MP.1

I made a class called RingBuffer which holds 2 elements and overrides old ones when new ones come in. I implemented an iterator to make It a drop in replacement for the vector.

MP.2

I implemented all detectors by calling the appropriate openCV functions

MP.3

I implemented a function called FilterKeypoints In Matching2D.cpp which takes In a cv::Rect and a vector of KeyPoints that discards all the points in the vector that are outside the Rect. The used the predefined variable vehicleRect as the input for it.

MP.4

I implemented all descriptors by calling the appropriate openCV functions.

MP.5

I implemented FLANN matching by calling the appropriate openCV function.I implemented KNN by calling the appropriate openCV function.

MP.6

I implemented the ratio test by setting the KNN to only return the best 2 results. I then divided their distances and if the ratio was under 0.8 I considered It a match.

MP.7

The perf.pdf spreadsheet contains the average amount of detected key-points for each detector type. Although each detector detected a different amount of key-points, they were all found in more or less the same areas. These areas included the license plate, tail-lights, the outline of the seats in the car, and the outline of the car. AKAZE was unique In that it did not get any key-points on the license plate.

MP.8

The perf.pdf spreadsheet contains the average amount of key-point matches for each detector / descriptor combo.

MP.9

The perf.pdf spreadsheet contains average run-times for detection, description, and matching of key-points for each detector/descriptor combo.

I would recommend using the following detector/descriptor combos, FAST/ORB, HARRIS/ORB, ORB/FREAK.

If detection speed is our main goal then it narrows it down to FAST/ORB and FAST/BRIEF, but I would recommend FAST/ORB because it gets more key-point matches in roughly the same time.

If we want to maximized the amount of key-points matched, then I would recommend going with HARRIS/ORB. Harris gives by far the most key-points, and the ORB matcher yielded the most matches.

If we want to minimize the amount of key-points matched then I would recommend ORB/FREAK.

NOTE:

I could only get AKAZE descriptors to work with the AKAZE detector. Any other combination threw an error.

The SIFT/ORB combo threw an out of memory exception.