2D Tracking - Descriptors/Detectors

UDACITY

Contents

1 Implementation

1.1 MP.1 Data Buffer Optimization

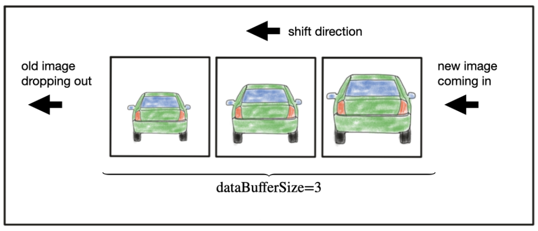


Figure 1: Keypoint count and size distribution - plotted for first image

```
if (dataBuffer.size() == dataBufferSize)
    dataBuffer.erase(dataBuffer.begin());

dataBuffer.push_back(frame);
```

1.2 MP.2 Keypoint Detection

Different keypoint detectors were implemented using the OpenCV module - see figure 2. Classic detectors depend on image gradients and calculate some function of the eigenvalues of the associated hessian. The most famous ones are the Harris or Shi-Thomaso corner detector. The Harris detector was implemented as:

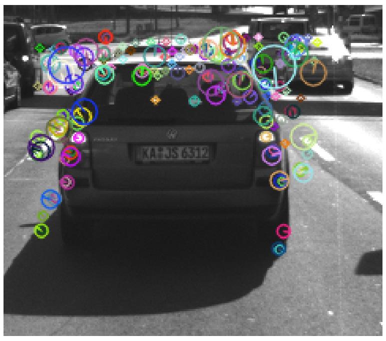
```
cv::cornerHarris(img, dst, blockSize, apertureSize, k, ...);
# some code for non max suppression
```

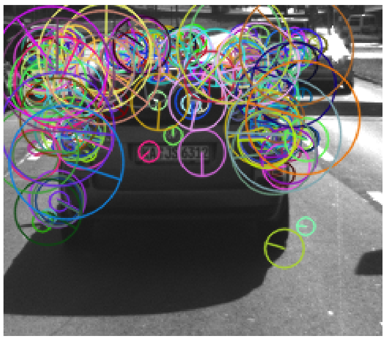
The Shi-Tomaso was implemented as

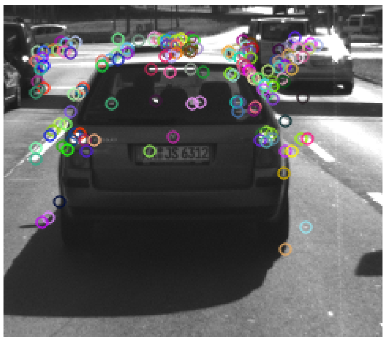
```
cv::goodFeaturesToTrack(img, corners, maxCorners, ...);
```

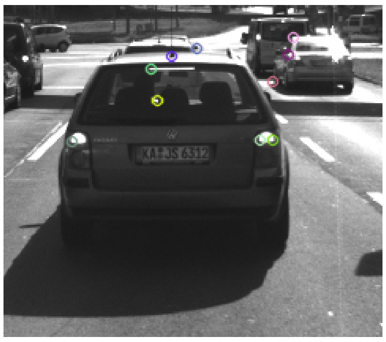
More modern keypoint detectors often don´t necessarily calculate gradients and rely on binary information. There is a unified framework for modern detectors in OpenCV.

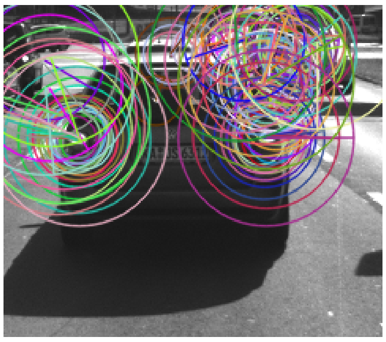
```
detector = cv::SIFT::create();
detector->detect(img,keypoints);
```

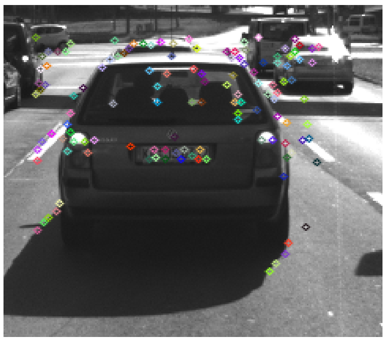

(a) AKAZE


(b) BRISK


(c) FAST


(d) HARRIS


(c) ORB


(d) SHI-TOMASI

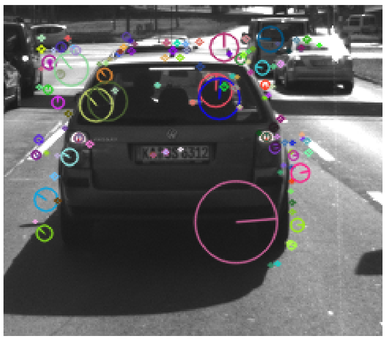

(c) SIFT

Figure 2: Visualisation of the detected keypoints on the first frame over the different detectors

1.3 MP.3 Keypoint Removal

For a collision detection system, the preceding vehicle is the region-of-interest (ROI). Thus, it's better to remove keypoints outside the pre-defined ROI window. This can be done using the 'contains' function.

```
if (vehicleRect.contains(it->pt))
    continue;

it = keypoints.erase(it);
```

1.4 MP.4 Keypoint Descriptors

A variety of different keypoint descriptor methods (BRIEF, ORB, FREAK, AKAZE, SIFT) were implemented. All of them are available as OpenCV DescriptorExtractor classes. For the parametrisation of these descriptors, just default parameters were choosen - as a multitude of parameters exist.

```
 extractor = cv::BRISK::create(threshold, octaves, patternScale);
 extractor->compute(img, keypoints, descriptors);
```

1.5 MP.5 Descriptor Matching / MP.6 Descriptor Distance Ratio

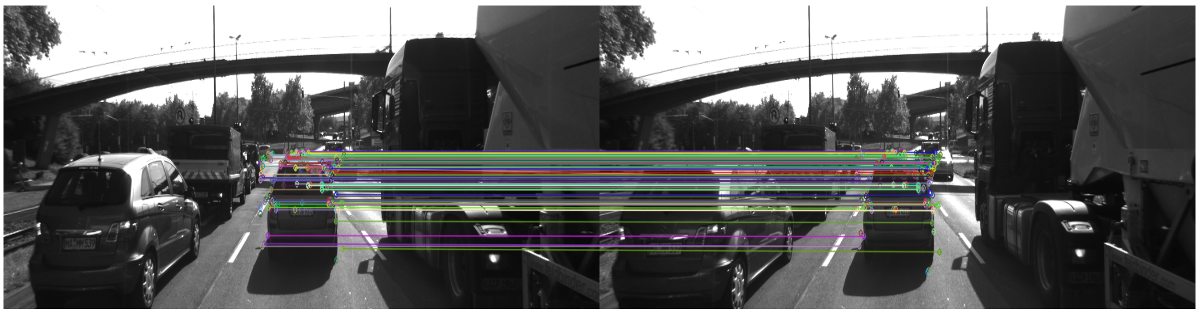


Figure 3: Keypoint count and size distribution - plotted for first image

The descriptor matching is done using a similarity metric. For binary descriptors the hamming distance was used. For continous descriptors L1 or L2 norms can be used. The actual matching can be done in a brute force fashion. This means comparing the keypoint of one image to all keypoints of the other image and take the best match. OpenCV provides DescriptorMatcher class implementations for this purpose.

```
matcher = cv::BFMatcher::create(normType, crossCheck);
```

This would be the single NN approach. To speed up this calculation, the FLANN (fast libraryfor approximating NN can be used.

```
matcher = cv::DescriptorMatcher::create(cv::DescriptorMatcher::
                                          FLANNBASED);
```

For robustification also a K-NN approach was implemented.

```
matcher->knnMatch(descSource, descRef, knn_matches, nMatches);
```

To invoke the matcher the following call can be used:

```
matcher->match(descSource, descRef, matches);
```

1.5.1 Distance ratio

When setting k=2 for the k-NN case, the two best matches are obtained. Calculating the distance ratio between these two matches gives an indication for the robustness of the match.

$$r = \frac{d(desc_a, desc_{b1})}{d(desc_a, desc_{b2})} < \varphi \tag{1}$$

If the distance ratio is low - the pair may be a good match, as the second keypoint is much further away (in similarity metric). In practice often a threshold of $\varphi = 0.8$ is used.

1.5.2 Cross check

To further robustify the matching - cross check matching can me implemented. This means that, only those keypoint pairs are kept, where the matching from ImageA to ImageB and the matching from ImageB to ImageA is identical.

2 Performance Evaluation

2.1 MP.7 Performance Evaluation 1 - Detectors

Task: Count the number of keypoints on the preceding vehicle for all 10 images and take note of the distribution of their neighborhood size. Do this for all the detectors you have implemented.

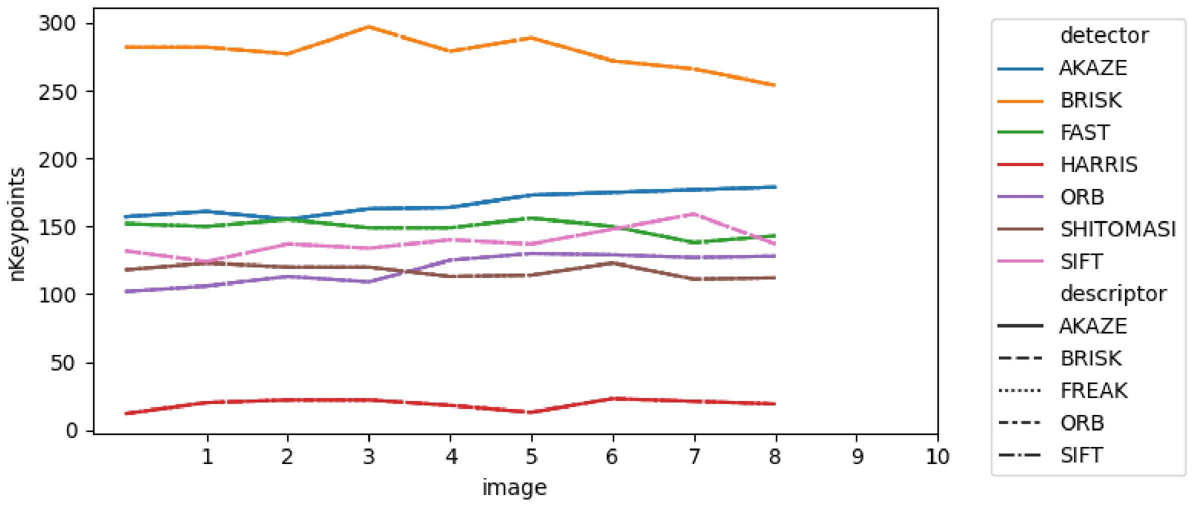


Figure 4: Keypoint count over different images

The figure 5 displays the keypoint count on the preceding and the keypoint size distribution for all implemented detectors. The following detectors do not provide any size information in OpenCV: Fast, Harris and Shi-Thomasi.

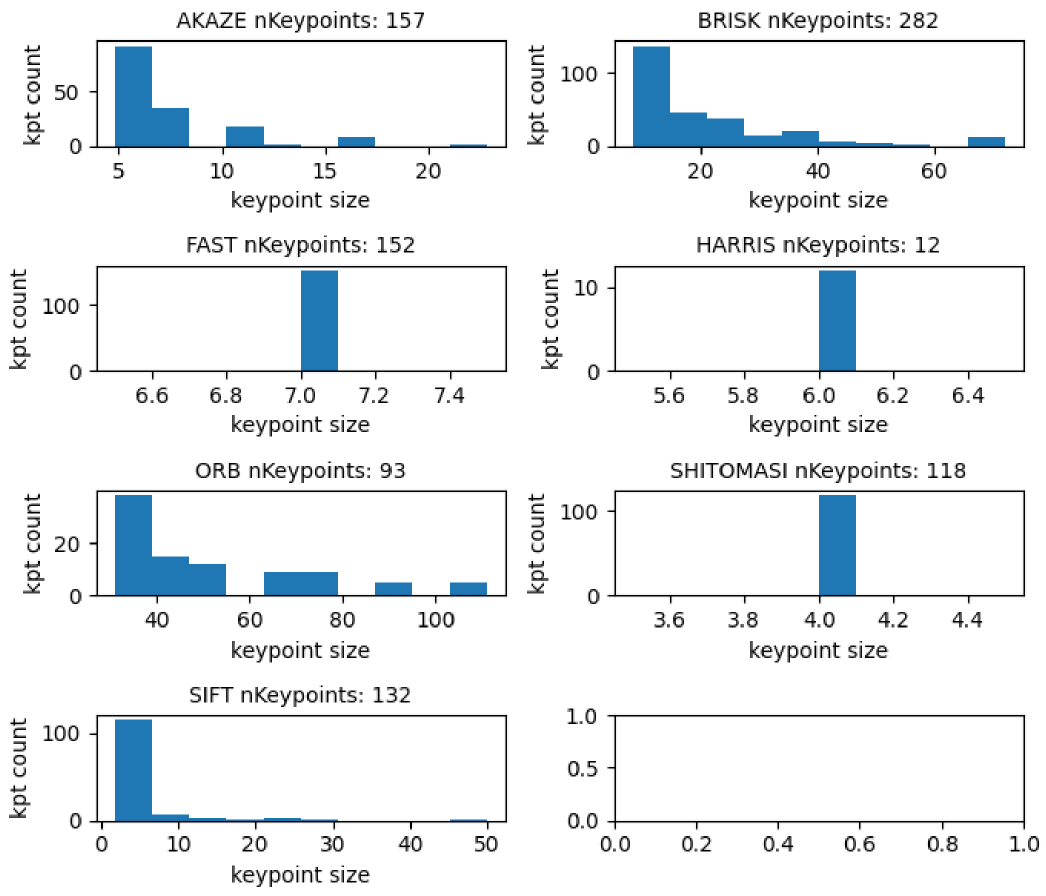


Figure 5: Keypoint count and size distribution - plotted for first image

2.2 MP.8 Performance Evaluation 2 - Matching

Task: Count the number of matched keypoints for all 10 images using all possible combinations of detectors and descriptors. In the matching step, the BF approach is used with the descriptor distance ratio set to 0.8.

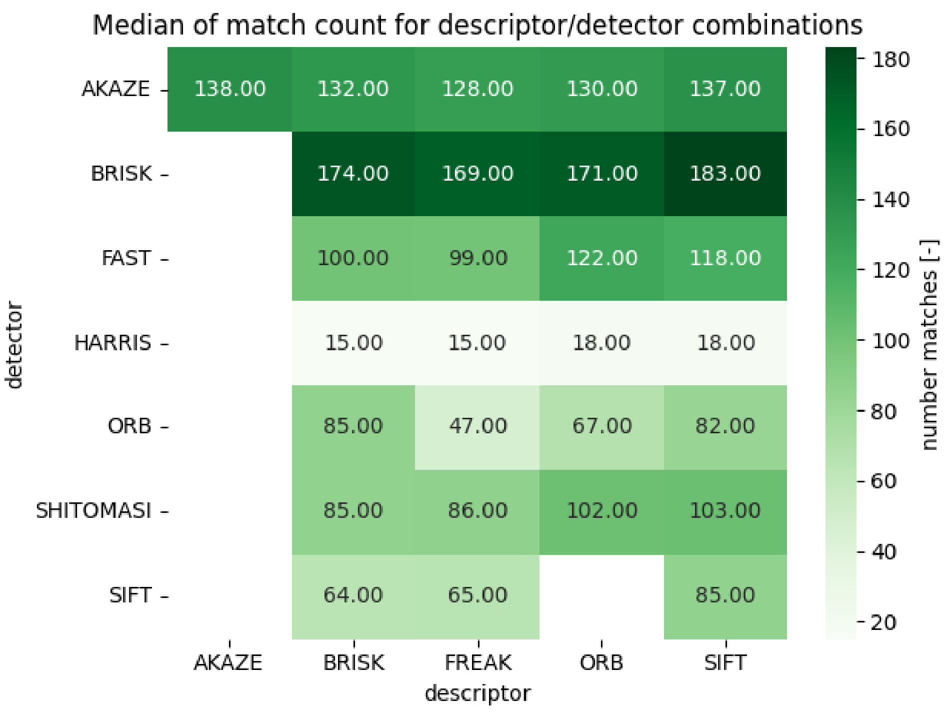


Figure 6: Keypoint count and size distribution - plotted for first image

2.3 MP.9 Performance Evaluation 3 - Timeing

Task: Log the time it takes for keypoint detection and descriptor extraction. The results must be entered into a spreadsheet and based on this data, the TOP3 detector / descriptor combinations must be recommended as the best choice for our purpose of detecting keypoints on vehicles.

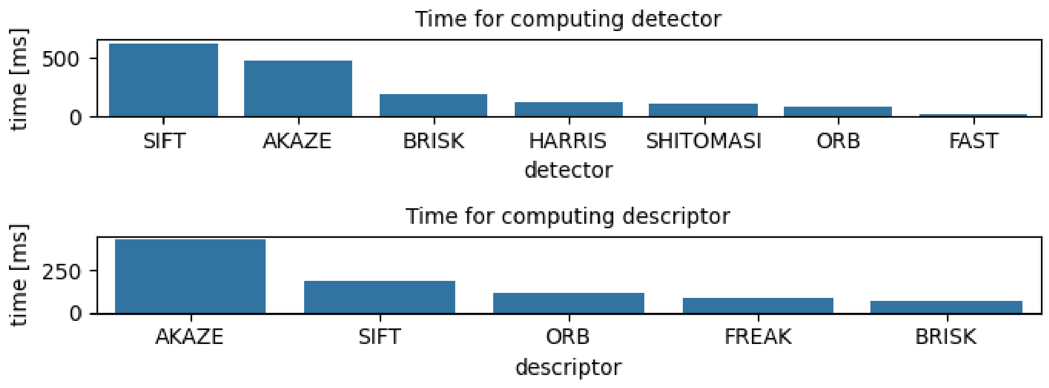
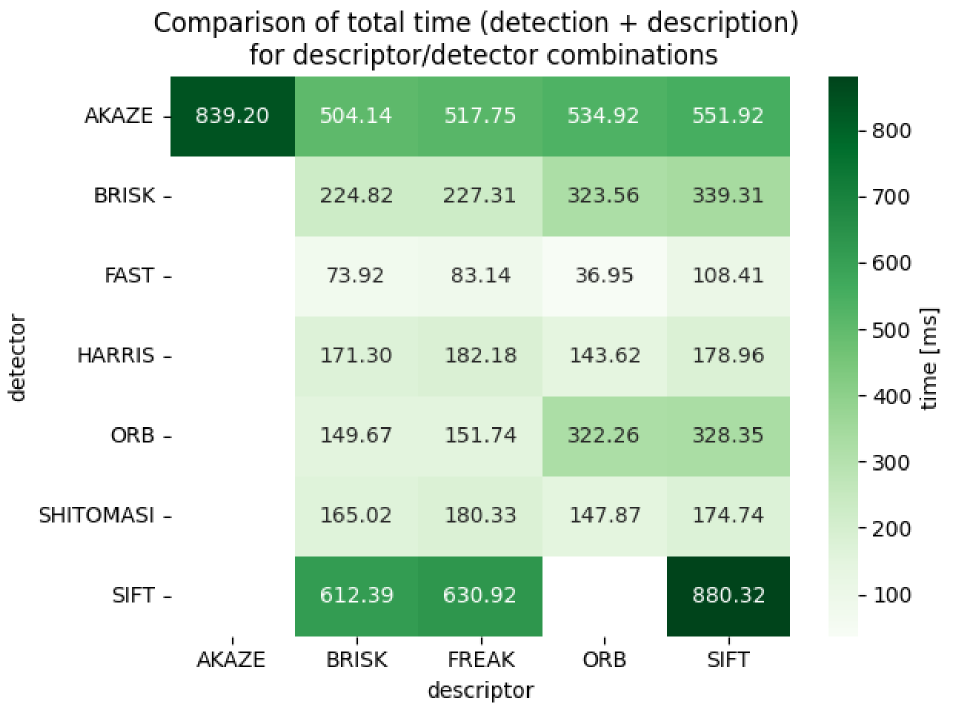


Figure 7: Keypoint count and size distribution - plotted for first image



detector \ descriptor	AKAZE	BRISK	FREAK	ORB	SIFT
AKAZE	839.20	504.14	517.75	534.92	551.92
BRISK		224.82	227.31	323.56	339.31
FAST		73.92	83.14	36.95	108.41
HARRIS		171.30	182.18	143.62	178.96
ORB		149.67	151.74	322.26	328.35
SHITOMASI		165.02	180.33	147.87	174.74
SIFT		612.39	630.92		880.32

Figure 8: Keypoint count and size distribution - plotted for first image

2.3.1 Detector/Descriptor recommendation

As the matching should be performed in real-time, the following three combinations are suggested as they are inherently fast:

- FAST / BRISK
- FAST / BRIEF
- FAST / ORB

As the FAST / ORB combination finds the most matches, it may be more robust - so this method may be the best choice for the problem.