ECE 471: Assignment 3 (Part B): Prototyping

Geena Smith
Faculty of Engineering
University of Victoria
Victoria, Canada
gmsmith@uvic.ca
V00835915

Joshua McIntosh
Faculty of Engineering
University of Victoria
Victoria, Canada
jmcintosh@uvic.ca
V00875715

I. Approach Flow-chart

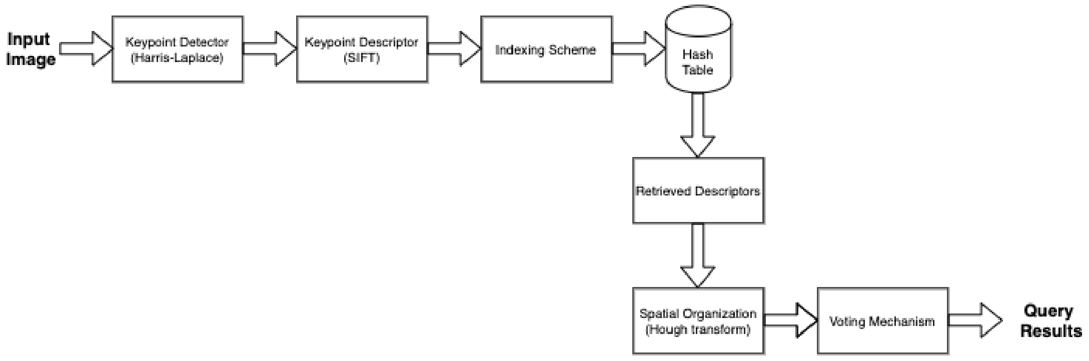
Input Image
Keypoint Detector (Harris-Laplace)
Keypoint Descriptor (SIFT)
Indexing Scheme
Hash Table
Retrieved Descriptors
Spatial Organization (Hough transform)
Voting Mechanism
Query Results

Fig. 1. Flow-chart of the proposed approach

II. Validation

Given that our proposed approach relies heavily on existing algorithms, validation of each section may be difficult. Libraries such as OpenCV have existing implementations of the Harris-Laplace, SIFT, and Hough Transform algorithms [1,2,3] and thus all become black-boxes should we decide to leverage those. This would mean that validation of those algorithms can only be done using their output, which may not be too difficult considering the plethora of examples online to which they can be compared to, but in the event that a problem does arise using one of those algorithms, it would be nearly impossible to fix. Alternatively, those algorithms can be implemented manually. Individually, each step of the Harris-Laplace and SIFT algorithms are fairly easy to validate, but that would require a drastic increase in development time which is not feasible given the current timeline. In either case, hyperparameter tuning such as the threshold values for SIFT will require testing and validation to maximize their effectiveness.

For parts of our algorithm that are unique, such as the indexing scheme, hash table, retrieval of descriptors, proximity graph, and voting mechanism, they will need to be implemented manually. This allows for a higher degree of validation since we can add any intermediate steps necessary to allow for observation and validation of each step. Validating the insertion and retrieval of descriptors from the hash table will be simple enough: insert a set of test descriptors into the hash table and test that they are retrieved correctly. The proximity graphs during the spatial organization step will require validation before performing the Hough Transform, which can be done by manually creating a test set. Since we will be using a $k = 5$ proximity graph, this will not be too intensive to create. Finally, the voting mechanism will be difficult to validate without using other parts of the algorithm. Since it is the final step and it requires retrieving descriptors and for those descriptors to be spatially organized, we will wait until validation is complete for the other steps of the algorithm before testing the voting mechanism. The output should be the final output of our algorithm, namely locations within the document, so testing using toy examples will be enough to verify the voting mechanism works before analysis of our algorithm can begin.

III. Assumptions

Our proposed approach makes many assumptions, mostly about the data set. Since we are not aiming to have excellent recognition rates, we are simplifying our algorithm by using data in a consistent form and using building blocks in the form of off the shelf keypoint detectors and descriptors.

The first assumption we are making is that we only need to be scale and rotational invariant. We do not deal with the case of perspective changes, thus we do not need to account for affine transformations. Along with this, our query symbols and the symbols within our data set are drawn almost identically. In fact, we are taking our query symbols directly from our data set. By doing this, we are introducing a level of predictability to our system. We are also assuming that there is a high level of structure within the symbols that allows for enough keypoints to be detected. Harris-Laplace identifies keypoints as areas with high curvature, so by using symbols such as typed text with high curvature, we are ensuring there are many keypoints detected. If we did not have sufficient keypoints within a symbol, multiple symbols may be classified as the same. Lastly, since we are using off the shelf keypoint detectors and descriptors such as Harris-Laplace and SIFT, we expect our results to not be very representative of how the system would perform with other algorithms, or even from the original paper which looked at multiple descriptors.

If any of our assumptions are wrong, assuming our algorithm is validated, we will mainly be able to tell based on the results we obtain. Since we designed the system to be predictable, we assume both the recall and precision will be high. Poor results could indicate our assumptions are wrong, and that we need to re-evaluate.

IV. Timeline

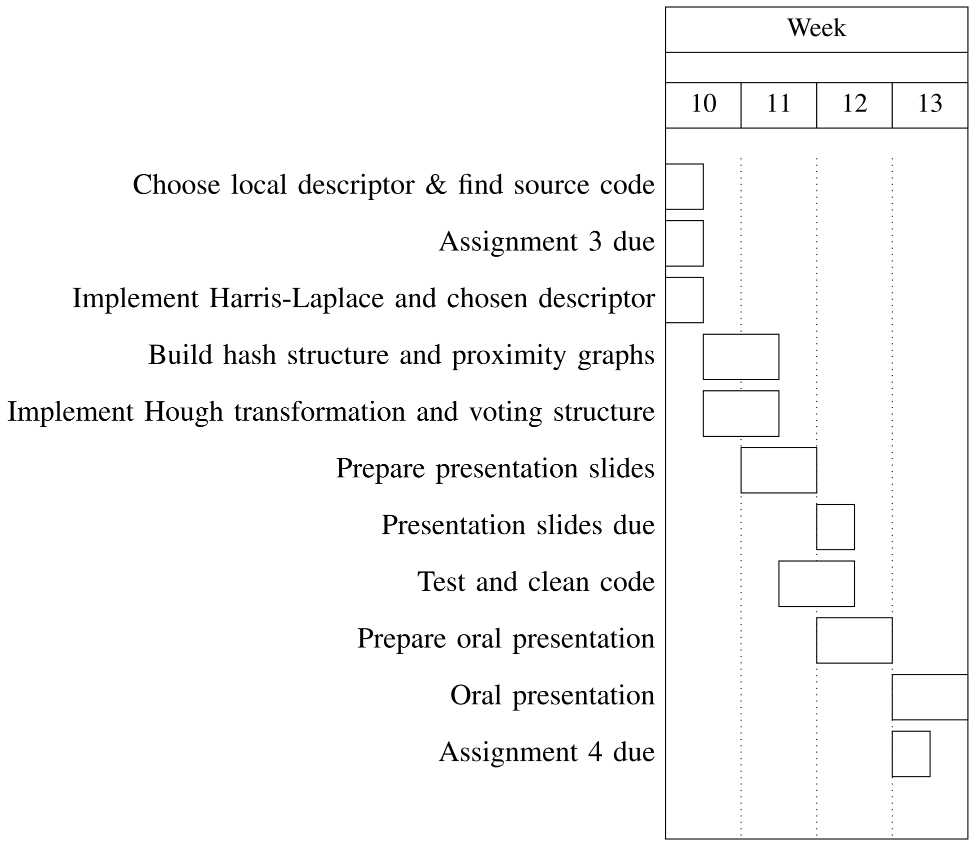

Fig. 2. Updated Gantt Chart

As of Sunday March 16th, we have not made much implementation progress. This is largely due to us being busy with other assignments and also the uncertainty of how courses will proceed for the remainder of the semester. As such, we have had to shift our timeline forward slightly and condense our development time. The first half of week 10 will consist of implementing our keypoint detector and descriptors. We are using off the shelf implementations, so we feel this will not take as long as we had anticipated. The only other change to our timeline as of this point is we cut half a week of time off of building the hash structure and proximity graphs to compensate. This will likely be the most difficult portion of development, but with in-person classes being canceled, we feel we will make up the lost time with time we would have otherwise spent commuting and on campus.

REFERENCES

[1] "Harris Corner Detection," *OpenCV*. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_features_
[Accessed: 17-Mar-2020].

[2] "Introduction to SIFT (Scale-Invariant Feature Transform)," *OpenCV*. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html. [Accessed: 17-Mar-2020].

[3] "Hough Line Transform," *OpenCV*. [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghlines
[Accessed: 17-Mar-2020].