Технически университет – София

Факултет „Автоматика“

Техническа документация на сортировъчна машина

проект по дисциплина
„Програмно осигуряване
на средствата за измерване“

Изготвена от студенти, IV-ти курс:

Богдан Маринski, ф.н. 011216098

Константин Боев, ф.н. 011216024

Николай Кирилов, ф.н. 011217115

Райно Стефанов, ф.н. 011216006

2020 г.

Съдържание:

I. *Избор на измервана величина, описание и характеристики:*

Измерваните величини са трите стойностите за всеки пиксел (*представени, чрез 8-бита*) на изследваното изображение на капачка като те са представени в HSV (*hue, saturation, value*) цветови модел, който е по-близък до човешкото възприятие от RGB.

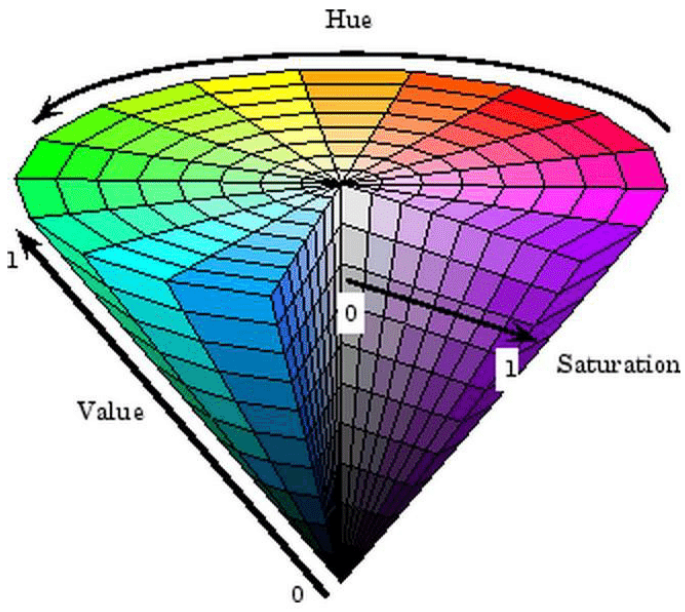
Hue (*цвят, оттенък*) – представя основните цветове и се измерва в градуси. Стойността 0° представлява червено. С увеличаване на стойността, цвета се променя и преминава през жълто, зелено, синьо, докато накрая не се върне на червено (*магента*) при стойност 360°.

Saturation (*наситеност*) – представя нивото на сиво спрямо цвета и се представя в проценти. Стойността 0% представлява най-сивия тон, а 100% – най-цветния.

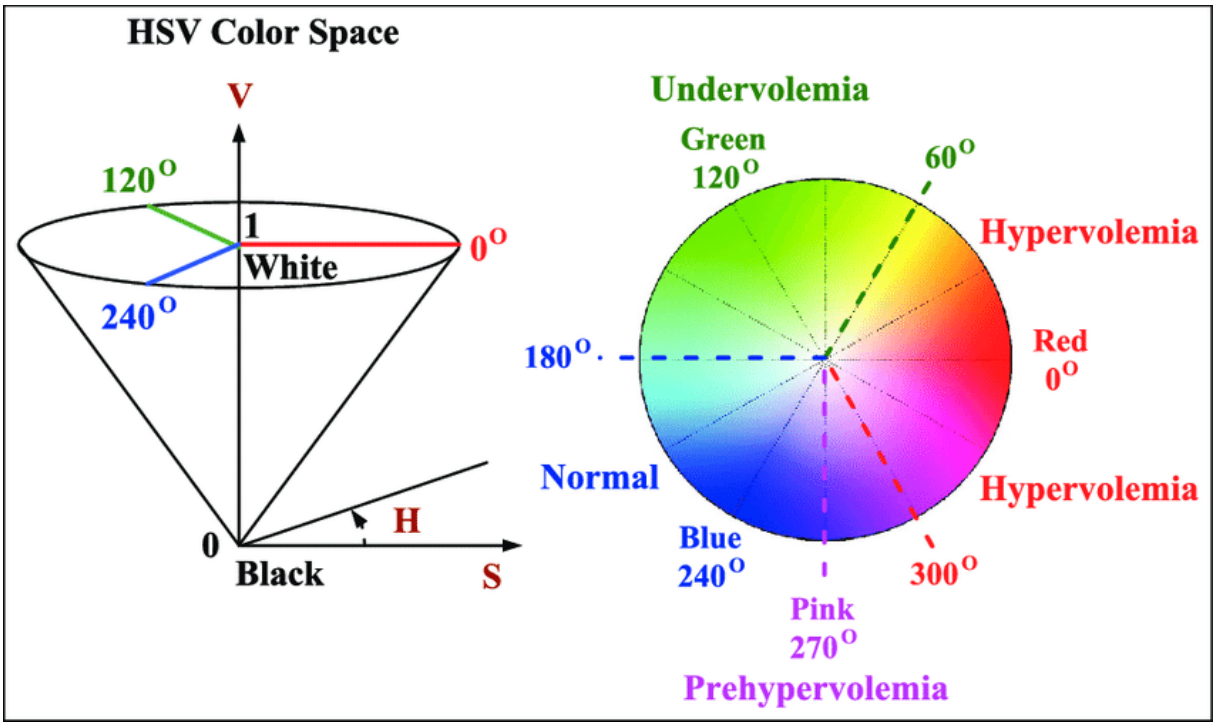
Value (*стойност*) – представя осветеността на цвета и се представя в проценти. Стойността 0% представлява чисто черно, а 100% – най-светлия нюанс.

Забележка: Saturation и Value работят заедно.

Модела е представен графично на **фигури 1.1** и **1.2**. А на **таблица 1.3** са представени някои основни цветове в RGB и HSV формат.



Фиг. 1.1



Фиг. 1.2

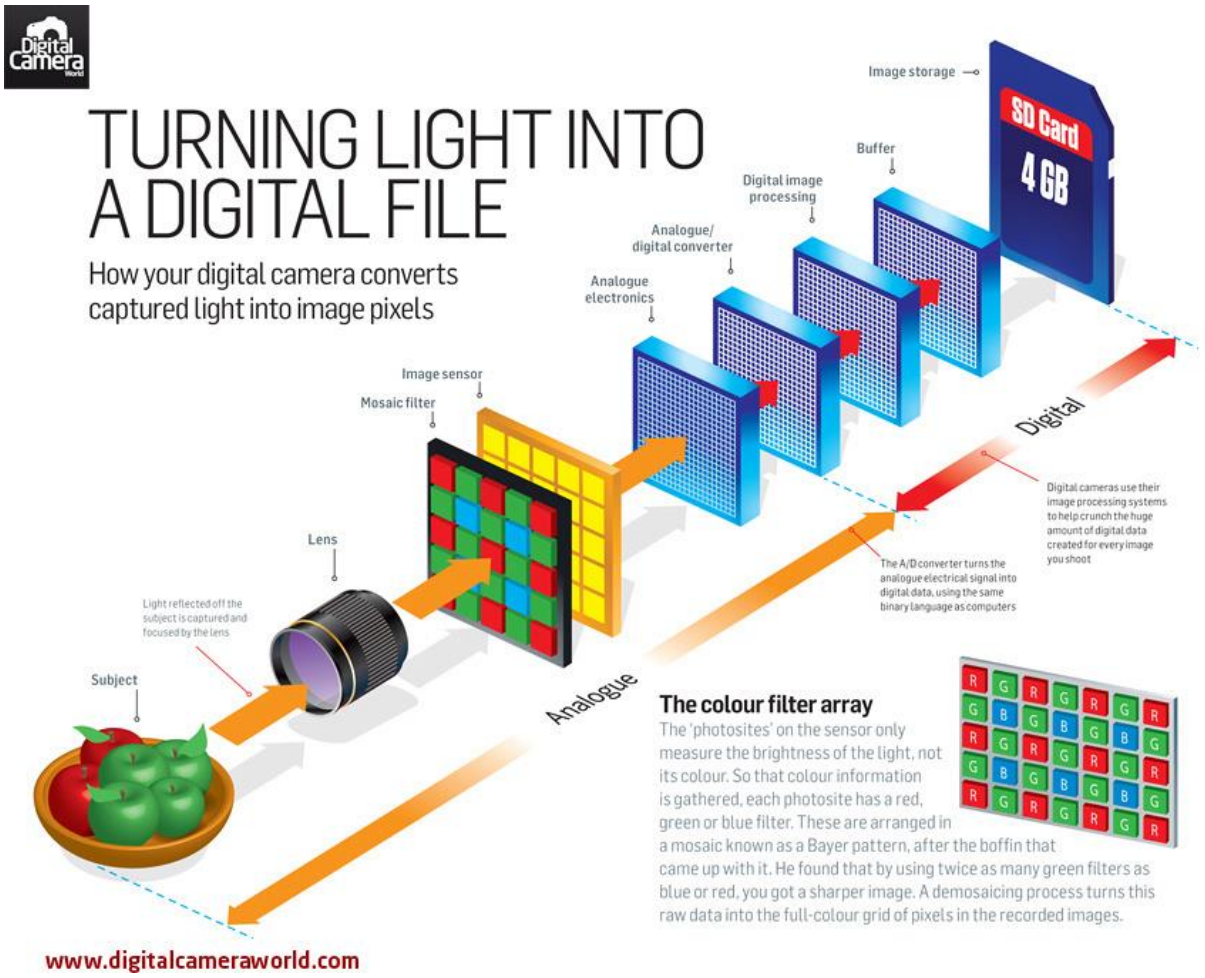
CSS 1–2.0 / HTML 3.2–4 / VGA color names

Color	Name	Hex triplet	Red	Green	Blue	Hue	Satur	Value	CGA number (name); alias
	White	#FFFFFF	100%	100%	100%	0°	0%	100%	15
	Silver	#C0C0C0	75%	75%	75%	0°	0%	75%	7 (light gray)
	Gray	#808080	50%	50%	50%	0°	0%	50%	8 (dark gray)
	Black	#000000	0%	0%	0%	0°	0%	0%	0 (black)
	Red	#FF0000	100%	0%	0%	0°	100%	100%	12 (high red)
	Maroon	#800000	50%	0%	0%	0°	100%	50%	4 (low red)
	Yellow	#FFFF00	100%	100%	0%	60°	100%	100%	14 (yellow)
	Olive	#808000	50%	50%	0%	60°	100%	50%	6 (brown)
	Lime	#00FF00	0%	100%	0%	120°	100%	100%	10 (high green); green
	Green	#008000	0%	50%	0%	120°	100%	50%	2 (low green)
	Aqua	#00FFFF	0%	100%	100%	180°	100%	100%	11 (high cyan); cyan
	Teal	#008080	0%	50%	50%	180°	100%	50%	3 (low cyan)
	Blue	#0000FF	0%	0%	100%	240°	100%	100%	9 (high blue)
	Navy	#000080	0%	0%	50%	240°	100%	50%	1 (low blue)
	Fuchsia	#FF00FF	100%	0%	100%	300°	100%	100%	13 (high magenta); magenta
	Purple	#800080	50%	0%	50%	300°	100%	50%	5 (low magenta)

Таблица 1.3

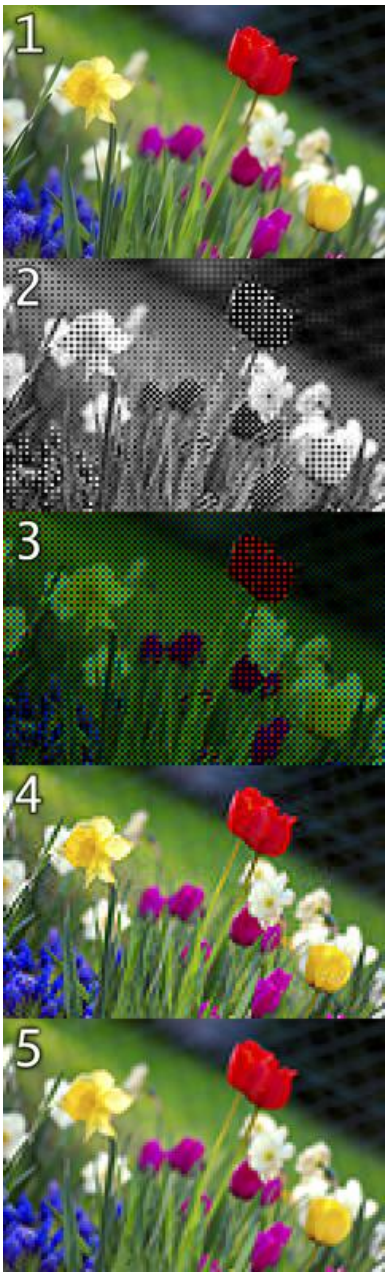
II. *Описание на реалните сензори, използвани за измерване на величината. Описание на методите за измерване:*

Измерваните величини се търсят от изображения, получавани от камера. Трябва да се уточни, че има различни видове камера за различни цели. Камерите, които са подходящи за “Machine/Computer Vision” (машинно зрение), е препоръчително да имат следните параметри: голямо бързодействие, възможност за работа при ниска осветеност, добър набор от входно-изходни връзки и добра защита от външни условия.



Фигура 2.1

На Фиг 1.1 е представена работата на съвременна дигитална камера. Отражението на наблюдаваният обект влиза в лещите на камерата и бива фокусирано. След това прожекцията преминава през филтър на Байер, който е поставен директно върху фото-сензор. Това се прави, защото фото-сензорът отчита само нивото на осветеност, която попада върху региони от него. Филтърът на Байер има червени, залени и сини зони, които са подредени като мозайка и пропускат само определените цветове. След това полученият аналогов сигнал се превръща в цифров, чрез ADC. Следва премахване на ефекта на мозайка от изображението и по-нататъшна цифрова обработка според камерата, която може да включва изглаждане и изправяне на изображението поради характера на сензора, добавяне на ефекти и компресия на данните. След това изображението се записва в цифров вид и подходящ формат, върху носител на данни. Процеса на обработка на изображението е показан на фигура 2.2.



Фигура 2.2

III. *Избор на програмно осигуряване, синтез и описание на алгоритмите за работа на сензора:*

Използвано програмно осигуряване: Windows 10 x64, LabVIEW 2019 (32-bit), Python 3.6.8 (32-bit); допълнителни библиотеки за Python: numpy 1.18.2, opencv-python 4.2.0.34, scipy 1.4.1.

Програмата се изпълнява в средата LabVIEW, докато не се натисне бутона “STOP”. Това е реализирано посредством събитийна структура в “while” цикъл. Събитийната структура има две състояния: “Timeout” и „Next”. Timeout състоянието се изпълнява докато не се натисне бутона “Next”. То единствено задава състояние “false” на индикатора “Status” (*фиг. 3.1*), което изписва “IDLE” на зелен фон и означава, че програмата изчаква команда от оператор. Това е реализирано, чрез булев флаг. Когато бутона “Next” е натиснат, събитийната структура преминава в режим “Next”. При този режим се изпълнява по-голямата и основна част от цялото приложение, а именно получаването на изображение и неговата обработка. Кода се изпълнява последователно, чрез “sequence” структура. В първия „кадър“ на тази структура се пуска виртуалният инструмент “get_cap” и се подава флаг “true” на индикатора “Status”. Във втория кадър се взима пътя към изображението за обработка (получен от “get_cap”) и то се обработва от виртуалният инструмент “color_detect”. В третия кадър се изобразяват обработваното изображение, цветовите маски и получените низови данни върху операторския панел.

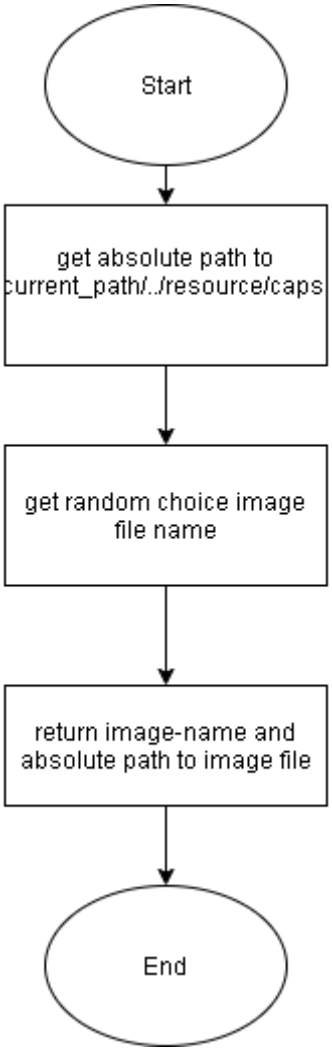
Фиг. 3.1

За целите на проекта, камерата е заменена от виртуален инструмент в LabVIEW, който се нарича “get_cap”. Основната функционалност на този инструмент е реализирана, чрез езика за програмиране Python. Посредством “Python Node” се изпълнява скрипт, който връща име и път към случайно изображение в отделна директория “caps”, която съдържа набор от изображения на капачки, снимани отгоре. За целта се използват библиотеките „random“ и „os“. Библиотеката „os“ осигурява функции, чрез който се подават заявки към ядрото на операционната система (*гарантира се функционалност за операционни системи: Windows, Linux и Mac OS*), за получаване на списък със всички файлове в избраната директория. С помощта на библиотеката “random” се избира случаен файл от този списък. Полученият път се използва за зареждане на изображението на предния панел и се подава на алгоритъма за разпознаване на цвят.

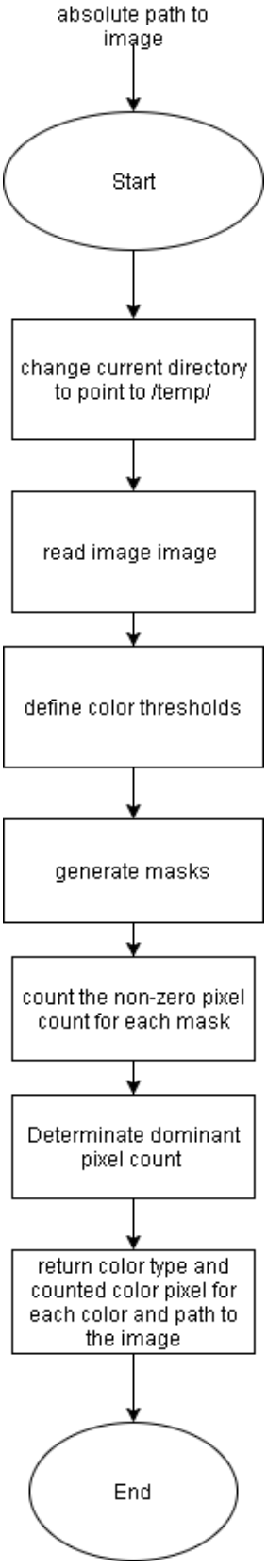
Алгоритъма за разпознаване също е реализиран, чрез програмния език Python и се казва “get_color”. Той приема път до изображението за обработка и връща разпознатия цвят и стойности на цветните пиксели като масив от низове. Изображението се зарежда в BGR цветови модел и се конвертира в HSV модел. След това се генерират маски за всеки избран цвят, посредством горни и долни граници за трите стойности на искания цвят според HSV модела. Тези маски съдържат само стойности 0 и 255. На мястото на всеки цвят, който е извън желания диапазон се записва 0, а другите се попълват с 255. Цветовите маски се записват в директория *“Sorting machine\Temp“*. Те се покажат, за по-подробна информация на операторския панел. Маските се проверяват за брой на значещите пиксели и тези бройки се записват в нов масив. Прави се проверка, кой елемент на масива има най-голям брой значещи пиксели и индекса му се предава на речник, в който са записани индексите на всяка маска и цвета, който и съответства. Речникът притежава и стойност за грешка. Тя се използва, когато няма достатъчен брой значещи пиксели за избран цвят.

Получения масив, от алгоритъма за разпознаване на цвят, се използва от операторският панел като той се разделя и съответните парчета се подават на определените за целта елементи на предния панел.

На фигури 3.2 и 3.3 са показани алгоритмите на функциите “get_cap” и “get_color” в блокова диаграма.


Start
get absolute path to current_path/../resource/caps
get random choice image file name
return image-name and absolute path to image file
End

Фиг 3.2

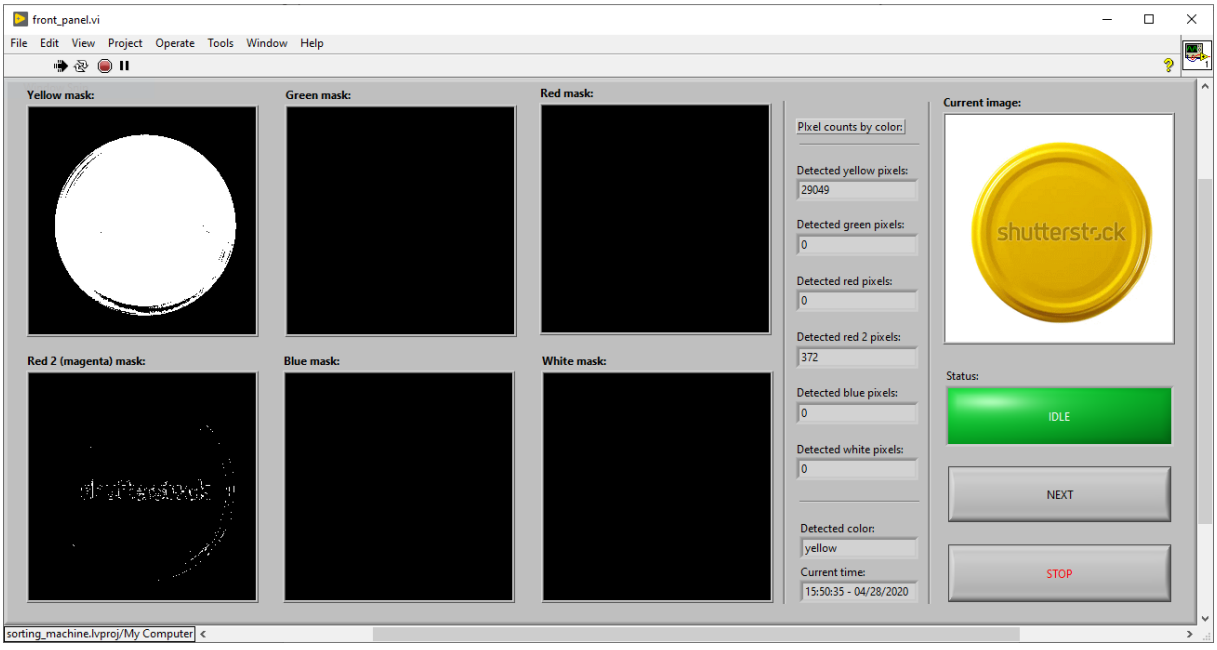

absolute path to image
Start
change current directory to point to /temp/
read image image
define color thresholds
generate masks
count the non-zero pixel count for each mask
Determinate dominant pixel count
return color type and counted color pixel for each color and path to the image
End

Фиг 3.3

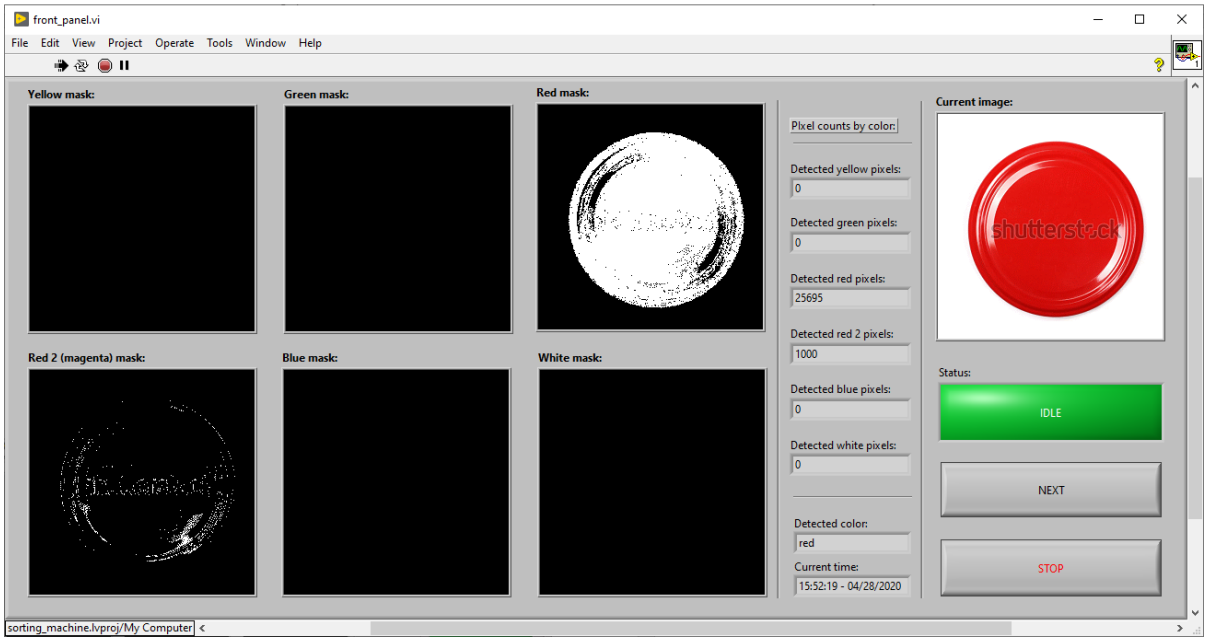
IV. *Описание на тестовите процедури и резултати от проверката на работоспособността на сензора:*

По време на разработката на Python-скриптовете се направиха редица от тестове за потвърждение на правилната им функционалност. Това се случи, чрез дебъгване в средата Visual Studio Code, с инсталирана приставка за Python поддръжка.

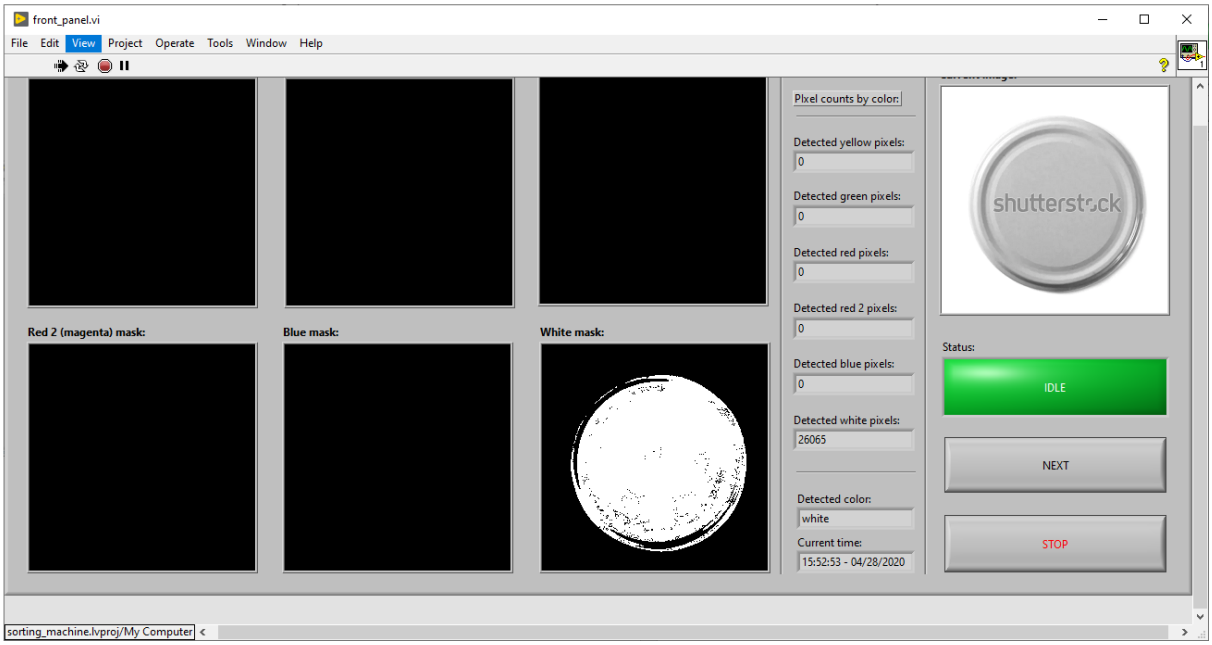
На входовете на функциите се подаваха необходимите данни и се проверяваха вътрешните променливи на функцията след всеки нов ред изпълнен код и нейният изход. Тези процедури бяха повторени след интеграцията на функциите в LabView като резултати от проверката са представени на **фигури 4.1, 4.2 и 4.3** по-долу:



Фигура 4.1



Фигура 4.2



Фигура 4.3

Използвана литература:

https://opencv-python-tutroals.readthedocs.io/en/latest/

https://forums.ni.com/t5/forums/filteredbylabelpage/board-id/170/label-name/labview?profile.language=en/

https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019ZR4SAM&l=en-BG/

https://docs.opencv.org/4.2.0/

https://stackoverflow.com/questions/

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.ExcelWriter.html

http://www.ni.com/manuals/

https://www.youtube.com/watch?v=6AY5p1uC5gM

https://docs.python.org/3.6/library/os.html/

https://en.wikipedia.org/wiki/HSL_and_HSV/

https://www.researchgate.net/profile/Wei-Ling_Chen/

https://www.shutterstock.com/

https://www.ikonphotographs.net/blog/2015/3/digital_cameras/

https://en.wikipedia.org/wiki/Bayer_filter/

https://www.quora.com/What-is-the-best-camera-for-computer-vision/