Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Obstacle Warning for Texting

Bachelor Thesis

Christian Hagedorn

hagedoch@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Jara Uitto, Klaus-Tycho Förster
Prof. Dr. Roger Wattenhofer

September 22, 2014

Abstract

Walking and texting on the phone can be dangerous. People often hurt themselves because they do not pay attention to obstacles in their way. Fortunately most phones nowadays have an integrated back camera. This can be taken to advantage. The stream of pictures from the back camera can be analysed to recognize obstacles. This can be done by dividing the pictures into regions with similar colors. In successive pictures obstacles are matched and their movement directions can be determined. If a dangerous obstacle is approaching a quick vibration warning on the phone can protect the phone holder from serious accidents.

Contents

Chapter 1

Introduction

Recent studies have shown that the number of injuries related to cell phones has been increasing [**?**]. People are distracted by the phone while walking. They pay too little attention to the path in front of them. As a consequence they walk straight into obstacles like poles or walls and hurt themselves badly. To counteract we came up with the idea to use the phone to warn walking people of dangerous obstacles in their ways. We accomplished this task without using any additional hardware.

1.1 Implementing an App to Avoid Accidents

In this project we implemented an app[1] which takes pictures from the back camera. We get a stream of images from the path in front of us. The basic idea behind the app is to recognize obstacles due to their color difference. By dividing the picture into regions with similar colors we could separate obstacles from the background. One of the most difficult parts was the division of the directly following picture again into regions with same colors and detect the same obstacles from the previous picture in it. By doing this we could make predictions where an obstacle was moving to. If the obstacle will directly hit the phone holder we trigger a warning by letting the phone vibrate. Maintaining a large enough frame rate made the matching step easier and more reliable.
Our approach was safety-first. We did not try to reduce false warnings on cost of missed warnings of dangerous obstacles. A false warning will cause the user to look up from the phone for around half a second. On the other hand a missed warning of a dangerous obstacle could result in a broken nose.
Thoroughly testing 23 dangerous situations allowed us to set the different parameters inside the app to efficient values. The resulting final app settings were able to correctly warn the user at the right time in all 23 situations, having only one false warning per situation on average. These results were very satisfying

[1] We mention here that the current version of the app can only be run in foreground mode. Once the app is minimized it doesn't work any more. An improvement for supporting a service embedding is left for future work (see Section 5.2).

and followed our aimed safety-first approach.

1.2 Previous Projects

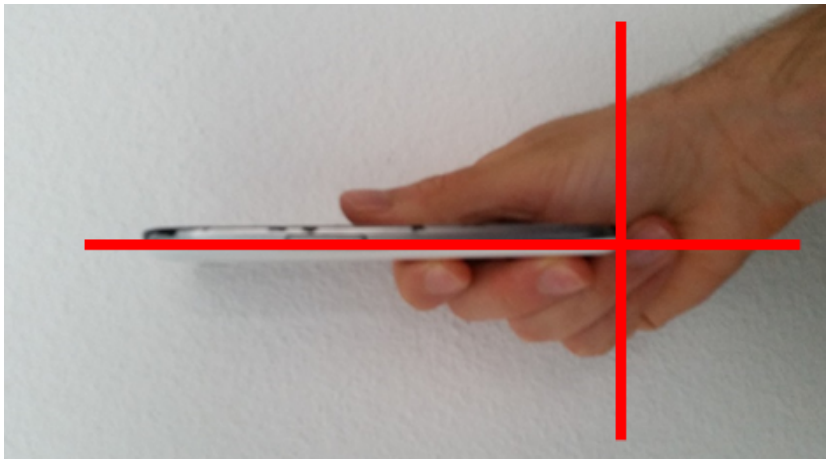
A similar project was done before, where the user gets warned from obstacles whose color differentiated from the background. We did not extend this project. Instead we started from scratch and came up with own ideas and implementations.

Chapter 2

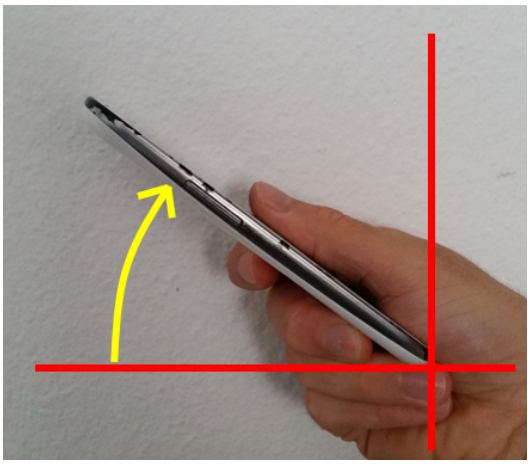
App Usage

2.1 How to hold the Phone

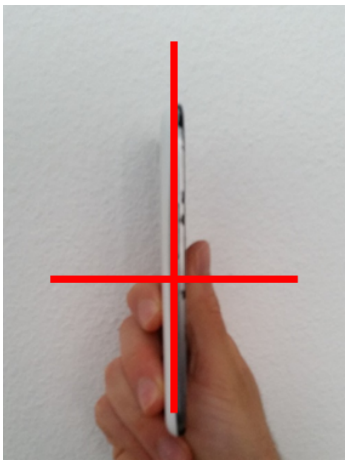
The most important aspect is how the user holds the phone. If you are texting while pointing with the phone to the sky, the app can only warn you about approaching clouds. A nice idea for weather forecast, but not exactly what we want. To figure out a good phone-holding position we made use of the integrated orientation sensor in the phone. This sensor returns a value of 0 degrees if the phone is parallel to the ground (Figure 2.1 (a)) and -90 degrees if the phone is vertical to the ground, pointing to the walking direction (Figure 2.1 (c)). A good value lies between these two extremes. In the previous project a value of -45 degree was taken. Nevertheless this value felt unnatural to us. Therefore we walked around pretending to be texting and measured the average value returned from the sensor. The resulting values were around -30 to -35 degrees (Figure 2.1 (b)). The corresponding field of vision on the phone screen with this setting is around 2.5 to 3 meters for a 1.85 meters tall human. Moreover we assume that the user walks upright with the phone in his hands in front of his body (Figure 2.2). Very important to mention here is that all functionalities of this app only work if the user holds the phone in portrait position (Figure 2.3).
The phone contains a sensor (namely the gyro sensor) for detecting fast movements of the phone (for example if you suddenly turn your phone quickly). The idea was to detect unexpected changes in the walking direction to disable triggered warnings during this period. However integrating this sensor was more of a curse than a blessing, since the gyro sensor is very sensitive. While walking the phone is shaking naturally and the gyro sensor peeked too often even though the walking direction wasn't changed. Therefore we stepped away from using this sensor and just assumed the user of the app isn't changing his walking directions randomly every second. This assumption seems realistic, since most people tend to be walking straight forward and if a direction change is done people are most likely to look up from the phone to verify if nothing dangerous is approaching their ways. Moreover we assume the user of the app is walking in a normal speed without being in a hurry.



(a) 0 degrees - phone parallel to the ground



(b) -30 to -35 degrees - ideal phone position



(c) -90 degrees - phone vertical to the gound

Figure 2.1: Visualization of the orientation sensor measurements. Figure (b) shows the optimal way to hold the phone.

2.2 Starting Point and Reaction to Warnings

We assume the user starts the application in a safe place. That means there are no dangerous obstacles in front of him[1]. Moreover there should not be complicated colored patterns on the ground to avoid false warnings (see Section

[1]More precisely there should not be a dangerous obstacle in screen-region 2, 3, or 4 (see Figure 3.1 and Section 3.2.2)

Figure 2.2: A proper phone holding position [**?**]

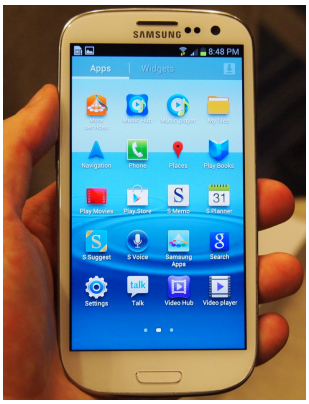


Figure 2.3: Portrait position of the phone [**?**]

3.3 for more details). A safe place to start is for example on a street which has more or less a uniform gray color. Once the user gets a correct warning, facing a dangerous obstacle, we assume that he chooses a new safe place before proceeding.

Chapter 3

App Design and Implementation

This chapter will cover how the app was designed and what the ideas were behind the implementation. First we look at the fundamental question when and how to warn the user. Afterwards we give an overview of the key functionalities and how they interact together. More important aspects are explained in greater detail.

3.1 Warnings

3.1.1 When Should the User be Warned?

Obviously the user should get warned when an obstacle is approaching directly towards him. There should be enough time to react when a warning is triggered. For that reason we tried to provide warnings around **1 to 1.5 meter in front of a dangerous obstacle**.

3.1.2 How Should the User be Warned?

A straight forward way to warn is the vibration of the phone which allows the user to react instantly. The duration of the vibration should be distinguishable from other common kinds of vibration (from Instant Messengers, SMS or other Notifications). To ensure this property the user can select the duration of the vibration by himself.

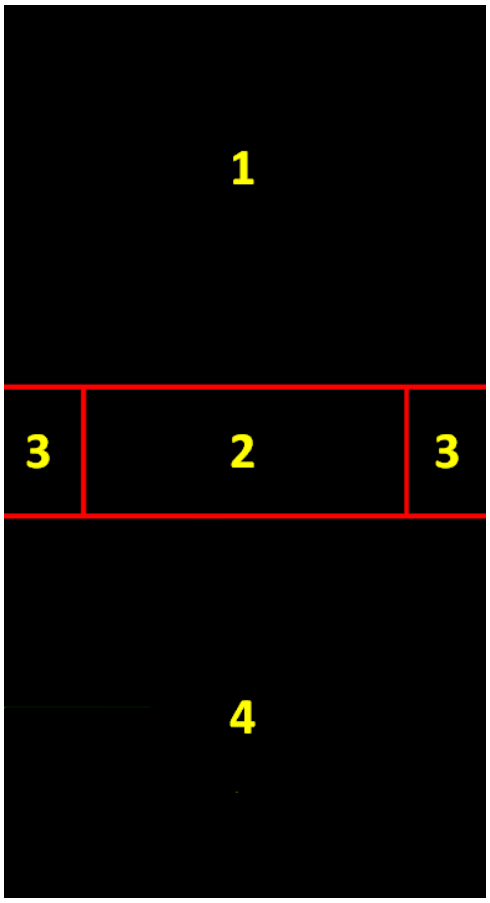


Figure 3.1: Screen divided into 4 screen-regions

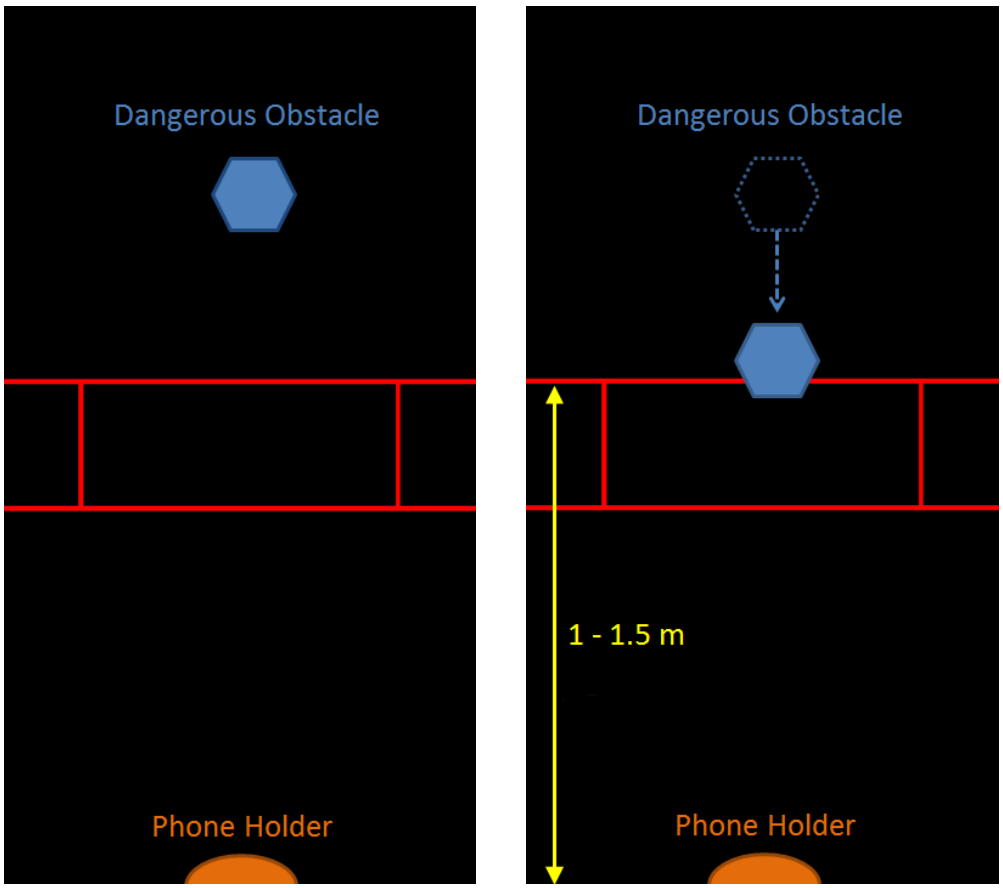
3.2 Details of the App Procedure

3.2.1 Divide the Phone Screen into Regions

In the following sections we will refer to numbered screen-regions. The screen-regions are shown in Figure 3.1. The purpose of this division is to process the right pixels or the right color regions (see Section 3.2.4), depending on their position on the screen. As mentioned in the previous chapter the whole screen covers a field of view of around 2.5 to 3 meters. Our goal is to warn the user around 1 to 1.5 meters in front of a dangerous obstacle. Therefore we created screen-region 2. When a dangerous obstacle touches this region and is directly moving towards us[1] we trigger a warning (Figure 3.2 (b)). Everything above in screen-region 1 is not yet dangerously close. Therefore we don't trigger a warning for obstacles that are only contained in screen-region 1 (Figure 3.2 (a)).

[1] What exactly "moving towards us" means is explained in Section 3.2.4

Screen-region 3 fills the space next to screen-region 2. We excluded the space in screen-region 3 from screen-region 2, because there might be obstacles which aren't moving directly towards us. For more details see Section 3.2.4.
Screen-region 4 covers all the space below screen-region 2 and 3. All obstacles that are fully contained in screen-region 4 are too close and thus not important any more. We either gave already a warning (Figure 3.3 (a)) or the obstacle wasn't moving towards us (Figure 3.3 (b)).



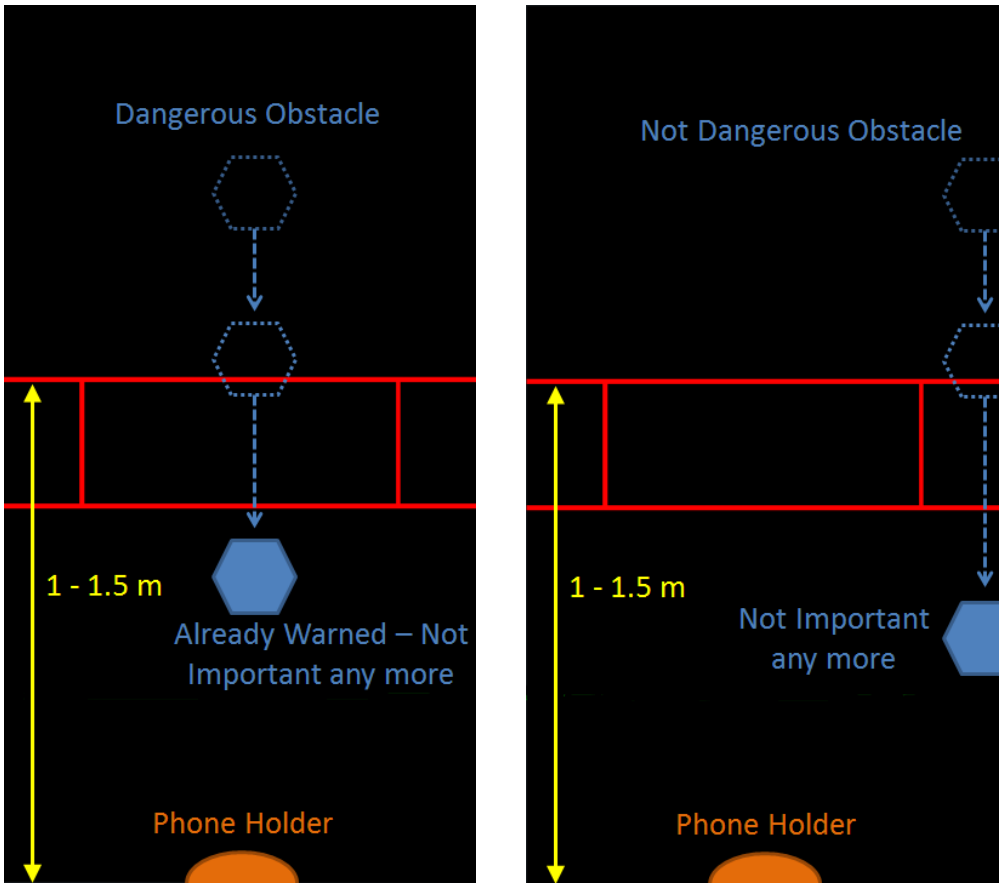
(a) Dangerous obstacle - No warning yet, because it's still fully contained in screen-region 1 and thus too far away.

(b) Dangerous obstacle moving towards us and has pixels in screen-region 2 - A warning is triggered!

Figure 3.2: Explanation for screen-region 1 and 2

3.2.2 Divide the Picture into Color Regions

After the preprocessing phase the main phase of the app is initiated. The goal is first to **divide the picture into regions with pixels of the same color**,



(a) Already warned dangerous obstacle - Not important any more

(b) Obstacle is not moving towards us (not dangerous) - Fully contained in screen-region 4 and not important any more

Figure 3.3: Explanation for screen-region 4

called *color regions*. The question is how to define "same color". Analogously you can ask how much a pixel can differ from its neighbour pixel to be counted to the same color region. An ideal threshold was found through testing (see Chapter 4 - Testing). The following algorithm is applied for every input picture. Some special care must be taken into account for the very first picture processed at the start or after a warning was triggered (see Section 3.2.3).

Algorithm for Defining Color Regions

```
1  Mark all pixels as not visited.
2  Start with a first pixel. (The first pixel is ideally one of the four edge
   pixels). Call this pixel reference-pixel.
3  Create a new color region which initially contains only the reference-pixel.
4  Proceed by Breadth-first search and compare each color of a new pixel
   p with the color of the reference-pixel:
5  while not all pixels visited do
6      while queue not empty do
7          if color of reference-pixel and p doesn't differ more than threshold
           then
8              Add p to color region.
9              forall the neighbour pixel p' of p do
10                 if p' not visited before then
11                     Add p' to queue.
12                     Mark p' as visited.
13                 end
14             end
15         end
16     end
17     //The color region is completed.
18     if color region has 10 or less pixels then
19         Discard that color region.
20     else
21         actualRegions-list.add(color region)
22     end
23     if all pixels visited then
24         //We are done!
25         Return.
26     end
27     Search a new pixel p which wasn't visited before.
28     Create a new color region with p as reference-pixel.
29 end
```

Algorithm 1: Pseude code for defining color regions

Keep track of the right color regions

Throughout the whole process we keep track of relevant color regions. This is done by maintaining two lists. The ***actualRegions-list*** contains all color regions from the last picture created by the algorithm. The ***interestingRegions-list*** contains interesting color regions[2] from the second last picture which represent possibly dangerous obstacles. The two lists are used in the matching process (see Section 3.2.4)

3.2.3 Define Interesting Regions in the very first Picture

We need to define continuously interesting color regions, which are potentially dangerous obstacles. After the first picture the *interestingRegions-list* is still empty. Thus we first need to initialize it. As seen in Section 2.2. we assumed that the closest dangerous obstacle can only lie in screen-region 1 in the first picture. Therefore it seems natural to define all color regions, which contain pixels only in screen-region 1, as interesting. We put them in the *interestingRegions-list*. All other color regions are uninteresting and can be discarded. We clear the *actualRegions-list* and start creating color regions for the next image, since there is nothing to match yet. Note that these steps are only done after the very first picture.

3.2.4 Match Color Regions from two successive Pictures

Given the *interestingRegions-list* and the *actualRegions-list* from the steps done before we do the matching as described in the algorithm on the following page. A few lines need some further explanation.
On line 4 we use center-points of a color region. The center-point is defined on the position of the pixels in the picture. For the x value take the median of all x value positions and for the y value take the median of all y value positions of the pixels in the color region.
On line 3 and 4 we calculate the color value difference and the distance between both center-points using the euclidean distance.
On line 5 we use two thresholds. We refer to Chapter 4 about testing for more details on the specific choice of the threshold values.
On line 24 we use screen-regions 1, 2 and 3. We did not pick only screen-region 1 because testing showed that some crucial warnings were missing. By additionally including screen-region 2 and 3 we could face this problem.

[2] What interesting means will be discussed further down

```
1  foreach iRegion in interestingRegions-list do
2      foreach aRegion in actualRegions-list do
3          colorDiff := Color value difference between both reference-pixels.
4          distanceDiff := Distance between both the center-points
5          if colorDiff < colorThresh and distanceDiff < distanceThresh
           then
6              //see Figure 3.5 and 3.6
7              Mark iRegion as possible matching candidate.
8          end
9      end
10     //try to find a match of iRegion with a candidate
11     if at least one matching candidate then
12         Sort the candidates according to colorDiff.
13         The candidate with the smallest colorDiff comes first.
14         foreach candidateRegion do
15             Draw a line L through the center-point of iRegion and the
               center-point of candidateRegion.
16             if L intersects with segment S then
17                 //see Figure 3.7
18                 if a pixel from candidateRegion is contained in
                   screen-region 2 then
19                     //see Figure 3.8
20                     Trigger a warning!
21                 end
22                 //avoid match that region again
23                 Remove candidateRegion from actualRegions-list.
24                 if candidateRegion has only pixels in screen-region 1, 2 or 3
                   then
25                     Mark candidateRegion as new interesting.
26                 end
27                 break;
28             end
29         end
30     end
31 end
32 Clear the interestingRegions-list and add all regions marked as interesting.
33 Clear the actualRegions-list.
```

Algorithm 2: Pseude code for matching color regions

3.2.5 Input Data

The pictures are captured from the back camera and have a resolution of 480x640 pixels. They are provided in a matrix format where each element represents a pixel holding its color value in RGB format. In the final version of the app this is the only source of input data. In the debug version there is an additional possibility to load saved images from a folder. This was used for testing (more details in Chapter 4 - Testing).

3.2.6 Preprocessing the Image

The preprocessing of the input image aims to simplify and speed up the image processing and matching steps afterwards. Once we get a picture from the camera it is first preprocessed. We start by **downscaling the image by a factor of 12**. There are some reasons for this decision. First we cannot afford to work with such large pictures. The phone takes too long to process this amount of pixels in real time. We need a certain amount of pictures per seconds. Having too few makes the color region matching harder, because the obstacles move too far from one picture to the next. On the other hand downscaling too much reduces the precision of the picture. Tests have shown that a downscaling factor of 12 gives the best balance between precision and pictures per second[3]. Additional tests have shown that fewer false warnings were thrown by downscaling the image.
After downscaling the image we performed a **Gaussian blur** of the image. This smoothed the image and removed sharp edges, allowing us to define color regions more efficiently.

3.3 Limitations

Since the app only uses the back camera of the phone for obstacle warning a few limitations seem to be natural. The camera only provides a picture with colors. Therefore the object detection can only be done by comparing colors. A few scenarios are hard to solve with the chosen approach for this app. Some examples are listed on the following page.

[3]With this setting we measured five pictures per second in the running app

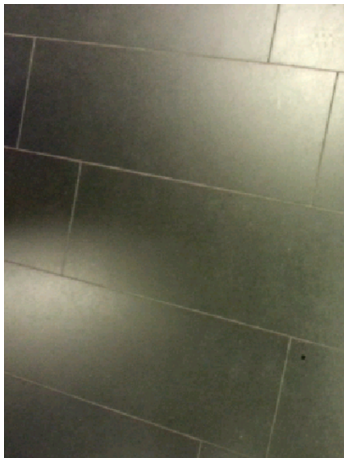
- Dangerous obstacles with an undistinguishable color from the background aren't recognized as obstacles[4] and don't trigger a warning (Figure 3.4 (a))
- Bright reflections from light or sunlight can be recognized as obstacles (Figure 3.4 (b))
- Dark areas caused from shadows can be recognized as obstacles (Figure 3.4 (c))
- Patterns on the ground in a different color can be recognized as obstacles (Figure 3.4 (d))

3.4 Thresholds

A remaining fundamental question is how to choose the different thresholds in the app. There are thresholds in the creation of color regions, in the matching step and in the preprocessing. Moreover the choice of the screen-regions and the appropriate lines in Figure 3.1 is important. All these different possible setups of the app must have been **tested thoroughly**. The next chapter describes the methods used to figure out good values.

[4]The reason for this behaviour becomes more clear with an extreme example. Imagine you are in a completely unicolored white room (white ground, white walls ect.). The camera points to the ground. The app will detect one white obstacle filling the whole screen. You start walking towards the wall. Once the wall appears on the screen the app won't detect it as new obstacle since it has the very same color.

(a) No warnings from obstacle with an undistinguishable Color

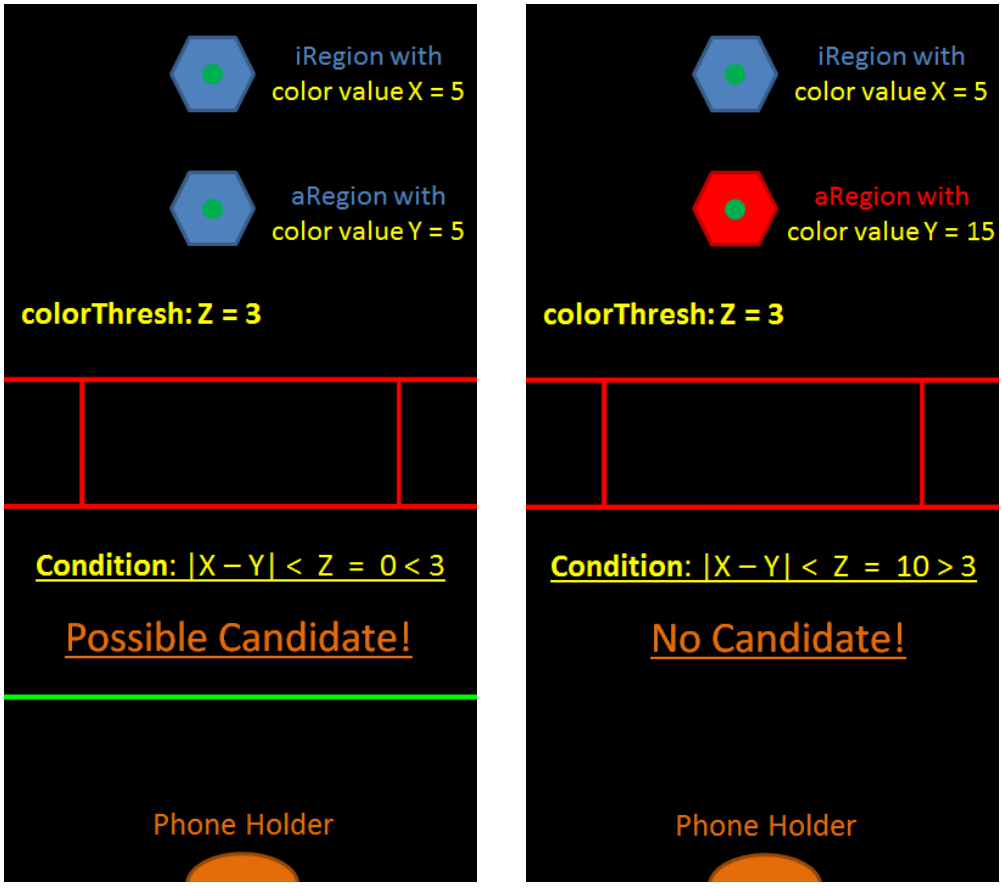


(b) Bright light reflections as a dangerous obstacles

(c) Shadows as dangerous obstacles

(d) Patterns on the ground as dangerous obstacles

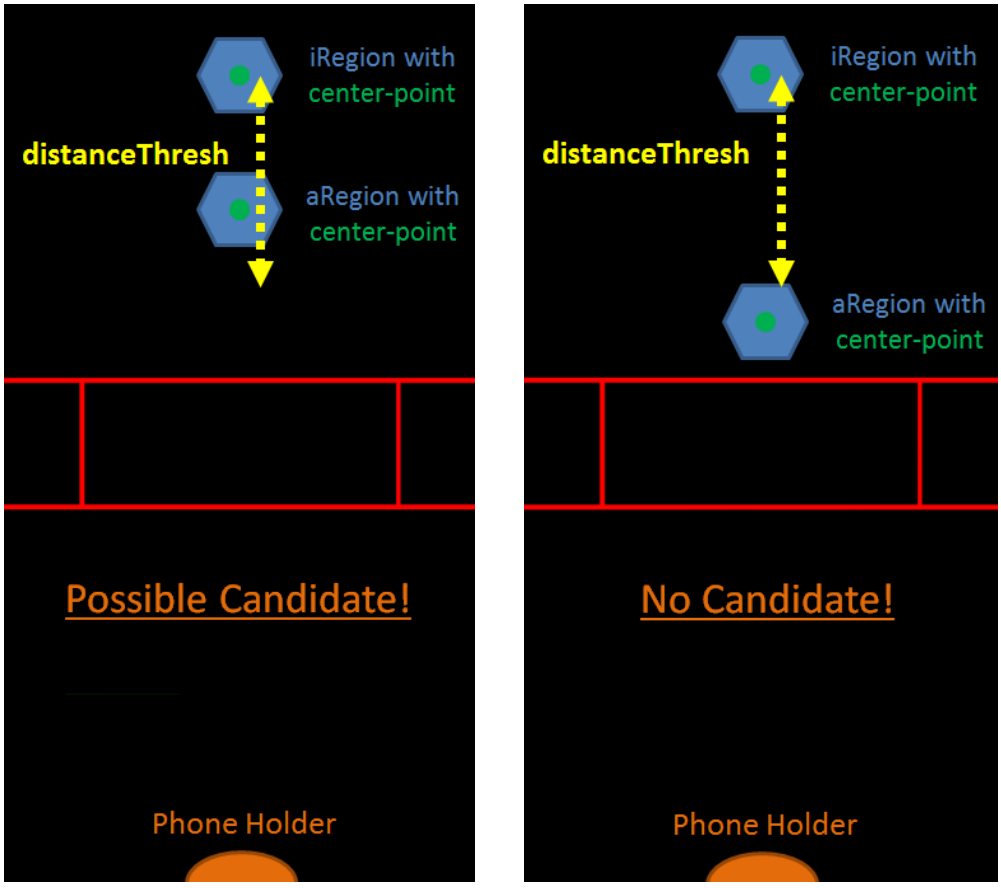
Figure 3.4: Scenarios of possible failures of the app



(a) Color difference is smaller than the threshold - a possible candidate

(b) Color difference is bigger than the threshold - no possible candidate

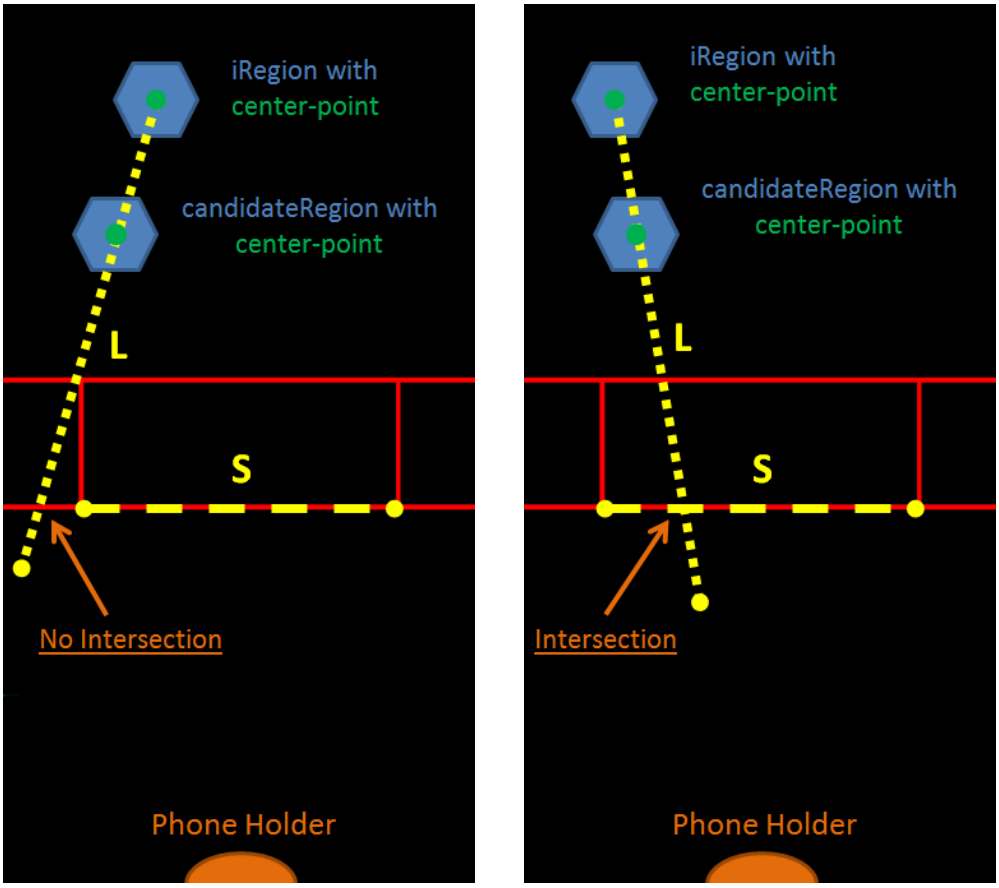
Figure 3.5: An example of determining a possible matching candidate - Color threshold



(a) Distance difference is smaller than the threshold - a possible candidate

(b) Distance difference is bigger than the threshold - no possible candidate

Figure 3.6: An example of determining a possible matching candidate - Distance threshold



(a) No intersection of L and S

(b) L and S intersects

Figure 3.7: Matching regions - Check for intersection

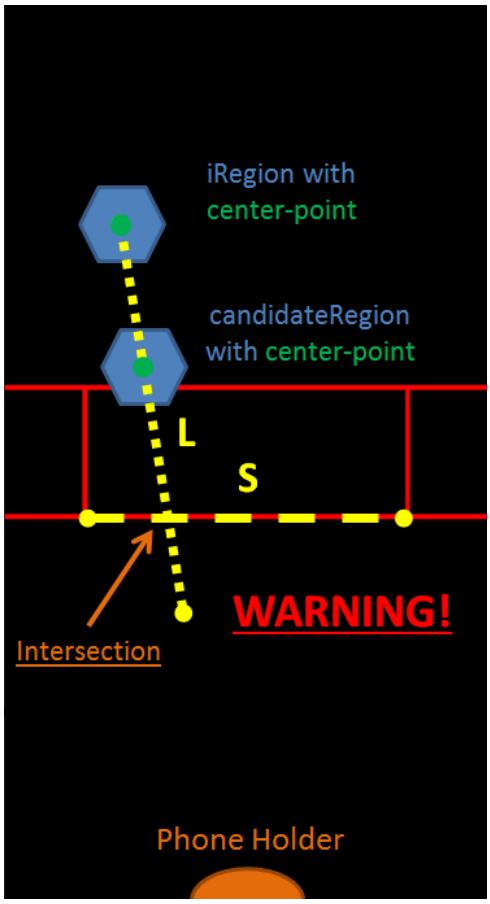


Figure 3.8: iRegion was matched with candidateRegion which is directly moving towards us. Since it has pixels in screen-region 2 a warning is triggered!

Chapter 4

Testing and Results

In this chapter we present all testing results about the different setups for the app. First we state the approach to test our app. We describe the tests and which parameters settings were probed. We conclude the chapter with the results and their interpretations.

4.1 Offline Testing

The big advantage of offline over online testing is the possibility to repeat a test and getting the same results. This allows us to efficiently test all wished parameter settings without changing the test environment.

4.1.1 Approach

We used the phone to record standard videos with the back camera (with a resolution of 480x640 pixels), where a dangerous obstacle is coming towards us. Afterwards we captured an image every 0.2 second and saved it. We tried to come as close as possible to the running app version (online) where we processed around five pictures per second. These pictures were fix and did not change any more. We modified the app to change the input from the camera to the input from these captured fix test frames. The big advantage of this approach was the possibility to reuse the test pictures and testing the different parameter settings while keeping the rest unchanged.

4.1.2 Test Environment

By removing all android specific code it was possible to simulate the functionality of the app on the computer. This allowed us to test the parameter settings much more efficiently (less time needed, more available space etc.) than on the phone itself.

4.1.3 Tests

In total there were 23 test videos recorded. There is a dangerous obstacle moving towards the user at the end of each test. We will refer to this kind of test as "active test". We examined every active test video carefully and wrote down in a file which frame numbers should cause a warning and compared them with the caused warnings of a specific parameter setup. A test was successful if the user got a warning in one of the right frame numbers. Additionally we recorded 6 test videos in which nothing dangerously happens. In other words these tests should not throw a warning at all. We will refer to this kind of test as "passive test". These passive tests were only applied for the final settings. Much more important for us was to pass the active tests. We rather had correct warnings and some more false warnings than only few warnings and a broken nose from a missed warning.
We made value ranges for the important parameters and tested all possible combinations in all active tests (a test suite). These test suites were repeated for different settings (see Section 4.3). They also helped us to find some crucial bugs in the implementation.

4.2 Tested Parameters

In the following sections we explain why a specific parameter was tested.

4.2.1 Gaussian Blur - Gaussian Kernel

The Gaussian kernel size was tested with different values. Larger values caused a stronger smoothing effect and different color regions. For details about the Gaussian Blur we refer to [**?**]. In the following sections we will refer to this threshold as "blurring threshold".

4.2.2 Threshold for Creating Color Regions

As seen in the previous chapter this threshold is crucial for defining color regions. A larger threshold creates fewer color regions which contain more different color values. Precision is given up in favor of fewer false warnings. In the following we will refer to this threshold as "creation threshold".

4.2.3 Threshold for Matching Color Regions - Color based Threshold

Another crucial threshold must be defined for matching color regions. Each such region contains a reference-pixel with a specific color value. It needs to

be determined how much deviation is allowed in the matching step. One might think we could use the same threshold as for creating color regions. But on a closer look these are two separate things. Moreover the tests confirmed that setting the two values equal did not achieve the best results. In the following sections we will refer to this threshold as "matching color threshold".

4.2.4 Threshold for Matching Color Regions - Distance based Threshold

To avoid false matchings between color regions that have a very similar color, but are located far away from each other (see Figure 4.1), we introduced a distance threshold. This threshold could more or less be estimated by comparing how far an obstacle can move from one picture to the next. Nevertheless for fine tuning a few values were tested. In the following sections we will refer to this threshold as "matching distance threshold".

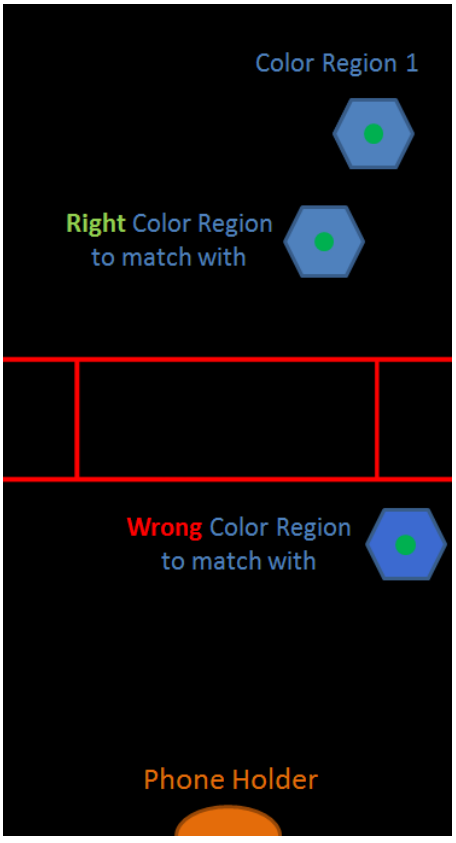


Figure 4.1: Avoid matching the color region 1 with the wrong color region by introducing a Distance Threshold

4.2.5 Lines that Define Screen-Regions

A rough placement of the lines that define screen-regions could have been estimated. We need warnings in a certain distance, forcing us to place the lines somewhere in the middle of the screen. Additionally we need to place segments 1 and 2 (see Figure 4.2), because we don't want to get warnings from obstacles that move past us and not towards us. A few settings were tested.

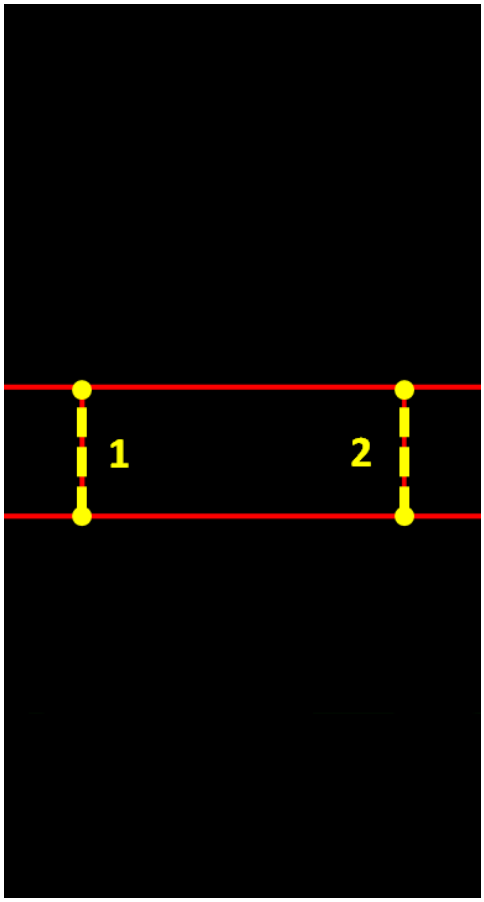


Figure 4.2: Introducing segement 1 and 2 to get rid of warnings of obstacles that are not directly moving towards us

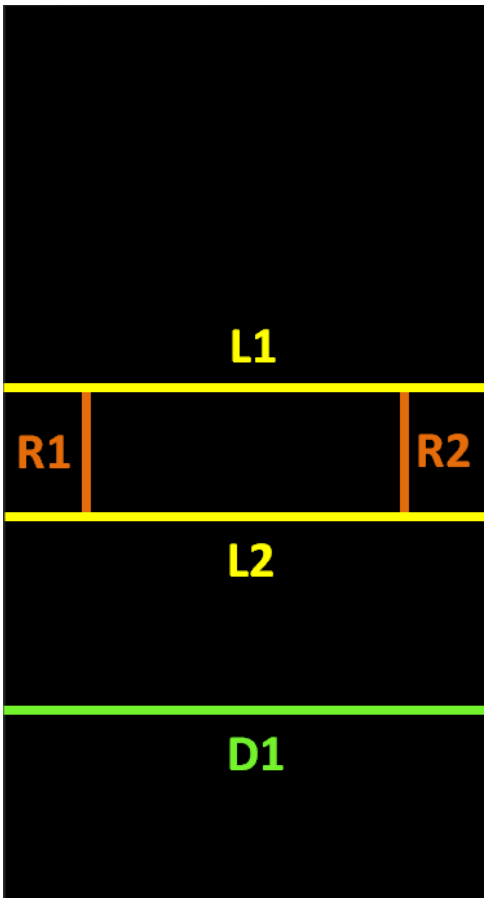


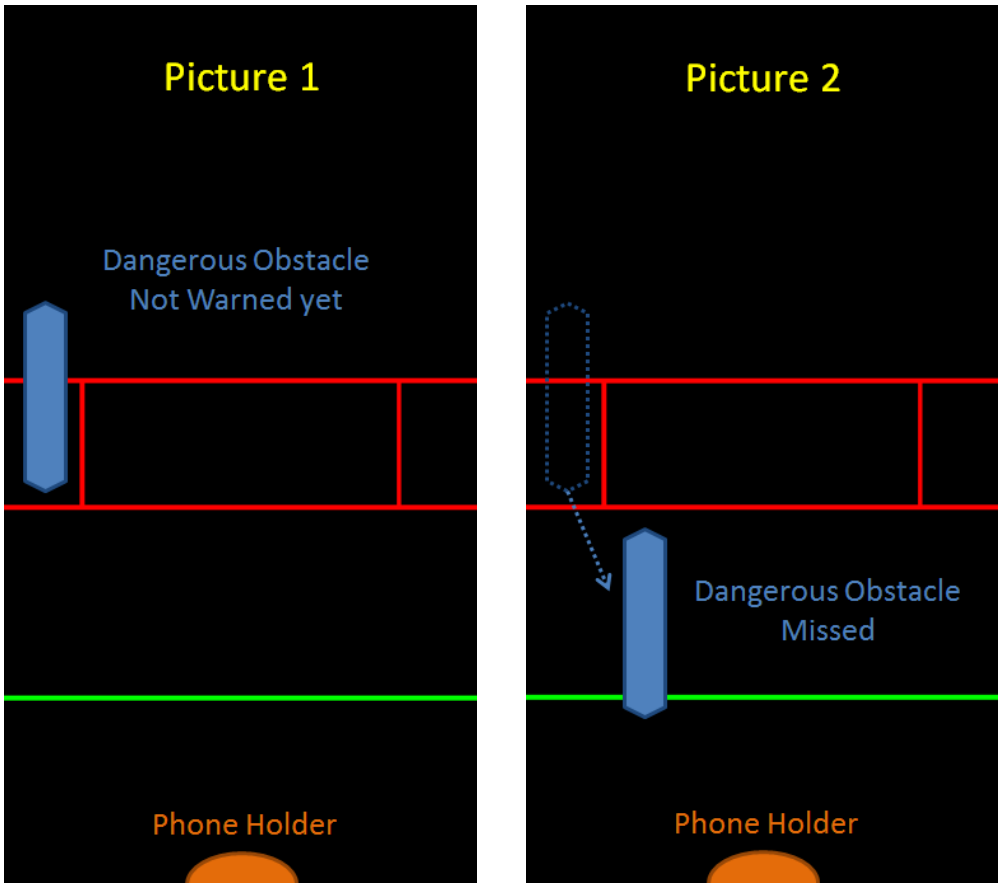
Figure 4.3: Names of the lines used in the app

4.3 Test Suite Setups and Goal

In each test suite setup we tested a wide range of blurring threshold, creation threshold and matching color threshold values. The other parameters were set to a specific value. Our goal was to find setups where we passed all active tests and pick the one with the least amount of warnings (right warnings + false warnings). For the best setting we ran the passive tests as well. For a better overview we present all test suite setups, results and their interpretation together in the next section.

4.4 Results and Discussion

In the representation of the following test results we will give the different lines on the phone screen names to avoid confusion (see Figure 4.3). We additionally introduced Line D1. We tried to discard every color region which contains a pixel below this line. The reason for this was if there is a dangerous obstacle contained in screen-region 1 and/or 3 only, it's nearly impossible that it moves all the way down below line D1 in the next picture, since we have five pictures per second processed and we assumed you walk in a normal speed. This optimization was tested thoroughly with good results without missing crucial warnings. In the first few test settings our optimization achieved better results. We focused on using this approach. In the end we found satisfying settings with the optimization by using line D1. We still wanted to tested again our original approach. Against all expectations we achieved better results. An interesting turnaround after a lot of tests. The results and discussion of specific results are shown below. The ordering is chronological. To the most important setups we provide pictures showing the distribution of the results.



(a) Dangerous obstacle in screen-region 1 and 3 - No warning was triggered yet

(b) Obstacle moved from Picture 1 to Picture 2 and is discarded without triggering a warning - Nearly impossible since it can't move this far having five pictures per second

Figure 4.4: A nearly impossible situation

Abbreviations and Conventions:
H = Screen height
0H = Top border of the screen
1H = Bottom border of the screen
0.5H = Middle of the screen
W = Screen width
0W = Left border of the screen
1W = Right border of the screen
0.5W = Middle of the screen

4.4.1 Small Test Suites

Blurring threshold: 1, 3, 5, 7, 9
Creation threshold: 9, 10, 11, ..., 34
Matching color threshold: 1, 2, 3, ..., 14
Number of tests in one test suite: 1820 (= 5*26*14)

1. Setup

- D1 = 0.9H
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.5H
- L2 = 0.7H
- Matching distance threshold = 7

Results

- Best setting:
  - 21/23 active tests passed, 74 warnings in total
  - 20/23 active tests passed, 60 warnings in total

Discussion

No setting could pass all active tests. We also waited too long for triggering a warning. In the next setup we tried to give warnings earlier by moving L1 and L2. upwards.

2. Setup

- D1 = 0.9H
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.4H
- L2 = 0.6H
- Matching distance threshold = 7

Results

- Best setting:
  - 22/23 active tests passed, 57 warnings in total
  - 21/23 active tests passed, 102 warnings in total

Discussion

One specific setting clearly dominated all other settings. Still we did not achieve our goal to pass all active tests, but it was obviously a step in the right direction.

3. Setup

- D1 = 0.9H
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.4H
- L2 = 0.6H
- Matching distance threshold = 7
- New center-point definition: Sum up all X values [Y values] and divide them by the number of pixel in the color region

Results

- Best setting:
  - 21/23 active tests passed, 103 warnings in total
  - 20/23 active tests passed, 52 warnings in total

Discussion

The idea of a new definition of the center-point did not help us. We did not include this new definition in the following setups.

4. Setup

- D1 = 0.9H
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.45H
- L2 = 0.6H
- Matching distance threshold = 7

Results

- Best setting:
  - 20/23 active tests passed, 60 warnings in total
  - 19/23 active tests passed, 75 warnings in total

Discussion

Reducing the size of screen-region 2 again wasn't a good decision. We therefore let L1 and L2 untouched and focused on the other parameters.

5. Setup

- D1 = 0.9H
- R1 = 1.5/6W
- R2 = 4.5/6W
- L1 = 0.4H
- L2 = 0.6H
- Matching distance threshold = 7

Results

- Best setting:
  - 21/23 active tests passed, 80 warnings in total
  - 20/23 active tests passed, 50 warnings in total

Discussion

We tried to make screen-region 2 more tight by changing R1 and R2. It did not improve our last results. Therefore we kept R1 and R2 in the original position in all following setups.

6. Setup

- D1 = 0.9H
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.4H
- L2 = 0.6H
- Matching distance thresh = 10

Results

- Best setting:
  - 21/23 active tests passed, 42 warnings in total
  - 20/23 active tests passed, 33 warnings in total

Discussion

In this setup we tried to be more tolerant by setting the matching distance threshold to 10. It was possible to decrease the amount of warnings, but against our hope it could not improve our last setups regarding the number of passed active tests.

4.4.2 Large Test Suites

By keeping the creation threshold and matching color threshold values unchanged we could not find one single setting which passed every active test. Therefore we decided to make the interval of these thresholds larger

Blurring threshold: 1, 3, 5, 7, 9
Creation threshold: 1, 2, 3, ..., 50
Matching color threshold: 1, 2, 3, ..., 30
Number of tests in one test suite: 7500 (= 5*50*30)

1. Setup

- D1 = 0.9H
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.4H
- L2 = 0.6H
- Matching distance threshold = 10

Results

- Best setting:
  - 23/23 active tests passed, 71 warnings in total
  - 22/23 active tests passed, 45 warnings in total

Discussion

We finally reached our goal. Some parameter settings were able to pass all active tests. The aim was now to reduce the number of warnings in total while still passing all active tests. To mention is the really low number of false warnings of the best setting which passed 22 of 23 active tests. This showed that we still have potential to improve the current settings.

2. Setup

- D1 = 0.9H
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.4H
- L2 = 0.6H
- Matching distance threshold = 8

Results

- Best setting:
  - 23/23 active tests passed, 65 warnings in total
  - 22/23 active tests passed, 71 warnings in total

Discussion

We could eliminate some false warnings with a smaller matching range, but the small number of warnings for the best setting which passed 22 tests was vanished.

3. Setup

- D1 = 0.9H
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.4H
- L2 = 0.6H
- Matching distance threshold = 8
- New blurring values: 11, 13, 15, 17, 19

Results

- Best setting:
  - 23/23 active tests passed, 101 warnings in total
  - 22/23 active tests passed, 95 warnings in total

Discussion

Blurring the images more did not bring the hoped improvement. The number of false warnings increased a lot. This results showed us to keep the blurring kernel size small. In the following setups we maintained the original blurring threshold values.

4. Setup

- D1 = 0.8H
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.4H
- L2 = 0.6H
- Matching distance thresh = 8

Results

- Best setting:
  - 23/23 active tests passed, 65 warnings in total
  - 22/23 active tests passed, 69 warnings in total

Discussion

The best test wasn't affected by moving D1 upwards. But some parameter settings triggered less false warnings. Thus we kept D1 on 0.8H. Moreover this optimizes the image processing.

5. Setup

- D1 = 0.8H
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.3H
- L2 = 0.5H
- Matching distance threshold = 8

Results

- Visualized in Figure 4.6 and 4.7
- Best setting:
  - 23/23 active tests passed, 51 warnings in total
    - Blurring threshold = 9
    - Creation threshold = 42
    - Matching color threshold = 23
  - 22/23 active tests passed, 40 warnings in total
    - Blurring threshold = 1
    - Creation threshold = 42
    - Matching color threshold = 13

Discussion

We decided to move screen-region 2 upwards again. This setup had very convincing results. We closely came down to one false warning on average. An additional advantage is that warnings will triggered a little bit earlier, giving the user more time to react.

6. Setup

- No D1!
- R1 = 1/6W
- R2 = 5/6W
- L1 = 0.3H
- L2 = 0.5H
- Matching distance threshold = 8

Results

- Visualized in Figure 4.8 and 4.9
- Best setting:
  - 23/23 active tests passed, 47 warnings in total
    - Blurring threshold = 1
    - Creation threshold = 42
    - Matching color threshold = 26
    - 1/6 passive tests passed with a total of 6 warnings with this specific setting
  - 22/23 active tests passed, 41 warnings in total
    - Blurring threshold = 1
    - Creation threshold = 42
    - Matching color threshold = 13

Discussion

We almost achieved 1 false warning per active test and no dangerous obstacle was missed. The passive tests also triggered only 1 false warning on average. These results were very satisfying. Therefore we decided to choose this setting in the final app.

4.5 Summary of Testing

During many hours of testing we slowly moved towards a good setting. The chosen one passed every active test and triggered only few false warnings. Surprisingly this was a setup from which we stepped away in the beginning of the whole testing phase. As seen in the results a higher creation threshold was needed. This traded just enough precision to still pass every test but remove a lot of false warnings. Surprisingly the blurring value 1 appeared to be the best. This means that only little preprocessing is required. In the previous project a blurring value of 7 was used instead. In Figure 4.5 we again summarize all results and highlight the optimal setup.

Small Test Suites

	D1	R1	R2	L1	L2	Matching distance threshold	Special settings	Active tests passed	Number of warnings in total
1. Setup	0.9H	1/6W	5/6W	0.5H	0.7H	7	--	21/23	74
2. Setup	0.9H	1/6W	5/6W	0.4H	0.6H	7	--	22/23	57
3. Setup	0.9H	1/6W	5/6W	0.4H	0.6H	7	New center-point definition	21/23	103
4. Setup	0.9H	1/6W	5/6W	0.45H	0.6H	7	--	20/23	60
5. Setup	0.9H	1.5/6W	4.5/6W	0.4H	0.6H	7	--	21/23	80
6. Setup	0.9H	1/6W	5/6W	0.4H	0.6H	10	--	21/23	42

Standard Settings for Small Test Suites	
Blurring threshold	1, 3, 5, 7, 9
Creation threshold	9, 10, 11, ..., 34
Matching color threshold	1, 2, 3, ..., 14

Large Test Suites

	D1	R1	R2	L1	L2	Matching distance threshold	Special settings	Active tests passed	Number of warnings in total
1. Setup	0.9H	1/6W	5/6W	0.4H	0.6H	10	--	23/23	71
2. Setup	0.9H	1/6W	5/6W	0.4H	0.6H	8	--	23/23	65
3. Setup	0.9H	1/6W	5/6W	0.4H	0.6H	8	Blurring threshold: 11, 13, 15, 17, 19	23/23	101
4. Setup	0.8H	1/6W	5/6W	0.4H	0.6H	8	--	23/23	65
5. Setup	0.8H	1/6W	5/6W	0.3H	0.5H	8	--	23/23	51
6. Setup	--	1/6W	5/6W	0.3H	0.5H	8	No D1	23/23	47

Standard Settings for Large Test Suites	
Blurring threshold	1, 3, 5, 7, 9
Creation threshold	1, 2, 3, ..., 50
Matching color threshold	1, 2, 3, ..., 30

Figure 4.5: Table showing all test results. The best setup is highlighted in green

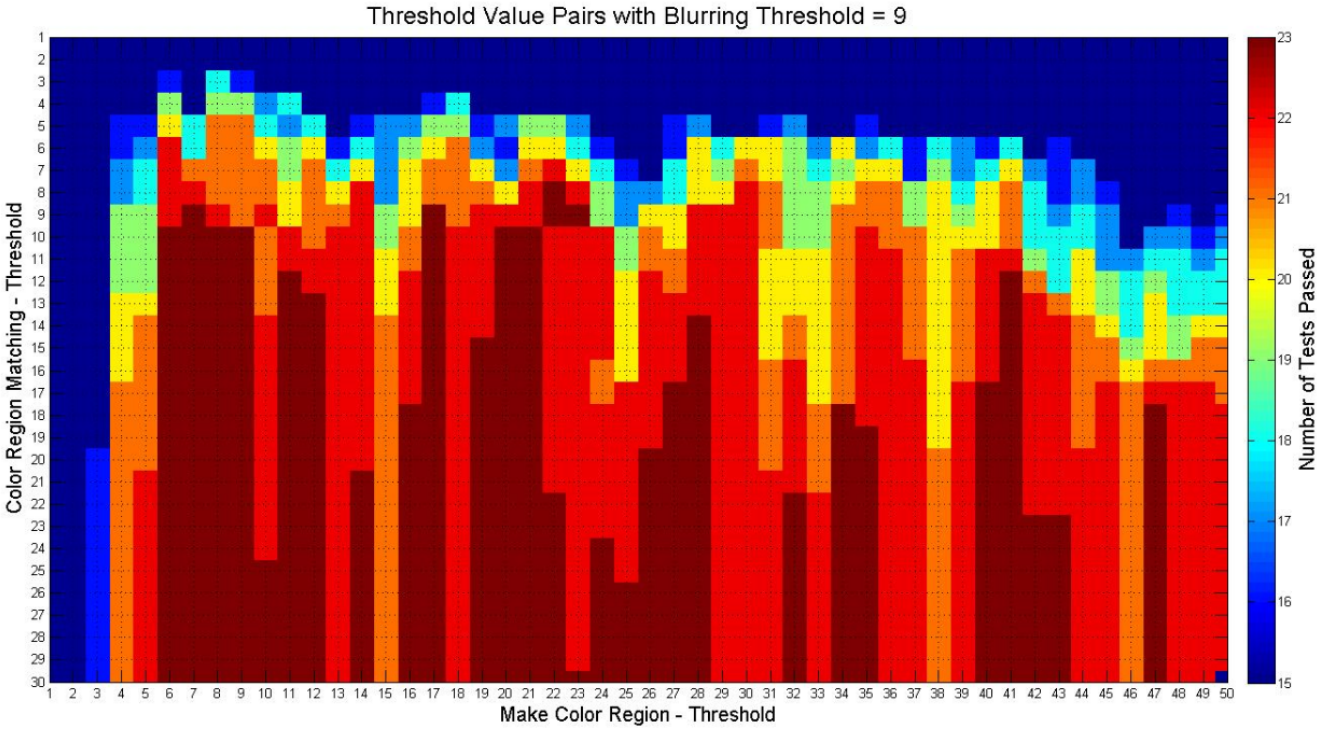


Figure 4.6: Distribution of the number of tests which passed. All tests which passed 15 or less tests have the same color

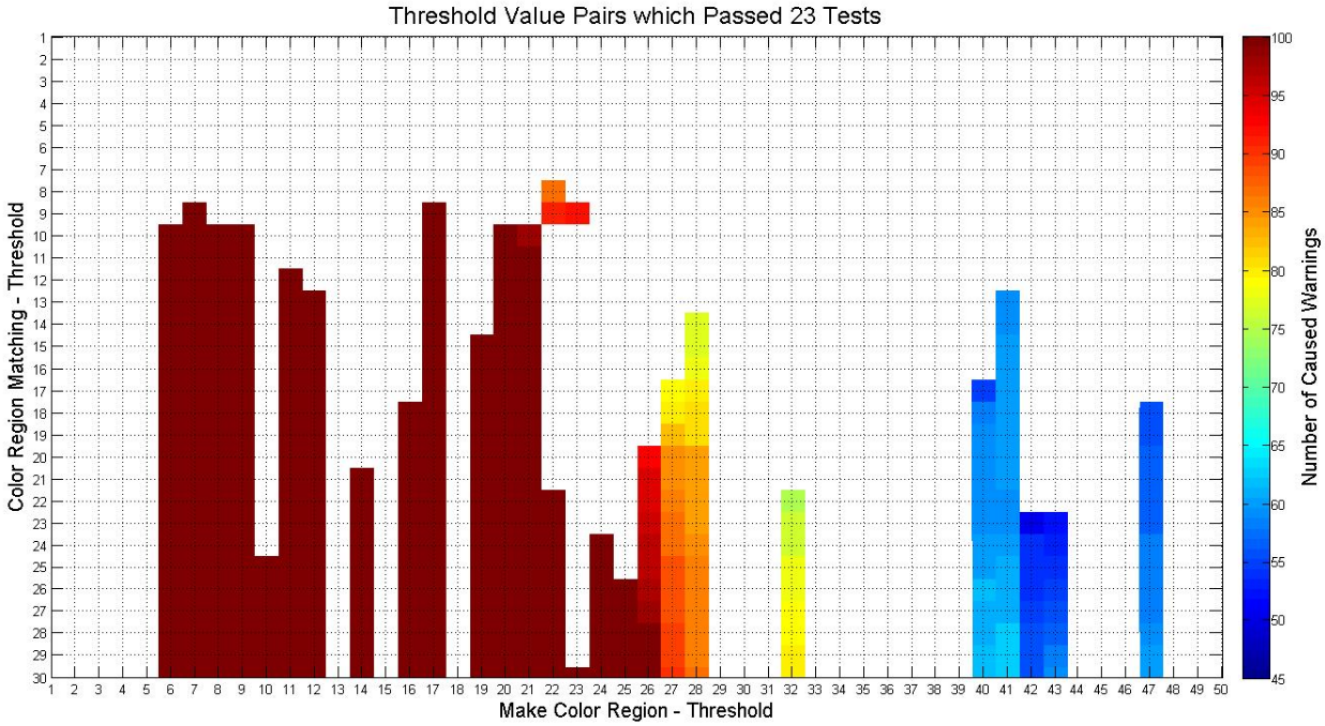


Figure 4.7: Distribution of the number of warnings of all settings which passed all tests. We see that the best test has a creation threshold of 42 and a Matching color threshold of 23

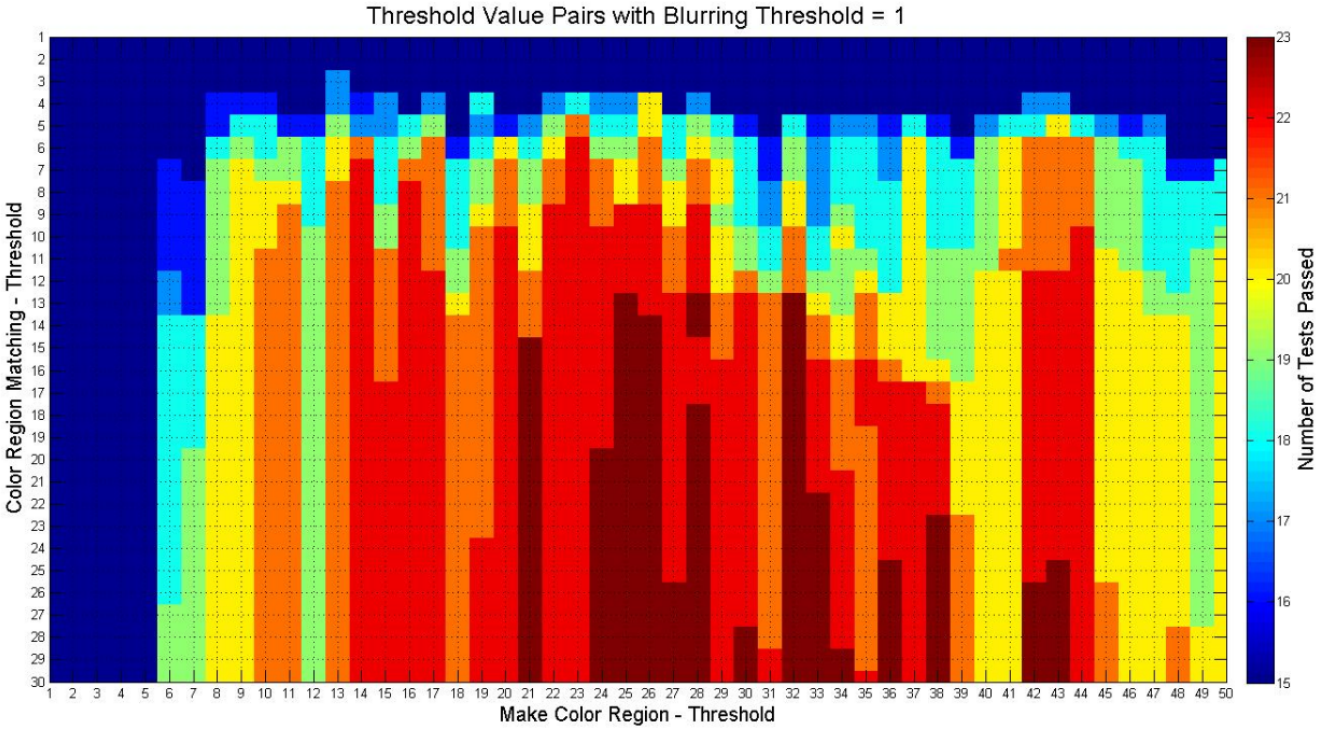


Figure 4.8: Distribution of the number of tests which passed. All tests which passed 15 or less tests have the same color

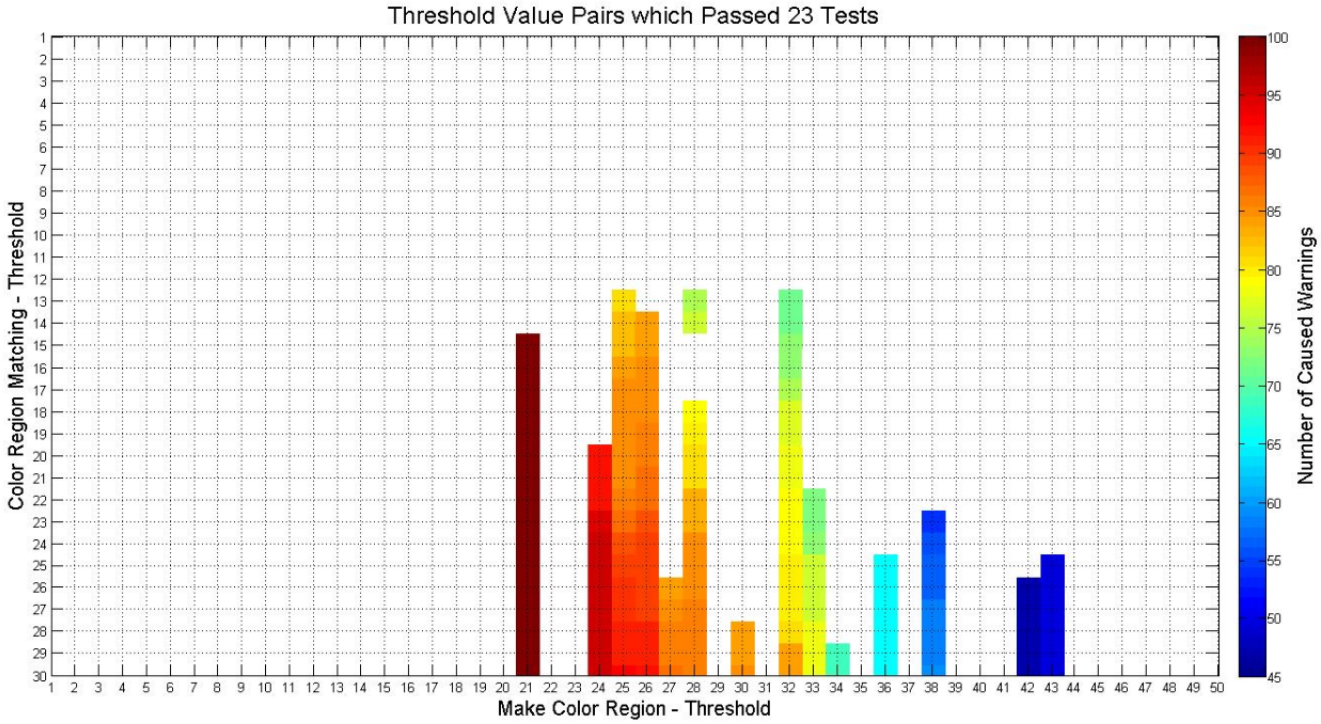


Figure 4.9: Distribution of the number of warnings of all settings which passed all tests. We see that the best test has a creation threshold of 42 and a Matching color threshold of 26

Chapter 5

Conclusion

The app combines simple ideas to efficiently warn the user of dangerous obstacles which are moving directly towards him while he is walking and paying no attention to the way in front of him. Dividing a captured picture from the camera into color regions allowed us to define probably dangerous obstacles. In a matching step we tried to link obstacles from one picture to the next. Considering their movement we could predict whether a dangerous obstacle was going to hit the user. Many different parameters needed to be set in such a way that the user gets only few false warnings but in case of a dangerous obstacle he is warned early enough. Thoroughly testing was necessary and delivered satisfying results.

5.1 Some Differences to the previous Project

As mentioned in the introduction a similar project was done before. Some ideas are related to the ones used in the previous project. Nevertheless there are a few differences to state. In this project we did strict offline testing to find good settings whereas in the previous project only online testing was performed. Here we downscaled the images less to work with much more pixels than in the previous project. The aim was to gain more precision. As stated in Section 2.1 we also directed the phone more to the ground, which felt more naturally.

5.2 Future Work

Despite the good results there are still a lot of settings to be tested and things to optimize in the future. The current approach still has some limitations as seen in Section 3.3. Handling shadows and bright reflections will reduce false warnings. An extension to separate the background from the dangerous obstacles will help facing this problem.

The current version of the app needs to be running in foreground. It doesn't work any more once it is minimized. Supporting a service embedding of the app makes it much more usable in practice. The user receives warnings while the app is running in the background.

A whole new approach provides phones with integrated sensors to do 3D-measurements [**?**]. This simplifies the whole process and makes it more reliable. In combination with the current app it opens up a lot of new possibilites.

Bibliography

[1] : Phone Accidents. http://www.buffalo.edu/news/releases/2014/02/022.html Accessed on 17 Sept 2014.

[2] : Holding the phone. http://relationshipsadvice.selfemployed123.netdna-cdn.com/wp-content/uploads/2014/02/How-to-text-a-girl-you-like.jpg Accessed on 17 Sept 2014.

[3] : Portrait position. http://dienthoaidoc.com/wp-content/uploads/2014/05/samsung-galaxy-s3-anh-3.jpg Accessed on 17 Sept 2014.

[4] : Gaussian Blur. http://docs.opencv.org/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html Accessed on 17 Sept 2014.

[5] : Google Phone. https://www.google.com/atap/projecttango/#project Accessed on 17 Sept 2014.