Ryan Holloway
300570174

CGRA 352 - Assignment 4 - Report

All the code is written within one .cpp file. I only included that one .cpp file, since the entire project is stored in a very large folder. To get the code to run, you can copy paste all the text within the .cpp (or .txt) file into any IDE that is currently running C++ with OpenCV. The only change that needs to be made to the code is on line 209, where the file path is defined for the location of the image-sequences images, as this will be specific to where the images are stored on the machine that is running the code. Be sure to use forward-slashes, and not backslashes between each directory.

Functions:

float calc_ssd_KeyPoint_KeyPoint(KeyPoint feature_point_1, KeyPoint feature_point_2)
This function returns a float that is the SSD between two KeyPoint values. This function is useful for calculating what feature point matches are inliers or outliers.

float calc_ssd_KeyPoint_Point2f(Point2f feature_point_1, Point2f feature_point_2)
This function returns a float that is the SSD between two Point2f values. This function is useful for calculating what feature point matches are inliers or outliers.

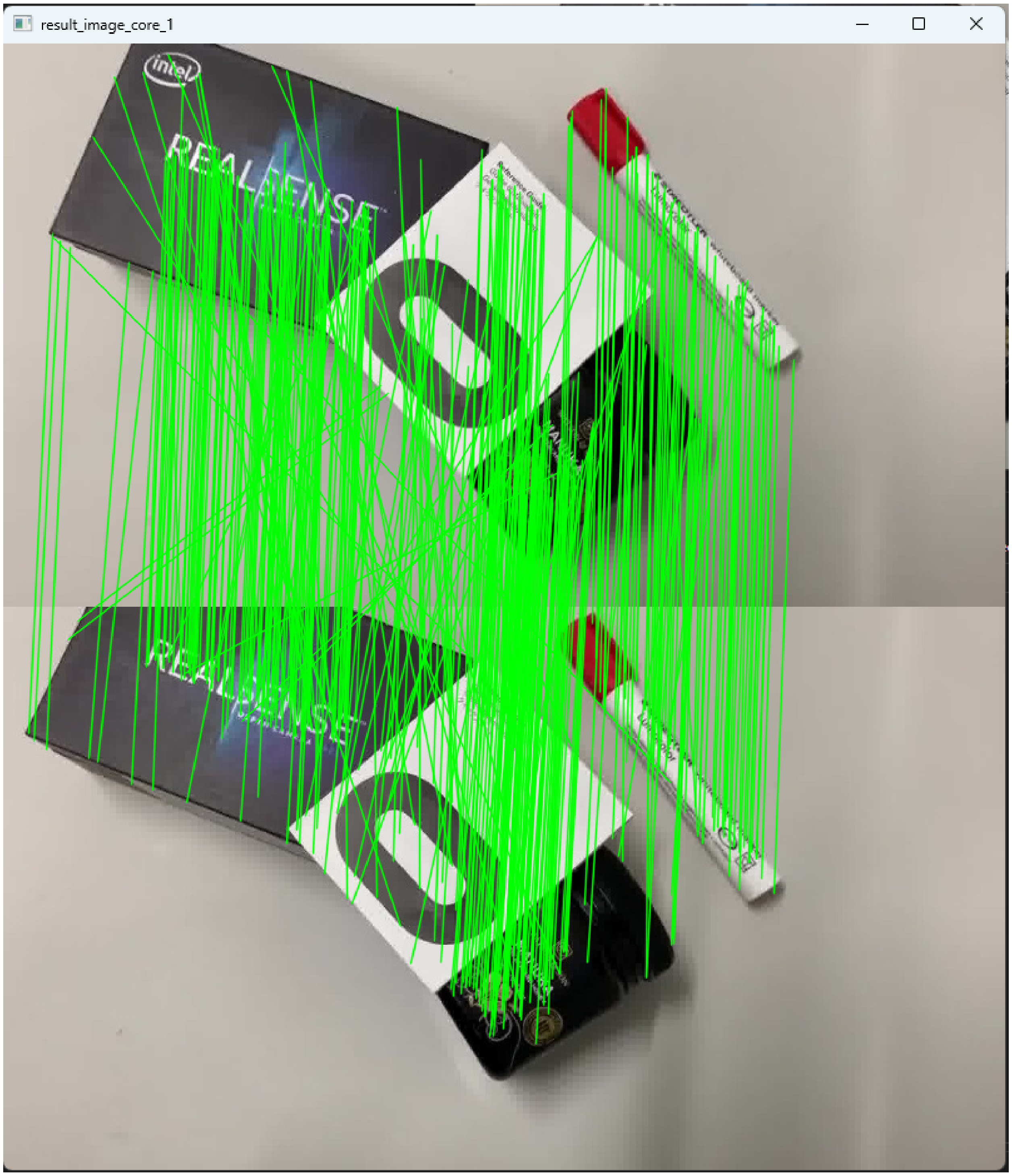
cv::Mat calc_best_homography(cv::Mat current_frame, Mat next_frame)
This function detects all the SIFT feature points of the two input images from the function's parameters and stores them in corresponding vectors. Then brute force matching is processed using cv::BFMatcher and stores the matches in a vector of type DMatch. Then the RANSAC process is started, which iterates 100 times. For each iteration, four random matching pairs are selected and are used to find the homography transform between the matching points. This homography transform is used to identify inliers and outliers by applying the homography transform to one of current frame's feature points, then computes the SSD to where the result is compared to its known matching feature point. If the SSD is less than the epsilon value, then it is an inlier, if it is greater than the epsilon value, then it is an outlier. The inliers' and outliers' corresponding indices are stored in respective vectors of int's. Additionally the number of inliers is counted in order to determine which iteration had the most inliers, which determines which homography transform was the best and most accurate version. The best homography transform found in the 100 iterations is then applied to the best iterations' found inliers, which produces the final and best homography transform. This homography transform is returned as a cv::Mat.

Core Section
The core section is very similar to the calc_best_homography() function; however it specifically performs the operations of frame_039 as the current frame, and frame_041 as the next frame. The results of all the feature points matches are displayed in core part 1 as green lines connecting the points across the two images. In core part 2, the green lines correspond to the inliers, while the red lines correspond to the outliers. Core part 3 attempts to warp frame_041, using the best homography transform found from the RANSAC process, then overlay the resultant image on top of frame_039. While the warped version of frame_041 appears to produce a somewhat accurate warping result, the overlay of the images seem to be off. This is possibly due to the border offset and how the images are aligned afterwards.

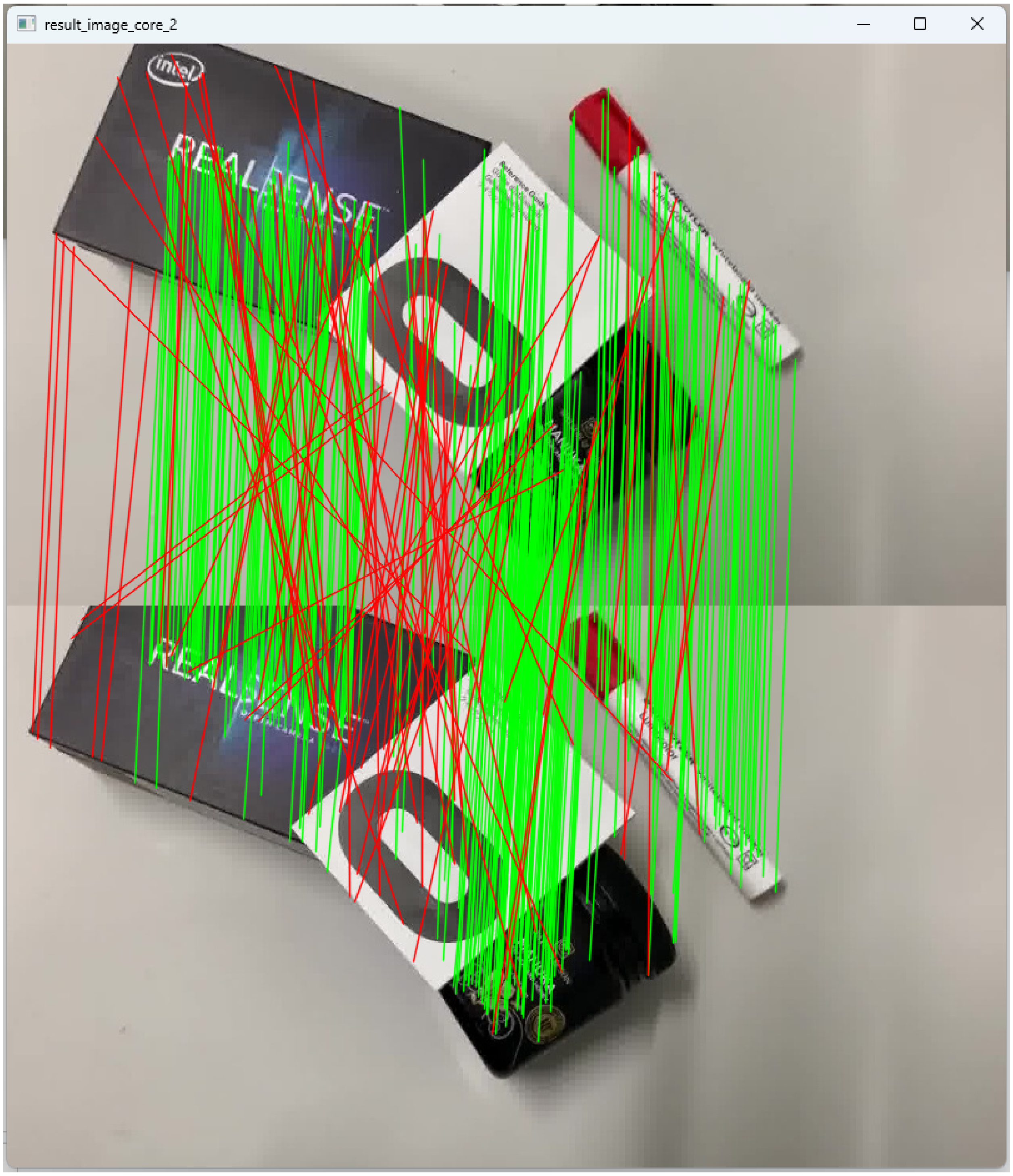
Completion Section
Completion is an attempt to implement the found homography transforms to produce a video stabilization algorithm across the entire image-sequence. I believe the general algorithmic process that I've implemented is correct; however, the results are either difficult to see if they are working, or are not making a significant change from the original image sequence. I found the homography transform between each frame, while cumulatively creating and storing all the H_tilde values in a vector, as well as a cv::Mat for the current H_tilde value. I found the gaussian of every H_tilde value by applying the formula with discrete weighted values producing a gaussian result. I then computed the inverse of all the H_tilde_gaussian values and stored them in a vector with corresponding indices to the H_tilde_gauss vector. Having those two vectors enabled me to compute the U transform values for each respective frame where a warp should be applied. I then iterated through all the original frames and applied the U transform to the frames that should be warped and output those resultant images as png's to the disk.

<u>Results images:</u>

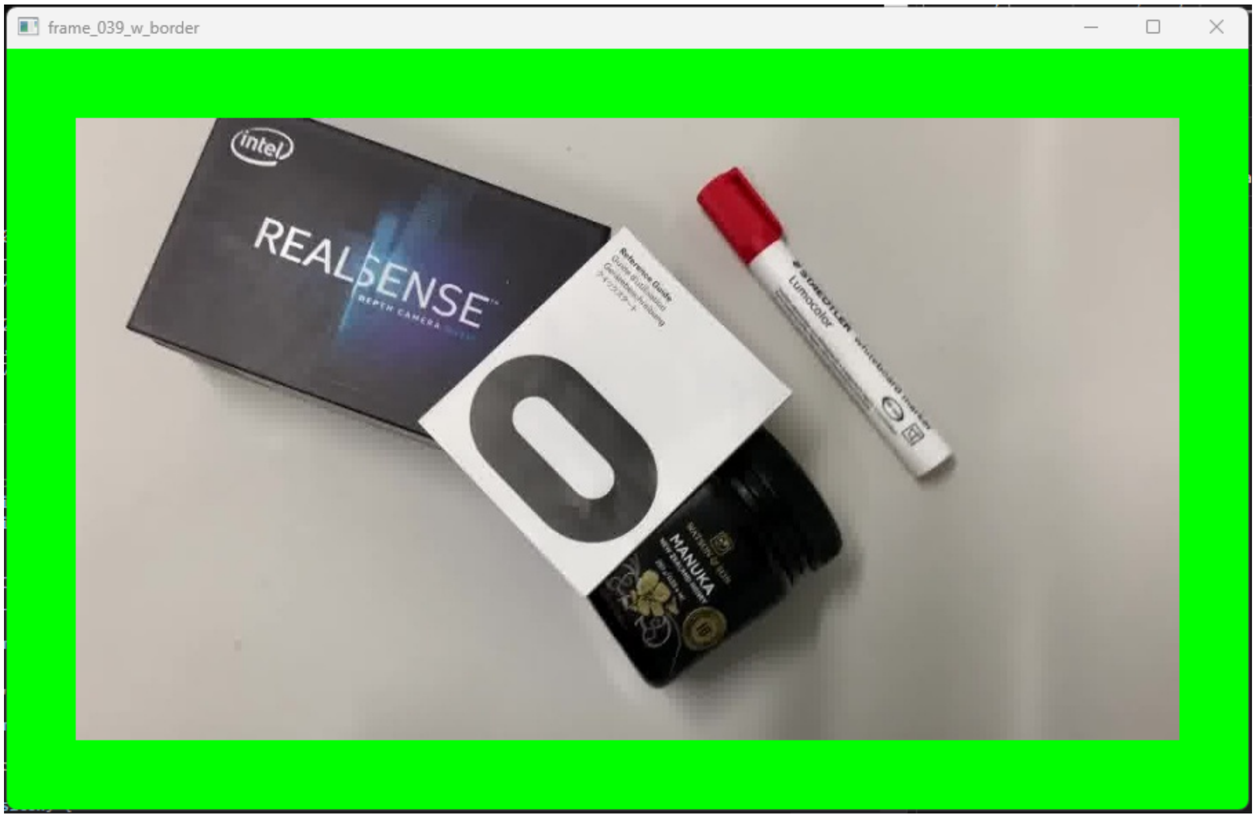
<u>Core Part 1:</u>

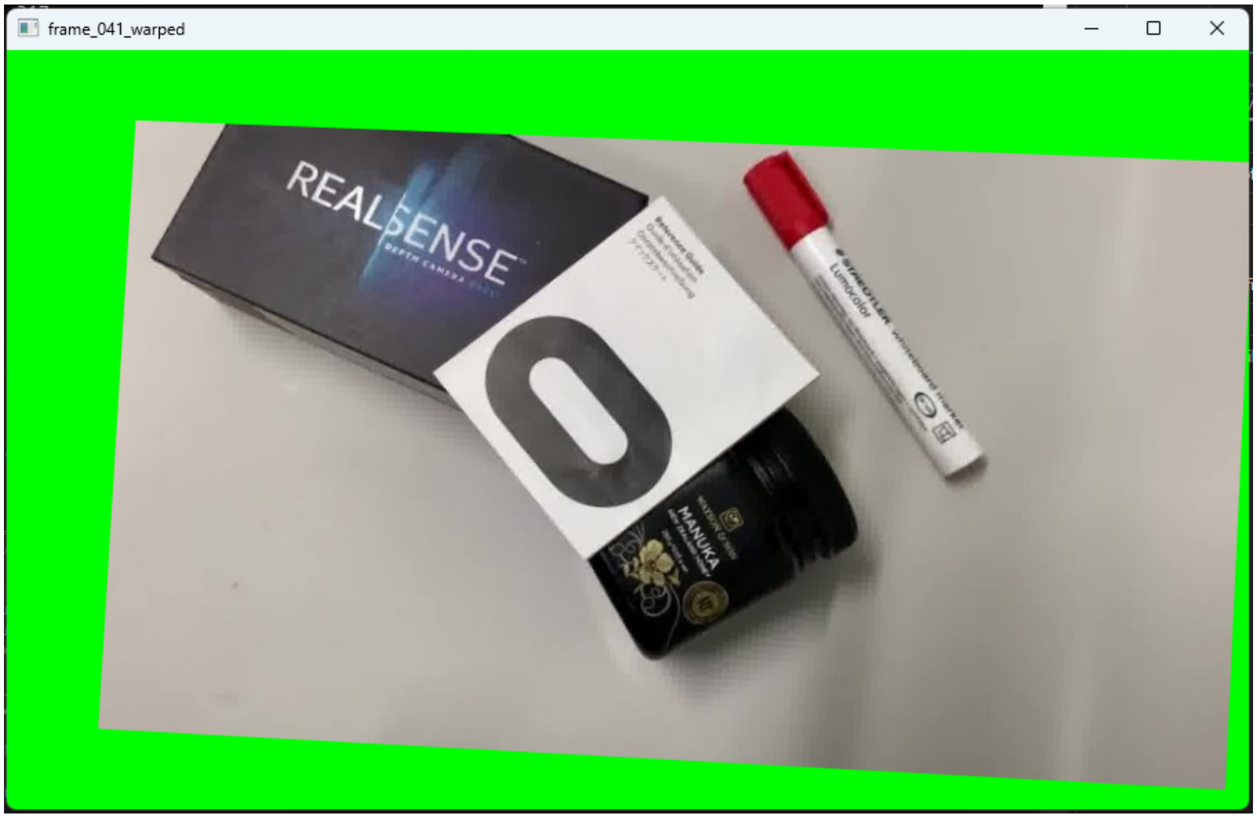


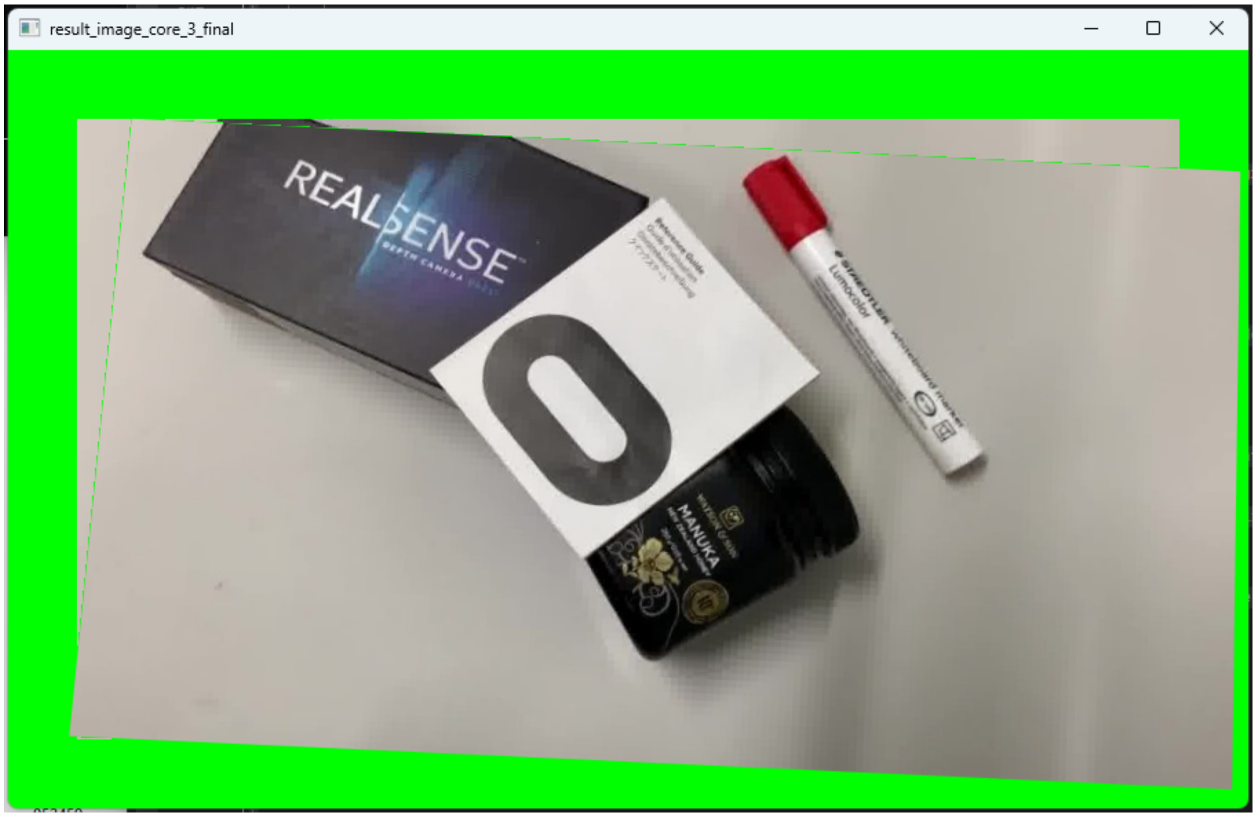
Core Part 2:



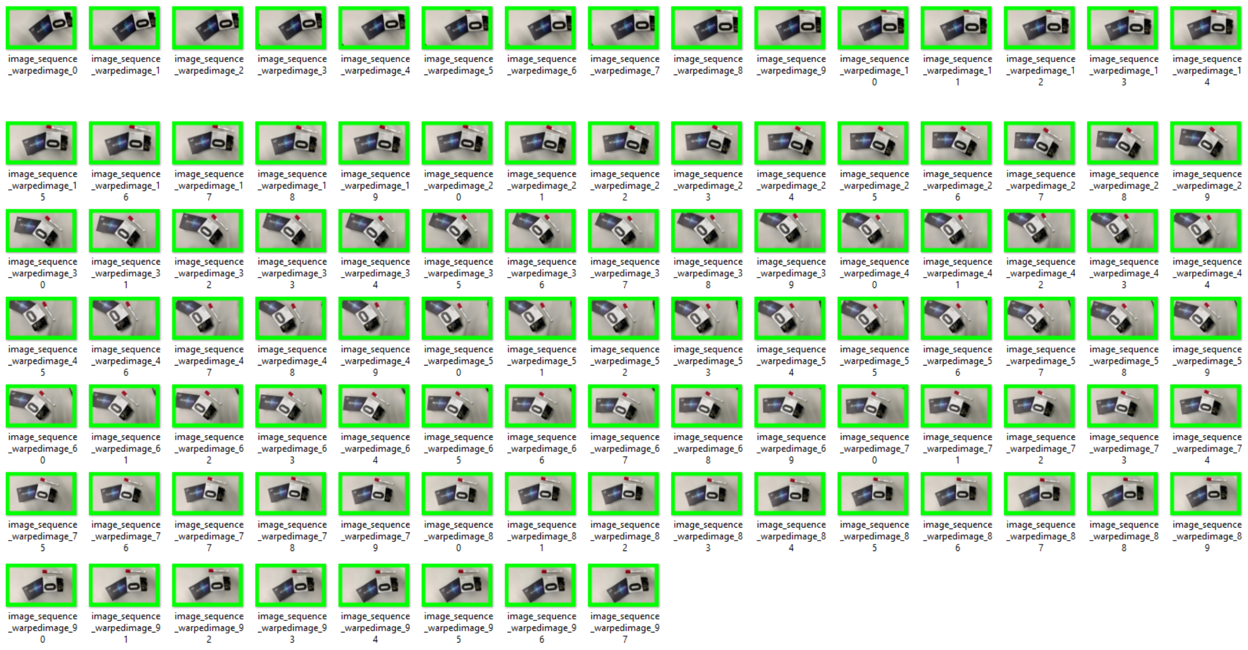
Core Part 3:


frame_039_w_border
intel
REALSENSE
DEPTH CAMERA
Reference Guide
STAEDTLER Lumocolor whiteboard marker
MANUKA


frame_041_warped
REALSENSE
DEPTH CAMERA
Reference Guide
STAEDTLER Lumocolor whiteboard marker
MANUKA



Completion:


image_sequence_warpedimage_0
image_sequence_warpedimage_1
image_sequence_warpedimage_2
image_sequence_warpedimage_3
image_sequence_warpedimage_4
image_sequence_warpedimage_5
image_sequence_warpedimage_6
image_sequence_warpedimage_7
image_sequence_warpedimage_8
image_sequence_warpedimage_9
image_sequence_warpedimage_10
image_sequence_warpedimage_11
image_sequence_warpedimage_12
image_sequence_warpedimage_13
image_sequence_warpedimage_14
image_sequence_warpedimage_15
image_sequence_warpedimage_16
image_sequence_warpedimage_17
image_sequence_warpedimage_18
image_sequence_warpedimage_19
image_sequence_warpedimage_20
image_sequence_warpedimage_21
image_sequence_warpedimage_22
image_sequence_warpedimage_23
image_sequence_warpedimage_24
image_sequence_warpedimage_25
image_sequence_warpedimage_26
image_sequence_warpedimage_27
image_sequence_warpedimage_28
image_sequence_warpedimage_29
image_sequence_warpedimage_30
image_sequence_warpedimage_31
image_sequence_warpedimage_32
image_sequence_warpedimage_33
image_sequence_warpedimage_34
image_sequence_warpedimage_35
image_sequence_warpedimage_36
image_sequence_warpedimage_37
image_sequence_warpedimage_38
image_sequence_warpedimage_39
image_sequence_warpedimage_40
image_sequence_warpedimage_41
image_sequence_warpedimage_42
image_sequence_warpedimage_43
image_sequence_warpedimage_44
image_sequence_warpedimage_45
image_sequence_warpedimage_46
image_sequence_warpedimage_47
image_sequence_warpedimage_48
image_sequence_warpedimage_49
image_sequence_warpedimage_50
image_sequence_warpedimage_51
image_sequence_warpedimage_52
image_sequence_warpedimage_53
image_sequence_warpedimage_54
image_sequence_warpedimage_55
image_sequence_warpedimage_56
image_sequence_warpedimage_57
image_sequence_warpedimage_58
image_sequence_warpedimage_59
image_sequence_warpedimage_60
image_sequence_warpedimage_61
image_sequence_warpedimage_62
image_sequence_warpedimage_63
image_sequence_warpedimage_64
image_sequence_warpedimage_65
image_sequence_warpedimage_66
image_sequence_warpedimage_67
image_sequence_warpedimage_68
image_sequence_warpedimage_69
image_sequence_warpedimage_70
image_sequence_warpedimage_71
image_sequence_warpedimage_72
image_sequence_warpedimage_73
image_sequence_warpedimage_74
image_sequence_warpedimage_75
image_sequence_warpedimage_76
image_sequence_warpedimage_77
image_sequence_warpedimage_78
image_sequence_warpedimage_79
image_sequence_warpedimage_80
image_sequence_warpedimage_81
image_sequence_warpedimage_82
image_sequence_warpedimage_83
image_sequence_warpedimage_84
image_sequence_warpedimage_85
image_sequence_warpedimage_86
image_sequence_warpedimage_87
image_sequence_warpedimage_88
image_sequence_warpedimage_89
image_sequence_warpedimage_90
image_sequence_warpedimage_91
image_sequence_warpedimage_92
image_sequence_warpedimage_93
image_sequence_warpedimage_94
image_sequence_warpedimage_95
image_sequence_warpedimage_96
image_sequence_warpedimage_97