University of Technology of Belfort-Montbéliard
Distributed Artificial Intelligence and Knowledge Laboratory

INTERNSHIP REPORT

for

The fourth year of Dipl. Ing.
in Computer Science

presented by

Hongkun ZHEN

Image Data Anonymization for the "EU Long-term Dataset with Multiple Sensors for Autonomous Driving"

Report defended on February 28, 2020 before the committee:

Yassine RUICHEK	Professor, UTBM	Follower
Zhi YAN	Assistant Professor, UTBM	Tutor

Acknowledgments

I would like to express my gratitude to all those who helped me during the internship.

My deepest gratitude goes first and foremost to Dr. Zhi Yan, my tutor, for giving me a good project, and patiently talking with me on an equal footing in the process, guiding me to solve problems, developing my way of thinking, and teaching my way of learning. He has walked me through all the stages of writing of this thesis. Without his consistent and illuminating instruction, this thesis could not have reached its present form.

I would like to express my gratitude to Prof. Yassine Ruichek, for offering me this internship and having followed me throughout it.

I'm greatly indebted to the EPAN Research Group. It has advanced technology and a strong academic atmosphere. Here I have the opportunity to communicate with experts from various places to learn the latest knowledge in multiple fields.I can also improve my expertise day by day under the guidance of my tutor.

I want to thank my colleagues for providing me with life and technical assistance during the internship, making my internship very smooth and enjoyable, and leaving many unforgettable memories.

I should finally like to express my gratitude to my girlfriend Miss. Lian Jie who supported me during my internship, stayed with me, and worked hard to cook for me, giving me a lot of encouragement.

Finally, I would like to thank my girlfriend Miss. NAME-HERE, who supported me as always, stayed with me and worked hard to cook for me, which gave me a lot of encouragement.

Abstract

To comply with the EU's latest privacy protection regulations, i.e. the General Data Protection Regulation (EU) 2016/679 (GDPR)[1], we need to remove all private information from public data. The main goal of my internship is to blur the faces and license plates in the images of the "EU Long-term Dataset with Multiple Sensors for Autonomous Driving[2]" [8]. Due to the huge amount of data to be processed, manual removal is not advisable. To this end, we have developed a deep learning based efficient automatic image anonymization system[3]. The entire system (pipeline) consists of three modules. The first module is to extract the image data from the ROS (Robot Operating System) [2] *rosbag* and make sure to extract all the details about the image including timestamp, playback time, frames, etc. The second module uses deep learning-based methods to detect the face [4] and the license plate [5] of the image data and blur the recognized area. The third module is to record the image data back to the ROS rosbag along with all the detailed information such as the frequency and timestamp of the original data. The developed system is used to process images from the two stereo and the two fisheye cameras in the dataset. The difficulty and focus of the entire internship is the processing of the data recorded by the Bumblebee XB3 stereo camera, while the latter outputs three overlapping shadow images in Bayer format. We compared several methods and chose the best one to parse *rosbag*. We also tried various deep learning frameworks and chose the most suitable for our system. We made a comprehensive study of the data recorded by Bumblebee XB3 with reference to a large amount of literature and related source code.

[1] `https://eur-lex.europa.eu/eli/reg/2016/679/oj`
[2] `https://epan-utbm.github.io/utbm_robocar_dataset/`
[3] `https://github.com/epan-utbm/image_anonymization`

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 CIAD

I do the internship in the CIAD laboratory. The CIAD (Distributed Artificial Intelligence and Intelligence) laboratory is a public research laboratory under the supervision of the University of Burgundy and the supervision of the University of Technology of Belfort-Montbéliard. It is a laboratory of the University of Burgundy Franche-Comté. This laboratory is made up of around 70 people (Teacher-Researchers, Doctoral Students, Engineers, Post-Doctoral Students, Administrative Staff). Our goal is to design Hybrid, Distributed and Explainable Artificial Intelligence. Our activity is made up of two parts. The first concerns research on scientific obstacles and the publication of our approaches in international journals and conferences in the field. The second concerns the development of proofs of concepts (TRL 7 level) based on our research results and the support of companies (from spin-off to multinational through start-ups) in their innovation process technology.

1.2 Background

Since Google's self-driving car obtained the first U.S. self-driving vehicle license in May 2012, self-driving cars have become an important direction for automobile development. The accident rate of self-driving cars can almost drop to zero. Even if it is disturbed by the incidence of other automobile accidents, the rapid growth of the market share of autonomous vehicles will steadily reduce the overall accident rate. The driving mode of self-driving cars can be more energy efficient, so traffic congestion and air pollution will be reduced. The popularity of self-driving cars will mean that the government's investment in transportation infrastructure such as ultra-wide lanes, guardrails, speed bumps, wide shoulders and even stop

Figure 1.1: The UTBM autonomous driving platform [8].

signs can be greatly reduced.

Autonomous driving is one of the research directions of the CIAD laboratory. The lab has a multi-sensor platform (see Figure 1.1) for data collection, developed by [8]. It is equipped with 11 heterogeneous sensors, including two stereo cameras, two fisheye cameras, three 3D lidars, a 2D lidar, a radar, a GNSS-RTK, and an IMU (Inertial Measurement Unit), which allow the vehicle to get a wealth of information about the surrounding traffic. Datasets are indispensable for the advancement of autonomous driving research. The EU long dataset was first released as a public non-commercial use dataset in November 2018. However, the first release did not contain any image information. This is mainly subject to the EU's latest privacy protection regulation, i.e. i.e. the General Data Protection Regulation (EU) 2016/679 (GDPR).

The GDPR is a regulation in EU law on data protection and privacy in the European Union (EU) and the European Economic Area (EEA). Controllers and processors of personal data must put in place appropriate technical and organizational measures to implement the data protection principles. Business processes that handle personal data must be designed and built with consideration of the principles and provide safeguards to protect data (for example, using pseudonymization or full anonymization where appropriate). Data controllers must design information systems with privacy in mind, for instance use the highest-possible privacy settings by default, so that the datasets are not publicly available by default, and cannot be used to identify a subject. The GDPR was adopted on 14 April 2016,

and became enforceable beginning 25 May 2018. As the GDPR is a regulation, not a directive, it is directly binding and applicable. So we need to perform privacy processing on the data before publishing the data set. We need to remove all content that concerns the privacy of others, mainly faces and license plates. After the data is collected, it is stored in the *rosbag* data package under the ros system. Our purpose is to process the face and license plate data of the images in *rosbag*, and make sure that other data of *rosbag* is unchanged. Such as the timestamp of the message, the playing time of *rosbag*.

1.3 Structure of the Report

The remainder of this report is organized as follows. Chapter 2 introduces the main technologies we use and what we have learned. The scope is slightly broader, because the entire project involves many details, and some of which are not the main technology will not be introduced, and most of them are very basic. Chapter 3 mainly analyzes the *rosbag*, which uses a lot of ROS system technologies. In order to choose the best method, we have tried many ROS frameworks and tools, and will present many analyses and comparisons. The purpose is to quickly and easily open *rosbag* and extract image data. One can then re-record an identical *rosbag*. Chapter 4 covers the application of deep learning packages. After comparing many deep learning algorithms on the Internet, we chose one and used it to identify the parts of the data that need to be processed. In order to process the identified part in time, we also rewritten the identification package and did a lot of testing. Chapter 4 introduces the application of the community provided deep learning software. After comparing many deep learning toolkits available on the Internet, we chose an algorithm and used it to detect the face and the licence plate in the image data that need to be processed. In order to process the detected subjects in a timely manner, we have also rewritten the related detection packages and performed extensive testing. Chapter 5 introduces a special image processing. The data captured by the Bumblebee XB3 stereo camera saves the images from three different lens into a single image. This type of image requires special processing and therefore making it difficult to record into new *rosbag*s. This part is the hardest part during my internship. Chapter 6 describes the integration of different modules into an entire system.

Chapter 2

Technical Basis

2.1 ROS

The ROS provides libraries and tools to help software developers create robot applications(see Figure 2.1). It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license. Software in the ROS Ecosystem can be separated into three groups:

- language-and platform-independent tools used for building and distributing ROS-based software;
- ROS client library implementations such as roscpp, rospy, and roslisp;
- packages containing application-related code which uses one or more ROS client libraries.

Figure 2.1: Composition of ROS[6]

Our entire system is based on the ROS middleware, mainly with C++ programming. The libraries used include:

- OpenCV[1]: is a library of programming functions mainly aimed at real-time computer vision. Widely used in image processing and supports the deep learning frameworks TensorFlow, Torch / PyTorch

- *cvbridge*: is a ROS library that provides an interface between ROS and OpenCV. It is used to convert between ROS image messages and OpenCV images.

- *rosbag*: is a command line tool used to record and playback ROS message data, which log ROS messages by listening to topics and recording messages as they come in. Playing messages back from a bag is largely the same as having the original nodes which produced the data in the ROS computation graph, making bags a useful tool for recording data to be used in later development.

- *rviz*, is a three-dimensional visualizer used to visualize robots, the environments they work in, and sensor data. It is a highly configurable tool, with many different types of visualizations and plugins.

- *catkin*, is the ROS build system, having replaced rosbuild as of ROS Groovy. catkin is based on CMake, and is similarly cross-platform, open source, and language-independent.

- *roslaunch*, is a tool used to launch multiple ROS nodes both locally and remotely, as well as setting parameters on the ROS parameter server. roslaunch configuration files, which are written using XML can easily automate a complex startup and configuration process into a single command.

2.2 YOLOv3

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network(see Figure 2.2). YOLOv3 [4] is an artificial intelligence recognition model developed based on deep learning. the architecture of YOLOv3 is shown Figure 2.2.

Unlike other detection systems which repurpose classifiers or localizers to perform detection, and apply the model to an image at multiple locations and scales, while high scoring regions of the image are considered detections. YOLOv3 uses

[1] `https://opencv.org/`

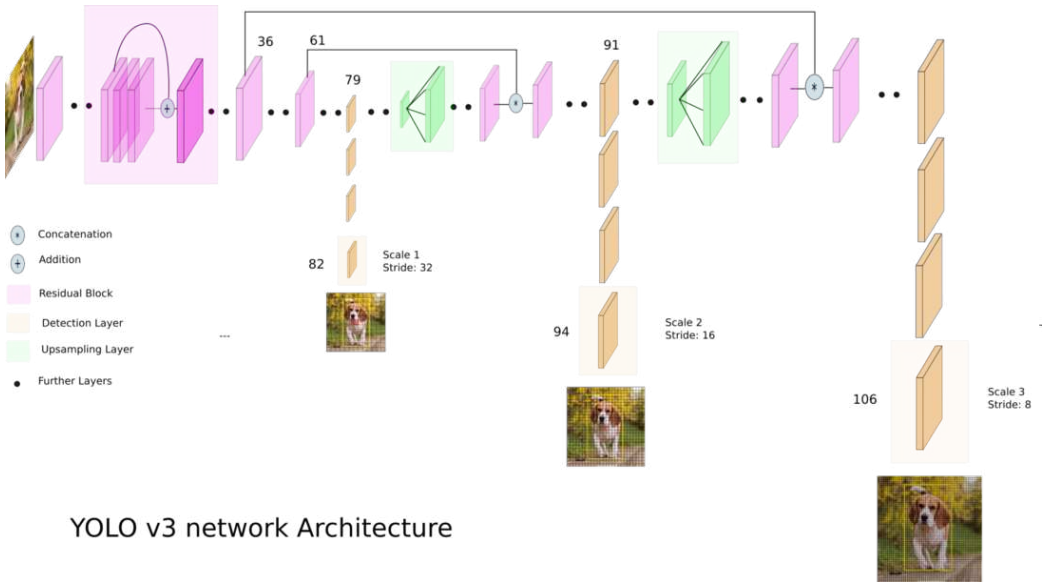

Figure 2.2: YOLOv3 network Architecture [4].

a totally different approach, which applies a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. This model has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.

Chapter 3

Extraction and Recording of *rosbag*

A *rosbag* is a file format for storing ROS message data. It is also the release data format of the EU long-term dataset. Due to its binary format, it cannot be processed directly. We therefore need an efficient and robust system for *rosbag* data extraction, conversion, and recording. This chapter gives relevant details.

3.1 ROS publisher and subscriber

"Node"[1] is the ROS term for an executable that is connected to the ROS network. Here we'll create a publisher node which will continually broadcast a message. The subscribe call is how you tell ROS that you want to receive messages on a given topic. This invokes a call to the ROS master node, which keeps a registry of who is publishing and who is subscribing. Messages are passed to a callback function, so that We can process the data.

3.1.1 Feasibility study

According to the project requirements, we must first extract the data from *rosbag*. *rosbag* is a ROS-environment playable bag. When it's played, all previously recorded ROS "messages"[2] to this bag will be synchronously broadcast according to the corresponding timestamp. At this point, one only needs to select the topic where the required messages is located and subscribe to it to get the recorded data.

In our case, we first publish the required messages by a ROS publisher, and then use a subscriber to receive them. Data processing starts after the reception. Although this method can achieve the extraction of *rosbag* data, it is not convenient to record new *rosbag*s. Since a large number of *rosbag*s need to be pro-

[1] `http://wiki.ros.org/Nodes`
[2] `http://wiki.ros.org/Messages`

cessed, a large number of publishers need to be started. Each Publisher node plays a *rosbag*, which is a waste of resources. We need the new *rosbag* to have the same timestamp as the original timestamp. This is a piece of information that publishers need to republish separately, which adds to the burden. In theory, the two processes of opening a *rosbag* and recording a new *rosbag* should be synchronized. Obviously, the method of recording *rosbag* in this framework needs to be reconsidered.

3.1.2 Test

We built a Publisher and Subscriber model.Play *rosbag* on the Publisher,Subscriber receives messages for two topics. One is the image message and the other is the image's head, The head contains the timestamp of the message.

This method is very efficient, and the data can be stored in time after the Publisher releases the message. The specific delay time will be discussed in the experimental section later. In my sample tests, There are no missing messages in the results of several tests, The data transfer process is very stable, and the speed is as fast as *rosbag*'s playback speed, which is very efficient. Each message and the head are published synchronously, effectively saving the timestamp.

The first problem we encountered was that we couldn't play multiple *rosbag*s in order. There are no such examples online. We can only achieve this by starting the Publisher node multiple times with the help of a script. At the same time, the problem of system resource occupation will be discussed in later experiments.Since the saved image data and head data are in one-to-one correspondence, the best way is to process them in time without saving them locally.But during the test, we do n't know how to record *rosbag* by Subscriber.

There is also a server and client model, similar to this. Not discussed for now

3.2 Launch file

The launch file is used to start the node. It can start multiple nodes in the background at the same time, give detailed parameters, and automatically close all nodes at the end of the run.In the official wiki tutorial, the way to exporting image data from *rosbag* is to use a launch file.Created 2 nodes in the launch file. One is the play function of *rosbag*, and the other is the exporting image from the *rosbag* running according to the topic parameter. It is equivalent to executing two ros function packages in two terminals, and specifying detailed parameters in the launch file, such as the path of *rosbag* and the name of the topic. For example, this 500-frame video data is stored in *rosbag*, and 500 messages are sent during playback, while another node will extract 500 images based on these 500 messages

and save them to the local default path.

3.2.1 Feasibility study

This is the easiest way to extract image data from *rosbag*. This is also the official method, simple and intuitive. But this method did not involve the recording of the new *rosbag* at all. because the results are directly saved in the local as image format, it is impossible to extract the timestamp of the source *rosbag* message.Although this method is very convenient for extracting image data, due to the strong directivity of the function, many details required for the project cannot be achieved.

3.2.2 Test

We tested with the launch file recommended in the ros wiki tutorial.Analysis of the test results of multiple groups, we found.

The launch file can use many existing ros packages, which is very simple and convenient.Data images are saved in jpg format, we don't need to do data conversion anymore, which makes us more convenient for image processing later.Image data sequence is correct and complete. The launch file will automatically run each node as a background process, no need to distribute operations on multiple terminals, which also makes the operation more concise.After the launch file was running, We observed the background process. We found that after the launch file was finished, the process started by the launch file was automatically ended, avoiding the process conflict caused by multiple operations.

The biggest problem with this method is that it cannot get the timestamp of the source *rosbag* message. Since we need these timestamp data when we record a new *rosbag* at the end, if we open the *rosbag* with a launch file, which means that we need to open the source *rosbag* again to get the timestamp when we record a new *rosbag* at the end, which makes the project more complicated , causing unnecessary waste of resources. And this method can only process one *rosbag* at a time. If you want to process a large number of *rosbag*s, you still need a script. And the image data obtained from the launch file is initially stored in the default path, and the data must be transferred after each processing. There is also the problem of dropped frames, The launch file has a rate for *rosbag* image extraction. The default is 0.1 seconds, which means that it extracts ten times per second. Although the parameters can be modified to increase the rate of extracting images, it cannot be completely coincided with the playback rate of *rosbag*. So there will be a few dropped frames.

3.3 *rosbag* C++ API

rosbag has both C++ and Python APIs for reading messages from and writing messages to bag files.Because we am more familiar with C++, we used *rosbag* C++ API.This is a C ++ library that specializes in *rosbag*. The *rosbag* C++ API works on the premise of creating "views" of one or more bags using "queries". A Query is an abstract class which defines a function that filters whether or not the messages from a connection are to be included. This is a set of tools for recording from and playing back to ROS topics. It is intended to be high performance and avoids deserialization and reserialization of the messages.With this complete C ++ library, we can flexibly perform multiple operations on *rosbag* using programming

3.3.1 Feasibility study

The *rosbag* C ++ API has complete *rosbag* processing functions.From the opening of *rosbag* to the closure of *rosbag*,,there are access to topic name, datatype, md5sum, message definition as well as the connection header. Using this method, you can complete the data extraction from *rosbag* to the final recording of the new *rosbag*,including keeping *rosbag* timestamp unchanged. However, since it is a programming language, the probability of error will be relatively higher, and it will be more complicated and difficult to use. Many problems still need to be determined after testing

3.3.2 Test

I tried to achieve the *rosbag* image extraction and the recording process of the new *rosbag* in a ros package. Since we didn't find related examples on the Internet, according to the official source code of the *rosbag* C ++ API, we wrote a simple ros package. After testing, this method can achieve all the functions we need. *rosbag* image extraction, data conversion, message timestamp extraction, recording of new *rosbag* and so on. We have tested each function individually. The *rosbag* C ++ API can make every function perfect.Especially the method of recording new *rosbag*. So far, in addition to using the *rosbag* C ++ API for programming recording, all we know is to use the ros instruction *rosbag* record for real-time *rosbag* recording. But this method is not very operable,which is difficult to implement all kinds of details

But compared to several other methods, the programming process is more difficult and tedious. Integrating many functions into a piece of code will cause many unexpected problems, so more detailed testing is needed. And in testing, we found that the recorded *rosbag* playback rate is equal to the rate of the program running loop. In contrast, the other *rosbag* methods do not need to change the

recorded *rosbag* rate because the *rosbag* playback and recording are synchronized and the playback is not affected by other processes.The programming with the *rosbag* C ++ API is different. We just open the *rosbag* with a function and then read it one by one according to the topic. The loop rate is the rate at which the program runs. Therefore, the two steps of extracting pictures from *rosbag* and recording new *rosbag* need to be separated separately, because there is also a process of image data processing in the middle, which will affect the rate of new *rosbag*.Therefore, the previous analysis, this method can simplify the combination of *rosbag* data extraction and new *rosbag* recording, which is not feasible.

3.4 Analysis and operation

My mentor suggested that we create this entire framework following the basic structure of ros, the model of publishers and subscriber. After trying, we found that this method has many disadvantages. Firstly, because the publish and subscrib in this model are performed synchronously, operations that take too long cannot be implemented in the middle. However, my project needs to perform artificial intelligence recognition processing on image data in the middle, so it will inevitably take a long time. Secondly, using this model alone cannot accurately process the internal parameters of *rosbag*, such as timestamps. So in the end we gave up this method.

After many tests and comparisons, we find that different methods have different advantages, so we will probably use multiple methods to deal with multiple processes in the system separately. First, to read *rosbag* messages and record new *rosbag*, we chose to use the *rosbag* C ++ API. The stability of the three methods is qualified, and there will be no obvious drop frames, but because we need to multi-step processing the *rosbag* message, and to extract various information (time stamp, etc.) of the message, we also need to record new *rosbag*, so the *rosbag* C ++ API which is the most operable method is the best choice .Next, if some processes only require a large amount of image data extraction, we will choose the launch file because this method is the simplest and most convenient. However, this is only an idea, and it depends on the future needs.

The following is the actual operation process

I refer to the basic *rosbag* C ++ programming template on the ros wiki tutorial, and we have viewed the source code of the *rosbag* C ++ API. The *rosbag* API has two main classes: bag and view.

First, we use the open function of the bag class to open the *rosbag*, we use the view function of the view class to mark the topic to be processed, and then loop the specified topic internally. Each loop is a frame of the *rosbag*. In the loop, the information of each frame (a message) is extracted, including the image, the

Algorithm 1: open *rosbag* and extract image

Data: *rosbag B*, topic of the *rosbag* T_B, message m, timestamp of the messaget_m
Result: image im^*

```
open B;
foreach m in the T_B do
    convert m to image format;
    im* ← m;
    name im* as t_m;
    save im*;
end
```

timestamp in the header, and the image is saved locally for subsequent processing. The image is stored in *rosbag* as a message, so we need the cvbridge library which is used to convert the image from the message to an image matrix. In order to do experiments at the end, more information needs to be extracted. After extracting the timestamp for each message, we use the timestamp as the name of the image for that message. The timestamp is an eighteen-bit floating-point number, which requires nineteen bytes. To convert it to a string, you cannot use the general number-to-character method.I input the timestamp variable directly into a stringstearm variable using the stream method, and it can be saved as the name of the picture.Although the string object is very convenient, the string type is likely to be the source of an engineering efficiency problem. In product-level applications, you should try to avoid using the string type in deep loop nesting, so using C ++ streaming is better. Since this step only saves the image locally, there is no need to consider frequency.

The next step is to record the new *rosbag*. In order to make the newly recorded *rosbag* highly consistent with the original *rosbag*, we need to open the original *rosbag* first, and then record the new topic synchronously in the message loop of the original *rosbag*. In this way, the topic name of the newly recorded *rosbag* and the number of message frames remain unchanged, and the header of each frame of the original *rosbag* can be extracted and assigned to the new *rosbag*. The specific method is: first we need to create a bag object to open *rosbag*, and then re-create a bag object, loop the original *rosbag* message in the specified topic, read a frame of images in sequence in the loop, and convert it using cvbridge to msg format, and then write into the new bag object. The newly recorded *rosbag* also needs to keep the same playback frequency as the original *rosbag*, but the frequency of the new *rosbag* recorded is equal to the frequency of the original *rosbag* message loop. This frequency is not equal to the *rosbag* playback frequency, so the frequency of

Algorithm 2: Record new *rosbag*

Data: original *rosbag* B_1, processed image *im*, topic of the *rosbag* T_B, message *m*, timestamp of the message t_m ,duration of the original *rosbag* t_{B_1}, number of original *rosbag* frames n_{B_1}

Result: newly recorded *rosbag* B_2^*

```
open B_1;
open B_2^*;
foreach m in the T_B do
    read im;
    convert im to message format;
    write im into B_2^*;
    t ← t_{B_1} / n_{B_1};
    sleep t;
end
```

the message loop needs to be changed. In the view class, you can get the time when *rosbag* starts to play and the time when it ends, and then we subtract it to get the length of *rosbag*. We get the number of *rosbag* frames in the view class. You can get the frequency of *rosbag* by dividing, so that we adjust the original *rosbag*'s message loop to this frequency. Although there will be slight errors in the calculation process, it can also roughly keep the *rosbag*'s playback frequency same.

3.5 Discussion

Compared with other data storage methods, *rosbag* is quite complicated. In a *rosbag*, you can store quite a lot of data, and all data is very neatly synchronized. This is like a future movie. You can see the images synchronously every frame, hear the sound, smell the smell, feel the temperature, and even feel the emotion. Although the extraction and reconstruction of data becomes more complicated in this process, this kind of data storage can support a complex system such as robots with multi-point sensing.However, there are still some problems with this storage method

As far as the image problem we am dealing, the final recorded *rosbag* will be much larger than the original image data. This is very inconvenient for uploading and sharing data. After consulting many sources, we found that the official ros solution is compression. This is for image data. There is a compression format / compressed for storing images in the ros system, which can effectively reduce the data write size. However, this compression format will damage the quality of the

image and has certain defects, but it will reduce the size of *rosbag* by more than 2 times. However, the reason why the size of the *rosbag* is larger than the data is still unknown. Another problem is that the playback frequency of my newly recorded *rosbag* is calculated by the data. There is a slight error. The reason is that we can't get the *rosbag* playback frequency directly. If we really can't get the *rosbag* playback frequency directly, then we think *rosbag*'s library needs to be improved.

Chapter 4

Face and License Plate Removal

After determining the method of *rosbag* data extraction and packaging, we need to perform face recognition and license plate recognition on the saved image data, and then perform mosaic processing. At present, artificial intelligence image recognition technology is very mature, and there are already many recognition algorithms and recognition frameworks. Based on the characteristics of my project, we need to choose the most suitable framework for identification. First of all, we need to process a lot of data, there are about 2 T image data, so the processing speed is a very critical index. If the processing speed is too slow, it will affect the establishment of the entire online data set. However, due to the EU's latest privacy protection legislation, there are considerable requirements for the accuracy of identification. At least on the premise that the human eye can recognize the face or license plate, the system must be able to recognize the target. In addition, the GPU consumption of the recognition system and the loss of image data after processing are indicators that need to be considered. So we did the following attempts and information review.

4.1 Comparison and analysis

At present, the more popular algorithms can be divided into two categories. One is based on Region Proposal's R-CNN system algorithms (R-CNN, Fast R-CNN, Faster R-CNN). They are two-stage and need to be used first. Heuristic method (selective search) or CNN network (RPN) generates Region Proposal, and then performs classification and regression on Region Proposal. The other is one-stage algorithms such as Yolo and SSD, which only use one CNN network to directly predict the categories and positions of different targets. The first method is more accurate, but slower, but the second algorithm is faster, but less accurate. This can be seen in Figure 4.1 .

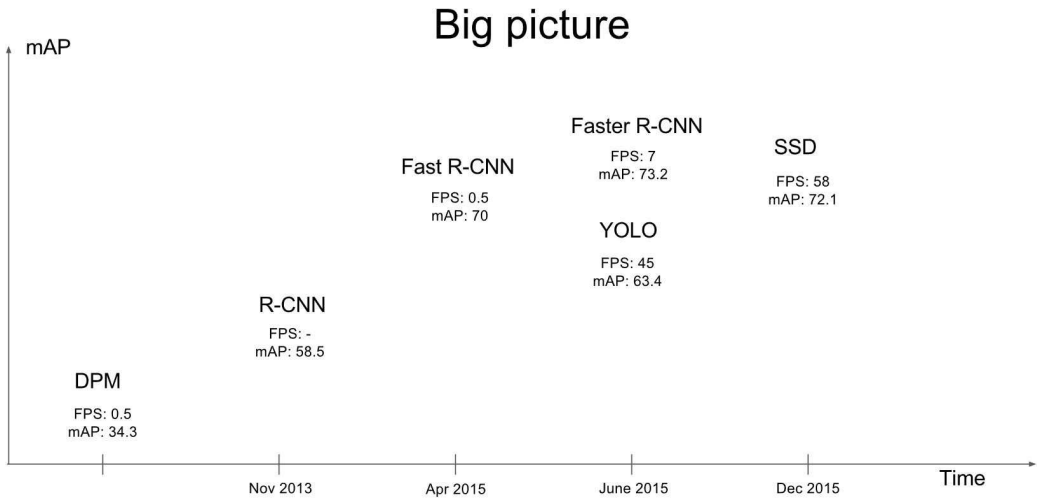

Figure 4.1: Progress and comparison of target detection algorithms

Among them, mAP represents an index for measuring the accuracy of recognition in target detection. Since the calculation principle is too complicated, it will not be described here. FPS can indicate the speed of processing. As can be seen from the figure, the recognition accuracy is high and the processing speed is fast. There are mainly two types: YOLO [3] and SSD. So these two methods are my main research directions

Later, we consulted more scholarly papers, and we saw more parameter comparisons between the two methods. After commemorative updates, YOLO has been updated to the third generation, and YOLOv3 performance has been greatly improved in all aspects. YOLOv3 mainly uses techniques such as bounding box prediction, category prediction, and prediction at different scales to improve recognition speed while ensuring accuracy. Bounding box prediction uses a dimension cluster as the anchor box to predict the bounding box. Each box then uses multi-label classification to predict the classes that the bounding box may contain. It does not use softmax because we found that it has no impact on performance, but just uses a separate logical classifier. During training, it uses binary cross-entropy loss for class prediction. In addition, YOLOv3 can predict boxes of 3 different sizes, which significantly improves the accuracy of recognition. In the past, YOLO was not good at detecting smaller objects. With the update of these technologies, we now see that this situation has changed. Due to the new multi-scale prediction method, we see that YOLOv3 has relatively high APS performance.

Figure 4.2 compares the performance of the latest smart recognition detectors. It can be seen from this figure that under the same level of hardware support,

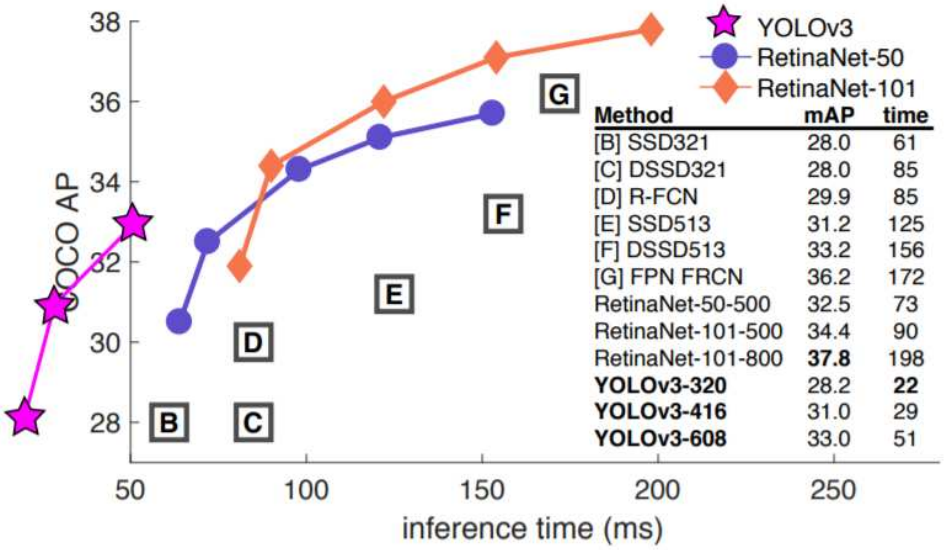

Figure 4.2: Performance on the same GPU[3]

YOLOv3 [4] has the fastest processing speed and the accuracy is also relatively good. Compared with RetinaNet-101, which has the best accuracy, the accuracy is not much different, but the speed is more than eight times. So obviously, the overall performance of YOLOv3 is the best, and it is the most suitable for my project.

4.2 Face recognition yoloface

The first step is face recognition. After much searching, we finally found yoloface, a feature pack that can recognize faces with deep learning based face detection using the YOLOv3 algorithm

4.2.1 Application

Yoloface [1] is a mature pre-trained deep learning model, you can directly add the pre-trained YOLOv3 weights file. So it is very convenient to use. We directly use the yoloface code provided by sthanhng on github. This code is very comprehensive and can perform image recognition, video recognition, and real-time camera recognition. Although our data is stored in *rosbag* in the form of video, the video actually consists of many sequential image frames. And because the new *rosbag* needs to be recorded, the new *rosbag* needs to be synchronized with each frame of the original *rosbag*, so the data must be recorded frame by frame in

the form of an image in the end. Then we just need to perform image recognition. yoloface uses OpenCV Deep Neural Networks (dnn module), OpenCV dnn module supports running inference on pre-trained deep learning models from popular frameworks such as TensorFlow, Torch, Darknet and Caffe. And because depth needs to configure a lot of dependencies, we use Virtual environment to run yoloface, this allows us to experiment with different versions of dependencies.After the configuration is complete, the test results are as follows. After the configuration is complete, we chose a picture with blurred faces and a large number of faces for testing.Figure 4.3

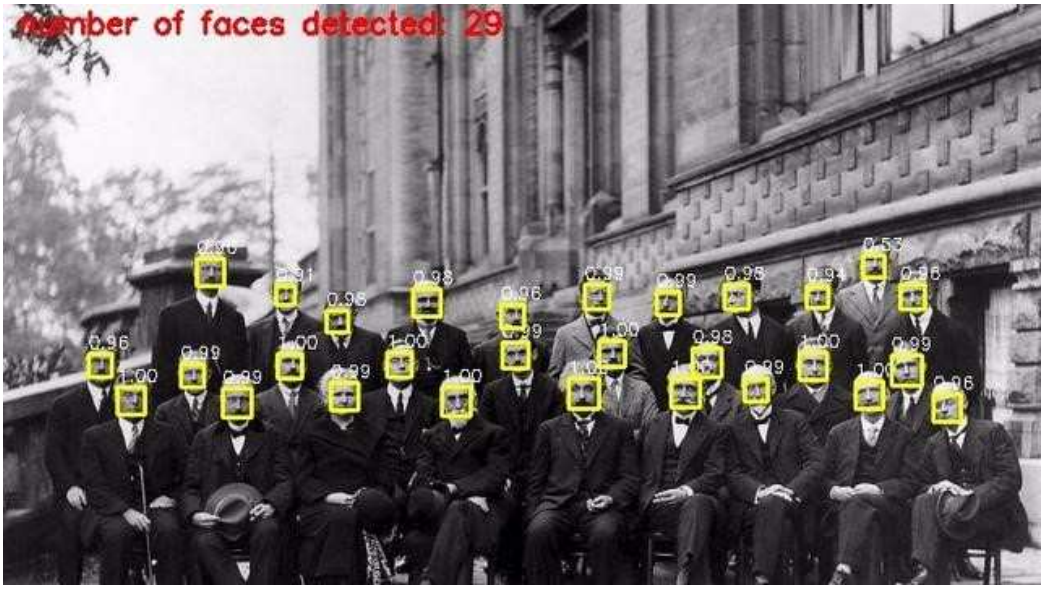

Figure 4.3: Test Results

As can be seen from the figure above, although the facial features are already blurred and very small, and there are a lot of them, all the faces have been identified and framed by a yellow box, and there is a quantitative statistics. The entire recognition process is only about 2 seconds. This shows that the recognition speed and accuracy of yoloface are very reliable. In combination with my project, because the project requires automation, yoloface itself can only process one image at a time, so a script is needed to assist in automatic processing, which will be explained in detail later.

4.2.2 Image recognition area blur processing

The original yoloface will recognize the facial area of the image. The specific method is to get the four coordinates of the top, bottom, left, and right of the

image face area, and then mark it with a yellow rectangular frame. However, we need to perform mosaic processing on the face area of the image, so we need to modify the marked yellow box part in the original code.

In the original code, the function that draws the v yellow box has obtained the upper, lower, left, and right coordinates of the face area, and the image matrix. The image matrix is a two-dimensional array, and the number of each point represents its color. Mosaic is to average multiple pixels into one value, which is equivalent to greatly reducing the resolution of the image.

Algorithm 3: Mosaic in designated area

Data: Image matrix:IM, The value of the image matrix IM (x, y):$IM_{x,y}$, Four coordinates of the face area :top, $bottom$, $left$, $right$

```
n ← top;
m ← left;
while n<bottom do
    n ← n + 10;
    while m<right do
        m ← m + 10;
        IM_{n−5:n+5,m−5:m+5} ← IM_{n,m}
    end
end
```

Algorithm 3 is equivalent to changing 10*10 pixels to the same color, and looping this algorithm in the face area successfully mosaics the face. Changing the value of 10 can change the blur degree of the mosaic.

The recognition area of yoloface is generally slightly larger than the face and is rectangular. If the entire rectangular area is covered with mosaic, some unnecessary parts will be covered and it will be very unsightly. So at the teacher's suggestion, we will upgrade the face mosaic. According to the shape of the face, the best way is an oval mosaic. The ellipse needs to determine the position of the center of the circle, length and width. Obviously, the center of the rectangular area identified by yoloface here is the center of the circle, the length is the length of the rectangle, and the width is also the width of the rectangle. After modification, as follows

Calculate the center, length, and width of the circle based on the four coordinates of the face area: up, down, left, and right. We bring these data into the ellipse equation and we get the ellipse mosaic. However, it can be found from the test results that this method has some errors. The entire ellipse has a slight translation to the upper left corner. After checking the formula multiple times and analyzing it, we found that the reason is that each pixel is theoretically a small square block

Algorithm 4: Oval Mosaic in designated area

Data: Image matrix:IM, The value of the image matrix IM (x, y):$IM_{x,y}$, Four coordinates of the face area :top, $bottom$, $left$, $right$

$w \leftarrow bottom - top$;
$l \leftarrow right - left$;
$x \leftarrow (right + left)/2$;
$y \leftarrow (right + left)/2$;
$n \leftarrow top$;
$m \leftarrow left$;
while *n<bottom* **do**
  $n \leftarrow n + 10$;
  while *m<right* **do**
    $m \leftarrow m + 10$;
    if *square(w)*square(n-y)+square(height)*square(j-x) < square(w*h)/4:* **then**
      $IM_{n-5:n+5,m-5:m+5} \leftarrow IM_{n,m}$
    end
  end
end

that displays a color. However, it is inaccurate to use a coordinate to represent this square block. To be precise, the coordinates represent the position of the top left vertex of the square block, so dividing the ellipse by coordinates will cause some positional errors in the entire ellipse. But it is small, so the impact is not significant and can be ignored.

4.2.3 test

I tested my modified yoloface using about 30 images of different situations. First of all, yoloface can detect all faces, which proves the reliability of YOLOv3 again. Five of the 30 images have very large face areas. At this point, we found that the unit block of my mosaic was too small, making that the face was not sufficiently blurred, and the human eye could barely recognize it. There are also five small facial areas in the five images. The test results of these five images do not show that my mosaic is elliptical, because maybe the entire facial area may only have four mosaic unit blocks. Only the remaining 20 images turned out to be good. This result shows that the optimal size of the mosaic unit block will change as the image size changes. The best solution is to change the size of the mosaic unit block into a variable, which is proportional to the length and width of the face

recognition area.

4.3 License Plate Detection and Recognition

After the face recognition and processing of the image, the license plate recognition and processing is needed. Here we choose a license plate recognition system similar to yolo that uses Darknet deep learning framework: alpr-unconstrained

4.3.1 Application

We learned that Darknet deep learning framework is an open source neural network framework written in C and CUDA by Joseph Redmon. It is fast to install, easy to install, and supports CPU and GPU computing. YOLO is also built on this framework. All the reasons we choose it are the same as YOLO.

Serjim Silva has the original code on GitHub. After downloading from GitHub, alpr-unconstrained [5]is also a pre-trained intelligent recognition system, which is very convenient to use. With yoloface's experience, various environment configurations are complete. After the configuration is complete, the test results are shown in Figure 4.4.

Figure 4.4: alpr-unconstrained test results

As can be seen from Figure 4.4, the information of the recognition results is very detailed. First, the entire vehicle area was identified and framed by a yellow rectangle. Then the license plate was recognized in the area of vehicle and framed in red. Finally, it can also identify the code number in the license plate

and display it on the image. After we researched its original code, we found that the entire recognition process has several steps. The first step is to identify the vehicle area in the image . Like the yoloface, a bounding box is used. After the vehicle area is identified, the area is cut out and saved as a new image with a text file that records the position of the new image in the original image. The second part performs the second intelligent recognition on the new image, which means identifying the position of the license plate in the area of vehicle. If the license plate is recognized, it will be cut out and saved as a new image as in the previous step , and a tex t file will be saved with its position in the image of the car. Then the third recognition, the number in the license plate image. Finally, mark the vehicle, license plate and license plate number in the original image according to the text file of the location.

After understanding the whole process, we admire its meticulousness and rigor. The license plate is not like a human face, and has very unique characteristics. If the license plate is directly recognized, perhaps the billboard and road signs will be recognized as the license plate . So first identify the area of vehicle, it is rigorous to identify the license plate within the area of vehicle.

4.3.2 Image recognition area blur processing

The image features of the license plate are relatively simple, and the content is just a series of numbers. If you use mosaic processing like yoloface, it will still be easy to expose the license plate content, which violates the original intention of my project. Therefore, in the intelligent recognition system of license plate detection, the image blur processing does not use mosaic, and it is directly filled with a color. Since the license plate area is small, the unsightly appearance caused by color filling can also be ignored.

The specific process is very simple. We just change it directly in the last step of the code. After three intelligent recognitions, in the last step, the original image obtained the area of the car, the area of the license plate, and the number of the license plate. We removed all the changes from the original code in the original image, and then used the area of the license plate and an OpenCV function to fill the area completely with white.

4.3.3 test

I tested 30 different vehicle images using modified alpr-unconstrained. First, all visible license plates are completely filled with white, including some license plates that cannot be discerned with the naked eye. This shows that the recognition accuracy of alpr-unconstrained is very high, and the method of filling color

can ensure the privacy of the license plate information. This intelligent recognition system batches the images, which automatically runs with my project. The processing time of 30 images is 3 minutes and 40 seconds. Since each image needs to undergo intelligent recognition at least twice, its processing time is longer than that of yoloface, but it is acceptable.In each of the 30 test charts, the license plate of each vehicle presents a different angle. In the image plane, they are all parallelograms, not regular rectangles. However, the test results can still be fully identified. This turned out to be very reliable.

4.4 Discussion

After learning and using yolo, we discovered many shortcomings of yolo. YOLO performs poorly on small targets. In principle, the YOLO target detector divides the input image into S * S grids, and each unit grid predicts only a single target. If there are multiple targets and small targets in a single cell, YOLO will be difficult to detect and eventually lead to false detection. And after the *rosbag* data test, because the data of *rosbag* is video data, some frames of the image will be moved by the lens And become blurred. So there are often faces that can't see the facial features undetected. Although this has no effect on my project, because it will not leak privacy, it can be seen that yolo's judgment on human faces is not as good as human eyes.

Chapter 5

Bumblebee XB3

In my lab, the *rosbag* data to be processed was collected by a car. At the rear of the car is an ordinary binocular camera, which will record the image data of two topics in *rosbag*. This data can be processed by the method we said above. The front of the car uses a special three-eye camera, Bumblebee XB3. The data recorded by Bumblebee XB3 has only one topic in *rosbag*. After opening this topic, we found that this three-track ghost image. For the processing of this topic, we cannot simply use the *rosbag* c ++ API in chapter3 to handle it. This is a more complicated question.

Bumblebee XB3 is a three-eye camera. It is the third-generation stereo vision product of Point Grey Research. The video from the three angles taken with it will be overlapped into a video and stored in *rosbag*. Figure 5.1.

Figure 5.1: Picture taken by Bumblebee XB3

This overlapping data can be split. In order to comply with EU privacy laws, we need to ensure that users do not reveal privacy when using any of the split videos. So the first problem to be solved is to split the overlapping videos and then perform artificial intelligence recognition one by one. And in order to identify accurately, it is necessary to ensure that the split image is color and clear. After the data is processed, we are equivalent to having three pieces of video data. In order to compress the size of the *rosbag*, and to comply with the principle of

minimizing other elements of the *rosbag*, we need to combine the three pieces of data back into one, just like the overlapping images in the original *rosbag*. This is the new problem we need to solve

5.1 Three-track separation of images

I need to get three sets of image data from a topic recorded by Bumblebee XB3. Each frame in this topic is still an image, but it has data for three images. Then you must use a lot of image processing technology. We think of several ways.

5.1.1 Extraction and separation

The first thing we thought of was the same method as in Chapter 3, using it as a common topic to get a set of overlapping image data. Then use image processing technology to separate it into three groups of images. For testing purposes, we used a launch file to get a set of overlapping images. From the overlapping colors, three colors of red, blue, and green can be seen. It is natural to know that this is a group of RGB-encoded images, and maybe one set of the images is one of the three channels of RGB.

When using OpenCV's function to get a three-channel image, we found that this is not just an ordinary grayscale image in three channels. The image is covered with small square mosaics. Through searching we learned that this is a coding method, Bayer filter. A Bayer filter mosaic is a color filter array for arranging RGB color filters on a square grid of photosensors. Its particular arrangement of color filters is used in most single-chip digital image sensors used in digital cameras, camcorders, and scanners to create a color image. This can also explain how the color information of the three sets of images is stored in a set of images.

I tried the Bayer to rgb algorithm. Due to the different encoding order of Bayer, there are several algorithms. After trying it, we finally determined that my Bayer data was RGGB encoded. Images in the Bayer format are single-channel, each point has only one value, Fifty percent of which is green, Twenty five percent is red, and Twenty five percent is blue. In order to complement the values of the other two channels, it is sufficient to obtain the average of the values of the same color around the point. According to this method, an accurate original image is indeed obtained, However, we cannot guarantee that the quality of the image will not be lost, because it cannot be distinguished with the naked eye, and this method is tedious, requires many steps, it is very inconvenient to operate. Because these methods are existing, and Bumblebee XB3 is a commodity, there must be a dedicated coding tool available.

5.1.2 Bumblebee xb3 ros package

With the help of my tuteur, we found a bumblebee-xb3 ros package. The function of this ros package is very comprehensive. It combines the two steps of extracting image messages from *rosbag* and separating and transcoding the extracted images into three sets of color images. With an extract image ros node, you can get three sets of color images directly from *rosbag*, which is very convenient and fast.

Bumblebee xb3 ros package main starts a camera1394 node which will publish a interlaced (combined) bayered image.When the camera1394 node is started, *rosbag* is played. The topic of *rosbag* containing bayer data is automatically processed by this node, and the message is converted into an image matrix and separated. The separation process is completed by node image-proc/stereo-image-proc.Figure 5.2

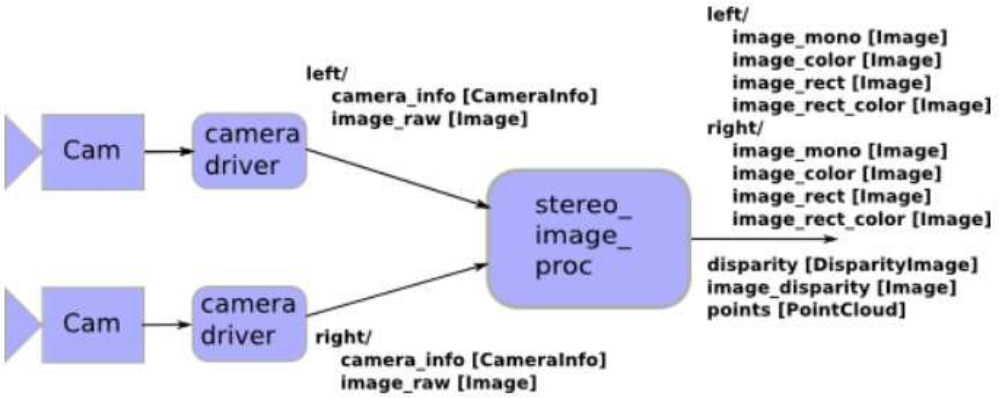

Figure 5.2: stereo-image-proc working process

As shown in Figure 5.2, stereo-image-procx processes the binocular camera data, and the three-eye camera is the same. It will separate the image data of the three cameras into three groups of topics. While testing, we found through the ros visualization tool rviz that when using the bumblebee xb3 ros package, the playback of *rosbag* can detect some topics that the original *rosbag* does not have. Originally there was only one topic about bumblebee xb3 in *rosbag*, and now there are three new topics left/image-color, right/image-color, center/image-color. Through the visualization tool rviz, you can directly see the playback of these three topics. This is indeed the result of the original topic three-channel separation. At this step, you can directly use the method introduced in Chapter 3 to extract the image. Since it needs to work with bumblebee xb3 at the same time, a launch file with multiple nodes started at the same time must be the best choice.

A total of three nodes are set in the launch file, the first one is *rosbag* playback, the second one is Bumblebee xb3 for separating topics, and the third one is

image extraction from the three separated topics after processing.Although in this way, many details of extracting image data, such as extraction time stamp, will be ignored. But because this is a special topic processing, we must first consider the completion function. And this method is simple and can complete the processing of a whole package with one click, which is convenient for project automation

5.2 Three image merge

The Bumblebee xb3 image extraction problem was solved. The subsequent intelligent image recognition is the same as the general situation. Finally you need to re-record the image back to *rosbag*. The last thing our project needs to process is a data set. The ultimate purpose is to upload the data to everyone for sharing after processing. This step is only to record three sets of image data from different angles into three topics and save them in *rosbag*. , So it is acceptable to not combine the three sets of images into one set. But following the principle of exploration, and in order to reduce the size of *rosbag* as much as possible, we still tried to merge three groups of images back into one group of images.

5.2.1 Three channels of RGB

From the data in the original *rosbag*, it can be seen that merging the three sets of images into a set of images is to use the three sets of images as three channels of the new image. These three channels represent R red, G green, and B blue, respectively. There is a merge () function in OpenCV. After trying, we found that these three sets of images must be grayscale to be merged. A grayscale image is actually a single-layer black-and-white image. It originally could only represent the depth of one color. The human eye looks like it is black-and-white. Put it in one of the layers of the new image to represent the depth of one of the RGB colors of the new image.

Algorithm 5: Merge three images into one

Data: channel n of image A A_n, channel n of image B B_n, channel n of image C C_n

Result: channel n of new image IM IM_n^*

split A,B,C;
$IM_1^* \leftarrow A_1$;
$IM_2^* \leftarrow B_2$;
$IM_3^* \leftarrow C_3$;
merge IM^*;

As in Algorithm 5, first, split the three images to get the three channels of each image, and then assign the corresponding color channels to the new image. Finally merge the new three channels into one image. After testing, the resulting overlay image is very similar to the data in the original *rosbag*. The overall color and image position are very consistent. However, the question is, can such overlapping images be re-separated to obtain three different sets of color images? The answer is of course no. Each pixel of an image needs to have three colors of RGB information to become a color image. If you simply use a grayscale image to represent an image, you can only turn into a black and white image in the end. However, the single track of the new image can only use a single channel image. How to use a single channel image to store color information. That is the biggest problem.

5.2.2 RGB to Bayer filter

Bayer filter[7] coding is a coding method that stores color information in a single channel. In the current public information, everyone is discussing the Bayer to RGB method. Because many cameras' raw data and image data are stored in Bayer format, in most practical scenarios, people are looking for an RGB format that can be operated. So we can't get any reference to the RGB to Bayer method. Although the theory of the conversion of the two encodings is known, we can only try to simulate this process briefly.

Although Bayer also has several different encoding formats, the same is that the proportion of the three colors of RGB does not change. Because human eyes are more sensitive to green, green accounts for Fifty percent and the other two account for Twenty five percent each. The specific distribution is shown in Figure5.3

The specific method is to first separate the image into three channels, each channel represents red, yellow, and blue, and then clear the corresponding pixel value to 0 according to Figure 5.4, The three groups of channels are then merged back into an image matrix, and the resulting image is similar to Figure 5.5.

It can be seen that the image is mainly greenish, and there is a very obvious mosaic unique to Bayer on the image. These are the salient features of Bayer coding, and you can roughly judge that the coding result is correct. The first test is whether this image can be directly converted into a color image. This is also the problem of RGB to Bayer, which can be converted directly using the OpenCV function, and the result is successful, and the color can be restored losslessly.Then test if it can be saved in a single channel of the new image. Color images must not be saved, so you must first convert the image to a single-channel image. Single-channel images are black and white to the naked eye, and Bayer's single-channel images still have obvious mosaics. As in Section 5.3.1,I used Bayer's single-

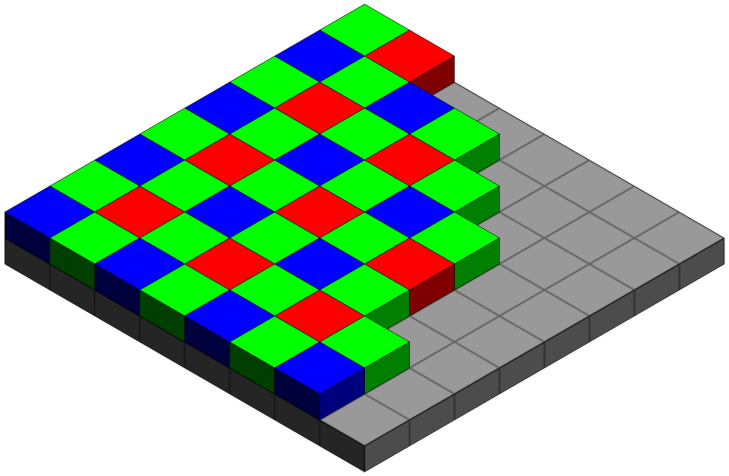

Figure 5.3: Bayer color distribution[7]

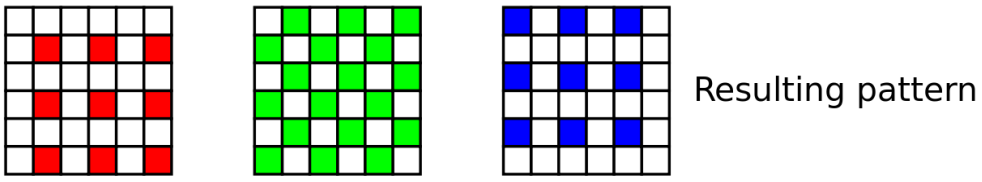

Figure 5.4: The process of RGB to Bayer filter[7]

channel image to reconstruct the overlay image. In order to test result, and also to test whether the user can extract the image normally , we recorded the new image data into a *rosbag*. The recorded topic is exactly the same as the Bumblebee xb3 topic in the original *rosbag*. then we used the Bumblebee XB3 ROS package to get the image viewing results directly from the newly recorded *rosbag*. The result is the image shown in Figure 5.5. We already know that Figure 5.5 can get the original color image. To put it simply, this is our goal, but from the user's point of view, this makes it more difficult for the user to use it, and we did not record the *rosbag* back to its original state. This result may not be used.

Figure 5.5: Results from my Bayer to RGB algorithm

5.2.3 Discussion

In the process of processing the *rosbag* data recorded by Bumblebee xb3, the main problem is the encoding and conversion of the image. In contrast, RGB to Bayer methods and discussions are too scarce. It is true that this process has less practical uses, only for reducing the image size for storage. Although the method of converting RGB to Bayer is well understood, from the results of my tests, we still can't get the Bayer image that can directly get the color image. According to my analysis, maybe my coding is wrong. The technology is constantly updated, and the encoding method of Bayer is also continuously improved and improved. We only use the earliest encoding arrangement of Bayer, and now there are more. such as Panchromatic cells, Fujifilm EXR color filter array, Fujifilm X-Trans filter and so on. However, we have no way of knowing the type of bayer encoding of the data recorded by Bumblebee xb3, so we can't do the right thing. However, we still lament the ingenuity of Bayer coding, using the principle of image color gradation, using Twenty five percent points to save all color data. We still upload

my RGB to Bayer method. Although it may not be useful, we still hope to share the discussion with you.

Chapter 6

System Integration

Now that the main steps have been completed, in order to make the project run automatically and process about 2T of data in one click, we need to use a shell script to integrate all the steps and ensure the process is accurate. In order to complete the project, many details need to be resolved.

6.1 Data collection

The project was officially running, we also need to give the overall performance and the results do an assessment, then the data is essential. Data collection is the basis for scientific data analysis. The accuracy of data collection directly determines the value of data analysis. The main data to be collected is, of course, the number of faces and license plates detected in each frame of the image, and then the time stamp of each frame is used to represent this frame. The source of the data is two deep learning packages, yoloface and alpr-unconstrained. My approach is that after yoloface recognizes an image each time, the number of recognized faces is added to the end of the image name, as is alpr-unconstrained. we finally included the names of all the images in a dat file according to the timestamp for analysis.

6.2 Topic of information

In the original *rosbag*, in addition to the topic of the image, each image topic has a corresponding info topic. There are various kinds of information stored in each frame. Considering the user experience, although this information can be extracted from *rosbag*, it is more convenient for users to record a topic separately.

The topic of info must be synchronized with the corresponding image topic, otherwise all information is meaningless. So when recording the topic of info,

we need to open the corresponding image topic to record synchronously. Since this topic is just some string information, it takes up very little space and does not affect the overall size of *rosbag*.

6.3 Automation script

When all the steps have been tested, we need to connect all the steps with a script. Since the amount of data to be processed is very large, the script needs to be able to process all the data in one click, all we need to do is wait. We use a shell script. First, in order to process all *rosbag*s in one click, the framework of the script needs to be able to traverse all files in the parameter directory. As in Algorithm 6, all

Algorithm 6: File traversal under a directory

```
   Data: directory dire
1  foreach m in the dire do
2      if m is folder then File traversal under a directory(m);
3      ;
4      else process m;
5      ;
6  end
```

*rosbag*s are placed in one path and can be processed at once. At this processing step, it is sufficient to execute each step in a scripting language. There are mainly three image topics in the *rosbag* we deal with. One is the topic of Bumblebee XB3. We need to use three launch files to get three groups of images directly, and then use the *rosbag* C ++ API to process the other two common topics. There are five groups in total. Then the five sets of images were processed with my rewritten yoloface and alpr-unconstrained, and all the privacy-related parts were mosaicked. Finally, the three sets of Bumblebee XB3 images were recorded as a new *rosbag*, and the other two sets of images were recorded as another *rosbag*, and also recorded on the info topic.At this point, the entire project is complete

Chapter 7

Experiments

After the system is completed, we need to do an experiment on the recognition efficiency of the system and its completion. We have added experimental data collection functions to the yoloface and alpr-unconstrained deep learning recognition packages in advance, and make a statistics of the number of faces and number of license plates recognized in each frame. Since there is a lot of data to be processed, we can only make a statistical value and evaluate our project based on this value.

We have selected three days of data, ten minutes of *rosbag* video data per day. The content of the video is a street view of a car driving on the street. The frequency of the video is 15 frames per second. After processing these data, we got the identification number results for three days.Figure 7.1

According to the statistical results in Figure 7.1, we can point out that the number of license plates is much greater than the number of faces. According to analysis, this result is reasonable. The reason is that the on-board camera only captures the forward and backward angles, and it is very intuitive to shoot the vehicles in the front and rear lanes, and the moving speed of the front and rear vehicles relative to the shooting vehicle is slow, so the shooting results of the vehicle are very clear and the recognition effect is very good . However, for pedestrians on both sides of the sidewalk, the shooting angle is very biased, and the relative speed is very fast. Most human faces are so blurred that they cannot be identified. We can only identify pedestrians on the sidewalk when the vehicle is decelerating, or pedestrians on the zebra crossing while waiting for a red light.Compare the data for three days, the first day is noon, the second day is evening, and the third day is afternoon. It can be seen that the intensity of the light will directly affect the recognition result. The better the light, the greater the number of recognitions, and the better the recognition effect. This is because the light affects the clarity of the image, which affects the recognition results.

In general, the results of the identification were similar to those expected and

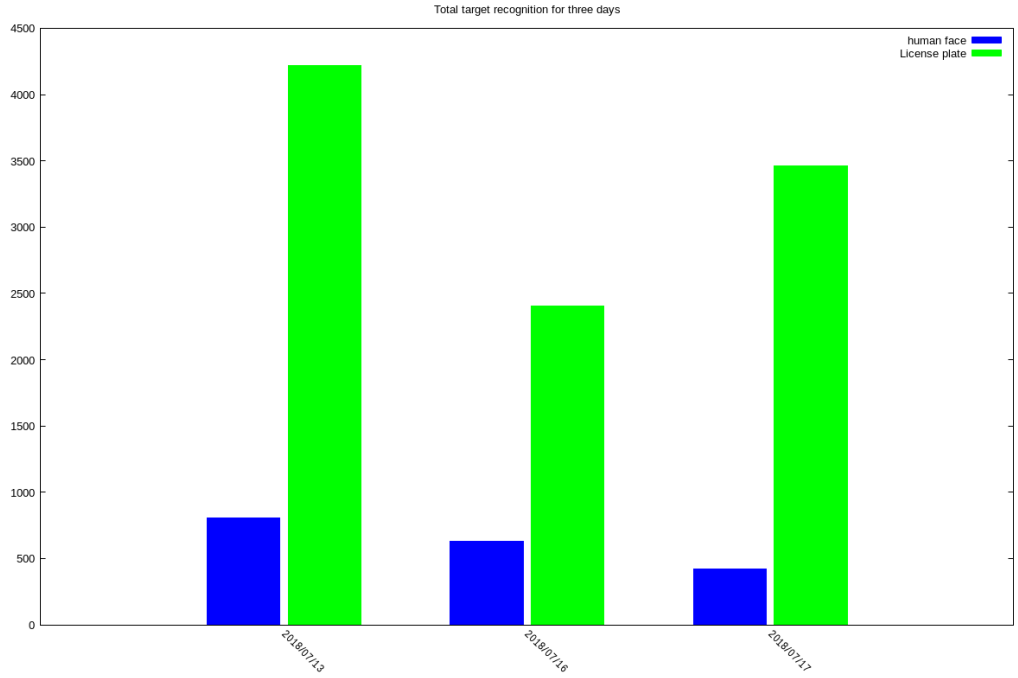

Figure 7.1: Experimental results

met the project requirements. Judging from the results, the entire system still has some shortcomings. The first is that the last three groups of Bumblebee XB3 images cannot be merged into one group. If this can be done perfectly, the data storage size can be greatly reduced. The second is that the face recognition is not sophisticated enough. Although the current results have met the requirements, the human eyes cannot identify those faces that are not recognized. However, if it is family or friends, it may be recognized, and the image processing technology is very advanced now. So if the accuracy is not improved, it is still easy to leak privacy.

However, the project still met the requirements and began to work on the establishment of a data set to meet the EU Privacy Act.

Chapter 8

Conclusion

After experiments, we can basically determine that the actual effect of the project can meet the establishment of the data set. Now some data have been uploaded to the dataset after being processed by my project. In this internship, in addition to the results, the most I learned is the method of doing research. The laboratory environment made me understand that achieving the goal is not the most important thing. The most important thing is to understand the whole process and the details in it, and I can learn a lot of neglected knowledge. Doing research also develops the ability to discover and discover. When you encounter a problem, the first thing that comes to your mind is not how to solve it utilitarianly, but why this problem occurs, and study the cause in detail. This has helped me a lot in my future studies.

Of course, there are still many shortcomings in this process. My plan is very unclear. Due to the lack of a complete plan, many detours were taken halfway. Doing any project in the future must first do a good job of investigation and complete opportunities. To be more specific, my programmers have bad habits. Typography, notes, version replacement, I did very poorly. I think it takes time to accumulate, and I believe it will improve in the future. There are still many shortcomings in this project. The first is that it does not solve the data processing problem of the Bx3 camera. How to convert RGB to bayer still interest me. And my project is not smart enough. It is completely used for the data in my laboratory. If I change a batch of rosbag, I can't deal with it in a targeted manner.These shortcomings, I will improve and update my GitHub bit by bit in the future.

After this internship, I have a new understanding of deep learning and image processing. Maybe this will be my direction for life. I look forward to my future as an excellent artificial intelligence engineer.

Bibliography

[1] Ayoosh Kathuria. Yoloface — GitHub, the free encyclopedia. `https://github.com/sthanhng/yoloface`, 2019. [Online; accessed 16-Aug-2019]. 17

[2] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009. ii

[3] Ali Redmon, Santosh Divvala. You only lool once. *arXiv*, 2016. v, 16, 17

[4] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018. ii, v, 5, 6, 17

[5] S. M. Silva and C. R. Jung. License plate detection and recognition in unconstrained scenarios. In *2018 European Conference on Computer Vision (ECCV)*, pages 580–596, 2018. ii, 21

[6] Wikipedia. What is ros — Wikipedia, the free encyclopedia. `http://wiki.ros.org/cn/ROS/Introduction`, 2014. [Online; accessed 2-May-2014]. v, 4

[7] Wikipedia contributors. Bayer filter — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Bayer_filter&oldid=939392140`, 2020. [Online; accessed 9-February-2020]. v, 28, 29

[8] Zhi Yan, Li Sun, Tomas Krajnik, and Yassine Ruichek. EU long-term dataset with multiple sensors for autonomous driving. *CoRR*, abs/1909.03330, 2019. ii, v, 2