# Section 1
## TASK MP.7

Your seventh task is to 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.

Note: In my conclusions I will pay more attention to the detectors we will likely to use for TTC system due to their low execution time.

1. SHI-THOMASI detector

| Frame# | Time (ms) | All keypoins count | Preceding vehicle kypoins count |
|---|---|---|---|
| 1 | 16.5894 | 1370 | 125 |
| 2 | 11.8486 | 1301 | 118 |
| 3 | 12.3126 | 1361 | 123 |
| 4 | 12.4399 | 1358 | 120 |
| 5 | 12.2408 | 1333 | 120 |
| 6 | 12.5787 | 1284 | 113 |
| 7 | 12.3434 | 1322 | 114 |
| 8 | 12.0824 | 1366 | 123 |
| 9 | 12.2524 | 1389 | 111 |
| 10 | 12.1379 | 1339 | 112 |
| Average | 12.68261 | 1342 | 117 |

Example:



Conclusion:
Average time of the detection is about 13ms. Shi-Tomasi detector requires gradient calculation and calculation of eigen values of the covariance matrix of gradients. It can be computationally expensive. It's not the slowest algorithm, but there are faster solution. Additionally we applied non-max suppression, min-distance filtering, overlap threshold, limited max amount of keypoints

based on min distance and image size. The Shi-Tomasi score seems to work better then Harris, actually Shi-Tomasi is an improvement of Harris detector which is based on different score calculation approach. The keypoints are evenly distributed, not very close to each other, and not forming dense clusters.

Another drawback is that the algorithm does not determine keypoint scale and orientation. The user can define the size before the detection procedure. This information is useful during matching. Without it such a keypoint will not be stable across different scaling and transformation changes.

2. Harris detector

| Frame# | Time (ms) | All keypoins count | Preceding vehicle kypoins count |
|---|---|---|---|
| 1 | 13.6489 | 871 | 85 |
| 2 | 11.9468 | 874 | 85 |
| 3 | 12.0642 | 876 | 88 |
| 4 | 11.4609 | 859 | 87 |
| 5 | 11.8507 | 860 | 92 |
| 6 | 11.9592 | 869 | 85 |
| 7 | 11.9755 | 847 | 84 |
| 8 | 12.3689 | 871 | 90 |
| 9 | 12.9332 | 864 | 85 |
| 10 | 11.7966 | 852 | 86 |
| Average | 12.20049 | 864 | 86 |

Example:



Conclusion:
Average time of the detection is about 12ms. Harris detector is similar to Shi-Thomasi. It's simple gradient based detector which different score function. The points from Shi-Thomasi are more evenly distributed, however with all the same filtering (as in previous case) the two distributions are similar. The drawbacks are the same.

3. FAST detector

| Frame# | Time (ms) | All keypoins count | Preceding vehicle kypoins count |
|---|---|---|---|
| 1 | 0.672111 | 1359 | 119 |
| 2 | 0.697225 | 1337 | 115 |
| 3 | 0.568667 | 1324 | 123 |
| 4 | 0.56966 | 1331 | 120 |
| 5 | 0.561422 | 1291 | 118 |
| 6 | 0.916513 | 1309 | 124 |
| 7 | 0.564081 | 1307 | 110 |
| 8 | 0.551529 | 1242 | 114 |
| 9 | 0.590027 | 1270 | 116 |
| 10 | 0.547701 | 1248 | 105 |
| Average | 0.6238936 | 1301 | 116 |

Example:



Conclusion:
As the name suggests, the algorithm is very fast. Its idea is based on intensity difference calculation and does not require computationally expensive gradients. However the quality of the detection is lower. We have dense clusters of points very close to each other. The situation was even worse with the default intensity threshold and ti has been increased to 40.
Also the algorithm does not calculate the scale of the point and its orientation. If some additional processing is applied this algorithm however could be a good candidate for our tasks because of its speed. However its keypoints does not have scale and orientation information which is very important in changing environment for robust matching.

## 3. BRISK detector

| Frame# | Time (ms) | All keypoins count | Preceding vehicle kypoins count |
|---:|---:|---:|---:|
| 1 | 15.2608 | 909 | 113 |
| 2 | 13.8141 | 914 | 106 |
| 3 | 13.8292 | 900 | 104 |
| 4 | 13.3665 | 871 | 118 |
| 5 | 13.1378 | 846 | 112 |
| 6 | 13.0431 | 844 | 113 |
| 7 | 13.1166 | 845 | 111 |
| 8 | 13.3271 | 843 | 114 |
| 9 | 13.1254 | 849 | 116 |
| 10 | 13.1342 | 844 | 102 |
| **Average** | 13.51548 | 866 | 110 |

Example:



Conclusion:
Brisk is more advanced detector, than the rest that was tested before. It uses FAST method for keypoint detection (AGAST, which is an evolution of FAST to be more specitic), however significantly enhances it. Brisk determines keypoint scale and orientation. When this information is used for descriptor extraction and matching it will be more robust and invulnerable to scaling or rotations. However we pay for it with computation time.  I've picked up the intensity threshold (80) to reduce the amount of keypoints which with default value (30) was huge. This reduced computation time as well. The distribution of keypoints is somewhat similar to FAST, because it's a base algorithm for BRISK. However keypoint scale and orientation are also displayed.

## 4. ORB detector

| Frame# | Time (ms) | All keypoins count | Preceding vehicle kypoins count |
|---|---|---|---|
| 1 | 39.5305 | 500 | 92 |
| 2 | 5.85159 | 500 | 102 |
| 3 | 6.16629 | 500 | 106 |
| 4 | 5.93644 | 500 | 113 |
| 5 | 5.97321 | 500 | 109 |
| 6 | 5.98172 | 500 | 125 |
| 7 | 6.01532 | 500 | 130 |
| 8 | 6.25197 | 500 | 129 |
| 9 | 6.04368 | 500 | 127 |
| 10 | 6.2978 | 500 | 128 |
| **Average** | 9.404852 | 500 | 116 |

Example:



Conclusion:
ORB is an attempt to utilize the benefits of FAST point detection algorithm, enhancing it with keypoint scale and orientation. As a result we've obtained a powerful tool. The scaling approach is somewhat similar to BRISK (using pyramid), however not the same. Orientation is determined using patch intensity centroid, where the centre of the patch is the analyzed corner. The orientation vector will be a vector between corner point and centroid. This algorithm appears to be faster than BRISK in general. I would assume the orientation computation is a bit faster and scaling procedure is not as complicated. The keypoints distribution is similar to BRISK, however scaling is different.

Note: The first launch is always slow (8 times slower than intermediate). I've made up to 10 attempts, the result is the same. I would try to study this behavior. It may be related to allocations or caching.

5. AKAZE detector

| Frame# | Time (ms) | All keypoins count | Preceding vehicle kypoins count |
|---|---|---|---|
| 1 | 78.1461 | 727 | 99 |
| 2 | 73.5137 | 709 | 91 |
| 3 | 76.2677 | 701 | 89 |
| 4 | 71.8619 | 704 | 95 |
| 5 | 74.7398 | 704 | 99 |
| 6 | 74.0709 | 711 | 99 |
| 7 | 97.4276 | 703 | 106 |
| 8 | 72.7411 | 698 | 109 |
| 9 | 71.421 | 719 | 109 |
| 10 | 74.621 | 721 | 108 |
| Average | 76.48108 | 709 | 100 |

Example:



Conclusion:

AKAZE feature detector computes scale and orientation aware keypoints. They are more evenly distributed then BRISK or ORB features, however this detector is very slow for our purposes.

## 5. SIFT detector

| Frame# | Time (ms) | All keypoins count | Preceding vehicle kypoins count |
|---|---|---|---|
| 1 | 122.142 | 1437 | 138 |
| 2 | 82.1473 | 1371 | 132 |
| 3 | 85.7365 | 1381 | 124 |
| 4 | 82.1355 | 1336 | 138 |
| 5 | 81.487 | 1303 | 134 |
| 6 | 79.3214 | 1370 | 140 |
| 7 | 80.3353 | 1396 | 137 |
| 8 | 83.3394 | 1382 | 148 |
| 9 | 80.3281 | 1462 | 159 |
| 10 | 92.0622 | 1422 | 137 |
| **Average** | 86.90347 | 1386 | 138 |

Example:



Conclusion:
At last, SIFT, very powerful and robust feature detector. The keypoints have scale and orientation. This detector with SIFT descriptors shows great results in keypoint matching. However it has lacks which make it unusable for our purposes. The biggest issue is that the detector is very slow, the descriptors are float vectors which require additional computational effort to be compared. Another problem with it - it is patented and non free to use.

# Section 2

## TASK MP.8

Your eighth task is to count the number of matched keypoints for all 10 images using all possible combinations of detectors and descriptors. In the matching step, use the BF approach with the descriptor distance ratio set to 0.8.

## SHI-THOMASI detector with:

### BRISK Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 1.3995 | 0.495631 | 84 |
| 2 | 1.43996 | 0.450922 | 80 |
| 3 | 1.41702 | 0.454796 | 73 |
| 4 | 1.45875 | 0.439354 | 77 |
| 5 | 1.34103 | 0.425872 | 74 |
| 6 | 1.44934 | 0.39396 | 70 |
| 7 | 1.44527 | 0.432599 | 79 |
| 8 | 1.3243 | 0.428055 | 81 |
| 9 | 1.35191 | 0.393705 | 72 |
| Avg | 1.403 | 0.43498 | 76 |

### BRIEF Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 0.822473 | 0.678849 | 86 |
| 2 | 0.907647 | 0.254 | 84 |
| 3 | 0.775653 | 0.256295 | 87 |
| 4 | 0.778864 | 0.255176 | 91 |
| 5 | 0.83527 | 0.240797 | 87 |
| 6 | 0.771025 | 0.231273 | 76 |
| 7 | 0.783436 | 0.250322 | 81 |
| 8 | 0.855377 | 0.257174 | 88 |
| 9 | 0.810406 | 0.228699 | 88 |
| Avg | 0.81557 | 0.29473 | 85 |

### ORB Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 0.942569 | 0.638678 | 86 |
| 2 | 0.888609 | 0.266432 | 84 |
| 3 | 0.79436 | 0.284952 | 87 |
| 4 | 0.795427 | 0.253781 | 91 |
| 5 | 0.773743 | 0.254729 | 87 |
| 6 | 0.774466 | 0.232764 | 76 |
| 7 | 0.782438 | 0.24185 | 81 |
| 8 | 0.772053 | 0.243964 | 88 |
| 9 | 0.774565 | 0.22357 | 88 |
| Avg | 0.81091 | 0.29341 | 85 |

### FREAK Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 18.4876 | 0.50438 | 66 |
| 2 | 18.2235 | 0.429512 | 66 |
| 3 | 18.196 | 0.419714 | 64 |
| 4 | 18.2064 | 0.424179 | 63 |
| 5 | 18.5277 | 0.397152 | 62 |
| 6 | 19.0086 | 0.599585 | 64 |
| 7 | 18.7857 | 0.40024 | 61 |
| 8 | 18.2393 | 0.397156 | 6 |
| 9 | 18.4876 | 0.50438 | 66 |
| Avg | 18.46248 | 0.45292 | 57 |

| SIFT Descriptor | | | |
|---|---|---|---|
| **Frame** | **Extraction time (ms)** | **Match time (ms)** | **Matches** |
| **1** | 13.3789 | 0.360684 | 112 |
| **2** | 16.3917 | 0.290317 | 109 |
| **3** | 13.4515 | 0.288712 | 104 |
| **4** | 14.1808 | 0.289567 | 103 |
| **5** | 13.9186 | 0.297497 | 99 |
| **6** | 13.5072 | 0.252646 | 101 |
| **7** | 15.84 | 0.246969 | 96 |
| **8** | 13.6622 | 0.264566 | 106 |
| **9** | 14.2794 | 0.248864 | 97 |
| **Avg** | 14.29003 | 0.2822 | 103 |

Conclusion: BRIEF and ORB descriptors give the same amount of matches for SHI-THOMASI keypoints. ORB actually utilizes BRIEF approach (ORB = oriented FAST and rotated BRIEF). Brisk gives less matches and its descriptors are a little bit more computationally expensive. However we should take into account that orientation is not computed for these keypoints. The situation may change when orientation comes into play. FREAK descriptors are computationally inefficient, same about SIFT descriptors, however SIFT descriptors produced the biggest amount of matches.

Please, note that AKAZE descriptors can only be used with KAZE or AKAZE keypoints.

## Harris detector with:

| BRISK Descriptor | | | |
|---|---|---|---|
| **Frame** | **Extraction time (ms)** | **Match time (ms)** | **Matches** |
| **1** | 1.14585 | 0.296056 | 58 |
| **2** | 1.22186 | 0.253916 | 61 |
| **3** | 1.26057 | 0.258074 | 62 |
| **4** | 1.19123 | 0.252953 | 60 |
| **5** | 1.13888 | 0.262244 | 58 |
| **6** | 1.14298 | 0.249395 | 55 |
| **7** | 1.19413 | 0.357426 | 63 |
| **8** | 1.14545 | 0.25962 | 61 |
| **9** | 1.14941 | 0.291525 | 58 |
| **Avg** | 1.1767 | 0.27568 | 59 |

| BRIEF Descriptor | | | |
|---|---|---|---|
| **Frame** | **Extraction time (ms)** | **Match time (ms)** | **Matches** |
| **1** | 0.82888 | 0.564577 | 64 |
| **2** | 0.829484 | 0.16315 | 64 |
| **3** | 0.82159 | 0.161558 | 64 |
| **4** | 0.824847 | 0.163992 | 64 |
| **5** | 0.816893 | 0.166395 | 66 |
| **6** | 0.810877 | 0.153997 | 63 |
| **7** | 0.826108 | 0.170134 | 63 |
| **8** | 0.89514 | 0.216646 | 70 |
| **9** | 0.828839 | 0.201872 | 66 |
| **Avg** | 0.8314 | 0.21803 | 64 |

**ORB Descriptor**

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 0.843488 | 0.493393 | 64 |
| 2 | 0.840085 | 0.17767 | 64 |
| 3 | 0.828317 | 0.160051 | 64 |
| 4 | 0.802683 | 0.164684 | 64 |
| 5 | 0.840036 | 0.166228 | 66 |
| 6 | 0.824467 | 0.152541 | 63 |
| 7 | 0.794435 | 0.161409 | 63 |
| 8 | 0.865121 | 0.31056 | 70 |
| 9 | 0.796613 | 0.155812 | 66 |
| Avg | 0.82613 | 0.21581 | 64 |

**FREAK Descriptor**

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 18.2569 | 0.588948 | 51 |
| 2 | 20.3794 | 0.258158 | 51 |
| 3 | 21.6442 | 0.250672 | 47 |
| 4 | 21.2897 | 0.260134 | 50 |
| 5 | 22.0734 | 0.252713 | 49 |
| 6 | 21.8314 | 0.23866 | 54 |
| 7 | 21.5219 | 0.254784 | 50 |
| 8 | 22.4491 | 0.247644 | 49 |
| 9 | 21.6894 | 0.409285 | 51 |
| Avg | 21.23726 | 0.30677 | 50 |

**SIFT Descriptor**

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 12.4226 | 0.536377 | 73 |
| 2 | 17.3167 | 0.169612 | 76 |
| 3 | 16.4722 | 0.164625 | 79 |
| 4 | 15.5119 | 0.208908 | 82 |
| 5 | 16.2099 | 0.406611 | 78 |
| 6 | 17.2293 | 0.171333 | 70 |
| 7 | 18.331 | 0.226429 | 78 |
| 8 | 15.5226 | 0.19185 | 85 |
| 9 | 17.305 | 0.18891 | 78 |
| Avg | 16.25791 | 0.25162 | 77 |

Conclusion: The conclusion here is pretty similar to the previous case.

Please, note that AKAZE descriptors can only be used with KAZE or AKAZE keypoints.

# FAST detector with:

## BRISK Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 1.22416 | 0.449877 | 62 |
| 2 | 1.19857 | 0.426557 | 75 |
| 3 | 1.21096 | 0.437784 | 71 |
| 4 | 1.37181 | 0.406523 | 81 |
| 5 | 1.41592 | 0.745102 | 57 |
| 6 | 1.11627 | 0.469355 | 72 |
| 7 | 1.12119 | 0.390727 | 69 |
| 8 | 1.13706 | 0.405476 | 65 |
| 9 | 1.06861 | 0.378696 | 77 |
| Avg | 1.20717 | 0.45667 | 69 |

## BRIEF Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 0.999692 | 0.292313 | 81 |
| 2 | 1.98731 | 0.49875 | 80 |
| 3 | 0.763779 | 0.254343 | 74 |
| 4 | 1.00057 | 0.296317 | 75 |
| 5 | 0.776361 | 0.265419 | 74 |
| 6 | 0.991065 | 0.316405 | 83 |
| 7 | 0.844482 | 0.230191 | 73 |
| 8 | 0.877409 | 0.238937 | 74 |
| 9 | 0.851624 | 0.227572 | 84 |
| Avg | 1.01025 | 0.29113 | 77 |

## ORB Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 1.06837 | 0.295886 | 81 |
| 2 | 1.53265 | 0.605361 | 80 |
| 3 | 0.765054 | 0.256637 | 74 |
| 4 | 1.82802 | 0.430182 | 75 |
| 5 | 1.96695 | 0.31563 | 74 |
| 6 | 1.56657 | 0.749751 | 83 |
| 7 | 0.874705 | 0.315567 | 73 |
| 8 | 1.45975 | 0.222076 | 74 |
| 9 | 0.84428 | 0.233783 | 84 |
| Avg | 1.32292 | 0.38054 | 77 |

## FREAK Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 19.7145 | 0.555228 | 52 |
| 2 | 20.9629 | 0.422692 | 62 |
| 3 | 19.9655 | 0.442672 | 56 |
| 4 | 21.0313 | 0.428963 | 63 |
| 5 | 21.9088 | 0.807594 | 44 |
| 6 | 21.2435 | 0.569772 | 63 |
| 7 | 20.2681 | 0.447837 | 63 |
| 8 | 21.0886 | 0.417234 | 56 |
| 9 | 22.2529 | 0.460513 | 72 |
| Avg | 20.93734 | 0.50583 | 59 |

## SIFT Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 15.2005 | 0.695439 | 93 |
| 2 | 19.4523 | 0.294326 | 96 |
| 3 | 18.1123 | 0.678115 | 97 |
| 4 | 16.6678 | 0.86506 | 94 |
| 5 | 22.3421 | 0.336876 | 92 |
| 6 | 19.4454 | 0.415064 | 100 |
| 7 | 19.2524 | 0.226793 | 88 |
| 8 | 16.7792 | 0.302111 | 89 |
| 9 | 19.3644 | 0.270036 | 91 |
| Avg | 18.51293 | 0.45375 | 93 |

Conclusion: BRIEF and ORB descriptors give the same amount of matches as for SHI-THOMASI and Harris keypoints. Looking at the pictures it seems like FAST + ORB (or BRIEF) with 0.8 distance ration filtering gives quite good results in matching.

Please, note that AKAZE descriptors can only be used with KAZE or AKAZE keypoints.

## BRISK detector with:

### BRISK Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 1.15423 | 0.438866 | 60 |
| 2 | 1.11048 | 0.364695 | 62 |
| 3 | 1.2095 | 0.384248 | 62 |
| 4 | 1.82392 | 0.405264 | 60 |
| 5 | 1.17242 | 0.414243 | 52 |
| 6 | 1.19799 | 0.373926 | 61 |
| 7 | 1.33176 | 0.483171 | 56 |
| 8 | 1.20769 | 0.384771 | 58 |
| 9 | 1.10638 | 0.375018 | 78 |
| Avg | 1.25715 | 0.40268 | 61 |

### ORB Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 3.15854 | 0.262849 | 45 |
| 2 | 3.15202 | 0.208113 | 50 |
| 3 | 3.12707 | 0.213737 | 37 |
| 4 | 3.12546 | 0.226049 | 41 |
| 5 | 3.74564 | 0.228813 | 33 |
| 6 | 3.18419 | 0.239591 | 51 |
| 7 | 3.14786 | 0.220785 | 51 |
| 8 | 3.4004 | 0.268004 | 50 |
| 9 | 3.13359 | 0.210822 | 59 |
| Avg | 3.24164 | 0.23097 | 46 |

| FREAK Descriptor | | | |
|---|---|---|---|
| Frame | Extraction time (ms) | Match time (ms) | Matches |
| 1 | 17.9613 | 0.400256 | 57 |
| 2 | 18.8108 | 0.390046 | 58 |
| 3 | 18.3408 | 0.327228 | 45 |
| 4 | 18.1373 | 0.349936 | 54 |
| 5 | 17.8986 | 0.338668 | 42 |
| 6 | 18.0149 | 0.340212 | 59 |
| 7 | 17.9712 | 0.352958 | 55 |
| 8 | 17.9118 | 0.356454 | 56 |
| 9 | 17.9222 | 0.346861 | 66 |
| Avg | 18.10765 | 0.35584 | 54 |

| SIFT Descriptor | | | |
|---|---|---|---|
| Frame | Extraction time (ms) | Match time (ms) | Matches |
| 1 | 30.0896 | 0.670491 | 80 |
| 2 | 29.2712 | 0.236263 | 78 |
| 3 | 30.1829 | 0.249489 | 71 |
| 4 | 30.5608 | 0.256139 | 81 |
| 5 | 35.0109 | 0.275971 | 72 |
| 6 | 29.3608 | 0.252147 | 80 |
| 7 | 29.5236 | 0.26082 | 75 |
| 8 | 33.433 | 0.271449 | 78 |
| 9 | 30.4119 | 0.255584 | 85 |
| Avg | 30.87163 | 0.30315 | 77 |

Conclusion: BRISK + BRISK descriptor is the best here as a combination of speed and amount of matches. BRIEF descriptor is not orientation aware thus is not compatible with brisk.
ORB descriptors does not seem to work well with BRISK keypoints. The computation is slower, and we've got less amount of matches with BRISK + ORB descriptor then with BRISK + BRISK. When orientation came into play the ORB descriptor is slower to be computed.
SIFT descriptors are good with BRISK keypoints in terms of matching but very slow.
AKAZE descriptors can only be used with KAZE or AKAZE keypoints.

## ORB detector with:

| BRISK Descriptor | | | |
|---|---|---|---|
| Frame | Extraction time (ms) | Match time (ms) | Matches |
| 1 | 1.0564 | 0.304643 | 60 |
| 2 | 1.07901 | 0.303775 | 65 |
| 3 | 1.12991 | 0.331082 | 65 |
| 4 | 1.17595 | 0.316355 | 76 |
| 5 | 1.32375 | 0.345562 | 72 |
| 6 | 1.28437 | 0.416485 | 83 |
| 7 | 1.27265 | 0.426219 | 83 |
| 8 | 1.26953 | 0.422161 | 73 |
| 9 | 1.29051 | 0.411263 | 72 |
| Avg | 1.20912 | 0.36417 | 72 |

| ORB Descriptor | | | |
|---|---|---|---|
| Frame | Extraction time (ms) | Match time (ms) | Matches |
| 1 | 4.89016 | 0.232596 | 40 |
| 2 | 3.4916 | 0.205822 | 57 |
| 3 | 3.63736 | 0.217563 | 49 |
| 4 | 3.48543 | 0.230157 | 54 |
| 5 | 3.60833 | 0.290192 | 57 |
| 6 | 3.57737 | 0.313244 | 68 |
| 7 | 4.01727 | 0.293056 | 71 |
| 8 | 3.51376 | 0.264625 | 62 |
| 9 | 3.51889 | 0.33535 | 72 |
| Avg | 3.7489 | 0.26473 | 58 |

| FREAK Descriptor | | | |
| --- | --- | --- | --- |
| Frame | Extraction time (ms) | Match time (ms) | Matches |
| 1 | 17.6314 | 0.26724 | 39 |
| 2 | 20.0352 | 0.137741 | 33 |
| 3 | 25.5372 | 0.158653 | 37 |
| 4 | 21.3649 | 0.147849 | 40 |
| 5 | 21.7512 | 0.149282 | 33 |
| 6 | 20.4011 | 0.205909 | 40 |
| 7 | 22.9012 | 0.474524 | 41 |
| 8 | 23.8922 | 0.215808 | 39 |
| 9 | 22.6054 | 0.213164 | 44 |
| Avg | 21.79108 | 0.2189 | 38 |

| SIFT Descriptor | | | |
| --- | --- | --- | --- |
| Frame | Extraction time (ms) | Match time (ms) | Matches |
| 1 | 42.3606 | 1.60751 | 67 |
| 2 | 40.296 | 0.234957 | 79 |
| 3 | 41.7295 | 0.242783 | 78 |
| 4 | 38.9984 | 0.25473 | 79 |
| 5 | 40.076 | 0.250395 | 82 |
| 6 | 45.2688 | 0.295889 | 95 |
| 7 | 46.3496 | 0.309662 | 95 |
| 8 | 48.8021 | 0.332539 | 94 |
| 9 | 48.9395 | 0.346414 | 94 |
| Avg | 43.64672 | 0.43054 | 84 |

Conclusion: ORB + BRISK give descent performance here.
ORB descriptors does not seem to work well with BRISK keypoints. The computation is slower with ORB+ORB descriptor combination, and we've got less amount of matches.
SIFT descriptors are the slowest. Freak descriptors give a small amount of matches being slow also.
AKAZE descriptors can only be used with KAZE or AKAZE keypoints.

## AKAZE detector with:

| BRISK Descriptor | | | |
| --- | --- | --- | --- |
| Frame | Extraction time (ms) | Match time (ms) | Matches |
| 1 | 2.90015 | 0.341518 | 72 |
| 2 | 1.21945 | 0.279872 | 65 |
| 3 | 1.21848 | 0.292849 | 69 |
| 4 | 1.26664 | 0.31727 | 72 |
| 5 | 1.30306 | 0.933945 | 72 |
| 6 | 1.99557 | 10.6136 | 76 |
| 7 | 1.34255 | 0.348019 | 82 |
| 8 | 1.37805 | 0.356038 | 91 |
| 9 | 1.37111 | 0.553429 | 84 |
| Avg | 1.555 | 1.55961 | 75 |

| ORB Descriptor | | | |
| --- | --- | --- | --- |
| Frame | Extraction time (ms) | Match time (ms) | Matches |
| 1 | 2.81437 | 0.220654 | 54 |
| 2 | 2.45546 | 0.16908 | 56 |
| 3 | 2.49843 | 0.165298 | 52 |
| 4 | 2.44241 | 0.178076 | 49 |
| 5 | 2.48019 | 0.184253 | 57 |
| 6 | 2.4594 | 0.19462 7 | 4 |
| 7 | 2.54841 | 0.239719 | 61 |
| 8 | 2.52103 | 0.215841 | 70 |
| 9 | 2.60904 | 0.212162 | 73 |
| Avg | 2.53652 | 0.19813 | 52 |

**FREAK Descriptor**

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 21.509 | 0.325566 | 60 |
| 2 | 22.1012 | 0.263871 | 58 |
| 3 | 21.4934 | 0.2688 | 57 |
| 4 | 22.5462 | 0.30124 | 60 |
| 5 | 22.623 | 0.417779 | 58 |
| 6 | 21.9348 | 0.414061 | 73 |
| 7 | 22.9433 | 0.371404 | 79 |
| 8 | 21.0892 | 0.34273 | 81 |
| 9 | 20.9169 | 0.356697 | 72 |
| Avg | 21.90633 | 0.34023 | 66 |

**AKAZE Descriptor**

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 66.9835 | 0.60445 | 69 |
| 2 | 78.7245 | 0.376527 | 72 |
| 3 | 67.8543 | 0.262253 | 73 |
| 4 | 70.4368 | 0.281864 | 70 |
| 5 | 71.6553 | 0.286226 | 69 |
| 6 | 63.4273 | 0.31238 | 78 |
| 7 | 75.9721 | 0.363309 | 83 |
| 8 | 63.6144 | 0.342516 | 85 |
| 9 | 63.9981 | 0.355171 | 88 |
| Avg | 69.18514 | 0.35385 | 76 |

**SIFT Descriptor**

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 63.0897 | 0.352599 | 69 |
| 2 | 61.4415 | 0.262174 | 72 |
| 3 | 63.1075 | 0.264974 | 73 |
| 4 | 63.0312 | 0.276637 | 70 |
| 5 | 63.3006 | 0.306531 | 69 |
| 6 | 61.6456 | 0.299512 | 78 |
| 7 | 62.7544 | 0.362887 | 83 |
| 8 | 66.8431 | 0.340827 | 85 |
| 9 | 61.9417 | 0.356329 | 88 |
| Avg | 63.01725 | 0.3136 | 76 |

Conclusion: With AKAZE BRISK descriptors give a good performance in terms of time and matches. The best are AKAZE and SIFT descriptors but showing 69ms and 63ms computation time respectively against 1.5ms for BRISK.

# SIFT detector with:

## BRISK Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 2.24131 | 0.914301 | 57 |
| 2 | 2.02642 | 0.4866 | 63 |
| 3 | 2.16835 | 0.496524 | 59 |
| 4 | 2.08761 | 0.529234 | 62 |
| 5 | 2.11574 | 0.526059 | 55 |
| 6 | 2.10859 | 0.525064 | 52 |
| 7 | 2.20409 | 0.570455 | 54 |
| 8 | 2.27848 | 0.633671 | 63 |
| 9 | 2.09828 | 0.593252 | 73 |
| Avg | 2.14765 | 0.58612 | 59 |

## FREAK Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 21.9511 | 0.572422 | 59 |
| 2 | 21.6366 | 0.480617 | 63 |
| 3 | 22.3572 | 0.656924 | 55 |
| 4 | 23.9334 | 0.508987 | 65 |
| 5 | 22.181 | 0.518629 | 51 |
| 6 | 22.2819 | 0.54401 | 50 |
| 7 | 22.7762 | 0.540729 | 47 |
| 8 | 23.2096 | 0.624821 | 53 |
| 9 | 22.1589 | 0.588292 | 65 |
| Avg | 22.49843 | 0.55949 | 56 |

## SIFT Descriptor

| Frame | Extraction time (ms) | Match time (ms) | Matches |
|---|---|---|---|
| 1 | 71.0978 | 1.69729 | 82 |
| 2 | 73.0649 | 0.345079 | 81 |
| 3 | 79.1031 | 0.348835 | 86 |
| 4 | 94.7405 | 0.349361 | 94 |
| 5 | 71.2129 | 0.340893 | 90 |
| 6 | 71.6165 | 0.315829 | 81 |
| 7 | 70.5774 | 0.367391 | 82 |
| 8 | 73.0489 | 0.439851 | 102 |
| 9 | 72.0573 | 0.361594 | 104 |
| Avg | 75.16881 | 0.50734 | 89 |

Conclusion: SIFT + SIFT is a beast. Slow but powerful. ORB descriptor is not compatible with SIFT, other performed worse in terms of the amount of matches.

# Section 3
### TASK MP.9
Your ninth task is to log the time it takes for keypoint detection and descriptor extraction. The results must be entered into a spreadsheet and based on this information you will then suggest the TOP3 detector / descriptor combinations as the best choice for our purpose of detecting keypoints on vehicles. Finally, in a short text, please justify your recommendation based on your observations and on the data you collected.

Conclusion:
During tasked 7 and 8 I've logged time for both, keypoint detection and matching. The information can be found in the tables above.
Let's choose the best 3 combinations now, basing on this information.
1. The fastest combination is **FAST detector + BRIEF descriptor - 1.92 ms** for keypoint detection, descriptor generation and matching with 0.8 distance ration filtering. This is amazingly fast combination and seems suitable for realtime tasks. However keypoints and descriptors (in this combination) are not scale and orientation aware. This means that such system can be vulnerable to rotations or scale changes. It requires further quality analysis to make more robust conclusions about such combination performance in terms of keypoints detection/matching quality.
2. The second candidate would be **ORB + BRISK - 10.98 ms** for keypoint detection, descriptor generation and matching with 0.8 distance ration filtering. Taking into account that such combination is scale and orientation aware 11ms is not bad. ORB detector seems to be faster than BRISK detector. This is (IMHO) because it uses more simple scale detection procedure. However brisk descriptor seems to be faster to be calculated and gives more matches.
3. It's a hard decision however I would consider **SHI-TOMASI + BRIEF - 13.5 ms** as a third candidate. SHI-TOMASI gives evenly distributed keypoints, in some approximation they can be matched reliably if the scale change is not huge, for example if we take frames fast enough. On the test frame sequence this combination seems to work. However it's not enough tests to make a conclusion if this combination can be reliable enough for TTC system.

Of course it would be great to use SIFT or AKAZE combination, however up to 100ms (almost 50 times slower than the first candidate) seems too slow.