Visual motion of a player's fingers
AICV PROJECT 2018 - 2019

Leonardo Bisica

Contents

1 Project description

Aim of this project is to mark out fingers motion on a Piano Musical Track. Specifically there were need to identify pressure of keyboard keys from two source videos obtained by as many synchronized cameras/

Provided `MATLAB`® script operate in a "frame-by-frame" manner in order to:

- output `frame` number in which a `key` is classified as **pressed**
- produce an new "Highlight Video", joining the two camera sources with the addition of *special markers* whenever a key will result as **pressed**.

In order to identify world coordinates of *fingers* and *pressure points* Cameras have been put in such a way to reconstruct the 3D space from input frames; There were no need to compute neither Cameras Calibration Matrices nor Stereo Fundamental Matrix due to usage basic Euclidean and Projective Geometry rules. This choice was supported by the approximating opportunity that the setup's known measures gave, making necessary only to find a series of features that could be easily translated from image planes to real world space.

Aforementioned rules, explained in detail on Section 3, are thought in the "ideal setup" circumstance. Provided Script on the other hands aims to

1. Compute rules variable in the image plane (Section 4.2).
2. Implement rules in the real world scenario (Section 5).

2 Ambient Setup

Cameras Definition

The two implied cameras come from two identical OnePlus 6T cellphones. Videos have been shoot with identical camera settings.

2.1 Cameras Placement

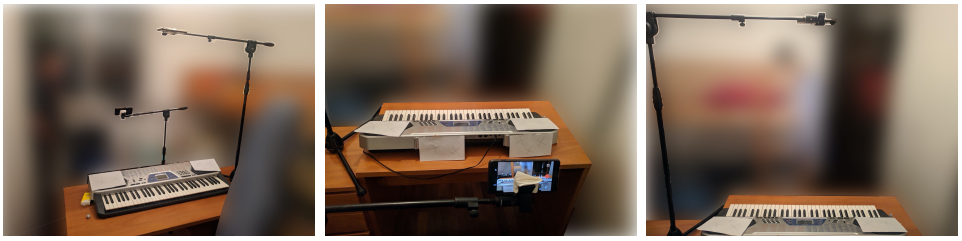


Figure 1: Setup and placement of tripods with cameras with respect to keyboard desk

Both cameras have been hooked on two extensible tripods placed such that the scene would be filmed:

- Behind the keyboard, shooting both keyboard and player's body; referenced from now on as ***Front Camera***
- From top of the keyboard, shooting desk, keyboard keys and player's fingers; referenced from now on as ***Upper Camera***

Figure 1 shows camera placement in the room.

2.2 Main Assumption

Theoretical rules defined in further sections are based on some main assumptions:

1. Keyboard's "Playing Surface" is parallel to desk plane where keyboard is placed on.
2. Both used cameras are assumed to be perfectly aligned *w.r.t.* Desk's axis as:
    - Upper Camera's image plane is parallel to Keyboard Playing Area plane

- Front Camera's image plane and Keyboard Playing Plane horizontal axis are parallel in between them.

3. Viewing Rays Exiting from Front Camera going to Keyboard can be roughly flattened as follow:
   Any series of viewing rays belonging to a 2D plane in the World Space that:

   - is perpendicular to Room Floor
   - passes through Camera Center
   - contains a portion of Keyboard Playing Area

   Can be approximated with an unique viewing ray that is the average of rays reaching beginning and ending of keyboard playing area. From now on, this key feature will be referred as Principal Viewing Ray. Actual computation will be addressed later in this section, while an example of can be seen in Figure 2a.

Given that some later software correction where made, those assumption where proved not to affect much algorithm results. Further details are given in section 4.2

2.3 Known World Measures

In the given setup, some measurements have been taken by hand after video shooting.

Regarding Camera position, assumptions on Upper Camera lead to no interest in computing measurements referred to it; on the other hand most of measurements are referred to Front Camera and its relation to Desk. Moreover, in order to compute fingers position, accurate measures of Keyboard's specifics had to be taken in account. Figure enlights all used necessary measures.

All accounted measures are listed below:

Desk

- **Desk Height** (starting from floor): $75cm$
- **Desk Depth**: $60cm$

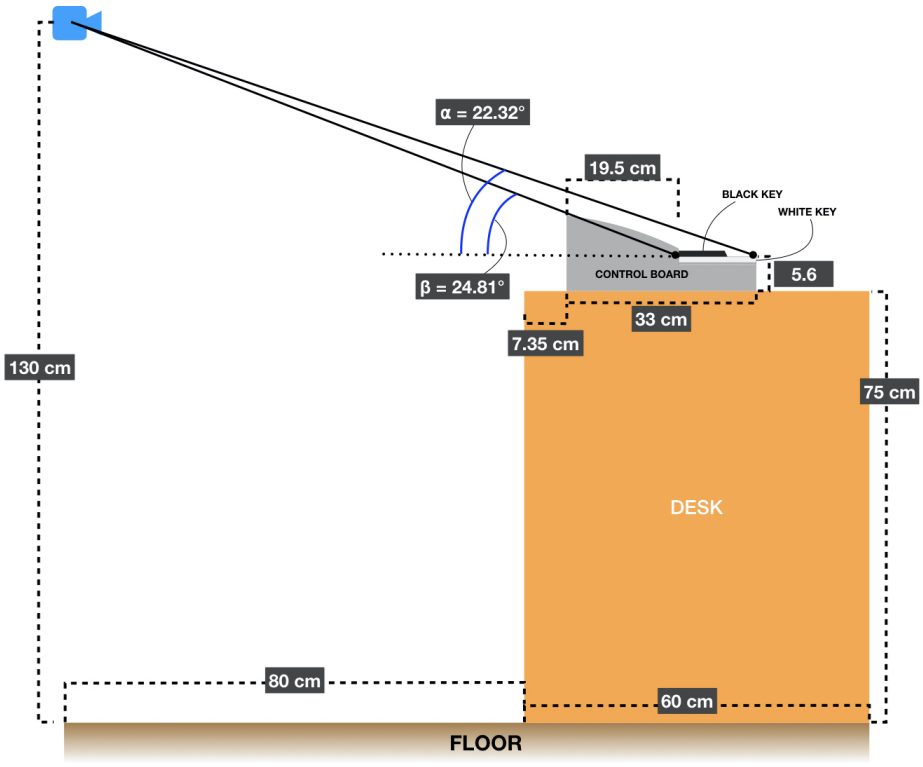
Front Camera

- **Camera Height** starting from floor: $130cm$
- **Camera - Desk Distance**: $80cm$

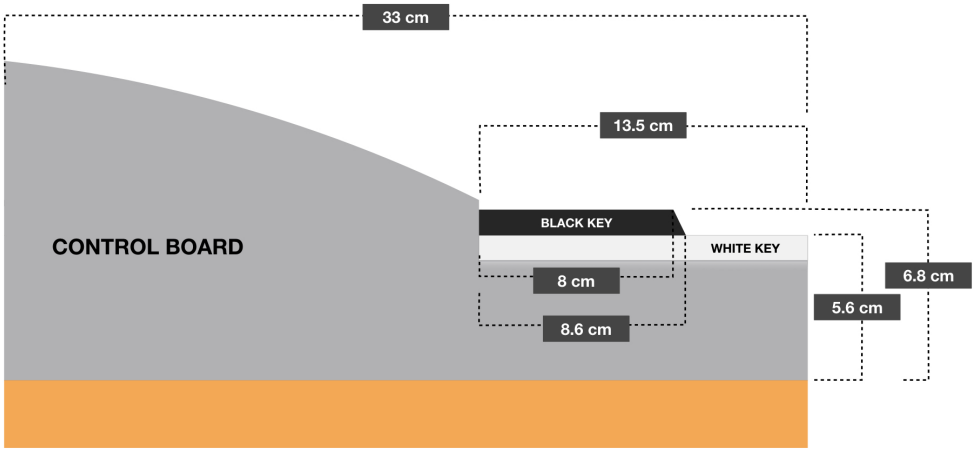
Keyboard

Macro Measures

- **Keyboard Total Depth** : $33cm$
- **Keyboard's Control Board Depth** : $19.5cm$
- **Keyboard's Keys Plane Depth** : $13.5cm$
- **Keyboard Minimum Height**: $5.6cm$, corresponding to Height of White Key's top border *w.r.t.* desk plane
- **Keyboard to Desk Border distance**: $7.35cm$, computed as distance within desk's border and Keyboard's Control Board.



(a) Measurements refered to Front Camera's tripod position, taking in account Desk and Keyboard



(b) Details of Keyboard internal measurements

Figure 2: Details and Metrics of the whole setup referred to Front Camera 2.3

Micro Measures

- **White Key Height**: $5.6cm$, corresponding to *Keyboard Minimum Height*
- **White Key Length**: $13.5cm$, corresponding to *Keys Plane Depth*
- **Black Key Height**: $6.8cm$
- **Black Key Min Length**: $8cm$
- **Black Key Max Length**: $8.6cm$

Derived Measures Two Angles have been computed in order to approximate *Front Camera viewing rays* pointing to Keyboard playing area.

Both viewing rays starts from Front Camera and respectively end on starting and ending of playing area plane (remember that playing area is defined by White Keys plane)

angle α: pointing to end of keyboard plane

$$\alpha = \arctan\left(\frac{\text{plane height}}{\text{plane distance to camera}}\right)$$

$$\text{height} = \text{Camera Heighth} - \text{Desk Height} - \text{KBD min Height} =$$

$$= 130 - 75 - 5.6 = 49.4cm$$

$$\text{distance} = \text{Camera Distance} + \text{Desk KBD Depth} + \text{KBD Depth} =$$
$$= 80 + 7.35 + 33 = 120.35cm$$

$$\alpha = \arctan\left(\frac{49.4}{120.35}\right) = 22.32^\circ$$

angle β: pointing to beginning of keyboard plane

$$\beta = \arctan\left(\frac{\text{plane height}}{\text{plane distance to camera}}\right)$$

$$\text{height} = \text{Camera Heighth} - \text{Desk Height} - \text{KBD min Height} =$$
$$= 130 - 75 - 5.6 = 49.4cm$$

$$\text{distance} = \text{Camera Distance} + \text{Desk KBD Depth} + \text{KBD Control Board Depth} =$$
$$= 80 + 7.35 + 19.5 = 106.85cm$$

$$\beta = \arctan\left(\frac{49.4}{120.35}\right) = 24.81^\circ$$

2.3.1 Viewing Ray approximation

Principal Viewing Ray tilting angle can be computed as the arithmetic mean of the two angles α and β. An example is given in Figure 2a.

$$\theta_{v_r} = 23.5646^\circ$$

3 Theoretical Rules

This section explains all necessary Theoretical Background applied during Algorithm design. Purpose is to apply Projective and Euclidean Geometry rules to transform Finger Tips coordinates to knowledge in World Space. The former is defined by a pair of $[x, y]$ coordinates on pixel reference system, while the latter is represented by a tuple of $[X, Y, Z]$ coordinates in Metric System; specifically, during script execution values refers to millimeters distance.

Front Frame will use Cross Ratio in converting pixel coordinates into *depth w.r.t.* keyboard playing area.

Upper Frame will apply proportionality rules from Pixel Metrics to Metric System.

3.1 Reference System

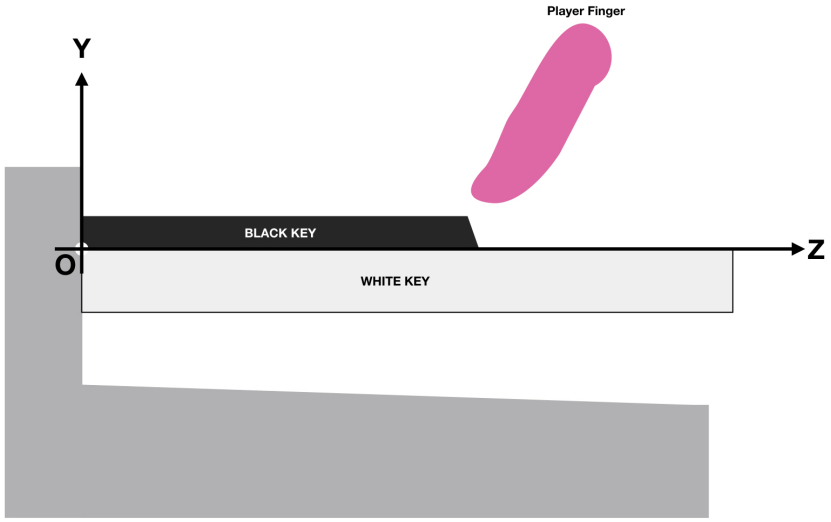


Figure 3: Reference system positioning. Y axis' growth indulges world's *height*, while Z axis defines how *depth w.r.t.* Front Camera placement. X axis is defined by the *right hand rule*, with growing direction "exits" from this paper.

While Figure 2a expresses camera-keyboard distances, Figure 3 defines a new reference system, focused on small details onto Keyboard Playing

Surface.

This new system can be viewed as a rigid translation of World reference system having origin placed onto Front Camera, having:

- Y axis' growth representing world's *height*,
- Z axis' growth representing world *depth w.r.t.* Front Camera placement,
- X axis defined by *right hand rule*, growing from left to right

Upcoming section is devoted to prove that X_{WORLD} axis can be ignored in further computation.

3.1.1 MATLAB® s Frame System to World Frame System

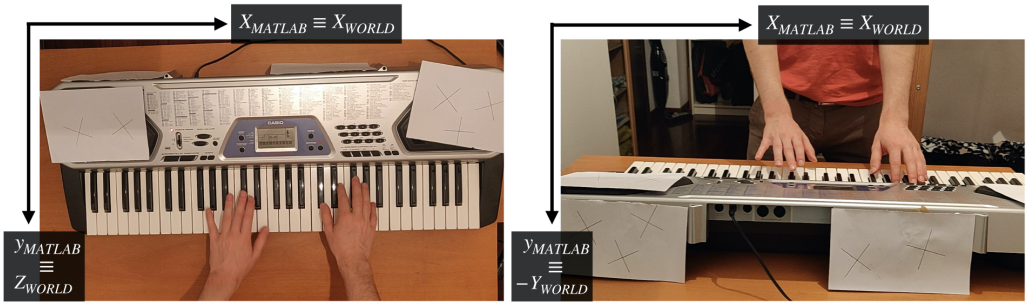


Figure 4: Reference system positioning

Recalling assumptions stated in section 2.2, Cameras' produced image planes, indexed by MATLAB® $[x, y]$ coordinates. Pairing with World Keyboard Reference system is done as:

- **Front Camera**'s image plane:
  a) $x_{MATLAB} \equiv X_{WORLD}$
  b) $y_{MATLAB} \equiv -Y_{WORLD}$
- **Upper Camera**'s image plane:
  a) $x_{MATLAB} \equiv X_{WORLD}$
  b) $y_{MATLAB} \equiv Z_{WORLD}$

Since both x_{MATLAB} axes are consistent to each other, X_{WORLD} coordinate defines only "Finger Tips ordering" in the World system. If tips from both frames can be matched altogether it will be possible to exclude X_{WORLD} component from computation, leading to focus only on *height* and *depth* of each separate Finger Tip. Those properties in Keyboard Reference system are described by Y_{WORLD} and Z_{WORLD} axes. Tips Matching Process is described in section 5.3

3.2 MATLAB® Coordinates Convertion

This section defines how coordinates identified on frame can be translated and combined to produce a Finger Tip's World $[Y, Z]$ Coordinates.

3.2.1 Front Camera to World

Introductory Note Following reasoning will apply **only** to identified Finger Tips that have a negligible real world height *w.r.t.* Keyboard Playing Area. This is due to the fact that Script must focus only on identifying Finger Tips that are actually pressing keys. Finger tips that are far from Keyboard Playing Area will be applied the same rules but will surely return numerical errors and therefore produce an inconsistent result. This kind of exception has to be properly handled.

Finger tip Coordinates Identification Tip are identified in the image plane by the two $[x, y]$ coordinates, but only the latter embeds knowledge regarding its height and depth. Whenever a finger touches a key, height is near 0, so y coordinate expresses only depth property. Direct implication is that the only extractable feature from Front Camera could be **depth**, that will set on Z_{WORLD} axis.

In moments preceding and following key pressure Tip's height will be negligible, leading identified y coordinate to be treated as if the finger still touches keyboard. Height's negligible-ness is enforced by the fact that Front Camera is placed far away from Keyboard (see section 2.3)

This procedure summarizes as the "projection of Tip coordinates along the direction defined by Camera viewing ray until Keyboard Playing Plane is reached".
Resulting points will respect the following relation, recalling that MATLAB® image origin is places in the upper right corner:

$$x_{proj} \approx x$$

$$\begin{cases} y_{proj} \leq y & \text{if tip over White keys level} \\ y_{proj} \approx y & \text{if tip near White keys level} \\ y_{proj} \geqslant y & \text{if tip under White keys level} \end{cases}$$

Figure 7 shows this behavior in the different cases.

Sidenote on Finger Projection As defined in Section 2.2, only viewing rays that reaches keyboard surface are considered useful and can be summarized into a unique ray tilted about 20 degrees *w.r.t.* that same surface. Natural consequence of this assumption is that projection will be done along this particular ray. This means that finger that are really far away from keyboard surface must be ignored in order to avoid numerical errors.

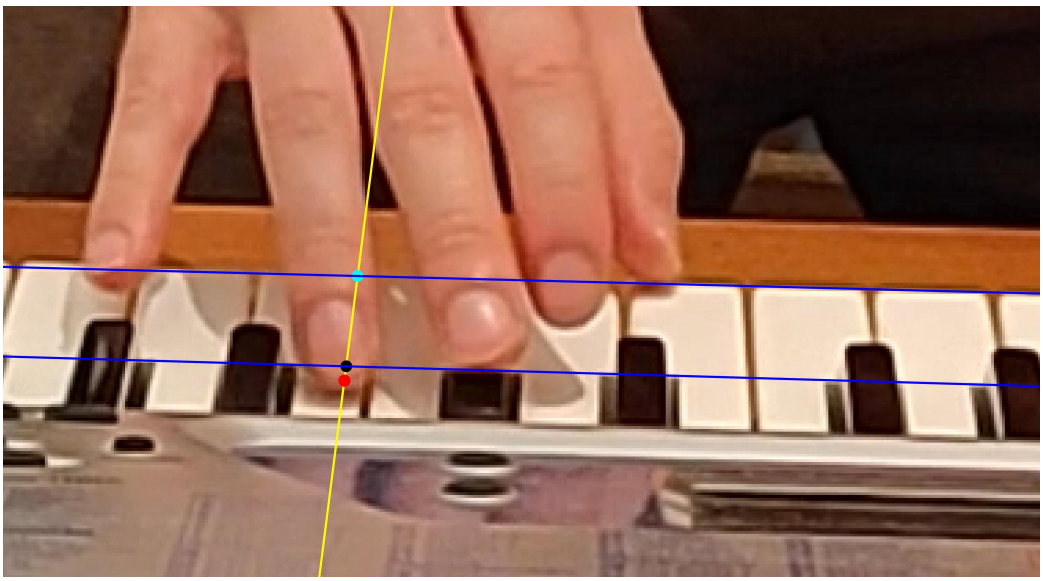


Figure 5: Example of Cross Ratio Rule application. Red marker refers identified Finger Tip, Yellow Line is the vertical plane, oriented to image's vanishing point; Black Marker defines intersection between Black Keys Extrema and vertical plane, while Cyan marker defines it intersection with White Keys Extrema. Cross Ratio returned a value of 0.8685 with resulting real world depth of $71.6749mm$ *w.r.t.* Coordinates System's Origin

Cross Ratio Projected finger is than translated into Real World Depth by mean of Cross Ratio Computation:
When vertical plane defined in Section 2.2 crosses Playing Surface it generates two points that have fixed distance allover Keyboard, and can easily be

extracted with a series of pre processing steps from background frames:

1. Keyboard Farther Extrema, coincident with White Keys extrema;
2. Black Keys Extrema, in the locus where they encounter White Keys.

Figure 5 shows points' markers on an example frame.

Fourth and last fixed point needed is the "*Keys' Longest Side* Vanishing Point"; computation details are postponed up to Section 4.2.1

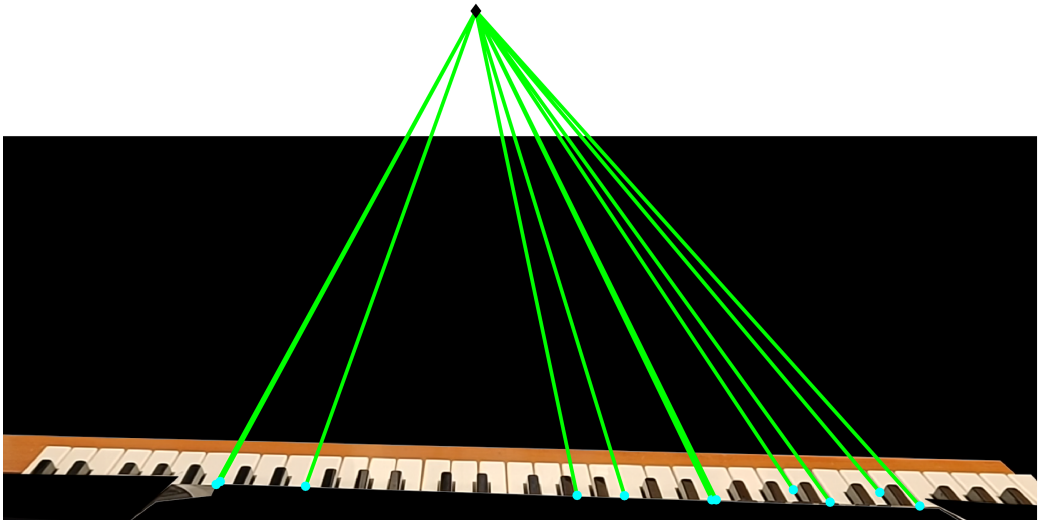


Figure 6: Vanishing Point obtained by intersecting lines derived from Keys' Longest Side. Computational Details are explained in Section 4.2.1

Points Arrangement Convention Given the four points, Cross Ratio convention is fixed as:

$$Cr(P_1, P_2, P_3, P_4) = \frac{\overline{P_1P_3} \cdot \overline{P_2P_4}}{\overline{P_2P_3} \cdot \overline{P_1P_4}}$$

$$\begin{aligned} P_1 = \text{Black\_End} &\xrightarrow{\text{refered as}} B \\ P_2 = \text{Finger\_Tip} &\xrightarrow{\text{refered as}} T \\ P_3 = \text{White\_End} &\xrightarrow{\text{refered as}} W \\ P_4 = \text{Vanishing\_Point} &\xrightarrow{\text{refered as}} V_p \end{aligned}$$

$$Cr(B,T,W,V_p) = \frac{\overline{BW} \cdot \overline{TV_p}}{\overline{TW} \cdot \overline{BV_p}} \tag{3.1}$$

Cross Ratio in World Space In World Space distances segments that go towards Vanishing Point tends to Infinity, leading ratio $\frac{\overline{TV_p}}{\overline{BV_p}}$ to be ignored since it doesn't provide anymore usefull details.

Thus, in Real World Cross Ratio is computed as:

$$Cr_{WORLD} = \frac{\overline{BW}}{\overline{TW}} \tag{3.2}$$

From Image Plane to World Cross Ratio is Invariant in Projective Transformations, thus value computed in image frame can be arithmetically matched with Real World's one. Defined equality, with the addition of the knowledge on $\overline{BW}$ and $\overline{OW}$ segments' length defined in Section 2.3 gives that *Finger Tip's depth* [1] defined by $\overline{OT}$ segment can be computed as:

$$\overline{OT} = \overline{OW} - \overline{TW} = \tag{3.3}$$

$$= \overline{OW} - \frac{1}{Cr} \cdot \overline{BW} \tag{3.4}$$

Sidenotes on Cross Ratio Based on Cross Ratio value, Tip's Position can be distinguished in three distinct cases *w.r.t.* points (O,B,W): [2]

1. $Cr \geq 1$: Point are disposed as $O \to B \to T \to W$
2. $0 < Cr < 1$: Point are disposed as $O \to T \to B \to W$
3. $Cr \leq 0$: Point are disposed as $O \to B \to W \to T$

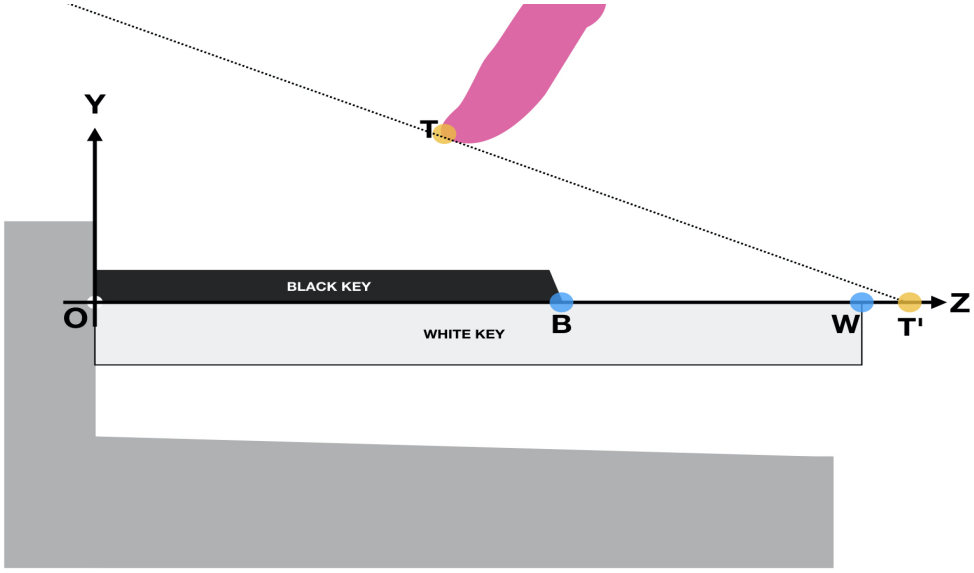
Note that in the third case, segment $\overline{TW}$ in equations 3.1, 3.2 and 3.3 results to be of negative length

3.2.2 Upper Camera to World

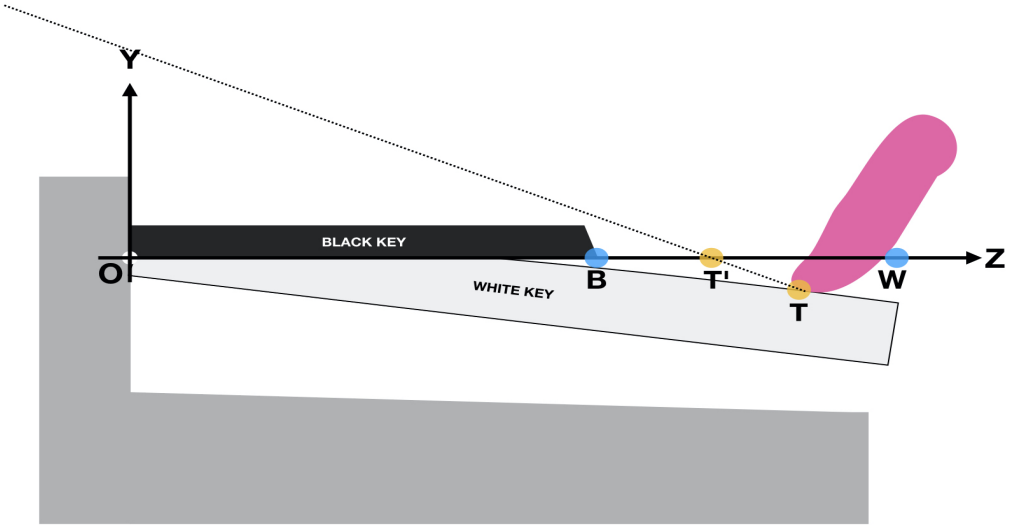
As referenced in Section 2.2 main assumption regarding Upper Camera states parallelism between Keyboard's Playing Surface and Image Plane. Given

[1] Depth is refered to Coordinate system origin, placed on Keys's Joint on Control Board

[2] Vanishing Point can be set aside since it always be the farthest point of this sequence



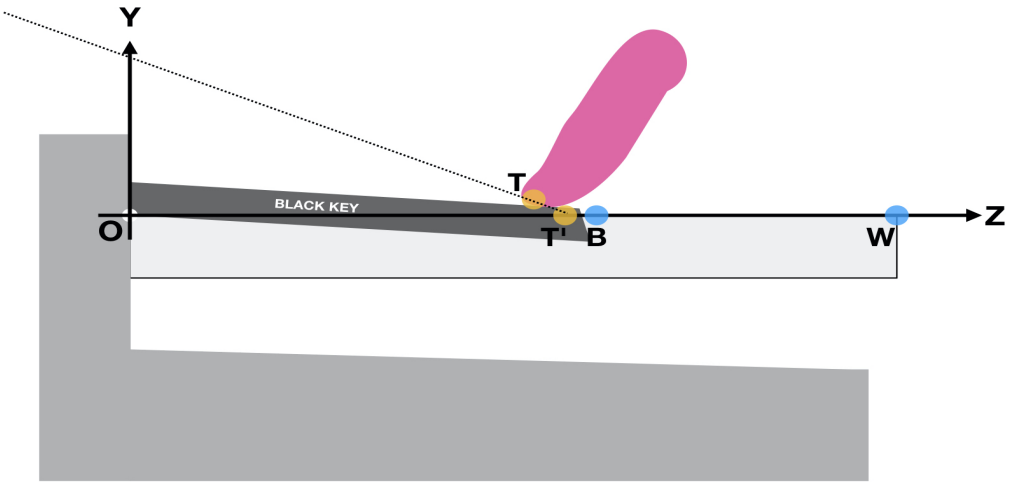
(a) Distant Finger producing a projection out the Playing Area



(b) Finger pressing a white key, which projection is closer to the origin

Tip's $[x, y]$ coordinates, only the latter expresses information about *depth*. No information about Tip's height can be inferred.

Depth can be computed by converting from Pixel Measures to Metric system the segment joining Tip to "Keys Joint to Control Board". *Pixel to Millimeters ratio* computational details are postponed to Section 4.2.2.



(c) Finger pressing a black key, having its projection closer to it

Figure 7: Examples of Finger Tip Projection in relation to fixed breakpoints O (System Origin) B (Black Keys Extrema), and W(White Keys Extrema)

Figure 8: Example of segments joining Tips to Keyboard Joint Line, identified by the blue line. Text in boxes defines pixel distance dividing nearest tips. After conversion to metric system, left pinkie will result in a distance of 81.9931mm, left thumb 115.8343mm and right middle finger 11.6153

3.3 Rules for Key Pressed

After conversion to Metric system, both frames provide only knowledge about Tips' *depth w.r.t.* Keys Joint to Control Board.

While Upper Camera defines a true distance, Front Camera provides a projected depth; therefore, real world height reconstruction is made possible using *Principal Viewing Ray*'s Tilting Angle θ computed in Section 2.3.1.

From now on, Upper Camera depth will be referred as Z_{up}, while Front Camera's as Z_{front}. Following procedure for height reconstruction is illustrated by Figure 9 and is defined as follows:

Draw vertical w line passing though Z_{up}

Draw a line v_r tilted clockwise by θ to Z axis and passing though Z_{front}

Define d segment lying on Z axis which extrema are Z_{up} and Z_{front}

Define h segment belonging to w which extrema are Z_{up} and intersection of w and v_r h corresponds to Finger Tip's actual height.

h length can be computed as:

$$h = d \cdot \tan(\theta) = |Z_{up} - Z_{front}| \cdot \tan(\theta) \tag{3.5}$$

'White Key Pressed' Rule Note that Z_{up} value will always be smaller that Z_{front} unless Finger Tips is found to be lower than White Keys' Plane. Main rule for Identification of White Key Pressure is:

$$Z_{up} \geqslant Z_{front} \tag{3.6}$$

'Black Key Pressed' Rule h segment value can be numerically compared to *Black Keys' Height w.r.t.* White Keys' Plane (referenced as h_b). Recalling Measurements defined in Section 2.3 this distance is defined as:

h_b = Black Keys Height - White Keys Height = $68mm - 56mm = 12mm$

Main rule for Identification of White Key Pressure is:

$$h \leqslant h_{\text{black\_key}} \tag{3.7}$$

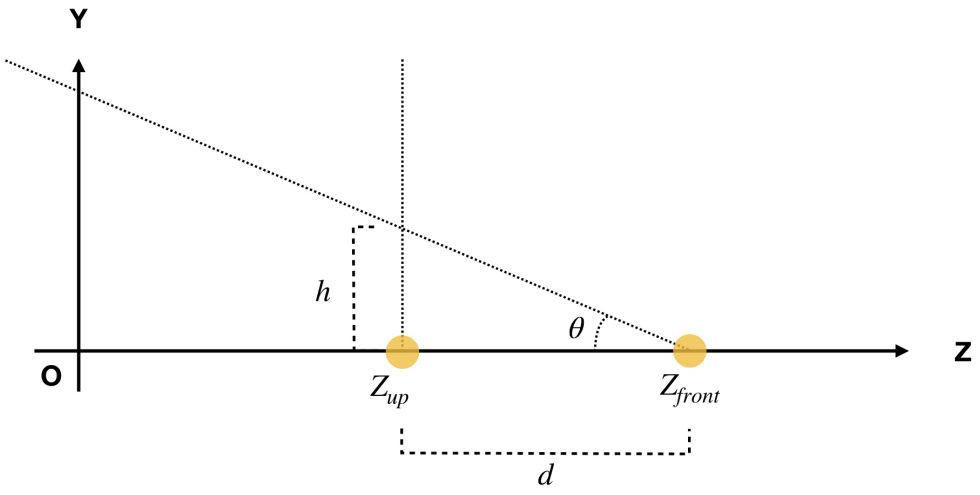


Figure 9: Finger Tip's height reconstruction schema

4 Script Pre Processing Steps

This Section illustrates how basic elements (from now on *features*) needed by the two "Key Pressed" rules are actually computed in provided `MATLAB®` script.

All features are computed after the so called *Keyboard Masks*, binary `ROI`s that defines Playing Area; first part of this Section concerns about this Mask Identification for both frames.

Right After there will be computation of *Geometric Features*.

A new element will be introduced: *Keys Centroids*, needed by routine step of *Tips Matching*.

Masks are computed starting from special video contained in project repository containing only background elements; last part of this section is devoted to Video Pre Processing Steps and background computation.

4.1 Binary Mask Processing

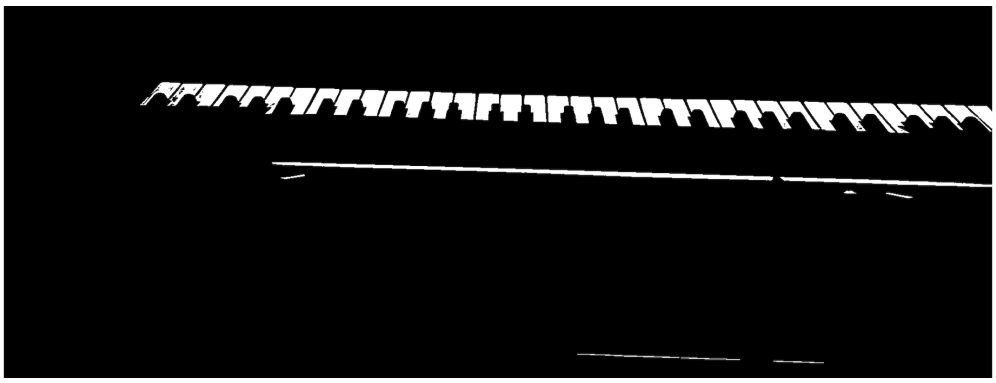
Key Mask defines contour and borders of the already mentioned -and from now on renamed from- *Keyboard Playing Area*. This element is a Binary Matrix where 1 marks interesting elements. Later on, during script routine

those will be used to identify fingers that are over keyboard.

Computation is done by mean of isolation and later merge of the two keys colors: Black and White. Hardest aspect challenge for the algorithm is color identification having due to intense light reflections on both frames.

4.1.1 Front Keys Binary Masks

Figure 10: Details on Front Binary Mask Computation Procedure



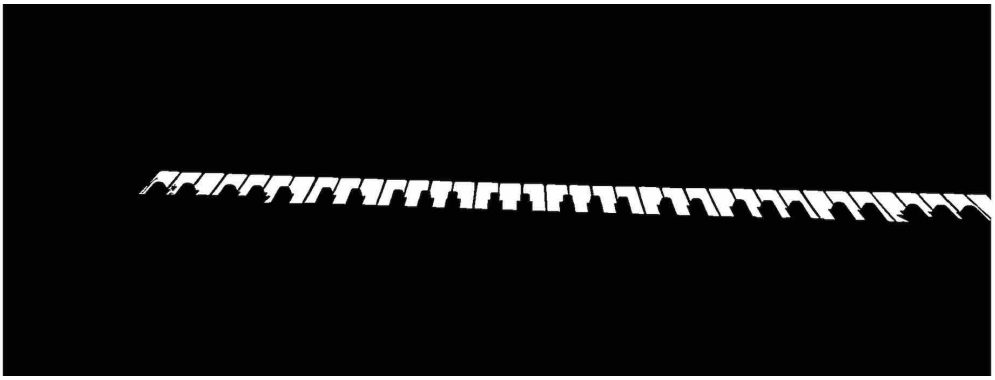
(a) RAW Mask resulting from Binarization and Thresholding

(b) Wider element Isolation

White Key Mask Computation states as follows, by mean of MATLAB® morphological operators; Procedure results are illustrated in Figure 10

1. Binarization and Thresholding of each single RGB component. 10a

Resulting RAW Mask enlights keys filled contours, with the addition of some noise given by other ambient elements that matched thresholding rules.



(c) Final White Key Mask

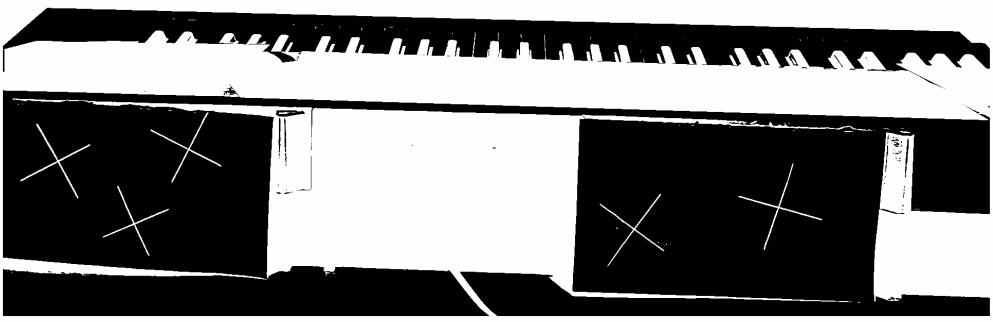
2. Dilation of positive elements by a line parallel to Mask's Principal Horizontal Component, and Isolation of the largest element. 10b

Resulting image returns all keys joined in an unique wide element. Embiggen noisy results in a lesser area *w.r.t.* the former and therefore deleted by this pruning.

3. Logical conjunction with initial RAW Mask. 10c

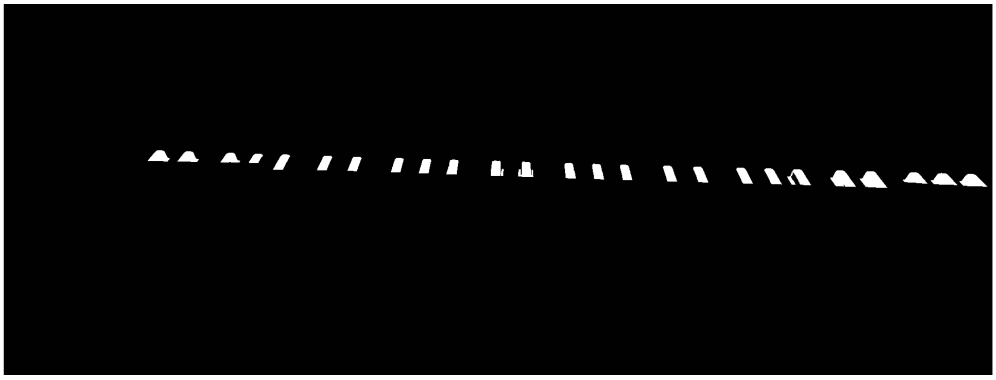
Final result will have Truth value only in presence of original white key.

Figure 11: Details on Front Binary Mask Computation Procedure



(a) RAW Mask resulting from Binarization and Thresholding

Black Key Mask Computation states as follows. Procedure results are illustrated in Figure 11



(b) Final Black Key Mask

1. Binarization and Thresholding of each single RGB component. 10a

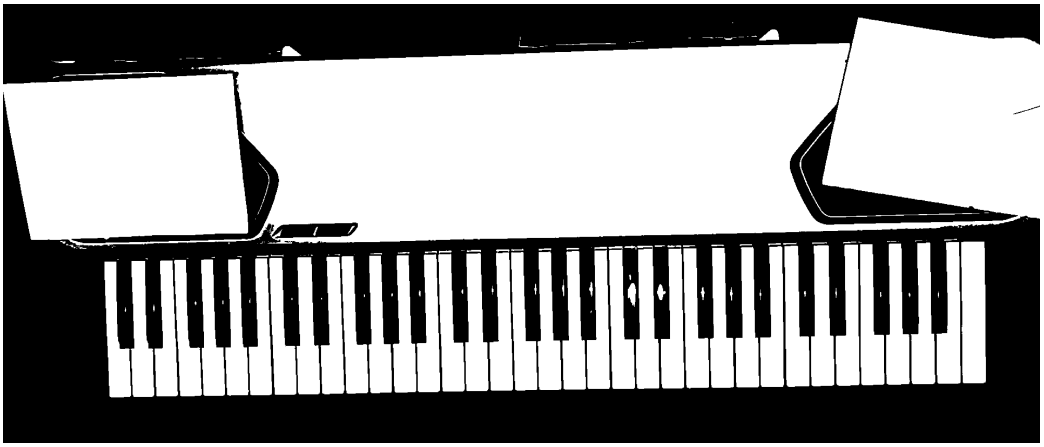
Resulting RAW Mask enlights keys filled contours, with the addition of some noise given by other ambient elements that matched thresholding rules.

2. Logical conjunction with initial White Keys Dilated Mask. [3] 10c

Final result will have Truth value only in presence of original white key.

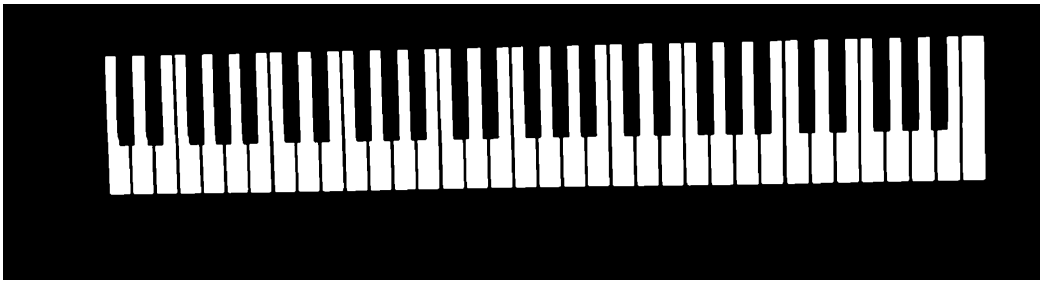
4.1.2 UP Keys Binary Masks

Figure 12: Details on Front Binary Mask Computation Procedure



(a) RAW Mask resulting from Binarization and Thresholding

[3]Dilated mask computed for White keys comprehends by definition also Black keys since Joint on Control Board is the same for every kind of Key.



(b) Final White Key Mask

White Key Mask Computation states as follows, by mean of `MATLAB`® morphological operators; Procedure results are illustrated in Figure 12

1. Binarization and Thresholding of each single RGB component. 12a

Resulting RAW Mask enlights keys filled contours, with the addition of some noise given by other ambient elements that matched thresholding rules. Also wide pieces of Control Board survived thresholding.

2. Dilation of positive elements Contours and Deletion of largest element and small noise pieces. 12b

Dilation is used to create a unique `ROI` containing control board; dilation parameters have to be enough to maintain Keys contours disjoint.

Final result will have Truth value only in presence of original white key.

Black Key Mask Computation states as follows. Procedure results are illustrated in Figure 13

1. Binarization and Thresholding of each single RGB component. 13a

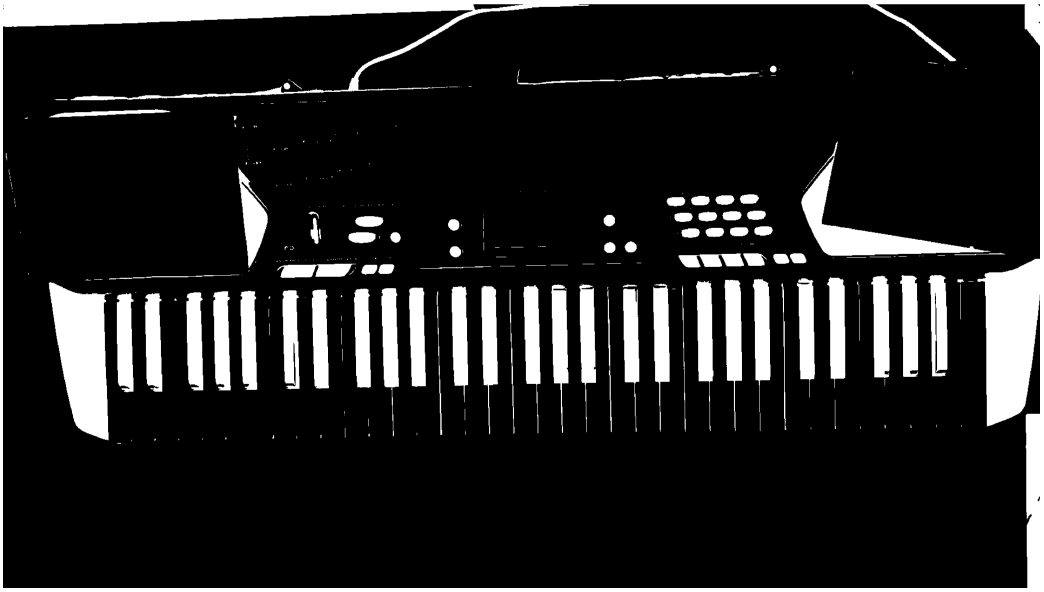
Resulting RAW Mask enlights keys filled contours, with the addition of some noise given by other ambient elements that matched thresholding rules.

2. Dilation of positive elements by a line parallel to Mask's Principal Horizontal Component, and Isolation of the largest element. 13b

Resulting image returns all keys joined in an unique wide element. Process will also include sides of control board.

3. Logical conjunction with Original RAW Mask. 13c
4. Deletion of the two biggest elements, corresponding to Control Board Sides. 13d

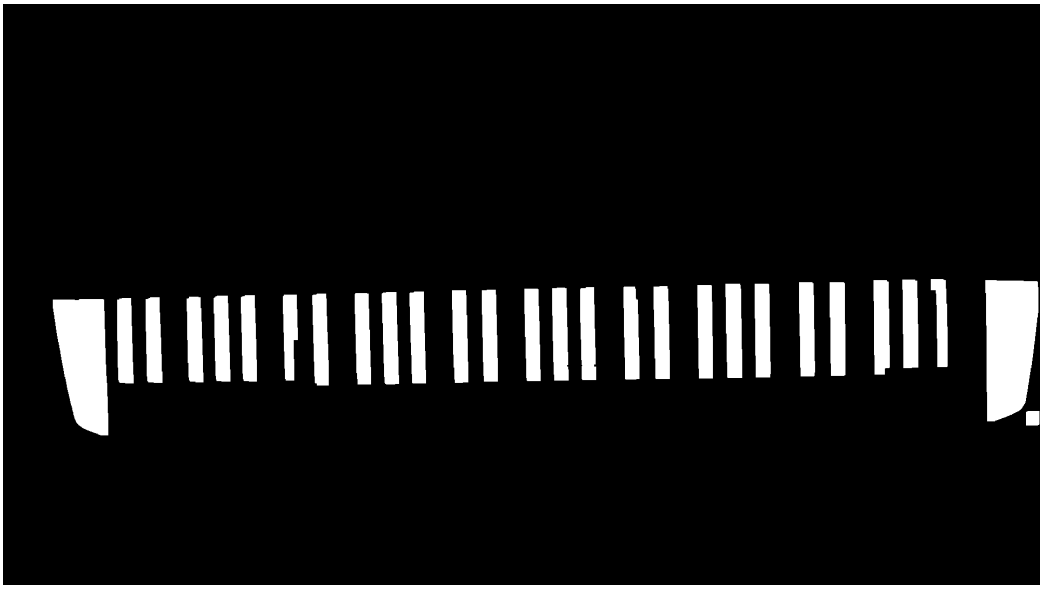
Figure 13: Details on Front Binary Mask Computation Procedure



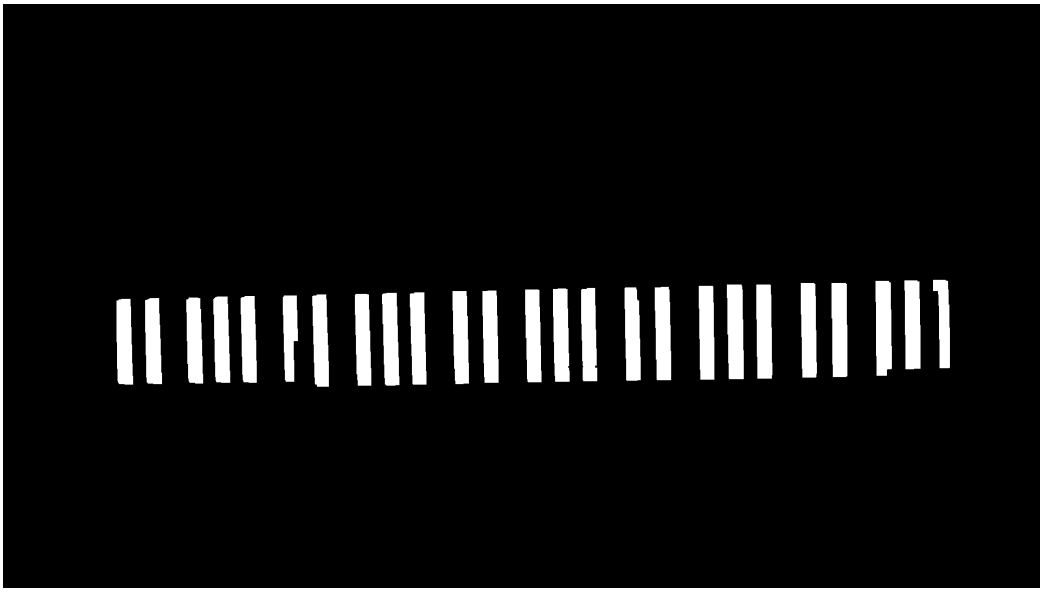
(a) RAW Mask resulting from Binarization and Thresholding

(b) Wider Element Isolation

Final result will have Truth value only in presence of original white key.



(c) Logical Conjunction with Original Mask



(d) Final Black Key Mask

4.2 Geometric Features Computation

Geometric Features refers to "Vanishing points", "Joint and Ending Lines" and "Ratios". It can be noted that the first two main assumption on Cameras' position and orientation stated in Section 2.2 are no longer valid during Script Execution. In order to make the algorithm use theoretical rules stated

in Section 3.3 some interpolation must be made on Keyboard Masks.

4.2.1 Front Features

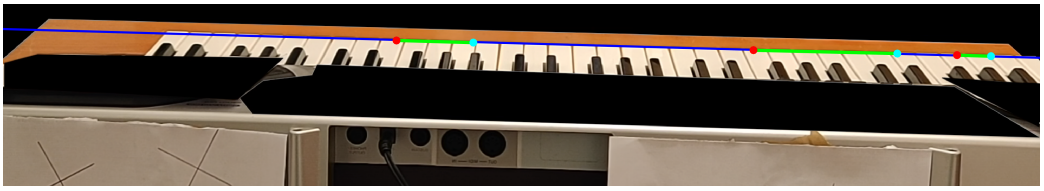


Figure 14: Detail on White Keys Horizon line, showed in blue. Green segments represent `hough lines` with extrema marked with red and cyan circles.

White Keys Horizon Line An useful representation in Homogeneous Coordinates have been computed by mean of `Hough Transform` MATLAB® function. Search was done on White Keys Mask's perimeter confined with parameters:

- `Theta` = `[-90° : 0.5 : -75]` [4]
- `Rho Resolution` = `1`

Processing Hough Matrix with `houghpeaks` and `houghlines` three long segment are obtained. Those segments are interpolated in order to obtain an unique line. Figure 14 shows resulting line and generating segments.

Confirming setup's non ideality (as well as Camera distortion) note that resulting line does not match perfectly White Keys' Horizon when arriving to Keyboard side. However introduced discrepancy does not interfere much with final results.

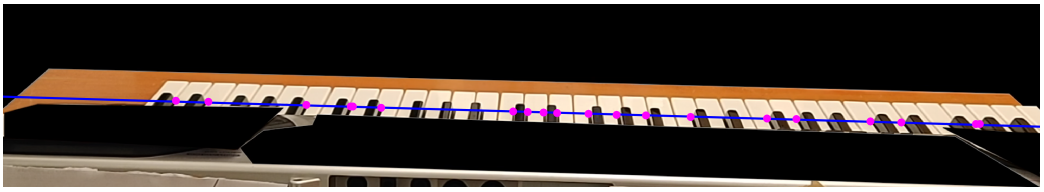


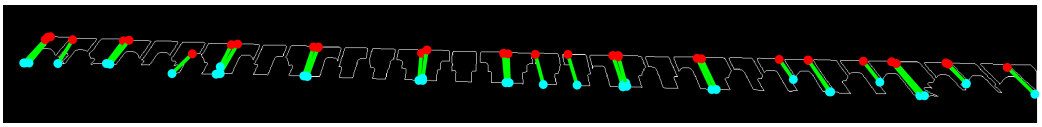
Figure 15: Detail on Black Keys Horizon line, showed in blue. Magenta markers identify points that have been interpolated.

[4] note that `Theta` angle reference is on top x axis and grows clockwise

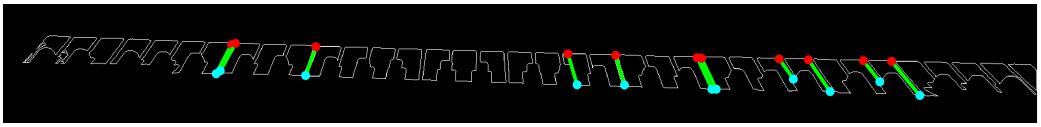
Black Keys Horizon Line Some particular points, belonging to Black Keys Mask where perimeter joins to White Keys Mask have been took by `MATLAB®` picker. [5]

Horizon Line's homogeneous representation was computed by averaging lines obtained by from points cross products. Figure 15 shows resulting line and marks averaged points

Figure 16: Details on Front Binary Mask Computation Procedure



(a) All Computed Hough Lines



(b) Filtered Hough Lines

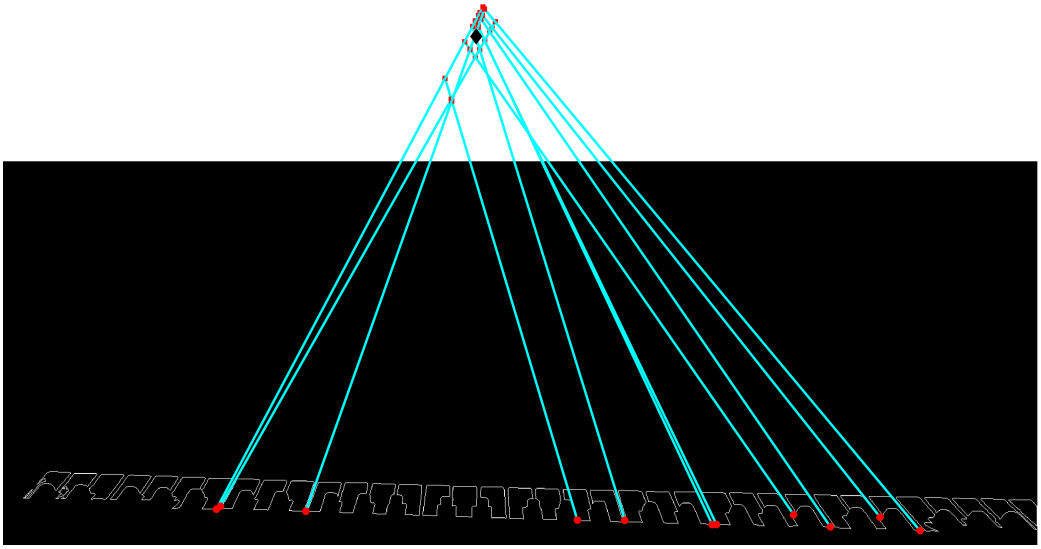
Vanishing Point Vanishing Point Homogeneous representation is computed by exploiting lines lying on "*Keys' Longest Side*" Since in real world those segments are parallel in image plane lines lying on them will intersect on Vanishing Point. However, setup non ideality and camera intrinsic distortion forces to find a number of segments in order to interpolate in between resulting intersection points. Procedure results are illustrated in Figure 16

1. 16a `Hough Transform` with parameter:
   - `Theta` = `[-45 : 0.5 : 45]` [6]
   - `Rho Resolution` = `1`
2. Noise reduction by joining segments closer that `5px` or having coincident extrema.

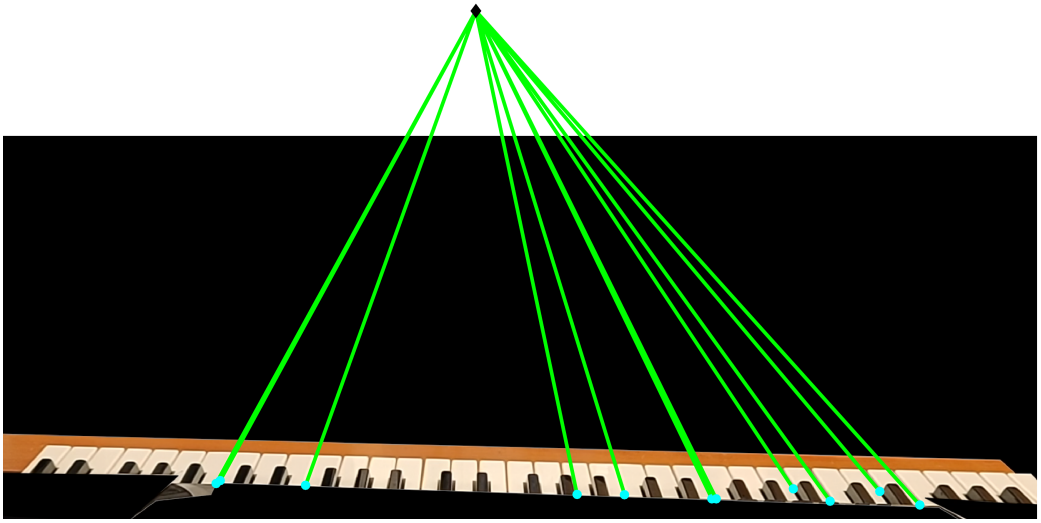
Not all resulting lines can be produce reliable intersection points: some segment couples have been proved to produce intersections that are on the

[5] for code readability, script reports those points as hardcoded in a matrix `P`

[6] note that `Theta` angle reference is on top x axis and grows clockwise



(c) Vanishing Points obtained from pairwise cross products. Black diamond expresses averaged -and Final- Vanishing Point



(d) Detail of Final Vanishing Point

opposite side *w.r.t.* intersection of lines on keyboard's sides. Therefore lines that are roughly vertical must be deleted.

3. 16b Remove lines having:

$$\texttt{-40} \leq \texttt{theta} \leq \texttt{-15} \text{ or } \texttt{15} \leq \texttt{theta} \leq \texttt{30}$$

4. Divide Lines in "left" and "right" side sets; a.k.a. $\texttt{theta} \lesseqgtr \texttt{0}$.

5. Pairwise cross product of each line in left set to right set. 16c
6. Average of obtained point. 16d

4.2.2 Upper Features

Horizon Lines Both Black and White Keys were computed by mean of `Hough Transform` function. Search was done on White Keys Mask's perimeter confined with parameters:

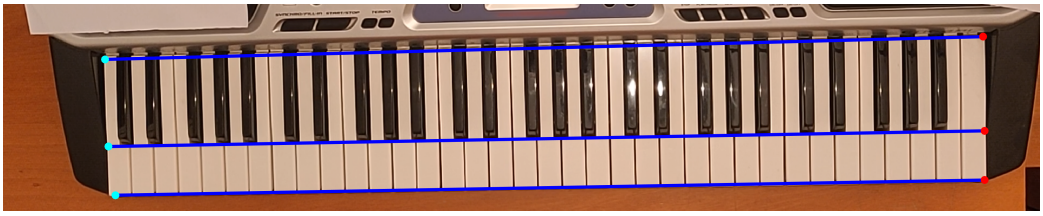


Figure 17: Detail on Upper Lines. From top to bottom: Keys Joint, Black Horizon and White Horizon Line

- `Theta` = `[80° : 0.5 : 89]` [7]
- `Rho Resolution` = `1`

Hough Matrix processed with `houghpeaks` and `houghlines` -with additional parameter `FillGap` set with an huge value of `103`- returns three long lines. This is due to the proximity of Keyboard to the Upper Camera, that combined to noise reduction process of Mask Computation described in section 4.1.2 segments are brought to be more aligned rather than Front Camera's ones. Results are shown in figure 17

The third line computed is the one representing Joint of Keys to Control Board needed for Finger Depth computation. Details in Section 3.2.2.

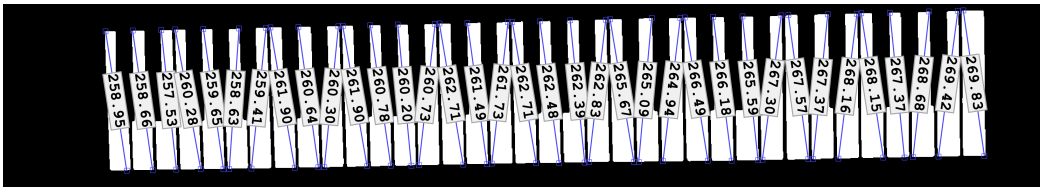
Pixel to World Ratio In the ideal setup "millimeter to pixel" ratio can be computed from known measurements:

$$\texttt{ratio} = \frac{\text{know\_measure } [\texttt{mm}]}{\text{know\_measure } [\texttt{px}]}$$

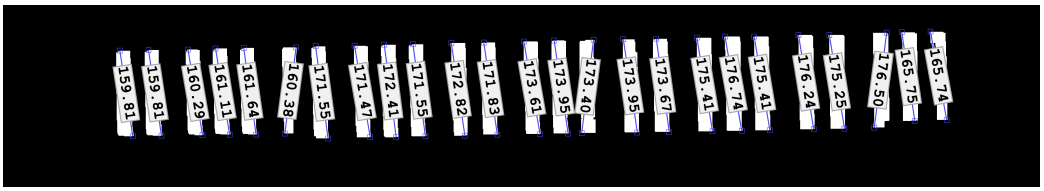
A reliable measure can be identified from both real and image spaces as a Key diagonals. This measure is less prone to noise in image plane since can

[7] note that `Theta` angle reference is on top x axis and grows clockwise

Figure 18: Diagonals computed



(a) Detail on White Keys diagonal pixel measurements. Averaging returned $263.4364px$, compared to $136.7648mm$



(b) Detail on Black Keys diagonal pixel measurements. Averaging returned $170.0124px$, compared to $84.6821mm$

be computed as the distance of the two farthest points of each key. Noise can be even more reduced by computing the convex hull of each key getting a smoother contour.

In order to handle setup non ideality diagonals from all White or Black Keys have been averaged together and confronted to corresponding real world measure. Figure 18 shows all measurements computed in pixels. Ratios for White Keys and for Blacks are then computed as:

$$\texttt{WHT} - \texttt{ratio} = \frac{136.7648mm}{263.4364px} = 0.5192$$

$$\texttt{BLK} - \texttt{ratio} = \frac{84.6821mm}{170.0124px} = 0.4981$$

Final Ratio is computed by averaging White and Black Ratios obtaining a value of:

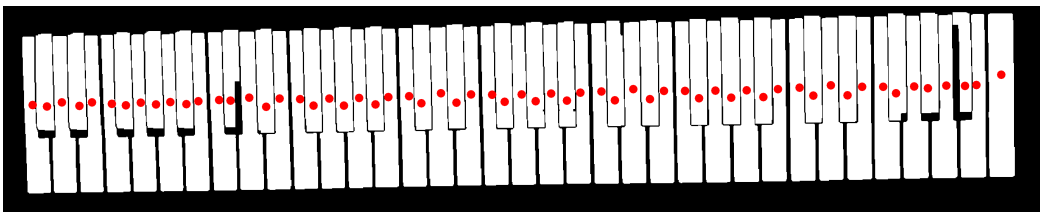
$$\texttt{Final} - \texttt{ratio} = 0.5086$$

4.3 Keyboard Centroids

Centroid concept summarize shape of Keys during the whole script execution; centroids are a pair of $[x, y]$ coordinates put in strategic points of each area. From either Upper and Front Frame those coordinates are identified starting from both Black and White Binary Masks:

(a) Detail on Front Centroids. Green Markers shows Centroid lying on Joint Line while Reds the ones lying on Black Horizon Line.

Figure 19: Diagonals computed



(b) Detail on Upper Centroids

Upper Frame: All Centroid where put so that they formed an horizontal line passing though the "Keyboard Horizontal Symmetric Axis".

Front Frame: In order to make *Tips Matching* consistent with Upper Frame (see Sec. 5.3) there where necessary to put two different centroids for each key. In particular:

- White Key: One centroid placed on middle of short-lower border -that is Keys' Joint Line to Control Board-, while the second is placed on the 2D center, a.k.a. the "center of Mass".
- Black Key: One centroid placed on middle of short-lower border, while the second is placed middle of short-upper border -that can be approximated as the Black Keys Horizon Line-.

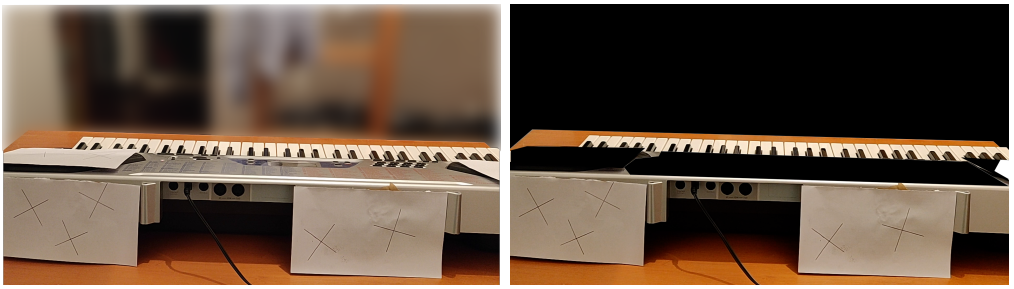
Figure 19 shows all computed centroids.

Since, as stated in Section 3.1 X axis is consistent for both frames, Centroids can be stored into a list sorted by x component so that each index results in the same key

4.4 Backgrounds Processing

Due to light variation during video recording in order to obtain the most reliable features, both background have been computed from a couple of separately recorded videos. Both videos, stored in this repository, are shoots from the same cameras and configured with the same settings as the others; they image the same scene without any human interference.

A preliminar script gathers all frames from each video and averages them on each RGB component.



(a) Complete background with paper artifacts

(b) Background with Processing applied masks

Figure 20: Comparison of Original and Processed Frames

Further processing for Front Background Due to Desk position in the room, light variation made options written on top of control board to glow with different behaviors for each gathered frame. Since this phenomenon interfered with Mask computation It was made necessary to delete those noisy part by applying manual artifacts in setup definition (paper sheets) and post production masks on shoot frames. Figure 20 shows preprocessing effects.

Videos Synchronization

For a consistent execution of Script Routine, both videos have been externally synchronized in term of:

- Starting and ending timesteps
- Frame rate

5 Script Routine

Script Routine aims to convert theoretical rules for Key pressure from ideal case by the application

Algorithm's Routine work "frame-by-frame" gathering corresponding frames from Front and Upper Source; it is composed by some distinct steps:

1. **Finger Tips Extraction**
   Identification of hands, and selection of the top-most extreme of each finger
   Both input sources are handled separately in order to isolate the (from now on) `ROI`s containing hands. Identification is done by mean of *color segmentation* using both RGB and HSV color spaces;
   In each `ROI` Finger Tips identification is done by treating hands' contour as a two-dimentional signal so that *Peaks* can be computed;
2. **Tips Matching** Identified Finger Tips in both videos are paired together;
3. **Matched Tips conversion** Application of Geometric rules defined in Section 3.2
4. **Check for Key Pressure** Application of Rules defined in Section 3.3. Some additional rules and fuzzy threshold had to be added to handle noise.
5. **Update of *Finger Motion Track* structure** As a further, and final method of noise reduction a specific Queue storing point is implemented. This is because Finger Motion has to take at least three frames to complete its route from Pressure to Release. When a Tip is proposed as Candidate to Pressure in the active frame it is inserted queue's top in order to be compared with previous frames. If candidate appears in at least three frames then it is promoted as "Pressing Tip" and output as a result. Queue has a maximum length of five frames.

5.1 Hands Segmentation

Procedure states as follows, Figure 21 shows results for both frames:

1. Color Thresholding is applied to both RGB and HSV color spaces.

   Resulting in two distinct binary masks for each color space

2. Binary addition of resulting masks

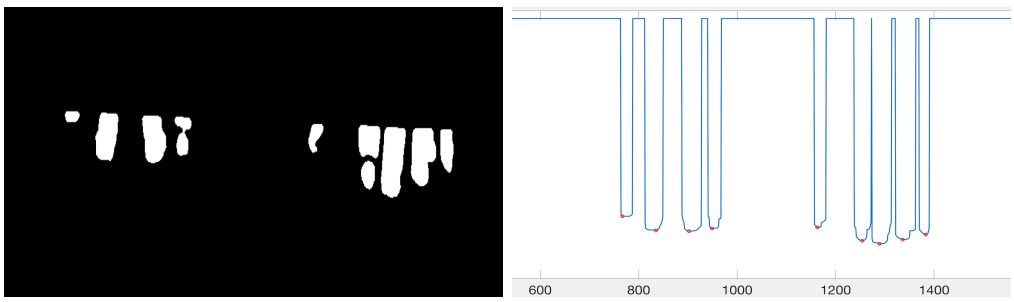
(a) Front Frame (b) Upper Frame

Figure 21: Details of Hands Color Segmentation Process

Front Frame		**Upper Frame**	
RGB	HSV	RGB	HSV
$175 \leq$ `RED` ≤ 235	`HUE` ≤ 0.0565	$175 \leq$ `RED` ≤ 235	-
$65 \leq$ `GREEN` ≤ 140	$0.375 \leq$ `SAT` ≤ 0.65	$85 \leq$ `GREEN` ≤ 165	$0.56 \leq$ `SAT` ≤ 0.75
$65 \leq$ `BLUE` ≤ 140	$0.675 \leq$ `VAL` ≤ 0.9	$65 \leq$ `BLUE` ≤ 140	$0.57 \leq$ `VAL` ≤ 0.775

3. Mask application to frame

5.2 Finger Tip Identification



(a) Finger Cropping in Playing area (b) Identified Local Extrema

In term of `ROI`s, Finger Tips are defined as the "Frontier Point of Distinct Extremities of Isolated Hand". This means that Tips can be identified as a local Minimum (for Front Frame) or Maximum (for Upper Frame). [8]

Procedure states as follows:

[8]Reference is placed as a human would see the image. `MATLAB`® coordinates however has its origin placed on upper left corner and *height* grows downwards.

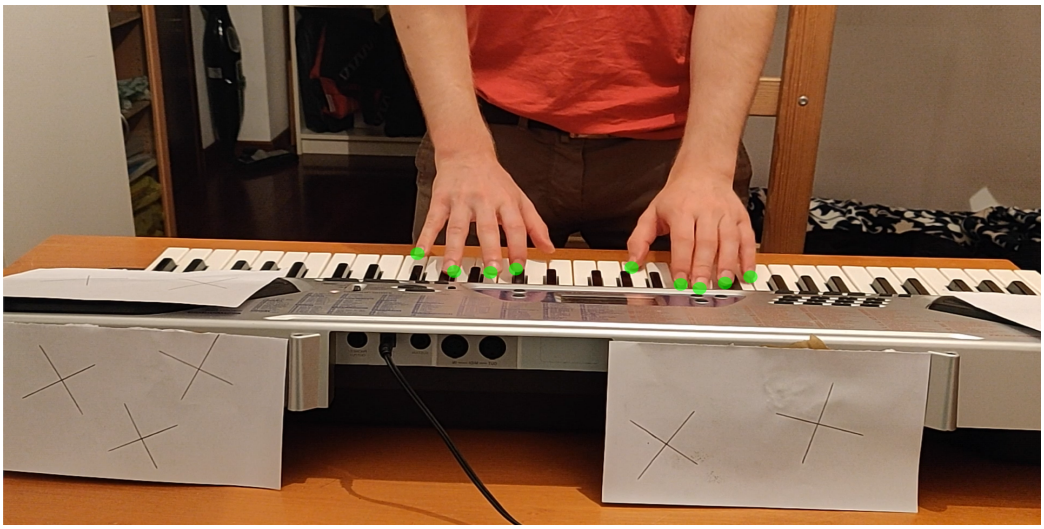
Front Frame

1. Cropping to Keyboard Area: elimination of Fingers that are far from Keyboard Playing Area, by applying previously computed Keyboard Masks to Segmented Hands
2. Perimeter Computation: find Border of isolated area
3. Redesign Binary Image as Bidimensional Signal: for each column of the image the lowermost `True` row is identified as the one with maximum y value.
4. Find Local Peaks by mean of `findpeaks` function

Figure 22 shows procedure's result.

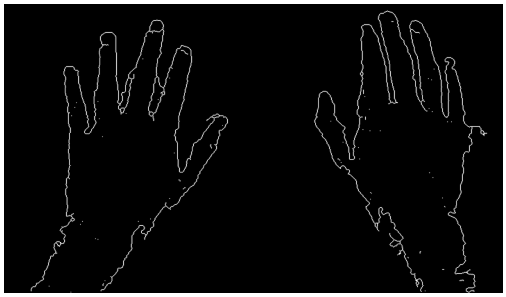
Upper Frame

1. Perimeter Computation Borders are found by mean of a series of denoising operation involving Canny and Sobel edge operators:
2. Redesign Binary Image as Bidimensional Signal For each column of the image the topmost `True` row is identified as the one with minimum y value.
3. Find Local Peaks by mean of `findpeaks` function. Note: results have to be overturn to Original System Image since `MATLAB`® function origin is placed on lower left corner.

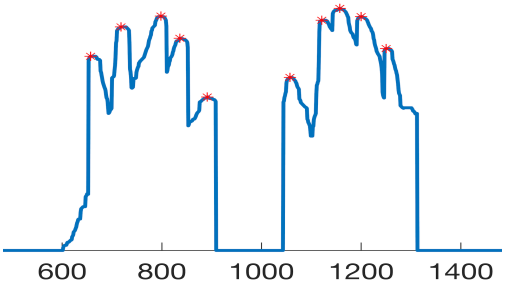


(c) Tips Coordinates Brought back to original frame

Figure 22: Front Finger Tips Identification process



(a) Binary Mask Perimeter



(b) Identified Local Maxima

(c) Tips Coordinates Brought back to original frame

Figure 23: Up Finger Tips Identification process

Figure 23 shows procedure's result.

5.3 Finger Tip Matching

Even if identification process is the same for both frames, results could not be the same. This is due to:

- Front Algorithm cropping some fingers.
- Both `findpeaks` functions returning false positives.
- Fingers that are adherent for most of their length and are recognized as a unique peak.

The Only invariant structure is the fixed order of Keys Centroids: if a

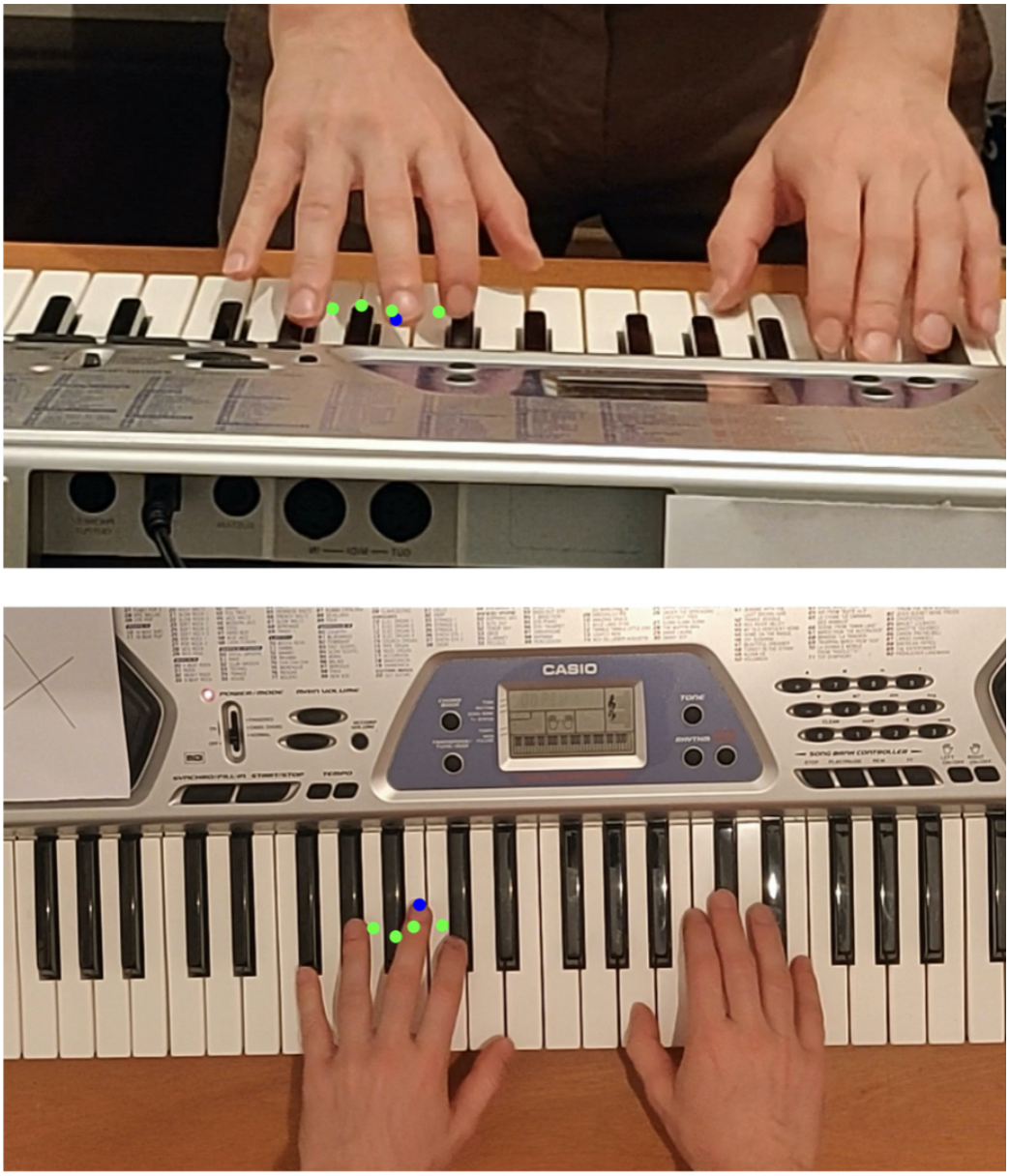


Figure 24: Detail of a single finger tip defined by the blue marker, identified in both frames. Used Keys are the marked in green

finger is near a Key in a Frame it has to be near the same Key in the other Frame. Distance is computed via Euclidean measure. Since Front Frame lacks in depth of the real world some front keys can me mismatched. In order to avoid this effect, Front Video is provided by two distinct keys structures, INF and SUP, storing for each key different centroid coordinates computed

Figure 25: Detail of a single finger tip defined by the blue marker, identified in both frames. Used Keys are the marked in green

at superior and inferior extrema of each key. For each Key, proper structure is chosen depending on finger position *w.r.t.* Black Horizon Line.

Figure 24 shows Centroid used for matching one key, while 25 shows all matched keys:

6 Final Results and Comments

Input Source Video are composed by 97 frames, which only 15 of them pictures the "pressure of a Key":

- 8 Black Keys Pressure
- 6 White Keys Pressure

Script outputs 19 Pressures in total:

- 14 Black Keys Pressure, which:
  - 8 are actually correct (True Positives)
  - 3 miss prediction (False Positives)
- 5 White Keys Pressure
  - 5 are actually correct (True Positives)
  - 1 miss prediction (False Negative)

Black Keys miss prediction are due to numerical error gathered though frame computation and frame-by-frame relations. Those actually take place when finger's dynamic passes over a black key within a close height triggering rule 3.7.

White Key miss prediction happens because of Front Frame setup: Control board crops final part of Finger, leading to Identified Tip to be "higher" than necessary. In this situation Tip projection will much deeper, and therefore rule 3.6 will not be triggered.

There are a variety of aspects that introduces numerical errors:

Setup non ideality As said before, some measurements averaging where implemented to correct Cameras placement. A more accurate positioning would have made avoided those kind of errors, like:

- " Front Camera cropping some part of Keys Joint Line
- " Upper `Pixel_to_mm` ratio

"Known measurements" Of All measures provided in section 2.3 only the"Keyboard's Micro" and "Desk Macro" are perfectly accurate. All other measures had to be inferred in various ways, due to abrupt approach changing time after having all setup dismantle. Most wakened measures where

- ” Front Camera Height
- ” Front Camera to Desk Distance
- ” Keyboard to Desk Border Distance

With consequent distortion of α and β angles, from which *Principal Viewing Ray* where computed.

Skin segmentation Due to abrupt light changes in the shooting room skin detection algorithm had to be fine tuned to get the most accurate average result; this implies that some of hand border where eroded, therefore some finger tips where not as accurate as they would have been.

As a final note, source video terminates with key pressure by right middle finger. This pressure will not be recognized by the script due to the small number of previously identified Candidates already present in the queue.