Gate detection using feature clustering

J. Hummel
Delft University of Technology

Abstract

A novel algorithm for gate detection based on feature clustering is presented. It is fast and efficient so very suitable for drone racing. The detection rate is still too low for actual applications, though not bad at 60% at a speed of around 100 images per second. A detection is successful if it correctly identifies the corners of the gate. Note that this holds only for the closest gate - it cannot detect more than one gate.

It works by detecting features in the image, without matching them to a reference features. There will always be lots of features in the corner of the gate as it has a chequerboard pattern there, so these can be clustered. From all clusters, the best fit to a square can be selected to be the most likely gate in view.

The detection rate is correlated with the distance to the gate through a parameter of the model. So with knowledge on how close the next gate currently is, the detection rate could be increased.

1 Introduction

Drone racing is becoming increasingly popular, both autonomous and human piloted. Autonomous drone flight is still a huge challenge, as human pilots are still about a factor 2 faster[1].

One of the challenges is detecting the gates through which the drone has to fly. The drone is only equipped with one frontal camera and since drones need to be lightweight, processors and memory need to be small. So a gate detection algorithm should be fast and efficient.

In this paper a novel approach for gate detection using feature clustering is presented. It is trained and tested on images from an actual drone racing competition.

2 Algorithm selection

Four different types of detection algorithms were considered: colour detection, a cascade classifier like Viola & Jones [1], Convolutional Neural Networks (CNNs), and feature-based object recognition.

Colour detection was quickly discarded because the gate doesn't have any colour that particularly stands out from the background, as can be seen in Figure 1.

CNNs were also not tried, simply for the reason that I don't have any experience with them so the risk was too high. Furthermore, it is very doubtful whether a dataset of around 300 images would have been enough to train and test with.

OpenCV has an option to train a cascade classifier[2]. Since this is quite a tedious task, some prototyping was performed. The gates have these very characteristics chequerboard-like patterns in the corners, as can be seen in Figure 1. It was hypothesised that using chequered features could result in good matches. Varying amounts of blocks, with varying widths were tried but the prototyping results were disappointing. It was thus concluded that training a cascade classifier would be too much of a risk.

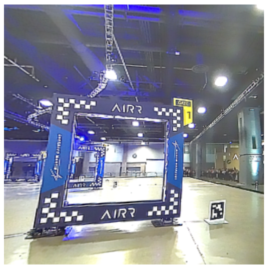


Figure 1: Example of a gate that needs to be detected.

OpenCV has several built-in feature detection and feature matching algorithms. In Figure 2, an ORB detector detects features in both images and tries to match them. As can be seen this works quite poorly. Besides trying to match an entire gate, it was also tried to match each corner individually or match the image to a chequerboard to try to detect the corners but none of this worked well enough.

However it can also be seen that the feature detector often selects features in the corners of the gate. So a novel approach was developed that exploits this.

3 Algorithm explanation

Gate detection using feature clustering has three steps:

1. Detect features in the image.
2. Cluster these features.
3. Fit a square to a combination of these clusters.

These steps are illustrated in Figure 3 and will be further elaborated on below.

[1] `https://www.prnewswire.com/news-releases/lockheed-martin--drone-racing-league-announce-ai-robotic-racing-circuit-champions-give-team-1-million-cash-prize-for-fastest-autonomous-racing-drone-300971338.html`

[2] `https://docs.opencv.org/4.2.0/dc/d88/tutorial_traincascade.html`

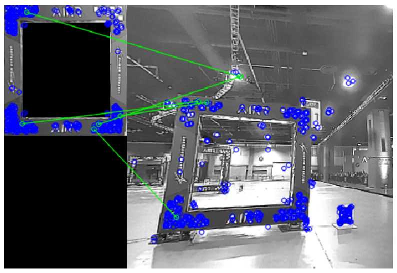


Figure 2: ORB feature detection with flann matching.

Step 1: Feature detection Feature detection for this algorithm needs to prefer the patterns in the corners of the gate and be computationally efficient.

The decision was made to use ORB for feature detection. It uses the FAST algorithm with a circular radius of 9 for keypoint detection and augments this with a Harris corner filter to reject edges [2]. So it chooses lots of features in the corners of the gate.

Step 2: Clustering After detecting the features, they must be clustered. There are lots of different clustering algorithms and after some experimentation, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) was selected.

This algorithm performs well for data that has a few clusters of similar high density points with noise around it [3]. So this is perfect for the keypoints that were detected in the previous step because they have a similar high density around the corners and a lower spread out density in other parts of the image that can be regarded as noise. An additional advantage is that the noise cluster can also be ignored from now on.

DBSCAN requires just two parameters: ϵ and `min_samples`. It works by looping through all datapoints and starting a cluster if there are more than `min_samples` other datapoints within distance ϵ. The points added to this cluster can then let the cluster grow if they can also find enough new points within sufficient distance. If a point can't find a cluster it will simply be regarded as noise.

Step 3: Square fitting To find the best fit of clusters, the algorithm loops through all combinations of centres of clusters and looks for a square shape. Even though the gates are not perfectly square due to perspective, this works sufficiently well to pick them out of the other shapes that can be made by combining clusters.

A good square is said to have the following properties:

- There are four edges that have a similar length and two diagonals that have a similar length.
- When drawing all lines between all vertices there should be $8 \times 45\,\mathrm{deg}$ and $4 \times 90\,\mathrm{deg}$ angles.

To get a cost between 0 and 1, Equation 1 is used, where σ is the standard deviation. It was found to work sufficiently well without weighting between the different terms. Improvement is definitely possible but out of the scope of this article.

$$s = \max(1 - \sigma_{4\text{shortlines}} - \sigma_{2\text{longlines}} - \sigma_{8\text{smallangles}} - \sigma_{4\text{bigangles}}, 0) \tag{1}$$

Currently, only one gate is searched for. However, it might be possible to detect multiple gates by masking the gate that was found and rerunning the algorithm.

3.1 *Tuning process*

The only two parameters that need to be tuned are ϵ and `min_samples` for DBSCAN.

The two drivers for these parameters were chosen to be the detection rate and the computation speed, as these are the main drivers for drone racing. A successful detection is counted when the four corners that the algorithm predicts are in total no less than 75 pixels away from the actual corners of the gate.

Even though the risk of overfitting with just two parameters is extremely small, for good measure the data is split in training and test data with a 50-50 split.

ϵ is varied from 8 to 33 while `min_samples` is varied from 14 to 34. For these parameters, the detection rate is shown in Figure 4 and the computation speed is shown in Figure 5.

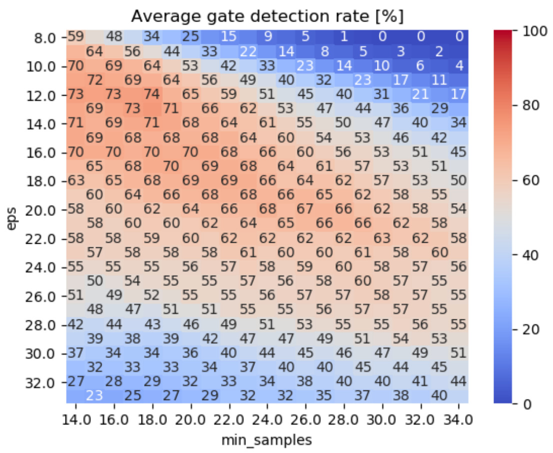


Figure 4: Detection rate as a function of algorithm parameters.

On the training data, the algorithm reaches detection rates of up to 75%, and speeds of up to 160 images per second. A combination of ϵ = 21 and `min_samples` = 25 was chosen as this combines a high detection rate of 65% with a high speed of 105 images per second on the training dataset.

3.2 *Implementation*

The algorithm is implemented in Python and available on GitHub[3].

[3]https://github.com/jesseishi/MAV_individual_assignment

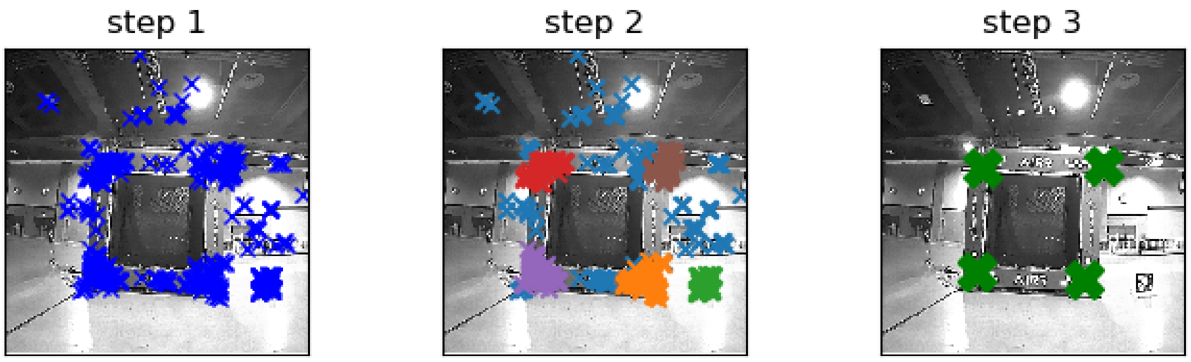


Figure 3: The three steps of the algorithm.

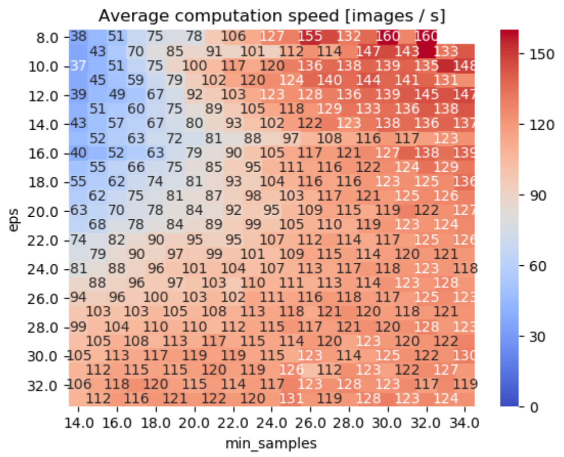


Figure 5: Computation speed as a function of algorithm parameters.

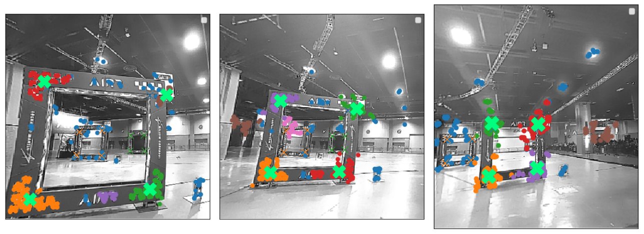


Figure 6: Some successful detections.

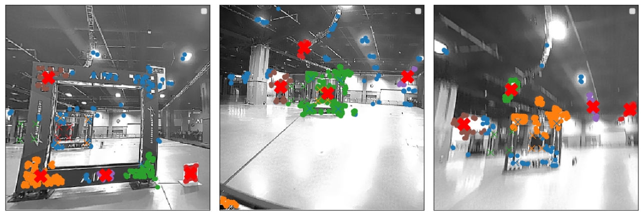


Figure 7: Some unsuccessful detections.

4 Results

On the test data the detection rate is 60% at a speed of 99 images per second. Which is only marginally worse than the results on the training data.

4.1 *Successful detections*

In Figure 6, some successful detections can be seen. It shows the different clusters that the algorithm finds and the final prediction of the corners of the gate. It is very clear that the algorithm has worked as intended in these cases.

More successful detections are shown in Appendix A:.

4.2 *Unsuccessful detections*

In Figure 7, some unsuccessful detections are shown. Some of the top reasons are: corners being out of view, not recognising a corner as its own cluster, or clustering multiple corners together. These last two issues were found to be distance dependent and will be elaborated in the next section.

More unsuccessful detections are shown in Appendix B:.

4.3 *Dependency on distance*

Given a certain `min_samples`, ϵ determines how dense a cluster should be. Since a cluster of features becomes less dense as the gate is closer, it was found that increasing this parameter makes close-by detections more likely. This effect is shown in Figure 8.

In an on-board version of this algorithm, the drone could change ϵ according to the distance to the last gate that was successfully detected. Or since this algorithm is so computationally efficient, step 3 of the algorithm could be run with three different values for ϵ.

4.4 *ROC curves*

ROC curves are only defined for binary classifiers. Since this algorithm is not a binary classifier some workarounds had to be made to make a ROC curve for it.

Once a gate is detected, a line can be drawn between the estimated coordinates of the corners. This line is then made

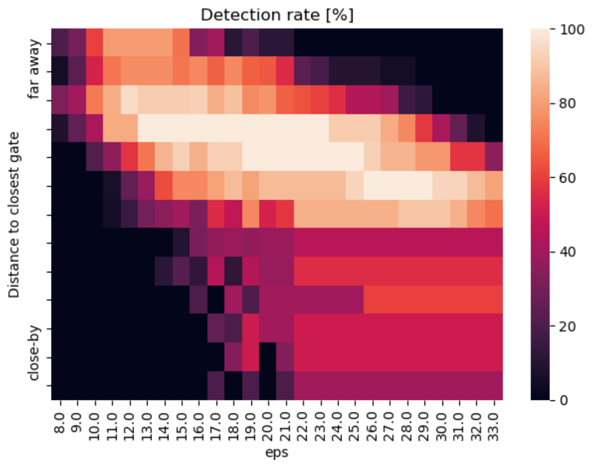


Figure 8: Detection rate as a function of ϵ and distance.

thicker with a certain ratio to the average length of the edges of the estimated gate. This can then be compared to the true mask of the gate, as shown in Figure 9. It is already clear from this picture that getting a very high true positive rate will be impossible since this algorithm only detects the closest gate and will thus not get a match with the mask of the distant gates.

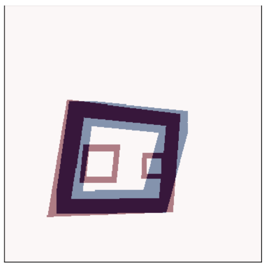


Figure 9: True (red) and estimated mask (blue) of the gate.

Three ratios of the width of the gate to its edge length are used an the resulting ROC curves are shown in Figure 10. With a higher ratio, the borders will be thicker thus the true positive rate can become higher, however this is always at a cost of more false negatives.

A ratio of 0.25 (which is approximately the true ratio) seems to be the sweet spot and can correctly identify 55% of the true positives with only 6% false positives.

4.5 *Computational effort*

In Figure 5 of subsection 3.1, the computational effort was shown as a function of different algorithm parameters. With the selected parameters the speed was about 99 images per second. This is on an Intel i7 and programmed in Python.

This is quite fast, much higher than needed on an embedded application where you would maybe want to process around 25 images per second for avoidance. Furthermore, when implemented in C this algorithm will be even faster.

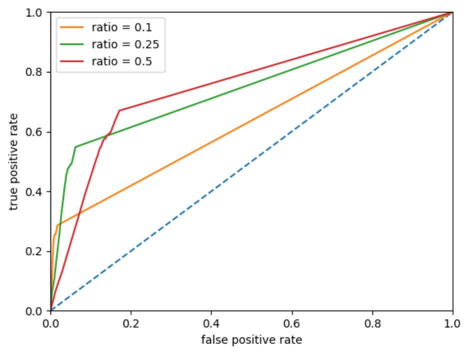


Figure 10: ROC curve

On the other hand, lightweight processors on drones have less computing power and might need to make certain calculations less efficient. For example, when they don't have dedicated silicon for multiplications.

All in all, no stringent requirements on an on-board processor would be needed to run this quite efficient algorithm, but more research is needed.

5 Conclusion and discussion

The presented algorithm is quite novel so improvements are very likely possible. It already achieves a detection rate of 60% at a rate of 100 images per second. Which is quite promising but not good enough yet for actual drone racing.

The background features do sometimes lead to false detections. So if this algorithm could be paired with a feature matching algorithm that can find the chequerboard features in the corners of the gate, it would probably achieve a higher detection rate.

Furthermore, the distance where a successful detection was most likely is dependent on ϵ. Further research could come up with a smart way of continuously updating its value while running the algorithm.

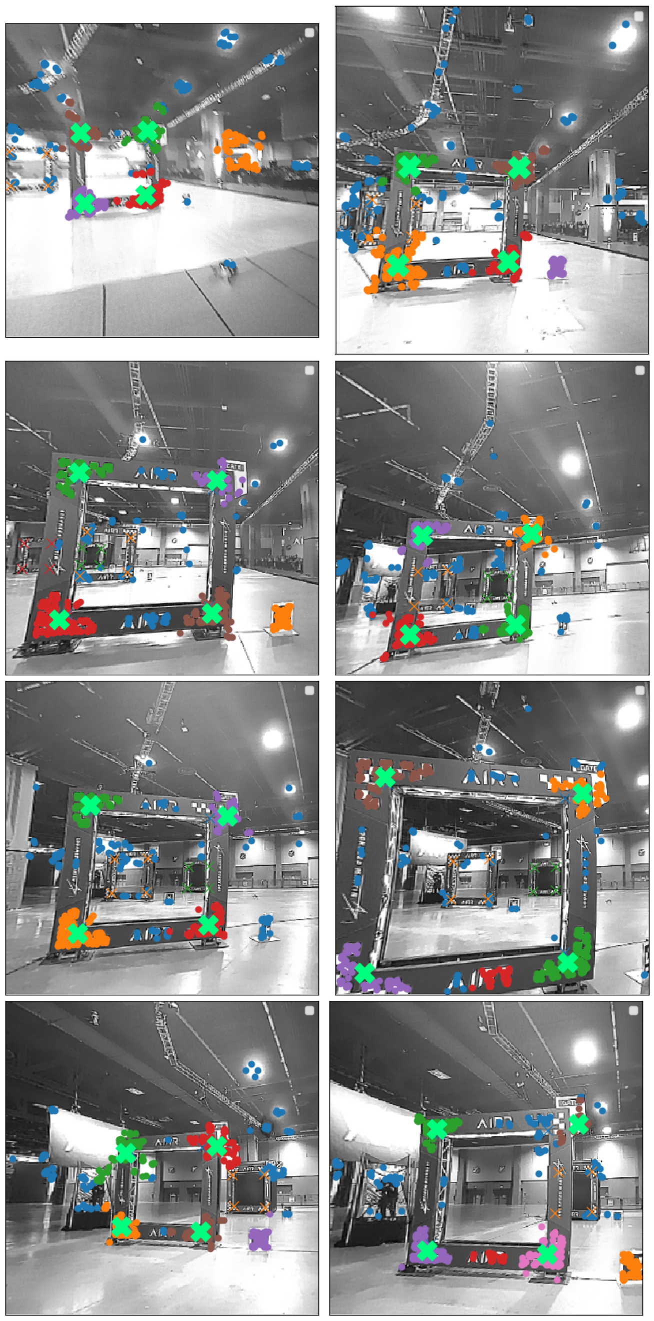
References

[1] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. 2001.

[2] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. 2011.

[3] Martin Ester, Hans-Peter Kriegel, Jiirg Sander, and Xiaowei Xu. Density-based algorithm for discovering clusters. 1996.

Appendix A: Successful detections



Appendix B: Unsuccessful detections