Faraaz Beyabani

Camera Based 2D Feature Tracking Final Report

FP.1 Match 3D Objects

In my implementation of the matchBoundingBoxes function, I iterate over all of the provided matches and assign the previous and current keypoints to a bounding box in the previous and current dataframes, respectively. I do this by using the ROI method where I check if the keypoint is within the bounding box's specified region of interest using the OpenCV built-in Rect::contains() function. While doing so, I also keep track of the number of keypoints that are associated with that pair of bounding boxes.

After iterating over all of the matches provided, I then check all pairs of bounding boxes, and pair up the ones who share the greatest number of points. I believe my method is the most optimized for this task, as iterating through the bounding box keypoint lists may be O(n) time, as opposed to simply checking ROI which is O(1) time, since we need not iterate through a long list.

FP.2 Compute Lidar-based TTC

I compute the TTC of Lidar much like as was explained to me by the instructor in a previous lesson. I first create a vector of all of the previous and current lidar point X values, as that is the only coordinate which is valuable for computing TTC. I then sort each vector using the C++ standard sort method. I then use the Inter-quartile Range (IQR) to calculate the quartiles of values for the x positions and find a lower bound for valid x-values. Any x-values lower than the obtained value are most likely outliers.

Using this newfound bound, I proceed normally, calculating the minimum x-value for both the previous and current lidar points, considering that valid x-values must be above the threshold calculated via IQR. I then follow the formula laid out in previous lessons for calculating the TTC.

FP.3 Associate Keypoint Correspondences with Bounding Boxes

The clustering method for keypoint matches was fairly straightforward to implement. To begin with, I initialize a data structure to keep track of all of the matches within the given bounding box and their calculated distance for use in filtering later on. I then iterate through every match and check to see if the bounding box's ROI contains the current keypoint of the match. If it does, then I calculate the distance between the previous keypoint's position and the current keypoint's position, which manifests as a change in position on the image (for instance, if the preceding vehicle gets closer due to braking).

While doing so, I also update a variable used to take the mean of the Euclidean distances later. Finally, I iterate through every match in my intermediate data structure and only add it to the bounding box's matched keypoints list if it does not exceed 2.5 times the mean distance. Through some testing, I concluded that this was the best value to use in order to avoid outliers (I found that in some cases, the mean distance was ~16 while the maximum distance was ~315).

FP.4 Compute Camera-based TTC

Again, the code for the compute Camera TTC function was based on my own code for and the instructor solutions for an activity does prior in the course. In this function, I just iterate through the list of keypoint matches passed in and compare every keypoint to every other keypoint and take the distance. I only accept those distances who are non-zero (determined by using the minimum epsilon value) and less than an upper bound of 100.0.

I then sort the distance ratios and find the median of the calculated ratios. The median is used here because it is resistant to outliers given a fairly large data set. With the calculated median distance ratio, I follow the formula given to calculate the TTC.

FP.5 Performance Evaluation 1

There are a few instances where the computed lidar TTC fluctuates somewhat wildly, and I do believe these results are not due to my implementations of the functions. For instance: On frame 7, the computed TTC jumps from about ~12.3 to 34.34. Likewise, on frame 5, the TTC is 7.4, having halved from its previous value of ~14.2, with not much movement in the camera image.

I believe this is the result of converting the 3D lidar points viewed from a top-down angle into points on a camera image viewed from a different axis. In order to estimate the true distance of the preceding vehicle, I viewed the lidar points from a top down angle and looked at how the points moved in each measurement. I saw that the relative positions of the ego vehicle and preceding vehicle were unchanged, which did not warrant the large change in TTC computation.

Another explanation could be that the vehicles in the image were coming to a halt. If so, the computed TTC would get larger and larger since the change in position would be minimal.

FP.6 Performance Evaluation 2

Detector	Descriptor	TTC 1 (s)	TTC 2 (s)	TTC 3 (s)	TTC 4 (s)	TTC 5 (s)	TTC 6 (s)
Shi-Tomasi	BRISK	14.38	13.41	12.82	13.55	15.15	12.4
	BRIEF	12.5	10.02	26.1	12.18	13.99	12.5
	ORB	12.92	13.31	13.32	12.3	-inf	10.86
	FREAK	14.11	14.28	11.07	12.8	12.59	14.02
	SIFT	14.07	14.47	11.93	12.26	13.22	12.99
Harris	BRISK	10.9	10.58	-90.85	11.76	-inf	13.55
	BRIEF	9.74	10.19	-10.85	44.91	14.65	28.61
	ORB	8.75	10.58	-161.7	13.64	-inf	17.62
	FREAK	9.74	NaN	-18.26	12.12	13.64	12.33
	SIFT	10.9	52.79	-80.85	11.57	13.27	15.89
FAST	BRISK	18.93	25.97	11.15	50.6	12.47	11.3
	BRIEF	12.43	-10.13	11.06	12.14	69.69	-84.63
	ORB	10.46	11.87	36.49	11.2065	-inf	-inf
	FREAK	12.16	-inf	12.56	12.51	12.62	10.73
	SIFT	11.64	12.71	16.31	14.4	28.11	12.61
BRISK	BRISK	16.2	19.05	12.29	13.68	22.03	18.79

	BRIEF	20.39	7.15	31.52	15.93	37.03	-84.34
	ORB	16.17	17.23	19.61	17.89	17.32	18.46
	FREAK	15.87	30.04	13.54	13.67	41.24	14.82
	SIFT	16.16	16.53	12.54	29.68	13.32	15.55
ORB	BRISK	18.77	38.78	9.11	-inf	67.15	19.71
	BRIEF	15.49	-inf	63.49	15.94	21.12	13.89
	ORB	13.87	12.57	11.91	26.9	10.92	36.85
	FREAK	-inf	-inf	10.48	9.16	-inf	-inf
	SIFT	14.67	13.2	13.42	0	84.53	96.87
SIFT	BRISK	11.21	11.82	14.97	22.53	15.7	13.19
	BRIEF	12.04	8.23	20.51	18.09	21.56	18.38
	FREAK	11.85	15.26	14.25	31.09	12.76	12.44
	SIFT	11.61	12.51	12.81	18.09	12.58	11.22
AKAZE	BRISK	12.73	19.59	15.49	17.64	15.08	17.88
	BRIEF	12.43	26.9	15.49	18.72	18.51	24.42
	ORB	13.35	13.47	18.13	19.45	11.82	15.51
	FREAK	12.05	15.65	14.09	17.13	14.58	16
	SIFT	14.44	13.38	16.31	14.71	15.52	13.83
	AKAZE	12.03	13.78	13.55	16.48	12.46	13.61

The above data table contains the TTC computations for up to 6 frames for each of the detectors & descriptors previously implemented. Of the values above, I have highlighted in blue particularly stable and/or accurate TTC computations, and orange those combinations which resulted in wildly inconsistent TTC computations.

In particular, I wanted to draw attention to the FREAK and AKAZE detectors. In the mid-term project, I reported that FREAK was the best detector to use for our purposes in regard to its extremely fast speed and acceptable accuracy. However, when used for TTC computations, I noticed that it was woefully inaccurate in its calculations, resulting in large negative results.

The AKAZE detector, on the hand, has surprisingly stable and accurate results for many of the descriptors. In particular, I find the SIFT and AKAZE descriptors to have very stable and believable results given the respective images. Furthermore, I did not find the runtimes of these AKAZE detector combinations to be particularly long. Thus, I would like to revise my answer to the midterm project: I now believe that the AKAZE detector is much more useful for our purpose because of its reliability as compared to other detectors, like ORB or HARRIS.

I believe the above inaccuracies are not a result of my implementations for this project, but rather stem from shortcomings of detectors, descriptors, and the difficulties of calculating TTC for a 3D object using only a 2D camera image. These results would be much more different if we were somehow able to combine the lidar and keypoint positions, using lidar points for 3D distance and depth, and using keypoints for 2D distance on the left and right of the ego car.

Below, I have picked some detector and descriptor combinations which I found interesting, either due to their accuracy or lack thereof. The AKAZE and SIFT combinations

follow roughly the same computations, while the FAST + FREAK and ORB + FREAK combinations are missing data due to my having to replace the negative infinities (as otherwise the graph would be biased downward for TTC computation).