Foucault Pendulum Tracking System Using a Raspberry Pi 4

Claudia Raffaelli
Department of Information Engineering
University of Florence
Via di Santa Marta 3, Florence, Italy
claudia.raffaelli@stud.unifi.it

Francesco Scandiffio
Department of Information Engineering
University of Florence
Via di Santa Marta 3, Florence, Italy
francesco.scandiffio@stud.unifi.it

***Abstract*—The aim of this paper is to describe the implementation of a real-time tracking system developed to detect the centre of a moving Foucault pendulum. Since tracking can be applied in many ways, we will describe the failed approaches and the reasons why Template Matching has proved to be the best one for the context under consideration. The system hardware consists of a Raspberry Pi 4 and a Raspberry camera, devices certainly less powerful than a modern general purpose computer: for this reason the optimization of the code and a wise use of the available resources played an important role in the realization of the project. In order to deliver a program whose performances were the highest as possible, the code has been developed in C++ and a multi-thread approach has been employed.**

***Keywords*—Foucault pendulum, tracking system, Raspberry Pi, Template Matching, multithreading, C++.**

I. Introduction

Video tracking has become an useful process widely employed in many fields, such as surveillance, autonomous driving, human-computer interaction and much more [9]. For example, video tracking can be particularly useful for tracking the movements of an object in order to analyse its motion. Anyway, there is a close relation between video tracking and detection. The first one enables to trace the same object moving throughout a video, labeling it as needed, while the latter aims to locate a specific object within an image. A naive way to perform tracking is to apply a detection algorithm to each frame of a video, that is what has been performed in the proposed work. More sophisticated ways to obtain the same result usually achieve recognition through a classification process. However, the available hardware plays an important role in choosing the method which best fits the developing context. In this case the program had to be light enough to run on a Raspberry, a device with significantly less resources than modern general purpose computers. The Raspberry Pi is a small single-board computer widely employed in projects that require portability or have a limited budget.

A Foucault pendulum is a long pendulum suspended above the ground and set into planar motion by the spinning movement of the Earth (Fig. 1). This device was in fact introduced as an experiment in order to prove the rotation of our planet. In this case, we consider a modified Foucault pendulum to which an electric circuit capable of producing an electromagnetic field has been added. The purpose of this addition is to return to the pendulum the energy lost due to air friction, allowing it to oscillate and rotate without ever stopping.

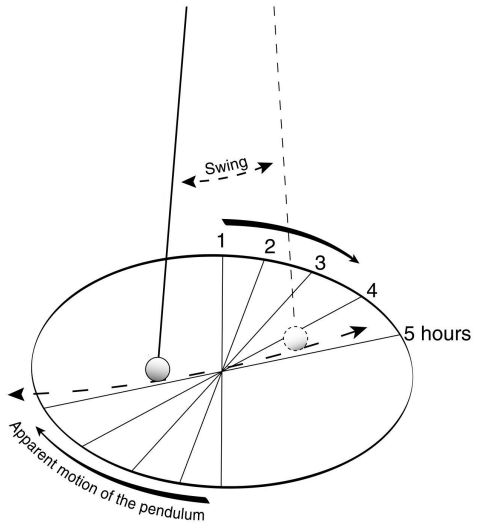


Fig. 1: Foucault Pendulum diagram [21].

The aim of the project is to develop a tracking system that could continuously and accurately extract the coordinates of the pendulum, making it possible to analyse its motion.
This paper is organized as follows. Section II provides information about the hardware used for the project and the system requirements. Section III describes the tracking algorithms we tested and illustrates the results obtained by each one of them. Section IV is about implementation shrewdness, such as multithreading and prospective correction. Finally, in Section VI we draw the conclusions and propose ideas for future developments.

II. System Model and Requirements

In the two next subsections is given a basic understanding of the model adopted. The *hardware* subsection takes care of the physical tools involved in the realization of the system. Even though the actual building of the model goes out of the scope of this paper, having an idea of the measures and structures that make up the whole system, can deliver a more technical view, which allows a deeper understanding of the overall operation. On the other hand the *system requirements* subsection provides an overview on the underlying software principles upon which the model should be shaped.

A. Hardware

- The structure that sustains the pendulum itself.
- The hardware used for the tracking system, which is, as already mentioned, make up of a Raspberry Pi and a Raspberry camera.

In Fig. 2 is displayed the basic structure of the pendulum support.

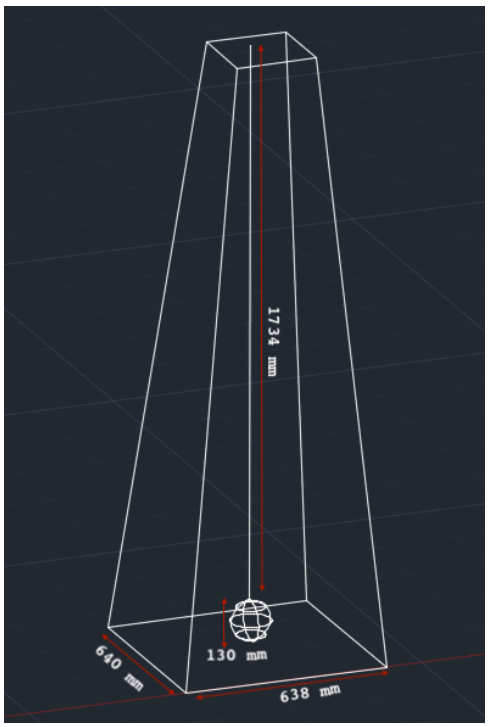


Fig. 2: Physical schema of the pendulum.

The actual pendulum is a cast iron sphere held suspended upon the ground by a long wire. The pendulum is equipped with a coil, capable of generating a magnetic field. In comes into operation when the pendulum starts to swing, and it guarantees that the pendulum can oscillate indefinitely.

What is important to notice is that the camera, used for the tracking system, is positioned on the left side of where the string is attached. Moreover, the camera is slightly toed-in toward the pendulum. This information is crucial in order to understand why it was deemed necessary to carry out a perspective correction, explained in section IV-B.

The Raspberry Pi used for the task of tracking is a Raspberry Pi 4. Among the other specifics, what is useful to know in the context of this task, is that it comprises of four cores. This information will prove itself valuable while discussing the multithreading section (IV-C).

The Raspberry is connected to a 5 megapixel camera. The video stream acquired from the camera, though, for the task required, has been set to a resolution of 640×480. After all, an excessively high resolution frame was not vital in order to allow a proper recognition of the pendulum center. As a matter of fact, having to deal with less pixels made possible to speed up the whole process. According to the specific of the camera, with this kind of resolution, it was possible to make use of up to 90 frames per second. However, since a real-time processing was needed, it has been proved mandatory to comply to the frame rate that the system could allow, as is explained later in the previously cited multithreading section (IV-C). Also the camera provided is of type Rolling Shutter [23]. In this kind of cameras, each frame is not captured by taking a snapshot of the entire scene at a single instant in time, but rather, by scanning across the scene rapidly. This peculiarity causes the fact that, since not all the pixels in the image are recorded at the same time, some distortions can be originated by this behaviour and, above all, with fast moving objects.

B. System Requirements

The purpose of the program is to trace and save the movements of the pendulum so that its motion can be analysed. The coordinates must therefore be extracted precisely and quickly enough to have an adequate number of points for each oscillation. Since the pendulum takes approximately 2.7 seconds to complete an oscillation, the lower limit has been set to at least 4 frame to be processed at each second. Moreover, the software has to draw a real-time graph of the pendulum position, therefore the performances must be such that the frames do not accumulate over time. Even the given hardware affects the developing of the system. The use of an elaboration unit such as a Raspberry Pi requires a wise use of the available resources, especially of the processor: in fact is known that this kind of device easily reaches Thermal Throttling point[1], reason why it is a good practice to avoid overloading the CPU [1]. Lastly, since the camera is decentralized and titled with respect to the pendulum oscillation plane, it is essential to evaluate end correct possible perspective distortions.

III. Tracking algorithm

The fundamental element of this project is surely the algorithm of recognition and extraction of the pendulum coordinates. The automatic identification of figures is a subject widely discussed in literature, especially in relation to the image segmentation techniques employed in the field of Machine Learning [24].

As discussed in the previous section, the software must be accurate and fast in analysing the frames and extracting the pendulum coordinates. The programming language used is the first element that influences the execution performance. For this project, C++ language has been selected because of the speed that distinguishes it from other languages, such as Python and Java [22]. Since a tracking system can be developed in many ways, an empirical selection process has been applied in order to identify the most suitable recognition technique for the context under consideration.

[1]Thermal Throttling is the phenomenon in which the heat produced by a device increases the temperature of the device itself until it causes a reduction in the working frequency of the CPU.

A. Detection by colour extraction

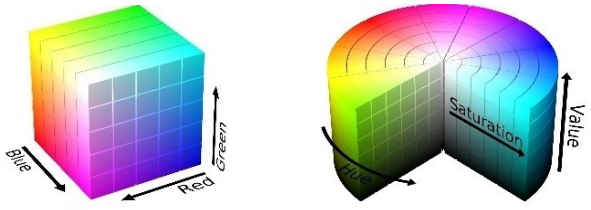
The first attempt of developing a tracking method is based on image segmentation by colour extraction. The idea is to place a coloured marker on the center of the pendulum in order to find its position by filtering the incoming frames and excluding everything that is not of the selected colour. The filter is realized through *Thresholding*, an image processing technique which provides a binary segmentation of the Input image, i.e. it generates a black and white image in which the white pixels are all and only related to the content of interest [2]. The thresholding operation is carried out by comparing each input pixel with a `threshold` value through an appropriate formula. For example, considering a grayscale image[2] and assuming the `threshold` as known, the Output is obtained by applying (1) to the Input image.

$$Out_{pixel} = \begin{cases} 0 & \text{if} \quad I_{pixel} < threshold \\ 1 & \text{if} \quad I_{pixel} \geq threshold \end{cases} \tag{1}$$

The formula to be used for pixel comparison must take into account the colour space of the image, that is the pixel coding method and the map that associates each colour tone with its unique code. The `threshold` value depends both on the colour space and the elements of interest to be extracted, so an appropriate choice requires information about the content of the image to be analysed. In this case we have chosen a red marker because this colour is not present in the rest of the image and it provides a good contrast of tones with respect to the mainly black background. The aim is to create a filter that excludes everything that is not red.

OpenCV library encodes the frames acquired by the raspberry camera according to Blue-Green-Red model (**BGR**). Each colour is therefore uniquely identified by a triplet of integers with values in the range $[0, 255]$, each of which indicates the colour intensity of the respective channel. Although the BGR model is still used for the representation of images on electronic devices, colour *Thresholding* operations are more effective and robust when applied to images encoded with the **Hue Saturation Value** colorspace (HSV) [4]. In fact, while BGR colours are defined by adding different amounts of Blue, Green and Red, the HSV space separates the information about the colour **Hue** from the information about its intensity (**Value**) and saturation. The idea behind this kind of separation of information is that changes in the brightness induced by shadows and reflections are supposed to not modify the Hue.

[2]In a grayscale image each pixel is an integer value in range $[0, 255]$ following the convention for which 0 is black while 255 is white.



(a) GBR color cube. (b) HSV cylinder.

Fig. 3: Three-dimensional representation of BGR and HSV color spaces.

Source: OpenCV documentation

As shown in Fig. 3, HSV colour space is a cylinder and the Hue value is expressed in degrees. Notwithstanding the HSV standard spreads the colours over a full circle, OpenCV uses only half of it, so colours are encoded in the positive range of $[0, 180]$ degrees [16]. By definition, the origin of the circumference of the Hue is positioned on the red, with the result that this colour is contained in the region $[-10, 10]$, as shown in Fig. 4.

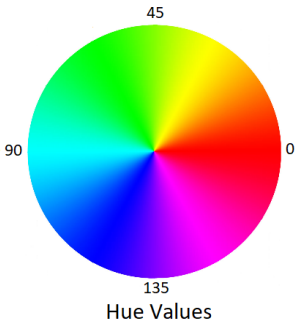


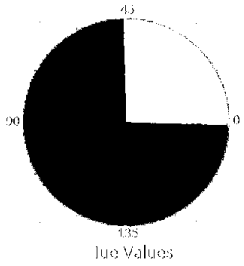
Fig. 4: OpenCV Hue color wheel: the hue values goes from 0 to 180 degrees. The origin is placed in the middle of the red colour.

Source: OpenCV forum

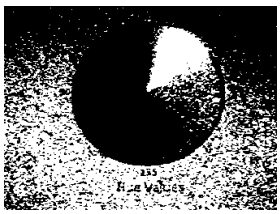
From what has been said about the range of the hue wheel, it is not possible to use negative values as a reference. For this reason an ideal filter on $[-10, 10]$ corresponds to two cascade filters as follows:

$$(0 \leq hue \leq 10) \vee (170 \leq hue \leq 180)$$

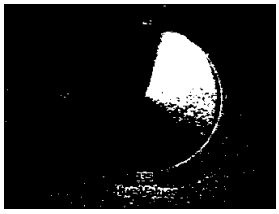
What has been said so far deals with the problem from a purely theoretical point of view. In the real application, in fact, brightness alterations also affect the hue value. This also happens because non-professional cameras, such as low-end webcam and the raspberry camera, are sensitive to brightness change and this has repercussion for the way colour is perceived. The following experiment provides evidence to the previous statement. A filter has been applied to the same input image under three different conditions: the first time it was loaded into the program as a file, the other two it has been printed and framed with the camera in well-lit and low-lit conditions respectively. The filter, chosen for demonstration purposes only, is low pass on hue values in the $[0, 45]$ range.



(a) Thresholding directly applied to the image file.

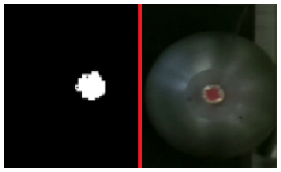


(b) Threhsolding applied to the image extracted by the camera with low light conditions.

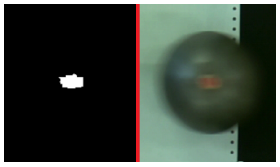


(c) Thresholding applied to the image extracted by the camera with good light conditions.

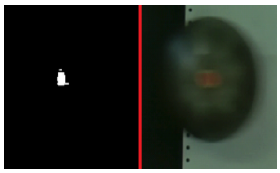
Fig. 5: Thresholding applied on the same image in different light conditions.



(a) Thresholding applied to the pendulum at the inversion point of an oscillation.



(b) Thresholding appliend to the pendulum in motion and in a well-lit condition.



(c) Thresholding applied to the pendulum in motion when it is in the darkest zone of the frame. The area is dark due to the shadows produced by the pendulum structure.

Fig. 6: Thresholding technique applied to different frames depicting the pendulum in motion.

The results of the experiment show that as the brightness decreases so does the amount of correctly identified colour while the noise increases (Fig. 5b). Moreover, we can observe that even in well-lit condition (Fig. 5c) the colour perception of the camera is different from the *ground truth* depicted in Fig 5a. We can therefore infer that only in an experimental way it is possible to identify the most suitable parameters to achieve the objective. After field trials, the following values have been selected:

$$Out_{pixel} \begin{cases} 1 & (140 \leq hue \leq 179) \wedge \\ & (139 \leq saturation \leq 255) \wedge \\ & (120 \leq value \leq 255) \\ \\ 0 & otherwise \end{cases} \quad (2)$$

In order to extract coordinates from a group of pixels, some kind of centroid calculation algorithm is required. However, tests have highlighted several issues related to the accuracy of the results obtained through *Thresholding* (Fig. 6). The first problem encountered was about **false positives**: some screws present in the plane at the base of the pendulum were in fact recognized as red depending on the angle of incidence of sunlight. The problem has been effectively solved by covering the screws with black duct tape. As previously discussed, also brightness reduction has a negative impact on colour recognition. This means that shaded areas or poor lighting conditions directly affect the quality of the output. Keeping the room lights on has mitigated the problem but has not completely solved it. In fact, there was a case where the shadow zone produced by the pendulum structure itself reduced the amount of pixels correctly identified, as shown in Fig. 6c.

Lastly, Fig. 6b shows the effects of a rolling shutter camera when employed to frame a moving object. As we can observe, the centre of the pendulum is deformed, with the consequence that the thresholding result is also deformed.

As far as performance is concerned, the Thresholding proved to be very fast, allowing us to analyse about 25 frames per second. However, following the qualitative analysis illustrated above, we have decided not to implement a method for extracting the centre of the white pixel group. It is in fact obvious that if the group of white pixels changes in number and shape over time, i.e. does not perfectly recognise the centre of the pendulum, the extracted coordinates will not be 100% accurate. We therefore preferred to test further methods before refining the *Thresholding* technique.

B. Circle Detection

Circle detection plays a prominent role in a large variety of applications, taking part in a wide number of image processing algorithms. To name a few, circle detection finds a variety of uses in biomedical applications, ranging from iris detection to white blood cell segmentation. It can be also used for automatic inspection of manufactured products, aided vectorization of drawings, target detection and more [5]. A solution based on circle detection can be applied even in this case. The idea behind this line of work is the following: since the shape of the pendulum is a circle, the pendulum itself can be located by searching throughout the image for a circle. In order to implement this strategy, each frame extracted from the camera needs to follow the preliminary pipeline listed hereafter:

1) As first thing the frame is converted to the gray-scale color-space. The gray-scale conversion allows a faster

computation of the next phases, since the operations are applied to a single channel instead of three.

2) As second point, a Gaussian Blur filter is applied to the image using the OpenCV `GaussianBlur` function[13]. The aim of this step is to smooth out unnecessary details of the image. Blurring is in fact an important preparatory stage that ensures a good final outcome.

The most common approach while detecting circles, as well as any other geometric shape, is to first detect the edges of the objects in the image, i.e. find the points for which there is a sharp change of color. As stated above, smoothing the Source image is an essential requirement for an edge detection task, because of the fact that most edge-detection algorithms are sensitive to noise. Using a Gaussian Blur filter before edge detection aims to reduce the level of noise in the image thus, simplifying the concept, it removes some edges to only keep those which are supposed to be the most relevant [11].
Circle detection is usually done using the Circle Hough Transform (CHT), a technique used for identifying the locations and orientations of certain types of features in a digital image [20]. In the beginning this transform was concerned with the identification of lines in an image, but later it has been extended to identifying positions of arbitrary shapes, or as in this case, circles.

A circle in R^2 can be described by (3):

$$(x-a)^2 + (y-b)^2 = r^2 \tag{3}$$

where (a, b) is the centre of the circle and r is its radius. If the radius is fixed (as in this case, since the pendulum radius is known) the parameter space is reduced to the position of a circle center. The algorithm follows the next steps:

- An *accumulator* is created. It is a two-dimensional array which is made up of a cell for each pixel. All of its cells are initialized to 0.
- For each edge point (x, y), it can be defined a circle centered at (x, y) with radius r according to [3]. Among those edge point, there will also be the points belonging to the original circle.
- The intersection point of all such created circles in the parameter space is indicative to the center point of the original circle. The accumulator matrix is used for tracking the intersection point.
- When a new circle centered in (x, y) is traced, for all values of a that satisfy (3), all possible values of b are found. The values at the position (a, b) in the accumulator matrix are then incremented.
- The local maximum point is searched in the accumulator space. The position (a, b) of the maxima would be the center of the original circle, errors excepted.

Figure 7 shows an example of the described process.

Although this method seems to work well, it has its limitations. Much of the efficiency of the Hough transform is dependent on the quality of the input data. The shapes edges must be well detected in order for the method to work. As first, the function has been applied to daily life circle shaped objects

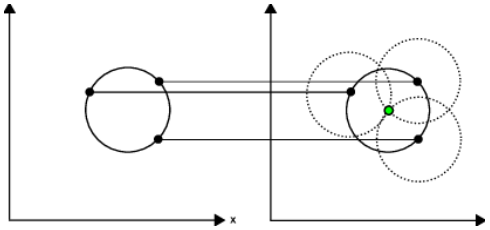


Fig. 7: Circular Hough Transform example [20].

in movement, with good results. It was then applied to a still image of the pendulum, again with a good outcome. The result of this last trial is reported on Fig. 8. However, transferring

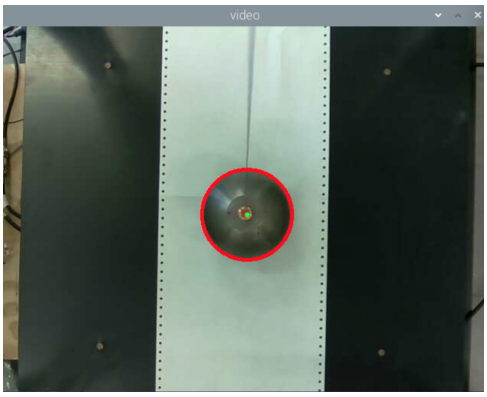


Fig. 8: Circle detection on still pendulum frame

this method on the video stream originated from the Raspberry camera, led to frames in which the pendulum edges were not as well defined as in the fixed frame. This phenomenon is due to the fact that the camera used for the extraction of the frames is of type **Rolling Shutter** as mentioned in the hardware section II-A. Since the pixels contained in a single frame are not entirely captured at the same time, some distortion in the image can be produced, especially while working with fast moving objects. For this reason, although in each of these frames is still applied the Gaussian blur for edge detection, this was not sufficient to provide edges that could be interpreted as circular edges.

As a further experiment, the circle detection method was used jointly to the thresholding already described. This variant applies Circle Detection to the black and white frames produced by Thresholding. Again, the results were unsatisfactory. In fact, as discussed in Section III-A, the group of white pixels is subject to deformation and cannot be detected as a circle. Given the results, it was decided to explore a different approach.

C. Template Matching

Template Matching is an image processing technique widely employed in the field of deep learning, specifically for facial and small objects recognition. Basically, recognition is achieved by searching within an Input image for the group of pixels that is most similar to the content of an another image

used as reference, the **Template**. The intuitiveness of this brief description gives the algorithm a conceptual simplicity that is closely related to human perception of the world. Since our childhood we have been practicing daily the ability to visually compare objects and figures by operating a complex but extremely rapid process of analysis of shapes, positions and colours. However, the concept of image comparison must be defined mathematically so that it can be implemented in a software.

In Signal Theory, the cross correlation coefficient is widely employed as similarity measure between signals. Since images are two-dimensional signals, this is also applied in the field of image processing. Consider an Input image $I(x, y)$ of size $M \times N$ and a Template $T(x, y)$ of size $K \times L$ where $K \leq M$ and $L \leq N$. The correlation between the two images at a point (x, y) is given by (4) [7].

$$C(x,y) = \sum_{i=0}^{K-1} \sum_{j=0}^{L-1} T(i,j) I(x+i, y+j) \tag{4}$$

In the Formula above the sub-signal T realizes what in computer science is called a **sliding window** or **mask filter**. The Template is superimposed on the Input image, identifying a region where their pixels overlap. A coefficient of similarity between the two overlapped regions is computed and the result is stored in an output matrix, at the same location of the current top-left vertex of the examined region. After that, the Template is moved by one pixel and the operation is repeated. Is important to highlight that the comparison between images is only made when the Template totally fits the Source, so T is superimposed on I a total of $(M - K + 1) \times (N - L + 1)$ times. This means that it's more difficult to spot moving objects at the edges of the image because they are more likely to partially be out of the scene. Finally, the maximum (or minimum, depending on the metric implemented) value of the output matrix indicates the location of the best match. Fig. 9 illustrates the sliding window procedure.

The correlation approach involves the correlation coefficient in the computation of the similarity between the template and the target image. However, the basic correlation in Formula 4 is very sensitive to amplitude changes in the images. For example, it is easy to see that if the intensity of the Source doubles, so will the values of C [7] [3]. To prevent this from happening it is good practice to use Normalized Cross Correlation (NCC), resumed in (5).

$$C(x,y) = \frac{\sum_{i,j} T(i,j) I(x+i, y+j)}{\sqrt{\sum_{i,j} T(i,j)^2 \cdot \sum_{i,j} I(x+i, y+j)^2}} \tag{5}$$

In order to precisely identify the most suitable metric, we have carried out a comparison test between different matching coefficients. We have applied Template Matching on a short video depicting the pendulum in motion, each time varying the formula used for the calculation of similarity. The evaluation parameter used in this test is the accuracy of the match, i.e. the

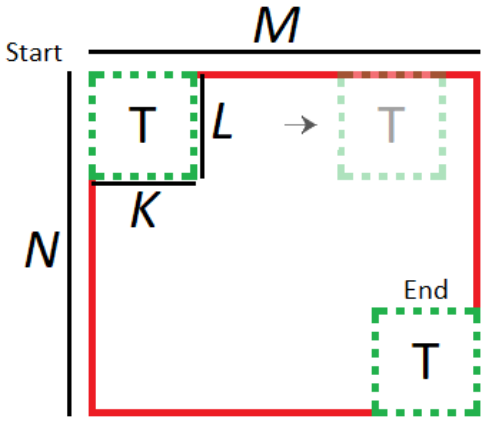


Fig. 9: Sliding window procedure. The sub-image $T_{K \times L}$ is superimposed on the image $I_{M \times N}$. T is moved one pixel at time, always left to the right and top to the bottom. At the end T will have been superimposed $(M - K + 1)(N - L + 1)$ times.

percentage of times the algorithm has correctly identified the center of the pendulum. The results of the test, summarised in Table I, show that the formula that best suits our goal is the **Zero-mean Normalized Cross-Correlation** (**ZNCC**).

TABLE I: Comparison of various similarity metrics between two images of the pendulum. The acronyms in the table are those used in the OpenCV manual [15].

Formula	Match when still	Accuracy (%)
SQDIFF	yes	54.5
SQDIFF NORMED	yes	52.2
CCORR	no	10.1
CCORR NORMED	yes	97
CCOEFF	no	13.3
CCOEFF NORMED (ZNCC)	yes	100

The ZNCC formula is very similar to the Normalized Cross Correlation, where normalization provides more robustness against noise. However, NCC is not invariant with respect to constant changes in brightness. This property can be provided by subtracting from each pixel the mean value of the overlapped region, obtaining (6) [10].

$$R(x,y) = \frac{\sum_{i,j} ((T(i,j) - \mu_T) \cdot (I(x+i, y+j) - \mu_I))}{\sqrt{\sum_{i,j} (T(i,j) - \mu_T)^2 \cdot \sum_{i,j} (I(x+i, y+j) - \mu_I)^2}}$$

$$\mu_T = \frac{\sum_{k,l} T(k,l)}{K \cdot L} \qquad \mu_I = \frac{\sum_{k,l} I(k,l)}{K \cdot L} \tag{6}$$

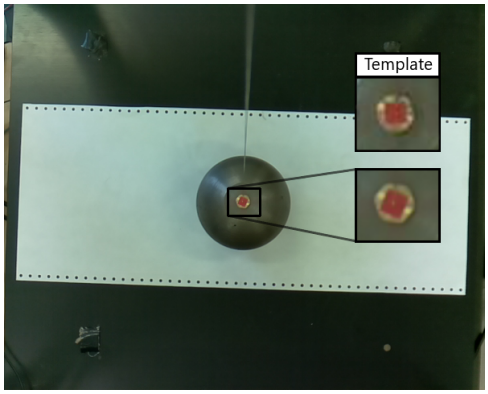


Fig. 10: Template Matching applied to a frame. The area that the algorithm identifies as the best match is surrounded by the black rectangle. On the right of the image, a comparison between the recognized region and the employed Template.

By subtracting the average value it is possible to take into account the brightness level of the entire region being compared: the darker pixels get a negative value, the lighter pixels a positive one. When compared, the product operation gives a negative result, thus excluding the point as a possible best-match. In fact, from normalization follows that the result has values in the range $[-1, 1]$ where 1 indicates a perfect match.

The developed program does not implement any kind of pre-processing on the Source image nor to the Template, such as rotation or size normalization. It is therefore important that the object to be identified has the same dimensions and orientation in both Template and Source. In this project the Template was obtained by cutting out a frame of the pendulum when still, thus selecting only a small area around the center containing the marker. We observed that by reducing the selection to only the red marker, the number of mismatches produced increased, becoming not zero even in the case of ZNCC. In order to avoid that, a small portion of the image around the marker has been included. Fig. 10 shows the Template used and a frame on which Template Matching has been applied. The high accuracy provided by the calculation of the ZNCC coefficient is however linked to a high computational cost. Tests conducted with the pendulum in motion showed that the raspberry was able to process only 5 frames per second. In order to not squander the excellent results achieved in the tracking process, we decided to improve the performance by applying the multi-threading programming paradigm (Section IV-C).

IV. Result production

A. Output content

The output produced by the tracking algorithm is a set of positions recorded while performing the detection operation on different frames. In order to make the results accessible for a later use, the coordinates are saved in a `CSV`[3] file. In this type of document, values are stored in a tabular format i.e. each column identify a data-field while each row is a record. When starting the tracking, the system asks the user if he wants to apply perspective correction to the extracted points. (Section IV-B). If the user refuses, **only one** file is created with the name `year_month_day_hours_minutes_N`. This file contains the coordinates without perspective correction. On the other hand, if the user accepts, a CSV is generated **in addition** to the previous one. This new file has name `year_month_day_hours_minutes_Y` and contains transformed coordinates. In both cases the CSV file structure is as follows:

```
Time; X; Y
```

- **`Time`**. Relative time at which the associated coordinates have been extracted. The measure is relative because it indicates how much time has elapsed since the first detection which coincides to time $t = 0$. This instant of time is expressed in seconds and has an accuracy of 6 decimal place.
- **`x,y`**. Pendulum centre coordinates expressed in pixels. The origin of the axis is placed on the top-left corner of the image.

B. Perspective correction

Perspective distortion is a very important element to take into account when developing a precise and reliable coordinate extraction system. Non-professional cameras introduce distortions that curve the lines of objects in a way that is more accentuated the greater is their distance from the focal axis. Also the position of the camera with respect to the framed scene has a significant impact on the obtained results. A tilted camera alters the perception of the positions of objects and their relative distances, giving rise to the **parallax** phenomenon that is the object position seems different with respect to the background when it is viewed along two different lines of sights (Fig. 11) [18].

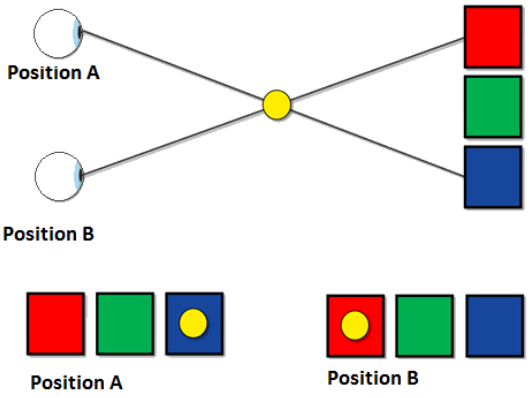


Fig. 11: Parallax phenomenon. Since the focal axis of the observers is not perpendicular to the plane on which the framed object lies, the perceived position of the object changes depending on the position of the observer.

Source: Parallax effect from ionos.com

[3]More info at RFC 4180

This is exactly what happens in the system under examination: the sealing cable of the pendulum did not allow the camera to be vertically aligned with the position occupied by the pendulum when it is idle, so the camera is off-centre and tilted in order to have the focal axis incident on the red marker.

The quickest and most effective method to remove distortions is to calibrate the camera. The calibration process consists of taking a series of photos of a *calibration plate*[4] changing its distance and angle with respect to the focal axis. Starting from these images, specific algorithms are able to determine, therefore correct, the distortion matrix thus the frames can be correctly projected onto the camera viewing plane [12]. Following the spread of Covid-19, restrictions on travel and on laboratories access prevented us from carrying out this operation. Since the absence of a perspective correction mechanism would have negatively affected the accuracy of the extracted coordinates, we developed an application through which is possible to manually correct the effect produced by the tilt of the camera (Section V-C, Section 2.2 of [19]). The idea is to apply a perspective transformation to the acquired frames (Input) so that it removes the effects of inclination, thus obtaining images projected on the plane perpendicular to the focal axis (Output). In order to apply this change, the **transformation matrix** associated with the *Homography* between Input and Output is needed (Lemma 1).

Lemma 1: Given a transformation $T : R^n \rightarrow R^m$ and a point $x \in R^n$, the transformation matrix $M_{n,m}$ associated with T is the matrix such that

$$T(x) = Mx$$

The matrix can be defined through (7) from four pair of points of the type $< P_{in}, P_{out} >$. Geometrically, the points correspond to the vertices of two quadrilaterals, one on the input image and the other one on the output image. The pairs define the correspondences between the vertexes of the two figures, i.e. they indicate how to map each P_{in} on the output frame [14].

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = M \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{7}$$

where $i = 0 \dots 3$ and $P_{in} = (x_i, y_i), P_{out} = (x'_i, y'_i)$

The criteria with which the points has to be placed on the images are as follows. The four points on the input image must be positioned at the vertices of the quadrilateral delimiting the region to which the perspective transformation is to be applied. The quadrilateral of Output is always a rectangle which coincides with the output image itself. The P_{out} points are therefore placed in the corners of the frame, which is why their position depends on the size of the output image. To prevent the transformation from applying distortion to the objects ratio, the size of the output must coincide with the smallest rectangle that can circumscribe the input quadrilateral. Since the actual positions of the input points depend exclusively on the context, the developed program allows to manually select the coordinates using a *Trial and error* procedure: while the user selects them a window shows the result of the transformation, allowing the user to modify the coordinates of the vertices until the desired output is reached ([19]). The input quadrilateral used in this project is the black square at the base of the pendulum. Fig. 12 shows an example of perspective correction.

[4]A plate showing some geometric pattern that can be used to calibrate a camera.

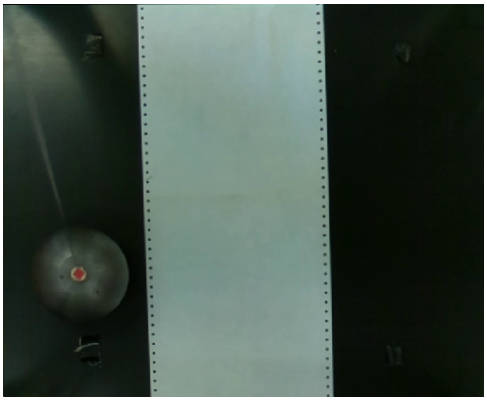


Fig. 12: Example of perspective correction applied to a frame acquired by the raspberry camera.

In order to implement the *Trial and Error* procedure, applying the transformation to the whole image is a necessary step. Anyway, it is a useless waste of resources if done during the execution of the tracking routine: in this case it is sufficient to correct only the coordinates of the point of interest that is the centre of the pendulum.

As asserted in Lemma 1, the dimensions of the matrix are defined by the dimensions of the spaces involved in the transformation. On the implementation level, the transformation matrix is obtained by providing the 8 points to the OpenCV `getPerspectiveTransform` function [14]. Since this function returns a 3×3 matrix, we have to think the bi-dimensional points extracted from the image as they were points in R^3: this goal can be achieved by adding to each point the third component $z = 1$. Applying the equation of Lemma 1 we obtain (8)

$$\begin{aligned} x_{new} &= \frac{M_{1,1}x + M_{1,2}y + M_{1,3}}{M_{3,1}x + M3, 2y + M_{3,3}} \\ y_{new} &= \frac{M_{2,1}x + M_{2,2}y + M_{2,3}}{M_{3,1}x + M_{3,2}y + M_{3,3}} \end{aligned} \tag{8}$$

where (x, y) are the coordinates given by Template Matching [17]. The denominator guarantees that also z_{new} is always one: applying the transformation to the z component it is possible to verify that this is always true, as in (9).

$$z_{new} = \frac{M_{3,1}x + M_{3,2}y + M_{3,3}}{M_{3,1}x + M_{3,2}y + M_{3,3}} = 1 \tag{9}$$

Moreover, (8) is the same employed in the OpenCV `warpPerspective` function which is the one used in the calibration system to apply the perspective transformation to the entire image: this ensures the consistency of the transformation, whether applied to a single point or to the entire image.

C. Multithreading

Multi-threading is the ability of a CPU to execute simultaneously multiple tasks. Such capacity can come in handy even in this project. After implementing Template Matching algorithm, the system was able to elaborate approximately 5 frames per second. A greater number of coordinates extracted at each second would describe better the pendulum motion. It is therefore of interest to develop a solution capable of increasing the FPS number. As a first intuition, a speedup could be achieved by reducing the computation time of the frames elaboration. The analysis of the execution times has shown that it takes $0.1726ms$ to acquire and process a frame, $0.1698ms$ of which needed to apply Template Matching, i.e. 98.4% of the elapsed time. The Raspberry Pi 4, as already mentioned on the hardware specifications section (II-A), has a four-cores CPU thus a multi-threading approach could be pursued to increase the FPS number. It comes natural to use four threads in order to keep all the cores busy. A greater number of threads could in fact lead to a performance reduction due to CPU overload and excessive overhead costs.

The classic producer-consumer structure consists of a *producer* thread that creates something and enqueues it into a buffer. The *consumer* thread dequeues the element and uses it to perform a task. This process is shown in Fig. 13.

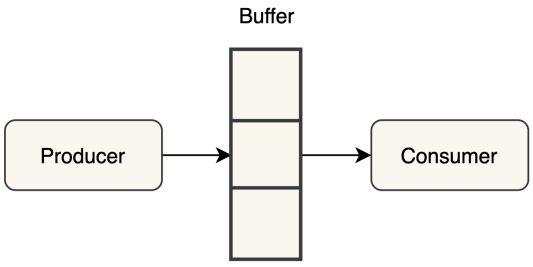


Fig. 13: Producer-Consumer pattern schema.

In this specific case the multi-thread structure that has been employed is a cascading producer-consumer structure. The first level producer handles the extraction of the frames while two consumers process them applying Template Matching algorithm. The consumers are also producers: in fact they extract the coordinates of the pendulum which are consumed by another thread responsible for writing them in the output file. The detailed structure of the multithread system is as follows:

- The **main** thread acquires the frames from the camera and puts them in two queues, `frameQueueA` and `frameQueueB`, on an alternate basis.
- Two threads, `ThreadA` and `ThreadB`, are delegated to *frame computation* i.e. they perform points extraction. They execute the same code, but each one extracts the frames from the respective queue, that is either A or B. Once the frame has been elaborated, the result is added to a specific thread output queue, `resultQueueA` or `resultQueueB`.
- The last thread is targeted to **writing** in a CSV file the results produced by the two computation threads. In order to do so, this thread pulls out the results provided in the two result queues.

The diagram of the applied producer-consumer structure is outlined in Fig. 14

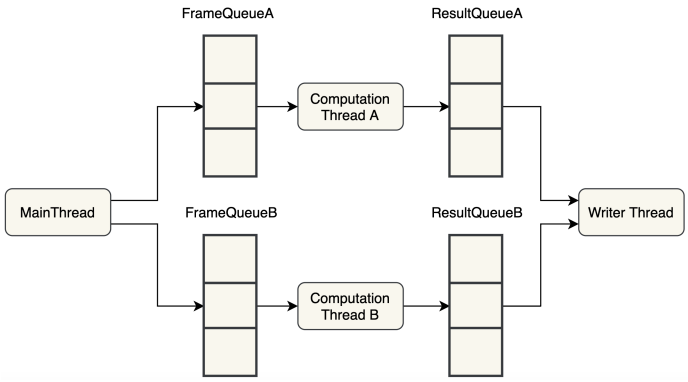


Fig. 14: Producer-Consumer relation between threads.

To recap:

- The `MainThread` acts as a producer on the two queues `FrameQueueA` and `FrameQueueB` with respect to the two consumers `Computation Thread A` and `Computation Thread B`.
- At the same time `Computation Thread A` and `Computation Thread B` behave as producers with respect to the consumer `Writer Thread`.

Fig. 15 shows the pattern of dependencies between threads. Following this figure, the main thread extracts from the camera a certain amount of frames per second (1), and associates to each one an `id` and a `time` of extraction; this information are employed in the reconstruction phase of the output. The frames are therefore inserted alternately in the two frame queues (2). In the meantime the two threads that handle the computation phase pop from their respective `frameQueue` a new frame to process (3). The pendulum detection and determination of the coordinates (4) takes place in this two threads simultaneously, allowing to handle two frames at once. This is what ensures an almost doubling of the frames per second. Once the result is produced, it is pushed to the respective result queue (5). Remains to be discussed the last thread, tasked with the writing to the output file. The results appended to the two queues at the previous step are sorted by extraction time (6) and then are written in the CSV file (7). This thread makes sure that the coordinates detected by the computational threads and popped from the two queues are merged back together properly.

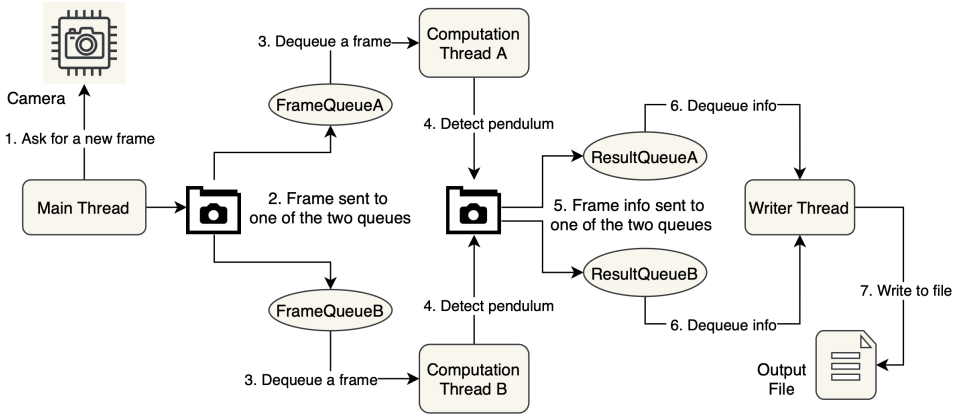


Fig. 15: Multi-threading flowchart of threads.

At this point it is necessary to make a consideration upon the general producer-consumer pattern. Usually, with this pattern, an aspect that deserves great attention is the synchronization between threads. The problem of synchronization raises when multiple threads needs to access a same resource, leading to possible race conditions. In this case, the structures under contention are the buffers that hold the frames and related information pre-computation and post-computation. Instead of two queues up-stream the two computation threads, it was possible to make use of only one of this buffers, regulating then the access by the two threads to that queue. Likewise, downstream the computation threads, there are two buffers. Those, similarly could be simplified in only one queue, monitoring the access to it by the two threads. In this case though, has been preferred to establish a little more complicated structure, with a queue per Computation Thread in order to avoid having to face synchronization issues that would have slowed down the whole system.

As highlighted by this discussion, there exists an underlying trade-off in a general multi-threading approach. In this case, the number of frames dealt each second depends on the CPU processing power. It was then important to find the right balancing between the two aspects. This task has been carried out using an empirical approach which has shown that it was possible to have a maximum extraction of 9 frames per second against the previous 5. The choice of the processing of this number of FPS is not accidental: it is the major number of frames per second receivable from the camera that the two computational threads can handle. A higher FPS rate would lead to `frameQueues` that would only grow in size, and the system would fail to keep up to the increasing number of frames received. This is not reasonable, since it would undermine the real-time system requirement. Moreover, at some point the memory capacity would no longer be capable of holding the whole buffer, leading to the raise of an error.

A more performing multi-threading system could be built by using a more powerful hardware. This could allow the increase of frame per seconds drawn out from the camera, or even justify the addition of a further computational thread.

V. User Interface

In addition to tracking, the program allows the user to calibrate the camera and display, both in real-time and offline, a graph of the pendulum motion. Each of these features corresponds to a different execution mode that can be selected by specifying an optional parameter in the start command. The different possible modes are as follows:

- **Tracking mode:** carries out the tracking process, shows the results in real-time and stores them in a CSV file.
- **2D Offline graph mode:** plots the points stored in a CSV file of a previous tracking execution.
- **Calibration mode:** allows the calibration of the camera.

The software is also equipped with a simple graphical interface. Fig. 16 shows the GUI of the tracking mode.

Fig. 16: GUI of the tracking system.

This window allows the user to receive a feedback from the tracking algorithm about the detected region. The buttons at the top of the interface make it possible to perform a series of operation, like zooming and saving the frame shown. The rightmost button enables the switching to a graph view, addressed in the next subsection.

A. 2D Real-time Graph

As previously said, it is possible to draw a 2D graph of the real-time recorded coordinates. Since the final goal of the program is to study the trajectory of the pendulum, this feature allows the user to get a quick idea of the motion of the tracked object.

The graph, at any given time, shows by default up to 100 points. This number can be adjusted using the proper slider placed at the bottom of the interface. Having a fixed number of points causes the fact that, as new points come, the older points are deleted from the graph, leaving a freshly updated graph.

B. 2D Offline Graph

The idea behind this modality is similar to the previous one, except for the fact that *offline graph* shows the points stored in a CSV selected by the user. Moreover, since this graph is asynchronous with respect to the original stream of frames, a few additional operations can be performed:

- The appearing of new coordinates can be **stopped and played** once again.
- The **speed** of visualization of new points can be changed.
- The **number of points** displayed at a time can be changed as well.

The fact of being able to increase the speed of appearance of new coordinates can help reach quickly a point in time the user is particularly interested in. It also offers a straight away view of the data. On the other hand, decreasing the speed, or even stopping the visualization can support a more fine grained review of the movements. In Fig. 17 an example of the graph produced after a brief execution of the program. This view is obtained with the aid of a CSV generated while the pendulum was in motion. The number in the top left corner indicates the relative time with respect to the launch of the tracking. In other words, it is the time contained in each record of the CSV. The number of points displayed by default is set

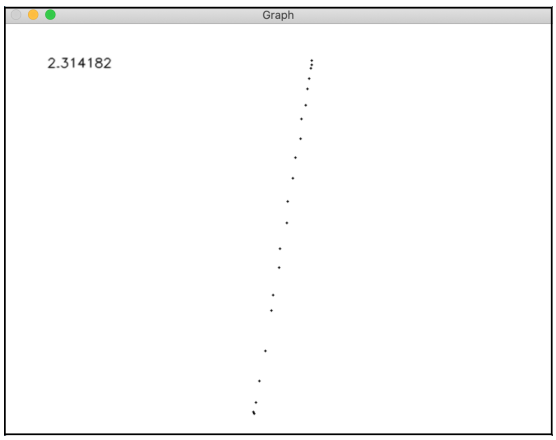


Fig. 17: 2D offline graph example.

to 30. The ability of changing the number of points to an arbitrary number can help achieve different understanding of the movements. Given that, as opposed to the real-time graph display, the program only has to draw points without further computations. For this reason, it is not set an upper limit for the number of coordinates that can be plotted. Bringing the number of points to the limit case of all the coordinates of the incoming CSV file will produce a graph similar to the one in Fig. 18.

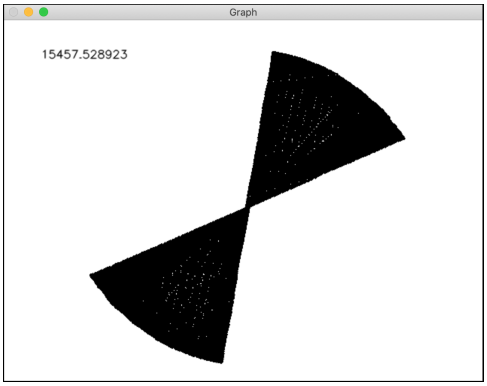


Fig. 18: Example of a 2D graph produced with an amount of points equivalents to 4 hours of execution.

A chart like this can assist into the spotting of anomalies in the pendulum or provide other useful information to the user. For more step by step instructions on the usage, refer to the proper section of the Pendulum Manual [19].

C. Camera calibration

Lastly, the program can be started in the camera calibration mode. Performing a calibration is an essential procedure to be done before the first execution of the program in tracking mode. Actually, the calibration has to be performed every time the camera is moved, even if by accident. As mentioned in Section IV-B, to apply the correction the user must indicate the region to which the perspective should be applied by placing the vertices of a quadrilateral. In order to help the user in the selection of the points, the program shows the *Warped frame* in real time. To provide additional visual feedback, Template Matching is also applied during calibration. We underline that this has a pure point selection assistance purpose, so no `CSV` file is generated, so no coordinates are saved. The best match point obtained by Template Matching is transformed by the system using the transformation matrix, as it happens during the execution in tracking mode. This point is finally drawn on the Warped image, so that the user can check that it is correctly positioned even once the perspective correction has been applied. Fig. 19 shows a frame extracted from the camera on which the points for the perspective transformation have been positioned. The result of this transformation has been already shown in Fig. 12. Further information and a tutorial on the position of stitches can be found in Section 2.2. of [19].

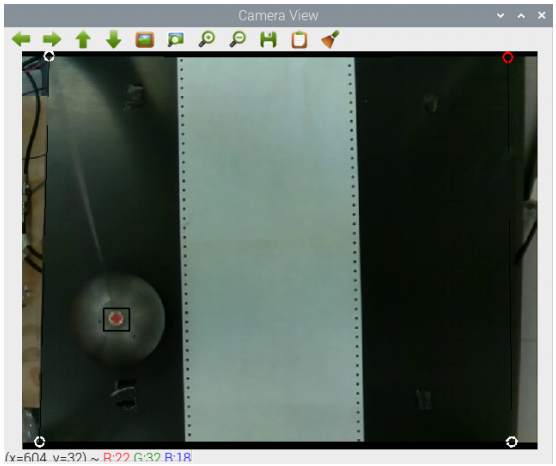


Fig. 19: Frame of the original image shown in calibration mode. On the image are visible the points that define the quadrilateral on which to apply the perspective transformation.

VI. Conclusions

The aim of the project was to develop a tracking system that could continuously and accurately extract the coordinates of a moving Foucault pendulum, making it possible to analyse its motion. From the experiments carried out, detection by Template Matching has been found to be the only technique that always guarantees maximum precision and adequate robustness to noise, such as changes in brightness and deformations of object shapes. The high computational complexity of the selected method has limited the maximum number of coordinates per second that can be extracted. In order to increase that number, a producer-consumer multi-thread structure has been employed, bringing the FPS limit from five to nine. A future development could be to speed up the matching process, for example by investigating the algorithms for calculating a fast `ZNCC` [10][6][8].
As further addition, the tracking system could be flanked by an ideal model simulator in order to juxtapose the real and the expected data, so as to facilitate the detection of any anomalies. Lastly, the raspberry camera has to be properly calibrated with a calibration plate.

References

[1] Alex Bate. *Thermal testing Raspberry Pi 4*. Nov. 2019. URL: https://www.raspberrypi.org/blog/thermal-testing-raspberry-pi-4/.

[2] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson Education International, 2002, pp. 595–611. ISBN: 8178086298.

[3] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. 3rd. University of Tennessee: Pearson Education International, 2007, pp. 891–893. ISBN: 978-0131687288.

[4] H.D. Cheng et al. "Color image segmentation: advances and prospects". In: *Pattern Recognition* 34.12 (2001), pp. 2259–2281. ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(00)00149-7. URL: http://www.sciencedirect.com/science/article/pii/S0031320300001497.

[5] Luciano da Fontoura Costa and Roberto Marcondes Cesar. *Shape Analysis and Classification: Theory and Practice*. 1st. USA: CRC Press, Inc., 2000. ISBN: 0849334934.

[6] Luigi Di Stefano, Stefano Mattoccia, and Federico Tombari. "ZNCC-based template matching using bounded partial correlation". In: *Pattern Recognition Letters* (May 2005). DOI: 10.1016/j.patrec.2005.03.022.

[7] *IMAQ VISION, Concepts Manual*. http://www.ni.com/pdf/manuals/322916a.pdf. National Instruments Corporation. Austin, Texas, Oct. 2000, pp. 216–217.

[8] Jianwen Luo and Elisa Konofagou. "A fast normalized cross-correlation calculation method for motion estimation". In: *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control* 57 (2010), pp. 1347–1357. DOI: 10.1109/TUFFC.2010.1554.

[9] Zalili Musa and Junzo Watada. "Video Tracking System: A survey". In: (Mar. 2008).

[10] Nakhmani and Tannenbaum. "A New Distance Measure Based on Generalized Image Normalized Cross-Correlation for Robust Video Tracking and Image Recognition". In: *Pattern Recognition Letter* 34 (2013), pp. 315–321. DOI: 10.1016/j.patrec.2012.10.025.

[11] Ferrante Neri and Jaco Fourie. "Robust Circle Detection Using Harmony Search". In: *Journal of Optimization* 2017 (2017), p. 9710719. DOI: 10.1155/2017/9710719. URL: https://doi.org/10.1155/2017/9710719.

[12] OpenCV. *Camera calibration With OpenCV*. URL: https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html (visited on 05/15/2020).

[13] OpenCV. *Gaussian Blur function*. URL: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=gaussianblur#gaussianblur (visited on 04/20/2020).

[14] OpenCV. *GetPerspective function*. URL: https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#Mat%5C%20getPerspectiveTransform(InputArray%5C%20src,%5C%20InputArray%5C%20dst).

[15] OpenCV. *MatchTemplate function*. URL: https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html.

[16] OpenCV. *Opencv HSV color space*. URL: https://docs.opencv.org/trunk/de/d25/imgproc_color_conversions.html#color_convert_rgb_hsv.

[17] OpenCV. *WarpPerspective function*. URL: https://docs.opencv.org/master/da/d54/group__imgproc__transform.html#gaf73673a7e8e18ec6963e3774e6a94b87.

[18] *Oxford English Dictionary, 3rd edition*. Oxford University Press, 1989.

[19] Claudia Raffaelli and Francesco Scandiffio. *Programma di tracciamento del pendolo di Foucault. Manuale utente.* Aug. 2020.

[20] Bruno Ribeiro et al. "Arbitrary ball detection using the circular Hough transform". In: (Sept. 2020).

[21] Ampulheta do Saber. *Pêndulo de Foucault*. http://ampulhetadosaber.com/fisica/curiosidades/.

[22] Bjarne Stroustrup. "An Overview of the C++ Programming Language". In: (Jan. 1998).

[23] Jonas Vautherin et al. "Photogrammetric accuracy and modeling of rolling shutter cameras". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* III-3 (June 2016), pp. 139–146. DOI: 10.5194/isprsannals-III-3-139-2016.

[24] Tranos Zuva, Oludayo Olugbara, and Seleman Ngwira. "Image Segmentation, Available Techniques, Developments and Open Issues". In: *Canadian Journal on Image Processing and Computer Vision* 2 (Jan. 2011).