Computer Vision Final Report

Mikey O'Connor and Kevin Peng

December 2022

1 Abstract

Robotic are a fundamental field of Computer Science as there are many ways to control and move a robot. Some of the ways to control such a robot include joysticks, touch pads, and in the case of this project, hand motion. We will use both optical flow and gesture recognition to move a MyroC Scribbler 2 Robot. We combine a Lucas Kanade Optical flow method with a convex hull gesture recognition algorithm in order to create the robot motion and in ideal scenarios, the algorithms work with great success.

2 Introduction

Optical flow is a well-studied method to detect changes immediate changed between two images in an object. We plan to leverage these optical flow algorithms to recognize motion of a hand captured by camera and then transfer that motion into a motion descriptors. For our motion descriptions, we plan to use magnitude calculations based off the Optical Flow diagram in addition to gesture recognition, another well used technique which will be fed into our robot functions. After the conversions, we plan to use these results to create robot speed, motion and duration.

3 Background Literature

For our optical flow algorithm, the fundamental piece of literature would be the work from Lucas and Kanade's research on the topic [3]. In this paper, they describe the foundations of optical flow being motion from one object to another, assuming that motion is very minimal between the two objects. However, for our implementation, we mostly focused on the implementation from Trucco and Verri and their textbook analyzing the optical flow algorithm [4]. Overall, their optical flow algorithm was proceduralized over several images to find the optical flow between several images. From this, we have developed and created our general algorithm for optical flow.

For our gesture recognition algorithm, we based our algorithm off of the research from Universidad Autónoma de Nuevo León, which contains most of our implementation [1]. In this paper, they use a skin pigmentation, edge detection, and convex hull algorithms to determine gesture of the hand as a whole.

4 General Robot Motion Process

Our general robot motion process consists of several different algorithms and methods working together to create the motion we desire from the robot. In this section, we will go into great detail describing these sections as a whole.

4.1 Optical Flow

This section will be split into two parts, one part describing the optical flow algorithm, and the other describing the direction and angle from the optical flow vector field.

4.1.1 Optical Flow Algorithm and Implementation

As stated in our background literature section, our understanding of optical flow is derived from the Lucas and Kanade method [3], as in this paper, it finds the difference between the two functions, which will represent the motion of the object as a whole. However, following the implementation from Trucco and Verri [4] and refinements in the meta analysis of Optical Flow techniques [2], is where we found and implemented our actual representation of optical flow. In this implementation, we take a series of images, which will be based off a standard deviation of .5, and smooth them both spatially and temporally, solving for our change in the x, y and time directions respectively. After solving for these components, we then solve a directional vector at point p, using the $Ax = b$. The A component will be derived from the change in x and y direction in a patch (15x15) around p, while the b component will be the inverse of the change in the temporal component in the patch around p. We increased the patch from the initial 5x5, as to improve the speed of the algorithm, especially in higher resolution images. From this, we use the least squares approach to solve for x, which will produce our vector representing the change in motion across the set of images for that specific point, p. We then produce this over the entire image, resulting in a vector field that represents the change of motion over the entire image.

4.1.2 Optical Flow Directionality

After we produce our vector field, we need to quantize it so that we can have four discrete directions. We will do this by finding the magnitude of the vector field and only using the largest vectors to determine our direction. After that, we find and average the x and y component of these vectors and utilize that to produce an angle using arctan, which we will then use the unit circle to describe direction the angle points. We thank Jerod for his help for finding the best way to implement this algorithm.

4.2 Gesture Recognition

In addition to our optical flow algorithm, we will also include our gesture recognition algorithm which is based off 3 different parts, skin pigmentation, edge detection, and convex hull.

4.2.1 Skin Pigmentation

For our skin pigmentation, the color space of RGB will not be particularly useful and a conversion into the YCbCr color space was needed in order to perform our analysis of the skin pigmentation. From this, we are able to use the color space to filter out any colors that match with the skin tone. [1] However, there are various faults with using skin pigmentation as it may not be able to cover both darker and paler skin tones as the human skin is a very diverse color palette. In addition, there are quite a few false positives with this algorithm, as wood for example, is a false positive that the skin pigmentation will detect. Further exploration of hand detection would lead to further growth in the algorithm, however, as convex hull is a major bottleneck and hand data from the vector field is very coarse with no useful hand geometry information, this is a speedy way to differentiate the hand from the rest of the objects.

4.2.2 Hand Contour Extraction

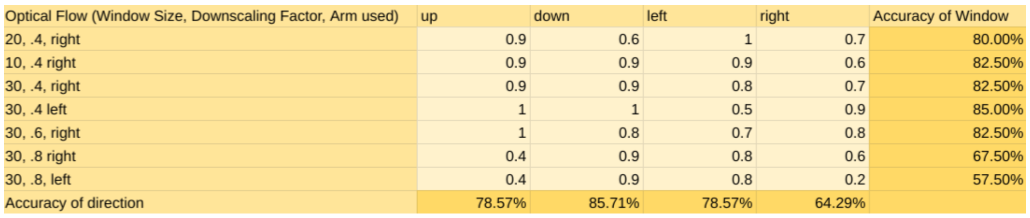
After detecting the hand with skin pigmentation, the next step in the algorithm is to find the edge of all the objects identified as skin in the images so that we can get the contour of the hand. [1]. This process will use a basic edge detection function, provided by MATLAB. After finding the edges, we will subset the edges into different connected sets and get the largest set assuming that the hand as a foreground is the largest set. This is achieved with MATLAB's built-in function, bwconncomp, suggested by Jerod. Since the functions we will use later assume that the contour of the hand is a thin line, we use MATLAB's bwskel to reduce the hand edge into a thin line. The function was also suggested by Jerod. Then we will use convex hull over that set. The issues with the color space continue to follow with issues in this step. For instance, if other colors are pigmented similarly to the hand, the hand could merge into the object, creating a blob that is unusable. In addition, if any skin-like pigment is larger than the hand, the bwconncomp function will take the larger skin-like pigment rather than the hand we desire. The solution we had for both skin pigmentation and edges was to control the background that pigmentation and segmentation happened over, which we decided to be a gray backdrop, as it yielded the best results for the separations of the color pigments.

4.2.3 Convex Hull

After producing the hand contour set, we use MATLAB's built-in convhull function on it to put the whole hand into a simple polygon with much fewer vertices than a complex hand shape. Because of the protruding fingers, each finger tip will be one of the vertices of the polygon. Then, to find the troughs between fingers (i.e. convexivity defects), all the pixels along the hand edge between two consecutive vertices need to be identified. This was one of our bottleneck with the gesture detection algorithm since the hand contour set returned by bwconncomp is not in a connected fashion such that consecutive coordinates are adjacent pixels in the hand. Instead, the pixels are arranged in columns. So it makes our task to find convex defects tremendously harder. To accommodate that, we wrote our own functions (findMiddleStart and findMiddle) which gets a hand and two consecutive vertices (found by convhull) on the hand and finds all the pixels between the two vertices along the hand contour. The functions findMiddleStart and findMiddle basically start from a vertex, look at the 3*3 window around that vertex (pixel), our start point, follow any adjacent pixels, save the pixels into list(s), and hopefully along the path find vertex 2. The difference between findMiddleStart and findMiddle is that findMiddleStart looks for vertex 2 in both directions since we don't know which direction is correct. At the beginning of findMiddleStart, the function also checks the two adjacent pixels, sees which one is closer to our target, and search in that direction twice every time. Then, once we have found at least one path along the hand contour between two consecutive vertices, we are able to set the pixel immediately prior to the target vertex to 0, so that when our target vertex becomes the start vertex and we need to find the path to the next vertex, our algorithm will not look back in the wrong direction, and we can just search for the next vertex linearly in one direction using findMiddle. Then, for the points between every pair of consecutive vertices, we use another function to find the point that is the farthest from the two vertices and pronounce that point the defect. The function that finds the defect also returns the perpendicular distance from the defect to the vertices and the angle between the two vertices at the defect, which we will later use to determine the validity of a defect.

4.2.4 Gesture Classification

To finally produce a number that indicates the number of fingers, we count the number of valid defects and plus one. To determine whether a defect is valid or not, we check two things. First, we check if the perpendicular distance of the defect from the two vertices is greater than half of the palm radius. We calculate the palm radius by first find the center of all defects, and then find the mean distance from the defect center to all the defects. Second we check if the angle of finger opening is bigger than 90 degrees. We reject the defects whose angle calculated is greater than 90 degrees since most human should not be able to have a finger opening greater than that angle. If a defect satisfies both criteria, it is identified as a finger trough. Finally, the number of valid defects plus one



Optical Flow (Window Size, Downscaling Factor, Arm used)	up	down	left	right	Accuracy of Window
20, .4, right	0.9	0.6	1	0.7	80.00%
10, .4 right	0.9	0.9	0.9	0.6	82.50%
30, .4, right	0.9	0.9	0.8	0.7	82.50%
30, .4 left	1	1	0.5	0.9	85.00%
30, .6, right	1	0.8	0.7	0.8	82.50%
30, .8 right	0.4	0.9	0.8	0.6	67.50%
30, .8, left	0.4	0.9	0.8	0.2	57.50%
Accuracy of direction	78.57%	85.71%	78.57%	64.29%	

Figure 1: Results from analysis of optical flow

is the result. If the result is not between 0 to 5 inclusive, it will return -1, which indicates nullGesture.

5 Experimental Data

For experimentation with our data, we plan to do experimental testing of the accuracy of our optical flow output, and gesture recognition output separately and independently, as they are the elements that power our robot motion.

5.1 Setup for Optical Flow Testing

For our optical flow, we will run the 8 tests 4 times, resulting in 16 different tests for each of the motion categories (up, down left and right), showcasing the various conditions that our optical flow algorithm can handle and not handle. In addition, we will include some of the results of changing various parameters like the window size and the down sampling factor of our algorithms. Overall, we tested our data sitting down in front of a webcam provided by the CS Department.

5.2 Setup for Gesture Recognition Testing

For our gesture recognition algorithm, we will run our tests for all the fingers and report the accuracy of the finger recognition and compare those results to that of the gesture recognition paper [1]. Our testing of the gesture algorithm will be testing 7 different cases, including 0, 1, 2, 3 ,4 , 5 fingers from four different angles (back, palm, front, and lateral), and null gesture case where we will just feed the algorithm random images. For all the finger number test cases, we will be recording the times the algorithm returns the right number, while for the null gesture case, we count it as a success as long as it does not crash. We will be testing this with a go pro connected to our MacBook with the back of portable whiteboards as our background.

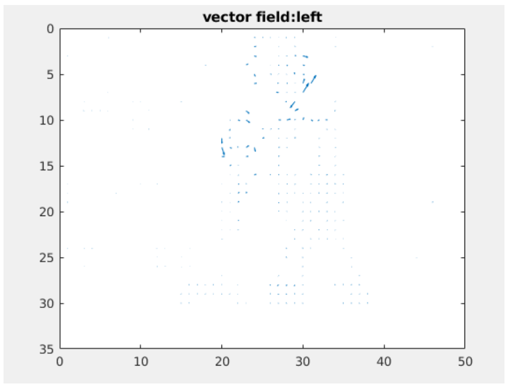


Figure 2: Sample Vector Field for Down, with a patch size of 20 ad down sample factor of .4

6 Analysis of Results

After establishing our testing methods, we plan to analyze the results of our various tests on the algorithms, determining how successful the algorithms were. Our analysis will include the various test cases with did with hand motion, gesture recognition, and overall success of the algorithm. In addition, we will analyze the weakness each algorithm.

6.1 Analysis of Optical flow testing

For our optical flow data, we found that it had several success and failures based off the motion in general. Overall, we found that a larger patch size with a smaller down sampling scale produced the best results for our optical flow as shown below:

Overall, our data was not as accurate as we would have hoped for all the different parameters of the data. However, the algorithm did seem to excel once, with a down sampling factor of .4 and a patch window of 30.

6.1.1 Success with Optical Flow

The strengths of this optical flow algorithm is the speed of the data as the slow picture taking camera was a major limitation in how good our data was. Our algorithm was quite fast as it was able to sample the image efficiently with both a large patch size and the sampling factor from our algorithm itself. Even with this speed, the accuracy was quite high especially, in the ideal data range for up and down.

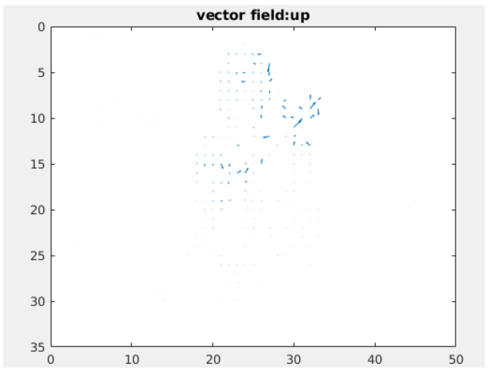


Figure 3: Failure, Detecting left as up. Window patch 30, Down sampling factor .4

6.1.2 Limitations with Optical Flow

Despite these advantages, there was some limitations with our optical flow especially with a slow camera. Overall, optical flow requires small changes in the motion and with a slow camera, it is very hard to observe the changes in motion. Some of these factors can include extraneous motion in other body parts or the background and motion being too fast, both of which, is very hard to control, when your arm. However, in our robot motion and gesture recognition, the data will be in a lot more of an ideal state, allowing for changes in motion to be a lot more obvious.

6.2 Analysis of Gesture Recognition

When testing our gesture recognition algorithm, we found an interesting correlation between the success rate and the number of fingers.

Notice from the graph that there is a positive relationship between the back/palm success rate (the orange line) and the number of fingers. We believe that this is caused by our method of differentiating numbers of fingers. Since our method relies on detecting finger tips as vertices and troughs between fingers as defects, the lack of them in fists or one-finger gesture makes our algorithm really bad at telling apart 0 and 1 finger. Moreover, there is a negative relationship between the front/lateral success rate (the gray line) and the number of fingers. We fathom that it is because with fewer number of fingers, the hand looks more identical from different angles, while with more fingers, the fingers lose their depth when viewed from the front, and overlap with each other when viewed from sides. If we ignore the front/lateral success rate (since we can easily avoid these two angles when we wave our hand in front of the camera), all the success rate are above 50% with 100% accuracy on 4 and 5 fingers. Regarding the data of nullGesture testing, we found that the program will never crash as long as there is some skin-colored things in the image. We quickly fixed that and now

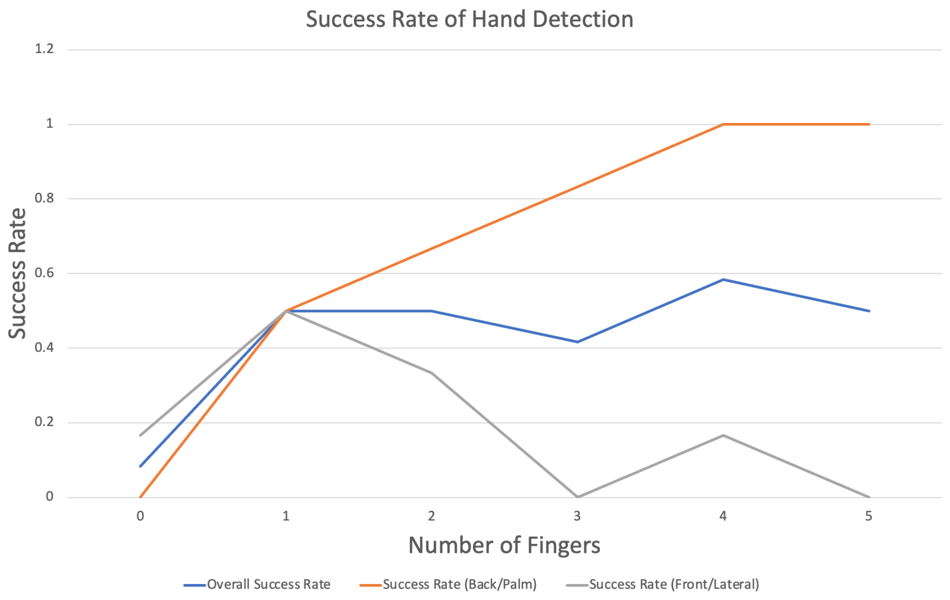


Figure 4: Graph of Success Rate of Hand Gesture Detection

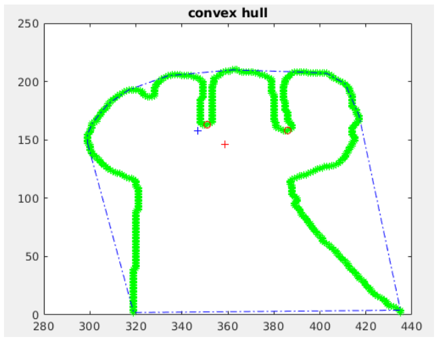


Figure 5: Failed Gesture for Zero, Right Palm

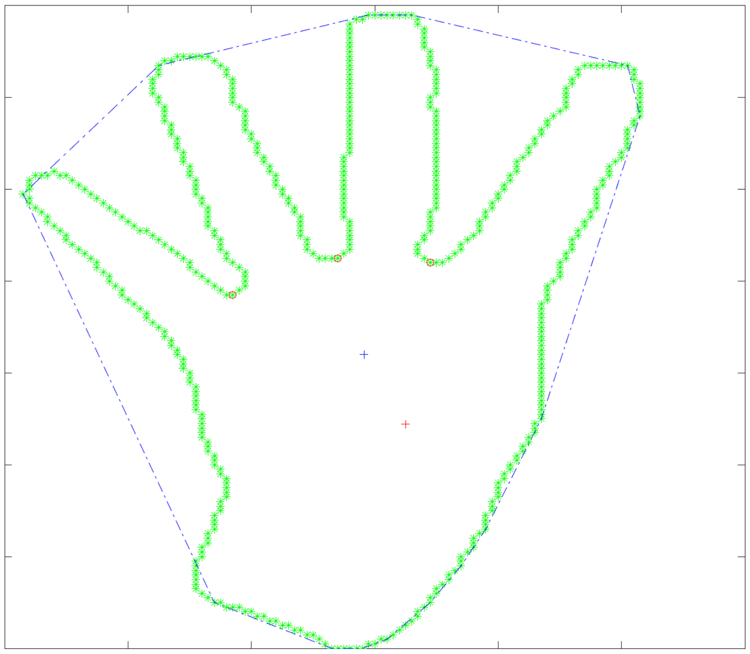


Figure 6: Failure of Gesture Test with 4 fingers

we are almost sure the algorithm will not crash due to invalid inputs.

6.2.1 Success with Gesture Recognition

When we have more fingers, the algorithm recognizes the finger number more successfully. The following is a nice example of the algorithm reading a 4-finger image.
However, the algorithm tend to fail under some circumstances too.

6.2.2 Limitations with Gesture Recognition

Skin pigmentation is the main limitation/bottleneck our algorithm has. Since it detects hands depending on the color, it depends on a lot of things like skin tone, lighting and shadow, and backgrounds. Most of the times the algorithm fails when it picks up other objects as skins, or when it fail to count part of the hand as skin. From testing, we noticed that our test result can be improved significantly if we have good background that does not have skin color at all, and if we have good lighting on the hand (so that there will not be shadows on the hand that makes the algorithm think is not skin).

7 Conclusion

Our project in robot motion using optical flow and gesture recognition was successful in ideal conditions and struggled in most other conditions that strayed away from ideal. Examples of these include lots of extraneous motion in our optical flow or palm type and lighting in our gesture recognition. Further experimentation would continue improving both of these algorithms. For optical flow, increasing the general accuracy would be important with the use of stronger and faster cameras and testing other optical flow methods, like feature detection, which could be able to detect motion better. For gesture recognition, shifting away from a pigment centered skin approach and using another approach that would be resistant to changes in the lighting would create better convex hulls for our algorithm as a whole.

8 Acknowledgements

- OpenCV's optical flow tutorial: An optical flow tutorial provided by OpenCV.
- Structure from motion lab Professor Jerod Weinman's lab on structure from motion, which served as a starting point for understanding optical flow
- FOLKI algorithm optimization for GPU usage: A potential optimization in our algorithm by using the GPU to hasten multiplication. Further experimentation could be used to speed up our algorithm.
- Lucas's 1984 doctoral paper The
- Jerod Weinman's gkern function, his help with the motion direction algorithm and the general advice and help he was able to give throughout our entire process
- Our Initial Proposal, Progress Report, and Progress Report Slides as we built off and used various sections
- Henry Walker's MyroC Libraries : Henry Walker's MyroC library, which we ulitized to create robot motion. Lisenced under Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0). http://creativecommons.org/licenses/by-nc-sa/4.0/
- Jaechul and Paul's MyroC imlpementation : Sample implementations of MyroC that we did not end up ulitizing to solve the issues with had with MyroC
- Lucas-Kanade in a Nutshell: This material provided insights and helped us better understand the mathematics behind the Lucas-Kanade method
- Literature on Hand Gestures: This was additional literature that help informed us on gesture recongition.

References

[1] Sara E. Garza-Villarreal David J. Rios-Soria, Satu E. Schaeffer. Hand-gesture recognition using computer-vision techniques. *21st International Conference on Computer Graphics, Visualization and Computer Vision*, 2013.

[2] DJ Fleet JL Barron and SS Beauchemin. Performance of optical flow technique. pages 43–77, 1994.

[3] Bruce Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. pages 121–130, 1981.

[4] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall Inc, 1998.