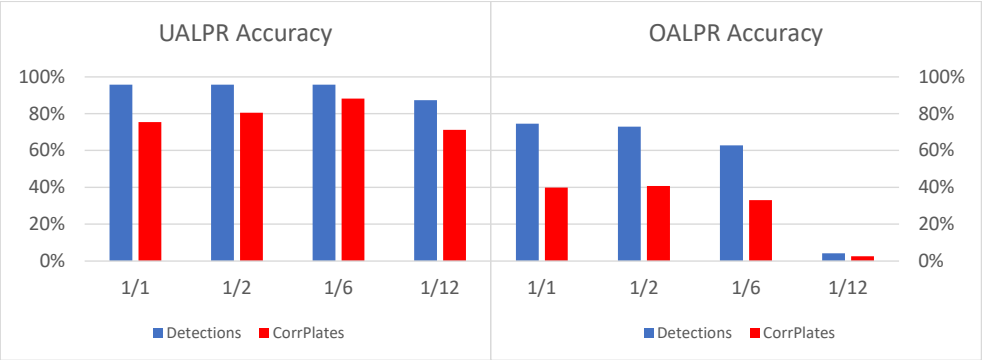


Figure 5.8: Detection accuracies and correct plates for resized datasets.

The experiments conducted with the resized Custom Dataset have more directly comparable results. Accuracy results in the form of plate detections, and correct detections can be seen in figure 5.8. Much like with speed performance, the effect of resolution depends on what ALPR software is utilized. Here, UALPR has impressive consistency, even for the lowest-resolution images, where even reading the plate as a human becomes difficult. The detection rate for UALPR is consistent for full, half, and 1/6th resolution, while the correct plate increases! The increase is likely due to the lack of post-processing, sometimes providing wrong plates as results (read more in section 4.7). As the resolution decreases, background detections become less likely, as vehicles and license plates further away become less and less detailed. The overall detection rate and correct detections dip as the resolution reaches the 1/12th original resolution, and detections of the main subject in the image become difficult.

OALPR accuracy decreases in a predictable pattern as the resolution decreases, except for correct plates on the half-size dataset, where it increases with just under one percent. This might be the same reason for UALPR, as there is no post-processing of results. There is a massive drop for the 1/12th resolution dataset, with only five detected and three correct license plate detections out of the 118 images.

In general, the resolution impacts performance; however, utilizing images that are not too high resolution appears to be beneficial with regards to the trade-off between accuracy and speed. Speed performance gets better quicker than accuracy gets worse for this type of image, with vehicles photographed from a relatively short distance. Images taken from further away, perhaps of multiple vehicles in cases where ALPR over multiple lanes of a road for example, will likely benefit from having higher resolution. As the implementations are implemented in this project, a resolution about that of the 1/6th resolution Custom Dataset or that of the Olavsplates datasets seems to be a quite good middle-ground for UltimateALPR, while for OALPR, somewhere between the resolutions of Custom

Dataset half and 1/6th.

5.5.2 Speed

When looking at the average detection time for the various datasets presented in section 5.3, one can see that the Custom Dataset takes significantly longer than the Olavsplates datasets across all three software implementations. Information about the datasets, including average resolution, pixel count, and file size, can be seen in table 4.4 in section 4.4.4. The Olavsplates dataset with the largest avg pixel count is Duplicates, with 258 392 average pixel count, only 2.9% of the average pixel count of images in the Custom Dataset. A graph with the data from the table overlayed can be seen in figure 5.9.

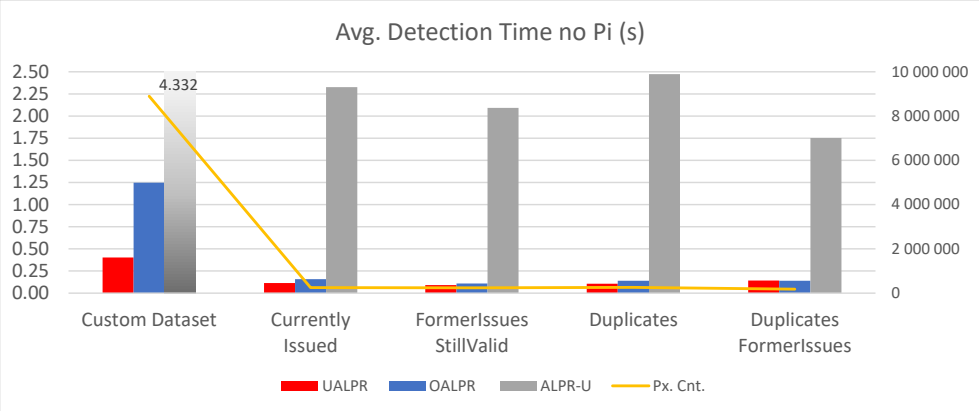


Figure 5.9: Average plate detection time across all devices with average pixel-count.

All implementations used more time to detect images in the Custom Dataset. This is likely due to the larger amounts of data in each image to be processed. To what extent the performance scales with regards to data size is difficult to conclusively decide based on these data points, but there is a clear impact on performance as the images get larger. The same goes for performance on hardware implementations, as shown in the graph in appendix 5.10, which features average detection times for high- vs. low-resolution images for UltimateALPR and OpenALPR. All devices use longer to detect images that are larger in size. For OpenALPR, the effect is so significant that the Jetson in LowPower mode performing detection on the smaller images even outperforms the Desktop on large images. ALPR-U has a similar distribution to that of UALPR. A graph of that distribution can be found in appendix A.1.1.

The results from the experiments utilizing resized datasets can be seen in figure 5.11. This graph makes it apparent that it is software-dependent to which extent the resolution affects the detection speed. UALPR shows a considerable cut in detection time from full resolution to half, with the time dropping to below half of the full time. It is worth remembering that despite the resolution only being

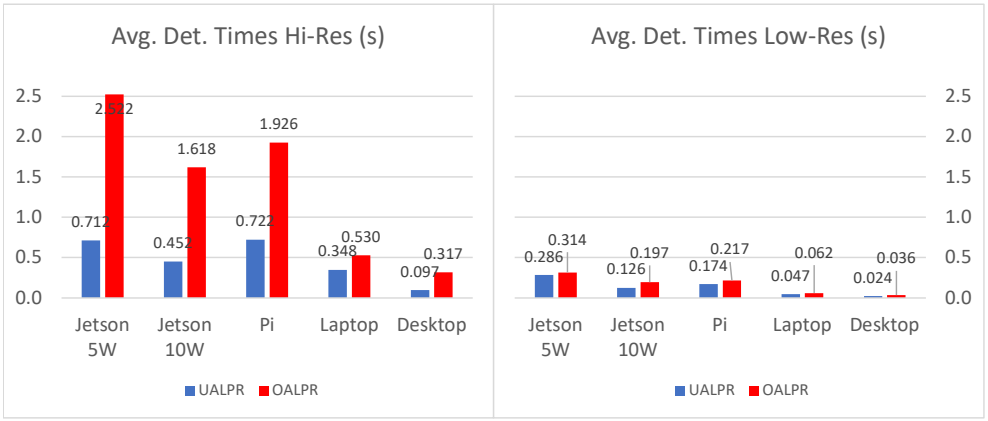


Figure 5.10: Average plate detection time for high vs low-res images.

"halved," this is along two axes and with JPEG compression on top. The average file size for images in the half-size dataset is only 8% of the full size (see table 4.6, section 4.5.2). For the following resolutions, the results show diminishing returns for processing speed as the resolutions continue being reduced, With only a reduction in average detection time from 0.115 to 0.105 between 1/6th and 1/12th resolutions.

OALPR shows a more surprising behavior, with a low reduction of only between full and half resolution, but a massive reduction when going from half to 1/6th resolution. We can speculate in the reason for this being the time saved for only doing part of the processing for images where no vehicles or license plates are found, as it starts failing detections for more and more images. Or simply that the size of the data being processed is so small that bandwidth, memory, or storage limitations disappear, and the software becomes more efficient. Investigating this must be left for future work due to time constraints.

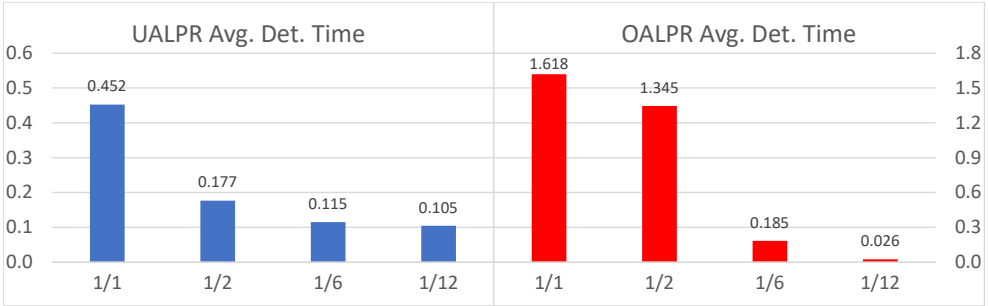


Figure 5.11: Average plate detection time for resized datasets. (In seconds)

5.6 Price per Performance

There are multiple ways to measure price per performance. The main focal points used for this project will be the price in terms of monetary value and power budget. Accuracy per time is also a consideration when looking at which software implementation to use, which is also presented in this section. Power budgets are often considerations when using mobile devices like the Raspberry Pi and NVIDIA Jetson, as these can operate on battery power in remote locations. Information about the devices and hardware specifications are listed in table 4.1 and in section 4.2.1. As there is no clear correlation between device and accuracy performance, the focus of this section will be the speed of the devices.

Table 5.9: Device Prices and Power draw.

Stat \Device	Raspberry Pi	NVIDIA Jetson	Laptop	Desktop
Price ($USD)	$59*	$99*	$827	$3000**
Power (Watts)	15 W	5/10 W	150 W	550 W

*Prices from official retailers **Purchase price estimation

5.6.1 Price

Pricing for the mobile devices was calculated during the middle of May. For the Laptop and Desktop, the price at purchase was utilized. Prices might be found in other currencies and converted to USD using mid-may conversion rates.

The prices for each device are listed in table 5.9, where one can see the steep increase in both power consumption and price as soon as we move from mobile devices over to more regular consumer devices. Figure 5.12 shows the price in USD for each license plate detection per second. The average is the average of the result from UALPR and OALPR, omitting ALPR-U, as one of the devices is missing a result for this. The calculation is done by dividing the price of a device on the average amount of license plate detections that device can do per second. For example, the NVIDIA Jetson is priced at $99 and can perform 5.221 license plate detections per second in 10W-mode. This places the Jetson at 36.74 USD per license plate detections per second. A bit of an odd statistic, but it shows price per performance in this case.

Looking at the results presented in the graphs in the figure, the Desktop and Laptop are multiple times as expensive compared to both the Jetson and the Pi, with the Desktop being the definitively most expensive option. The Raspberry Pi and the NVIDIA Jetson in 10W mode have similar performance per price, with the Pi coming out slightly ahead with its lower price point. Putting the Jetson in low-power mode nearly doubles the price per performance.

In turn, UALPR is the software implementation that gives the best performance per price for all devices in this comparison.

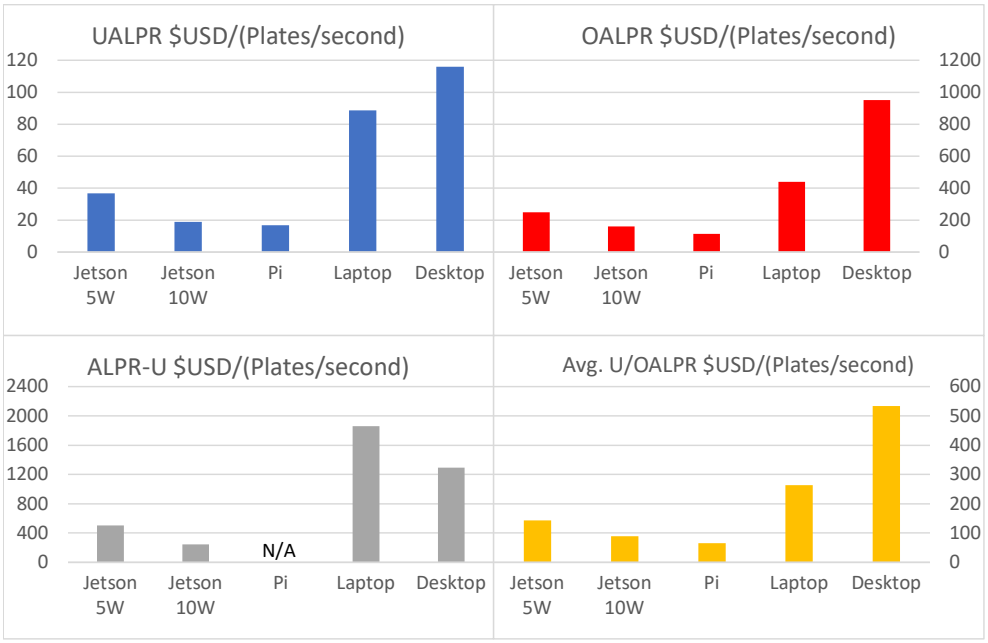


Figure 5.12: Hardware Cost Efficiency. Distribution of Dollars per Plate Detections per Second.

5.6.2 Power

Power consumption can be quite crucial in scenarios where a device does not have access to the power grid. Whether it is powered by battery power or maybe even solar power, the lower the power it consumes, the easier it becomes to build a power system around a device away from common power infrastructure. This is some of the appeal for devices like the Raspberry Pi and NVIDIA Jetson, as these can be easily powered by either USB, through GPIO pins, or in the Jetsons case, even DC barrel-jack power, which gives quite a few options when it comes to providing power. Therefore, having a device that provides high performance per power consumed can be desirable. In other words, a device able to perform ALPR efficiently.

The graph in figure 5.13 shows the efficiency of each device in the form of plate detections per second per watt of power consumption. In order to stay directly comparable, the average is only taken between UALPR and OALPR, as ALPR-Unconstrained is missing the readings for the Pi. The power consumption of each of the devices can be seen in table 5.9 or in table 4.1 in section 4.2.3. A higher number of detections per second for each watt of power consumed indicates better efficiency. Results seen in figure 5.13 show that there is little difference between the efficiency of the Jetson in low and high power mode. The CPU-only Raspberry Pi performs at just below half efficiency compared to the GPU-equipped Jetson. And last come the Laptop and Desktop, with their power-hungry low efficiency but relatively high speeds.

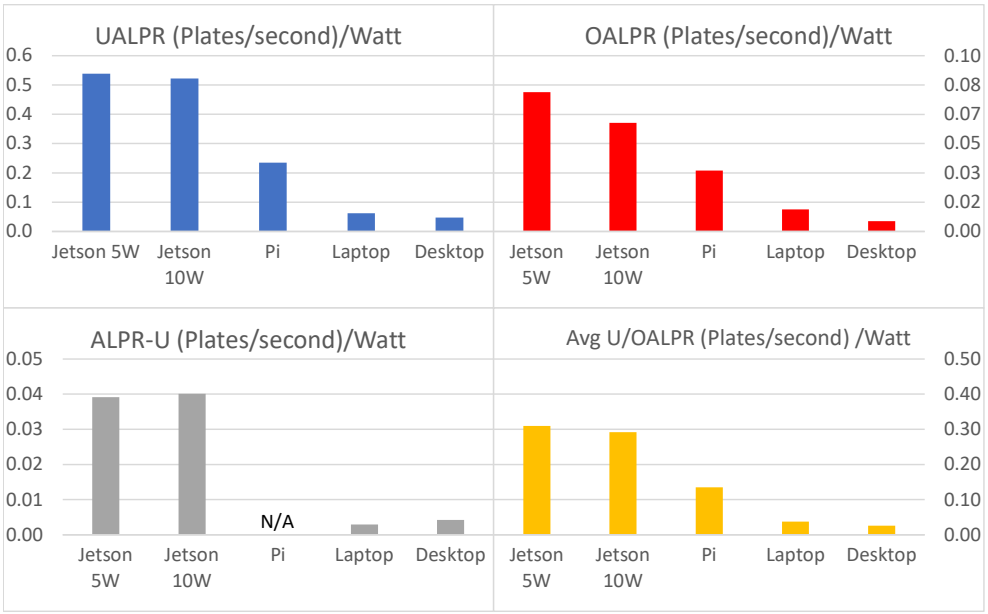


Figure 5.13: Hardware Power Efficiency. Distribution of Plate Detections per Second per Watt.

There seems to be a correlation between lower power consumption and efficiency for these devices. The exception being the results achieved for ALPR-Unconstrained, where the Jetson in 10W-mode performed ever so slightly more efficiently than in 5W-mode, and the Desktop performed 30% better than the laptop.

5.7 Failed Detections

None of the software integrations used for this project achieved a 100% correct plate detection accuracy. In this section, some of the possible reasons for this result will be analyzed, and observations made during the project will be presented. Reasons for failures depend on the individual software implementations and the functionality and methods used by these. It is worth noting that it can be difficult to distinguish between vehicle and license-plate detection errors, as some ALPR implementations simply return the predicted answer. There are various reasons for failed detections, which happen at various stages in the ALPR process. At the early stages of the process, there might be vehicle types that are not supported or some external factors that make it difficult for the model to detect a car in the image. Even if a vehicle is detected, the license plate might not be found in the image. At the end stages of the process, characters might be misread on the license plate, resulting in the incorrect output.

5.7.1 General

Some of the causes leading to failed detections, in general, are factors that make it difficult for the software to complete all the steps of ALPR. Reasons for this can be factors like obstructions in front of the vehicle or even license plate, including snow and dirt. Images taken in dark environments or bright environments where the image, or parts of the image, ends up being either underexposed or overexposed. The software implementations with the better detection rate and correct plate accuracy rates deal better with these types of images, as both types are present in the datasets used in these projects. OpenALPR appeared to struggle where plates were not consistently lit, with detection failures on images where portions of the license plate were shaded.

5.7.2 Vehicle Detection

When it comes to vehicle detection, there are multiple scenarios and factors that can lead to failures. This only applies to ALPR software that applies vehicle detection. Some implementations skip this step and simply apply license-plate detection. During this project, several potential reasons for failures were encountered.

Unsupported vehicles can be one reason for failed detections. If an implementation is limited in which vehicles the detection model supports, for example, only having trained a detection model for cars, it might fail or perform worse when presented with other vehicle types like trucks, buses, motorcycles, or trailers. UALPR failed at detecting one type of trailer while succeeding at another. The trailer it failed to detect was open, with no front or rear panels, while the successfully detected trailer had tall sides and a tarp over the top. The detector might have trained on only some particular types of trailers and could not detect others. There is not enough data on this particular issue collected in this project, so more research would need to be done in order to draw any concrete conclusions. We will have to leave this for future work.

Incorrect vehicle detection is also a factor, primarily if no post-processing is implemented. Vehicles in the background can be detected and analyzed before the primary subject in the image. Background detection was the most prominent reason for incorrect vehicle detection in this project.

In general, the most common factor for failed detections is license plate obstruction. The most normal case for this is a car that is too dirty or covered with snow. Secondary is plates that are less common, like the green plate with black text. The following subsections will present various reasons for failures across different categories.

5.7.3 License Plate Detection

License plate detections can depend on factors such as license plate types, how dirty the license plate is, and whether there are any other obstructions that can disrupt detection. The detection quality depends on the detector used in the ALPR

software implementation and how well it is trained. One particular type of license-plate detection failure was prevalent in this project. That was the detection of advertising or other text on the vehicle detected as a license plate rather than the license plate itself.

Figure 5.14: Reflection detected as license plate..

Another case of false detection is objects roughly shaped as a license plate within the bounds of a detected car. ALPR-U draws the result of detections on the given image, and the researcher's car gave such a result and can therefore be used as an example. This result can be seen in figure 5.14, where the "license plate" detected by ALPR-Unconstrained is drawn in red on the hood of the car.

5.7.4 Character Detection

Sometimes during the Optical Character Recognition part of the ALPR process, much like humans, computers might mistake some characters for other, similar characters. This section will highlight which characters are typically misread by the various implementations used in this project. These statistics are gathered by manually analyzing the results of detections from each of the software implementations.

The definitively most misread characters were zeroes and O's. Mixups between these two characters went both ways; however, which way depended on the implementation. UltimateALPR only mistook both the "O" and "0" for each other once for each of the two; in other words, both characters were equally mistaken for the other. OpenALPR, on the other hand, misread both zeroes and O's more than the other two implementations, with 0's read as O 33 times across all datasets and O as 0 18 times. Interestingly, which of the two characters was misread the most depended on which dataset was analyzed. ALPR-Unconstrained misread O as zero 18 times but did not have any false detections the other way around. Another character often misinterpreted was W. It tended to get interpreted as either the letter N or split into multiple characters as "VV" or "VN."

Something of note here is that as of 20th August 2021, standard Norwegian license plates do not contain the letter O [39]. This means the letter should only be present on personalized plates. With the majority of license plates still being regular plates, the implementations that favor the number 0 will likely statistically fare better. This majority of zeroes over "O"'s will be represented in the results in this project, as there are more zeroes present in the datasets than the letter O.

A complete list of misidentified characters can be found in appendix A.1.4. The characters and character combinations most commonly misinterpreted during the experiments in this project are:

- M and N
- 1, J, I , L, and T
  U as J
- B, 8 and 3
- S and 5
- G and 6
- O, 0, Q, D, and C
- W as "VN" or "VV"

None of the three implementations used in this project recognized the hyphen as a character. The hyphen is typically present on antique vehicles and some state cars. At first, this might seem like a problem; however, when looking up vehicles online, it seems like this character is ignored. This is at least the case on the Norwegian Public Roads Administration's page, where it is possible to lookup information on vehicles using their registration number [40].

The Norwegian characters "Æ," "Ø," and "Å" seem like they are not supported by the implementations either. Of these three letters, only "Ø" is present in the datasets in a minimal number of cases. A single-case test was conducted to confirm this, presented in section **??**. UltimateALPR detected "ÆØÅ" as "EOA," ALPR-Unconstrained returned the result "E0A" (notice the zero), while OpenALPR ignored "Æ" and "Ø", and returned an "A" for the "Å."

License plate incompatibility is also a problem encountered in this project. This problem is quite software-specific, as this is method-specific. Two-tiered plates were problematic for two of the implementations tested in this project. OpenALPR tended to miss the top row of such plates entirely, resulting in only reading the numbers for plates with the standard two-letter-five-number format.

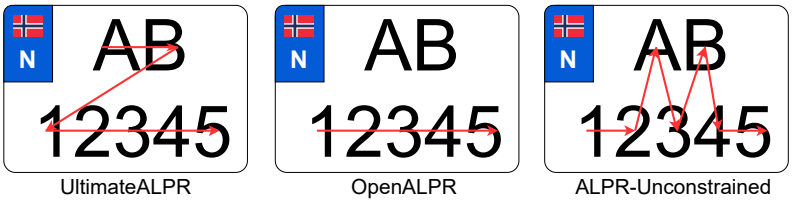


Figure 5.15: License Plate Reading Patterns.

A visualization of the different behaviors for this specific type of plate can be seen in figure 5.15. ALPR-Unconstrained had a different behavior for two-tiered plates. It consequently read these plates in the wrong order. The OCR was applied strictly left-to-right, resulting in the top row being read in-between the bottom row. The result from the example given in figure 5.15 would be "12A3B45".

Misidentifications

False-positive detections also happen from time to time. One, in particular, appeared for all three implementations across the project. This false detection was for plates with the outdated validation stickers used in Norway between 1993 and 2012 before being replaced with the use of ALPR [41][42][43]. The registration/validation sticker was typically placed between the two leading letters and the five following numbers, figure 5.16 shows a visualization of how this looked for a regular license plate. This led to cases where this sticker read as characters for all implementations.

Figure 5.16: Visualization - License Plate Validation Sticker on Standard Plate.

- UALPR
  Read as "I" x2
- OALPR
  Read as "I" x11
- ALPR-U
  Read as "I" x21
  Read as "1" x3
  Read as "C" x2

Cases for each of the implementations across all datasets (across only one device) can be seen in the list above. ALPR-U has the most cases of these false detections and even read the registration sticker as C twice. Validation stickers also had another consequence for ALPR-Unconstrained. In a few instances, the sticker led to clustering of characters on either side of it, leading to the characters surrounding the sticker being ignored. This is especially prevalent on older license plates, where this designated space was not accounted for, and the gap between the leading letters and the numbers was slightly narrower.

5.7.5 Camera Performance

In the custom dataset we know which cameras took the images. This section investigates whether there is a significant correlation between camera used and ALPR performance. Table 5.10 shows the number of failed detections for each of the two cameras used in the custom dataset. Note that background detections are left out of this data. The table shows that the dataset has 70% images taken with the Mate 20 Pro, leaving 30% for the Pi Camera. For every software, the rate is slightly lower, with UALPR at 6% lower.

Table 5.10: Failed detections per camera.

	Mate 20 Pro	Pi Camera V2	Mate %
UALPR	7	4	64 %
OALPR	46	22	68 %
ALPR-U	20	10	67 %
DS Total	83	35	70 %

Due to the low number of images in this dataset, however; it is difficult to draw any conclusions based on this limited data. There seems like a slight correlation, but the images taken with the Pi could be more difficult overall. The use of two different cameras does not seem to have had a large impact on the experiments. Performing thorough comparisons between cameras will have to be left for future work.

5.8 Privacy/Offline Implementations

All three software implementations used in this project were tested for on-device processing capabilities. The result from these tests are that all devices were able to perform ALPR with no internet connectivity. Thus, all processing happens on the device itself.

Chapter 6

Discussion

6.1 Dataset Creation

We built a custom dataset to increase the research results for this project. It allowed for the creation of a tailored dataset for the research intentions of this project. While creating and annotating the dataset, the realization of many varying factors for images, like lighting conditions, shadows, license plate types, vehicle types, and how clean the vehicle became apparent. Photographing vehicles had a positive impact in the sense that it gave us more knowledge about the imaging part of the ALPR process and the product of the dataset itself.

During the application process and conversations with the Norwegian Centre for Research Data, we learned about privacy regarding research projects and what is considered personal information. The fact that license plates are considered personal information, and what processes have to be in place to process this information. Going through the application process and ensuring that the data collection followed Norwegian law was informative and time-consuming. The application required background work with regard to planning various potential alternate angles to be used during the project in order to ensure that the permission given by NSD covers any changes to the methodology. This planning and the application process took up much time and led to the dataset creation being pushed back further in the project than planned. Additionally the extra work of designing and printing an information pamphlet to inform participants of the data collected.

The dataset is of high quality, with detailed annotation. It is unfortunate that it can not be shared and is required to be deleted after the end of the project. The reason for this is regulation, and that part of the agreement stipulates that the personal data collected during the project should not be retained longer than necessary to complete the project.

The impact of these regulations was visible from the other end when we tried obtaining other datasets, as sharing datasets containing license plates has become complicated regarding privacy. This limitation can hamper research, and a privacy-friendly way to share this information would be desirable.

6.1.1 Minor Environmental Impact of Research Project

The information pamphlet created together with NSD resulted in two pages of text. To limit the resource usage and environmental impact of having to provide each participant with this pamphlet, the paper was printed such that each A4 page resulted in two copies of the pamphlet. This halved the resource usage while still being easy enough to read.

Hopefully, all the pamphlets were recycled; however, this is not something we have control over. The author did not find any of the pieces of paper in the area where data was collected in the weeks after, which at least indicates they were not improperly discarded. With the information being on a light piece of paper placed between the windshield and wipers of the vehicles, there is a chance some might have come loose due to wind.

6.2 Implementation

Through this project, we implemented three different software solutions. This portion of the project was one of the most time-consuming, as in addition to the expected setup, it required much troubleshooting to get working across all platforms. In this section, we present a discussion on this implementation process.

6.2.1 Software Implementations

The various software implementations chosen for the project were of varying quality, making them suitable for the project. UltimateALPR and OpenALPR were adequately developed and relatively easy to integrate into custom solutions. ALPR-Unconstrained was somewhat more cumbersome but a complete program in itself. ALPR-U only needed some minor adjustments in order to add timing, and already had support for detecting directories of images.

6.2.2 Implementations Process

The implementation process includes installation and setup of all components to achieve full functionality on each device.

UltimateALPR

UltimateALPR had proper instructions for installation and pre-compiled binaries for each platform used in this project. The only requirements to make it work were the installation of appropriate TensorFlow libraries on Linux x86 platforms and a python setup using an included script. Some problems were encountered on the NVIDIA Jetson, but these were resolved using linux/aarch64 binaries rather than the jetson/aarch64 binaries. Setup instructions can be found alongside the code at GitHub [1]. Note that this repository can change over time, due to maintenance

[1] `https://github.com/Erlein/ultimateALPR-SDK`

and upgrades.

OpenALPR

OpenALPR put up a little bit of a fight while being installed; however, this was primarily due to an outdated Python package that did not work with the setups used in this project. This issue is resolved by building and installing Openalpr from source, as per the instructions supplied on the GitHub for the software, available at GitHub [2].

ALPR-Unconstrained

The first software chosen and attempted implemented during the project. ALPR-Unconstrained is open-source software easily cloned from GitHub, installed with a few setup commands, and run through a supplied script. This process worked fine for the Desktop and Laptop, with Linux running on the regular x86_64 architecture. However, making this work on the Raspberry Pi was a different story. First, the software is written with Python 2.7 and requires a set of Python libraries, namely TensorFlow with Keras frontend and OpenCV. The specific versions required are not available for Raspberry Pi. After hours of attempting to make it work, we found a Python3-port of the project, which was opted for instead. For Python3, we were able to acquire/build the required packages. Most of the program worked; however, OCR did not return any results for an unknown reason. Implementing ALPR-U entirely on the Pi was abandoned, as it had already claimed too much time.

However, this lengthy process did make the installation for the NVIDIA Jetson much less painless, which ended up working relatively quickly. Python package support is a lot more prevalent on aarch64 than armvl7.

6.2.3 Takeaways from Implementation

Two main things should have been done differently in the implementation process. First, ALPR-Unconstrained should have been abandoned on the Raspberry Pi way sooner than it did. All the time spent trying to make it work could have been better spent looking for other ALPR solutions to implement and test, which likely could have improved the results.

Secondly, the lack of post-processing of the results from the ALPR implementations could have given slightly more representative results. This realization came too late to rectify. A check for which of the results provided by the ALPR software has the highest confidence or another way to ensure the correct plate is used for the result would have been desirable.

When images have been sent through ALPR and a result is returned, it is typically returned in JSON format or some list. This is either to give multiple alternatives in cases where the ALPR is not 100% confident etc. What we did not

[2] `https://github.com/openalpr/openalpr/wiki/Compilation-instructions-(Ubuntu-Linux)`

account for when programming is that the lists are not necessarily sorted for confidence, size, or other metrics when given to the user. The first plate returned from the ALPR implementation was used for this project, as this was thought to be most likely the most prominent and correct plate on the image. There are cases where the first result returns incorrectly, for example, in UALPR, where in situations where there is a second car visible in the background, that car might be processed first and returned as the first car, and therefore also the first identified plate when results return. This background detection leads to the wrong plate (and often very incorrectly read plates as distant plates are very noisy) being compared to the true plate of the vehicle in focus on the image. An oversight that was discovered too late in the project to correct. However, the method is similar across all the three implementations, meaning that the chance for failure is the same for all, and the images tested are the same, and, therefore, hopefully, do not skew the comparative results too heavily.

The comparison shows the performance of the ALPR implementations in their default state, and it is worth noting that the results could have been better with some slight post-processing implemented. A manual review of the results from the custom dataset detections for one of the devices showed that UltimateALPR was affected the most by this, with 15 background detections for the custom dataset. OpenALPR had two, while ALPR-Unconstrained had none. These were cases where the detections clearly had read the plate of a car visible in the background.

6.3 Experimentation

The experimentation process was efficient and precise, as it was properly planned. Having an experimentation process planned out and ready to go made sure the experimentation process went smoothly. Writing the direct results to files for later statistics collection was a good idea, as it helped us by making the tedious process of collecting all the numbers a separate part of the process and did not disrupt the data collection during the experiments. The only hiccups encountered during experimentation were some minor heat-soak, where a device did not have ample time cooling down between runs and therefore got hot enough to trigger some thermal throttling. Thermal throttling is a safety mechanism where the device limits power to the processing units to prevent overheating. This, in turn, lowers performance. Exploring thermal throttling as a potential issue with mobile devices performing ALPR would be an interesting topic to pursue but is outside this project's scope.

The statistics collection could have been further automated by exporting the numbers generated into a format that could easily be pasted into the spreadsheet. Automation would also help limit human error, as the process relied on some manual reading and typing of numbers from the terminal into the spreadsheets manually.

6.4 Analysis

Due to the organized nature and relative standardization of the results spreadsheets, accessing the numbers for comparisons was simple. When performing analysis, due to the large number of tests and data points, comparisons between broader categories like devices and software implementations were made using averages. This has the unfortunate consequence of potentially introducing some statistical errors.

Missing data points for an entire device-software pair complicated the analysis a bit. Any direct comparison or average statistic between the three software implementations could not include data from the Raspberry Pi, as it could lead to bias or skew in the data.

6.4.1 Differences in Analysis of Pi vs. No-Pi Data

Cases where the Raspberry Pi results were omitted in case of skew appeared a few times in the project. For example, in section 5.3 in the Results chapter. Here, the timing performance across all the different datasets for each software implementation was presented.

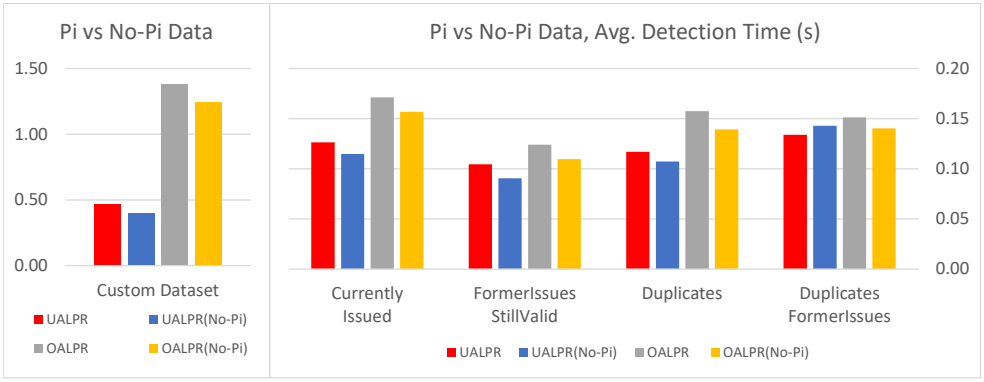


Figure 6.1: Average Detection time for each dataset, Pi and No-Pi data plotted.

Figure 6.1 shows the average detection time across each dataset for each software implementation. This figure is similar to one of the figures presented in 5.3; however, this time, it has the data from the Raspberry Pi plotted alongside to look at the effect that data has on the results. The graph only shows UltimateALPR and OpenALPR, which are the two software implementations affected. Differences between the data with and without the Pi are not too significant; however, the difference reaches as much as 14% for some.

Looking at the same type of data for accuracy, figure 6.2 shows correct plates for UALPR, with Pi and No-Pi data plotted alongside each other. The differences are minor for most datasets, with a few plates difference for each. With the exception of the "Duplicates" dataset, where the No-Pi data is significantly worse, with a

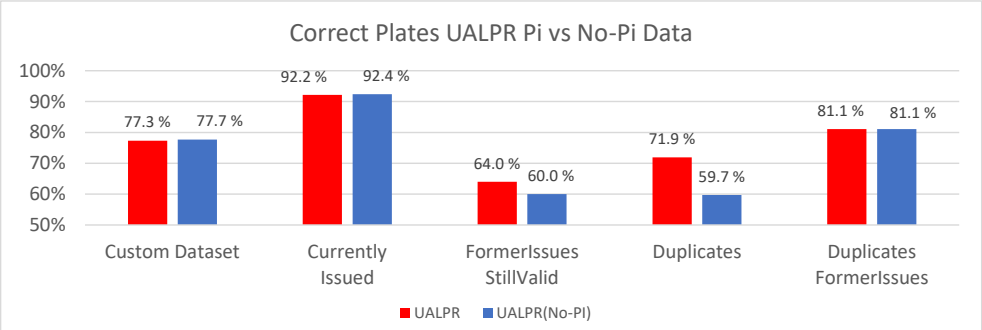


Figure 6.2: Correct Plates Accuracy for each dataset, Pi and No-Pi data plotted.

drop of almost 12.3%. A similar figure for OALPR can be found in appendix A.1.2, figure A.2.

Differences in the Pi vs. No-Pi data indicate that comparing ALPR-U with the No-Pi data from UALPR and OALPR likely was the correct choice with regards to keeping the data as comparable as possible.

6.4.2 Low- vs. High-resolution Images

During the analysis, while looking for the answer to what sort of impact resolution has on the results, it became apparent that the data was not conclusive. There were too many differences between the datasets other than the resolution that could potentially influence the results. Especially because of the lack of post-processing, giving more false detections for the Custom Dataset compared to the lower resolution dataset of Olavsplates. Therefore, a quick secondary experiment with resized dataset was conducted. This experiment was quick, as the experimentation framework was already in place, and only a script to resize the dataset was needed to perform the tests. Results from the tests gave a second set of data to draw conclusions from, which was quite valuable in this case.

6.4.3 Metric Calculation

The character accuracy calculation presented in section 4.6.1 is not perfect. While this approach allows for cases where the ALPR detection has either missed or added characters, there is the possibility that the shift can lead to "incorrect" detections (false negatives) where there were true ones. It also does not catch scenarios where the detection has added and missed the same number of characters. Applying this algorithm is still considered beneficial, as some manual review indicates the shift-checking algorithm resulting in more true positives than false negatives with these datasets.

6.5 Offline/Privacy Experimentation

The software implementations were tested and succeeded at performing ALPR without internet connectivity. This was a requirement for this project, especially due to the Custom Dataset and the guidelines imposed by the NSD surrounding the privacy of the contents of said dataset.

More thorough testing of whether the device shares information would have been desirable, however due to time constraints was dropped. Other such tests could include network packet monitoring, in order to see whether the software sends information off the device, despite performing all the processing locally.

We still got results that prove that these ALPR systems can be used in a privacy friendly manner. The limitation will be the system built around them, and how they handle the data.

6.6 Societal Consequences

Some of the information in this paper could potentially be used for malicious purposes. Information about factors that typically lead to failures in ALPR could potentially be used by a bad actor in an attempt to circumvent ALPR detection. Research regarding circumventing ALPR is suggested for future work.

The positive outcomes from the information provided can help with both monetary value, law enforcement, and energy consumption. Developers or hobbyists looking to implement an ALPR implementation can utilize the findings in this paper to make an informed choice of what device and software will provide the best solution in their situation. This informed choice will allow the savings of both money and power consumption. With the results regarding failed detections, measures can be implemented to limit these and enhance performance.

With the low bar for entry with setting up a simple variant of an ALPR system, there are considerations to be made about the degree of surveillance potentially being conducted. While anyone can technically set up ALPR or camera surveillance, there are quite strict regulations regarding this in Norway. Any person intending to set up any form of camera surveillance will need to learn these rules and have a legal basis or basis for processing for the video surveillance [44].

Chapter 7

Conclusion

7.1 Summary

Four devices were tested across three different ALPR software implementations during this project. These combinations were tested against five varied datasets, with a second experiment conducted with resized datasets. A custom dataset was created containing 118 images of vehicles. The devices tested were of different price points, mobility, performance, and power consumption.

One of the leading research questions for the thesis regarded the performance of mobile devices running ALPR and comparisons of their performance compared to more powerful devices. During the project, it has become abundantly clear that while these low-powered mobile devices perform ALPR well enough that they are suitable in many situations. Regarding device performance, having a GPU present impacts performance in a significant way when it comes to efficiency. The NVIDIA Jetson, with its 10W power consumption, beats out the 15W Raspberry Pi in detection speed. Larger devices with larger power budgets and price points perform better; however, mobile devices perform the best regarding price-to-performance and power-consumption-to-performance by a significant margin.

The experiments provided much information about the performance of the devices and software implementations. Results indicate that the main factor for license plate detection accuracy is the software utilized, not the hardware. On the other hand, speed depends on both factors, with hardware being important, but software coming in as an almost just as important factor. For comparisons, the fastest device achieved a 0.43 second average detection time for the slowest software implementation, while the slowest device achieved 0.37 seconds for the fastest software. Again, that time was cut down to almost 1/10th by being run on the fastest device.

One of the sub-questions, and requirements for this project, was the ability to run these devices in a privacy-friendly and secure manner. This was proven possible, as the implementations could run successfully on devices without internet connectivity, meaning ALPR detection can happen without sending data away from the device. With this ability, privacy and security rely on the platform built

around the ALPR implementations to handle the data responsibly.

Image resolution affects the performance of automatic license plate recognition. Lower resolution images are detected faster than images with high resolution; however, the accuracy performance suffers when the resolution gets too low. Applying clever feature-scaling techniques like UltimateALPR does with object-scaling helps reduce the impact of resolution on performance. Certain license plate types are more challenging to read than others, especially for some software implementations. All three implementations had a different way of reading two-tiered license plates, with only one being correct. And green plates with black text used by some commercial vehicles in Norway had a tendency to be harder to detect than others.

The registration stickers present between 1993 and 2012 can lead to some issues with false detections. These false detections were present across all implementations, but more so for some than others.

Optical Character Recognition is not perfect for machines, much like humans. Characters that are similar can be mistaken for each other. The most common misinterpretation of characters was between 0 (zero) and the letter "O". These misconceptions were one of the primary reasons for ALPR failure in this project, with dirty and snow-covered cars being secondary. Dirty cars are a common factor both for complete license plate detection failure, as well as mistakes in OCR.

7.2 Future Work

During this project, multiple potential research problems were encountered, which would be interesting to pursue, but were outside the scope and time of this project. Continuations or extensions of the research made in this paper is a natural starting point. Performing comparative evaluations using more software implementations, devices, and datasets. Utilizing devices from other categories like servers for high-performance comparisons or exceptionally low-power devices used for Internet of Things could yield exciting results. Implementation of post-processing to the implementations used in this project to see how big of an impact it has on the results. The post-processing method of pattern matching works well when plates are standardized; however, could the personalized license plates available in Norway (and other countries) result in the post-processing having a worse or even negative impact? Investigating weaknesses in vehicle detection models where training has occurred on datasets with minor variation. For example, are models trained almost exclusively on SUVs good at detecting sports cars?

Investigating whether thermal throttling can be a problem impacting performance for mobile devices performing ALPR. Devices like the Raspberry Pi comes with a processor without any cooling solution, which can run hot during heavy loads. Prolonged ALPR processing, like a live video feed of a high-traffic area, could perhaps lead to overheating and subsequently thermal throttling?

Further investigations of the effects of resolution and image compression on ALPR performance is an interesting point that can be explored to a significantly

larger extent than in this project. For example, pitting various image-compression methods, image file formats, and more variations in resolutions to uncover which part of the image-scaling process most significantly impacts performance.

Analyzing misuse-cases in order to disrupt ALPR. Using existing research and experimenting with various modifications of vehicles and license plates in order to attempt disrupting ALPR detections. Seeing to what extent legal modifications can hinder license plate detection. Such research will need proper ethical consideration, as information could be used for unethical activity.

Bibliography

[1] D. J. Roberts and M. Casanova, 'Document title: Automated license plate recognition (alpr) use by law enforcement: Policy and operational guide, summary,' 2012.

[2] Statens Vegvesen, *This is how section speed control works*, (Accessed: 31 May 2022). [Online]. Available: https://www.vegvesen.no/en/fag/fokusomrader/trafikksikkerhet/automatic-speed-control/section-speed-control/?lang=en.

[3] E. Commission, *What is personal data?* (Accessed: 18 May 2022). [Online]. Available: https://ec.europa.eu/info/law/law-topic/data-protection/reform/what-personal-data_en.

[4] I. Consulting, *Gdpr - personal data*, (Accessed: 18 May 2022). [Online]. Available: https://gdpr-info.eu/issues/personal-data/.

[5] Openalpr, *Openalpr design*, (Accessed: 19 May 2022). [Online]. Available: https://github.com/openalpr/openalpr/wiki/OpenALPR-Design.

[6] S. M. Silva and C. R. Jung, 'License plate detection and recognition in unconstrained scenarios,' in *2018 European Conference on Computer Vision (ECCV)*, Sep. 2018, pp. 580–596. DOI: 10.1007/978-3-030-01258-8_36.

[7] Y. Zhang and C. Zhang, 'A new algorithm for character segmentation of license plate,' in *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No.03TH8683)*, 2003, pp. 106–109. DOI: 10.1109/IVS.2003.1212892.

[8] K. Yogheedha, A. Nasir, H. Jaafar and S. Mamduh, 'Automatic vehicle license plate recognition system based on image processing and template matching approach,' in *2018 International Conference on Computational Approach in Smart Systems Design and Applications (ICASSDA)*, 2018, pp. 1–8. DOI: 10.1109/ICASSDA.2018.8477639.

[9] Lubna, N. Mufti and S. Shah, 'Automatic number plate recognition:a detailed survey of relevant algorithms,' *Sensors*, vol. 21, p. 3028, Apr. 2021. DOI: 10.3390/s21093028.

[10] tesseract-ocr, *Tesseract*, (Accessed: 25 May 2022). [Online]. Available: https://github.com/tesseract-ocr/tesseract.

[11] Openalpr, *Pattern matching*, (Accessed: 22 May 2022). [Online]. Available: https://github.com/openalpr/openalpr/wiki/Pattern-Matching.

[12] S. T. H. Rizvi, D. Patti, T. Björklund, G. Cabodi and G. Francini, 'Deep classifiers-based license plate detection, localization and recognition on gpu-powered mobile platform,' *Future Internet*, vol. 9, no. 4, 2017, ISSN: 1999-5903. DOI: 10.3390/fi9040066. [Online]. Available: https://www.mdpi.com/1999-5903/9/4/66.

[13] P. Mukhija, P. K. Dahiya and P. Priyanka, 'Challenges in automatic license plate recognition system: An indian scenario,' in *2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, 2021, pp. 255–259. DOI: 10.1109/CCICT53244.2021.00055.

[14] P. K. Ringset, *Automatisk nummerskiltgjenkjenning for mobile enheter*, nor, 2015. [Online]. Available: http://hdl.handle.net/11250/2353602.

[15] H. N. Do, M.-T. Vo, B. Q. Vuong, H. T. Pham, A. H. Nguyen and H. Q. Luong, 'Automatic license plate recognition using mobile device,' in *2016 International Conference on Advanced Technologies for Communications (ATC)*, 2016, pp. 268–271. DOI: 10.1109/ATC.2016.7764786.

[16] A. G. S Fakhar, M. Saad H, A. Fauzan K, R. Affendi H and M. Aidil A, 'Development of portable automatic number plate recognition (anpr) system on raspberry pi,' English, *International Journal of Electrical and Computer Engineering*, vol. 9, no. 3, pp. 1805–1813, Jun. 2019, Copyright - Copyright IAES Institute of Advanced Engineering and Science Jun 2019; Last updated - 2019-04-01. [Online]. Available: https://www.proquest.com/scholarly-journals/development-portable-automatic-number-plate/docview/2201019066/se-2?accountid=12870.

[17] DuobangoTelecom, *Ultimatealpr-sdk*, (Accessed: 4 May 2022). [Online]. Available: https://github.com/DoubangoTelecom/ultimateALPR-SDK/tree/d10d44db887e37f486cdeeef142e2e7555134f28.

[18] openalpr (Rekor), *Openalpr*, (Accessed: 4 May 2022). [Online]. Available: https://github.com/openalpr/openalpr/tree/736ab0e608cf9b20d92f36a873bb1152240daa98.

[19] S. M. Silva, *Alpr-unconstrained*, (Accessed: 4 May 2022). [Online]. Available: https://github.com/sergiomsilva/alpr-unconstrained/tree/a5b333e710fe474e10ee385c4bb5582e058a3978.

[20] J. Voas, N. Kshetri and J. F. DeFranco, 'Scarcity and global insecurity: The semiconductor shortage,' *IT Professional*, vol. 23, no. 5, pp. 78–82, 2021. DOI: 10.1109/MITP.2021.3105248.

[21] Raspberry Pi Foundation, *Products*, (Accessed: 26 May 2022). [Online]. Available: https://www.raspberrypi.com/products/.

[22] Raspberry Pi Ltd., *Raspberry pi 4 model b datasheet*, (Accessed: 27 May 2022), 2019. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf.

[23] NVIDIA, *Datasheet nvidia jetson nano system-on-module*, (Accessed: 27 May 2022). [Online]. Available: https://developer.nvidia.com/embedded/dlc/jetson-nano-system-module-datasheet.

[24] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris and N. D. Lane, 'Embench: Quantifying performance variations of deep neural networks across modern commodity devices,' in *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications*, ser. EMDL '19, Seoul, Republic of Korea: Association for Computing Machinery, 2019, pp. 1–6, ISBN: 9781450367714. DOI: 10.1145/3325413.3329793. [Online]. Available: https://doi.org/10.1145/3325413.3329793.

[25] Techpowerup, *Nvidia geforce rtx 2080 ti*, (Accessed: 26 May 2022). [Online]. Available: https://www.techpowerup.com/gpu-specs/geforce-rtx-2080-ti.c3305.

[26] Techpowerup, *Nvidia geforce gtx 1050*, (Accessed: 26 May 2022). [Online]. Available: https://www.techpowerup.com/gpu-specs/geforce-gtx-1050.c2875.

[27] Techpowerup, *Nvidia jetson nano gpu*, (Accessed: 26 May 2022). [Online]. Available: https://www.techpowerup.com/gpu-specs/jetson-nano-gpu.c3643.

[28] Raspberry Pi Foundation, *Raspberry pi documentation - camera*, (Accessed: 28 May 2022). [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html.

[29] xeonqq, *Alpr-unconstrained*, Python3 port of sergiomsilva's repository. (Accessed: 4 May 2022). [Online]. Available: https://github.com/xeonqq/alpr-unconstrained/tree/60fdeecac8cca66fc0f686550281cbd947ca7da1.

[30] Rekor, *Rekor carcheck*, (Accessed: 4 May 2022). [Online]. Available: https://www.openalpr.com/software/carcheck.

[31] man7, *Time(1) - linux manual page*, (Accessed: 16 April 2022). [Online]. Available: https://man7.org/linux/man-pages/man1/time.1.html.

[32] Python Software Fountation, *Time - time access and conversions*, (Accessed: 26 May 2022). [Online]. Available: https://docs.python.org/3/library/time.html.

[33] Python Software Foundation, *Time - time access and conversions #perf_counter*, (Accessed: 16 April 2022). [Online]. Available: https://docs.python.org/3/library/time.html#time.perf_counter.

[34] Webucator, *Python clocks explained*, (Accessed: 16 April 2022). [Online]. Available: https://www.webucator.com/article/python-clocks-explained/.

[35] Python Software Foundation, *Time - time access and conversions #process_time*, (Accessed: 16 April 2022). [Online]. Available: https://docs.python.org/3/library/time.html#time.process_time.

[36] O. A. Brekke, *Olav's license plate pictures*, (Accessed: 3 May 2022). [Online]. Available: http://www.olavsplates.com/index.html.

[37] D. Stenberg, *Curl.1 the man page*, (Accessed: 29 May 2022). [Online]. Available: https://curl.se/docs/manpage.html.

[38] A. Clark, *Pillow (pil fork) documentation - image module*, (Accessed: 18 May 2022). [Online]. Available: https://pillow.readthedocs.io/en/stable/reference/Image.html.

[39] Statens Vegvesen, *Number plate series*, (Accessed: 15 May 2022). [Online]. Available: https://www.vegvesen.no/en/vehicles/own-and-maintain/number-plates/number-plate-series/.

[40] Statens Vegvesen, *Check vehicle information*, (Accessed: 13 May 2022). [Online]. Available: https://www.vegvesen.no/en/vehicles/buy-and-sell/vehicle-information/sjekk-kjoretoyopplysninger/.

[41] O. A. Brekke, *Norwegian revalidation stickers*, (Accessed: 28 May 2022). [Online]. Available: http://www.olavsplates.com/norway_validation_stickers.html.

[42] M. Stokka, 'Slutt på oblatklistremerke på bilen,' *NRK*, Apr. 2012, (Accessed: 28 May 2022). [Online]. Available: https://www.nrk.no/rogaland/slutt-pa-oblatklistremerke-pa-bilen-1.8069668.

[43] L. J. Skarvøy and H. Hattrem, 'Denne våren får du ikke oblater til bilen,' *VG*, Apr. 2012, (Accessed: 28 May 2022). [Online]. Available: https://www.vg.no/forbruker/bil-baat-og-motor/i/3rJg9/denne-vaaren-faar-du-ikke-oblater-til-bilen.

[44] *Kameraovervåking - hva er lov?* (Accessed: 25 May 2022), Jan. 2022. [Online]. Available: https://www.datatilsynet.no/personvern-pa-ulike-omrader/overvaking-og-sporing/kameraovervaking/.

Appendix A

Additional Material

A.1 Additional Graphs/Tables

A.1.1 ALPR-Unconstrained Avg. Detection Time Hi vs. Low-res Performance

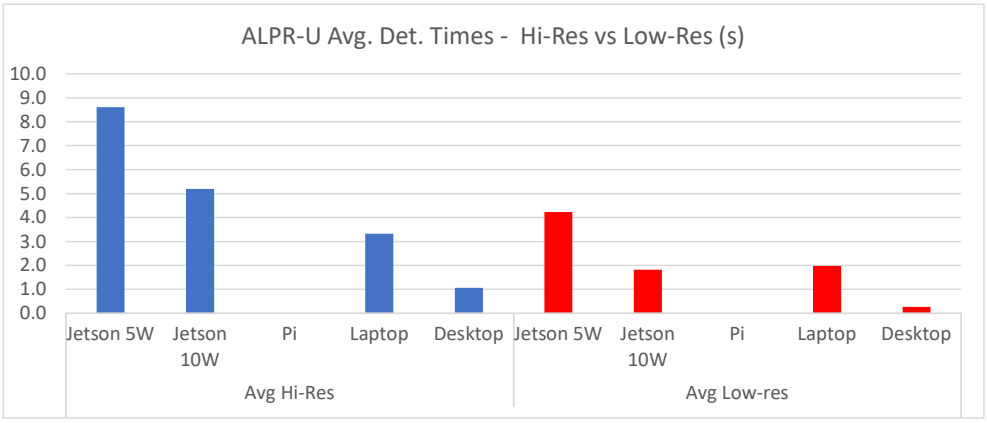


Figure A.1: ALPR-Unconstrained Average plate detection time for high vs low-res images.

A.1.2 Pi vs No-Pi Data Correct Plates OALPR

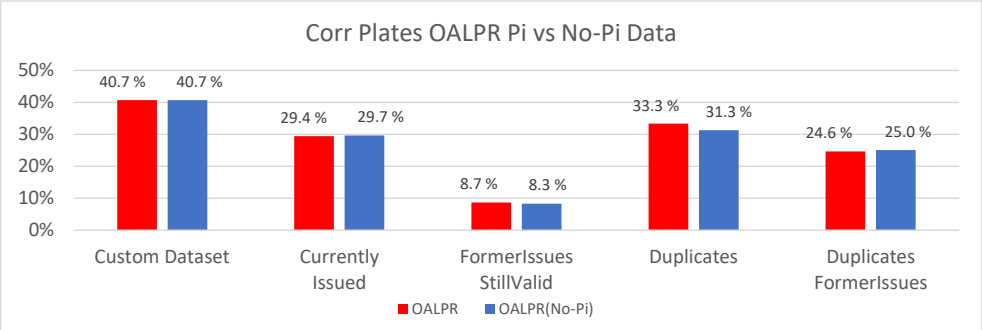


Figure A.2: Correct Plates Accuracy for each dataset, Pi and No-Pi data plotted.

A.1.3 Spreadsheet Format Example

This appendix shows the format used in the results-spreadsheet. The tables are presented section-wise, while in the spreadsheet these are attached horizontally. (meaning, table A.2 is a continuation to the right of table A.1 etc. in the spreadsheet) Each group of runs contains three valid runs, with the last row containing average values of the three runs. These averages are the numbers utilized as the results for the run.

Table A.1 contains general information about the run being run. This information is which number in the series of runs each line is, device run on, software utilized, which dataset is used and any other notes. The note typically contains information about power-modes utilized, like in the example.

Table A.1: General Run Information

Run #	Device	Software	Dataset	Note
1	NVIDIA Jetson	UALPR	Custom Dataset	5W LowPower mode
2	NVIDIA Jetson	UALPR	Custom Dataset	5W LowPower mode
3	NVIDIA Jetson	UALPR	Custom Dataset	5W LowPower mode
Avg	NVIDIA Jetson	UALPR	Custom Dataset	5W LowPower mode

Table A.2 contains the timing-section of the spreadsheet. Here, the various timing-information from each run is logged. Init shows the initiation-time for the run, total shows the time taken for detection of the entire dataset, including initiation. Next we have average and median license plate detection times, followed by the minimum and maximum times.

The final table, table A.3 contains the accuracy-portion of the runs. Here, the dataset size, number of detections, and entirely correctly detected plates are collected. From this information, the detection rate, failed detections and correct detection rates are calculated. Finally, character-count from detected plates, and the correct detections from these are gathered, and correct OCR rate is calculated.

Table A.2: Results Timing Section.

Init	Total (real)	Avg. det.	Median det.	Min	Max
0.626	87.286	0.719	0.706	0.397	1.392
0.462	85.599	0.71	0.698	0.398	1.376
0.473	85.204	0.707	0.696	0.397	1.36
0.520	86.030	0.712	0.700	0.397	1.376

Table A.3: Results Accuracy Section.

DS Size	Dets.	Det. rate	Failed:	Corr.	Corr. %	Chars	Corr	OCR %
118	113	95.763 %	5	89	75.424 %	677	567	83.752 %
118	113	95.763 %	5	89	75.424 %	677	567	83.752 %
118	113	95.763 %	5	89	75.424 %	677	567	83.752 %
118	113	95.763 %	5	89	75.424 %	677	567	83.752 %

A.1.4 Full List of Misidentified Characters

Table A.4 contains the list of misidentified characters for all three implementations.

A.2 Code Listings

A.2.1 Raspberry Pi Camera Code

`takePhoto.py` Counts how many images (files) are already in the destination folder. Then calls the built-in `raspistill` command in terminal with the number counted in the previous step plus one.

Code listing A.1: Raspberry Pi Camera - Image capture script

```
import os

count = len(os.listdir("/home/pi/DSimgs"))
command = "raspistill␣-o␣/home/pi/DSimgs/piImg" + str(count) + ".jpg"
os.system(command)
```

A.2.2 Dataset Statistics - Image Resolutions

Script traversing given dataset and collecting image data from each of the images. Calculating statistics using this data, returning average pixel-count, file size and resolutions.

Code listing A.2: Dataset Statistics - Image Resolutions

```
# Get average resolution of images in directory
import os               # OS interaction
import argparse         # Argument perser
from PIL import Image   # Pillow Image module.

# Argument parsing
ap = argparse.ArgumentParser()  # Init argparser
ap.add_argument("-i", "--input",    dest="input",   type=str,   required=True,
    help="Input␣directory␣path␣(required).")
args = ap.parse_args()

# Variables/lists
width = []
width2 = []
height = []
height2 = []
filesize = []
landscape = 0
pxcount = []

# Walk directory, collect data.
path = os.walk(args.input)  # Walk dir
for r, dir, files in path:
    for file in files:
        if ".jpg" in file:
            img = Image.open(args.input + file)
            w, h = img.size
            fs = os.path.getsize(args.input + file)
            print(args.input + file, w, h)
            # Default data:
            width.append(w)
            height.append(h)
```

```
            filesize.append(fs)
            pxcount.append(w*h)
            # Collections using largest dimension as width.
            if w >= h:
                width2.append(w)
                height2.append(h)
                print("Landscape")
                landscape += 1
            else:
                width2.append(h)
                height2.append(w)
                print("Portrait")

# Output and calculations
## General statistics
print("\nImage␣formats:␣Landscape", landscape, "Portrait:", len(width)-landscape)
print("Avg␣Pxcount:", round(sum(pxcount)/len(pxcount)))
print("Avg␣Filesize:", round((sum(filesize)/len(filesize))/1024/1024, 3),"MB␣\n")

# Resolution stats
print("Min␣max␣Widths:", min(width), max(width))
print("Min␣max␣Heights:", min(height), max(height))
avgW = round(sum(width)/len(width))
avgH = round(sum(height)/len(height))
print("Average␣w,h:", avgW, avgH)
print("")   # spacer
# "True" avg size
print("Min␣max␣width2s:", min(width2), max(width2))
print("Min␣max␣height2s:", min(height2), max(height2))
avgW2 = round(sum(width2)/len(width2))
avgH2 = round(sum(height2)/len(height2))
print("Average␣w,h:", avgW2, avgH2)
```

A.2.3 Metric Calculation

The script that parses CSV result files and returns calculated metrics.

Code listing A.3: Result Parsing - Metric Calculation

```
# Collect statistics from run-results.
import os
import argparse
import csv
import numpy as np

# Argument parsing
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input",    dest="input",   type=str,   required=True,
    help="Input␣csv␣file␣(required).")
args = ap.parse_args()

# Global variables
init        = 0             # Init time
predTimes   = []            # List containing all prediction times.
minTime     = 0             # Lowest timing value
maxTime     = 0             # Highest timing value
detAs       = [None] * 41   # Detected-as list
totals      = [0,0]         # Totals to keep track of character stats
```

```
identicals  = 0             # Counter for identical detections.
failures    = 0             # Plates not detected.

## FUNCTIONS
# Trim only filename and return in uppercase
def isolatePlate(file):
    if "_" in file:
        if software == "UALPR":     # If UALPR, trim last character.
            return file.partition("_")[0].upper()[:-1]
        else:
            return file.partition("_")[0].upper()
    else:
        if software == "UALPR":
            return file.partition(".jpg")[0].upper()[:-1]
        else:
            return file.partition(".jpg")[0].upper()

# Assign incorrect character to list
def assignWrong(a, b):  # Character a incorrectly assigned as b
    pos = None  # Init variable
    print(a)    # Debug
    # Get index for a
    if '0' <= a <= '9':
        pos = int(a)
    elif a == 'Æ':
        pos = 36
    elif a == 'Ø':
        pos = 37
    elif a == 'Å':
        pos = 38
    elif a == '.':
        pos = 39
    elif a == '-':
        pos = 40
    elif 65 <= ord(a) <= 90:
        pos = ord(a) - 55
    print(a,"assigned␣to␣index:", pos)  # Debug

    # Error checking:
    if pos == None or not 0 <= pos <= 40:
        print("Position␣out␣of␣bounds,␣invalid␣character", a , "given!␣Skipping!")
        return False

    # Iterate ticker and add charcter to list.
    if detAs[pos] == None:
        detAs[pos] = b
    else:
        detAs[pos] += b     # Add character to list of "detected as"

# Find accuracy with shift
def findAcc(truePlate, predicted):
    corrPlate = [0]     * len(truePlate)            # List tracking correct
        detections.
    corrPred  = [None]  * len(predicted)            # List trackign correct
        prediction indexes.
    maxShift  = abs(len(truePlate) - len(predicted))        # Maximum shift (diff)
    shift     = 0                                   # Current shift
    goodShift = 0                                   # Highest assumed correct shift

    for n, char in enumerate(predicted):
```

```
    # 1. Compare character
        if truePlate[n + shift] == predicted[n]:
            # Letter correct
            corrPlate[n + shift] = 1
            corrPred[n] = (n + shift)
    # Else (does not match), apply shift
        else:
            if shift < maxShift:
                while range(maxShift - shift):  # Up until maxShift
                    shift += 1  # Apply shift
                    if truePlate[n + shift] == predicted[n]:
                        # Shift seems correct, keep shift.
                        goodShift = shift
                        corrPlate[n + shift] = 1
                        corrPred[n] = (n + shift)
                        break
                    else:
                        if shift == maxShift:   # If maxShift reached and failed
                            shift = goodShift       # Set shift back to last known
                                "good" shift
                            break

    print("CorrPlate:\t", truePlate,"\t", corrPlate)
    print("CorrPred:\t", predicted, "\t", corrPred)
    corrCount = 0
    for index in corrPred:
        if not index == None:
            corrCount += 1
    print("Characters␣correct:", corrCount ,"/", len(corrPlate))
    return corrCount



# Count entries in csv file.
with open(args.input) as f:
    fileLen = sum(1 for line in f)

# Detect software utilized. (Used for applying vaious tweaks)
if args.input.split("_")[1] == "UALPR":
    print("Ultimate␣ALPR␣detected!\n")
    software = "UALPR"
elif args.input.split("_")[1] == "OALPR":
    print("OpenALPR␣detected!\n")
    software = "OALPR"
elif args.input.split("_")[1] == "ALPR-U":
    print("ALPR␣Unconstrained␣detected!\n")
    software= "ALPR-U"
else:
    print("ERROR␣detecting␣software!\n")
    exit()

# Open csv file and read contents.
with open(args.input) as csvFile:
    csvReader = csv.reader(csvFile, delimiter = ",")    # Read CSV file
    line = 0                        # Counter
    for row in csvReader:           # For each row
        # Variables
        corr = 0    # Counter for correct chars

        # Sort special lines
```

```
        if line == 0:                   # First line contains init time.
            init = row[0]
            print("Init time:", row[0])
        elif line == fileLen - 1:       # Last line contains the total time used.
            print("\n--------------------------- Statistics
                 ---------------------------")
            print("Filename:", args.input.split("/")[-1], "\n")
            print("TIMING:")
            if software == "ALPR-U":
                print("Output generation time:", row[1], "\n")
            else:
                print("Init time:", init)
                print("Total time:", row[1])
        # Loop the rest of the lines
        else:                           # For all other lines..
            print("Plate:",isolatePlate(row[0]))
            plate = isolatePlate(row[0])    # Isolate plate
            if software == "UALPR":     # If UALPR, trim last character.
                print("Pred :",row[1][:-1])
                pred = row[1][:-1]          # Gather prediction
            elif software == "ALPR-U":
                if len(row) > 1:
                    print("Pred :",row[1])
                    pred = row[1]               # Gather prediction
                else:
                    pred = "NoPlateFound"
            else:
                print("Pred :",row[1])
                pred = row[1]               # Gather prediction
            if not software == "ALPR-U":
                print("Pred time:", row[2],"\n")
                predTimes.append(float(row[2])) # Append prediction time to list.

            # Skip calculations if no plate was detected.
            if "NoPlateFoun" in pred:
                print("No plate found!\n")
                failures += 1
                line += 1
                continue

            # Perform comparisons
            if plate == pred:           # If identical (correctly identified)
                corr = len(plate)           # Set correct character count to all.
                identicals += 1
                print("Identical")
            else:                       # If not identical
                print("Not identical")
                # Check length
                if len(plate) == len(pred): # If same length, check each character
                     against eachother
                    print("Same # of characters")
                    for n, char in enumerate(plate):
                        if char == pred[n]:
                            print(char, pred[n], "same character.")
                            corr += 1
                        else:
                            print(char, pred[n], "different characters.")
                            assignWrong(char, pred[n])
                else:
                    print("Different # of characters")
```

```
                # If length differs
                # Apply shift-checking  (Real ooprog/algmet hours)
                    if len(pred) > len(plate):
                        print("Prediction longer than plate!")
                        corr = findAcc(pred, plate)
                    else:
                        corr = findAcc(plate, pred)
                        print("Shifted corr:", corr)

                # Calculate correct-rate
            print(corr, "of", max(len(plate) ,len(pred)))
            totals[0] += corr
            totals[1] += max(len(plate),len(pred))
            print(str(round((corr/len(plate)*100),3)) + "% accuracy.\n")

            print("")
        line += 1   # Iterate line counter


# Calculate and display statistics
if not software == "ALPR-U":
    print("Average prediction time:", round(np.average(predTimes),3))
    print("Median prediction time:", round(np.median(predTimes),3))
    print("Lowest prediction time:", min(predTimes), "highest:", max(predTimes),"\n
        ")


# Plates correctly detected:
print("PLATE STATISTICS:")
print("Plates detected:", (fileLen - 2) - failures, "of", fileLen - 2)
print(identicals, "of", fileLen - 2, "plates correctly detected and read.")
print("Correct plate detections: " + str(round((identicals/(fileLen-2))*100,3)), "
    %\n")

# Character statistics
print("CHARACTER STATISTICS/OCR:")
print(totals[0], "of", totals[1], "characters correctly recognized.")
print("OCR accuracy: " + str(round((totals[0]/totals[1])*100,3)) + "%")

# BETA FUNCTIONALITY: Does not work too well, needs more post-processing! Won't
    spend time on it though.
print("\nBETA FUNCTION:")
print("Incorrectly identified characters:")
print("\t Char:\t Identified as:")
for n, char in enumerate(detAs):
    if char is not None:
        if 0 <= n <= 9:
            print('\t', n, '\t', char)
        elif n == 39:
            print('\t', ".", '\t', char)
        elif n == 40:
            print('\t', "-", '\t', char)
        else:
            print('\t', chr(n + 55), '\t', char)
```

A.3 Additional Sources/Information

A.3.1 NVIDIA Jetson Power Mode

NVIDIA removed the documentation for the power modes for the NVIDIA Jetson Nano during the project. As an alternative, here is the power model definitions, gathered from the deivice itself. `/etc/nvpmodel.conf`

```
############################
#                          #
# POWER_MODEL DEFINITIONS  #
#                          #
############################

# MAXN is the NONE power model to release all constraints
< POWER_MODEL ID=0 NAME=MAXN >
CPU_ONLINE CORE_0 1
CPU_ONLINE CORE_1 1
CPU_ONLINE CORE_2 1
CPU_ONLINE CORE_3 1
CPU_A57 MIN_FREQ  0
CPU_A57 MAX_FREQ -1
GPU_POWER_CONTROL_ENABLE GPU_PWR_CNTL_EN on
GPU MIN_FREQ  0
GPU MAX_FREQ -1
GPU_POWER_CONTROL_DISABLE GPU_PWR_CNTL_DIS auto
EMC MAX_FREQ 0

< POWER_MODEL ID=1 NAME=5W >
CPU_ONLINE CORE_0 1
CPU_ONLINE CORE_1 1
CPU_ONLINE CORE_2 0
CPU_ONLINE CORE_3 0
CPU_A57 MIN_FREQ  0
CPU_A57 MAX_FREQ 918000
GPU_POWER_CONTROL_ENABLE GPU_PWR_CNTL_EN on
GPU MIN_FREQ 0
GPU MAX_FREQ 640000000
GPU_POWER_CONTROL_DISABLE GPU_PWR_CNTL_DIS auto
EMC MAX_FREQ 1600000000

# mandatory section to configure the default mode
< PM_CONFIG DEFAULT=1 >
```

Figure A.3: Power model definitions for the NVIDIA Jetson Nano.

Table A.4: Misidentified Characters Full List.

TRUE	UALPR	OALPR	ALPR-U
A		1H	R4
B			E
C			G0
D	C	C	C
E			9L
G		6	6
H		"II"	
I			J
J			III
L	1I	II	EI
M			NNNNN
N			M
O	0	0	900
R	B		
S	5	I	
T			1I
U		L	JJ
V	U		
W	"VV"N		"VN"x3 "VV"x3
X	K	K	
Y		V	
Ø	"OE"		
Å			
1	IIIT	I77III	I
2			Z
3		J	B
4		II	
5			S
6		0G4	0
8		B	9
9	0	I	6
0	O	OOQOOOOOOOOOOD OOOOOOOOOO OOOOOOOOOOO	QDC
-			
MISSED	————————- J——	78JID01I515VKL 0342315X5I UII9J1ST1WL1 25ØEA82–RA6D230 8ØØOUL-5–211TR7 41L–12E021 T3-A6P4	–C-A——— TF 2S TF2 – BT S RT UY BD 91 D ST SU3 - DL 4 A- TS EL GA – BP TV

A.3.2 Metadata/Categorization - Angle

Helping chart to help determine and illustrate the severity of angles from license plate plane. Green = straight, yellow = slight, orange = some, and red = severe.

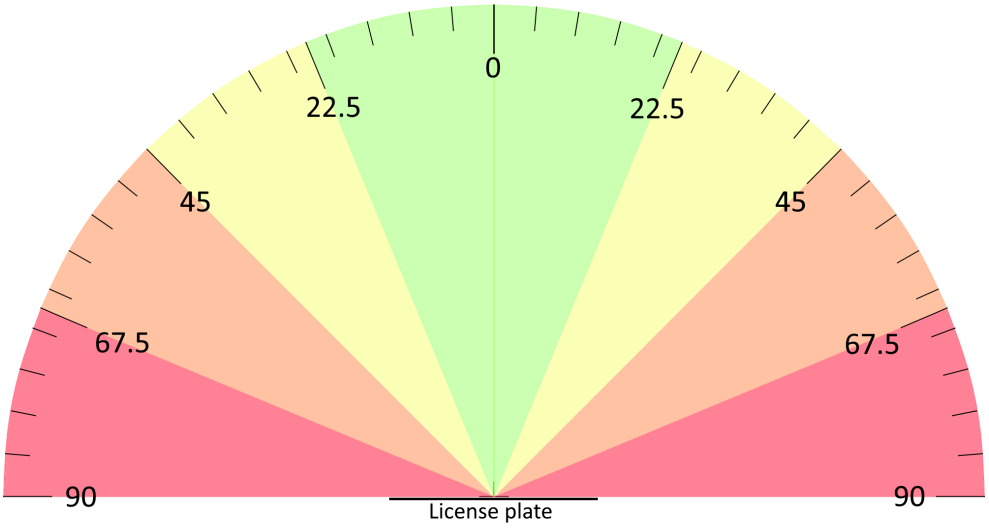


Figure A.4: Image depicting angle severity to plane of license plate.

A.4 NSD Information Pamphlet

Here you can find the NSD information pamphlet that was provided participants in the dataset. It is written in Norwegian.

Informasjon om forskningsprosjektet

Personvernsvennlig Automatisk Bilskiltgjenkjenning

I dette skrivet gir vi deg informasjon om målene for dette forskningsprosjektet og hva prosjektet innebærer for deg.

Formål
Formålet med prosjektet er å teste ytelsen på automatisk bilskiltgjenkjenning på bærbare enheter på en personvernvennlig måte. Dette kan hjelpe fremtidige valg av enheter og automatisk anonymisering for systemer som skal utøve automatisk bilskiltgjenkjenning. Prosjektet er en masteroppgave.

Hvem er ansvarlig for forskningsprosjektet?
NTNU er ansvarlig for prosjektet.

Hvorfor er du inkludert i studien?
Du (bilen din) er utvalgt fordi bilen din sto parkert på rett sted til rett tid. Utvalget er varierte biler som står parkert i «naturlige omgivelser». Utvalget er trukket basert på at det er ønskelig for prosjektet å ha bilder av biler med varierte omgivelser, med forskjellig lysforhold og andre faktorer som kan påvirke resultatet av skiltavlesinger. For eksempel, hvilken bil det er, hvor skitten den er og hvor skiltet er montert. Dermed er det tatt bilde av bilen din.

Hva innebærer prosjektet for deg?
Dine personopplysninger (bilde av bilen din med synlig registreringsnummeret) vil ikke bli knyttet opp mot hvem du (eier av bilen) er, eller publiseres på noe vis. Jeg er kun interessert i bildet av kjøretøyet ditt for å trene en maskinlæringsmodell og kjøre eksperimenter for å teste hvor bra bilskiltet kan leses fra bilder. Dataen innhentes ved at et bilde har blitt tatt av bilen.

Du kan protestere
Du kan når som helst protestere mot at du inkluderes i dette forskningsprosjektet, og du trenger ikke å oppgi noen grunn. Alle dine personopplysninger vil da bli slettet. Det vil ikke ha noen negative konsekvenser for deg hvis du velger å protestere.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger
Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

Kun forskeren (studenten) og veileder vil ha tilgang til bildene. Bildene skal lagres uten noen direkte kopling til eier (annet enn registreringsnummer på bildet selv), og kun lagres lokalt for å unngå at bildene kommer på avveie. Bilder av bilen din eller annen identifiserbar informasjon vil ikke publiseres eller brukes som eksempel i publikasjon.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?
Opplysningene slettes når prosjektet avsluttes/oppgaven er godkjent, noe som etter planen er juni/juli 2022. Bildene vil da slettes.

Hva gir oss rett til å behandle personopplysninger om deg?
Vi behandler opplysninger om deg fordi forskningsprosjektet er vurdert å være i allmennhetens interesse, men du har anledning til å protestere dersom du ikke ønsker å bli inkludert i prosjektet.

På oppdrag fra NTNU har Personverntjenester vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Dine rettigheter
Så lenge du kan identifiseres i datamaterialet, har du rett til:

- å protestere
- innsyn i hvilke personopplysninger som er registrert om deg
- å få rettet personopplysninger om deg,
- å få slettet personopplysninger om deg, og
- å sende klage til Datatilsynet om behandlingen av dine personopplysninger.

Hvis du har spørsmål til studien, eller ønsker å vite mer eller å benytte deg av dine rettigheter, ta kontakt med:

- Erlend Einmo (masterstudent, prosjektutøver), erlenein@stud.ntnu.no, NTNU.
- Lasse Øverlier (prosjektansvarlig, veileder), lasse.overlier@ntnu.no ved NTNU.

Vårt personvernombud: Thomas Helgesen, thomas.helgesen@ntnu.no, NTNU.

Hvis du har spørsmål knyttet til Personverntjenester sin vurdering av prosjektet, kan du ta kontakt med:

- Personverntjenester på epost (personverntjenester@sikt.no) eller på telefon: 53 21 15 00.

Med vennlig hilsen

Lasse Øverlier
(Forsker/veileder)

Erlend Einmo
(Masterstudent)

Erlend Einmo

ALPR Performance on Mobile Devices

NTNU
Kunnskap for en bedre verden