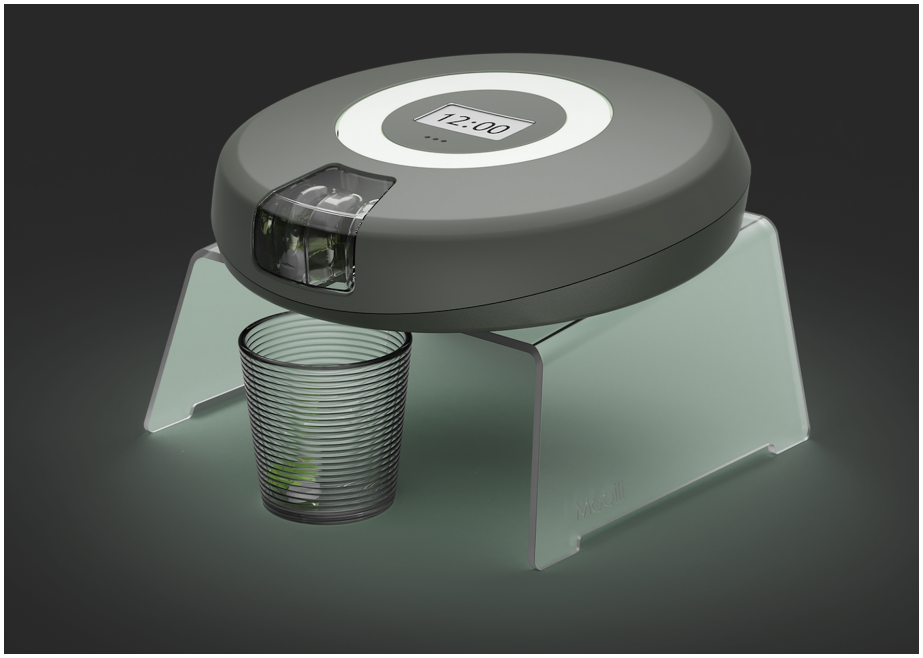
Building an accuracy calculating algorithm for the Mobili medication dispenser

Mobili by Medthings

Synne Mo Sandnes



1 Introduction

Mobili is a mobile medicine dispenser that alerts you when it is time to take your medication, and dispenses the correct dosage. Additionally, it verifies that the medication is delivered. By providing secure and faultless medication, the dispenser aims to decrease medication errors.

We have developed an algorithm to verify that the dispenser actually dispenses the correct medications. The algorithm can be used to determine how accurate the medication dispensing is before the dispenser is administered to real patients. Additionally, it enables us to run thousands of dispensations without without causing risks to patients. This enables us to evaluate the accuracy of the dispenser much quicker than with traditional methods. We have developed a couple of algorithms to make sure the dispenser reaches the correct chamber, and also explored the possibility of developing an algorithm to verify that the slot is fully open.

We will be finding a percentage of error for the medicine dispencing of the Mobili dispenser. This report will contain our tries and errors, and in the end we will present our results.

2 Theory

2.1 Accuracy Measures for Machine Learning

True positive rate

$$\frac{TP}{P} \tag{1}$$

True negative rate

$$\frac{TN}{N} \tag{2}$$

Precision

$$\frac{TP}{TP+FP} \tag{3}$$

False discovery rate

$$\frac{FP}{FP+TP} \tag{4}$$

False omission rate is the proportion of the individuals with a negative test result for which the true condition is positive. The false omission rate is found with the following formula

$$\frac{FN}{TN+FN} \tag{5}$$

Accuracy

$$\frac{TP+TN}{P+N} \tag{6}$$

Fβ

$$F\beta = (1+\beta^2)TP/(1+\beta^2)TP + FP + \beta^2 * FN) \tag{7}$$

2.2 Thresholding

Thresholding is a method of segmenting images by transforming them into binary images. If the image intensity of a pixel is less than a fixed value called the threshold, the pixel will be replaced by a black pixel. If the intensity is greater than the threshold, the pixel will be replaced by a white pixel.

3 Structure and function

3.1 Structure

The plastic cassette consists of three parts. The cassette frame is located at the bottom, and functions to hold the cassette in place. The chamber part of the cassette is where the medicine is added, and the hatch opens and closes each chamber.

Figur 1: The plastic cassette with numbers attached.

3.2 Code ring

The code ring is built up by black and white areas. The short black areas are 4 pixels, and the long is 7 pixels. When spinning, the black and white areas are counted by moving 1 step at a time. There is a total of 877 steps on each medicine chamber. In other words, each black and each white area requires many counts. The dispenser stops rotating when it detects the black area, as this marks the beginning of a new chamber.

3.3 Function

The chamber part of the cassette rotates counterclockwise. Clockwise rotation is not currently implemented, but may be beneficial at a later date. An app is used to create the medicine plan: app.medthings.no

4 Recognizing the chambers

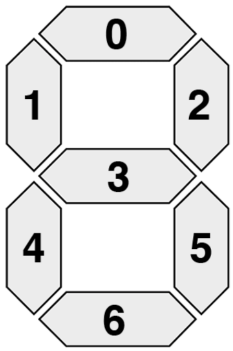
4.1 Thresholding

We begin by assigning and attaching digital numbers to every chamber, shown in figure 1, before creating a Python algorithm to identify them.

We first need to locate the number we want to recognize. This can be done using edge detection since there is enough contrast between the plastic and square around the number. We will be taping over the edges of the gap, in order to show only one chamber instead of three at a time.
With an input edge map we can find contours and look for outlines with a square shape, which can then be extracted through a perspective transform. Thresholding and morphological operations will be used to extract the digits from the white/black background.
Hopefully, we have extracted the digit region, so that we are able to recognize the actual digits. We will be dividing the region into 7 segments, and thereafter apply pixel count to the thresholded image to determine if a given segment is “on” or “off”. This is explained in figure 2. Code is found in *recognising_numbers.py*

MISTAKES AND COMPLICATIONS

The code struggled to differentiate the numbers 0 and 8. Originally, the segments seen in figure 2 would be marked as "on"if the total number of non-zero pixels was greater than 50% of the area. We tried to change this to other values instead, hoping it would



Figur 2: Shows how each digit can be represented by a seven-segment component, depending on whether each segment is turned "on"or "off".

help seperate the two numbers.

The problem may have been that the 0's were too bright in the middle, making the algorithm count it as the line in area 3 from figure 2. We experimented with different values for "if total / float(area) $>$ 0.5:"in an attempt to separate 0 and 8. However, it did not work.

As this method did not give satisfying results, we will be trying another method.

4.2 Using QR-codes

Since the number recognition did not produce satisfactory results, we opted to use QR-codes instead. The 27 separate chambers were represented by codes that we created and attached to the plastic cassette. This is shown in figure 3. In order to differentiate between the chambers, we developed a Python algorithm that extracted the numbers from the QR-codes. The code analyzes each chamber using the computer's webcam and determines whether it is the same as specified in the medication plan. Furthermore, it prints the number of times it lands on the correct chamber, and the number of times it lands on the wrong one.

Before the medication is administered to real patients for testing, we can use our algorithm to determine how accurate the medication dispensing is. If the dispenser turns out to have significant flaws, we will be aware of them and have the chance to correct them before they cause a serious accident. The algorithm can be run on all plastic cassettes before they are sent out to patients, to ensure they don't have any flaws.

4.2.1 Testing

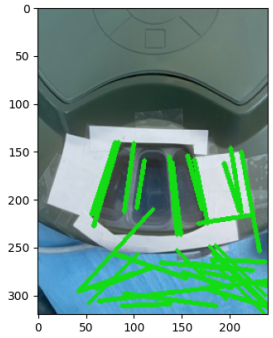
After creating several theoretical medication plans, we ran the algorithm to check if the medicine was dispensed as expected. For the first few test runs, we started off easy by letting the dispenser move in consecutive order. As this did not seem to be a problem for the dispenser, we switched up the order of the chambers to be dispensed, to see if the dispenser struggled. The code that ended up working is given in read_barcode_final.py.

4.2.2 Results

Our chamber detection algorithm was run multiple times with the same result every time. The Mobili dispenser reaches the correct chamber without exception, resulting in an accuracy rate of 100%. As for our algorithm, the chambers detected by the algorithm were correct in all tests.

However, we recognize that we should have let the algorithm run for a long period of time to increase the credibility. We spent two days testing the algorithm, but the results might have been different if we had time to run thousands of dispensations.

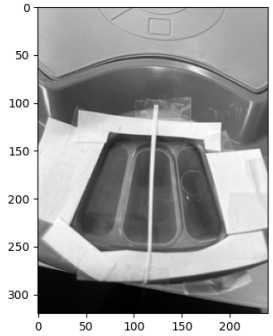
Figur 3: The dispenser with the QR-codes attached.



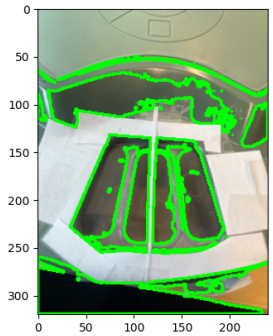
Figur 4: The edge detection method from sjekkluke4.py did not give satisfying results, as the figure shows.

5 Checking if the slot is fully open

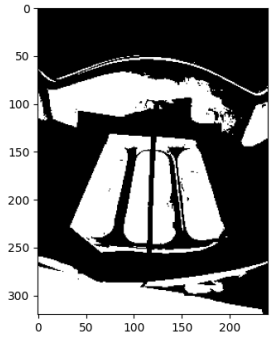
We were able to differentiate the slot from the rest of the cassette through edge detection. After making six different python algorithms, we were still not able to get satisfying results. We tried splitting the slot in half with a line of paper, hoping to implement a test to see if the line is in the middle of the slot. If the line was in the middle, it would mean that the slot was completely open, leaving no space for the pills to get stuck. Figures 4, 5, 6, 7 and 8 shows the images returned by some of the algorithms.



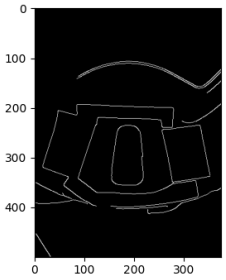
Figur 5: Black and white of the image used to investigate whether the line is the middle of the s



Figur 6: The figure shows the edges detected from the image with the line in the middle of the slot.



Figur 7: The inverse-binary thresholding of the image.



Figur 8: The figure shows the edge contours of the bottom side of the dispenser, where the slot is shown in the middle. This is the output when running luke.py.

5.1 Acknowledgements

I would like to express my special appreciation and thanks to my two advisors for their guidance and advice in this project.

Anne Gerd Granås. Professor, Section for Pharmaceutics and Social Pharmacy, Department of Pharmacy Visiting professor (20%), Norwegian Centre for e-Health Research

Bjørn Toreid. Founder & CEO Of MedThings. Industrial designer. Many years of experience as product manager within remote care.