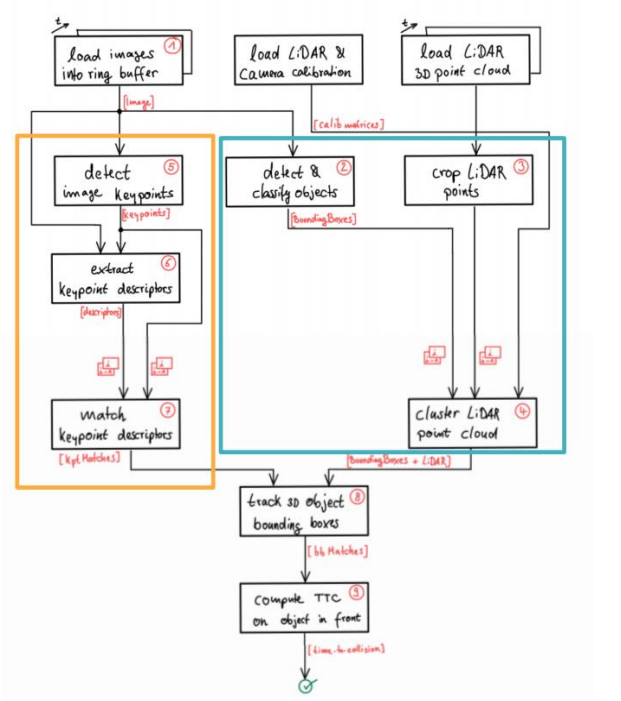
SFND 3D Object Tracking

Welcome to the final project of the camera course. By completing all the lessons, you now have a solid understanding of keypoint detectors, descriptors, and methods to match them between successive images. Also, you know how to detect objects in an image using the YOLO deep-learning framework. And finally, you know how to associate regions in a camera image with Lidar points in 3D space. Let's take a look at our program schematic to see what we already have accomplished and what's still missing.

TTC Building Blocks

Course Structure

- **Lesson 3** : Keypoint detection and matching
- **Mid-Term Project** : Develop the matching framework and test several state-of-the-art algorithms.
- **Lesson 4** : Lidar point processing and deep learning for object detection.
- **Final Project** : Track 3D bounding boxes and compute refined TTC



In this final project, you will implement the missing parts in the schematic. To do this, you will complete four major tasks:

1. First, you will develop a way to match 3D objects over time by using keypoint correspondences.
2. Second, you will compute the TTC based on Lidar measurements.
3. You will then proceed to do the same using the camera, which requires to first associate keypoint matches to regions of interest and then to compute the TTC based on those matches.
4. And lastly, you will conduct various tests with the framework. Your goal is to identify the most suitable detector/descriptor combination for TTC estimation and also to search for problems that can lead to faulty measurements by the camera or Lidar sensor. In the last course of this Nanodegree, you will learn about the Kalman filter, which is a great way to combine the two independent TTC measurements into an improved version which is much more reliable than a single sensor alone can be. But before we think about such things, let us focus on your final project in the camera course.

Dependencies for Running Locally

- cmake >= 2.8
  - All OSes: click here for installation instructions
- make >= 4.1 (Linux, Mac), 3.81 (Windows)
  - Linux: make is installed by default on most Linux distros
  - Mac: install Xcode command line tools to get make
  - Windows: Click here for installation instructions
- Git LFS
  - Weight files are handled using LFS
- OpenCV >= 4.1
  - This must be compiled from source using the `-D OPENCV_ENABLE_NONFREE=ON` cmake flag for testing the SIFT and SURF detectors.
  - The OpenCV 4.1.0 source code can be found here
- gcc/g++ >= 5.4
  - Linux: gcc / g++ is installed by default on most Linux distros
  - Mac: same deal as make - install Xcode command line tools
  - Windows: recommend using MinGW

Basic Build Instructions

1. Clone this repo.
2. Make a build directory in the top level project directory: `mkdir build && cd build`
3. Compile: `cmake .. && make`
4. Run it: `./3D_object_tracking`.

Solution - SFND 3D Object Tracking

Part I: Solution Description

FP.1 Match 3D Objects

Task: Implement the method "matchBoundingBoxes", which takes as input both the previous and the current data frames and provides as output the ids of the matched regions of interest (i.e. the boxID property). Matches must be the ones with the highest number of keypoint correspondences.

Implementation:

```
void matchBoundingBoxes(std::vector<cv::DMatch> &matches, std::map<int, int> &bbBestMatches,
DataFrame &prevFrame, DataFrame &currFrame)
{
	for (auto it1 = prevFrame.boundingBoxes.begin(); it1 != prevFrame.boundingBoxes.end(); it1++)
	{
		std::vector<vector<cv::DMatch>::iterator> match;

		for (auto it2 = matches.begin(); it2 != matches.end(); it2++)
		{
			if (it1->roi.contains(prevFrame.keypoints.at(it2->queryIdx).pt))
			{
				match.push_back(it2);
			}
		}

		std::multimap<int, int> bestMatches;

		for (auto it3 = match.begin(); it3 != match.end(); it3++)
		{
			for (auto it4 = currFrame.boundingBoxes.begin(); it4 !=
currFrame.boundingBoxes.end(); it4++)
			{
				if (it4->roi.contains(currFrame.keypoints.at((*it3)->trainIdx).pt))
				{
					bestMatches.insert(std::pair<int, int>(it4->boxID, (*it3)-
>trainIdx));
				}
			}
		}

		int idx = std::numeric_limits<int>::max();
		int bestMatchesCount = 0;

		if (bestMatches.size() > 0)
		{
			for (auto it5 = bestMatches.begin(); it5 != bestMatches.end(); it5++)
			{
				if (bestMatches.count(it5->first) > bestMatchesCount)
				{
					bestMatchesCount = bestMatches.count(it5->first);
					idx = it5->first;
				}
			}

			bbBestMatches.insert(std::pair<int, int>(it1->boxID, idx));
		}
	}
}
```

FP.2 Compute Lidar-based TTC

Task: Implement the method "matchBoundingBoxes", which takes as input both the previous and the current data frames and provides as output the ids of the matched regions of interest (i.e. the boxID property). Matches must be the ones with the highest number of keypoint correspondences.

Implementation:

```
void computeTTCLidar(std::vector<LidarPoint> &lidarPointsPrev,
                     std::vector<LidarPoint> &lidarPointsCurr, double frameRate, double &TTC)
{
	double y = 0.63; // = (laneWidth - 0.2) / 2; double laneWidth = 1.46;

	auto compareY = [&y](const LidarPoint &lidarPoint) {return abs(lidarPoint.y) >= y; };

	lidarPointsPrev.erase(std::remove_if(lidarPointsPrev.begin(), lidarPointsPrev.end(),
compareY), lidarPointsPrev.end());
	lidarPointsCurr.erase(std::remove_if(lidarPointsCurr.begin(), lidarPointsCurr.end(),
compareY), lidarPointsCurr.end());

	unsigned int maxClosestPointsNum = 150;
	auto compareX = [](const LidarPoint &lidarPoint1, const LidarPoint &lidarPoint2) { return
lidarPoint1.x > lidarPoint2.x; };

	if (lidarPointsPrev.size() < maxClosestPointsNum)
	{
		maxClosestPointsNum = lidarPointsPrev.size();
	}

	std::make_heap(lidarPointsPrev.begin(), lidarPointsPrev.end(), compareX);
	std::sort_heap(lidarPointsPrev.begin(), lidarPointsPrev.begin() + maxClosestPointsNum,
compareX);

	if (lidarPointsCurr.size() < maxClosestPointsNum)
	{
		maxClosestPointsNum = lidarPointsCurr.size();
	}

	std::make_heap(lidarPointsCurr.begin(), lidarPointsCurr.end(), compareX);
	std::sort_heap(lidarPointsCurr.begin(), lidarPointsCurr.begin() + maxClosestPointsNum,
compareX);

	auto sumX = [](const double sum, const LidarPoint &lidarPoint) { return sum + lidarPoint.x;
};

	double xMeanPrev = std::accumulate(lidarPointsPrev.begin(), lidarPointsPrev.begin() +
maxClosestPointsNum, 0.0, sumX) / maxClosestPointsNum;
	double xMeanCurr = std::accumulate(lidarPointsCurr.begin(), lidarPointsCurr.begin() +
maxClosestPointsNum, 0.0, sumX) / maxClosestPointsNum;

	TTC = xMeanCurr * 1.0 / frameRate / (xMeanPrev - xMeanCurr);
}
```

FP.3 Associate Keypoint Correspondences with Bounding Boxes

Task: Prepare the TTC computation based on camera measurements by associating keypoint correspondences to the bounding boxes which enclose them. All matches which satisfy this condition must be added to a vector in the respective bounding box.

Implementation:

```
void clusterKptMatchesWithROI(BoundingBox &boundingBox, std::vector<cv::KeyPoint> &kptsPrev,
std::vector<cv::KeyPoint> &kptsCurr, std::vector<cv::DMatch> &kptMatches)
{
	std::vector<double> distance;

	for (auto it = kptMatches.begin(); it != kptMatches.end(); it++)
	{
		const auto &kptCurr = kptsCurr[it->trainIdx];

		if (boundingBox.roi.contains(kptCurr.pt))
		{
			const auto &kptPrev = kptsPrev[it->queryIdx];

			distance.push_back(cv::norm(kptCurr.pt - kptPrev.pt));
		}
	}

	int distanceNum = distance.size();
	double distanceMean = std::accumulate(distance.begin(), distance.end(), 0.0) / distanceNum;

	for (auto it = kptMatches.begin(); it != kptMatches.end(); it++)
	{
		const auto &kptCurr = kptsCurr[it->trainIdx];

		if (boundingBox.roi.contains(kptCurr.pt))
		{
			int kptPrevIdx = it->queryIdx;
			const auto &kptPrev = kptsPrev[kptPrevIdx];

			if (cv::norm(kptCurr.pt - kptPrev.pt) < distanceMean * 1.3)
			{
				boundingBox.keypoints.push_back(kptCurr);
				boundingBox.kptMatches.push_back(*it);
			}
		}
	}
}
```

FP.4 Compute Camera-based TTC

Task: Compute the time-to-collision in second for all matched 3D objects using only keypoint correspondences from the matched bounding boxes between current and previous frame.

Implementation:

```
void computeTTCCamera(std::vector<cv::KeyPoint> &kptsPrev, std::vector<cv::KeyPoint> &kptsCurr,
                      std::vector<cv::DMatch> kptMatches, double frameRate, double &TTC, cv::Mat
*visImg)
{
	vector<double> distanceRatios;

	for (auto it1 = kptMatches.begin(); it1 != kptMatches.end() - 1; ++it1)
	{
		for (auto it2 = kptMatches.begin() + 1; it2 != kptMatches.end(); ++it2)
		{
			double distanceCurr = cv::norm(kptsCurr.at(it1->trainIdx).pt - kptsCurr.at(it2-
>trainIdx).pt);
			double distancePrev = cv::norm(kptsPrev.at(it1->queryIdx).pt - kptsPrev.at(it2-
>queryIdx).pt);

			if (distancePrev > std::numeric_limits<double>::epsilon() && distanceCurr >=
100.0)
			{
				double distanceRatio = distanceCurr / distancePrev;
				distanceRatios.push_back(distanceRatio);
			}
		}
	}

	if (distanceRatios.size() == 0)
	{
		TTC = NAN;
	}
	else
	{
		std::sort(distanceRatios.begin(), distanceRatios.end());

		long medianIdx = floor(distanceRatios.size() / 2.0);
		double medianDistanceRatio = distanceRatios.size() % 2 == 0 ?
(distanceRatios[medianIdx - 1] + distanceRatios[medianIdx]) / 2.0 : distanceRatios[medianIdx];
		TTC = -1.0 / frameRate / (1.0 - medianDistanceRatio);
	}
}
```

Part II: Performance Evaluation

FP.5 Performance Evaluation 1

Task: Find examples where the TTC estimate of the Lidar sensor does not seem plausible. Describe your observations and provide a sound argumentation why you think this happened.

Observation: Focused on the below shown successive two picture frames there can be recognized, that the vehicle in front of the ego is braking and the ego is moving closer to the car in front, but the TTC Lidar increases

Argumentation: The TCC Lidar is not correct, because of false positives of included Lidar observations. These false positives could be caused of proceeding observations of e.g. the front vehicles front mirror.

A possibility to decrease of the influence of including of such kind of outliers is decreasing of the maximum number of the closest Lidar points. There can also be calculated the median instead of the mean.

FP.6 Performance Evaluation 2

Task: Run several detector / descriptor combinations and look at the differences in TTC estimation. Find out which methods perform best and also include several examples where camera-based TTC estimation is way off. As with Lidar, describe your observations again and also look into potential reasons.

Several detector / descriptor combinations and look at the differences in TTC estimation:

Detector Type	Descriptor Type	Image Number	TTC Lidar	TTC Camera
SHITOMASI	BRISK	1	12,4168	14,1119
SHITOMASI	BRISK	2	15,2882	14,7273
SHITOMASI	BRISK	3	23,9784	13,4904
SHITOMASI	BRISK	4	15,54	12,398
SHITOMASI	BRISK	5	11,4256	12,6754
SHITOMASI	BRISK	6	12,0039	-inf
SHITOMASI	BRISK	7	20,0045	-inf
SHITOMASI	BRISK	8	20,2339	-inf
SHITOMASI	BRISK	9	13,2775	11,5126
SHITOMASI	BRISK	10	17,0552	-inf
SHITOMASI	BRISK	11	11,5379	11,3024
SHITOMASI	BRISK	12	10,9867	11,6524
SHITOMASI	BRISK	13	9,77931	11,7243
SHITOMASI	BRISK	14	11,214	11,8781
SHITOMASI	BRISK	15	8,52662	9,33785
SHITOMASI	BRISK	16	10,2624	11,3079
SHITOMASI	BRISK	17	13,9977	11,4487
SHITOMASI	BRISK	18	8,89551	9,12792
SHITOMASI	BRIEF	1	12,4168	13,8948
SHITOMASI	BRIEF	2	15,2882	13,1487
SHITOMASI	BRIEF	3	23,9784	22,6781
SHITOMASI	BRIEF	4	15,54	13,81
SHITOMASI	BRIEF	5	11,4256	12,1886
SHITOMASI	BRIEF	6	12,0039	15,1561
SHITOMASI	BRIEF	7	20,0045	18,3368
SHITOMASI	BRIEF	8	20,2339	12,3793
SHITOMASI	BRIEF	9	13,2775	12,463
SHITOMASI	BRIEF	10	17,0552	13,3448
SHITOMASI	BRIEF	11	11,5379	12,2
SHITOMASI	BRIEF	12	10,9867	12,2294
SHITOMASI	BRIEF	13	9,77931	12,3292
SHITOMASI	BRIEF	14	11,214	12,7102
SHITOMASI	BRIEF	15	8,52662	12,9788
SHITOMASI	BRIEF	16	10,2624	11,3856
SHITOMASI	BRIEF	17	13,9977	13,6063
SHITOMASI	BRIEF	18	8,89551	7,75039
SHITOMASI	ORB	1	12,4168	13,8801
SHITOMASI	ORB	2	15,2882	17,7333
SHITOMASI	ORB	3	23,9784	11,8924

SHITOMASI	ORB	4	15,54	12,1319
SHITOMASI	ORB	5	11,4256	13,0809
SHITOMASI	ORB	6	12,0039	14,5777
SHITOMASI	ORB	7	20,0045	13,8434
SHITOMASI	ORB	8	20,2339	12,2022
SHITOMASI	ORB	9	13,2775	11,2653
SHITOMASI	ORB	10	17,0552	13,4962
SHITOMASI	ORB	11	11,5379	11,3246
SHITOMASI	ORB	12	10,9867	12,7698
SHITOMASI	ORB	13	9,77931	12,1119
SHITOMASI	ORB	14	11,214	11,5594
SHITOMASI	ORB	15	8,52662	10,2592
SHITOMASI	ORB	16	10,2624	12,0686
SHITOMASI	ORB	17	13,9977	13,5261
SHITOMASI	ORB	18	8,89551	11,8629
SHITOMASI	FREAK	1	12,4168	13,7249
SHITOMASI	FREAK	2	15,2882	13,0927
SHITOMASI	FREAK	3	23,9784	11,4126
SHITOMASI	FREAK	4	15,54	12,5354
SHITOMASI	FREAK	5	11,4256	12,3705
SHITOMASI	FREAK	6	12,0039	14,2385
SHITOMASI	FREAK	7	20,0045	12,5645
SHITOMASI	FREAK	8	20,2339	12,8515
SHITOMASI	FREAK	9	13,2775	12,0414
SHITOMASI	FREAK	10	17,0552	13,0084
SHITOMASI	FREAK	11	11,5379	11,9619
SHITOMASI	FREAK	12	10,9867	11,8227
SHITOMASI	FREAK	13	9,77931	12,3436
SHITOMASI	FREAK	14	11,214	12,6612
SHITOMASI	FREAK	15	8,52662	10,297
SHITOMASI	FREAK	16	10,2624	11,3153
SHITOMASI	FREAK	17	13,9977	12,963
SHITOMASI	FREAK	18	8,89551	11,2811
FAST	BRISK	1	12,4168	12,3
FAST	BRISK	2	15,2882	12,3453
FAST	BRISK	3	23,9784	16,6163
FAST	BRISK	4	15,54	12,8857
FAST	BRISK	5	11,4256	-inf
FAST	BRISK	6	12,0039	13,0386
FAST	BRISK	7	20,0045	12,041
FAST	BRISK	8	20,2339	11,4066
FAST	BRISK	9	13,2775	11,8684
FAST	BRISK	10	17,0552	13,3473
FAST	BRISK	11	11,5379	12,9492
FAST	BRISK	12	10,9867	12,1174
FAST	BRISK	13	9,77931	12,7784
FAST	BRISK	14	11,214	11,6077
FAST	BRISK	15	8,52662	11,4079
FAST	BRISK	16	10,2624	12,2566

FAST	BRISK	17	13,9977	9,2933
FAST	BRISK	18	8,89551	11,8606
FAST	BRIEF	1	12,4168	11,1776
FAST	BRIEF	2	15,2882	13,0069
FAST	BRIEF	3	23,9784	14,8206
FAST	BRIEF	4	15,54	13,6686
FAST	BRIEF	5	11,4256	-inf
FAST	BRIEF	6	12,0039	41,7823
FAST	BRIEF	7	20,0045	12,7583
FAST	BRIEF	8	20,2339	12,7664
FAST	BRIEF	9	13,2775	13,9231
FAST	BRIEF	10	17,0552	16,2141
FAST	BRIEF	11	11,5379	13,6355
FAST	BRIEF	12	10,9867	12,9831
FAST	BRIEF	13	9,77931	13,1487
FAST	BRIEF	14	11,214	11,7034
FAST	BRIEF	15	8,52662	12,6071
FAST	BRIEF	16	10,2624	13,0069
FAST	BRIEF	17	13,9977	11,2586
FAST	BRIEF	18	8,89551	13,7908
FAST	ORB	1	12,4168	12,2019
FAST	ORB	2	15,2882	12,9348
FAST	ORB	3	23,9784	16,6262
FAST	ORB	4	15,54	14,0688
FAST	ORB	5	11,4256	-inf
FAST	ORB	6	12,0039	55,9788
FAST	ORB	7	20,0045	12,3866
FAST	ORB	8	20,2339	12,186
FAST	ORB	9	13,2775	12,8711
FAST	ORB	10	17,0552	16,2321
FAST	ORB	11	11,5379	14,1395
FAST	ORB	12	10,9867	12,9831
FAST	ORB	13	9,77931	13,5629
FAST	ORB	14	11,214	11,2961
FAST	ORB	15	8,52662	10,7133
FAST	ORB	16	10,2624	11,9229
FAST	ORB	17	13,9977	11,951
FAST	ORB	18	8,89551	13,7843
FAST	FREAK	1	12,4168	11,9
FAST	FREAK	2	15,2882	16,1387
FAST	FREAK	3	23,9784	13,2117
FAST	FREAK	4	15,54	14,0316
FAST	FREAK	5	11,4256	99,7643
FAST	FREAK	6	12,0039	12,3292
FAST	FREAK	7	20,0045	12,753
FAST	FREAK	8	20,2339	11,5467
FAST	FREAK	9	13,2775	-inf
FAST	FREAK	10	17,0552	13,4169
FAST	FREAK	11	11,5379	13,0192

FAST	FREAK	12	10,9867	12,0364
FAST	FREAK	13	9,77931	11,6599
FAST	FREAK	14	11,214	11,3487
FAST	FREAK	15	8,52662	11,0296
FAST	FREAK	16	10,2624	12,1019
FAST	FREAK	17	13,9977	11,0677
FAST	FREAK	18	8,89551	11,8606
BRISK	BRISK	1	12,4168	13,031
BRISK	BRISK	2	15,2882	23,1027
BRISK	BRISK	3	23,9784	17,5172
BRISK	BRISK	4	15,54	15,2031
BRISK	BRISK	5	11,4256	28,2877
BRISK	BRISK	6	12,0039	17,6743
BRISK	BRISK	7	20,0045	16,8563
BRISK	BRISK	8	20,2339	23,0521
BRISK	BRISK	9	13,2775	15,1936
BRISK	BRISK	10	17,0552	13,4121
BRISK	BRISK	11	11,5379	12,4948
BRISK	BRISK	12	10,9867	11,0107
BRISK	BRISK	13	9,77931	11,8505
BRISK	BRISK	14	11,214	12,0376
BRISK	BRISK	15	8,52662	16,061
BRISK	BRISK	16	10,2624	11,2969
BRISK	BRISK	17	13,9977	9,70123
BRISK	BRISK	18	8,89551	11,6176
BRISK	BRIEF	1	12,4168	14,863
BRISK	BRIEF	2	15,2882	23,7824
BRISK	BRIEF	3	23,9784	18,0446
BRISK	BRIEF	4	15,54	22,7832
BRISK	BRIEF	5	11,4256	25,5059
BRISK	BRIEF	6	12,0039	56,735
BRISK	BRIEF	7	20,0045	25,2647
BRISK	BRIEF	8	20,2339	18,4064
BRISK	BRIEF	9	13,2775	20,3103
BRISK	BRIEF	10	17,0552	18,4689
BRISK	BRIEF	11	11,5379	21,829
BRISK	BRIEF	12	10,9867	17,7779
BRISK	BRIEF	13	9,77931	17,4963
BRISK	BRIEF	14	11,214	14,8559
BRISK	BRIEF	15	8,52662	11,054
BRISK	BRIEF	16	10,2624	16,1143
BRISK	BRIEF	17	13,9977	15,0108
BRISK	BRIEF	18	8,89551	17,6601
BRISK	ORB	1	12,4168	27,0734
BRISK	ORB	2	15,2882	18,4096
BRISK	ORB	3	23,9784	18,2556
BRISK	ORB	4	15,54	17,8829
BRISK	ORB	5	11,4256	19,1438
BRISK	ORB	6	12,0039	19,4256

BRISK	ORB	7	20,0045	17,9004
BRISK	ORB	8	20,2339	17,5031
BRISK	ORB	9	13,2775	14,9704
BRISK	ORB	10	17,0552	12,4419
BRISK	ORB	11	11,5379	13,2661
BRISK	ORB	12	10,9867	15,3836
BRISK	ORB	13	9,77931	11,6599
BRISK	ORB	14	11,214	12,2939
BRISK	ORB	15	8,52662	12,0918
BRISK	ORB	16	10,2624	11,4475
BRISK	ORB	17	13,9977	13,6668
BRISK	ORB	18	8,89551	17,9372
BRISK	FREAK	1	12,4168	12,3601
BRISK	FREAK	2	15,2882	21,4704
BRISK	FREAK	3	23,9784	13,5807
BRISK	FREAK	4	15,54	13,8293
BRISK	FREAK	5	11,4256	22,615
BRISK	FREAK	6	12,0039	16,3304
BRISK	FREAK	7	20,0045	15,4463
BRISK	FREAK	8	20,2339	21,665
BRISK	FREAK	9	13,2775	19,2377
BRISK	FREAK	10	17,0552	13,9062
BRISK	FREAK	11	11,5379	16,2953
BRISK	FREAK	12	10,9867	12,7469
BRISK	FREAK	13	9,77931	16,11
BRISK	FREAK	14	11,214	12,1517
BRISK	FREAK	15	8,52662	18,2318
BRISK	FREAK	16	10,2624	10,3798
BRISK	FREAK	17	13,9977	9,24577
BRISK	FREAK	18	8,89551	11,4493
ORB	BRISK	1	12,4168	37,4084
ORB	BRISK	2	15,2882	-inf
ORB	BRISK	3	23,9784	35,1392
ORB	BRISK	4	15,54	21,7788
ORB	BRISK	5	11,4256	-inf
ORB	BRISK	6	12,0039	11,1075
ORB	BRISK	7	20,0045	-inf
ORB	BRISK	8	20,2339	114,017
ORB	BRISK	9	13,2775	-inf
ORB	BRISK	10	17,0552	-inf
ORB	BRISK	11	11,5379	8,34928
ORB	BRISK	12	10,9867	-inf
ORB	BRISK	13	9,77931	7,53169
ORB	BRISK	14	11,214	24,6173
ORB	BRISK	15	8,52662	13,594
ORB	BRISK	16	10,2624	-inf
ORB	BRISK	17	13,9977	23,6587
ORB	BRISK	18	8,89551	29,6605
ORB	BRIEF	1	12,4168	27,5288

ORB	BRIEF	2	15,2882	-inf
ORB	BRIEF	3	23,9784	113,496
ORB	BRIEF	4	15,54	16,0494
ORB	BRIEF	5	11,4256	30,2058
ORB	BRIEF	6	12,0039	-48,152
ORB	BRIEF	7	20,0045	-inf
ORB	BRIEF	8	20,2339	34,6298
ORB	BRIEF	9	13,2775	-inf
ORB	BRIEF	10	17,0552	168,345
ORB	BRIEF	11	11,5379	26,3378
ORB	BRIEF	12	10,9867	28,6548
ORB	BRIEF	13	9,77931	-inf
ORB	BRIEF	14	11,214	13,5006
ORB	BRIEF	15	8,52662	-inf
ORB	BRIEF	16	10,2624	13,0005
ORB	BRIEF	17	13,9977	20,3758
ORB	BRIEF	18	8,89551	25,8155
ORB	ORB	1	12,4168	-inf
ORB	ORB	2	15,2882	10,1531
ORB	ORB	3	23,9784	33,8655
ORB	ORB	4	15,54	-inf
ORB	ORB	5	11,4256	-inf
ORB	ORB	6	12,0039	18,4549
ORB	ORB	7	20,0045	-inf
ORB	ORB	8	20,2339	-inf
ORB	ORB	9	13,2775	-inf
ORB	ORB	10	17,0552	-inf
ORB	ORB	11	11,5379	8,28796
ORB	ORB	12	10,9867	-inf
ORB	ORB	13	9,77931	-inf
ORB	ORB	14	11,214	-inf
ORB	ORB	15	8,52662	-inf
ORB	ORB	16	10,2624	-inf
ORB	ORB	17	13,9977	16,3257
ORB	ORB	18	8,89551	-inf
ORB	FREAK	1	12,4168	12,0569
ORB	FREAK	2	15,2882	-inf
ORB	FREAK	3	23,9784	11,1077
ORB	FREAK	4	15,54	12,605
ORB	FREAK	5	11,4256	231,042
ORB	FREAK	6	12,0039	12,8194
ORB	FREAK	7	20,0045	-inf
ORB	FREAK	8	20,2339	11,9978
ORB	FREAK	9	13,2775	20,7736
ORB	FREAK	10	17,0552	-inf
ORB	FREAK	11	11,5379	8,34928
ORB	FREAK	12	10,9867	25,9864
ORB	FREAK	13	9,77931	9,59835
ORB	FREAK	14	11,214	63,442

ORB	FREAK	15	8,52662	-inf
ORB	FREAK	16	10,2624	10,62
ORB	FREAK	17	13,9977	11,3219
ORB	FREAK	18	8,89551	8,73235
AKAZE	BRISK	1	12,4168	16,6557
AKAZE	BRISK	2	15,2882	20,3352
AKAZE	BRISK	3	23,9784	13,3659
AKAZE	BRISK	4	15,54	17,5261
AKAZE	BRISK	5	11,4256	18,0806
AKAZE	BRISK	6	12,0039	14,7084
AKAZE	BRISK	7	20,0045	15,8833
AKAZE	BRISK	8	20,2339	13,9229
AKAZE	BRISK	9	13,2775	13,8224
AKAZE	BRISK	10	17,0552	15,7516
AKAZE	BRISK	11	11,5379	12,3626
AKAZE	BRISK	12	10,9867	15,532
AKAZE	BRISK	13	9,77931	10,0855
AKAZE	BRISK	14	11,214	14,1883
AKAZE	BRISK	15	8,52662	17,349
AKAZE	BRISK	16	10,2624	13,8622
AKAZE	BRISK	17	13,9977	11,5526
AKAZE	BRISK	18	8,89551	9,035
AKAZE	BRIEF	1	12,4168	19,0025
AKAZE	BRIEF	2	15,2882	15,0455
AKAZE	BRIEF	3	23,9784	16,3543
AKAZE	BRIEF	4	15,54	13,8482
AKAZE	BRIEF	5	11,4256	18,0872
AKAZE	BRIEF	6	12,0039	19,7851
AKAZE	BRIEF	7	20,0045	21,0141
AKAZE	BRIEF	8	20,2339	14,121
AKAZE	BRIEF	9	13,2775	21,3561
AKAZE	BRIEF	10	17,0552	11,5709
AKAZE	BRIEF	11	11,5379	15,7328
AKAZE	BRIEF	12	10,9867	17,6087
AKAZE	BRIEF	13	9,77931	10,1939
AKAZE	BRIEF	14	11,214	14,1695
AKAZE	BRIEF	15	8,52662	15,078
AKAZE	BRIEF	16	10,2624	11,8399
AKAZE	BRIEF	17	13,9977	11,0314
AKAZE	BRIEF	18	8,89551	10,4701
AKAZE	ORB	1	12,4168	15,9469
AKAZE	ORB	2	15,2882	19,9608
AKAZE	ORB	3	23,9784	13,1423
AKAZE	ORB	4	15,54	16,7571
AKAZE	ORB	5	11,4256	15,5183
AKAZE	ORB	6	12,0039	20,1502
AKAZE	ORB	7	20,0045	20,1355
AKAZE	ORB	8	20,2339	20,762
AKAZE	ORB	9	13,2775	13,3063

AKAZE	ORB	10	17,0552	11,5619
AKAZE	ORB	11	11,5379	16,4508
AKAZE	ORB	12	10,9867	19,249
AKAZE	ORB	13	9,77931	13,6232
AKAZE	ORB	14	11,214	12,2356
AKAZE	ORB	15	8,52662	10,3847
AKAZE	ORB	16	10,2624	12,3128
AKAZE	ORB	17	13,9977	11,1439
AKAZE	ORB	18	8,89551	11,8054
AKAZE	FREAK	1	12,4168	13,0099
AKAZE	FREAK	2	15,2882	15,7044
AKAZE	FREAK	3	23,9784	16,0624
AKAZE	FREAK	4	15,54	14,036
AKAZE	FREAK	5	11,4256	15,8923
AKAZE	FREAK	6	12,0039	14,4455
AKAZE	FREAK	7	20,0045	15,3629
AKAZE	FREAK	8	20,2339	19,8126
AKAZE	FREAK	9	13,2775	21,9214
AKAZE	FREAK	10	17,0552	11,8935
AKAZE	FREAK	11	11,5379	12,1617
AKAZE	FREAK	12	10,9867	16,9222
AKAZE	FREAK	13	9,77931	10,7833
AKAZE	FREAK	14	11,214	9,87627
AKAZE	FREAK	15	8,52662	9,89497
AKAZE	FREAK	16	10,2624	13,4375
AKAZE	FREAK	17	13,9977	11,0423
AKAZE	FREAK	18	8,89551	8,80695

The method SHITOMASI/BRIEF performs best, because the difference of TTC Lidar and TTC Camera, considered over all pictures, is the smallest. The method AKAZE/FREAK is also quite good according to that. SHITOMASI/ORB would be my 3rd choice.

Some detector/descriptor combinations produces very unreliable TTC Camera. The worst results delivers combinations which uses ORB detectors.