Wrocław University of Science and Technology
Faculty of Electronics

Camera based proximity sensor

Author:
Marcin Panek

Supervisor:
Witold Paluszyński, PhD

Abstract

This paper describes camera based proximity sensor project. The scope of the project was to implement algorithm to detect direction of movement of objects in front of the camera. Tools used for this purpose are Matlab 2017b and USB camera. Project was treated as a research for further work on implementation on STM32F7 microcontroller. Potential application is a sensor for robot that will let it avoid collision with obstacles - if object is moving in direction of robot it has to take up some action.

January 23, 2018

1 Introduction

This project was a first step to implement camera based proximity sensor on STM32F7 microcontroller. The idea is not to produce a precise sensor that will be able to measure actual distance to a chosen object but rather to come up with a device detecting if an obstacle is moving in direction of camera. This would inform system of mobile robot about potential collision. In terms of this project the algorithm has been implemented in Matlab environment. USB camera has been used as an input device. In next step code will be ported to STM32 device and STM32F4DIS-CAM module will be used as an input device.

The main tasks for the project were:

- developing software for color based object detection,
- developing software for object tracking,
- developing algorithm detecting if the detected object is moving towards camera or in the opposite direction
- making demo application working on image received from camera in live mode,
- performing tests of developed demo - objects moving towards camera simulating moving obstacles that autonomous vehicle potentially can collide with.

Tools used in project are:

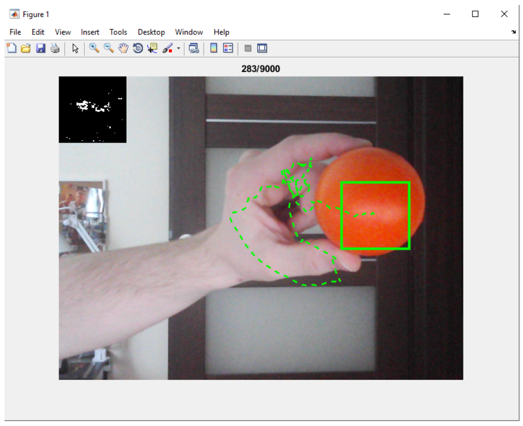
- *Matlab 2017b* - 30 day trial version has been used, it can be downloaded by anyone for the evaluation purposes,
- *MATLAB Support Package for USB Webcams* [4] - *Matlab* plug-in used for image acquisition
- *Matlab Guide* [2] - *Matlab* used to develop graphical user interfaces,
- *Matlab Image Processing Toolbox* [3] - set of software tools helpful in image processing,
- *LG AN-VC500* - USB camera.

Software tools used in this project are proprietary ones that are available for students and employees of Wrocław University of Technology but one could as well use open source alternative - *Octave* [1] and its corresponding tools.

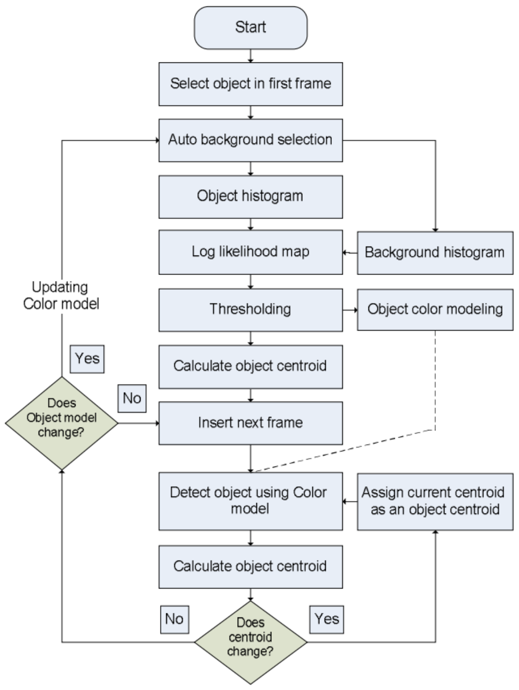
2 Algorithm research

The first part of the project was to decide on the algorithm that should be used. Main requirement for the algorithm was low computational complexity, due to the fact that is should be portable for STM32 microcontroller.

First idea was to use Adaptive Object Colour Modelling method described in [5]. Its block diagram is presented on figure 1(b). In this method, likelihood map is built based on object and background histogram. However the size of the area where the object is expected is manually chosen by user and constant therefore, when object moves towards camera it becomes bigger and gets out of the fixed sub-frame. Because of that, it is impossible to determine size of obstacle in particular frame and thus it becomes impossible to distinguish direction



(a) Screen from implementation


Start
Select object in first frame
Auto background selection
Object histogram
Log likelihood map
Background histogram
Thresholding
Object color modeling
Updating Color model
Yes
Calculate object centroid
No
Does Object model change?
Insert next frame
Detect object using Color model
Assign current centroid as an object centroid
Calculate object centroid
No
Yes
Does centroid change?

(b) Block diagram of the algorithm [5]

Figure 1: Object Tracking Using Adaptive Object Color Modeling

of movement. Figure 1(a) presents screen shot from demo application with Adaptive Object Colour Modelling algorithm implemented. The problem described above can be seen on the picture - green frame that was initially outside the object is now to small fro object to fit inside.

Modifying the approach to be distance tolerant is not an easy task because it would require recalculating likelihood for various size of the area. Since, in our application we will have to compare size of the object on the consecutive frames it is curial to be able to distinguish its size accurately in current frame. Due to that fact it would be necessary to calculate likelihood several times for each frame for a few sizes of outer box and thus the computation time would become high.

Due to the reason described above it has been decided to use another approach. What is taken into account is the difference between image transformed into grey scale and the image in layer corresponding to one of the base colors: red, green and blue. General idea of the algorithm is presented on block diagram on the figure 2.

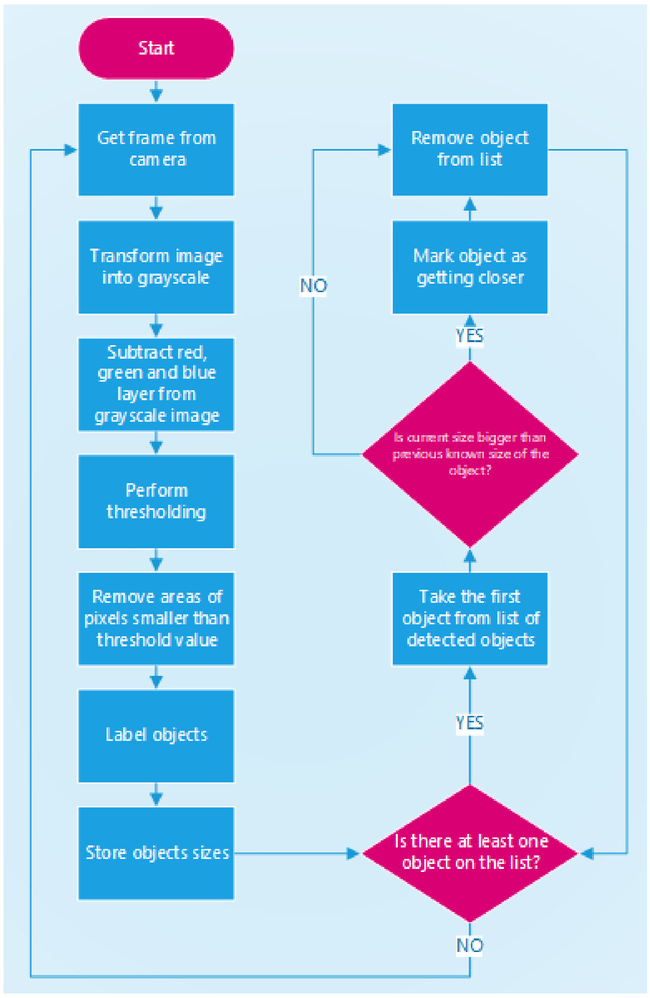

Start
Get frame from camera
Transform image into grayscale
Subtract red, green and blue layer from grayscale image
Perform thresholding
Remove areas of pixels smaller than threshold value
Label objects
Store objects sizes
Is there at least one object on the list?
YES
NO
Take the first object from list of detected objects
Is current size bigger than previous known size of the object?
YES
NO
Mark object as getting closer
Remove object from list

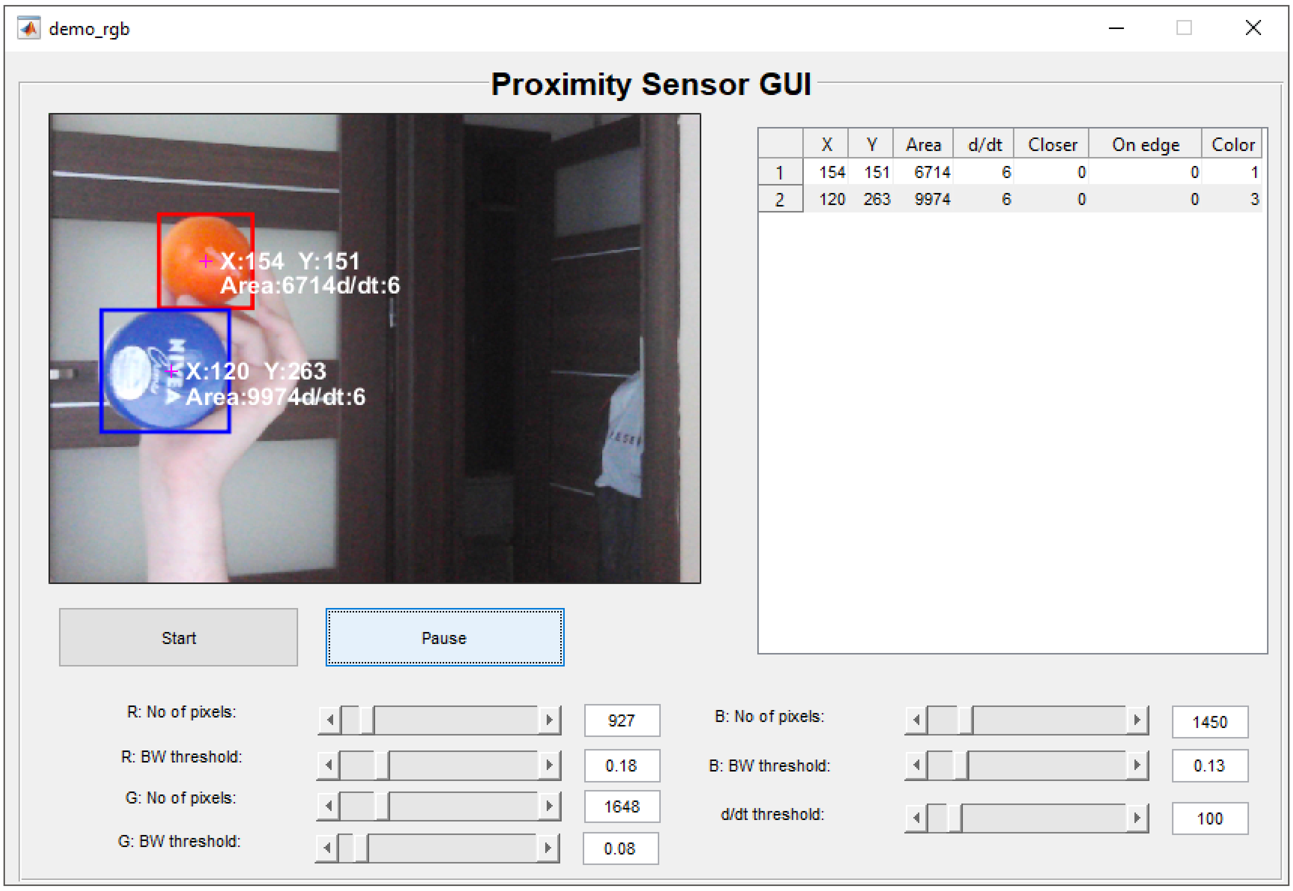
Figure 2: Block diagram of the algorithm used in project

At the beginning of the loop an image is captured from camera. Than its copy is transformed into grey scale (weighted sum of R, G and B components is used according to following formula: $0.2989 * R + 0.5870 * G + 0.1140 * B$).

Next, one of the color components is subtracted from the gray scale image. This procedure is repeated for each color layer (R, G and B). Then, in order to make objects clear in the frame thresholding operation is applied, threshold value can be chosen in graphical application for test purposes. Finally, to get rid of noise and some small object from the background, all closed areas that contain less than given value of pixels are removed from the frame. In this case, threshold value can be adjusted in user interface as well. Last operation is labelling of the objects and storing its size in order to be able to compare it between consecutive frames. This comparison lets to find out weather object is getting closer to camera, is standing still or is moving away. Results of above operations can be observed on the figure 4.

3 Description of implementation

Apart from the algorithm itself, user graphical application has been developed in order to be able to easily perform various test for various sets of parameters. It has been done using *Matlab GUIDE* tool for graphical interfaces.



	X	Y	Area	d/dt	Closer	On edge	Color
1	154	151	6714	6	0	0	1
2	120	263	9974	6	0	0	3

R: No of pixels:	927	B: No of pixels:	1450
R: BW threshold:	0.18	B: BW threshold:	0.13
G: No of pixels:	1648	d/dt threshold:	100
G: BW threshold:	0.08		

Figure 3: Graphical user interface developed in terms of project

On the left hand side there is a live preview of the image acquired from camera together with the boundaries and properties for objects detected by algorithm. On the right hand side there is a table with all the objects detected and their properties such as location in the frame, size in pixels, difference between size on the current and previous frame, information about weather object getting closer and if is on the edge of the frame, color in which object has been detected (1 for red, 2 for green and 3 for blue). On the bottom of the window there are sliders to adjust parameters described in previous chapter and additionally one parameter setting the required minimal difference between consecutive sizes of obstacle in order to detect it as moving.

4 Results of performed test

Figure 4 presents several results of test of object detection. First one is example of orange ball being detected using red layer. The ball is overexposed in the image thus it shape is not correct in the final outcome. On the second example, lighting has been changed and ball is not overexposed thanks to what it is fully visible in the final outcome of processing. However, since we are able to put boundary box correctly around obstacle in both cases, performance of the direction detection is similar. The example from figure 4(c) presents experiment with green object and green layer. Interesting fact is that in case of this color, threshold for binarization has to be significantly lower than in case of red and blue color. Here, we can also see that fact that object is not purely green does not influence detection performance. Finally, two last examples present experiments for blue color. The last one, shows object that has blue parts on both ends, yet is black inside. In this case algorithm will detect two blue object moving in camera's direction. However this should not be a problem in case when camera is only for alarming robot's system about potential danger.

5 Summary

All of the goals of the project has been achieved successfully. It turned out that number of features influencing quality of performance of developed algorithm is significant. However it is possible to arrange test conditions in which algorithm performs well. This will let implement described functionality on microcontroller and compare achieved performance to the computer implementation.

References

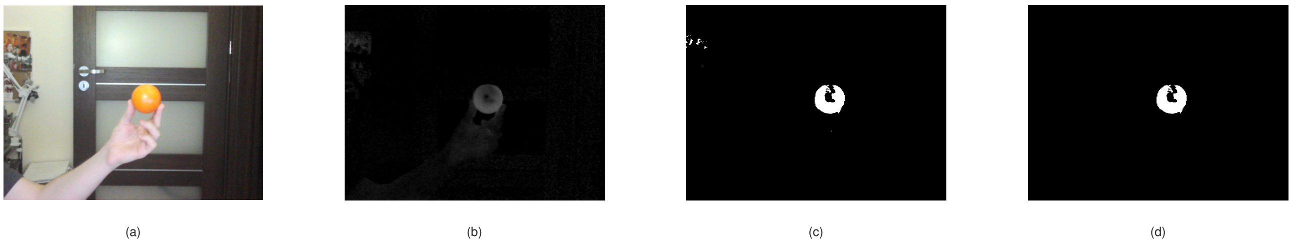
[1] https://wiki.octave.org. `https://www.mathworks.com/matlabcentral/fileexchange/45182-matlab-support-package-for-usb-webcams`.

[2] Matlab guide for gui. `https://www.mathworks.com/discovery/matlab-gui.html`.

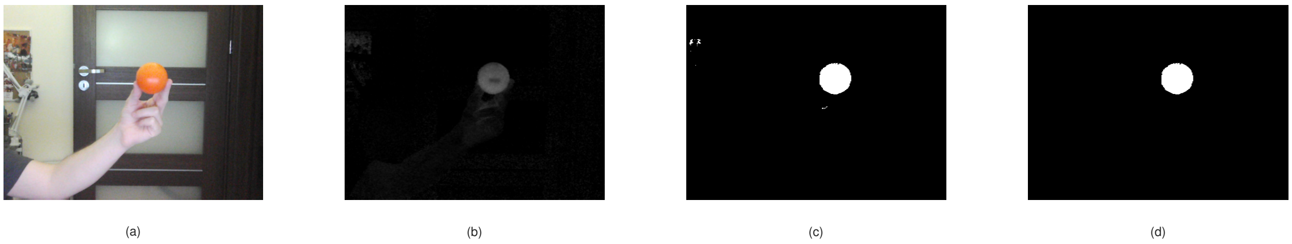
[3] Matlab image processing toolbox. `https://www.mathworks.com/products/image.html`.

[4] Matlab webcam support package. `https://www.mathworks.com/matlabcentral/fileexchange/45182-matlab-support-package-for-usb-webcams`.

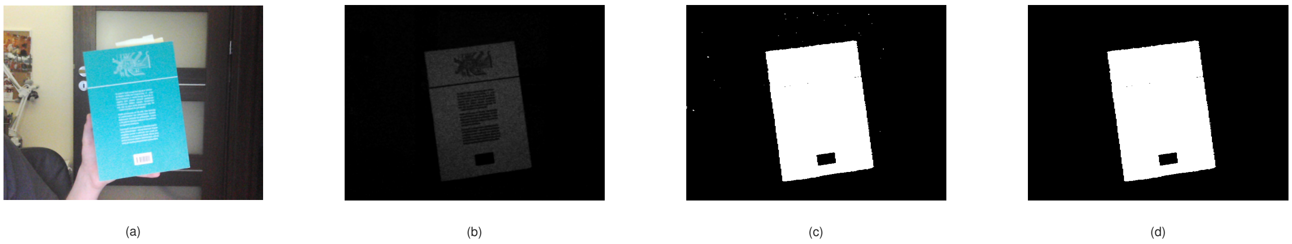
[5] A. Asvadi, M. Karami, Y. Baleghi. Object tracking using adaptive object color modeling. *Proceeding of 4th Conference on Information and Knowledge Technology*, strony 848–852, 2012.



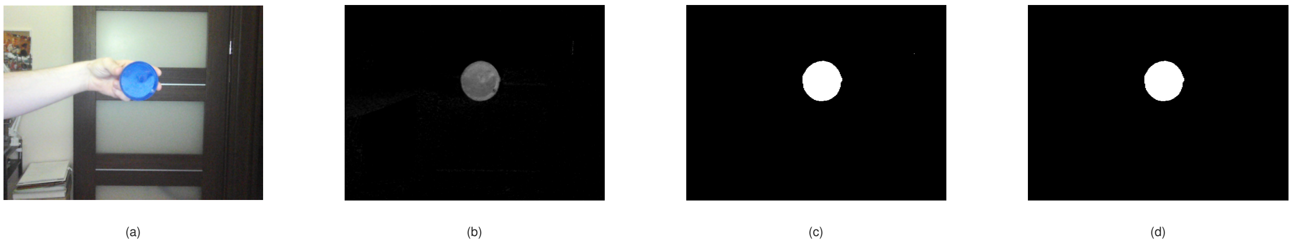
(a) Example for red layer with light shining directly on object



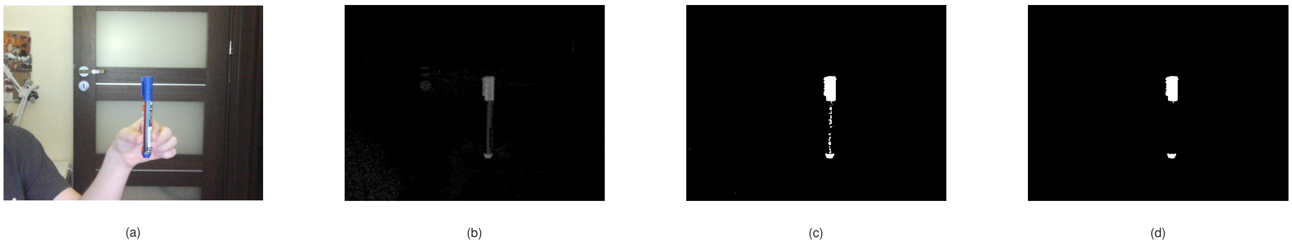
(b) Example for red layer with less amount of light



(c) Example for green layer



(d) Example for blue layer



(e) Example for blue layer

Figure 4: Results of several test of withdrawing objects from image