Video Joystick

Felipe Vaiano Calderan, Silvio de Souza Neves Neto

Federal University Of São Paulo (UNIFESP)

***Abstract*—Games have been around for around 60 years and input methods have evolved together with the games themselves. In the last decade, a big boom of movement detection based controllers, with Nintendo Wii, Xbox Kinect, PlayStation Move/Eye and others. Here, we document, briefly, the relevant history of input devices that led up to where we are now and we propose an input method that is completely controlled by image processing, so that anyone with a sheet of paper and a webcam can build their own joystick on the spot, without having to resort to the computer keyboard or an external joystick. We also present an analysis on the controller efficiency through video inspection, and also user experience, by collecting opinions from different people who tested the input method.**

***Index Terms*—Image processing, computer vision, video, joystick, games**

I. Introduction

THE main point of video games is to create an interactive environment where the player can, through an input method, change the state of what is being displayed in a monitor device.

The idea of creating an interesting input device have existed since the very beginning of video game history with William Higinbotham's *Tennis for Two* [1], 1958. Observing Figure 1, it's possible to see that even in those days a proper controller was created so that the game play factor was more enjoyable: an analog dial to move the player and a push button to perform actions.

Fig. 1. Tennis for Two modern recreation - Windell Oskay from Sunnyvale, CA, USA, CC BY 2.0 <https://creativecommons.org/licenses/by/2.0>, via Wikimedia Commons

Input methods are, in fact, so relevant they precede the existence of video games. The first *light guns* appeared in 1930, enabled by, then, recent developments in light sensing vacuum tubes. It wasn't long after that *light guns* started being used in arcade games [2]. This input method was later used in very famous consoles, like the Nintendo Entertainment System (as seen in Figure 2), Sega Master System, Atari, Inc., Sony PlayStation, Sega Genesis and others.

Fig. 2. NES Zapper. By Evan-Amos - Own work, Public Domain <https://commons.wikimedia.org/w/index.php?curid=50797667>

The shape of the joysticks can also determine the preferred kinds of games that will be played using them. Steering wheels are made for driving games, fight sticks are typically used for fighting games or classic arcade games that don't require analog input and flight sticks are made to play flight simulators/arcade games. The opposite is also true: there are games that determined the kind of preferred input methods that will be used, because they were made with a specific controller in mind.

Some input methods are more general, like keyboard and mouse (which are typically used to play games using a personal computer) and general-purpose joysticks, like Sony's Dualshocks/Dualsense or the Xbox One controller. They happen to be or are made to be effective in a wide variety of games, which is a desirable feature to have, considering they usually ship with the console and are the first experience the player will have interacting with games in that console.

As technology evolved, the games evolved and so did the input methods. The video game industry became more aware that ergonomics [3] are a considerable selling point to a console, therefore, better and more customizable controllers began being produced. Recently, motion based inputs, that uses computer vision (which is a subset of image processing) and advanced camera techniques [4] became popular among the players.

This big boom started with Nintendo Wii's *Wiimote*, launched together with Nintendo Wii in 2006. Nintendo Wii's bet wasn't on powerful graphics, but rather on providing a novel experience to its player base. The movement-based controllers payed off, since Nintendo Wii sold over 100 million units [5] (making it the fourth most sold console of all time). Sony PlayStation 3 and Xbox 360 eventually caught up on the technology of movement detection with the launch of PlayStation Move/Eye and Kinect (for Xbox), both in 2010), respectively. Although the technologies involved are different,

they accomplish similar objectives in respect to game play factors.

Kinect (Figure 3), in special, uses a vision-based human activity recognition system [6] based on a 3D depth camera. It turned out to be applicable not only in gaming activities, but also in many other fields [7], [4]. This kind of system is technologically very advanced and require special equipment (like the Kinect) to be executed properly, which differs from the approach that we take to produce a new input method based only on simple cameras.

Fig. 3. Kinect. By Evan-Amos - Own work, Public Domain <https://commons.wikimedia.org/w/index.php?curid=33217678>

Although the lens (and its capabilities) through which the computer will see the world are defining factors on what kind of computer vision systems are convenient to be built upon them, a software to process the incoming data is also extremely important, since it contains the logic that dictates what will be done with what the cameras/sensors are seeing. There are several studies that focus on identifying gestures, some use 3D data like in [8] and others use novel methods, like *Combined Local-Global* (CLG) optic flow [9]. It's also typical to use Machine Learning, specifically Neural Networks though Deep Learning to identify patterns and infer information about the data stream [10].

Sony's approach to gesture detection on PlayStation 3 is quite different: it is performed using 2 separate peripherals (and of course, the PlayStation it self for the bulk processing). The PlayStation Move is a motion controller that operates using an *Inertial Measurement Unit* (IMU) [11], which contains an *Accelerometer* (measures proper acceleration [12]), an *Angular Rate Sensor* (has a very similar purpose of *Gyroscopes*) and a *Magnetometer* (Calibrate the orientation in relation to the Earth's magnetic field). In general, PlayStation Move works in a very similar way to Nintendo's *Wiimote*. In the other hand, the PlayStation Eye detects the general ambient color to define the Move's LED ball color. The color should be very contrasting to the ambient in which the person using the Move is in, since PlayStation Eye will use the size of the colored ball to calculate the distance to the person and also track the position, improving gestures calculations and generalizing the different situations/ambient supported by the PlayStation vision-based human activity recognition system.

The controller proposed in this article doesn't try to recognize human gestures and attacks a different kind of games: all the input methods cited above are targeted to games where the person moves more than just their hands, whereas our controller tries to replace conventional joysticks using computer vision techniques (hence the name Video Joystick), therefore is usable in general purposes, like Sony Dualshocks/Dualsense. Create an easy-to-use system without special hardware and that fits most kinds of games (specially those that require fewer inputs, like platformers), while being completely Free and Open Source Software is the biggest motivation for this project.

II. Objectives

THE objective of this project is to create a compute vision system capable of analyzing a video feed coming from the computer's webcam or other video capturing device[1] and look for a sheet of paper with buttons drawn in order to interpret them as a controller. Once the paper has been properly identified, the computer should monitor the buttons to verify if they weren't obstructed by something. If there's an obstruction, it probably means that the person using the controller is pressing a button.

Figure 4 shows an illustrative diagram of how the system is built in a "hardware" perspective. After that, an enumeration following the marks in the figure will explain what each part is meant to be in the system.

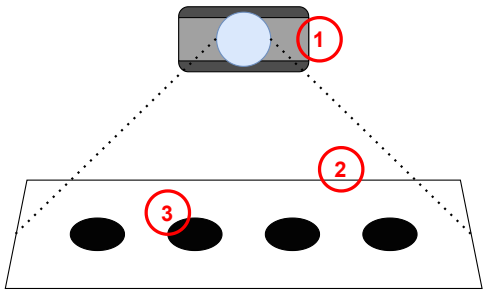

Fig. 4. Camera and paper diagram

1) This is the webcam/video capturing device. It should aim at the sheet of paper, ideally in a top-down view, so that shadows and other kinds of obstructions/distractions are minimized. To avoid unwanted shadows in the paper, it's also recommended that the strongest source of light comes from the direction against the person using the controller.
2) This is the sheet of paper. It should be a blank sheet of paper with buttons printed. The colors of the buttons are yet to be decided, but it's very important that there's enough contrast between the paper color and buttons, so things like light yellow buttons should be avoided
3) This is a button. Initially, we propose a simple system that can deal with 4 circular buttons (see in subsction III-I why), but this can be modified if needed. The quantity 4 comes from the perspective on how many old platform games work: they use two buttons to move the character left and right, one to jump and another to perform a miscellaneous action/pause the game. Ideally,

[1] as long as it's a valid video capturing device according to Linux definition of a valid device, that is, correctly exposed in /dev/videoX where X is the video capturing device.

all the buttons should be the same size, but if it's not the case, small changes to the source code or a more relaxed search for contours features (discussed in subsection III-G) should suffice.

We also provide a calibration tool to better configure the algorithm's parameters to fit the ambient lighting, distance and angle between the camera and the sheet of paper and the rotation of the paper. This calibration helps removing the noise caused by the shadows or poor illumination, crop the video to feed the computer only the necessary information (improving both speed and quality).

To detect the buttons, image segmentation techniques are applied to the video stream, so that the background ambient, sheet of paper, buttons and obstructions are properly identified and classified by the computer vision system. This makes possible the creation of a logic that, given the incoming data, tells which action the person using the Video Joystick is trying to perform.

Finally, the project is built using open source software. This is a very important factor to our project, since it's of utmost importance that Video Joystick's hardware can be physically built by anyone who has access to a sheet of paper and any way to draw on this paper (preferably a computer printer, for precision reasons), and any kind of computer that is able to run Python and has an integrated/dedicated camera. Whether it'll be used to play games, operate a computer, embedded device or for educational purposes is up to the person using the controller.

III. Methods

HERE are described all the frame image processing steps in order to build meaningful data to evaluate button presses. Each subsection's result (except for the subsection III-I) can be seen in Figure 6.

A. Matrix representation of an image

There are different ways to represent an image, but the usual way is to use matrices. For gray scale image, a simple two-dimensional matrix is enough to encode perfectly all the usual 256 values of gray. Take the left image depicted in Figure 5 for an example.

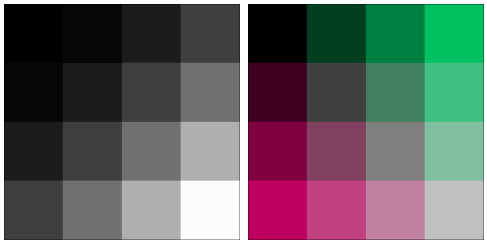

Fig. 5. a) 4x4 gray scale image; b) 4x4 colorful RGB image

It can be represented by the matrix 1:

$$gray = \begin{bmatrix} 0 & 7 & 28 & 63 \\ 7 & 28 & 63 & 112 \\ 28 & 63 & 112 & 175 \\ 63 & 112 & 175 & 252 \end{bmatrix} \quad (1)$$

To represent a colored image, multiple channels are needed. If the image is given by 3 channels: red, green and blue; it can be represented by 3 distinct matrices or by just one matrix where each element is a 3 element vector [13]. The last one is the representation of choice, since it's the way OpenCV[2] (the most important library in this project) loads images. Matrix 2 shows how the right image in Figure 5 is represented.

$$\text{color} = \begin{bmatrix} [0,0,0] & [0,64,32] & [0,128,64] & [0,192,96] \\ [64,0,32] & [64,64,64] & [64,128,96] & [64,192,128] \\ [128,0,64] & [128,64,96] & [128,128,128] & [128,192,160] \\ [192,0,96] & [192,64,128] & [192,128,160] & [192,192,192] \end{bmatrix} \quad (2)$$

If the image had an extra channel for alpha values, it would need 4 element vectors instead.

B. Convert RGB to gray scale

The most usual method to convert a RGB image to gray scale is to average the elements for each 3 element vector in the matrix:

$$aG(x,y) = \left\lfloor \frac{r_{(x,y)} + g_{(x,y)} + b_{(x,y)}}{3} \right\rfloor, \ \forall (x,y) \in img \quad (3)$$

But Equation 3 is not the method being used by Video Joystick. The one in use is shown in Equation 4, which is a weighted version of the average [14]:

$$wG(x,y) = \left\lfloor \frac{0.299 r_{(x,y)} + 0.587 g_{(x,y)} + 0.114 b_{(x,y)}}{3} \right\rfloor \quad (4)$$

and the reason this version is being used is because this is the standard on OpenCV.

C. Reducing noise with blur

Blur is an important filter to be applied here, since it reduces the amount of noise (due to primarily low illumination levels) in the video stream. An image with noise generates various micro-contours and this has very bad side-effects in the step III-G, since it uses statistical techniques to identify the correct feature ranges of the buttons.

This kind of filters are applied to the image through a convolution process. The generic formula for a convolution process is given by Equation 5.

$$g(x,y) = \omega f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy) \quad (5)$$

where $g(x,y)$ is the filtered image, $f(x,y)$ is the original image and ω is the kernel. If the filter is an average blur [15], ω is given by the matrix 6:

$$a\_blur = \frac{1}{N^2} \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & \cdots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \quad (6)$$

where the dimensions of a_blur (given by N) defines a and b for Equation 5.

[2]https://opencv.org/

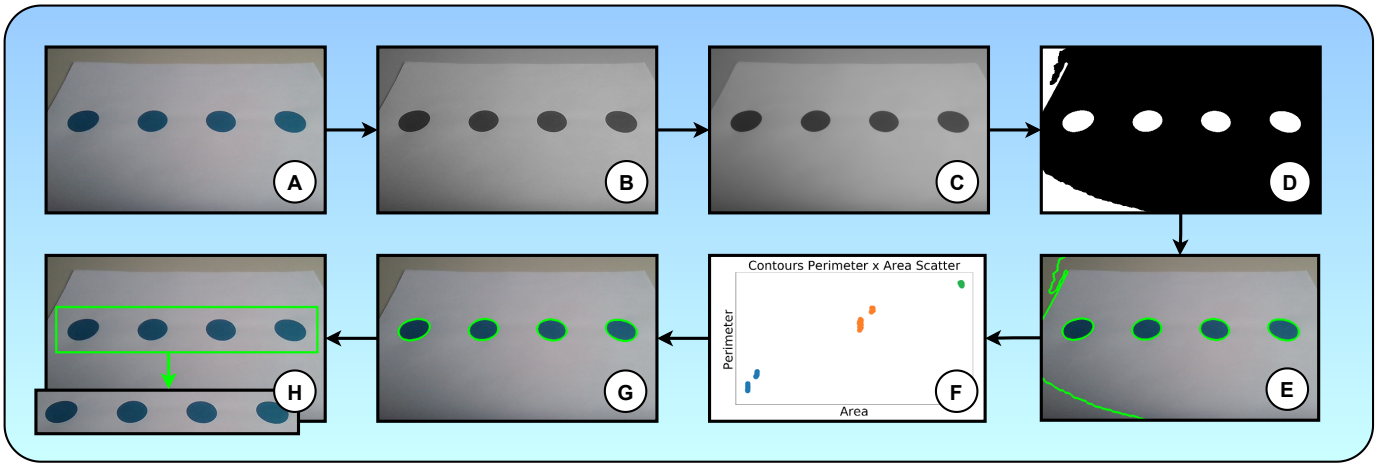

Fig. 6. Video Joystick's image processing flowchart. It's important to notice that the transformations are sequential. A) frame shown after transforming the raw data into a matrix representation of an image; B) RGB to gray scale conversion; C) noise reduction through the application of a median blur kernel; D) automatic image thresholding; E) contour finding based on the given threshold information; F) features area and perimeter extracted from each contour and plotted; G) automatic button identification enabled by mathematical analysis of the extracted features; H) automatic video cropping through the use of a bounding rectangle.

Since the focus is removing micro-contours in a thresholded environment (which means it's only black and white), a parallel can be drawn with removing salt-and-pepper noise, which is typically done by applying a median blur filter [15]. This kernel is given by the Equation 7:

$$m\_blur(x,y) = median(neighbors(x,y)) \tag{7}$$

where $neighbors(x,y)$ is, typically, a square of odd dimensions. In this square, the element (x,y) is the center and $(x+dx, y+dy)$ are the neighbors.

D. Image thresholding

Image thresholding is a technique to transform a gray scale image into a binary image, which are images that contain only black and white pixels. Which pixels will become fully white and which will become fully black depends on a threshold value (if a pixel has a value above the threshold, it'll white, otherwise it'll be black), hence the term *image thresholding*.

In many applications of image processing, the brightness of the items in the foreground differ from the ones in the background, so thresholding is an excellent choice of algorithm to segment the important elements from the unimportant ones [16]. This is specially true for the controller that we propose here, as it is composed of a white sheet of paper and dark-colored buttons (the exact color doesn't matter).

There are many thresholding techniques, but the most basic one is given by the Equation 8:

$$dst(x,y) = \begin{cases} 255 & if\ src(x,y) > thresh \\ 0 & otherwise \end{cases} \tag{8}$$

where $thresh$ is the threshold value. The inverted version of this binarization algorithm is shown in Equation 9:

$$dst(x,y) = \begin{cases} 0 & if\ src(x,y) > thresh \\ 255 & otherwise \end{cases} \tag{9}$$

which is the version used in Video Joystick (as seen in Figure 6 item *D*) both because it looks distinct from the non-thresholded images and because it follows OpenCV's contour-finding conventions[3].

One big problem with the thresholding formulas 8 and 9 is that they need the $thresh$ parameter specified, so there must be a way to know it beforehand, otherwise the user would have to set it manually and this is not ideal. To solve this problem, we combined the inverse binarization with Otsu's algorithm (or Otsu's method) [17]. This algorithm determines the $thresh$ value by minimizing the intra-class intensity variance (therefore maximizing inter-class variance). In other words, it finds t that minimizes Equation 10

$$\sigma_w^2(t) = q_1(t)\sigma_1^2 + q2(t)\sigma_2^2(t) \tag{10}$$

where

$$q_1(t) = \sum_{i=1}^{t} P(i);\quad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)};\quad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i-\mu_1(t)]^2 \frac{P(i)}{q_1(t)};\quad \sigma_2^2(t) = \sum_{i=t+1}^{I} [i-\mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

E. Contour finding

Contours are curves joining continuous points (along their boundaries). As stated previously, OpenCV's implementation considers objects in the foreground as white and the background as black, so the inverted thresholded image is an ideal scenario for contour finding.

The technique to identify contours need 3 distinct information: image that needs its contours identified, retrieval mode and contour approximation method.

[3]https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html

Retrieval mode is how the hierarchy of contours is going to be built. The hierarchy can be a flat list (which is the same as no hierarchy at all), a tree (that creates a full hierarchy of nested contours) and other different methods. Since the controller's system doesn't need to remember which contour is a child of which other contour, there's no need to spend the extra computational cost of building and operating over a tree, therefore, it makes more sense to use a flat list.

Approximation methods generate compressed or simplified versions of the actual contours present in the image [18]. Although they can be useful when a large amount of contours are present, they aren't necessary in this project, furthermore, there are reasons not to use approximations in this case: it's easier to get better/more precise features out of the contours found.

F. Contour's features extraction

Having identified the contours, it's now time to extract their features. The first feature to be found is the area, which is calculated by the Green's Theorem, described in Equation 11:

$$area = \oint_C (L\,dx + M\,dy) = \iint_D \left(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\right) dxdy \quad (11)$$

where C is a positively oriented, piece-wise smooth and simple closed curve in a plane; D is the region that is bounded by C; L and M are functions of (x, y) well defined on a open region that contains D and that have continuous partial derivatives. Also, the path of integration along C is counterclockwise.

The perimeter is another important feature that needs to be calculated. Since the video stream after thresholding is applied can be represented as a 2D matrix, the formula to calculate the arc length of a curve on a plane suffice. The formula is given by the Equation 12 [19]:

$$arclength = \int_a^b \sqrt{f'(t)^2 + g'(t)^2}dt = \int_a^b \|\vec{r}\|dt \quad (12)$$

where the smooth curve C is defined by

$$\vec{r}(t) = f(t)\hat{i} + g(t)\hat{j}, \quad a \leq t \leq b$$

Both the area and the perimeter are used in the automatic button identification, as described in subsection III-G. The next feature to be extracted are the moments of the contour. Equation 13 shows how the spatial moments are computed.

$$m_{ij} = \sum_{x,y} \left[array(x, y) \cdot x^j \cdot y^i\right] \quad (13)$$

and with m_{ij} calculated, it's possible to find the center of mass, also known as centroid, of the contour through the Equations 14:

$$\bar{x} = \frac{m_{10}}{m_{00}}; \quad \bar{y} = \frac{m_{01}}{m_{00}} \quad (14)$$

which are used in the code to verify the button presses, as it'll be explained in subsection III-I.

Finally, the bounding rectangle can be extracted. The bounding rectangles of the buttons contours are used in the automatic video cropping (subsection III-H). The bounding rectangle's points are given by the set of equations 15:

$$rect = \begin{cases} TL = (min_x(f(x,y)), min_y(f(x,y))) \\ TR = (max_x(f(x,y)), min_y(f(x,y))) \\ BL = (min_x(f(x,y)), max_y(f(x,y))) \\ BR = (max_x(f(x,y)), max_y(f(x,y))) \end{cases} \quad (15)$$

where TL is the top-left point, TR is the top right, BL is the bottom left and BR, the bottom right. With the points, it's trivial to draw the lines between them.

G. Automatic button identification

To perform the automatic button identification, first, 30 frames are captured and for each one, the area and perimeter for all the contours found are stored. The perimeter against area scatter plot in Figure 6 (which is shown magnified in Figure 7) for the data collected shows 3 distinct clusters of elements: the blue, the orange and the green.

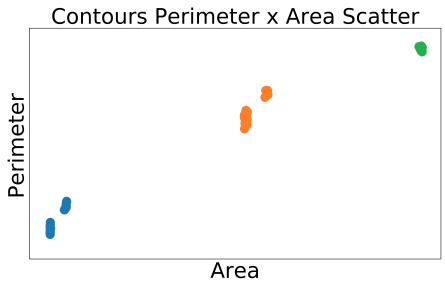

Fig. 7. Perimeter x Area scatter plot for all the contours found in the 30 frames recorded for analysis

The blue elements have very small perimeter and area, therefore can be classified as noise. The green elements are the opposite and they represent the contours related to the sheet of paper or other large contours found. The buttons will not be in any of those groups, since they are neither too small, nor too big, which means they'll be in the orange cluster of elements.

What is wanted in this part of the process is to find the buttons. Since the area and perimeter of the orange elements should match with a very good accuracy the average values of the buttons area and perimeter features, they can be used as exactly that, that is, for every frame streamed to the Video Joystick program from now on and for every contour in each of the frames, if the area and perimeter of this contour are within a range of the average calculated, that means it is a button, otherwise it isn't.

In practice, Video Joystick calculates the median of the area and perimeter of all the elements recorded, since it's the absolute fastest way to achieve the objective of finding good average area and perimeter that are inside the orange cluster. A more precise alternative would be to cluster the data with K-Means [20] and then find the real average of the orange cluster. With our experimental analyses, we concluded that this more precise version is unnecessary.

Thus, for each contour i in any give frame, it has to satisfy the Inequalities 16 to classify as a button.

$$\begin{cases} med(A(S)) - r_a \leq a(i) \leq med(A(S)) + r_a \\ med(P(S)) - r_p \leq p(i) \leq med(P(S)) + r_p \end{cases} \quad (16)$$

where med is the median, $A(S)$ and $P(S)$ are the set of areas and perimeters, respectively, of the set of elements S obtained during the 30 frame exposition time, $a(i)$ and $p(i)$ are the area and perimeter of i, and r_a and r_p are defined ranges or margins of error.

This method implies that all the buttons have approximately the same size. If this is not the case, there should be either different r values for the buttons, or one big enough to accommodate all different sizes. Although this is not a difficult feature to implement, it is not needed for this article, since we use the model shown if Figure 4 to execute the experiments, which assumes buttons with the same dimensions.

H. Automatic video cropping

Cropping the video frame, so that only the necessary parts of the frames appear, is an important step to reduce the possibilities of noise interfering with the calculations and also reduce CPU stress, since after cropping, there are fewer pixels to process.

Once the bounding rectangles of the buttons were extracted, it's trivial to define a satisfactory crop: the area to be cropped is another bounding rectangle that bounds all the previous bounding rectangles plus a tolerance value on each side to better accommodate the buttons on the screen.

I. Identifying button presses

This is another straightforward procedure once the features have been extracted from the buttons contours. For each button, let $(\bar{x}_r, \bar{y}_r)$ be its center of mass (or centroid) recorded during the 30 frames exposition time. In the running application (after the calibration ends), for each frame, check if the current centroid position $(\bar{x}_c, \bar{y}_c)$ of this button satisfies the Inequality 17:

$$\sqrt{(\bar{x}_r - \bar{x}_c)^2 + (\bar{y}_r - \bar{y}_c)^2} < r \quad (17)$$

where the inequality itself represents if the current centroid is within a radius r from the recorded centroid. If it is, the button is not pressed, since its contour is not deformed enough to change its centroid position. In the case where the new centroid falls outside the established range, it means something is interfering severely with the button's contour, therefore it must be obstructed (pressed). Notice how this works best if the buttons are of a convex (preferably circular/elliptical) shape. For non trivial shapes, this method does not suffice.

The interesting part of this approach is that it doesn't matter if a button is obstructed by a black-thresholded object or white-thresholded object. If the first case applies, there is, most of the time, a reduction of the button's area and perimeter, whereas if the second case happens, it's the opposite. For both cases there's a contour's shape deformity, which interferes directly on the centroid's position.

IV. Implementation Details

SOME important information on how Video Joystick is built and how some specific parts of the program works are described in this section.

A. Tools used

The tools used to build Video Joystick are all open source and should run on all major operating systems. Python 3[4] is the programming language of choice. Along with Python 3, some libraries are being used: NumPy[5] to deal with matrices and mathematical operations, Pynput[6] to send key inputs to the operating system and, of course, OpenCV as the heavy-lifter in respect to the video/image processing parts of the project (which are most of them).

B. Contour fine-tuning and key mapping

Although the buttons are automatically identified, Video Joystick may fail to properly process the contours due to unexpected factors and, consequently, fail to identify the buttons. With this in mind, the user may manually fine-tune the ranges in which buttons will be searched through a very basic user interface, as shown in Figure 8: Also, the user has to

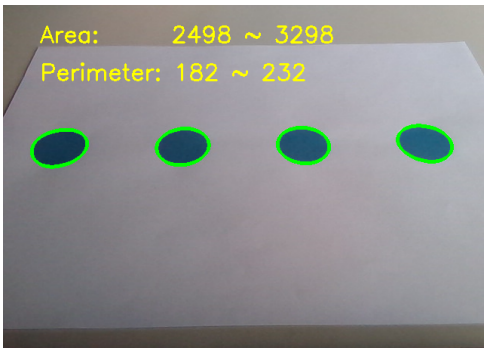

Fig. 8. Manual area and perimeter fine-tuning user interface

map which keyboard key will be represented by which Video Joystick button. This is done in a very similar fashion to the contour fine-tuning, as seen in Figure 9: A basic manual on

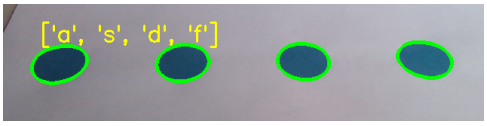

Fig. 9. Key mapping user interface

how to configure Video Joystick is provided along the source code for the project.

V. Experimental Methodology

[4]https://www.python.org/
[5]https://numpy.org/
[6]https://pypi.org/project/pynput/

EXPERIMENTAL methodology consists in letting different people play two different games: SuperTux[7] (Figure 10) and StarMines: The Next Generation[8] (Figure 11). The first is a Free and Open Source platformer similar to Nintendo's Super Mario and the second is a Free and Open Source Asteroids-like game. For SuperTux, the participants are required to go through the first level and for StarMines, they play until they lose all available extra lives.

The game sessions are recorded and the players are briefly interviewed to talk about their experience with the controller. We collect the following data about the recordings:

- Input delay - time between pressing the button and seeing the action
- Correctness - the command sent to the game is the one inputted?
- Contour detection failure - something that wasn't detected or that was but wasn't supposed to be characterizes a detection failure

The subjective questions asked to the players are:

- How did you feel about the controller?
- Did you get used to it already? If not, would you eventually?
- Which game was harder to play? Why?
- What are your suggestions to improve the controller?

Fig. 10. Screenshot from the game SuperTux <https://www.supertux.org/>

Fig. 11. Screenshot from the game StarMines: The Next Generation

VI. Results and Discussion

ANALYZING the collected footage, which was captured at 60 frames per second, we get that the average input delay between a button press and a action taking place in the games is 8.3ms. This was calculated by taking several examples of actions and counting the frames between the presses and actions.

All the actions happened at most 1 frame after a button has been pressed, so in the worst case there's a delay of 16.6ms, since:

$$\text{frames\_delay} \cdot \frac{\text{frames}}{\text{second}} = 1 \cdot \frac{1}{60} = 0.01\bar{6} = 16.\bar{6} \times 10^{-3} s, \quad (18)$$

and because approximately 50% of the times there was absolutely no frame delay, it makes sense to take the average between 16.6ms and 0.0ms, which is 8.3ms. This value is pessimistic, seeing that analyzing SuperTux, many times Tux was already rather distant from the floor in the next captured frame after the jump button has been pressed. Refer to Sections 2, 3, 4 and 5 of the Supporting Information to see a few of the analyzed inputs.

There was no problems related to correctness whatsoever during the performed tests. All the inputs, when recognized, were correctly sent to the computer and executed in the game. The key phrase here is "when recognized", since there were times where the controller failed to identify that a button was pressed, as seen in Section 6 of the Supporting Information. This is not a very common behavior and was observed more during button mashes or atypical situations.

Now, we display the opinions from three different people about the controller. They were asked the previously mentioned questions.

How did you feel about the controller?

Player 1: "The biggest shock regarding using the controller was the fact that I couldn't keep my hands on the buttons and keep in mind that shadows also press the buttons, which is different from any other controller that I've used in the past. I also felt that a few times the buttons I pressed weren't recognized or they kept going on even after I released the button."

Player 2: "I really liked it, although it felt odd not being able to have my fingers on top of the buttons all the time, like I would in a regular controller/keyboard. The responsiveness was great and it's intuitive to use. In the beginning, I just had to keep remembering myself that it's not pressing the buttons that's going to result in a game move, but keeping your fingers inside and outside of the responsive area; but that's easily learned from practice."

Player 3: "I enjoyed using it. Covering one button at a time resulted in good responsiveness, but sometimes, when I covered two or more buttons, one of them wasn't recognized or kept being recognized even after uncovering it."

Did you get used to it already? If not, would you eventually?

Player 1: "Yes, I got used to it fairly quickly, although I wouldn't use it in games where dexterity or extremely precise movement is fundamental"

Player 2: "I did, yes. And I think that if anyone tries the controller enough times, they'll get used to it. The main thing that made me better at it was realizing that I needed to keep my fingers away from the responsive area instead of just stopping pressing the button for my character to stop doing a move."

[7]https://www.supertux.org/index.html
[8]https://smtng.jpkware.com/

Player 3: "Yes. It was weird at first not being able to feel any button being pressed or where I was touching on the paper, but eventually I got used to the places where I should cover."

Which game was harder to play? Why?

Player 1: "The second game was much harder to play. The spaceship's inertia in the space makes it difficult to control. I felt that the game would have been a little easier on a traditional controller, where I could've input little nudges to straighten the ship more easily."

Player 2: "The second one. The spaceship was much harder to control than the penguin. The sensitivity and the fact that we had to use multiple buttons simultaneously made it very hard to move and to aim exactly where I wanted."

Player 3: "The second one, definitely. I think it would be harder using pretty much any controller, since it requires much more precision and coordination."

What are your suggestions to improve the controller?

Player 1: "I'd say that shadows pressing buttons is an inconvenience, so fixing this would make the controller easier to operate"

Player 2: "Sometimes, in the penguin game, I had trouble jumping while pressing the forward button; it would be great if that could be fixed. I don't know if it's something that's related to the game or to the controller itself, but I also think that we should be able to make the 'turn button' less sensitive to the spaceship game in order to make the ship turn around slower. That way we could control it much easier."

Player 3: "The problem that happens when multiple buttons are pressed was my only problem, actually. If that was fixed the controller would be perfectly usable for me."

VII. Conclusion and Future Works

There's a long history of companies trying to find new ways to provide a more ergonomic, easy-to-use and immersive gaming experience through new input methods. Video Joystick attacks primarily the easy-of-use aspect providing an accessible way to put together a joystick and play anywhere, given the player has access to a webcam, sheet of paper, pen and a computer with Python 3 and the required libraries installed.

The experience playing with Video Joystick is different from the usual, since it doesn't require touching anything, but also works just as fine if the buttons are, in fact, touched. It also requires some adaptation in respect to the way the players position their hands to operate the controller, which can be hard in the beginning, but our studies show that the adaptation process isn't complicated and most players can learn how to use it after a short period of time.

Although the tests ran gave positive results, specially related with the more technical aspects of the controller, it is not the perfect input method and still needs to be further calibrated and automated, so that it's even easier to use. For now, we provided a good enough system to play basic games, but since the project is fully open source (See Section 7 in the Supporting Information), we hope that interested and technically capable users can modify the developed code to better fulfill their needs.

References

[1] M. Wolf, *The Video Game Explosion: A History from PONG to Playstation and Beyond*. Greenwood Press, 2008. [Online]. Available: https://books.google.com.br/books?id=XiM0ntMybNwC

[2] M. Cowan, "Interactive media and imperial subjects: Excavating the cinematic shooting gallery," *European Journal of Media Studies*, no. 1, pp. 17–44, 2018. [Online]. Available: https://doi.org/10.25969/mediarep/3438

[3] R. Bhardwaj, "The ergonomic development of video game controllers," *Journal of Ergonomics*, vol. 07, 01 2017.

[4] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE MultiMedia*, vol. 19, no. 2, pp. 4–10, 2012.

[5] M. Liebl, "Wii lifetime sales surpass 100 million units - gamezone," *GameZone*, 2013. [Online]. Available: https://www.gamezone.com/news/wii-lifetime-sales-surpass-100-million-units/

[6] B. Romaissa, B. Nini, M. Sabokrou, and A. Hadid, "Vision-based human activity recognition: a survey," *Multimedia Tools and Applications*, vol. 79, 11 2020.

[7] R. Lun and W. Zhao, "A survey of applications and human motion recognition with microsoft kinect," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 29, no. 05, p. 1555008, 2015. [Online]. Available: https://doi.org/10.1142/S0218001415550083

[8] J. Aggarwal and L. Xia, "Human activity recognition from 3d data: A review," *Pattern Recognition Letters*, vol. 48, pp. 70–80, 2014, celebrating the life and work of Maria Petrou. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167865514001299

[9] M. Ahmad and S.-W. Lee, "Human action recognition using shape and clg-motion flow from multi-view image sequences," *Pattern Recogn.*, vol. 41, no. 7, p. 2237–2252, Jul. 2008. [Online]. Available: https://doi.org/10.1016/j.patcog.2007.12.008

[10] P. Wang, W. Li, P. Ogunbona, J. Wan, and S. Escalera, "Rgb-d-based human motion recognition with deep learning: A survey," 2018.

[11] N. Ahmad, R. A. R. Ghazilla, N. M. Khairi, and V. Kasi, "Reviews on various inertial measurement unit (imu) sensor applications," *International Journal of Signal Processing Systems*, vol. 1, no. 2, pp. 256–262, 2013.

[12] R. F. Tinder, "Relativistic flight mechanics and space travel," *Synthesis lectures on engineering*, vol. 1, no. 1, pp. 1–140, 2006.

[13] H. Singh, "How images are stored in the computer?" *Analytics Vidhya*, 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/03/grayscale-and-rgb-format-for-storing-images/

[14] M. M. Arat, "Rgb to grayscale conversion," *Arat, Mustafa Murat*, 2020. [Online]. Available: https://mmuratarat.github.io/2020-05-13/rgb_to_grayscale_formulas

[15] A. Rosebrock, "Opencv smoothing and blurring," *pyimagesearch*, 2021. [Online]. Available: https://www.pyimagesearch.com/2021/04/28/opencv-smoothing-and-blurring/

[16] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 146 – 165, 2004. [Online]. Available: https://doi.org/10.1117/1.1631315

[17] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[18] C.-H. Teh and R. Chin, "On the detection of dominant points on digital curves," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 8, pp. 859–872, 1989.

[19] G. Strang and E. erman, "Arc Length and Curvature," 1 2021, [Online; accessed 2021-08-04]. [Online]. Available: https://math.libretexts.org/@go/page/2596

[20] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, no. 3, p. 264–323, Sep. 1999. [Online]. Available: https://doi.org/10.1145/331499.331504