COVIS Lab 2 Report
Feature Detection

Prepared by Astha Gupta and Sthithpragya Gupta

The overview of the methods used in the first and second approaches and their respective tuning parameters is given in this report. Details about the methods and their accompanying variables and parameters is presented as comment blocks in the code. The approach of Part 1 is presented in the *"tracking_wrt_frame1.cpp"* file whereas the approach of Part 2 is in *"tracking_centroid.cpp"* file.

Part 1 - Matching the objects in the current frame to the objects in the first frame

tracking_wrt_frame1.cpp

Methods used:

void processFirstFrame(Mat& firstFrame) *(lines 29 - 48)*

This method has been defined to extract the object of interest from the first frame and use it for comparison with subsequent frames. The keypoints detected from first frame are stored in **keypoints_object** and their descriptors are stored in **descriptors_object**.

void calculateGoodMatches(std::vector< DMatch >& matches, std::vector< DMatch >& good_matches) *(lines 54 - 67)*

This method has been defined to calculate the "good matches" wrt the first frame using the calculateGoodMatches method. This is done to keep only the strong keypoint correspondences between the frames. Any potential correspondance with distance higher than 0.25 is ruled out. The good matches are stored in **good_matches**.

void calculateAndMatch(Mat& currentFrame) *(lines 76 - 133)*

This method detects and describes the keypoints of any subsequent frame and stores them in local variables **keypoints_frame** and **descriptor_frame.** Subsequently, **matcher** object of FlannBasedMatcher class is declared which matches and stores the keypoint correspondences between **descriptor_frame** (current frame) and **descriptor_object** (first frame). Following **calculateGoodMatches** method is called to filter out only the strong keypoint correspondences which are then stored in **good_matches.**

Tuned parameters:

It is recommended to have between 50 - 100 keypoints in the images for comparison. The Hessian threshold has been set to 350, following which SURF detects 190 keypoints in the first frame for comparison with the subsequent frames. Following keypoint detection and description in any subsequent frame, we calculate the "good matches" wrt the first frame using the calculateGoodMatches method. This is done to keep only the strong keypoint correspondences between the frames. Following this filtering, we are left with a healthy 50 - 150 keypoint

correspondances in the image frames for majority of the video except incase when the camera pans significantly towards the left where the number of good matches drops to 13 - 15. This happens because in such a case there aren't sufficient strong keypoints correspondances left for detection and comparison.

Part 2 - Matching the objects in the current frame to the objects in the previous frame

tracking_centroid.cpp

void makeROI(std::vector<Point2f>& scene_corners,Rect& roi_current) *(lines 28 - 94)*
This method is used to make a bounding box around the object of interest in the current frame using the coordinates of the quadrilateral obtained by homography computation from the bounding box around the object in the previous frame. The coordinates of the 4 vertices of the intermediate quadrilateral are stored in **scene_corners**. The coordinates of **scene_corners** are checked if they lie outside the image. The minimum x and y coordinates (not necessarily of the same point) from among these vertices is stored in **min_first.** Similarly, the maximum x and y coordinates are stored in **max_second**. The centroid of the quadrilateral **centroid** is calculated using **min_first** and **max_second** as their midpoint. Since the quadrilateral is always quite close to a rectangle, it has been assumed that the opposite sides are equal in length. These lengths are stored in **width** and **height**. Finally a rectangle is made using the **centroid**, **width** and **height.**

void processObjectFrame(Mat& firstFrame) *(lines 96 - 112)*
This method is used set the value of **objectFrame** by specifying the bounding box **roi** (from makeROI) around the object of interest in the frame (specified by the argument). Subsequently **keypoints_object** (detector information) and **descriptors_object** (descriptor info) are also set. The info about the bounding box vertices is stored in **object_corners**.

calculateGoodMatches and **calculateAndMatch** same as in Part 1.

Tuned parameters:
The value of the Hessian threshold is same as in Part 1.

Comparing approaches in Part 1 and Part 2:

- For the same Hessian threshold, in comparison to Part 1, we obtain greater number of strong keypoint correspondences in part 2 (13 - 150 in Part 1 vs 30 - 200 in Part 2). Furthermore, even when the camera pans to the extreme left, in part 1 the strong keypoint correspondances drop to around 15 whereas in part 2 we have around 30.
- The quadrilateral bounding the object of interest in any subsequent frame in Part 1 will not necessarily be a regular rectangle whose edges are parallel to the x and y axis of the first image frame since there can be significant changes in orientation of the subsequent image frame due to camera's motion. However in part 2, since we are comparing two subsequent frames, the bounding quadrilateral will closely resemble a rectangle with

edges almost parallel to axis of the previous image frame since there aren't drastic changes in orientation between two subsequent frames.

- Furthermore, in part 2, it is more probable that the bounding box is actually a rectangle and not a quadrilateral (say resembling a trapezoid). This due to more accurate computation of the homography and availability of adequate strong keypoint correspondences which make it easier to define the bounding box and thus track the object with greater accuracy.
- Overall, tracking an object is more accurate in Part 2.

References:

- https://docs.opencv.org/3.1.0/d5/d6f/tutorial_feature_flann_matcher.html
- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html