Interactive Object Tracking in a Live Video

By Duncan Calder
Clark University
For Computer Vision CSCI 262

Problem

The idea behind this project is to create a system that will allow you to select and track an object using a camera in real time and continue tracking when the camera is moved closer or further away. The real problem to solve is using the algorithms in OpenCV to implement an efficient and accurate object tracking method that can be used to track an object.

Motivation

The application of this will be to apply it to a ride-on toy car so that a disabled child can play by selecting something they want to go towards or follow and the car will move in that direction. Since setting up the mechanics of applying the project to a ride-on toy car would not fit with the goals of this course I will constrain myself to only working on the system that selects and tracks the object.

Related work

Distinctive Image Features from Scale-Invariant Keypoints, David G. Lowe
TemplateMatching OpenCV documentation and demo
SIFT Matching OpenCV documentation and demo
Mouse Events OpenCV

Methods

I tested SIFT, SURF, and template matching to find the best method to track the selected object. I have two different programs, one program that runs template matching function and one that runs the feature matching.

In the program that implements the feature matching I used a method found David G. Lowe's paper mentioned above to find the best matches. This method compares two of the best matches from the descriptors and using a variable ratio it keeps the matches whose best matches are close to each-other. This avoids keeping matches that are not good matches.

After the matches are found I took the average of all the locations of keypoints that were matched and used that to decide if the object was left or right of the camera. This function works the same in both of the programs created. It compares the location of the decided upon match with the calculated center of the display image to find if it is left or right of the center. The directions left and right are calculated in the perspective of the camera. There is some wiggle room given in the left and right calculations which can be changed. This wiggle room is represented in the percentage of the image that the "wiggle" will take up. When the object is in the "wiggle space" no output is sent to the console meaning that no movement is required. This method also sets up a way to implement a system for measuring up and down location but for this project it is not yet necessary as only left and right directions to the selected object are needed.

The object that is to be tracked is selected by the user pressing the key “p” and drawing a rectangle on the webcam image. The view of the rectangle is updated in a separate cropped window in real time. The user can reset and reselect the image at any time by dragging a new rectangle or clicking outside the rectangle to reset the rectangle. Once the user is satisfied with the object in the cropped image they can press “s” to start tracking.

The default drawMatches method is used to display the keypoints calculated on both the cropped image and the webcam image and lines are drawn between the two to show where the matches are being found.

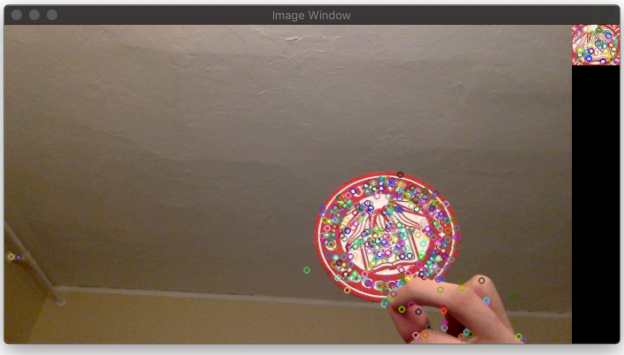
Evaluation

To test this program I compared the accuracy of the matches found in the three different matching methods, SIFT, SURF, and template matching. I took screenshots of the location found of the object (left or right) and the display from the webcam image showing the matched location. For SIFT and SURF I adjusted the ratio of distance between the two good matches chosen. I used a coaster featuring the Clark logo as the tracked object. I tried four different ratios, .2, .4, .6 and .8 to determine which worked the best with the tracking efficiency and accuracy. For template matching I was unable to make any changes to the matching so showed the results as is. With the best found ratio I then tested how the algorithm dealt with the tracked object getting further away.

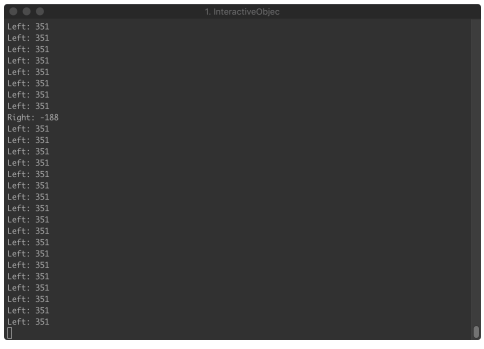
Results

SIFT:

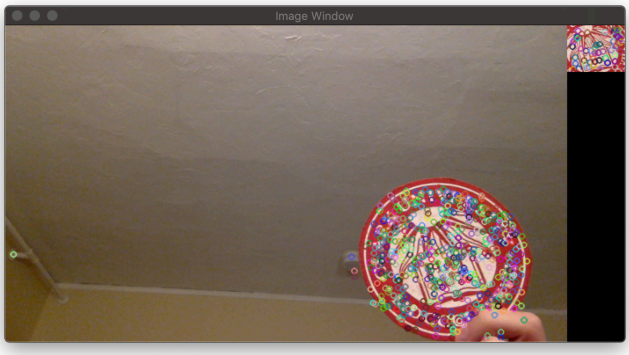
.2



.4

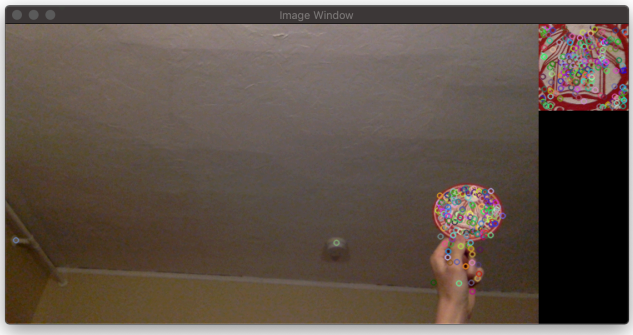


.6

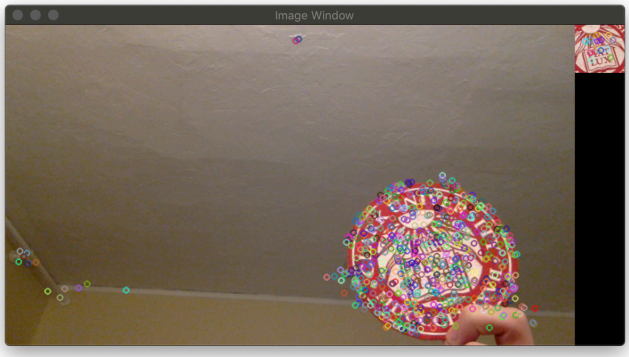


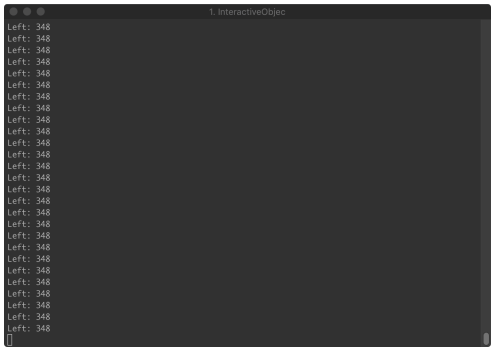
.8

Best (.7) Far:

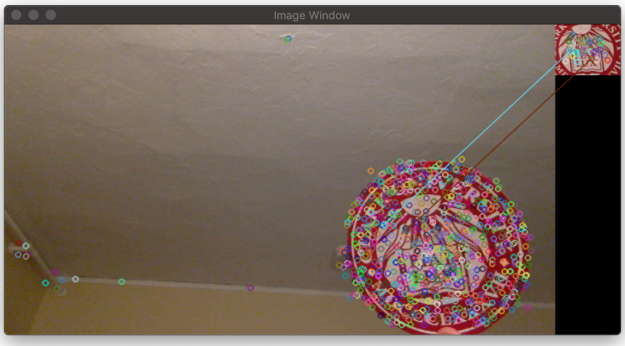


SURF:
.2

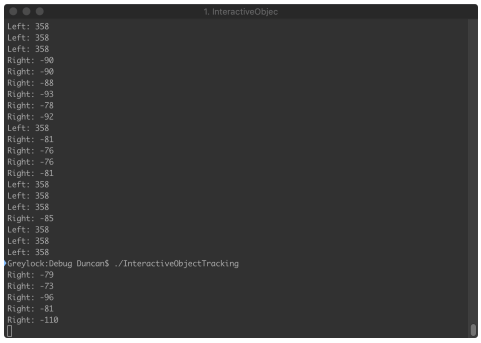




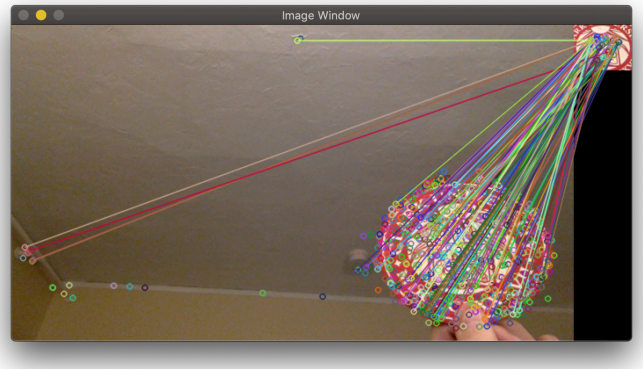
.4

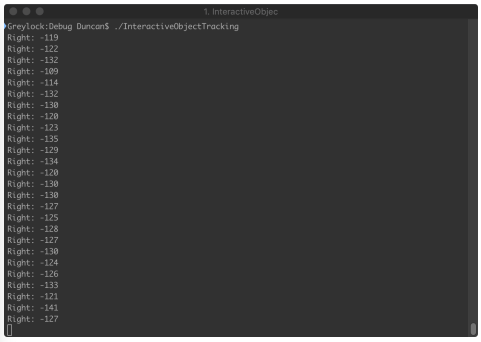


.6

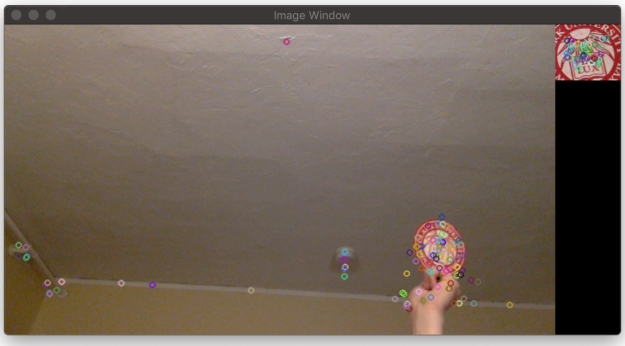


.8





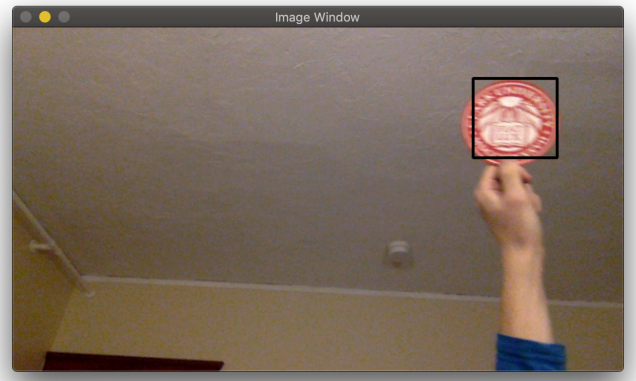
Best (.7) Far:



Template Matching:
Close:

Far:
Not Moving:

Moving:



Discussion

I did not completely finish all that I had hoped to do. Tracking an object in a live feed proved to be more complicated than I thought. I was unable to move onto the point of being able to figure out how far away an object is based on the tracking results and thus was unable to tell the

program when the object was close enough for the car to stop moving forward. The methods that I tested were a fair representation of the best that openCV provides to implement object tracking used at their base usage level. If more modifications were added to the tracking algorithms then the tracking could easily be made better.

I reviewed the testing and looked at the amount of variation in the direction that the console outputs and the amount of matches found. The target stayed on the right side of the screen the entire time so if the console shows that the object was detected on the left that must be read as a mistake in the program.

Following those analysis measures, SURF works best while using the ratio .7 at both close and far locations. SIFT was a close second, but template matching was not at all a good way to implement object tracking, at least not without modification. I was surprised that the template matching worked at a further away distance, though for some reason the matching worked best when the target was being moved. This was most evident in the test where the target was further away, but was also seen to some degree when the target was close.

In completing this project I learned a lot about how to combine different OpenCV methods together and to take from different sources to combine into my own project. I realize that I should have focused more on implementing one method than trying to test out multiple methods. I ran out of time to completely finish my project because I was trying to juggle multiple programs and method at the same time instead of focusing on one. I was surprised that SIFT does not work as well as advertised, it is not a catch all best method for tracking and matching. I should have tested the program on a blank background so I could start working on the scale tracker earlier. Since I was too caught up with getting the tracking to work well I was unable to move onto my later objectives. If I was able to do a bit more work and focus on implementing a scale tracker into this program I believe that I would be able to finish it.

Code: https://github.com/dunder51/Interactive-Object-Tracking