Spatial Feature Positioning using Neural Network based Localized Features Descriptor

Jinze Li; Wenbo Qi; Zhuokun Yang; Hao Cui

Supervisor: Dr. Jun Li

Faculty of Engineering and Information Technology

University of Technology Sydney

Table of Contents

Abstract

We propose a methodology of spatial feature positioning. It combines HardNet and ARKit to measure the spatial distance of detected feature points that it has memory of. The memory is created by storing the feature descriptors of features areas from a query object. HardNet is a refresh model of L2Net with a novel loss function that outputs descriptors with the same size as classical ones.

Project Introduction

Local correspondences have played major roles in many computer vision tasks, e.g. 3D mathematical modelling, robotics control, panoramic stitching, scene interpolation etc. Both detection and spatial measurement require complicated and challenging engineering process, in which expensive high-precision devices might be in use. Apart from the device costs, the processing of these streaming data is also computationally expensive and engineeringly sophisticated. Despite continual research effort, the aforementioned computer vision practices have not gained popularity from an industrial point of view due to its demanding knowledge threshold and time budget.

Traditional detectors and descriptors persist to exist due to their performance, robustness, and efficiency. Although there are many attempts to replace hand-crafted models with the learned ones e.g. DeepCompare, SIFT still outperforms the competitors in a number of computer vision practices such as 3D reconstruction. This situation has been altered after a new paper released at the end of 2017, when Anastasiya et.al proposed a novel loss function that is adopted in L2Net [10] along with test results on a bunch of datasets showing that its performance is better than all of its counterparts [4]. Similar to feature engineering, camera calibration, which also requires complicated hand-crafted processes, is transforming due to the rise of artificial intelligence; the best example and outcomes of this would be the ARKit announced by Apple.

In this project, we propose a combination of technologies that can perform spatial localised feature detection. The algorithm can recognise features that it has memory of in a scene and calculate the spatial distance of this feature with reference to the position of the camera. It is achieved with a combination of HardNet and ARKit. Although this technology is demonstrated on a simple car, it is highly scalable and extensible in terms of potential applications such as classification and behavioural decision.

Literature review

There is only one paper referred and studied thoroughly throughout the project. *"Working hard to know your neighbor's margins: Local descriptor learning loss"*[4], written by Anastasiya et.al, proposes a descriptor learning loss function trains a L2Net [10] for the purpose of localised descriptor matching which outperforms most of its counterparts. It potentially is at a better locally optimal position after convergence.

While traditional feature descriptors such as SIFT are generally driven by experience and intuition, neural network provides a mechanism that generates a descriptor in an end-to-end manner. End-to-end learning largely improves the scalability of feature describing model because it can be done by feeding datasets with ground-truth correspondence [4]. In this project team, there is no expertise on computer vision, so the paper, together with the help from Dr. Jun Li, delivers a good intuition on understanding localised descriptors. In fact, there is no need to understand every single step in the process of generating a descriptor, nor understanding the representation of each element in the 128-d vector. In summary of the paper, a 128-d descriptor generalises all the information including line patterns, edges, and lighting, etc, of a grayscale 32*32 image. An image distance is a measure of the difference between a query object and a detected object. This parameter is fundamental during matching phase.

A Github link is provided in the paper as well and that is where the HardNet code and model is referred from. However, the distance matrix defined for learning loss in the paper was not applicable in this project because of difference in purpose and functionality (i.e. its concepts was not explained in the paper). Dr Jun Li designed a more straightforward formula for distance calculation and the extrapolation done on paper is attached at the Appendix.

Most of the less technical concepts and problems such as potential applications were introduced by Dr. Jun Li during face-to-face hours. These will be further discussed in later section, but there is not much to refer from because both descriptors and AR used in this project are cutting-edge outcomes from senior engineers and academics. In terms of ARKit, only the most fundamental of technology, camera calibration, is used and the syntax was referenced from the ARKit documentation [1] instead of any published paper.

Design and analysis aspects

Conceptual Architecture

For the conceptual architecture, we assign the whole project to four part, the first one is the sensor part which is based on the Apple's recent technology AR [1]. The responsibility of this part is to collect data and transfer it to sever. In this sensor part, we use TCP protocol to transfer to sever because we need to ensure the sequence of transmission and a reliable connection. It means that we must be able to ensure the status of the sensor on the server. Therefore, the TCP is the best choose for this scenario. As for what data to transmit, the data package consists of the image from the sensor and the feature points (included 3D position in the real world and 2D position in image) in this image which is detected by the AR kernel. The second part is the server part, the responsibility of the server is to receive the data from the sensor part, transfer these data to the compute server, generate the command based on the results from the compute server. As the most important part in this project, the computer part generates descriptors based on the image and these feature points. It means the colourful picture is converted to unique digital matrix in this part. This is not a matrix of pixels, but a

matrix of image information feature. And then, we can compare these matrices between two different images with their feature point to determine the similarity of these two pictures. Finally, the car part is composed of the raspberry pi and a simple four-wheeled trolley structure. The raspberry pi which can receive command from the server and control the car. In the future work, we can integrate the entire structure into raspberry pi. At this stage, we use the UDP protocol on the command transmit because it can simply control the car.

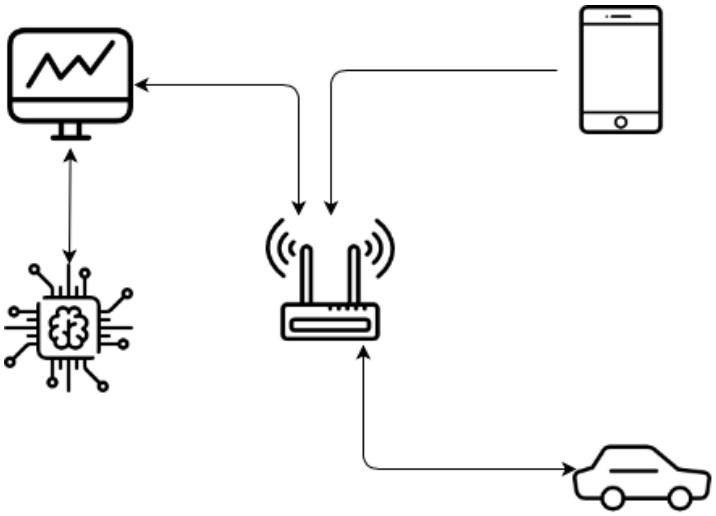


Figure 1 Conceptual Project Structure

Execution Architecture

In the execution part, we divide the whole project into five-part, data collecting, detecting, describing, matching and generate command. The detecting part is on the AR device, the AR kernel can track the objects and understand the scene. It means the AR can generate lots of points in both real world and the real-time image and establish a connection between the points in real world and real-time image. That is the key point to establish a connection between 3D and 2D, the real world and computer vision. And then the data collecting is the communication between the AR device and the server based on TCP protocol. After transferring data, the highlights of the project are on next stage, describing and matching. We use HardNet [4] to describe the part of picture containing feature points. The HardNet has the same dimensionality as classical descriptors (128) [2]. On a forward pass, computing a descriptor takes less than 1 millisecond on a low-end GPU [4]. For the matching part, we develop an algorithm with supervisor Dr, Jun Li which based on the HardNet. The algorithm can determine the similarity between the two put from the HardNet and we will generate the command to car based on the similarity. The last part is connection between the car and server and execute the command which car receive.

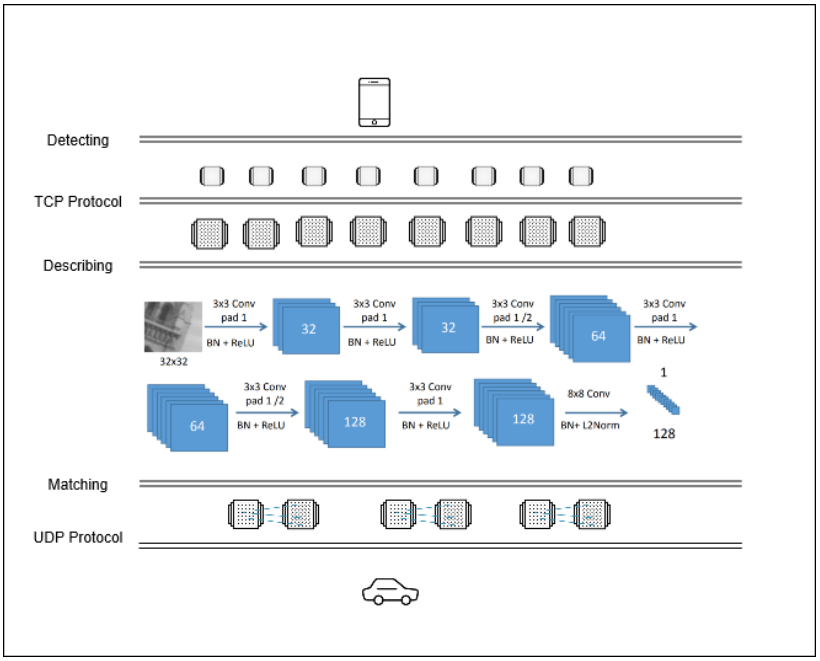
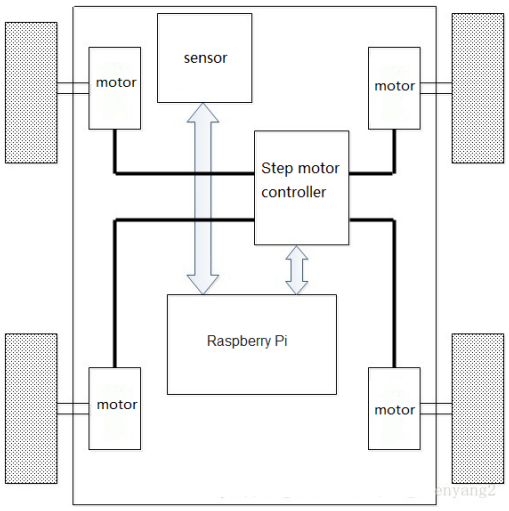


Figure 2 Execution Phases from up to bottom

Implementation

Mechanical Section



The car is powered by 4 DC motors driven by L298N. Control logic signals are generated by the Raspberry Pi and sent to the driver. Sensor is the iPhone and the action commands are received wirelessly using Intranet.

Communication between the server and AR device (TCP protocol)

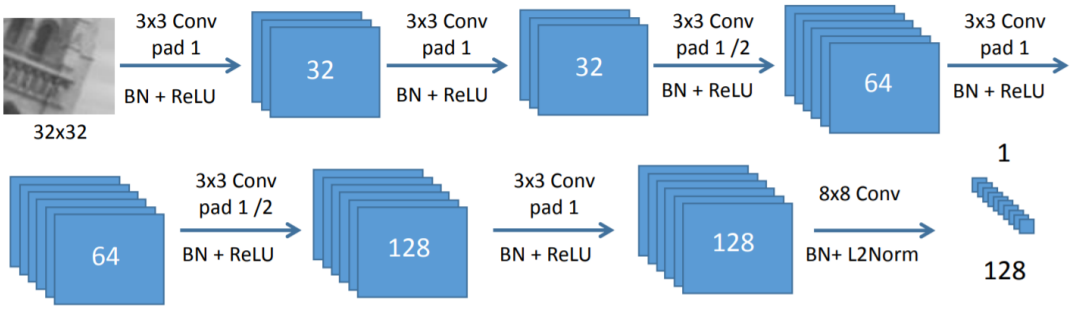
We implement a TCP server based on Python on our server. it is response on receiving the data from the AR client and parse the data to the python data structure as the input of neural network. And the implement a TCP client based on SWIFT4 on the AR device. It can extract the necessary information from the AR stream and send to the server. the client on the AR devices has other responsibility which is the traffic control. It can control the sending rate based on the frame rate on AR kernel.

Communication between the server and car (UDP protocol)

We implement a UDP server in the raspberry pi which support concurrent service based on the

Python. On the server, it can the basic function such as forward, left, right, stop and back. When the command is received from the UDP client, it can perform according to different instructions. For the transmission format. We use the digital from 1 to 5, which represent forward, left, right, and stop each. And then we implement a UDP client based on python as a tool for the next stage.

Neural Network Construction and training



Convolutional neural networks are essentially an input-to-output mapping. Compared to traditional SIFT [2], CNN can learn a large number of mappings between inputs and outputs without the need for precise mathematical expressions between any input and output. While SIFT cannot learn from itself, it also causes unnecessary proposed losses. As long as the convolutional network is trained using known patterns, the network has the ability to map between input and output pairs.

The convolutional layers are identical to L2Net, padding with zeros is applied to all convolutional layers to preserve the spatial size. except to the final one. There are no pooling layers, since we found that they decrease performance of the descriptor. This neural network input is the 32*32 pixels and then output is 128-D descriptor. For training, we use the pre-train model from Affnet (2018) [3] in our project. The inputs are grayscale 32*32 images cropped from the original image. The centre of the small images are the feature points fed from AR device.

Feature Matching

$$d(a_i, p_j) = \frac{a_i * {p_j}^T}{||a_i||_2 * ||p_j||_2{}^T}$$

The output matrix d would be the matching score between stored and detected descriptors, in which row and column numbers can be traced back to the matched ones. From the matched descriptors, it is possible to trace back to the (x_{img}, y_{img}) that crop this particular image, and hence the corresponding spatial information. Matching criterion is one-to-one matching the highest score in each row (query object-based). Then, the scores are filtered by predefined matching threshold that is found experimentally. The column indices of the qualified scores will be used in the trace-back process. And then the command will be generated based on the result from the output matrix. We take an average of the 3D coordinate as the 3D position of the target object and we also have the 3D coordinates of the camera (car) returned from the AR device. Then behavioural decision is hard-coded and sent to the car.

Challenges and solutions

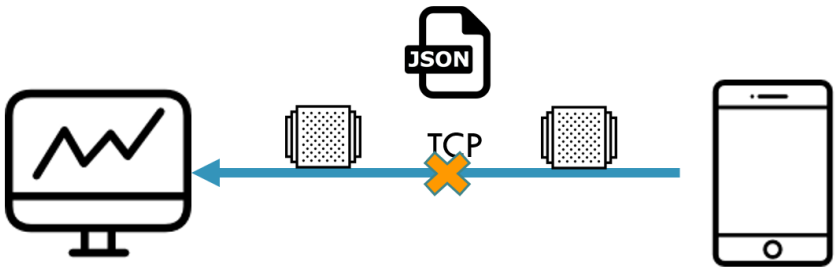
The most valuable technique behind this project is the spatial localised feature detection algorithm. There are only a few challenges on the derivation of algorithm itself. Rather, a lot of efforts went to dealing with less technical problems such as physical demonstration. There

are some weaknesses on this project that is currently unsolved and requires future work to improve.

Data Transmission

There is no official support for socket programming for point-to-point data transmission from Apple. Although Apple provides other modules for data transmission, the speed available is not fast enough to transmit data in a real time manner. Image squeezing and downsampling are prohibited in this project because the correspondence between the features points pixel locations $(x_{\text{img}}, y_{\text{img}})$ and image is extremely important to the accuracy of spatial information of detected features. Socket programming is thus required to deliver undistorted raw data, especially the size of 2K camera image from iPhone.

The only option is to use the third-party modules. This indicates that when an error occurs, it could be the defect of the module or of the code. The solution to coding or using the third party module in a correct way that is not officially documented largely depends on the size of community support. Originally, *swiftsocket* [11] was used. Successful transmission was done when images and feature points were converted into binary. But at the server (receiver) side, it is exceptionally hard to distinguish which part of the serialised binary data is image and feature points respectively, given the fact that, in essence, both of them are multidimensional arrays. The solution is to use JSON, a file type that has the functionality of dictionary, tagging each type of serialised data with a key for the ease of access.



When JSON was used and the App was successfully built and ran on iPhone, the n-d arrays being sent becomes empty for unknown reasons. It was misunderstood that there were something wrong with the code so a lot of time was spent to code-checking and modifying, but in vain; there is nearly no community support to this module. Consequently, after some research on CSDN and JianShu, *CocoaAsyncSocket* was used. All of a sudden, both image and feature points were successfully sent and recovered on the server side. Yet, real time point-to-point data transmission is not achieved due to the hardware limitation of the iPhone.

Unseen Problems

Discrepancies between matrix indexing and pixel location

Sometimes it is really hard to find a logic bug. This occurred when a code was written to crop 32*32 images from the target. Even if a distinctive object was chosen, the saved images does not show its distinctiveness (i.e. the line patterns should be easily recognised even for a 32*32 section). Doubts on the query object and the way the camera was positioned arose. Several days were spent on switching objects but the 32*32 images do not seem to be useful. This problem was not even realised until figure 1 was generated and compared to the ones saved in the directory.

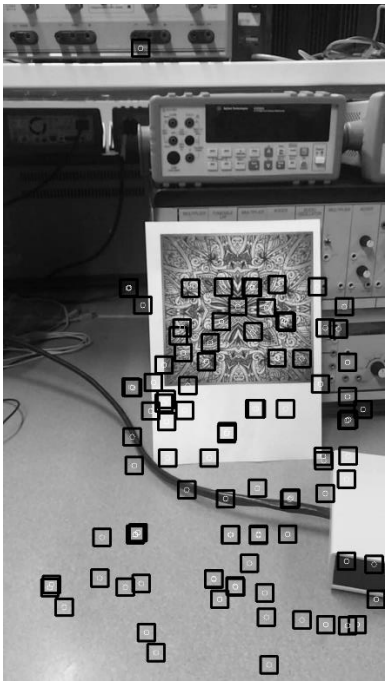


*Figure 3 Positioning the 32*32 rectangle cantered on the AR feature points*

This bug arise from the discrepancies between matrix indexing and the definition of pixel location within the image. As illustrated in the image below, the indexing of matrix and pixel location within the image are inverse to each other. Hence, wrong 32*32 features were constantly generated. The solution is simple once the problem was detected: switch the indexing of image matrix when performing the cropping operation. Although the solution seems to be simple, realising such error and solving it is of great importance to the future steps of storing target features and real-time descriptor computing and image-distance-defining.

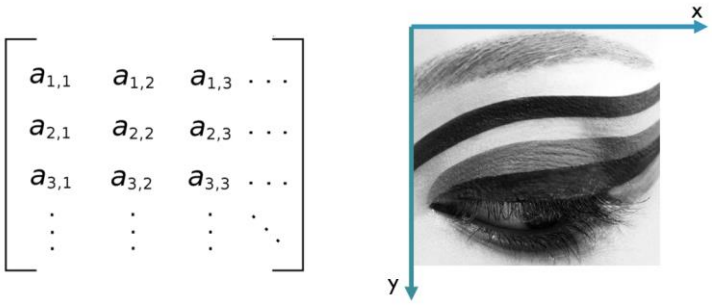


Figure 4 Matrix indexing (subscript) [x, y] and pixel location [x, y]

Feature Matching

As there is no mathematical details on the definition of feature distance equation in the paper ($a_i p_j$ could mean multiple operations if not defined clearly), Dr Jun Li recommended us to use matching functions from OpenCV. Although k-nearest-neighbour with k=1 can match the query features to detected features in a convenient way, there is no guarantee on successful match. This problem can be solved by defining a threshold. But, the range of feature distance is $[0, \alpha]$, where α is an unknown large upper limit and hence a threshold is not definable under this scenario.

The solution to this problem was based on the fact that the physical demonstration part does not require high accuracy matching (i.e. high accuracy requires one query features being matched to one detected features and vice versa with high confidence score; the redundant ones are dumped). In this case, the importance of high accuracy is not as great and hence a more straightforward matching approach can be used. For each single row in the distance matrix, find a distance that is maximum (i.e. distance equation defined by Dr Jun Li makes a positive proportionality between of likelihood of matching and distance). If two rows find the same column, in other words, a detected feature has a closed distance to two different query features, it will be accepted for later computation. Such one-many or many-one relationship will not affect the operation of the car based on the filtered feature points.

Compatibility between the car speed and ARKit

To be able to for the car to achieve the proposed action, the camera need to send the frame that includes the target, in which feature points are successfully generated upon that target. However, as mentioned in the previous section, data transmission speed was limited, so there are some frames dropped purposely in which a frame that includes the target might be dropped as well. In addition, the car is moving at a speed that is too fast for the ARKit to generate feature points that it has confidence on. As these negative events cascade, the mission become less possible. The only technique to alleviate this situation is to speed up transmission rate in a trial and error manner such that the app will not be dead.

Another limitation would be the performance of HardNet. Features with small changes are not guaranteed to have a closed descriptors. Although HardNet has the best performance among its counterparts, the data below indicates that the HardNet might not match up two features even if they are closed. In other words, after getting through a series of cascaded unflavoured events, it is possible that the detection fails given similar feature. All in all, the proposed action cannot be achieved at a success rate that it is expected to be.

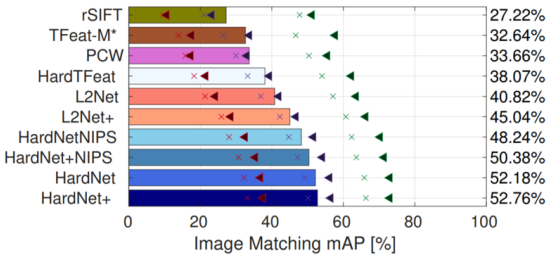


Figure 5 Feature algorithm performance

Outcomes and results

Spatial Distance of Interested Features

With a combination of ARKit and HardNet, it is able to detect its interested localised features, while knowing its spatial distance from the point of the camera measured in centimetre. The detection is done by calculating matching scores between the detected feature descriptors and the ones that are pre-computed and stored. These matching descriptors are filtered by a threshold value that is set by observing the matching scores when the camera is facing different objects. A higher threshold indicates that once matched, it has higher confidence that it finds

the same point. Figure **N** is a comparison between different threshold values under the same scenario.

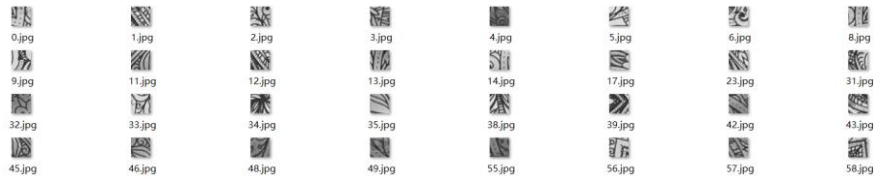


Figure 6 Query Feature Sets

Figure 7 detection with threshold 0.35, 0.3, 0.27 respectively from left to right.

All detected regions lay on desired area on the testing picture, so the performance is good. By setting high threshold, it has more confidence on the matched ones.

Car Movement under different threshold

Threshold greatly affect the performance of proposed action. The videos are attached in the Github: `https://github.com/Qanora/Spatial-Object-Positioning`

Innovation and Potential Application

Traditional and current localised descriptors are limited to pattern recognition in 2D discipline. Our project makes a step forward to give meaning to these feature areas using two state-of-the-art technologies from Apple and academics. The idea behind largely improves the application extensibility of feature detection. For example, if feature points are recognised in a spatial manner, a curved surface can be inferred and calculated as long as there are enough points; by this would be meaningful to some of the robotics control where researchers attempt to control a robotic arm to grasp a cup of water; 3D computer vision has become a challenge to such a seemingly simple control movement. Additional meaning such as classification to the detected

areas might be added using Mask-RCNN, which is a cutting-edge pixel-level-precision object detection and segmentation neural network [12].

Potential application would be 3D environment modelling and absolute world coordinate in Augmented Reality. Interestingly, 2018 WWDC has announced a new version of ARKit that allows developers to make AR Apps in which multiple users can view and interact with the same AR environment from separated devices; such feature would be possible if absolute world origin is used. In the previous release of ARKit, world origin is re-defined whenever a session is restarted, at the point where the application starts to run. Hence, it cannot know that, for example, the water bottle seen from last AR session is the same as the one seen in this session, even if the bottle has not moved. This is because two sessions have different world coordinate and therefore the bottle has two different spatial locations. With this issue solved, it is meaningful to spatial tracking and location inference. A simple analogy to this application would be a person going to home from university. This person knows the location of his home without seeing it. Instead, he knows the location by seeing a Coles supermarket because he infers the approximate location of his home with reference to this supermarket, or perhaps any other reference objects detected by him.

On the other hand, 3D environment mathematical modelling can be achieved simply by importing the spatial locations of the detected points into a virtual environment e.g. AutoCAD, that has its own world coordinate. This would be facilitating to many design, urban planning, etc. use.

Individual Contribution

I was involved in several stages throughout the project. On the iOS development stage, I figured out the particular area in the MVP architecture to fit codes in. Then I and my groupmates can start coding about the part they are assigned to. I wrote the code to extract feature points and camera position, which are essential to detection and car demonstration, respectively. Jinze refined the code later based on his favour because the rest on iOS stage was to construct a client socket to push data to the server. Swift data transmission part was really challenging so, as a novice to programming, I spent my time studying socket programming and configured client and server on the computer-car pair, in which computer completes the feature matching (i.e. this process is essentially filter for the feature points that lay on the features) and behavioural decisions (i.e. commands). The command is then transmitted to the car. The server on the car simply wait for commands in an infinite loop.

I also participated in writing, modifying, debugging the *Utils, matchingmachine, and main* modules on the main computing side, as well as some drawing utilities that can demonstrate the working of this project, for example, circling the filtered feature points. The major process was done on Jinze's computer because PyCharm, a really handy IDE for python, does not support compilation on the python version inside the Windows Subsystem Linux, of which PyTorch==0.3.0, the neural network module on python, needs to be installed in. Also, I figured out the way that can transform the GPU-version of pre-trained model to CPU version, which facilitates future loading process.

Specifically, I did the neural network forward pass modification and testing using fake images generated from NumPy, because the original file from the paper [4] was written on a way that

shows its performance compared to different networks and under different dataset. The purpose of using fake data is to ensure that we get an understanding on what data structure it is supposed to be from the network output, as well as further computation such as finding the distance matrix. The process of getting a distance matrix is a step that extra attention is to be paid to because the mathematical thinking behind several matrix manipulations involves parallel computing concepts; each element in the final matching-score matrix represents the relationship between two 128-d descriptors.

Then I wrote the client and server for the car implementation. Hao Cui was responsible for constructing the car and program the basic actions (i.e. forward, backward, left and right). There were not much of a problem to achieve the communication. I also got involved in the final car testing where a lot of fails occurred. By fail it means the car flipped off by bumping into the wall and the components fell off and a lot of effort went to repairing the car because Hao Cui was not involved in the testing stage and he was the one that is most familiar with the connections between the pins. Lastly, I wrote literature review, challenges/solutions, and outcomes in the final report.

Conclusion

This report presents a new way of making spatial feature detection. By combining HardNet and AR technology, we gradually improved the intelligent functions of unmanned operation, distance perception and measurement, and perception and interaction of objects. As a result, our smart vehicle can automatically find the selected target and plan the most reasonable route without the need for human operation when carrying an iPhone. In this semester, our team has only completed the basis of this neural network object detection project. Additional results require further effort on research and development in the future. We believe that this cutting-edge technology will be strongly recognized and promoted in the future.

Future work

At this stage we only completed the first phase of the three phases of our project's anticipation of the smart vehicle's detection and tracking of objects. However, the object detection we achieve requires us to manually select the feature points of the detected object. In future experiments, smart vehicles can also be upgraded to automatically select and classify feature points in the second stage, perhaps by using sparse coding or other classification neural network; but, most of image classification network is based on image, so we might need to make changes to the L2Net to make a multi-loss architecture. In the third stage, we will use dual cameras to improve the accuracy of feature point selection. The advantage is that it can react more accurately and quickly to detected objects. For example, when two smart vehicles are working at the same time, it is inevitable that they collide with each other when they are looking for targets. When a dual camera is installed, the two vehicles can “communicate” with each other to avoid physical collisions. In addition, at this stage, the route planning of the car is manually planned by our team, in the future work, we will build an end-to-end model to achieve the neural network to automatically control the car.

References

[1] ARKit, Apple Developer Documentation 2018, Developer.apple.com. viewed 10 June 2018, <https://developer.apple.com/documentation/arkit/>.

[2] Dong, J. & Soatto, S. 2015, 'Domain-size pooling in local descriptors: DSP-SIFT', Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on IEEE, pp. 5097-106.

[3] ducha-aiki/affnet 2018, GitHub. viewed 10 June 2018, <https://github.com/ducha-aiki/affnet>.

[4] Mishchuk, A., Mishkin, D., Radenovic, F. & Matas, J. 2017, 'Working hard to know your neighbor's margins: Local descriptor learning loss', Advances in Neural Information Processing Systemspp. 4829-4840.

[5] Rieke, J. 2017, Object detection with neural networks – Towards Data Science, Towards Data Science. viewed 10 June 2018, <https://towardsdatascience.com/object-detection-with-neural-networks-a4e2c46b4491>.

[6] robbiehanson/CocoaAsyncSocket 2018, GitHub. viewed 10 June 2018, <https://github.com/robbiehanson/CocoaAsyncSocket>.

[7] Rublee, E., Rabaud, V., Konolige, K. & Bradski, G. 2011, 'ORB: An efficient alternative to SIFT or SURF', Computer Vision (ICCV), 2011 IEEE international conference on IEEE, pp. 2564-71.

[8] Tian, B.F.Y. & Wu, F. 2017, 'L2-Net: Deep learning of discriminative patch descriptor in euclidean space', Conference on Computer Vision and Pattern Recognition (CVPR), vol. 2.

[9] Xu, J. 2016, Deep Learning for Object Detection: A Comprehensive Review, Towards Data Science. viewed 10 June 2018, <https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9>.

[10] Tian, B.F.Y. and Wu, F., 2017, July. L2-Net: Deep learning of discriminative patch descriptor in Euclidean space. In Conference on Computer Vision and Pattern Recognition (CVPR) (Vol. 2).

[11] swiftsocket/SwiftSocket 2017, Github. viewed 10 June 2018, <https://github.com/swiftsocket/SwiftSocket>

[12] He, K., Gkioxari, G., Dollár, P. and Girshick, R., 2017, October. Mask r-cnn. In Computer Vision (ICCV), 2017 IEEE International Conference on (pp. 2980-2988). IEEE.