Turning a Digital Camera Into an Adaptable Touchscreen

M.Drudis

Nowadays it's widely common to be around electronic devices with a processor. From smartphones to laptops, it's common for those devices to have a camera. Some of them use a keyboard to interact, while others have a touchscreen of a defined size. In this paper we want to explore if it would be possible to use a camera to interact with any device by creating a touchscreen of any wanted shape and size.
The goal is to be able to make a program that can be downloaded and calibrated by anyone that recognizes a pointer such as a finger or a pencil and gives back its position on its environment. Let's say over a piece of paper.
It could have many applications such as tracking your feet in a game or building a customized keyboard that one could draw on a piece of paper. However in for this paper we will stick to a board where one can draw in real-time by moving it's finger.
The approach used will be recognizing the tip of the pointer on the screen, and then convert those coordinates to it's real position by using a simple pinhole model.
An important point of this project is also to be user friendly. In order to do that we will be using Tkinter to make a graphic interface.

***Index Terms*—Computer interface, Finger recognition, Pinhole model, Image based touchpad.**

I. INTRODUCTION

THIS paper intends do implement an algorithm in python using openCV and Tkinter in order to make a pointer tracking system and later on to transform those coordinates into a rectangular shaped board where a dot will be displayed. We aim for a tracking system with an accuracy of about ten percent or better. We also want to explore the Tkinter library and try to figure out a way so that anyone can configure it's own setup to work.

II. SETUP

A. Physical setup

In order to be able to tackle this problem, we need to define how the camera must be set up. In this case, the camera will be set as close to the ground as possible and almost horizontally slightly tilted to the ground. The goal is to point at the finger sideways so that we can find it's tip. Something that will make the process of recognizing the finger easier, is to chose a flat and evenly lighted background without any strong lights producing harsh shadows on it. In our case (see fig. 1) we chose to put the camera in front of a piece of paper. The hole setup is inside a room with artificial light so that the illumination doesn't change over time. We have also printed some dots on the piece of paper to help with the calibration process in the future.

B. Segmentation of the image

The next step will be to divide the image in different parts. The first thing will be to identify where the so called "board" is. This will be the part of the image that we will use and corresponding to where we will place our finger (or pointer). It will be delimited by two points A and B at the top of it -we will talk about how to place those points in the interface section-.

This paper is made as an assignment for "Processament de Imatge i Visió Artificial" in Universitat de Barcelona and submited the 11^{th} of july 2020.

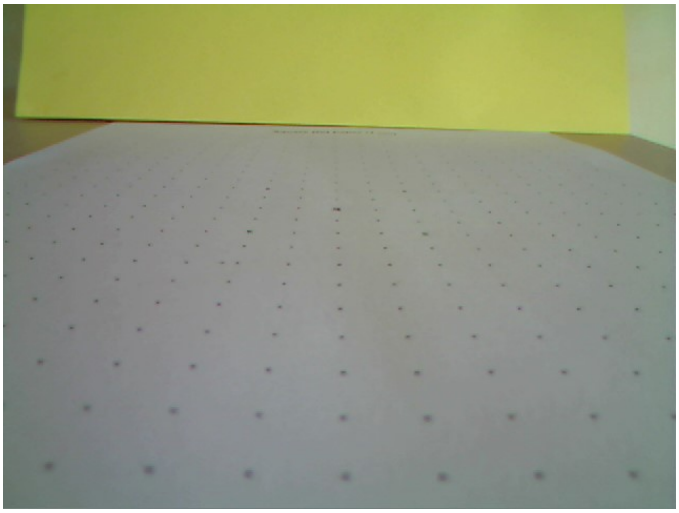

Fig. 1: Picture of a blue pen and a finger used as pointers and a white cross pointing where the program recognizes it's tip. In the pen image we can also see that the pixels that are not perceived as part of the pointer are displayed in black.

Next we will create a shallow rectangular shape "border" at top of the "board" that we will use to find where the finger is approximately in the x axis.

Once we know between which 2 values of x our finger is we will call that region of the board "VerticalFingerRegion" and it will be used to determine the position of the tip of the finger.

III. FINGER RECOGNITION

A. Finding the finger

Assuming our setup is correct, the first thing to do is a first approximation on which region contains the finger. In order to do so, we analyze the "border". First of all, we divide horizontally this section into multiple portions. Now, we take the average of color in every one of these portions. Note here that we could have done it with a 1 pixel row, but we decided

to give a height to this segment in case there are any artifacts in the image.

Now, we got a one dimensional array of colors and we assign a weight to each channel in order to calculate the "weighted function". This function will be further discussed in "Color calibration", but basically will allow us to see between which portions the change in color is more likely to indicate one of the borders of the finger. Once we get the index of those portions, knowing it's width, we can know between which x-coordinates our finger is contained.

Once again, we have divided the "border" in segments instead of treating every pixel individually to avoid finding false pikes created by artifacts of shadows. The size of those portions is crucial, they have to be narrow enough so that we can differentiate the borders of the finger, but wide enough to avoid those false pikes. We got our value at about a 5^{th} of the finger wideness. Note that in this step it's not important to get a precise position of the finger as the x position will be determined in the next step, but rather to be sure that there are no false positives at detecting the finger and that all of it is inside our region.

While we are at it, the function that computes this 2 coordinates also returns a boolean stating whether we have detected a finger or not. This Boolean corresponds to (Is the maximum increment in the weight function superior to a given threshold?) and will be used in the future to draw a dot on the blackboard or not.

B. Isolating the finger

Once we got our "VerticalFingerRegion", we want to find the exact position of the tip of the finger. How can we differentiate between a finger and the background?

Assuming that the finger is homogeneous enough compared to the background, we will take a color sample of the center of the finger in the "border" and assume that the rest of the finger has a similar color. We will start scanning from the top row and find all the pixels close enough to the sample color. This process will be discussed further on.

If the current layer has any selected pixels then we go to the next one until we find a layer with no pixels belonging to the finger. If in the next N layers (with N to be determined) we don't find any pixel belonging to the finger, we set the last layer as the y-coordinate of the finger and the average position of the pixels in that layer as the x-coordinate. By doing a vertical scan instead of treating the whole section at once we avoid potential false positives on detecting the finger.

Note that there are more precise ways of finding the borders of an object, but this is fast to compute and a simple solution that works well enough in our case.

C. Previous approach

It's worth commenting the first approach that we took for choosing the finger position and why the one that we just proposed is better.

At first, we chose the x position of the finger to be the average between it's borders in the "border" segment. After that, we would take the central pixels of "VerticalFingerRegion",

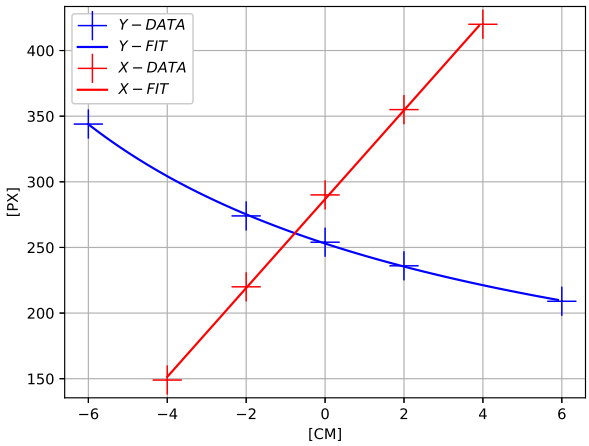

Fig. 2: Example of the data fitting the pinhole model. Note that both the x and y data has been taken over lines that are straights both in the physical board and in the image captured. That would be $x = 0$ for the $Y - DATA$ and $y = const$ for the $X - DATA$.

thus leaving the 20% of pixels to the right and to the left out and use that new segment to find the y-coordinate using the same method we used to find the borders of the finger but with only one maximum.

This approach has some disadvantages. The most important one, and the reason we changed it, is because when the finger is not completely vertical, it's tip wouldn't be in the region where we are looking for it, and we would get a slightly wrong coordinate in the x axis, but a completely wrong value in the y axis. This problem is increased by the fact that, as will be seen further, the y-coordinate transformation is highly sensitive to changes in the x-coordinate on the screen.

IV. Finding coordinates

First of all, let's introduce some some notation in order to simplify the explanation of this section. We will be working with 3 spaces. First of all, we will be treating with the coordinate of the tip of the pointer in the image captured by the camera. We will refer to this position as $(x, y)[PX]$. The next step will be to transform those coordinates in order to fit the real position of the pointer in the physical board. This position will be referred as $(x, y)[CM]$. Finally, we will want to visualize the real life coordinates back on the screen in order to make a non-deformed blackboard. The new coordinates will be linear with respect to $(x, y)[CM]$ and be called (x_B, y_B).

A. Pinhole model

The pinhole model is a simple geometrical model on how the image is captured by a camera [2]. We assume, in this case, that the sensor is in the (y, z) plane while the surface we are interacting with is in (x, z). Then, every point on the surface is projected on the sensor by a straight line intersecting a given point O, representing the focal point of the camera.

This results on a simple linear equation for x as the x-axis both for the physical board and the sensor are parallel:

$$x[CM] - x_0[CM] = b(x[PX] - x_0[PX])$$

With $x_0[CM], x_0[PX]$ and b , the translations in position we want to do and b the proportionality coefficient respective to the difference of distances with O between one plane and the other.

On the other hand, as the y-axis are perpendicular we get a slightly more complicated equation:

$$y[CM] - y_0[CM] = \frac{b}{(y[PX] - y_0[PX])}$$

With similar parameters to the previous equation.

Note that until now we have only solved the problem when x and y transformations don't depend on the other coordinate. However, as one can see in fig. 1 the x position depends linearly on y, as the further away from the camera we are the smaller x distances get on screen.

B. Blackboard coordinates

Once we got $(x, y)[CM]$ we want to be able to display it back in the screen, this time with the same proportions as the physical board. In order to do so let's imagine that our physical board is defined by $X_{min} \leq x[CM] \leq X_{max}$ and $Y_{min} \leq y[CM] \leq Y_{max}$. In that case we would get one of it's dimensions superior to the other. Lets say that L is that dimension, then for a given point $(x, y)[CM] = (a, b)$ we would get:

$$(xB, yB) = \left(\frac{a - x_{min}}{L}, \frac{b - y_{min}}{L}\right)$$

Where the limits of the physical board will be determined during the calibration process.

V. Calibration Process

A. Color Calibration

The most sensible part of this project is recognizing the finger and it's fingertip. As previously described this process has two parts.

Firstly, we want to find the borders of the finger in the "border" region. The procedure has already been described and it consisted on finding the 2 maximums of a weight function. Let's discuss this function further. On a first approach we were only taking one of the RGB channels into account to calculate the increment function. It worked well enough but was too sensitive to the shadows projected by the finger. The solution to this problem was to create a weighted average of the increment over each channel, including the saturation and value channels, and tweak the weights so that the program finds the borders with as few errors as possible. In order to visualize which channels are the more affected at those points we created a debug tool activated by pressing the "Return" key that allows to visualize the increment functions for each channel.

Note that the fact that we introduced the saturation channel into the weighted average is because it's not so affected by strong lights and is therefore a good candidate to have a high weight.

Secondly, we have to identify which pixels belong to the finger. In order to do that we said that we wanted to compare them to the color sample taken at the middle of the finger. After that, for each channel we set a threshold value and check if the difference between the difference between the sample color and this pixel are superior to it or not. It's enough for one of the thresholds to not be respected in order for the pixel to be excluded.

B. Position Calibration

In order to determine the position of pointer in the physical board, we could make precise measures of the geometry of the setup [1]. Instead, we will take a different approach so that any user can do it without having to touch the code. For each coordinate we will have a function with some parameters to be fitted. Later on we will optimize those parameters to fit some sample points.

This approach has the added benefit to be more resilient to human errors. For example if we have 2 users and one of them points with the tip of it's nail while the other uses it's fingerprint this approach will correct for it as long as they do it consistently.

All the user has to do is input some data of corresponding points in the physical board and the captured image as will be explained in the interface section. After adding a reasonable amount of points the parameters will be optimized and the position of the finger will be computed in real time and drawn on the screen.

C. Paraboloid approach

Despite the pinhole model could give very accurate results as seen in fig. 2, it has some key disadvantages. On one hand, this approach is very sensible to false calibration input, or tilted cameras. On the other hand, and most important of all, it can produce vertical assymptotes (and therefore NAN errors) if the calibration is not done right or if the initial guessing values are not the right ones (in particular if $y_0[PX]$ is too high or to low depending on the orientation of the camera).

In order to guarantee a user friendly experience we will fit the data with polynomials. A 2 dimensional paraboloid will be less sensible to a bad calibration process, will fit the points even if the camera is tilted sideways and can easily solve the y dependence problem in order to find the x coordinate.

Nonetheless, as we don't want to have too many parameters because that would mean having to take more calibration points in order to fit the model, we will take an educated guess on the shape of the paraboloid based on fig. 2.

X[CM] is already linear with x[PX]. Therefore we will only add a linear dependence on y and a mixed term xy:

$$x[CM] = a + bx[PX] + cx[PX]y[PX] + dy[PX]$$

y[CM] seems like a parabola on it's own. We will nonetheless introduce a midex term xy to give it some x dependance in case the camera is tilted or any other possible problem:

$$y[CM] = a + by[PX] + c(y[PX])^2 + dx[PX]y[PX]$$

VI. Interface

The interface contains 2 canvas with 2 different images. One is the image captured by the camera modified to visualize the detected $(x, y)[PX]$ coordinates obtained. It will also contain 2 points A and B represented by blue dots. They will indicate the limits of the "board" and therefore will determine the "border" region where the pointer will be detected at first. Those 2 points can be dragged with a left and right click on the mouse respectively and if the camera is leveled they should be almost at the same height.

Next to it we have the blackboard where at every tick of the program we will draw a dot of the color of the pointer (the color chosen will be the same used to find the pixels belonging to it). We chose to make this canvas square, but depending on the shape of the physical board there will be some unused space.

Under those we will find the calibration controls. From the left to the right we have the control to the thresholds to find the tip of the pointer, the weight used to find the borders of the pointer, a threshold to identify whether there is a pointer on screen or not and a calibration interface. In order to use the calibration interface one has to put the coordinates in the physical board at the same time as point to that point with the finger and then click "Calibrate". When initiating the program there are some default values for calibration points. If one wants to get rid of them just press the button "Reset".

Finally, on top of the screen we can see the coordinates on the screen on real time and a smoothed version of the coordinates on the board. The coordinates are smoothed out using an exponential moving average which purpose is to make the coordinates easier to as it reduces quick oscillations on the data output and also to smooth out the lines drawn.

VII. Results

The results of this project can be evaluated though 2 criteria: quantitative error on the position and qualitative evaluation of the shapes drawn with the program.

First, lets comment the quantitative data obtained:

At first glance the errors on the values obtained by the program are not too high, although they seem to be higher as we approach the horizontal limits of our board. To be more precise the error on the x-axis we have an average error of 2.36% of the total tested range (from $-3cm$ to $3cm$) and a dispersion of 1.95% and in the y-axis we have 1.52% average and 0.80% as dispersion.

As much as drawing capabilities, we can see in figure fig. 3 that horizontal and vertical lines are parallel between them (most of it's error is probably associated to having drawn them by hand) and proportions are well conserved.

A[CM]	B[CM]	x[CM]	y[CM]
2	2	2.06	2.13
2	-2	2.27	-2.16
-2	-2	-2.03	-2.15
-2	2	-1.98	2.1
0	0	0.1	0.06
3	0	3.32	0.09
0	5	0.2	5.04
-3	0	-3.24	0.03
0	-5	0.03	-4.94

TABLE I: A[CM] and B[CM] represent the points where the pointer is intended to be and x[CM] and y[CM] the result obtained by the program. The data corresponds to a calibration done with 8 points that are different to the ones tested on this table.

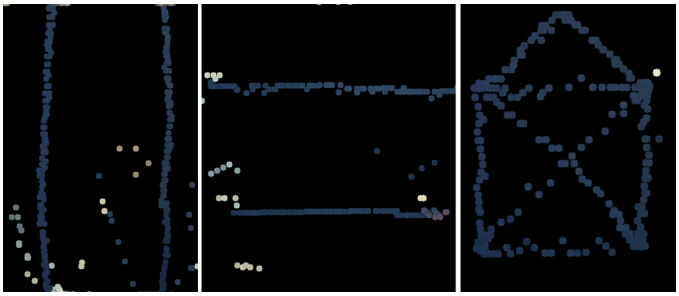

Fig. 3: 3 examples of Drawings made by hand. On the 2 first we can see how parallel lines appear on the program and the third one is to show the conservation of proportions. Note that the coordinates haven't been smoothed out with an exponential moving average in this case.

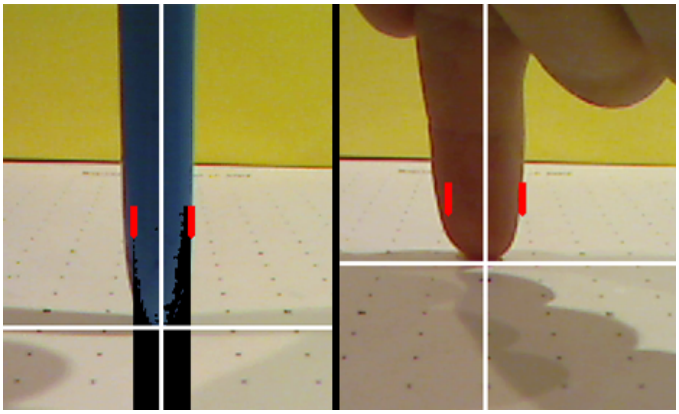

Fig. 4: Picture of a blue pen and a finger used as pointers and a white cross pointing where the program recognizes it's tim. In the pen image we can also see that the pixels that are not perceived as part of the pointer are displayed in black.

VIII. Conclusion

In conclusion, our goal has been reached as the error is way inferior to 10% which is what we were aiming for. The user interface is quite intuitive as well, and the whole program has a good latency.

One thing to correct would be the obtention of coordinates on the x axis. Maybe a more geometrically complex model should be considered in order to account for the y dependency of the x coordinate, but the extension of this paper didn't allow to explore further.

It would be interesting as well to a well calibrated pinhole model would compare to the paraboloid approach and to try to implement some other applications for the tracking system.

References

[1] Zhang, Longyu Saboune, Jamal El Saddik, Abdulmotaleb. (2014). Transforming a Regular Screen Into a Touch Screen Using a Single Webcam. Journal of Display Technology. 10. 1-1. 10.1109/JDT.2014.2312488.

[2] Sturm P. (2014) Pinhole Camera Model. In: Ikeuchi K. (eds) Computer Vision. Springer, Boston, MA