ZebraRecognizer: efficient and precise localization of pedestrian crossings.

Dragan Ahmetovic*, Cristian Bernareggi*, Andrea Gerino* and Sergio Mascetti*
*University of Milan
{dragan.ahmetovic, cristian.bernareggi, andrea.gerino, sergio.mascetti}@unimi.it

***Abstract*—Autonomous mobility is a challenge for visually impaired people. In the last years, a number of solutions have been proposed in the scientific literature to support visually impaired people during road crossing. In our previous work we presented *ZebraLocalizer* a mobile application that detects zebra crossings and guides the user to safely cross.**

In this paper we present the *ZebraRecognizer* algorithm that improves the detection module of our solution and that applies innovative solutions in the field of zebra crossings recognition. The major contribution resides in the fact that *ZebraRecognizer* rectifies the ground plane hence removing the projection distortion of the extracted features. This leads to two major advantages: first, it possible to compute the relative distance between the user and the zebra crossing in meters. Second, the grouping and validation criteria specifically designed for the rectified line segments are much more effective, hence improving the accuracy of the recognition. An additional contribution consists in a significantly improved computation time. Indeed, *ZebraRecognizer* is 3 time faster than our previous solution, thanks to the adoption of a personalized version of the EDLines algorithm to detect the line segments.

I. Introduction

In recent years mobile devices have shown their huge potential in supporting people with disabilities. Indeed, since these devices are accessible, a number of assistive technologies have been proposed to support people with disabilities in everyday activities. For example, thanks to screen reader software (e.g., Apple's "Voice Over") smartphones and tablets are accessible to visually impaired users. This renders it possible to develop ad-hoc applications that use the device to collect information from the environment (e.g., through the camera, the GPS or the accelerometers) and provide it to the user in an appropriate form (e.g., through audio information).

In this paper we focus on the problem of supporting visually impaired users to identify and localize zebra crossings. This problem is particularly relevant since blind and partially sighted people meet a number of challenges in walking independently without a guide. As shown in [9], [3], the main difficulties are concerned with acquiring information about the surrounding environment through tactile and kinesthetic perception as well as through the auditory channel. Due to the inability to localize distant objects, blind and partially sighted people run into difficulty especially in crossing a road. An experimental analysis shows that blind people wait at least three times more than sighted ones before crossing a road and that 6% of attempts are dangerous [4]. In an analogous experiment, Schroeder et al. also remark that blind people did not even try to find a zebra crossing [10].

To address this problem, solutions have been proposed in the literature to recognize pedestrian crossings through a smartphone camera and to guide the user. The first solution (Stephen Se [11]) can recognize a zebra crossing only if the image contains the whole pattern, not covered even partially by objects (e.g. cars). A more efficient approach (Uddin et al. [12]) is based on bipolarity segmentation (detect areas of alternating black and white stripes) and projective invariant validation (verify the cross ratio invariant between detected stripes). This solution yields good results in terms of precision and recall, although the experimental evaluation has been conducted on a small image set (about 100), all with similar illumination conditions. In [6], [7] Ivanchenko et al. propose two techniques for detecting pedestrian crossings: the former detects zebra crossings but does not compute their relative position with respect to the user. The latter detects the "two stripes" zebra crossings and adopts a rectification technique that, while not described, seems to share the same idea as our contribution. The solution we proposed previously [1] computes the approximate relative position of the user with respect to the zebra crossing. This information is then used to compute audio cues for guiding the user to a safe position for crossing and during actual crossing.

In this contribution we focus on the module that detects zebra crossings and computes their relative position. We propose the *ZebraRecognizer* algorithm (see Section IV) that significantly improves existing solutions along two directions: (a) *ZebraRecognizer* computes the position of the crossing with respect to the user in meters and (b) *ZebraRecognizer* significantly improves the performance in terms of accuracy of recognition and computation time.

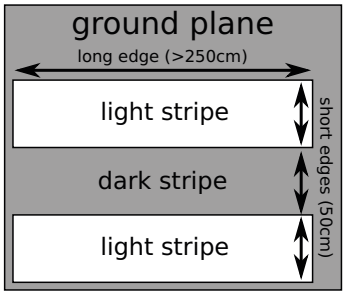
To obtain (a), *ZebraRecognizer* adopts a rectification phase that removes the projection distortion from the zebra crossing, hence making it possible to compute the relative position of the crossing with high precision and in meters. This is achieved through an innovative solution to compute the rectification matrix (see Section III). The impact of this solution goes beyond the aim of this paper as it is applicable in many cases in which objects on the ground plane need to be rectified.

Concerning (b), the rectification phase makes it also possible to define more effective constraints to validate the candidate zebra crossing, hence significantly improving the accuracy of the recognition. *ZebraRecognizer* is also about 3 time faster than our previous solution thanks to the adoption of a personalized version of the EDLines algorithm to compute the line segments [2]. We experimentally show the performance improvements in Section V.

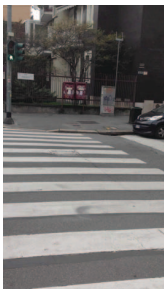
1051-4651/14 $31.00 © 2014 IEEE
DOI 10.1109/ICPR.2014.443

II. Problem specification

In this paper we consider zebra crossings as defined by Italian traffic regulations (see Figure 1a), however our solution can be easily adapted to most definitions used worldwide. A zebra crossing is described as a horizontal traffic sign consisting in an alternating pattern of dark and light stripes. It is composed by at least 2 light stripes and 1 dark stripe.



(a) Zebra crossing pattern specifications



(b) Zebra crossing

Fig. 1: Zebra crossing example and definition

The stripes are commonly rectangular and, less frequently, in case of diagonal crossings, parallelograms. They are $50cm$ thick and have a width of at least $250cm$. The dark stripes are of the same color of the underlying road while the light stripes may be white or, in case of road works, yellow.

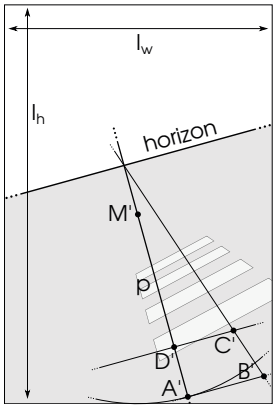
For the detection of zebra crossings, we rely on two data sources available on off-the-shelf smartphones: video camera and accelerometers. The former captures image frames that can then be analyzed with computer vision techniques. The latter, instead, detects the acceleration of the device, from which, through sensor fusion techniques available on recent mobile OS APIs, we extract the gravity acceletation component. This information is used to compute the orientation of the device with respect to the ground plane. Given these two data sources and the estimated user's height, we can compute the relative position of the user with respect to the detected crossings.

Technically, the input of the *ZebraRecognizer* consists in the user's height h, an image i with height i_h and width i_w and the gravity acceleration data represented as a three dimensional unit vector $a = \langle a_x, a_y, a_z \rangle$. Its elements a_x, a_y and a_z represent, respectively, the portion of the gravity that is applied on x, y and z axes of the device (see Figure 3).

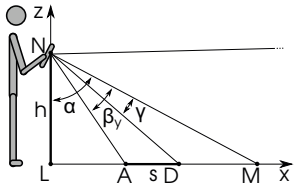
The output of the algorithm is an object representing the most suitable detected zebra crossing, if any. The crossing lies on the ground plane, with origin set in the position of the device and it consists in an array of stripes, each defined by its top and bottom segments and its color.

III. Metric rectification

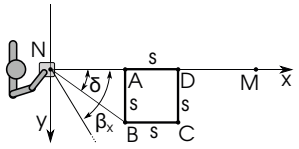
One of the most innovative aspects of the solution proposed in this paper is that it applies a rectification transformation to the pedestrian crossing to remove the perspective distortion, hence easing the validation of the geometrical constrains and making it possible to precisely compute the relative position of the crossing with respect to the user.



(a) Square inside image



(b) Reconstructed plane: Profile



(c) Reconstructed plane: Top view

Fig. 2: Square $A'B'C'D'$ inside the image and the corresponding square $ABCD$ on the reconstructed ground plane

The rectification matrix is calculated as the product of two matrices: the affine rectification matrix and the metric rectification matrix. In Section IV we show how the affine rectification matrix can be computed by directly adapting existing results to our pattern recognition problem. Vice versa, in order to compute the metric rectification matrix we propose a technique that, to the best of our knowledge, is original. An important aspect of this technique is that it can be used for the rectification of the ground plane in any case in which the gravity acceleration applied on the mobile device and the height of the device from the ground are known. Consequently this technique can be used in many different contexts in which mobile devices are involved and its field of application is much broader than the crossing recognition, which is our focus.

Liebowitz and Zisserman ([8]) show how to compute a metric rectification matrix given two pairs of lines in the image that have the following properties in the reconstructed plane: they lie on the ground plane, none of them is parallel to any of the three others and both pairs are perpendicular. Examples are provided in which the two pairs of lines are actually present in the image and there is prior knowledge about the above properties of the two pairs of lines. Our approach is different: we first artificially craft these lines on the image in such a way that the above properties hold. Then we apply the technique in ([8]). To craft the lines, we need the knowledge of the device's inclination with respect to the ground plane, the height of the device from the ground plane and the camera angle of view. In Section IV we show how to compute the first two parameters, the last one is a constant (for a given camera).

Our technique works as follows: we ideally draw four vertexes in the source image in a way that the projection of these points in the reconstructed ground plane yields a square $ABCD$ with edge of length s (see Figure 2). Once we have this square, the two pairs of lines are $\langle \overline{AB}, \overline{AD} \rangle$ and $\langle \overline{AC}, \overline{BD} \rangle$ and the technique proposed in ([8]) can be applied.

We call A', B', C', D' the four points that we construct in the image and that are projected on A, B, C and D, respectively. We construct A', B', C' and D' to force the following constraints, implying that $ABCD$ is a square:

1) $\overline{AB}$ is parallel to $\overline{CD}$;
2) $\overline{AB}$ and $\overline{CD}$ are perpendicular to $\overline{AD}$;
3) $|\overline{AD}| = s$;
4) $|\overline{AB}| = s$;
5) $|\overline{CD}| = s$.

To enforce 1), we draw $\overline{A'B'}$ and $\overline{C'D'}$ parallel to the horizon. For 2), we draw $\overline{A'D'}$ on the line p perpendicular to the horizon and passing through the center of the image M'.

To enforce condition 3) we first draw A' on the intersection between p and the circle centered in the center of the image with radius equal to half of the height of the image i_h (see Figure 2a). We compute D' as follows. Consider Figure 2b where h is the height of the device from the ground, α is the inclination of the device and β_x and β_y are half the horizontal and vertical camera angles respectively. Since we know $\overline{NL} = h$, $A\hat{L}N = \pi/2$ and $L\hat{N}A = \alpha - \beta_y$, we can easily compute $|\overline{LA}|$. Then, $|\overline{LD}| = |\overline{LA}| + s$ while the angle $\gamma = \alpha - atan(|\overline{LD}|/h)$. Consequently we compute D' as:

$$|\overline{M'D'}| = (\sin(\gamma) \cdot i_h)/(2 \cdot \sin(\beta_y))$$

Since condition 4) is analogous to condition 5), we only show the enforcement of the former, conceptually similar to condition 3). In this case we have the right triangle NAB (see Figure 2c) in which $|\overline{AB}| = s$ and $|\overline{NA}|$ can be derived since we know the width of the image i_w, $|\overline{AL}|$ and $|\overline{NL}|$. Consequently we can compute angle $\delta = atan(s/|\overline{NA}|)$ and therefore we can derive $|\overline{A'B'}|$ as:

$$|\overline{A'B'}| = (\sin(\delta) \cdot i_w)/(2 \cdot \sin(\beta_x))$$

IV. The *ZebraRecognizer* algorithm

ZebraRecognizer is divided in 4 main steps, as shown in Figure 3. "Image preprocessing" and "Rectification matrix computation" can be run in parallel, and, when both results are available, two steps follow: "Line segment detection" and "Line segment grouping and validation".

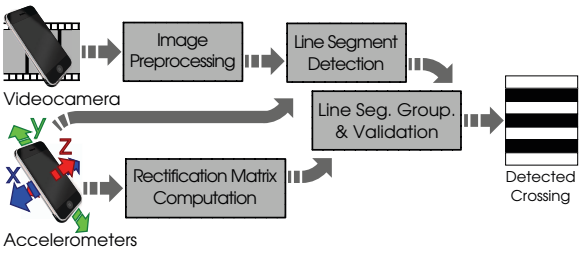
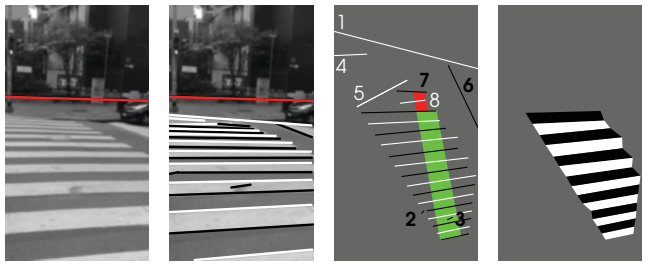


Fig. 3: Algorithm flowchart

A. Image preprocessing

In the image preprocessing step the image is prepared to ease the following computations. The input is the image captured by the camera while the output is the preprocessed image that we call "the image" in the following. The preprocessing consists in three main operations: first, the image is converted to grayscale. This loss of color information actually helps to detect crossings having differently colored stripes. The second and third operations are respectively the image resizing ("resolution" parameter, see Section V-A) and gaussian blur filtering[1]. The purpose of this step is to reduce execution time in the following stages of the algorithm and smooth imperfections in the image to ease the detection of line segments (see Section IV-C). Figure 4a shows an example of the preprocessing step applied to Figure 1b.



(a) Preprocessing (b) Seg. detection (c) Validation (d) Result

Fig. 4: Example of *ZebraRecognizer* applied to an image

B. Rectification matrix computation

The input of this step consists in the gravity acceleration values and its goal is to compute a 3x3 "rectification matrix". While it could be possible to multiply this matrix to every point of the ground plane in the image to remove the perspective distortion, it is computationally inefficient. Hence this step outputs the rectification matrix that, in the following steps, is applied to few selected points representative of the crossing.

The rectification matrix is calculated as the product of two matrices: the affine rectification matrix and the metric rectification matrix. In ([8]) it is shown how to compute the affine rectification matrix for a plane given its vanishing line. In our case the vanishing line of the ground plane is the horizon and we compute it from the gravity data as follows.

The gravity acceleration components a_x, a_y and a_z as acquired by the device accelerometers are measured in $g = 9.80665m/s^2$ and take values in $[-1, 1]$. Since the gravity force is normal to the ground plane, these values represent the sine of the inclination (with respect to the ground plane) of the corresponding axes of the device. Hence, given an image having width i_w and height i_h, captured by the camera with respectively half horizontal and vertical angles of view β_x and β_y, the inclination of the horizon on the screen is calculated as $\theta = atan2(a_x, -a_y)$ and it passes through a point

$$p = \left(\left(\frac{\sin(\theta) \cdot a_z}{\sin(\beta_x)} + 1\right) \cdot \frac{i_w}{2}, \left(\frac{\cos(\theta) \cdot a_z}{\sin(\beta_y)} + 1\right) \cdot \frac{i_h}{2}\right)$$

Therefore, the image's horizon line's equation is:

$$hl = \sin(\theta)x + \cos(\theta)y - \sin(\theta) \cdot p_x - \cos(\theta) \cdot p_y$$

For the computation of the metric rectification matrix we use the technique presented in Section III that requires as input the height of the device. To estimate this value we assume

[1]The resizing uses pixel area relation interpolation and the gaussian filtering parameters used are $\sigma = 1.1$ for sigma and $k = 5$ for kernel size.

that the user is holding the device in a position like the one depicted in Figure 2a in which the elbow is close to the hip and the forearm has an inclination of about $pi/6$ with respect to the ground plane. In our experiments with the users, this is the natural position of almost every user. Also, we need to know the height of the user hu (either estimated or asked to the user during the setup stage). Considering the proportions of the human body ([5]), on average the height at elbow is $he = 0.615 \cdot hu$ and the forearm length is $hf = 0.205 \cdot hu$. Consequently, the device height from the ground is:

$$hd = he + sin(\pi/6) \cdot hf$$

Clearly the above computation is subject to some approximation. However, the error is practically not significant. For example, considering a 175cm tall person we assume the height of the device is 125cm. If the device is actually kept at the height of the shoulders (142cm), a zebra crossing at a distance of 2m is computed as being 233cm from the user. This does not significantly affect, for example, the number of steps required to reach the starting of the zebra crossing.

C. Line segments detection

The line segments detection step is a modified version of the EDLines algorithm (Akinlar et al. [2]). The inputs of this step are a grayscale image and the horizon. The output is an array of detected segments in the image coordinate system. Figure 4b shows the results of the line segment detection algorithm applied to Figure 1b. Our implementation has three main differences with respect to the original.

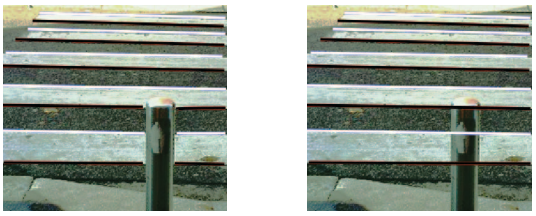
First, our technique ignores the portion of the image above the horizon since no zebra crossings will ever be found there. This approach is both useful to reduce the computation time and exclude possible false positives in that area.

Second, in addition to gradient orientation, our solution also computes the gradient direction of the detected segments. This is a useful information in the following steps since the direction of the gradient can differentiate between segments on top and on the bottom of stripes.

The third difference between our solution and the EDLines algorithm is that our technique also merges close segments. Two segments having slope distance and spatial distance both lower than specified thresholds are merged. The resulting segment is the union of the two segments' projections on the line having the slope angle and intercept parameter calculated as the weighted averages (based on segments' lengths) of the corresponding parameters of the lines on which the two segments lay. This step is useful to join two or more portions of a line segment that have been recognized as different line segments due to minor imperfections in the image, noise, flawed coloration of the stripes or objects between the observer and stripes. See Figure 5 for an example.

D. Line segments grouping and validation

The aim of this step is to group the segments to form candidate crossings and then to validate each group. To ease the definition of the geometrical properties underlying grouping and validation, each segment is first rectified using the rectification matrix. This makes it possible to straightforwardly check geometrical properties (e.g., parallelism) and to compute distances in meters. During the process, the set of segments is re-organized into a set of crossings, each one characterized by a set of stripes, that, in turn, are composed by a pair of segments each. The output is the most relevant crossing.



(a) Split segments (b) merged segments

Fig. 5: Segments split by an obstacle are merged

Before grouping, the segments shorter than required are pruned. In Figure 4c, segments 2 and 3 are removed due to this criteria. The remaining segments are grouped based on three criteria: "slope", "horizontal overlapping", and "gradient orientation". The idea behind the "slope" criteria is that the segments in the same crossing are all parallel among them. We use a hierarchical clustering technique to group segments whose difference in orientation is smaller than a threshold. Segments 5 and 6 in Figure 4c are excluded due to the "slope" criteria. The same approach is used to check the "horizontal overlapping" criteria that captures the fact that, in a crossing, the projections of all segments on a line parallel to them overlap. Segments 7 and 8 in Figure 4c have an overlap smaller than the "overlap threshold" parameter (see Section V-A) and are filtered out. With the "gradient orientation" criteria we ensure that two consecutive segments in a group have opposite gradient directions. Each resulting group is then structured into stripes consisting in pairs of consecutive segments.

The stripes are then validated according to "color consistency" and "width" principles. "Color consistency" validation checks if each light (or dark) stripe has a color consistently lighter (darker, respectively) than the average color of the candidate crossing. The minimum difference between the color of the stripe and the crossing is specified by the "color consistency magnitude threshold" parameter (see Section V-A). Thanks to "color consistency" validation, discolored and cracked stripes as well as structures that are geometrically similar to stripes but without consistent dark/light alternating colors are discarded. "Width" validation checks whether each rectified stripe is about $50cm$ wide. The stripe containing segment 1 in Figure 4c is pruned due to this constraint.

During the grouping and validation phases, we prune the groups containing less than minimum number of segments. In most of our settings this value is set to 5, guaranteeing that each crossing contains at least two white stripes.

In many cases either none or a single group is output by the grouping and validation step. However, it is possible that two or more groups are returned. This happens, for example, when the camera is viewing two distinct crossings or a single crossing is erroneously split in two parts. The last step of the algorithm therefore chooses a single crossing that is likely to be the most relevant for the user. We assume that the most relevant

crossing has roughly the same direction as the user. Therefore, we first check if any detected crossing has an orientation angle within a threshold from the user's orientation. If favorable crossings are available, then all other crossings are discarded. From the remaining crossings, the closest one to the user is returned. In Figure 4d we show the rectified image of the crossing detected starting from Figure 1b.

V. Experimental evaluation

In this section we report our experimental evaluation aimed at assessing the performance of our solution both in terms of computation time and reliability of the results. We also compare *ZebraRecognizer* with our previous solution (presented in [1]) using the same experimental setting, showing significant improvements. A direct comparison with other solutions is unfeasible because it is not possible to test other solutions with our experimental setting due to the unavailability of the implementations of previous solutions and, at the same time, it is not possible to test our solution with the same test sets used in previous results since they are not available. For a direct comparison of our solution with future work, we make our test set publicly available[2]. This is composed by a set of videos and the corresponding accelerometer data recordings.

A. Experimental setting

For the evaluation we used a dataset of 16 videos for which we also stored the frame-by-frame corresponding gravity data derived from the accelerometers. Videos were recorded in 720×1280 resolution at a framerate of 20 frames per second. In total 7408 frames were captured, 4480 of which contain zebra crossings while the remaining 2928 do not. Videos and gravity measurements were captured on an iPad 2 device in different illumination conditions (sun, rain and night).

We used a desktop pc for computationally intensive evaluations and a mobile device for evaluating the execution time on target platform. The desktop pc runs Gentoo linux with x86_64 3.12.13 kernel and openCV 2.4.6 on an Intel i5 2-core (4-thread) cpu and has 8GB ram. The mobile platform is an off-the-shelf iphone 5 smartphone.

Three indicators were evaluated: precision, recall and execution time. The precision, calculated as the the ratio of the correctly detected crossings and all the detected crossings, measures the amount of false positives. A precision score of 1.0 means that each detection corresponds to a crossings in the examined image, conversely a lower ratio implies that some crossings were detected where none was present. The recall metric is calculated as the ratio between the detected crossings and all the correct crossings in the dataset. While a score of 1.0 means that all the crossings were correctly detected, lower values indicate that some of the crossings were not. Given the safety concerns for the navigation of visually impaired users in a dangerous environment, we notice how anything less than a perfect precision score is unacceptable, while a high recall score, although important, is less critical. In the following, unless differently stated, we report our results in which the precision is always equal to one. The execution time defines the average time (among all 7408 frames) needed to run the *ZebraRecognizer*. Lower execution time allows higher frame rates, increasing the responsiveness of the detection with respect to the user's movements. Also, it means that the procedure is less computationally intensive, with a lower impact on the battery life of the device.

[2]http://webmind.di.unimi.it/ZebraTestSet/

Parameter	Min	Default	Max
Resolution	90×160	180×320	720×1280
Overlap length thres.	0	15	50
Color cons. mag. thres.	1	4	10

TABLE I: Most influential parameters and their values

While in our experiments we tuned all parameters, due to page constraints, we only report here our results when varying three of the most influential ones with respect to the considered metrics (shown in Table I together with their minimum, maximum and default values). The "resolution" parameter specifies the size of the image on which the detection is run. The "overlap length threshold" defines the minimum overlap length (in cm) required for segments belonging to the same group (see Section IV-D). The "color consistency magnitude threshold" defines the minimum difference in color intensity (value range between 0 and 255) that a stripe must have with respect to the whole crossing in order to be valid (see Section IV-D).

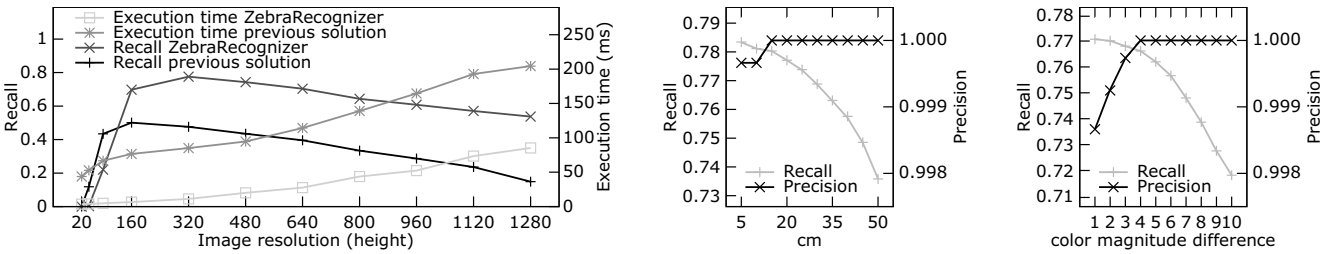
B. Experimental results

As expected, "resolution" significantly influences the execution time (See Figure 6a). This is due to the fact that the most computationally intensive task, the line segment detection, has a linear computational complexity with respect to the number of pixels of the input image.

For what concerns the recall metric, with a very low resolution (below 90×160) the features are hard to detect and hence recall diminishes drastically. On the contrary, for high resolutions (above 180×320) there is also a reduction in recall due to the fact that noise and imperfections are more visible and impair drastically the segment detection stage. This behaviour can be offset by using a stronger smoothing during the preprocessing step (see Section IV-A), however, at further expense of the execution time. The best results can therefore be achieved with a resolution of 180×320 for *ZebraRecognizer* and with 90×160 for our previous solution.

Comparing the two solutions (see Figure 6a), we notice that, at same resolution settings, *ZebraRecognizer* is 3 to 10 times faster. The maximum recall are 0.78 and 0.5 for *ZebraRecognizer* and our previous solution, respectively. The corresponding average execution times are about $12ms$ for *ZebraRecognizer* on the desktop platform while the previous one settled on a 7 times higher value ($76ms$). Lower execution time with respect to the previous works is obtained thanks to the usage of a faster line segment detection algorithm ([2]). Concluding, *ZebraRecognizer* significantly improves both recall and execution time with respect to our previous solution.

For what concerns the computation time on the mobile platform, with the settings that maximize recall, *ZebraRecognizer* is 3 times more efficient than our previous solution ($32ms$ and $87ms$ respectively). We suspect that the decrease in the gain of *ZebraRecognizer* with respect to the desktop platform (from



(a) Comparison with the previous solution (b) Overlap length threshold (c) Color magnitude thresholds

Fig. 6: Results of experiments

7 to 3 times faster) is due to the fact that *ZebraRecognizer* makes a larger use of floating point operations.

For "overlap length threshold" and "color consistency magnitude threshold" parameters we have similar behaviors. Indeed, increasing these two parameters, the grouping and validation step filters out more candidate crossings. Hence, for high values of these two parameters, we obtain better values for the precision metric but worse results for the recall metric. Numerically, for "overlap lengths threshold" lower than 15cm, precision decreases (Figure 6b). We choose as default setting a value of 15cm which yields 1.0 precision and recall of 0.78. As for the "color consistency magnitude threshold" (Figure 6c), values smaller than 4 yield some false positives, hence we chose this value as default to guarantee precision equal to 1. With this setting, the recall is 0.78.

VI. Conclusions and future work

In this paper we presented *ZebraRecognizer*, an algorithm to detect and localize zebra crossings. The algorithm makes use of data from both the camera and the accelerometers to rectify the extracted features. This enhances the process of feature aggregation and validation of candidate zebra crossings. The extensive experimental evaluation, conducted in different illumination conditions (including sun, rain and night) shows that *ZebraRecognizer* has zero false positives, hence never erroneously reports a crossing when not visible. Also, the algorithm has a significantly lower computation time and a higher recall with respect to our previous solution. Lower computation time results in a higher number of processed frames per second, hence guaranteeing a better responsiveness for the user, and a lower power consumption. Higher recall ensures that more crossings are correctly recognized.

Following the revisors' comments we intend to evaluate the precision of the position calculation and the impact of the occlusion of the stripes (e.g by vehicles or pedestrians). We are also investigating simpler and possibly more precise rectification techniques. As a future work we aim to embed *ZebraRecognizer* into a mobile app that can guide a visually impaired person towards the zebra crossing and during the actual crossing. For this goal it is necessary to devise a technique to compute safe and short paths, a step that will benefit from the position calculation capability of the *ZebraRecognizer*. Also, we intend to investigate the user interface issues involved in providing complex audio information to the user without diverting his/her attention from the environment. Finally, as a additional visual recognition problem, we are also working on the recognition of traffic lights.

Acknowledgment

We wish to thank the anonymous reviewers for insightful and detailed comments that were extremely helpful for the improvement of our work.

References

[1] Ahmetovic, D., Bernareggi, C., and Mascetti, S. Zebralocalizer: identification and localization of pedestrian crossings. In *Proc. of the 13th Int. Conf. on Human Computer Interaction with Mobile Devices and Services*. ACM, 2011.

[2] Akinlar, C. and Topal, C. Edlines: A real-time line segment detector with a false detection control. *Pattern Recognition Letters*, 2011.

[3] Arditi, A., Holtzman, J.D., and Kosslyn, S.M. Mental imagery and sensory experience in congenital blindness. In *Neuropsychologia*. Elsevier, 1988.

[4] Guth, D., Ashmead, D., Long, R., Wall, R., and Ponchillia, P. Blind and sighted pedestrians' judgments of gaps in traffic at roundabouts. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 2005.

[5] Huston, R. *Principles of biomechanics*. CRC press, 2008.

[6] Ivanchenko, V., Coughlan, J., and Shen, H. Detecting and locating crosswalks using a camera phone. In *Computer Vision and Pattern Recognition Workshop*. IEEE, 2008.

[7] Ivanchenko, V., Coughlan, J., and Shen, H. Staying in the crosswalk: A system for guiding visually impaired pedestrians at traffic intersections. In *Assist technol Res Ser.* IOS, 2009.

[8] Liebowitz, D. and Zisserman, A. Metric rectification for perspective images of planes. In *Proc. of Computer Vision and Pattern Recognition*. IEEE, 1998.

[9] Passini, R. and Proulx, G. Way finding without vision: an experiment with congenitally blind people. In *Environment and behavior*. Kluwer, 1988.

[10] Schroeder, B. J., Rouphail, N. M., and Emerson, R. S. W. Exploratory analysis of crossing difficulties for blind and sighted pedestrians at channelized turn lanes. *Transportation Research Board of the National Academies*, 2007.

[11] Se, S. Zebra-crossing detection for the partially sighted. In *Proc. of the conference on Computer Vision and Pattern Recognition*. IEEE, 2000.

[12] Uddin, M.S. and Shioyama, T. Detection of pedestrian crossing and measurement of crossing length - an image-based navigational aid for blind people. In *Trans. on Intelligent Transportation Systems*. IEEE, 2005.