USING COMPUTER VISION TO TEACH NUMBER LINE

KHANG VO HUYNH and LUKE MALEK
MSCS Department, St. Olaf College, Northfield, MN 55057, U.S.A.
emails: huynh6@stolaf.edu, malek1@stolaf.edu

Abstract

With the advancement of technology, especially computer vision and artificial intelligence (CVAI) in the last decade, CVAI has played an important role in enhancing the experience of the learner through multiple instances of educational applications that have these cutting edge technology as the main component. At the same time, knowing that the process of learning about numbers (especially through the number line) will be an important step in building the foundation of mathematics for children, as part of our Senior Capstone project at St. Olaf College, we implemented a solution for helping the child getting real time feedback based on the picture inputted into our implementation. The paper will focus on the technical description as well as the benchmarked result of our solution to this specific problem.

1 Introduction

Nowadays, although children have a chance to expose themselves to many (or unlimited) sources of mathematics information, the child's development in this field vastly depends on the individual. However, there is an undoubted fact that the poor performance of children in the process of learning mathematics could possibly cause more trouble later in their life [2]. At the same time, Butterworth, Brian in 2005 pointed out that the poor performance in arithmetic skill (mathematics) could be much worse than poor literacy skill [1]. As a consequence, it is better for the children to have a good foundation before all of the mathematics information becomes "chronic, pervasive, severe, and difficult to remediate" [3].

Hence, it is important to find a way to help the children build up a good mathematical background in the early stages of their education, especially children from 4 to 7 years, so that they would be prepared for higher education like high school or even university and workplace-level mathematics. The starting point of this process will be learning about numbers and in this research paper, the number line is the tool to use.

For students working on learning the number line, getting feedback is important. Teachers often provide this feedback, but in a big classroom, teachers can't be everywhere at once. Any sort of agent capable of providing feedback as accurately as a teacher while being more available would contribute to the classroom experience.

1.1 A Brief Background of Important Algorithms

Many modern Computer Vision applications make use of deep learning technology. However, not every problem in computer vision calls for such an approach. Certain problems can be solved purely with the help of time-tested algorithms founded in relatively simple and clear mathematics. The Canny Edge algorithm and the Hough Transform algorithm are two of such processes.

The Canny Algorithm calculates a gradient vector at every pixel in the image. We then have three cases. If the magnitude of the gradient is significantly greater than that of neighboring gradients, the pixel is marked as an edge. If the gradient is small, it is marked as not-an-edge. For the pixels in the middle ground, the Canny algorithm determines whether they are an edge or not by finding patterns from the previously determined pixels.[5]

The Hough transform algorithm is best suited to run on binary images, which happens to be exactly the type of image that a Canny algorithm returns. The set of all lines through an image can be thought of as a separate 2-dimensional space, where each point in this new space (also known as the Hough Space) describes one unique line in the original image. Each

point in the Hough Space is given an accumulator variable. This accumulator variable is incremented for every edge points in the binary image which overlaps with the corresponding line. Very prominent lines in the edge image will relate to local maximas in the Hough space.[6]

2 Our Approach

2.1 The flow of the process

We seek to create a solution which can be used by children that is capable of taking in a wide variety of angles and orientations of a number line but still provide accurate readings. Given an image which features a number line, we hope to identify the locations of the marks on the number line. Once the marks are found, we determine which mark is closest to the pointer finger of a pictured hand. The approach we used to solve this problem boiled down to three processes: determine a bounding rectangle for the number line, run a mark detection algorithm, and use a hand detection algorithm. In this section, we will describe the specifics of each of these components.

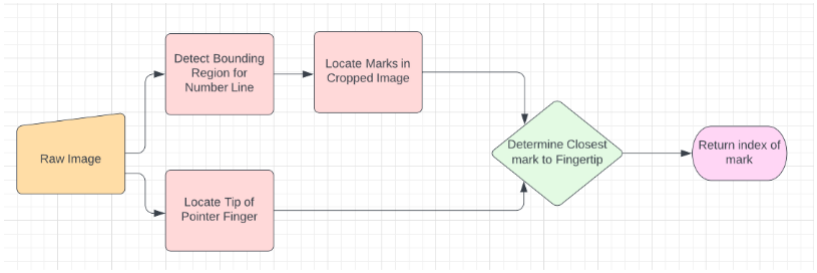

Figure 1: The algorithm

2.2 Number line detection

To find the rectangle which binds the number line, we took advantage of the distinct shape that our number lines have. Number lines have straight, parallel edges. If we can find a pair of prominent, parallel lines in the image, there's a good chance they bound a number line, or at least bind a segment of the number line. To find lines in the image, we utilize a probabilistic Hough Transform on a Canny edge image. Both of these algorithms have many parameters each, so we must determine what values will provide usable results. We hope to isolate some 10 candidate lines. So, we first run the Canny algorithm with very high parameters (`maxVal = 1000, minVal = 700`), which will yield few or no Hough lines. If we isolate fewer than 10 lines, we shrink the parameters by a factor of 0.9 and try again. There will come a point when we receive more than 10 candidate Hough lines, at which point we quit the loop, and truncate the list to 10 entries. With these 10 candidate lines, we give each of the 10-choose-2 pairs of lines a likelihood value. Parallel and close lines will receive the highest likelihood values. The pair with the highest likelihood will be kept, and returned.

With two lines found, we use simple geometric functions to capture a region of interest. We cannot assume that the entirety of an edge of the number line will be picked up by the Hough transform. More likely, some segments will be captured. Using some simple geometry, we can extend the lines to the edges of the screen. Now, the entirety of the edge of the number line will be accounted for between the pair of now-extended lines.

With the number lines which we work with, the base of the number line features a distinct, green triangle. In order for

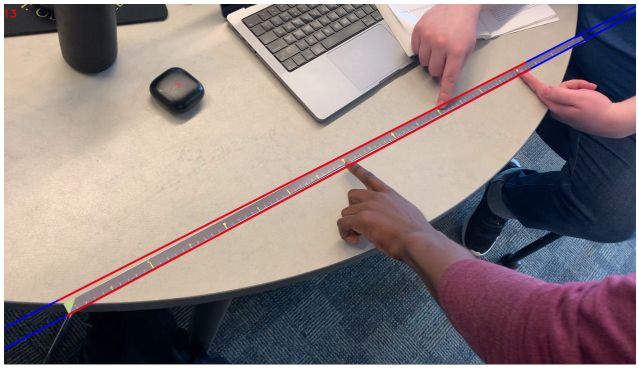

Figure 2: Bounding Line in Red - Extended Region of Interest in Blue

our mark readings to make any sense, we must know which mark is the first one. As a consequence, we must locate the green triangle. We crop the image between the lines. Then, we convert the image to an HSV image. We found that the Hue value of the green triangle is 47, and we expect any pixel in the green triangle to have a hue value rather close to 47. We use the openCV function `inRange()` to create a binary image. The pixels will have full activation if their hue value is near 47, and no activation otherwise. We can then run a contour analysis, and then approximate each contour as a polygon. If within our image is a triangle-like contour whose height is near that of the height of the region of interest, we take that to be the triangle at the base of the ruler. Our `findTriangle` function returns the coordinates of the center of the triangle, and returns -1 if no triangle was found.

At this point, we have a pair of lines and a coordinate of the center of the triangle. Last, we use additional simple geometry to extend the lines so that they reach to the provided coordinate. Now, between the lines is a rectangle, starting exactly where the number line begins. With that, we have found a bounding rectangle for the number line.

2.3 Mark detection and optimization

We crop the whole image down to the bounding rectangle of the number line, perhaps the image is of dimensions `length` $\times$ `height`. We partition the image into `length` 1-D cross-sections each of size `height`. Figure 3 shows the raw input to our solution and the result after the process. We refer to an individual cross-section as a slice. Next, we create an array to store the average pixel activation of each of each slice. Since the marks on the number line are perpendicular to the length, slices which overlap with a mark have a significantly higher average pixel activation than slices which do not overlap. This discrepancy is exaggerated even more by the coloring of the number line, where marks are bright yellow or green, whereas unmarked territory is dark gray. We then run an algorithm which returns local maxima of a sequence. The appearances of local maxima in the average pixel activation array should correlate with the appearances of marks on the number line. The function we use to find local maxima has many parameters, so we employ a strategy similar to that used when we performed a Hough transform. We determine how many marks we expect to find depending on the region's height/width ratio. Then, we run the function, first demanding that the maxima have high "prominence." If we find far fewer marks than expected, lower the prominence and try again.

Mark correction takes note of a common pattern in marks, then fills in gaps using what it learned. If a gap is roughly n times larger than the median gap size, create $n-1$ new regularly spaced marks to fill in the gap. Finally, we return a list of

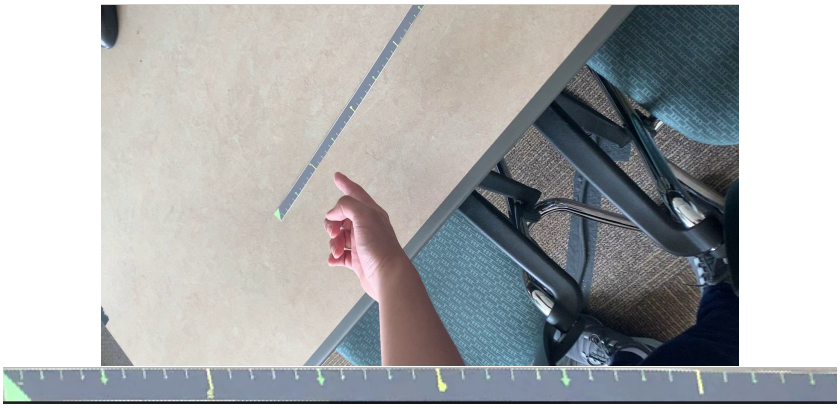

Figure 3: Input Image Above and Crop Region of Interest Below

mark coordinates.

2.4 Hand detection

2.4.1 Mediapipe - Why not?

For the hand detection (finger tip in this case), we had come to the decision of picking Mediapipe as our supporting platform. The reason is because based on Lugaresi, Camela et. al., "MediaPipe is designed for machine learning (ML) practitioners, including researchers, students, and software developers, who implement production-ready ML applications, publish code accompanying research work, and build technology prototypes. The main use case for MediaPipe is rapid prototyping of perception pipelines with inference models and other reusable components. MediaPipe also facilitates the deployment of perception technology into demos and applications on a wide variety of different hardware platforms. MediaPipe enables incremental improvements to perception pipelines through its rich configuration language and evaluation tools." [5]

2.5 How we use Mediapipe

In our solution, we used the Hands module provided by Mediapipe. The option for static image was set to true as well as set lower the minimum confidence of detection closed to 0.1. The process was simply inputting an image into the solution, then we processed the image using the hand Module provided by Mediapipe after converting it from BGR (standard for opencv) to RGB (standard for image processing). This process then results in a landmark that contains all the information about the hand that was detected. We then normalized the pixel coordinates of the finger tip which was contained inside the landmark and finally returned a two dimensional coordinate of the finger tip on the image.

3 Result

3.1 Individual Component analysis

Next, we consider the robustness of each portion of our process. From our testing, it is clear that number line detection is our weakest link. Since the rest of the steps of our process rely on having an accurate bounding rectangle, this is a substantial issue. On certain input images, the proposed bounding lines are completely unrelated to the number line. This is often due

to noisy Canny edge images making additional Hough lines appear. In practice, parallel lines in an environment are not as rare as we expected. Patterned floors or tables tend to pose an issue to our solution, as Canny edge images of patterned surfaces tend to have many edges. In turn, this creates many Hough lines.

Mark detection runs rather well, assuming that a proper bounding rectangle is provided. When the bounding region is correct, we can expect few or no marks to be missed, and for few or no marks to be included superfluously. An interesting instance occurs when the bounding rectangle captures the bottom edge of a ruler, but the upper edge is too low. In this situation, mark detection returns extra marks. The reason this happens is because mark detection looks for a certain number of marks depending on the shape of the region. A long, skinny region is expected to have more marks in it than a short, squat region. The strong performance of mark detection has use outside of this particular project. If we recontextualized the problem such that a child took a picture of the number line, then drew in a bounding rectangle by hand, we could expect strong, reliable results.

Hand detection is the strongest component since it has the support of the pretrained model from Mediapipe. However, there are still weaknesses in these modules. In order to obtain an accurate result, it requires the user to have their hand palm present on the image. Otherwise, the result will be inaccurate since the skeleton of the hand will be flipped around the y-axis.

3.2 Holistic Process Analysis

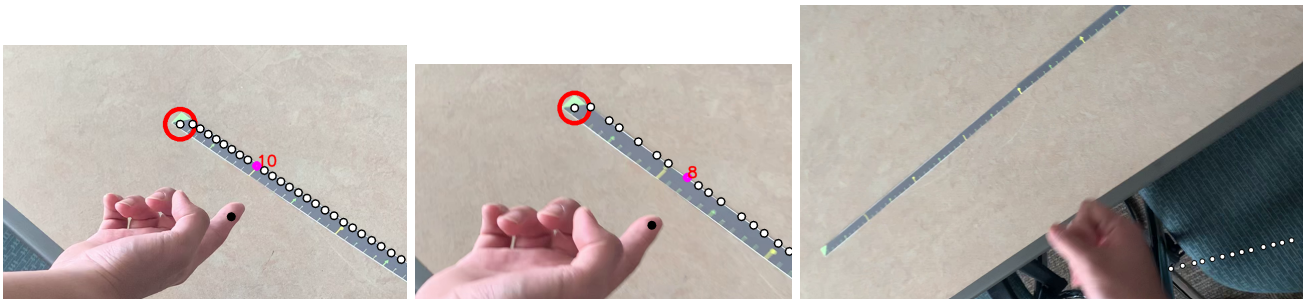

Figure 4: Strong, Average and Worse Use Case of the Process

When we combined all the components together, we were able to output the number based on the relative location between the set of marks on the number line and the location of the finger. However, the result accuracy vastly depends on the performance of all of the components. Our solution works extremely well when the image was in a decent quality. By "decent", we meant the marks could be visibly saw by human being's eyes. However, the solution will slowly worsen as the image became blurrier or there exists multiple confusing lines (confusing lines is the one that was wrongly detected but are closed to the actually number lines). Finally, the solution result will be completely inaccurate when the noise is enormous. Some of instances of noise that could cause the implementation to output completely off result could be the background of the image (line a striped floor), any object that have their shapes look like a number line, etc.

4 Conclusion

In this paper, we propose a solution to our specific problem about using computer vision to enhance students' experience in learning about number lines. By combining smaller techniques like Canny, Hough Lines, etc. the solution is able to output the pointed number by the user. However, there is still more work in the future to improve the robustness of each components so that we would be able to obtain a better result.

References

[1] Butterworth, B. (2005). The development of arithmetical abilities. Journal of Child Psychology and Psychiatry, 46(1), 3-18.

[2] Lannin, J., Garderen, D., Switzer, M., Buchheiser, K. & Jackson, C (2013). The mathematical development in number and operation of struggling first graders. Investigations in Mathematics Learning, 6(2).

[3] Fuchs, L. S. (2005). Prevention research in mathematics: Improving outcomes, building identification models, and understanding disability. Journal of Learning Disabilities, 38, 350-352.

[4] Lugaresi, C. et. al. MediaPipe: A framework for building perception pipelines.

[5] Ding, L., & Goshtasby, A. (2001). On the canny edge detector. Pattern Recognition, 34(3), 721–725. https://doi.org/10.1016/s0031-3203(00)00023-6

[6] Ballard, D. H. (1981). Generalizing the hough transform to detect arbitrary shapes. Pattern Recognition, 13(2), 111–122. https://doi.org/10.1016/0031-3203(81)90009-1