Universiteit Leiden
Opleiding Informatica

Finish Photo Analysis for Athletics Track Events using Computer Vision Techniques

Name: Roy van Hal

Date: 21/07/2017

1st supervisor: Dirk Meijer

2nd supervisor: Arno Knobbe

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

With the current timing systems for athletics track events, the race times of participants can be recorded very accurately. However, these systems require the race times to be determined manually by an operator using a finish photo, taking up to several minutes. To reduce the delay that this action requires, this research project aims to create a program to determine the race times automatically from a finish photo. The program first detects the participants in the image by detecting foreground and background pixels. An iterative optimization algorithm then merges or separates the foreground components using morphological operations to extract individual participants. Their torsos are detected by a heuristic scoring technique applied to the lines found by a Hough filter to determine the race time of the participants. The program is evaluated using a test set of 10 finish photos that vary in brightness, contrast and size. The race time determined by the program of 60% of the participants is within 10 ms of the time determined manually by an operator. A second test, where the program is run on 11 variations of a finish photo, shows that it works well for different conditions, but the results deteriorate when the contrast in the images increases or decreases too much.

Preface

You are now reading the bachelor thesis *Finish Photo Analysis for Athletics Track Events using Computer Vision Techniques*. It was written as part of my bachelor project, which is part of the bachelor Computer Science at Leiden University. Work on this project has been performed from February to July 2017.

The topic of this thesis is Finish Photos, in particular Finish Photos for Athletics Track Events. The idea for this topic is based on a personal interest. Next to my study Computer Science, I am a judge for track and field sports. I have been a judge since June 2010 at a sports club in a neighboring village. At the end of 2016, the club decided to buy an automatic timing system. Because of my interest in digital systems, I have been involved since the beginning. With a team of other enthusiasts, I now provide automatic timing services for different types of track events. Even though our timing system is new, I have already seen hundreds of finish photos. I was thinking about optimizing the tasks that we do, which inspired me to come up with the idea for this thesis.

Doing this project, I have learnt a lot. My knowledge of images, digital image processing and doing research in general has certainly been extended. This could not have been done without any help. I would like to thank my supervisor for all his time and effort to teach me new things. Without his ideas, tips and explanations, this thesis would not have existed.

I am grateful for having the opportunity to learn something new by combining two of the things that I like to do. I hope that you, dear reader, enjoy your reading.

Roy van Hal

July 21, 2017

Contents

1 Introduction **5**
1.1 Photo finish photos . . . 5
1.2 Waiting for results . . . 6
1.3 Goals . . . 6
1.4 Approach . . . 7

2 Participant segmentation **8**
2.1 Colors and color spaces . . . 8
2.2 Detecting foreground pixels . . . 8

3 Improving the participant layer **13**
3.1 Binary erosion and dilation . . . 13
3.2 Opening and closing . . . 14
3.3 Optimizing the closing kernel size . . . 15

4 Torso recognition **19**
4.1 Creating the contour . . . 19
4.2 Hough transform . . . 20
4.3 Ranking lines . . . 20
4.4 Assumptions . . . 21

5 Experiments **23**
5.1 Evaluation . . . 23
5.2 Program accuracy test . . . 23
5.3 Assessing robustness . . . 25

6 Conclusion **27**
6.1 Summary . . . 27
6.2 Further research . . . 28

References **29**

1 Introduction

A lot of sports are centered around being the fastest competitor to cover a certain distance. Examples of these sports are track running, horse racing, bicycle racing, rowing, auto racing and swimming. Each of the races can be divided into three consecutive phases: start, race and finish. During the start, the participants are lined up and wait for a *start signal*. This signal can be a light or a sound, depending on the sport. When it is given, the race has started and a timer keeps track of the time that has elapsed. When a competitor reaches the finish, which is usually a line, their time has to be determined.

1.1 Photo finish photos

In most sports, the finish is captured using a *line-scan camera* [1]. This camera is aligned with the finish line. The frames it records are only a single pixel wide, hence the name. Each frame that is captured is associated with a time stamp that is provided by the timer. The camera operates at a high frame rate, allowing very accurate finish times for each of the participants. When the frames are chronologically arranged, a *finish photo* is created, with the location on the finish line on the vertical axis and the time on the horizontal axis.

This type of race timing, in which the race timer is automatically started when the start signal is given and the finish times of the participants are derived from a finish photo, is called *automatic timing*. After the race has finished, the operator of the automatic timing system uses software to analyse the finish photo. For each participant, he assigns the race time. The exact finish time depends on the rules for the sport that is practiced. In bicycle racing, for example, a participant finishes when their front tyre reaches the finish line [2], while in rowing, a crew finishes the race when the bow of the boat has crossed the finish line [3].

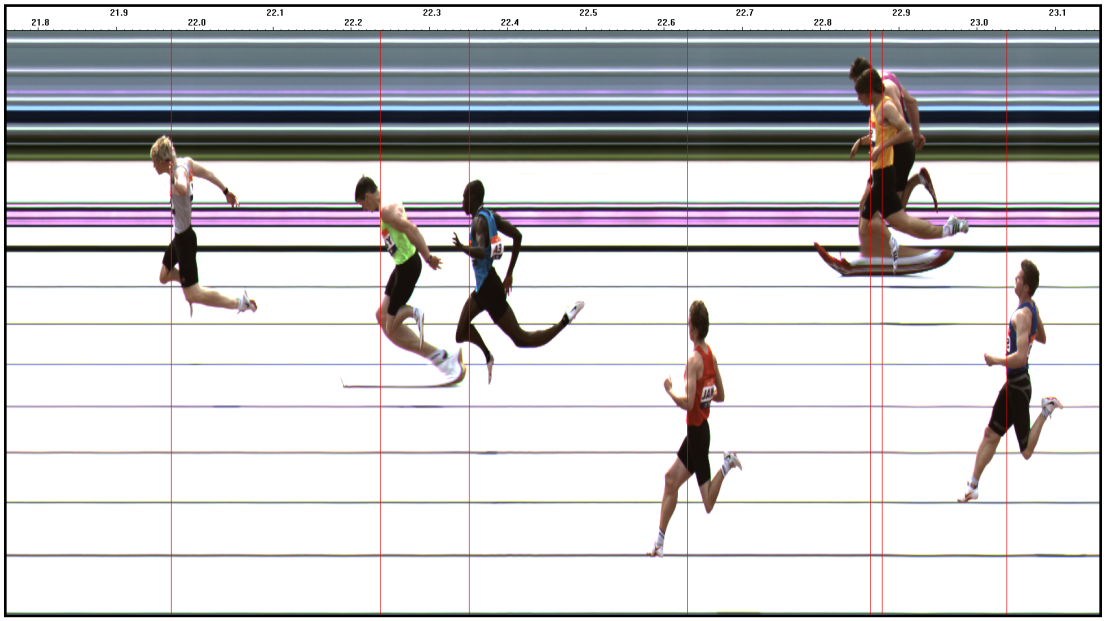


Figure 1.1: An analysed finish photo of a 200 m race with the time on the horizontal axis, the location on the finish line on the vertical axis. The finish time for each participant is indicated by a red line.

In this thesis, we focus on finish photos of track running, a subsection of the events that exist in track and field sports. An example of such a picture is shown in Figure 1.1. The time of a finishing competitor is determined by the moment that they touch the vertical plane of the finish line with any part of their torso, excluding the head, arms and legs [4, p. 171]. This can sometimes be difficult for the system's operator, since the torso of a participant cannot always be distinctly recognized. This is the case, for example, for the participant second from the left in Figure 1.1, because his torso is overlapped by another participant. This makes it hard to determine an accurate time stamp. Not only the time stamp has to be determined by the operator, but also which participant has achieved this time. The participants are identified by the lane that they are running in. Therefore, they finish on different vertical levels. This can also be a challenge for the operator, because the participants change in height constantly while running. In Figure 1.1, the second and third competitors from the left are at the same level, but they are finishing in different lanes.

1.2 Waiting for results

The difficulties of accurately identifying the time stamps and participants require an experienced and focused operator to analyse a finish photo. Because of this, the participants have to wait after their race for their official race time. For the first place, there is usually a photocell sensor at the finish line that gives an indication of the participants' time. However, it triggers on any body part that finishes and can therefore only give an unofficial indication of the race time. This is the reason that participants that finish close to each other do not know their placement and time and have to wait for their results. It takes about a minute for an experienced operator to determine all the time stamps. Then, the results have to get accepted by a Track Referee [4] and published. These delays leave the competitors waiting with tension. The process from the time that the participants finish to the moment the results are published is shown in Figure 1.2.

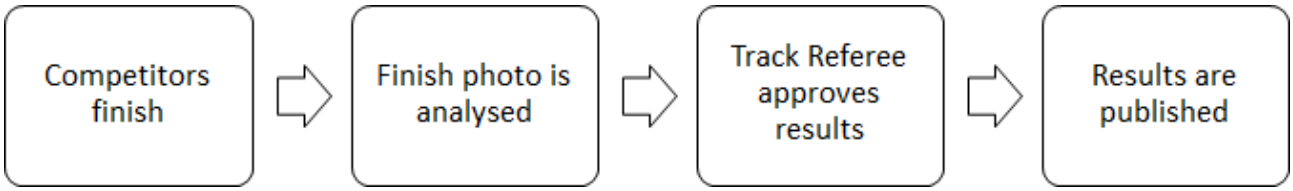


Figure 1.2: The process that is required to publish race results at the end of a race.

Small, regional events, can have dense schedules where there can be a lot of track events or heats after each other. The timing system operators may choose to analyse the finish photos at a time when the schedule is not as tight. Also, the Track Referee might be not available after every race to approve the results. Finally, the way that the results are published are different depending on the event. Sometimes the results are uploaded online and printed directly by the operators after they are approved, but usually the results are written on a piece of paper, delivered to the Competition Secretary and published when they get around to do it. Therefore, it can take between one minute, for big, international events, and multiple hours, for small, regional events, before the competitors are relieved from their tension.

1.3 Goals

In this thesis, an attempt is made to shorten the time of uncertainty of the competitors. The question that we want to answer is: *Is it possible to automate the determination of track running race times by analyzing a finish photo using computer vision techniques?*

To reduce the delay between finish time and publishing results, a computer program is developed that can analyse a finish photo to estimate the finish times of the competitors, to give them an idea of their placements and race time. The process for using this program when it is implemented is shown in Figure 1.3.

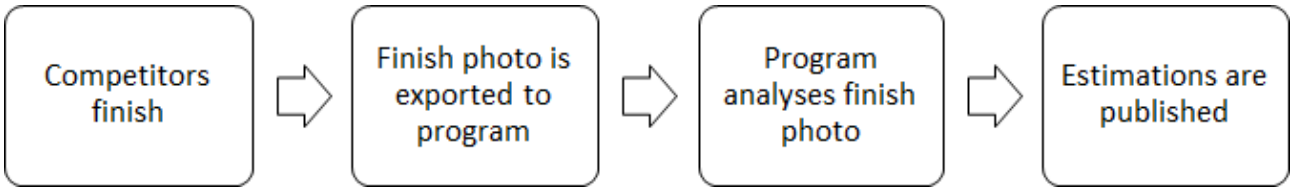


Figure 1.3: The process that is used to publish estimated race results at the end of a race using the program.

To be useful, the program must fulfill the following requirements:

- Recognize the torsos of the competitors.
  The torso of each participant has to be recognized to assign a time stamp. When a participant is not detected, he does not benefit from the program and still has to wait to know anything. Additionally, the recognition of too many participants will lead to redundant time estimations. It is therefore desired that only torsos are being recognized as torsos.
- Estimate finish times accurately.
  The times that are estimated by the program should be close to the official time that the operator will assign. If they are not accurate, there is no benefit in using the program, since the estimated times do not give any information to the competitors.

- Work in different environmental conditions.
  The program should work with different events, weather conditions, lighting, tracks and camera settings. It should be useful for light and dark images, with different heights and widths. Having a program that only works for a certain type of event under certain conditions make its usage unattractive.
- Generate the estimations quickly.
  The program is not useful if it is slower than manually determining the finish times. The manual finish times are official and estimations that come after have no value.

Even if this program turns out to be very accurate and works very reliably, we do not propose that the program replaces the judge. Firstly, we do not expect the program to be always accurate enough in all cases. There are a lot of difficult situations possible that the program might not be able to deal with. For example, participants finishing at the same time could overlap each other. This could result in missing or strange results, making human approval necessary. Secondly, full automation is prohibited by the official rules [4, pp. 173-175]. Related research has not been found. It seems like this thesis is the first attempt at such an approach.

1.4 Approach

The program that will be developed consists of multiple steps. An overview is shown in Figure 1.4. First of all, the finish photo is imported into the program. In Section 2 a background is computed to separate the finish line and the participants into two layers. The participant layer is then enhanced and the individual participants are extracted in Section 3. In Section 4 a Hough filter is applied to detect the lines of the torsos. These are used to determine a line that indicates when the competitor reaches the finish line. Finally, the finish photo with the lines determined by the program is shown. If required, a judge can correct the estimated lines to assign the official lines.

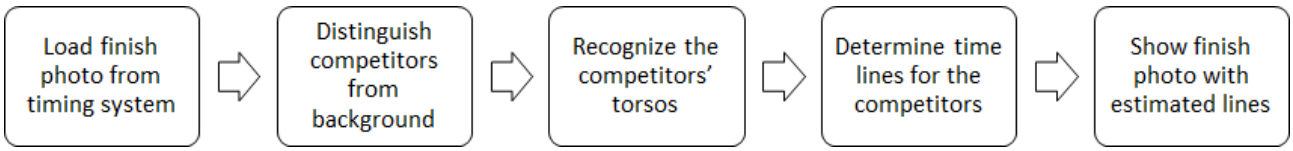


Figure 1.4: An overview of the steps that the program executes.

The program accuracy and its limits are tested in Section 5. A summary and possible further research is presented at the end of this thesis in Section 6.

2 Participant segmentation

Before the participants in the finish photo can be analysed, they need to be detected. The goal is to distinguish them from the background. To do this, pixels that are part of the participants need to be detected. These pixels form components which are labeled as foreground.

2.1 Colors and color spaces

Each finish photo consists of pixels that are aligned in a 2-dimensional grid. The number of pixels in the horizontal and vertical dimension are called the horizontal and vertical *resolutions* of the image. The horizontal resolution of a finish photo depends on the amount of frames that is recorded by the line-scan camera. Each column of pixels represents a frame. The more frames are captured, the higher the horizontal resolution. The number of frames depends on the recording time. The recording time required is determined by the speed and the distance between participants and the camera settings used. The vertical resolution is determined by the settings of the camera, set by the operator of the timing system. The operator bases this decision on the number of lanes that the track has and the camera lens that is used.

Each pixel has a color which is represented by a tuple of integer values. Each element in the tuple is called a *channel*. Each channel represents a different aspect of the pixel. The size of the tuple and the meaning of each channel in it is determined by the *color space* that is used to represent the colors. The color space also defines the values that each channel in the tuple can have. A channel value can be denoted by $V(i, j, c)$, where i is the row of the pixel that the channel is part of, j the pixel's column and c the channel index. Pixels represented in the same color space with equal tuples, represent the same color and vice versa. Two colors are equal when $\forall c : V(i_1, j_1, c) = V(i_2, j_2, c)$.

One example of a color space is the RGB color space. Each pixel is represented by a 3-tuple with the channels R, G and B, standing for the amount of red, green and blue in the color respectively. The value of all three channels is at least 0 and at most 255, so $0 \leq V(i, j, c) \leq 255$. This differs from the HSV color space for example. Its first channel, the Hue channel, is at most 180, but the Saturation channel and Value channel have a maximum of 255. Therefore, $0 \leq V(i, j, 1) \leq 180$, but $0 \leq V(i, j, 2) \leq 255$ and $0 \leq V(i, j, 3) \leq 255$.

2.2 Detecting foreground pixels

To separate the participants from the background, different methods have been considered. Each of them has their own strengths and weaknesses. The following sections describe three of the methods that have been considered: *Column Repetition*, *Common Color Set Matching* and *Tolerant Mode Matching*.

2.2.1 Column repetition

The camera that creates the finish photos is a line-scan camera and therefore each column in the photo is a frame (see Section 1.1). When the camera is recording and nobody crosses the finish line, the frames intuitively should be all equal. Because the time stamps are on the horizontal axis, each row of pixels should be a constant color. Using this assumption, the background of the photo can be reconstructed by duplicating the first first frame along the race time axis. Based on the finish photo in Figure 1.1 in Section 1.1, the reconstructed background would be as shown in Figure 2.1.

The foreground can be extracted by comparing the pixels of this background with the pixels of the photo. When the pixels at the same location are equal, which means that they have the same values in all channels of the color space, it is background and this is represented by a black pixel in the participant layer. When they differ, the pixel will be white. For a pixel P in the finish photo at row i and column j and background column pixel B at row i as inputs, the output is a pixel at row i and column j in the participant layer L. This equation is shown in Equation 2.1.

$$L[i, j] = \begin{cases} \blacksquare & \text{if } P[i, j] = B[i] \\ \square & \text{if } P[i, j] \neq B[i] \end{cases} \tag{2.1}$$

The result is a binary image in which the participants are supposed to be in white and the background in black. Applying this function to the example finish photo, the image shown in Figure 2.2 is obtained.

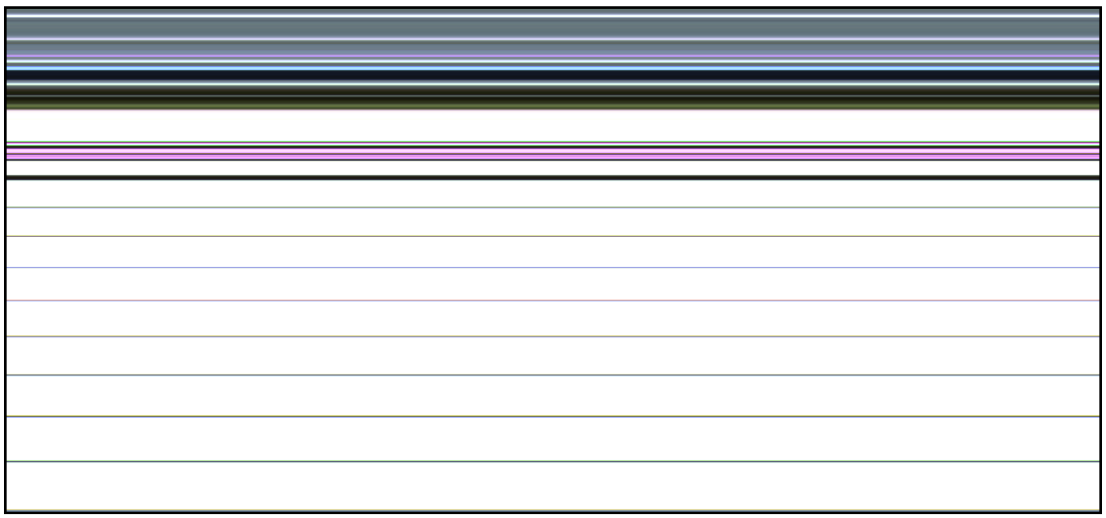


Figure 2.1: Reconstruction of background by repeating the first column.

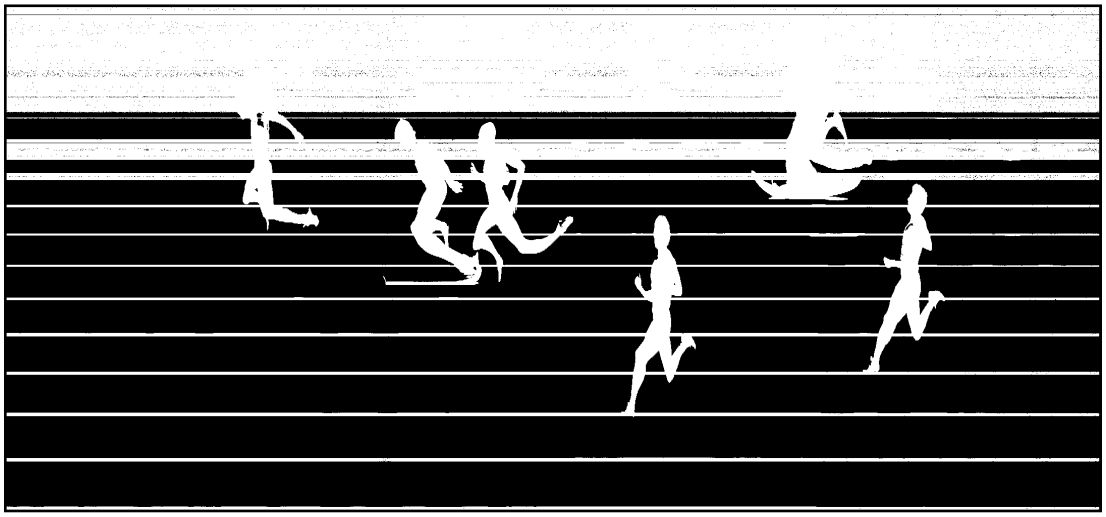


Figure 2.2: The resulting participant layer using column repetition.

However, this result is not what is desired. The participants are white and the finish lanes are black, as intended, but the lines between the lanes and most of the background above the track are white as well. This shows that our assumption was incorrect: the recorded frames change over time, even though nobody crosses the finish line.

There are multiple reasons for this. Firstly, the background above the track can change. It is not uncommon to have people sitting, standing or walking next to the track. These could be judges, referees, spectators or athletes that are not participating in an event at the moment. Additionally, the camera is usually operated outside. The light intensity changes constantly due to clouds. The camera constantly catches these changes and therefore even the slightest changes in lighting can change the frames. A third reason for the mislabeling is the noise generated by the sensor in the camera. The readings can change slightly between frames, while the colors do not change. This noise is too small to be noticed by an operator, but is large enough to create fluctuating pixel values. Because Column Repetition does not provide the detection of the participants on the desired level, this method was not used and other approaches were explored.

2.2.2 Common color set matching

The second method that has been considered, *Common Color Set Matching (CCSM)*, uses a set of values that occur often in a row. In each row, each channel in the color space of the image is associated with a set. When the values of a pixel in the row are all present in the channels' sets, the pixel is labeled as background in black. When any of the channel values does not appear in the corresponding set, the pixel is colored white, meaning that it is foreground.

Let $S_{i,c}$ denote the set of n values that occur most frequent in row i and channel c. The cardinality n of the set is predefined. For example, assume the finish photo is in HSV color space. $S_{3,1}$ would denote the set of most occurring values for row 3 and channel 1 (Hue). A channel value $V(i, j, c)$ of a pixel in the photo at row i, column j and channel c in a color space with m channels is labeled $L(i, j)$ as defined in Equation 2.2.

$$L[i,j] = \begin{cases} \blacksquare & \text{if } \forall c : V[i,j,c] \in S_{i,c} \text{ where } c = 1, 2, \ldots, m \\ \square & \text{otherwise} \end{cases} \tag{2.2}$$

Imagine an image in RGB color space. This means that each pixel in the image consists of three channel values. An example of a possible row i is shown in Figure 2.3. The channels are shown vertically and the columns horizontally. The color of the cells represent the color that the combination of channel values represent. For simplicity, the image in this example is only six pixels wide. The pixel in the first column ($j = 1$) has the values $R = 0$, $G = 200$ and $B = 255$. Therefore, $P(i, 1) = (0, 200, 255)$.

R		0	0	50	255	50	200
G		200	100	100	20	200	150
B		255	255	255	255	255	255

Figure 2.3: A pixel row with six columns and three channels.

When CCSM is applied, each channel in this row gets a set assigned. In this example, a set size of $n = 2$ is used. This means that for each channel, the two most occurring values are selected. This can be done by creating a histogram and picking the highest bars, or by simply counting the occurrences. For channel R, the two most common values are 0 and 50. Therefore, $S_{i,R} = \{0, 50\}$. In the same way, $S_{i,G} = \{100, 200\}$ and $S_{i,B} = \{255\}$. Notice that even though $n = 2$, only one value of the B appears in the row and there are no more values to put into the set. The sets are defined, so Equation 2.2 can now be applied. For $j = 1$, the left-most pixel, $0 \in S_R$, $200 \in S_G$ and $255 \in S_B$, so this pixel is labeled background and $L(i, 1) = \blacksquare$. In the same way, $P(i, 2) = P(i, 3) = \blacksquare$. However, for $P(i, 4)$ this is not the case, because $R = 255 \notin S_{i,R}$ and $G = 20 \notin S_{i,G}$. It does not matter that $B = 255 \in S_{i,B}$, because all three channel values need to be in their corresponding sets. So $L(i, 4) = \square$, $L(i, 5) = \blacksquare$, $L(i, 6) = \square$ and the resulting binary row $L(i)$ is shown in Figure 2.4. This is a binary image, so there is only a single channel and the values are either 0 (black) or 1 (white).

L		0	0	0	1	0	1

Figure 2.4: Participant layer row after applying CCSM with $n = 2$.

CCSM has been tested on the finish photo in Figure 1.1 in Section 1.1 in multiple color spaces and different values for n. One of the results is shown in Figure 2.5. This image was created using $n = 10$ using the HSV color space.

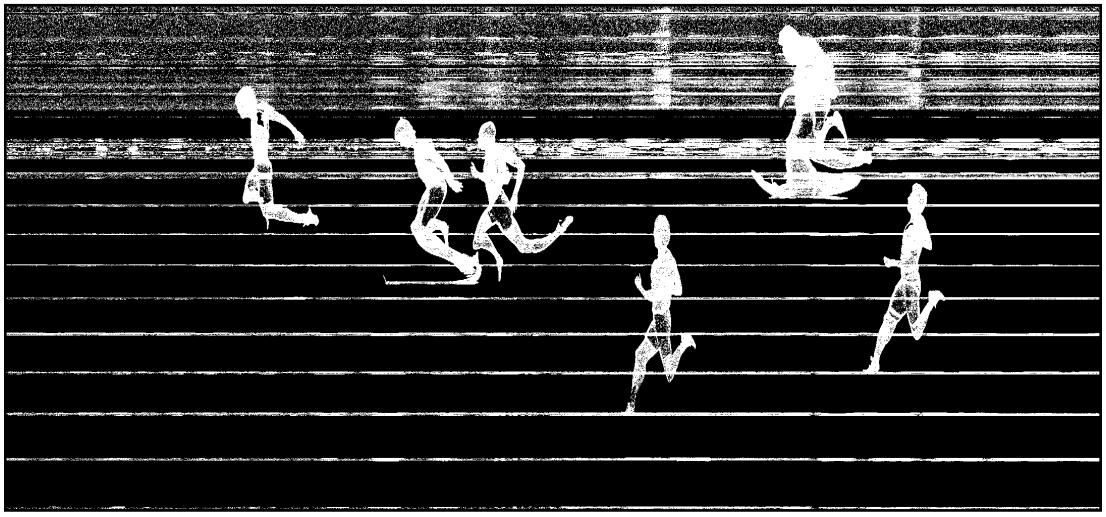


Figure 2.5: A resulting participant layer after applying CCSM with $n = 10$ in HSV color space.

This result is better than when Column Repetition is used, but it is still not great. A lot of false positives are present in the upper background above the track. This noise was created because not all channel values are in the sets, so the maximum set size is too small. However, there are a lot of false negatives as well. Parts of the participants' clothes are labeled as background, even though they are foreground. This means that the set size is too large for these rows, because values are put into the sets which do not occur often. So for some rows, n is too small, but at the same time, n is too large for others. Therefore, this method does not work well in this picture for any n in any color space. This method works well when the n most appearing colors in each row are all background. However, this is almost never the case for finish photos because the lanes contain very few distinct values and the upper background contain a lot of distinct values. The white areas of the lanes contains less noise, because they are often bright enough to saturate the pixel channels.

To improve this method, the size of the sets can be varied depending on the row and channel that is analyzed. For example, if the channel in a row will contain only a few different values, the set could be smaller and thereby the number of pixels that are labeled as background would be decreased. However, it is unclear how to determine these dynamic set sizes.

2.2.3 Tolerant mode matching

The final method that was explored is based on the idea that the frames look like each other and the pixels of a certain row only change a lot when a participant crosses the finish line. Most of the time, nobody crosses the finish line, so the channel modes in a row are background. The pixels that have values that are close to these are probably also background. Therefore, there is an interval around the channel modes in the row in which the pixels are labeled as background, hence the name of this method: *Tolerant Mode Matching (TMM)*. The mode that appears in each channel can by determined by counting the occurrences of each value. When the channel values are all within the thresholds, the result is a black pixel in the output participant layer. When a pixel contains one or more channel values that are not within their corresponding intervals, the pixel is labeled as foreground and colored white in the output layer.

For a value $V[i, j, c]$ that channel c in a pixel in the finish photo at row i and column j has, the label at this row and column $L[i, j]$ is defined in Equation 2.3. The color space has m channels, $F_{i,c}$ is the mode of the channel in the row and T_c is the threshold for channel c.

$$L[i,j] = \begin{cases} \blacksquare & \text{if } \forall c : |V[i,j,c] - F_{i,c}| \leq T_c \text{ where } c = 1, 2, \ldots, m \\ \square & \text{otherwise} \end{cases} \tag{2.3}$$

An example is shown in Figure 2.6. Colors in the RGB color space can be plotted in this coordinate system. Point M has the color of all channel modes combined. The black box around it shows the thresholds in each direction. Points A and B represent colors that are evaluated. Point A has all of its channel values within the thresholds of the mode's channels. Therefore, it will be labeled as background. Point B is located outside the threshold box, so it will be labeled as foreground.

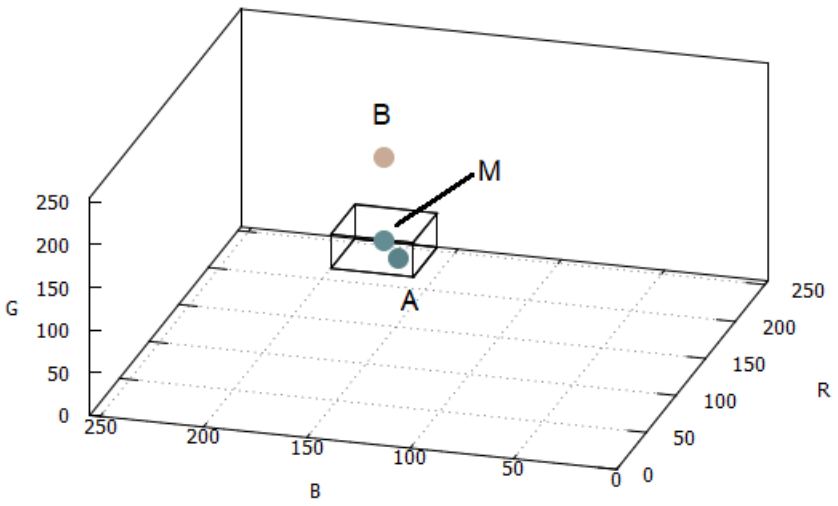


Figure 2.6: Threshold box $T_R = T_G = T_B = 10$ around $M = (F_{i,R}, F_{i,G}, F_{i,B})$ with candidate points A and B.

It should be noted that the minimum and maximum values of the row's intervals around the modes can get outside the domains of the channels. What then happens depends on the characteristics of the channel and the color space that is used. For most channels, the minimum value of the interval $I_{i,c,\min}$ cannot go below the minimum of the channel's domain $D_{c,\min}$. Therefore, $I_{i,c,\min} = \max(F_{i,c} - T, D_{c,\min})$. This also applies to the maximums, so $I_{i,c,\max} = \min(F_{i,c} + T, D_{c,\max})$. However, there are some channels that have different types of domains. For example, the Hue channel of the HSV color space has a circular domain. This domain does not have a minimum or maximum. In case the angle $F_{i,c} - T < 0°$, the interval minimum "loops around" and will be at a high value again. This is also the case when $F_{i,c} + T \geq 360°$. The interval maximum loops around and will be at a low value. An example of this is shown in Figure 2.7 where the mode is $F_{i,H} = 350°$ and $T = 20°$. The minimum for the interval is $I_{i,H,\min} = F_{i,H} - T = 350° - 20° = 330°$. The maximum is $I_{i,H,\max} = F_{i,H} + T = 350° + 20° = 370° \geq 360°$ so this will be corrected to $370° - 360° = 10°$.

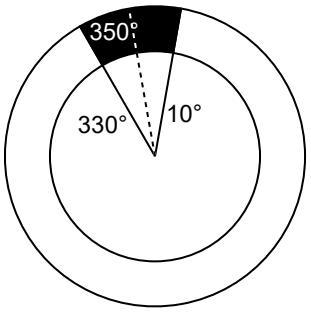


Figure 2.7: The interval $T_c = 20°$ around $F_{i,H} = 350$ after $I_{i,H,\max}$ has wrapped around.

To test this method, it was applied to the finish photo in Figure 1.1 in Section 1.1. Different color spaces and threshold values were tested. The resulting participant layer is shown in Figure 2.8, where thresholds $T_R = T_G = T_B = 10$ in RGB color space have been used.

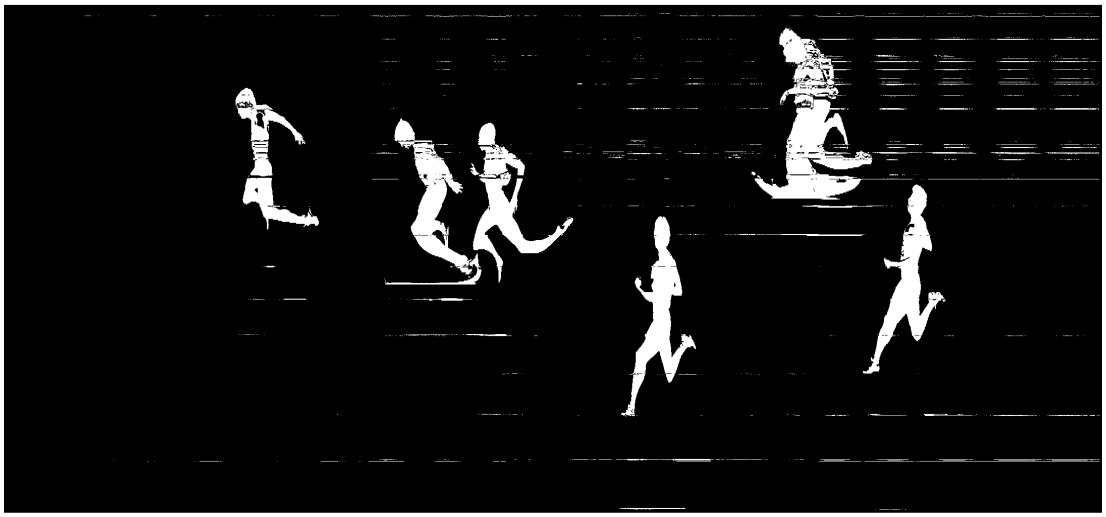


Figure 2.8: A resulting participant layer after applying TMM with $T_R = T_G = T_B = 10$ in RGB color space.

The best separation between foreground and background appears to be in RGB color space using thresholds $T_c \approx 10$. All channels in RGB seem to be equally important in the image, therefore a threshold that is the same for all channels works well. During testing, this was not the case in HSV color space. Pixels that have a low saturation (S) or value (V) do not have much information in the Hue channel, because they are all close to gray or black. The hue of the pixels in a dark row can change a lot, even though the color of the pixels are almost equal. The hue threshold does not influence the resulting participant layer a lot. Additionally, the tests show that the thresholds for the saturation and value channels needs to be very large to be useful, but this also creates a lot of false negatives.

3 Improving the participant layer

While Tolerant Mode Matching, described in Section 2.2.3, provides a basic separation between the participants and the background, it is not perfect. This is the case in Figure 2.8, after Tolerant Mode Matching was applied to a finish photo. False positive cases include the large fragments of the lines that separate the lanes and some noise in the background above the lanes. There are also components in the image that have not been labeled as foreground, but are part of it. These false negatives are created when parts of participants are similar to the background. They appear as holes in the participants.

To reduce these errors, *binary morphological operations* are applied to the pixels in the participant layer [5]. These operations can be applied to binary images and are based on the shapes in the image and the values in the neighborhood of the pixels that the operations are applied to. The two most common morphological operations are *erosion* and *dilation*. These can be combined to create the *opening* and *closing* operations.

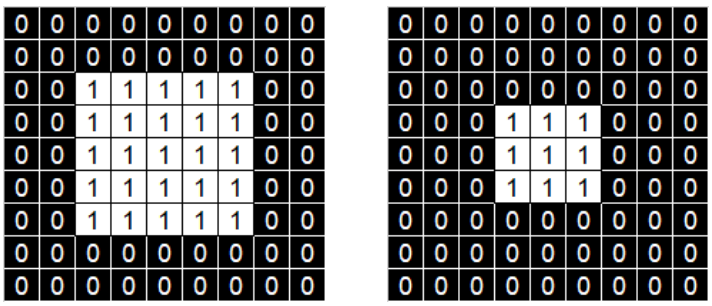
3.1 Binary erosion and dilation

Erosion and dilation operations change the value of a pixel depending on the pixels that are near it. Each pixel and its surrounding pixels in the input image are checked and the resulting value is calculated. This result is written to a binary output image. The surrounding pixels that are taken into account are specified by the *kernel*. This is a binary map that is centered on the current pixel. When the pixel in the kernel is 1, the underlying pixel in the input image is considered when erosion or dilation is applied, but when it is 0, the pixel in the input image is ignored.

3.1.1 Erosion

For erosion, the output is the minimum of the pixel and its neighbors. For each pixel in the input image, its surrounding pixels are evaluated and the minimum value is taken in the output. This is formally described by Equation 3.1, adapted from [5, p. 666]. In this equation, P is the input image, L is the output image, i is the row and j the column of the current pixel, which is the origin of the kernel. The set of coordinates that are considered within the kernel is denoted by k. $[s, t]$ is a coordinate in the kernel map at row s and column t.

$$L[i, j] = \min_{[s,t] \in k} (P[i + s, j + t]) \tag{3.1}$$



(a) Before erosion is applied. (b) After erosion is applied.

Figure 3.1: The effect of the erosion operation with a 3×3 kernel.

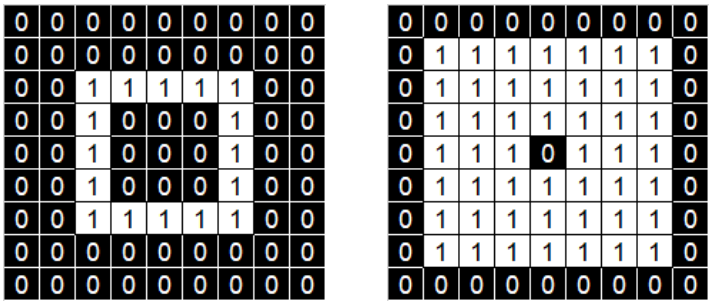
An example of an erosion is shown in Figure 3.1. A 3×3 kernel is used. In the input image Figure 3.1a, the 1 in the top left has five 0's in its kernel and becomes 0 because this is the minimum of the value in the kernel. The resulting image is shown in Figure 3.1b. The white component has become smaller, hence the name *erosion*.

3.1.2 Dilation

Dilation is an operation similar to erosion, but instead of the minimum value, a pixel that is being dilated takes the value of the maximum of the pixel and its neighbors as described by Equation 3.2, an adaption from [5, p. 666]. P is the input image, L the output image. The current pixel is located at row i and column j and is

the origin of the kernel. The coordinates in the map of the kernel are denoted by $[s, t]$, where s is the row and t the column. k represents the set of coordinates in the kernel that are considered.

$$L[i, j] = \max_{[s,t] \in k} (P[i + s, j + t]) \tag{3.2}$$



(a) Before dilation is applied. (b) After dilation is applied.

Figure 3.2: The effect of the dilation operation with a 3×3 kernel.

It will enlarge, or *dilate*, white components of a binary image. An example is shown in Figure 3.2, where the 3×3 kernel is used again. The pixels next to the square have become white as well, making the square thicker.

3.2 Opening and closing

The erosion and dilation operation can be applied consecutively. These are called the *opening* and *closing* operations. The opening and closing operations are used to improve the participant layer that was separated from the background (see Section 2).

3.2.1 Opening

An erosion followed by a dilation is called an *opening* operation. White components that are small, compared to the kernel, disappear in the binary image. The erosion causes the components in the image to shrink, making small components disappear. The dilation that follows increases the components that still exist to the size they approximately had before the erosion. This way, small noisy areas can be removed.

For the participant layer that is created, the opening operation can be used to remove small groups of pixels that have been incorrectly marked as foreground. The initial participant layer in Figure 2.8 in Section 2.2.3 shows that these are mostly horizontal line segments.

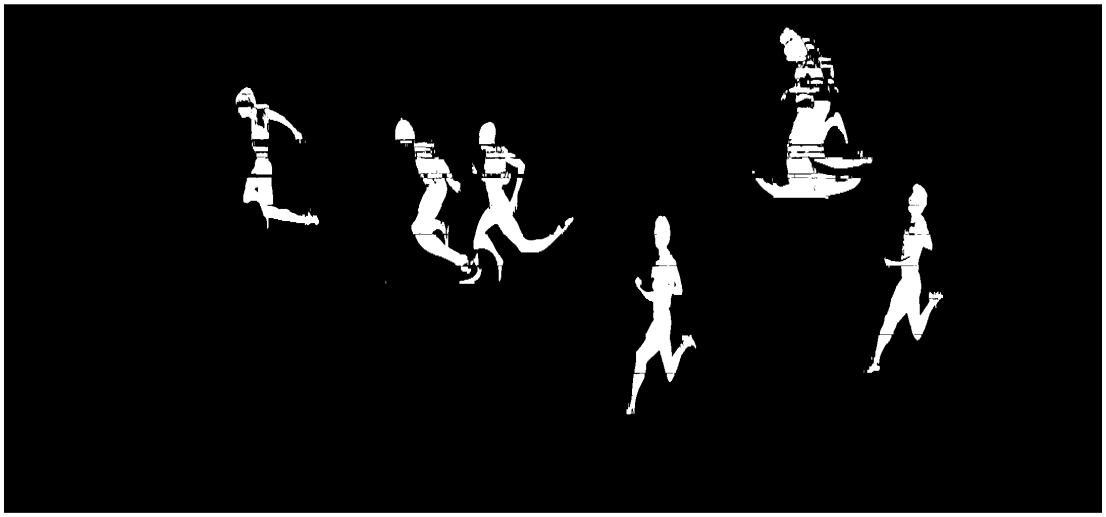


Figure 3.3: The participant layer after the opening operation has been applied.

The opening operation is used to remove these line artifacts. Because most of them are horizontal, a vertical kernel is used, to make sure that possible other incorrectly marked pixels to the left and right are not considered. Because most line artifacts are only one or two pixels in height, a kernel with a width of 1 and a height of 5 is used. Figure 3.3 shows the participant layer after the closing has been computed.

3.2.2 Closing

Related to the opening operation is the *closing* operation. First, a dilation is applied, which is followed by an erosion. The dilation expands all white components, which connects components that are close to each other. The erosion decreases the size of the components, but the components that were merged by the dilation stay connected. This way, holes between components are closed, giving the closing operation its name.

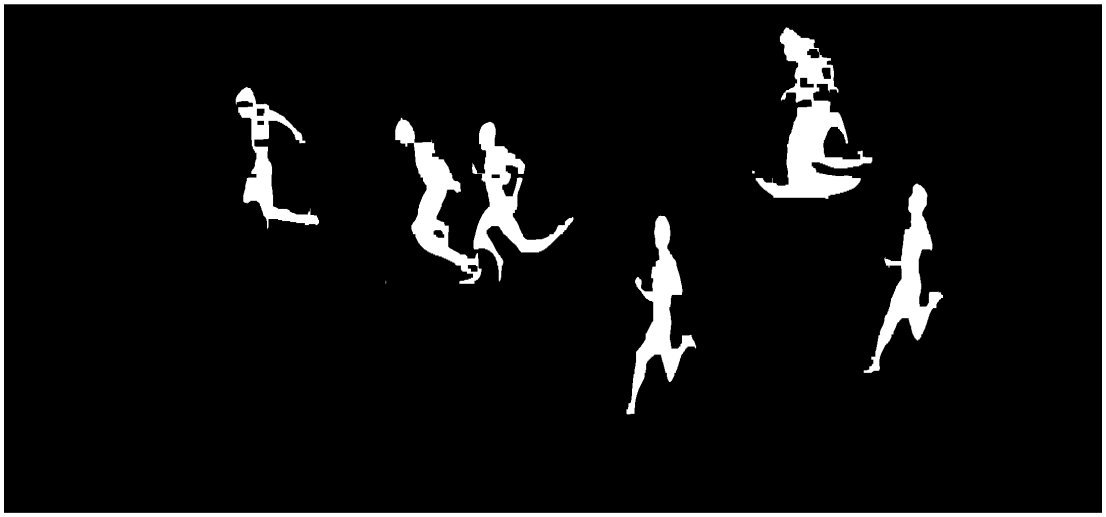


Figure 3.4: The improved participant layer after a closing with kernel size 3×3 has been applied.

After the horizontal line artifacts are removed by the vertical opening, a closing can be applied to remove false negatives from the participant layer. This is noise that has been marked as background, but are actually part of the foreground. For example, the left-most participant in the image in Figure 3.3 is split into two parts. His torso, head and arm form a single component, while his legs are part of another. To restore this, a closing is applied on the image which will close the distance between these components. The result is shown in Figure 3.4.

Indeed, the gaps are covered and the participants are less distorted. Some holes have not been filled, because the closing kernel was not large enough. However, a larger kernel might have merged components that do not belong together. This is the case when participants are close to each other. Therefore, a way to optimize the kernel is required.

3.3 Optimizing the closing kernel size

The kernel size for joining components together needs to be selected very carefully. When it is too small, components that belong to the same participant will not get merged together and a time will be assigned to each of the pieces, resulting in too many times. If the kernel is too large, different participants are merged into a single component with an unexpected shape. At least one time will be missing and one that is found is likely placed incorrectly.

The ideal kernel is one that is large enough to merge the components of an individual participant, but small enough such that it does not merge two participants together. To achieve this, different kernel sizes are considered in an iterative algorithm. The algorithm evaluates the effect of a closing with each kernel size and converges to a kernel size with the most suitable size.

A kernel has a height and a width. This makes finding the kernel dimensions a problem of two dimensions. However, when the ratio between the kernel's height and width is set to a fixed value, only one of them is required and the other dimension can be calculated. Using a fixed ratio also simplifies the problem of finding the kernel size to an optimization problem with only a single dimension instead of two. From testing, it appears that a ratio of 1, meaning that the kernel height is equal to the kernel width, works well.

3.3.1 Overview of the algorithm

The iterative algorithm looks for the optimal kernel size by applying candidate kernels to the participant layer after the closing operation has been applied. The areas of the resulting components are compared to an expected participant area. The number of components that are smaller and larger than this expected area are used to generate a score for the candidate kernel. This score is used to determine whether the kernel size should be increased or decreased and by how much in the next iteration.

When the kernel size for the next iteration is equal to a size that has been evaluated before, its score will not be different from the first time it was evaluated. Therefore, the algorithm stops iterating and for each candidate kernel size, a selection score is computed. The candidate kernel size with the best score is used in the actual closing operation. Pseudocode for the algorithm is shown in Listing 3.1

```
sizes_set = empty set
candidate_size = initial size

while candidate_size not in sizes_set do
    add candidate_size to sizes_set
    candidate_score = evaluate(candidate_size)
    if candidate_score > 0 then
        candidate_size = candidate_size + step size
    else
        candidate_size = candidate_size - step size
    end
end

best_size = 0
best_score = infinity

for each candidate_size in sizes_set do
    candidate_score = abs(evaluate(candidate_size))
    if candidate_score < best_score then
        best_size = candidate_size
        best_score = candidate_score
    end
end
```

Listing 3.1: Pseudocode for the iterative kernel size optimization algorithm.

3.3.2 Estimating participant size

To evaluate a candidate kernel size at iteration t, the different participant components are considered. They are split into three area categories: too small (S_t), expected (M_t) and too large (L_t). The number of components in each category determines the score for the candidate kernel size. The "expected" area category M_t consists of components with an area that corresponds to a single participant. Components that are a lot smaller than the expected area are put into the "too small" category S_t. These components are usually participants that have been split into multiple components but have not been merged by the closing. A lot of components with this area type indicate that the candidate kernel size is too small. Category L_t's elements are the components that are a lot larger than the expected participant area. These are probably multiple participants that have been merged into one. The existence of this type of areas indicate that the candidate kernel size is too large.

The expected area for a participant is different for every finish photo. It depends on the recording speed of the camera and event. For example, the participants of a 400 m hurdles race are slower than those of a 100 m sprint race and are therefore recorded for a longer time period. To obtain an estimation for the expected area, a closing with a relatively large kernel is computed. This merges the split components of a participant, but also merges participants that are close to each other. The small components that are left is noise that has not been removed by the opening operation described in Section 3.2.2. Additionally, there might be participants that have been merged together to a single component. However, this occurs less often than the components that are noise. Therefore, a complete single participant is expected to be located around the 70th percentile.

The 70th percentile of the set of component areas is calculated using the *linear interpolation between closest ranks method*. Let n be the number of elements in the set of areas A. The *rank* r of the qth percentile is defined in Equation 3.3.

$$r = \frac{q}{100}(n+1) \tag{3.3}$$

The integer part r_{int} of r is the number on the left side of the decimal point: $r_{\text{int}} = \lfloor r \rfloor$. The number on the right side is the fraction part r_{frac}, defined as $r_{\text{frac}} = r - r_{\text{int}}$. Let $A(i)$ be the area of the ith element in A when is sorted from lowest to highest. Then $A(1)$ is the minimal area of the components and $A(n)$ is the area of the largest component. The area P_q at the qth percentile is then defined as shown in Equation 3.4.

$$P_q = A(r_{\text{int}}) + r_{\text{frac}} \times (A(r_{\text{int}} + 1) - A(r_{\text{int}})) \tag{3.4}$$

The expected area of a participant is given by P_{70}. It is the interpolation between the elements of A that are greater than 70% of the areas in the set.

3.3.3 Kernel evaluation

Each kernel size that is evaluated by the algorithm gets an evaluation score assigned. This score indicates whether the kernel size should increase or decrease in the next generation. The evaluation score F_t for the kernel size considered at iteration t is shown in Equation 3.5. S_t and L_t are the number of components in the "too small" and "too large" categories, respectively.

$$F_t = S_t - 4L_t \tag{3.5}$$

It is assumed that too large components are around 4 times the size of the small components. Too small components are usually part of a split participant and therefore approximately half the expected size, while a component that is too large is about twice the expected participant size. The large component is then about 4 times the size of half a participant.

The algorithm tries to get F as close to 0 as possible, which means that it minimizes $|F|$. This happens when $S = 4L$, because then $F = 0$. This is a balance between the number of too small and too large components. Decreasing the kernel size will decrease L, but S will increase. However, increasing the kernel size will decrease S, but L will increase. Therefore, $F = 0$ is a balance that the algorithm tries to converge to. In the ideal case, $S = L = 0$, which means that there are no components too large or too small and every component is a single, complete participant.

The score also indicates whether the kernel size should shrink or grow. When $F < 0$, it holds that $4L > S$ which means that the kernel size should decrease to reduce the number of connected participants L. When $F > 0$, it means that there are too many components that are too small and the kernel size should increase to merge more of these together. The size of the kernel that is considered in the next iteration is computed by adding a *step size* Δ to the previous kernel size. The step size depends on the vertical resolution of the image. When the resolution is large, a larger step size is used. In this case, steps sizes of only a few pixels do not change the components that are merged compared to the previous kernel size. However, when the resolution is small, even a small increase in kernel size can change the arrangement of the components in the S_t, M_t and L_t categories excessively. Therefore, the step size used is a factor of the image height. A step size of 1% of the image height has shown the best results. For an image with a vertical resolution of h pixels, the step size Δ is defined as $\Delta = \frac{h}{100}$, rounded to the nearest even integer. When the algorithm starts with an odd kernel size, adding an even step size will make sure that the kernel size stays odd. The closing requires an odd kernel size since only kernels with an odd height and width will have a single center pixel that the closing applied to.

3.3.4 Improving converging time

When kernel sizes are considered that are not close to being the right size, a larger step size is desired. This decreases the time it takes for the algorithm to converge. When the number of components in the set with the expected areas M_t is too small, a larger step size is used. The assumption is made that there are at least three participants in a photo that can be categorized in M_t. This is a reasonable assumption because all athletics tracks have at least six lanes and the participants are usually distributed evenly over the series that they participate in. For example, imagine a track with six lanes and seven participants. The participants will then be distributed into two series of three and four participants each, instead of two series with six and one, to improve fairness and attractiveness of the event.

When the number of elements in M_t is less than three, the next kernel size will be increased or decreased by a large step size which is twice the small step size Δ. This is shown in Equation 3.6, where k_t is the kernel size

at iteration t, F_t is its evaluation score and m_t is the amount of components in M_t at that iteration. Δ is the small step size of approximately 1% of the image height.

$$k_{t+1} = \begin{cases} k_t + \Delta & \text{if } F_t > 0 \text{ and } m_t \geq 3 \\ k_t - \Delta & \text{if } F_t \leq 0 \text{ and } m_t \geq 3 \\ k_t + 2\Delta & \text{if } F_t > 0 \text{ and } m_t < 3 \\ k_t - 2\Delta & \text{if } F_t \leq 0 \text{ and } m_t < 3 \end{cases} \tag{3.6}$$

3.3.5 Selecting the best kernel

The algorithm uses the evaluation scores to converge to a kernel size that gives the best score. It stops when a kernel size is considered that has already been considered before. The kernel size used for the actual closing operation is selected based on the number of components in the too small S_t, expected M_t and too large L_t categories for each kernel size i. The candidate kernel sizes are ranked according to a selection score G_t. For candidate kernel size at iteratoin t, its selection score G_t is shown in Equation 3.7.

$$G_t = S_t + 4L_t \tag{3.7}$$

The candidate kernel size with the lowest selection score G is used for the actual closing operation. A closing with this kernel size will result in the least amount of components that are too small or too large compared to the expected component size. If there are multiple kernel sizes with the minimal selection score, the one with the highest number of components in the set with expected sizes M_t is used. This way, the actual closing will result in a maximum number of components that are of an expected size, while minimizing the number of components that are too small or too large.

4 Torso recognition

The official rules from the International Association of Athletics Federations (IAAF) states which time a participant should be assigned:

> The time shall be taken to the moment at which any part of the body of an athlete (i.e. torso, as distinguished from the head, neck, arms, legs, hands or feet) reaches the vertical plane of the nearer edge of the finish line [4, p. 171].

This rule requires that the program recognizes the torso of the competitors to be able to assign a time. This is done in four steps. First, each participant is extracted from the binary participant layer (see Section 2 and 3) and its contour is computed. Then, a Hough Transform [6] is applied to detect lines in the contour, of which one is hopefully (part of) the torso that reaches the vertical plane of the finish line. The lines that are found are ranked by a "score" that indicates how likely it is that a line indicates the part of the torso that reaches the finish line first. This score is determined by the position and angle of the line. Finally, the line with the highest score is selected and will be used to determine the race time of the participant.

4.1 Creating the contour

The individual competitors are extracted from the binary participant layer by detecting areas that are connected. These connected areas are called *components*. Two white pixels with the same label belong to the same component when they are adjacent. 8-connectivity is used, which means that pixels are adjacent when they are touching each other, diagonals included [5, p. 68]. Each competitor is a connected component. A component from the finish photo in Figure 1.1 in Section 1.1 is shown in Figure 4.1. There might be components that are not competitors. These artifacts are noise from the participant detection in Section 2 that were not removed by the morphological operations. However, these small artifacts can be detected and removed since they are an order of magnitude smaller than the participants.

Figure 4.1: A single participant component with its eroded version in white and contour in red.

The Hough Transform that finds the possible torso lines requires the contour of the participant. The eroding operation described in Section 3.1 removes this contour. To retrieve it, the eroded component is subtracted from the original component. This makes the pixels that are both in the component and in its eroded version black, leaving the pixels that is only in the original component. This is the contour of the participant, which is shown in red in Figure 4.1.

4.2 Hough transform

To detect lines in the contour, Progressive Probabilistic Hough Transform (PPHT) [7] is used. PPHT is a variation of the Standard Hough Transform [6] that requires less computation. It can be used to find lines, which in this case is part of the participants torso. Factors that PPHT uses to determine the lines include the minimum length of the line, a threshold for the number of pixels it overlaps and a maximum gap between two lines to still count them as a single line. An example of the lines found on the contour from Figure 4.1 is shown in Figure 4.2. It shows the contour of the participant in gray and the lines that have been found are colorized.

Figure 4.2: A single participant contour with the lines detected by the PPHT.

4.3 Ranking lines

The Progressive Probabilistic Hough Transform returns the lines that it has found in the contour. As Figure 4.1 shows, a lot of the detected lines do not belong to the torso or are on the right side. However, only the line left-most torso line is the one that corresponds to the right race time. To find this line, each line that has been found is associated with a score. A higher score indicates that a line is more likely to be a left-most torso line. The line score is determined by three factors: the line angle A_s, its vertical position P_s and the pixels that are next to it N_s. The line score S is determined by Equation 4.1. Each of the three factors yield a score between 0 and 1.

$$S = (1 + A_s) \times (1 + P_s) \times (1 + N_s) \tag{4.1}$$

The minimum line score is 1 for $A_s = P_s = N_s = 0$ and the maximum is 8 for $A_s = P_s = N_s = 1$. The torso line with the highest score is used to determine the participant's time.

4.3.1 Angle of the line

The first factor that influences the final score is the angle of the line. This is based upon the idea that when a participant crosses the finish line his torso is aligned vertically because he is in a running pose. The angle between the horizontal axis and the line determines the score. For a line between the points (x_1, y_1) and (x_2, y_2), the score is computed using Equation 4.2. A horizontal line of 0° has a score of 0 and a vertical line of 90° yields a score of 1.

$$A_s = \arctan\left(\frac{\Delta y}{\Delta x}\right) = \arctan\left(\left|\frac{y_1 - y_2}{x_1 - x_2}\right|\right) \tag{4.2}$$

4.3.2 Vertical position of the line

To favor lines at the torso, the vertical position of the line is taken into account. From observations, the first part of the torso that reaches the finish line is usually the chest, which appears at the row which is $\frac{3}{4}$ from the bottom of the participant component. For a line between the points (x_1, y_1) and (x_2, y_2), its vertical position subscore is determined by how close the center of the line $y_c = (y_1 + y_2)/2$ is to the row at $\frac{3}{4}$ of the component. The fraction of the height of the component that the line is off, determines the score. If the center of the line is at $\frac{3}{4}$ of the component, then the score is 1. A score of 0 is the subscore for a line at the bottom row, at the maximum distance from the row at $\frac{3}{4}$ of the component, as shown in Equation 4.3. The number of rows that the component has is denoted by r.

$$P_s = \frac{|y_c - \frac{3}{4r}|}{3r/4} \tag{4.3}$$

4.3.3 Neighborhood of the line

The pixels that are next to the line are also considered in the score. The lines at the front of the torso have black pixels to the left of them and white pixels to the right. The pixels up to distance d from the line are considered. For a line with height l, the amount of pixels considered A on each side is therefore $A = d \times l$. An example is shown in Figure 4.3, with $d = 3$.

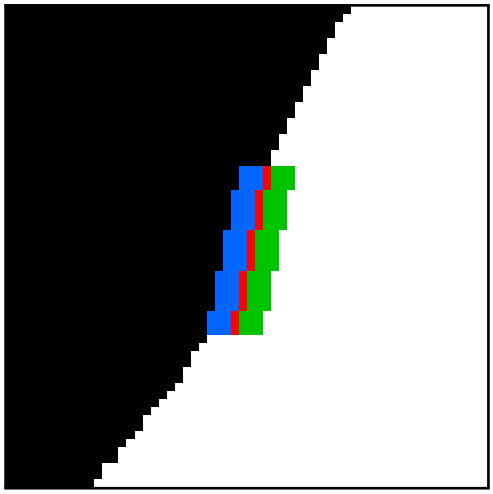


Figure 4.3: A line (red) detected by the PPHT and the considered left (blue) and right (green) neighborhoods.

The score is determined by the fraction of black pixels on the neighborhood on the left side and the fraction of white pixels on the right side of the line. The average of the two is the neighborhood subscore, as described by Equation 4.4. L_n is the number of black pixels on the left side of the line, R_n the number of withe pixels on its right side. A is the amount of pixels in the neighborhoods.

$$N_s = \frac{\frac{L_n}{A} + \frac{R_n}{A}}{2} = \frac{L_n + R_n}{2A} \tag{4.4}$$

If the pixels on the left side are all black and the pixels on the right are all white, the score is 1. When the areas are the wrong way around, the score is 0. This happens for example when a line on the right-most side of the component is considered.

4.4 Assumptions

The method described above determines the torso locations pretty well. However, some assumptions have been made. These assumptions simplified the torso detection, but also have the side effect that some torsos are not detected.

4.4.1 Extraction of individual participants

From the participant layer, the individual participants are extracted by selecting components that are connected. This assumes that the participants never overlap. However, this not the case, especially when participants finish close to each other. Because the two participants are merged into one, either the torso of one participant may be detected, or a line that is created accidentally by the merge may be selected, even though this is not a torso at all. This can create unexpected results.

4.4.2 Uncommon finish poses

Assumed is that a participant finishes in an upright running pose. However, the IAAF rules do not state that this is a mandatory pose to reach the finish line and therefore not all participants finish in this way. Sometimes people bend over to put their torso over the line. This means that the torso angle is much lower than 90°. Additionally, it is possible that a participant trips and falls on the finish line in which case the torso line could have any angle. In that case, the line is probably not centered close to the row at $\frac{3}{4}$ of the height.

5 Experiments

To test whether the program meets the requirements that were defined as the goals of this research in Section 1.3, some experiments are conducted. In the first experiment, the program is executed on different finish photos to test the accuracy of the torso lines that are placed by the program. In the second experiment, the program is executed on different variations on a single photo that works well. These variations are transformations of an original finish photo, like changes in brightness, contrast in size. This way, the limitations of the program can be determined.

5.1 Evaluation

In each experiment, the program is executed with a test set that contains different photos that wer not previously used in designing the program. It tries to determine the line that fits each participant as best as possible. The lines determined by the program for each photo are compared to the lines that have been assigned by the operator. For each participant in the photos, three cases are possible: the program was not able to find a line, it was able to find a single line, or two or more lines were determined by the program.

- The program has detected a single line for this participant.
  Both the operator and the program have determined a line that belongs to the participant. The deviation in time between the two is used as a measure for the program's time accuracy.
- The program has not detected any lines for this participant.
  In this case, there was a line assigned by the operator, but no line has been determined by the program. The participant could not be detected by the program. It might have been merged with another participant or the area of the participant was too small to search for a line.
- The program has detected too many lines for this participant.
  The operator has only assigned a single line, but multiple lines have been found by the program for a single participant. The program has detected multiple participants while there is only one. Except for one, the lines are redundant. This can be caused by a participant that has been split into multiple components in the foreground detected. The morphological operations in Section 3 failed to merge the components in a single participant. When a participant is assigned too many lines, the one closest to the line assigned by the operator will be used to determine the deviation.

The best line for each participant, if any, is compared to the participant's line assigned by the operator. The deviation in pixels between them is the error that the program has made. This distance is converted to time using the time scale that corresponds to the photo. The number of frames per time unit depends on camera settings and can therefore differ among photos. Therefore, representing the error by time suits better than using the distance between the lines in pixels.

5.2 Program accuracy test

One of the requirements of the program is that it has to be accurate. Without accurate results, there would be no benefit of running the program. In this experiment, the amount of lines detected by the program are compared to the amount of lines that were set by an operator of the timing system. Additionally, the deviation between the lines determined by the program and those assigned by an operator is determined to check whether the lines placed by the program are accurate enough to assign times to the participants.

5.2.1 Test set

The program is executed on 10 finish photos. Each photo has 4 to 7 participants, some of them close to each other, some far and some participants overlap each other. They wear different types and color of clothing. Additionally, the gender and age of the participants differs. All photos have a different resolution width, but some photos have the same resolution height, because these are predefined and occur often. The photos have a different brightness and contrast, because they were taken at different events and different times, so the weather and lighting is different for every race.

5.2.2 Results

Table 5.1 shows the results for the first experiment. There is a total of 48 lines placed by an operator. The program has determined 38 lines, which corresponds to 79.2%. The time deviation of these compared to the operator's line is between 0 ms (exactly the same pixel) and 43.6 ms. The average deviation is 7.15 ms and the median is 3.6 ms. The time deviations are evenly distributed along the photos, indicating that there is no bias toward a certain photo. Therefore, the program works well for all types of environments, participants, events and lighting conditions in the test set.

Photo	Operator lines	Detected lines	Deviation times (ms)	Missing lines	Incorrect lines
1	4	4	0.0; 3.6; 3.6; 9.1	0	0
2	6	2	1.8; 7.3	4	0
3	4	3	0.0; 12.7; 36.4	1	0
4	5	4	0.0; 0.0; 3.6; 43.6	1	0
5	4	3	1.8; 3.6; 14.5	1	0
6	4	4	3.6; 9.1; 12.7; 27.3	0	0
7	5	4	0.0; 1.8; 1.8; 9.1	1	0
8	5	5	1.8; 1.8; 3.6; 3.6; 10.9	0	0
9	7	5	0.8; 0.8; 1.7; 1.7; 15.0	2	0
10	4	4	1.7; 1.7; 3.3; 15.8	0	0
Total:	**48**	**38**	**271.6**	**10**	**0**

Table 5.1: Number of operator, detected, missing and incorrect lines per photo in the first test set.

Ten participants have not been detected, which is 20.8% of the participants. Photo 2 stands out, because out of the six participants could not be determined. This is a finish photo with a lot of participants running close to each other. The six participants are merged into two groups, with only a single line assigned to them. However, these two lines are still decent, with an around average time deviation. The last column of Table 5.1 shows that there were no incorrect lines, so there were no participants detected where there were none.

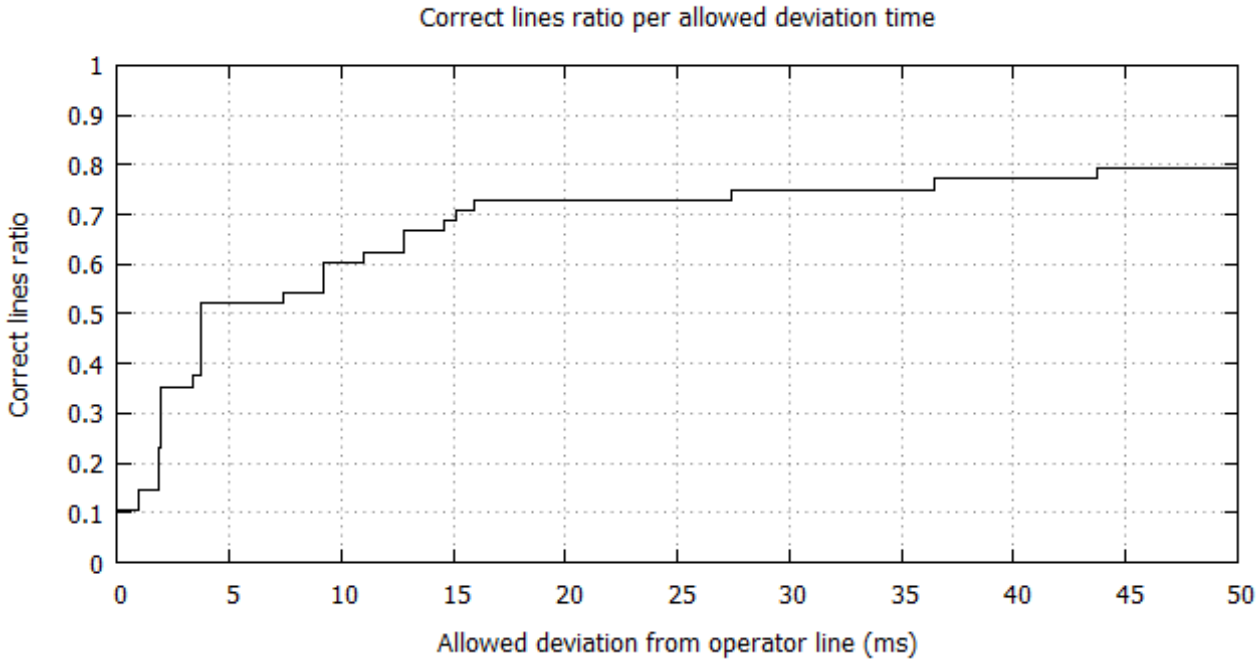


Figure 5.1: Ratio of matching lines per maximum allowed deviation.

Correct lines are the lines determined by the program that are considered to be within an allowed deviation margin. For a margin of E, the correct lines are the lines with a deviation time d that is at most the margin, so $d \leq E$. The lines that are not close enough, with $d > E$, are considered incorrect. Any participants that have one or more lines, of which none are within the allowed deviation margin, are considered missing lines. For

any given E, the correct lines ratio can be computed by dividing the number of correct lines by the number of participants. For the deviation times in Table 5.1, the correct lines ratio was computed for $d_{\min} \leq E \leq d_{\max}$, with a minimum deviation time of the detected lines in the set $d_{\min} = 0\,\text{ms}$ and maximum deviation time $d_{\max} = 43.6\,\text{ms}$. The correct lines ratio for the allowed deviation margins are shown in Figure 5.1.

As expected, the ratio of correct lines increases when the allowed deviation margin E is increased. For a maximum deviation of 10 ms, 60.0% of the participants can be detected correctly, while 79% can be determined correctly with a maximum deviation of 20 ms. With a margin of 44 ms, all participants that are detected have a correct time. This is in accordance with the maximum deviation time of the test set $d_{\max} = 43.6\,\text{ms}$ from photo 4. The last 21% of the participants have not been detected and have no line associated with them. Therefore, their line will never be correct for any value of E.

5.3 Assessing robustness

In the second experiment, the program's performance is examined in different conditions. It is required to work with different lighting, weather conditions and camera settings, so the program be used independently of the track, settings, time of day and weather. This is important because the lighting conditions change constantly between and during events, due to movement of the sun and clouds. A finish photo that is known to work well is edited to simulate 10 different distortions that might occur. The number of detected lines and their corresponding deviation times, as well as the number of incorrect lines are compared.

5.3.1 Test set

The original photo is Photo 1 from the first test, shown in Figure 5.2. It shows four female participants in different types and colors of clothing. None of them are close to each other when they finish.

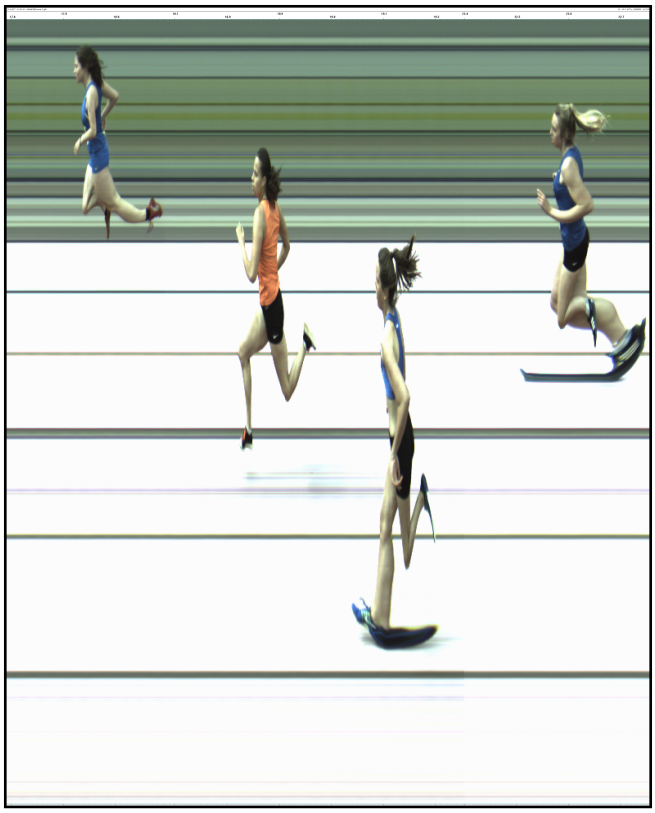


Figure 5.2: The original photo with four participants.

The 10 edited versions simulate the following conditions: severe increase and decrease of brightness and contrast, grayscale photo for certain camera types, decrease in size, increase and decrease of vertical resolution and a very dark and light environment. The lighter environment is simulated by combining increases in brightness and contrast, the darker environment was simulated by a decrease in brightness and a small decrease in contrast. These distortions are also shown in Table 5.2.

5.3.2 Results

The results for the second experiment are shown in Table 5.2. For each photo, it shows what distortion has been applied. Because the original photo contained four participants, the number of lines placed by the operator is four in every photo. The number of lines detected by the program is equal or larger than the number of participant. Therefore, there are no lines missing. However, some photos have additional lines that are incorrect.

Distortion	**Detected lines**	**Deviation times (ms)**	**Average deviation time (ms)**	**Incorrect lines**
Original image	4	0.0; 3.6; 3.6; 9.1	4.08	0
Brightness increased	4	1.8; 3.6; 5.5; 23.6	8.63	0
Brightness decreased	4	1.8; 3.6; 5.5; 7.3	4.55	0
Contrast increased	8	0.0; 7.3; 9.1; 29.1	11.38	4
Contrast decreased	12	0.0; 0.0; 5.5; 20.0	6.38	8
Image in grayscale	4	3.6; 3.6; 5.5; 34.5	11.80	0
Height and width decreased	4	3.6; 5.5; 5.5; 7.3	5.48	0
Height increased	4	3.6; 5.5; 9.1; 34.5	13.18	0
Height decreased	4	0.0; 1.8; 7.3; 9.1	4.55	0
Lighting increased	8	0.0; 3.6; 5.5; 10.9	5.00	4
Lighting decreased	4	3.6; 3.6; 10.9; 34.5	13.15	0

Table 5.2: Number of detected, missing and incorrect lines per photo in the second test set.

The distortions of brightness, grayscale, size and vertical resolution do not change the number of participants detected. However, most of the deviation times are worse. This indicates that the program still works, but is less accurate. The contrast increase in the photos with the contrast and lighting distortions created incorrect lines, as shown in Figure 5.3. The contrast increase lets parts of the participants look to much like the mostly white background, which splits the participants to the point that the morphological operations could not resolve them anymore.

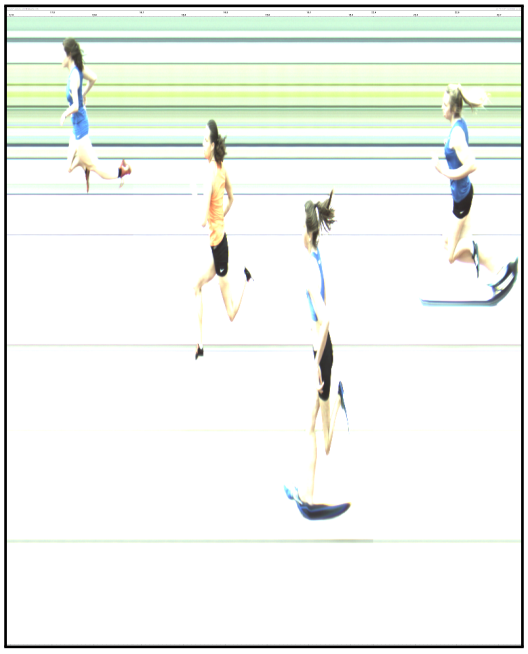


Figure 5.3: Photo with increased lighting as distortion.

Something similar also what happens in Photo 5. The contrast decreases to the point that most parts of the participants look too much like the background. Some small components are still bright enough to be detected, but the morphological operations cannot merge them. In Photo 11, the brightness also decreases, but the small increase in contrast apparently ensures that the small components can be merged and no redundant lines are detected.

6 Conclusion

This section provides a summary of this thesis. The problem and the approach to solve it are described. The steps that the program executes and the experiments conducted are summarized. The section concludes with possible future work and research that can be conducted to improve and expand the program.

6.1 Summary

In track running, a finish photo is used to determine the race times and placement of the participants. It is created by arranging the frames recorded by a line-scan camera chronologically. The resulting picture shows the finish line and finishing participants. Their positions on the finish line are shown on the vertical axis, while the horizontal axis represents the race time. A time line at the location of the torso of each participant associates them with the race time on the horizontal time axis. This time stamp is their race time and is used to determine the order of arrival.

The finish photo is interpreted by an operator of the timing system. This takes time which causes unwanted delay between the finish and the publication of the results. This thesis attempts to shorten this delay by developing a computer program that analyzes a finish photo to estimate the finish times of the participants. To do this successfully, the program must be able to recognize the torsos of the participants, estimate the finish times accurately, work in different environmental conditions and generate the estimation quickly. When the finish photo is loaded from the timing system, it distinguishes the competitors from the background. Then, it tries to recognize their torsos to determine the time lines. Finally, the estimated time lines are shown.

Multiple approaches to distinguish the participants from the background are considered. The method with the best results is tolerant mode matching, which is based on the idea that the background pixels do not change much over time. The modes of the channels in each row are determined. When the value of a channel in a pixel deviates too much from the channel mode of that row, the pixel is considered as foreground, therefore part of a participant. When the differences between the channel modes and all the channel values of a pixel are within a certain threshold, they are labeled as background and are not part of a participant.

The foreground layer produced by tolerant mode matching contains some noise. To improve the layer, morphological operations are applied. An opening operation is applied to remove line artifacts, removing areas of pixels that were incorrectly labeled as foreground. Then, a closing operation is used to close holes in participants, caused by pixels that were incorrectly labeled as background. The size of the kernel used for the closing operation is important. A kernel size that is too small is not able to reconnect participants that have been split into multiple components, while a kernel size that is too large will merge multiple participants. An optimal kernel size is determined by an iterative algorithm that applies closing operations with different kernel sizes. These are evaluated using a ranking system with a score that takes the size of the resulting components into account. The algorithm converges the kernel size to an optimal size, which is used in the final closing operation.

The resulting components are assumed to be single participants. To determine a time line, the front of their torsos need to be detected. A Hough filter is applied to each component to search for lines. Each line gets a score assigned that indicates how likely it is that the line indicates the part of the torso that reaches the finish line first. This score is based on three factors, which are the angle of the line, its vertical position in the component and the neighboring pixels. Assumptions made for the design of this heuristic are that each component consists of a single participant and that a participant finishes in an upright running pose. The line with the best score is used to set the time line for the participant.

Two types of experiments are conducted to assess the accuracy and robustness of the program. The performance of the program on a finish photo is evaluated by the number of lines that it has missed, the number of lines that it has placed incorrectly and the deviation of the lines that it has determined compared to the lines that an operator would assign. In the accuracy test, the test set consists of 10 photos with a total of 48 participants. The program was able to determine 38 of them (79.2%). No incorrect lines were determined, so there were no participants detected where there were none. The deviation of the lines determined by the program to those assigned by an operator was between 0 ms and 43.6 ms. The average deviation was 7.15 ms and the median was 3.6 ms. In the robustness assessment, one finish photo was edited to simulate distortions that might occur in different external conditions and using different camera settings. These 10 distortions slightly increased the deviation of the detected lines. No lines were missing, but the distortions with a major increase and decrease in contrast caused the program to determine multiple incorrect lines.

6.2 Further research

Even though the time lines determined by the program are relatively accurate and the program works in different environmental conditions, the program can still be improved. At the moment, participants that overlap or finish too close to each other are considered as a single component and get only one time line assigned. Further research to detect multiple participants in these types of components is required.

Additionally, the accuracy of the program can be improved. Not all lines that were determined by the program corresponded to the race time that the operator of a timing system would assign. A better method for torso recognition is required to improve this. The current implementation of the torso line recognition depends on the assumptions that a component only contains a single participant and that the participants finish in an upright pose. While this is often the case, this is not always true.

An extra feature that could be implemented is the recognition of lanes that the participants run in. Each lane corresponds to a single participant. This way, a time line can be assigned to the right participant. Additionally, the program can be improved to output actual race times instead of the time lines. The race time of a participant can then be assigned to a lane, which makes it possible to assign a race time to the name of a participant. Manual interpretation of the finish photo by an operator is then no longer required.

References

1. *Understanding Line Scan Camera Applications* Teledyne DALSA (2014).
2. Lefort, M. *Practical guide for the finish judge* Union Cycliste Internationale (n.d.), 8–9.
3. FISA. *Rules of Racing and Related By-Laws* Fédération Internationale des Sociétés d'Aviron (International Rowing Federation) (2013), 93.
4. IAAF. *Competition Rules 2016-2017* manual. International Association of Atheltics Federations (2016).
5. Gonzales, R. C. & Woods, R. E. *Digital Image Processing* 3rd ed. (2008).
6. Duda, R. O. & Hart, P. E. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Communications of the ACM* **15,** 11–15 (Jan. 1972).
7. Matas, J., Galambos, C. & Kittler, J. V. Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. *Computer Vision and Image Understanding* **87,** 119–137 (1 2000).