AE4317 Individual Assignment: Gate Detection Design

Student: Jianfeng Cui
Delft University of Technology, Mekelweg 5, 2628 CD Delft

Abstract

This report introduces the work on designing a gate detection method for autonomous drone racing, based on the given dataset of the individual assignment.

Keywords: Object detection, ORB feature

1 Introduction

This report aims at solving an gate detection problem for autonomous drone racing, which is based on the dataset `WashingtonOBRace` and for the AIRR facing drone with NVdia Xavier board and cameras. First, I explored three possible methods: ORB feature matching, binary classification, and Mask R-CNN model, to evaluate their performance and computational effort. Then, the combined method of ORB feature and Mask R-CNN is proposed and fine-tuned. The experiment was conducted on my own PC with processor of AMD Ryzen 7 4800h and graphics of GeForce GTX 1650.

2 Gate Detection: Explore Different Methods

2.1 ORB Feature Matching

The first idea came to my mind was to use hand-crafted features, and the ORB[1] feature could be a promising one, since it has been proved useful in SLAM system. As the paper's title says, it is an efficient alternative to SIFT or SURF in the computational effort, which is preferred in this task because in practice the drone needs to react very quickly with limited resources.

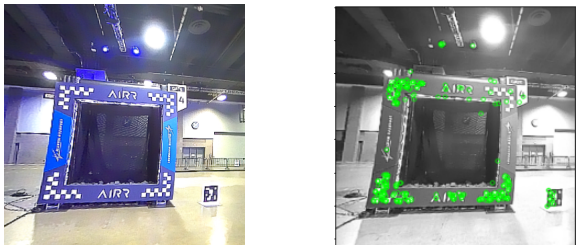
ORB is basically a combination of FAST keypoint detector and BRIEF descriptor with some modifications. For this task, I intended to use a template image and its ground truth mask in a specific frame of the dataset, to match corner points with all incoming images, calculate the perspective transformation and apply it on the ground truth mask to generate a predicted mask of current frame. The brief process is listed as below:

1. Initialize an ORB object using `cv2.ORB_create()`. Detect keypoints and compute descriptors on both the query(template) image and train(current frame) image. An example of detected features on `WashingtonOBRace/img_411.png` are shown in Figure 1.

2. Use `cv2.FlannBasedMatcher()` to initialize a matcher object and match query and train features using `knnMatch()`. For each query feature, k best corresponding candidates were found in train features.

3. Among matches, filter good matches that are correct and representative.

4. Use `cv2.findHomography()` to find the perspective transformation matrix, which is then used for `cv2.perspectiveTransform()` to map ground truth mask to the current frame.



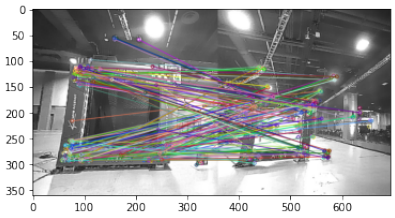
(a) Template image (b) ORB features

Figure 1: ORB feature extraction example

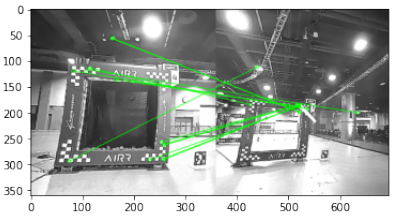
Matches are found by computing the distance between descriptors(the lower the better). Good matches are selected from best candidates, and reserved if it's distance also wins a margin over the second best candidate. And `cv2.findHomography()` uses RANSAC algorithm to further filter out inliers in good matches. The results are shown in Figure 2a and Figure 2b. We see that features are mainly captured on the chess grids located on the four corners of the gate because of high contrast on pixel intensity. During matching, they are very likely to match to the chess grid with other locations(e.g., top-left corner to bottom-left), and also with the lights in the venue. Therefore, I tried two ideas when filtering good matches:

1. Besides distance, add another score term to measure the vertical distance between the query and train corner points. This aims to match corners on the gate top-to-top and bottom-to-bottom. The results are shown in Figure 2c and Figure 2d. We see that this worked but accidentally some features are matched with the lights on similar heights.

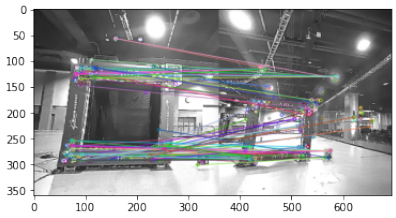
2. Add a score term to measure the vertical distance between each train corner point and the median value, because I assumed that matches at correct positions will still be the majority and the median value stands for a kind of popularity. But in fact this idea fails to improve.



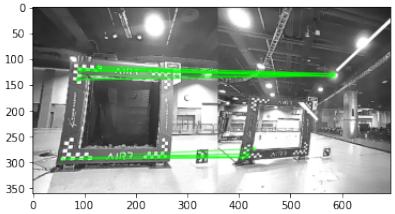
(a) Feature matches between query and train image



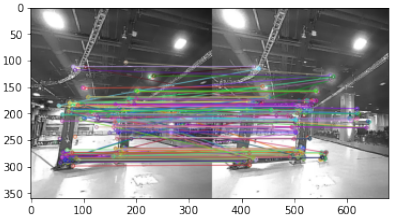
(b) Inlier matches between query and train image



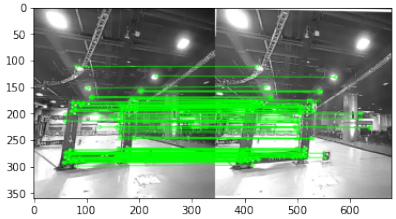
(c) Matches after adding a vertical position distance term



(d) Inlier matches after adding a vertical position distance term



(e) Matches between two consequent frames



(f) Inlier matches between two consequent frames

Figure 2: ORB feature matches

Based these observations, I recognized it hard to use a template to match all frames because of a relatively large different scene, since the matching cannot be "clean" enough all the time. But for two consequent frames the matches are viable for calculate the transformation, as shown in Figure 2e and Figure 2f. The predicted mask by the transformation is shown in Figure 3b. This is also possible to perform when we actually put the algorithm in use: maintain a running mask, and propagate it for each incoming images. Therefore, the proposed method for gate detection is mainly based on the consequent frames feature matching to propagate the mask.

To quantify the performance, an ROC curve is tested for a specific frame, using the propagated mask and its ground truth mask. The result is shown in Figure 4a. For the computation cost, under the given hardware, the processing time for each frame is evaluated as 0.0257 seconds.

2.2 Binary Classification

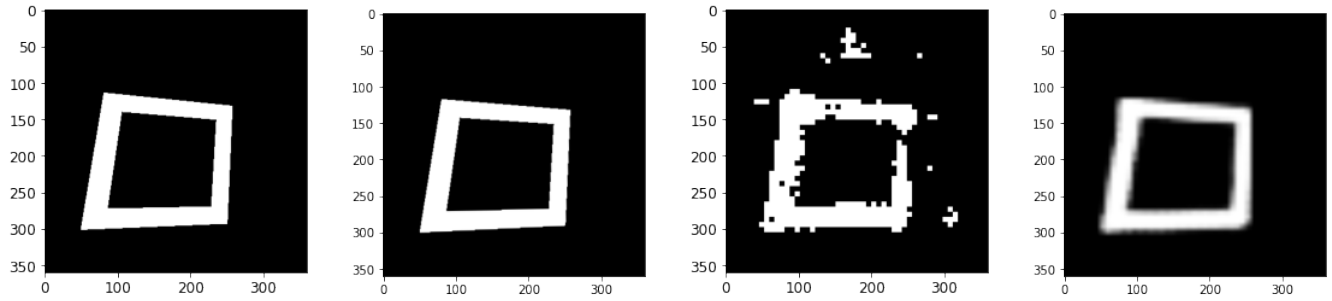
The idea in subsection 2.1 can serve as a basic method, but there are still problems. First, at least there still needs a model to generate a mask so that it can be propagated. Second, for some frames when new gates occur, just propagating the mask of the last frame will never contain them. Therefore, I managed to design a machine learning model to generate the mask directly from the image.

As a basic method, this task can be designed as a binary classification problem, which is also we are most familiar with. The idea is extracting many small square patches in each image, which are labelled by if their centroid pixels belong to gate("gate") or not("background"). A convolutional neural network is then designed to classify those patches. During the actual work, uniform points will be sampled on each incoming image with a limited resolution, and used for extracting patches. After classifying those pixels, nearby pixels can be assigned to the label of their neighbored centroid point(so pixels in the same point grid will have the same label). The work will be much on the dataset preparation, and several functions were modified or reused from the code I learned in course Machine Learning for Robotics(RO47002). The training process is illustrated as below:

1. Sample grid points on the image. There are two strategies: dense uniform sampling and sampling around gates. The uniform sampling is used when put the model into work, generating dense points covering a broad range of image so that the model can predict the labels everywhere. Sampling around gates make use of the `corners.csv` file to get a balanced amount of patches labelled gate and background. Points around and between corners of gates are specially sampled out. This is for training the model, because we want the model to see enough features about the gates, so that it can gain the ability to distinguish the gate and background. One sample result is shown in Figure 5.

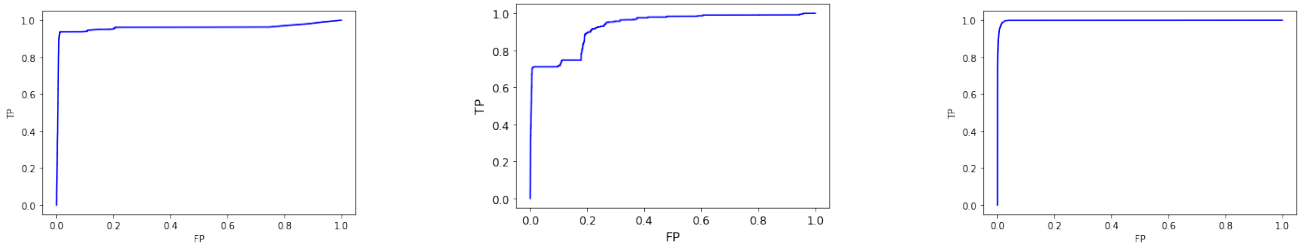
2. Make labels for sampled points using their corresponding value in the mask images. Then extract patches around sampled points with size (28, 28, 3) as the input of the neural network later. The result is like in Figure 6a and Figure 6b. Therefore, the training data and labels are ready.

3. Create and train the model. To prevent going into too much on designing the sophisticated architecture of the neural network, I add an extra fully-



(a) Ground truth mask (WashingtonOBRace/mask_411.png) (b) Predicted mask of ORB (transformation of the last frame) (c) Predicted mask by binary classification (d) Predicted mask by Mask R-CNN

Figure 3: Ground truth and predicted masks tested on `WashingtonOBRace/img_411.png`



(a) ROC curve by ORB feature matching (b) ROC curve by binary classification (c) ROC curve of Mask R-CNN

Figure 4: ROC curves of predicted masks on `WashingtonOBRace/img_411.png`

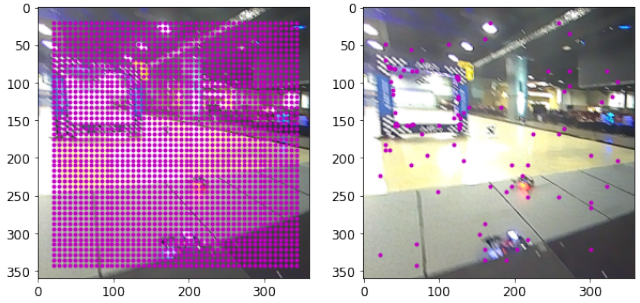
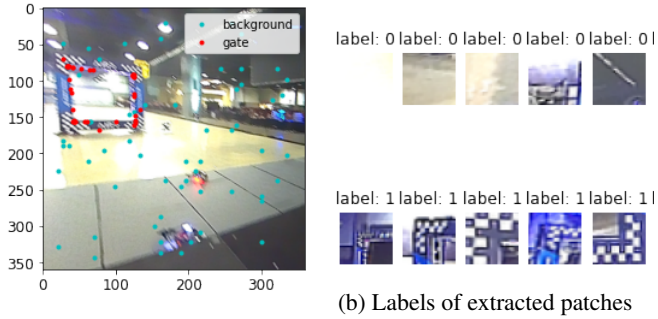


Figure 5: Uniform sampling(left) and sampling around gates(right)



(a) Labels of sampled points (b) Labels of extracted patches

Figure 6: Binary classification labels

connected layer at the end of ResNet18[2] imported from `torchvision.models` to propagate the original 1000 logits into 2. The training is with 15 epochs, Adam optimizer and learning rate 0.001.

4. During the work, each classified centroid point is responsible for its surrounding 7*7 square pixels, which will be assigned as the same label of the centroid.

The result is like in Figure 3c. We see that the performance is not good, with several false positives and limited resolution. The ROC curve is shown in Figure 4b. Moreover, the measured processing time for each frame is 1.9838 seconds, which is pretty slow as expected because for each frame it includes the process to extract patches and in this experiment, the model needs to predict 2209 patches in each image. This method definitely fails to work in reality.

2.3 Mask R-CNN

To achieve higher performance than the binary classification method in subsection 2.2, I used a well-known model Mask R-CNN[3] for image segmentation. Then this object detection problem is design by finetuning the last layer of Mask R-CNN which has been pre-trained on the COCO dataset on our given dataset. The process follows the tutorial provided PyTorch documentation:

1. Prepare the dataset. Wrap the given images and masks

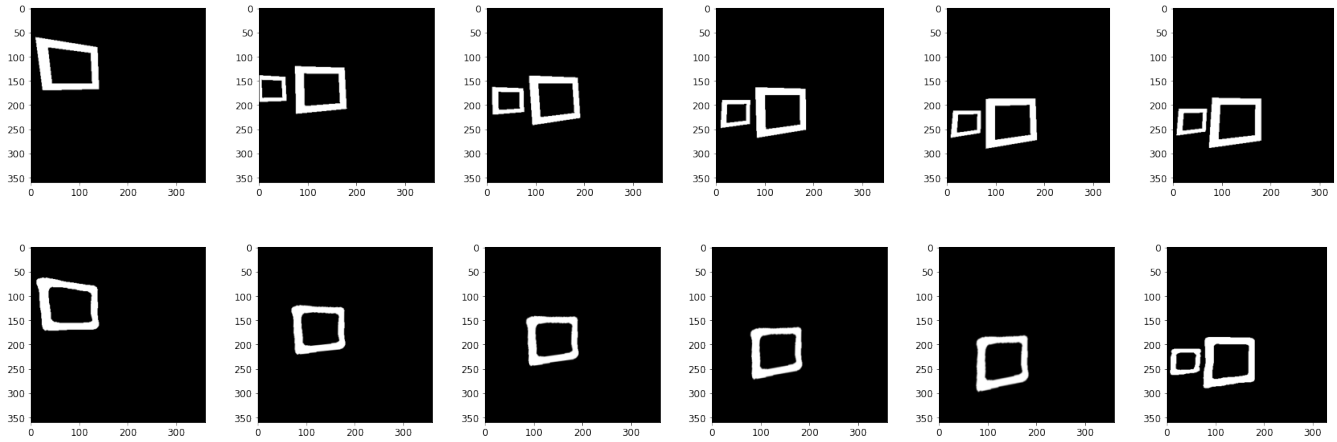


Figure 7: Ground truth masks(the upper row) and predicted masks(the bottom row) of 6 consecutive frames

into the dataset object. In this time we do not need to load the `corners.csv`.

2. Modify the Mask R-CNN model. Replace the box predictor and mask predictor of the pre-trained head with new ones that fit our number of classes.

3. Train the model. The training is with 10 epochs, momentum SGD optimizer, learning rate 0.005, momentum 0.9, weight decay(L2 regularization) 0.0005, batch size 2. Moreover, the data is augmented by using randomly flip the training images horizontally.

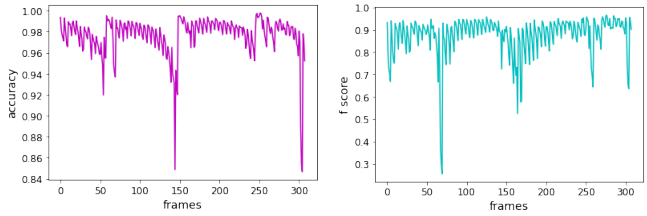
The test result is shown in Figure 3d. We see that intuitively the result is great and by setting appropriate threshold to filter the values in this blurred image, we can then generate our final predicted mask. In practice, this threshold is set to 0.5 after normalizing them into unit values.

The ROC curve is shown in Figure 4c, indicating a great performance. Considering the computational effort, the testing runtime is 0.1675 seconds per frame, which is acceptable but still much longer than that of ORB features.

3 Proposed Method and Fine-tuning

Based on the explore of different methods in section 2, the proposed method is a hybrid: use ORB feature matching and transformation to propagate the mask(e.i., the method in subsection 2.1) for normal frames, and use mask R-CNN to generate and update predicted mask directly from the image(e.i., the method in subsection 2.3) at the first frame and for every 5 frames. The motivation is that ORB features method is fast and works well for consequent frames, but can accumulate the error and cannot recognize newly occurred gates, thus using Mask R-CNN to correct the current mask at a constant frequency, since it will be a little slower but acceptable. Figure 7 shows an example of 6 consecutive frames as a worst case, in which another gate occurs just after initializing the first frame, and is corrected at the 6th frame.

Figure 8 shows the measured performance of the method. We see that as expected the accuracies and f-scores drop for every 5 frames and recover at the update frame. The overall average accuracy and f-score are 0.977 and 0.875. The f-score indicate a good balance between the precision and recall.



(a) Accuracies (b) F scores

Figure 8: Performance of proposed method along frames

The average runtime per frame is measured as 0.0593 seconds, which is ranked between the ORB feature and Mask R-CNN method. Table 1 summarizes the runtime tested till now.

ORB	Binary classification	Mask R-CNN	Hybrid
0.0257s	1.9838s	0.1675s	0.0593s

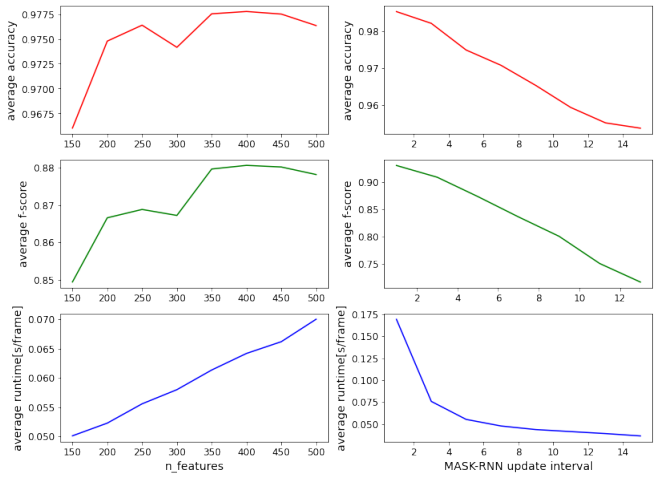
Table 1: Runtime(seconds) per frame of different methods

On fine-tuning the method, I considered two parameters: the number of features used in the ORB feature extraction, and how often the Mask R-CNN is imported to update the mask(i.e., the interval between them). In the above experiments they are set as 300 and 5 respectively. The performance was evaluated in terms of average accuracy, f-score and runtime per frame. The result is shown in Figure 9. For the num-

ber of features, we see that the performance(accuracy and f-score) almost saturate at 350 features, and runtime grows linearly. For the Mask R-CNN update interval, the performance linearly decreases with larger interval, while at the same time the runtime largely drops, which gives us the motivation to choose relatively larger interval.

From the experiment in video[1], RTX2060 is almost 2x faster than GTX1650, and from video[2], RTX2060 is 3x faster than NVidia Xavier. I assumed that the algorithm running on my PC is roughly 1.5x faster than NVidia Xavier.

The drone is still expected to run as faster as possible if using this algorithm, thus the final choice was setting the number of features to 200 and update interval to 6, which results in a 0.0479 average runtime, 0.9736 accuracy and 0.8489 f-score. Therefore, the FPS on NVidia Xavier will be $\frac{1}{1.5\times 0.0479} = 14$.



(a) Effect on number of features during ORB feature extraction

(b) Effect on frequency of using Mask R-CNN

Figure 9: Fine-tune the proposed method

REFERENCES

[1] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[3] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[1] https://www.youtube.com/watch?v=NHrVMRFGKlA

[2] https://www.youtube.com/watch?v=_W_gaKecuVY