POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTROTECHNIKI I AUTOMATYKI

Imię i nazwisko studenta: Arkadiusz Skórski
Nr albumu: 168879
Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Elektrotechnika
Specjalność/profil: -

PRACA DYPLOMOWA INŻYNIERSKA

Tytuł pracy w języku polskim: Urządzenie do kontroli wizyjnej 2D płytek drukowanych z wykorzystaniem platformy Raspberry Pi

Tytuł pracy w języku angielskim: PCB visual inspection device based on Raspberry Pi platform

Potwierdzenie przyjęcia pracy	
Opiekun pracy *podpis*	Kierownik Katedry *podpis*
dr inż. Daniel Wachowiak	dr hab. inż. Jarosław Guziński

Data oddania pracy do dziekanatu:

STRESZCZENIE

W pracy zaprezentowano urządzenie do kontroli wizyjnej płytek drukowanych, które realizuje algorytmy przetwarzania i analizy obrazu. Wykorzystywano do tego komputer jednopłytkowy Raspberry Pi 3B oraz kamerę internetową o rozdzielczości jednego megapiksela. Do wyświetlenia wyników analizy obrazu wykorzystano dotykowy wyświetlacz ciekłokrystaliczny. W kolejnych rozdziałach przedstawiono szczegółowo z jakich elementów składa się projekt układu oraz pokazano implementację stworzonych algorytmów w kodzie aplikacji. Zaprezentowano również przykładowe działanie programu, który sprawdza poprawność płytek typu Adruino UNO.

Słowa kluczowe: Przetwarzanie obrazu, Programowanie, Raspberry Pi

Dziedzina nauki i techniki, zgodnie z wymogami OECD: Elektrotechnika i elektronika

ABSTRACT

The theme of this engineering thesis is a PCB visual inspection device which executes simple algorithms for image processing and recognition. The device was built from a single board computer Raspberry Pi 3B and webcam of one-megapixel resolution. The liquid-crystal display with touchscreen is used to show user interface and the results of the picture analysis. The subsequent chapters present in detail of which components the device is built and how algorithms were implemented in application's code. The example program operation was shown, which checks correctness of the Arduino UNO type printed circuit boards.

Key words: Image Processing, Programming, Raspberry Pi

SPIS TREŚCI

WYKAZ NAJWAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

ADC (ang. *Analog Digital Converter*) – przetwornik analogowo-cyfrowy

AI (ang. *Artificial Intelligence*) – sztuczna inteligencja

API (ang. *Application Programming Interface*) – interfejs programowania aplikacji

CMOS (ang. *Complementary Metal-Oxide Semiconductor*) – technologia wytwarzania układów scalonych, wykorzystywanych między innymi w matrycach światłoczułych, składających się z symetrycznej ilości par tranzystorów MOSFET o przeciwnym typie przewodnictwa (p i n).

CUDA (ang. *Compute Unified Device Architecture*) – platforma oraz interfejs programistyczny stworzony przez firmę NVIDIA do obliczenia równoległego przez procesory graficzne

FLOPS (ang. *Floating Point Operations Per Second*) – jednostka mocy obliczeniowej komputera

GPIO (ang. *General-Purpose Input/Output*) – wejście/wyjście ogólnego przeznaczenia

IoT (ang. *Internet of Things*) – Internet rzeczy

I^2C (ang. *Inter-Integrated Circuit*) – synchroniczna, dwukierunkowa magistrala danych

I^2S (ang. *Inter-IC Sound*) – magistala służąca do łączenia ze sobą urządzeń audio.

LCD (ang. *Liquid-Crystal Display*) – wyświetlacz ciekłokrystaliczny

OS (ang. *Operating System*) – system operacyjny

PCB (ang. *Printed Circuit Board*) – płytka drukowana

RAM (ang. *Random-Access Memory*) – pamięć operacyjna o dostępie swobodnym

RGB (ang. *Red, Green, Blue*) – jeden z modeli przestrzeni barw, który jest opisywany współrzędnymi koloru czerwonego, zielonego oraz niebieskiego

SBC (ang. *Single-Board Computer*) – komputer jednopłytkowy

SDIO (ang. *Secure Digital Input/Output*) – interfejs wejść/wyjść dla kart SD

SDK (ang. *Software Development Kit*) – zestaw narzędzi programowania do tworzenia aplikacji

SoC (ang. *System-on-a-Chip*) – układ scalony posiadający kompletny system elektroniczny

SPI (ang. *Serial Peripheral Interface*) – szeregowy interfejs urządzeń peryferyjnych

SQL (ang. *Structured Query Language*) – strukturalny język zapytań do zarządzania bazami danych

UART (ang. *Universal Asynchronous Receiver-Transmitter*) – uniwersalny asynchroniczny nadajnik-odbiornik, służący do przekazywania i odbierania informacji przez port szeregowy

UWP (ang. *Universal Windows Platform*) – interfejs programowania aplikacji (API) od firmy Microsoft stosowany w systemach z rodziny Windows 10 i Xbox.

1 WSTĘP I CEL PRACY

Techniki cyfrowego rozpoznawania obrazu są bardzo ważnym elementem coraz większej ilości urządzeń przechwytujących wideo, używanych przede wszystkim w dziedzinach przemysłu, cyberbezpieczeństwa, fotografii, ale też w zastosowaniach automatyki budynkowej. Kamera podłączona do komputera, który oblicza i przetwarza zdjęcie lub nagranie, pozwala na przykład na:

- wykorzystanie kontroli wizyjnej płytek drukowanych w fabrykach podzespołów elektronicznych,
- zabezpieczenie poufnych danych poprzez algorytmy rozpoznawania twarzy,
- weryfikację tekstową tablic rejestracyjnych samochodów przy wjazdach na obszary parkingowe.

Rozwój technologii pozwolił na łatwiejszą implementację bardziej skomplikowanych algorytmów rozpoznawania i obróbki obrazu do komputerów o mniejszych rozmiarach, ale z wystarczająco wysoką mocą obliczeniową, aby zachować odpowiednią szybkość działania. Do takich komputerów można zaliczyć komputery jednopłytkowe, które posiadają kompletne podzespoły elektroniczne potrzebne do zainstalowania systemu operacyjnego i urządzeń peryferyjnych, o wymiarach niekiedy mniejszych niż karty płatnicze. Do takich minikomputerów zalicza się rodzinę Raspberry Pi. Tego typu platforma, oprócz portów USB oraz audio/wideo, posiada również porty GPIO – wejścia/wyjścia ogólnego przeznaczenia, do których można podłączyć mniejsze podzespoły, takie jak na przykład rezystory, diody, czujniki czy nawet silniki krokowe, aby tworzyć i sterować skomplikowanymi urządzeniami.

W niniejszej pracy poruszono dwa zagadnienia. Pierwszym z nich są cyfrowe metody analizy i przetwarzania obrazu. Opisano szczegółowo wszystkie wykorzystane algorytmy w programie oraz przedstawiono ich sposób działania na przykładowym zdjęciu.

Drugim zagadnieniem jest zaprojektowanie i stworzenie urządzenia do kontroli wizyjnej płytek drukowanych z wykorzystaniem platformy Raspberry Pi oraz napisanie aplikacji, która pozwoli na odczyt danych przesyłanych z kamery i wykonanie prostego algorytmu weryfikacji na podstawie przygotowanych zdjęć wzorcowych płytek drukowanych.

2 CYFROWE METODY PRZETWARZANIA I ANALIZY OBRAZU

2.1 Metody przetwarzania obrazu.

Przetwarzanie obrazów jest to zastosowanie funkcji powodujących między innymi: zmianę lub poprawę jakości zdjęcia źródłowego; uwypuklenie lub wyciągnięcie informacji powodujących lepszą możliwość dalszej analizy bądź obserwacji danego obrazu; przekształcenie fotografii do formy pozwalającej na łatwiejsze operacje wykonywane w kodzie programu. Początkowo metody miały charakter czysto estetyczny, ponieważ pozwalały one na przykład na zmianę odcieni szarości, poprawę kontrastu i ostrości czy redukowanie szumów i wyostrzanie krawędzi. Wiązało się to przede wszystkim z mniejszą mocą obliczeniową komputerów, które uniemożliwiały wykonywanie bardziej skomplikowanych obliczeń przy odpowiednio krótkim czasie działania.

Rozwój komputerów i postępująca miniaturyzacja doprowadziła do sytuacji, w której cyfrowe metody przetwarzania obrazów stały się bardzo proste w implementacji, ponieważ nawet urządzenie wielkości karty kredytowej jest w stanie poradzić sobie ze sprawnym obliczaniem i przetwarzaniem zdjęć mających niekiedy nawet po kilkadziesiąt milionów pikseli. Z tego względu powstają coraz bardziej złożone algorytmy, ale istnieje też szeroki zakres prostszych metod przetwarzania obrazów. Każda metoda może mieć zastosowanie ogólne lub być stosowana konkretnie w danej dziedzinie, gdzie ciągłe udoskonalanie algorytmów jest istotnym elementem zwiększania efektywności systemów stosowanych na przykład w astronomii i technologiach satelitarnych, medycynie, geologii, robotyce, przemyśle oraz badaniach naukowych [1].

Warto zauważyć, że obraz to wygodny sposób przechowywania informacji, gdyż jest bardzo łatwo przyswajalny dla człowieka. Codziennie do Internetu przesyłane są miliony zdjęć i większość z nich została w jakiś sposób przetworzona, aby informacja na nich zawarta lepiej trafiła do odbiorcy lub zajmowała mniej pamięci na serwerach poprzez odpowiednią kompresję.

Podział algorytmów przetwarzania obrazów jest niezmiernie trudny do zrealizowania, ponieważ każdy obraz można przekształcić na nieskończenie wiele sposobów, które mogą przynieść mniej lub bardziej praktyczne efekty. Co więcej, dwa różne typy przekształceń mogą dać ten sam wynik końcowy, ale także dwie zbliżone do siebie metody mogą dać odmienny obraz wyjściowy. W niniejszej pracy inżynierskiej zostaną przedstawione trzy podstawowe grupy przekształceń [2], do których przypisano algorytmy wykorzystane w programie:

- przekształcenia geometryczne,
- przekształcenia bezkontekstowe (punktowe),
- przekształcenia kontekstowe.

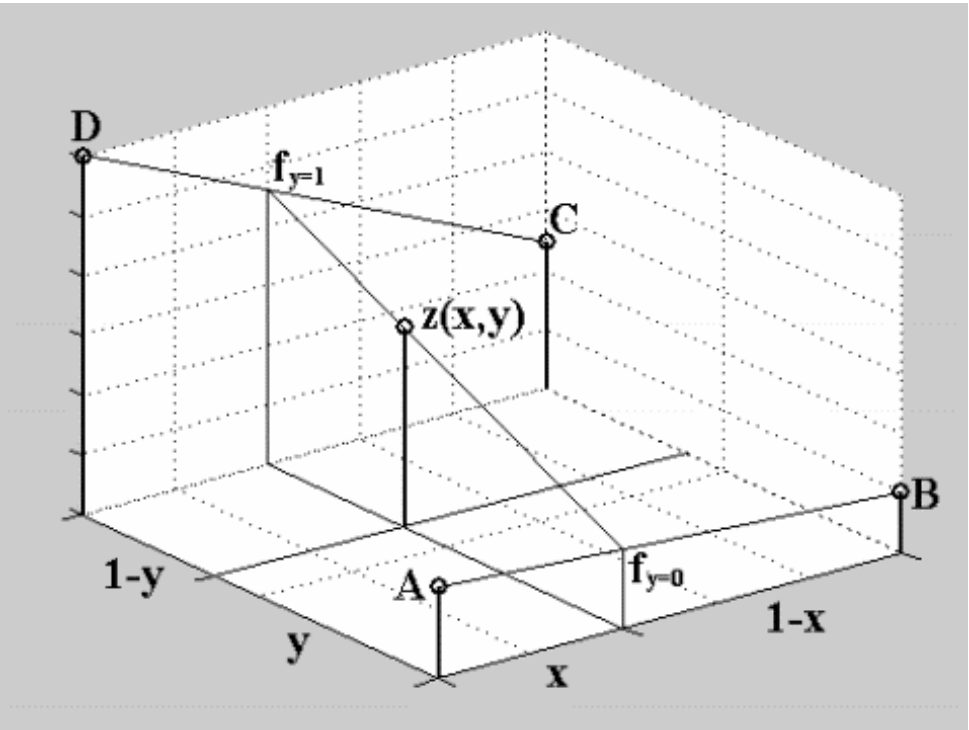
Przekształcenia geometryczne są najprostszą formą przetwarzania obrazów. Do nich należą przede wszystkim transformacje oparte na przesunięciach, obrotach czy deformacja. Jednak należy zauważyć, że każda prosta manipulacja obrazem może posiadać ogromną ilość podobnych wariantów. Przeważnie tego typu metody stosuje się jako pomocnicze przy wykonywaniu części bardziej złożonego przekształcenia. W programie stworzonym na potrzeby

realizacji projektu inżynierskiego wykorzystano algorytm obrotu obrazu o wielokrotność 90°, a także algorytm skalowania obrazu, który wykorzystuje interpolację.

Interpolacja jest jednym z bardziej złożonych przekształceń i w grafice jest ona wykorzystywana do zmieniania rozmiaru zdjęcia, które jest reprezentowane w dwuwymiarowym układzie współrzędnych kartezjańskich. Najczęściej wykorzystywane sposoby skalowania fotografii to metoda „najbliższego sąsiada", interpolacja dwuliniowa i dwusześcienna.

Metoda „najbliższego sąsiada" jest najprostsza, gdyż zasada działania opiera się na powielaniu pikseli obrazu wejściowego przy zwiększaniu rozmiaru lub pomijaniu przy zmniejszaniu. Każdy piksel ma tylko jednego najbliższego odpowiednika, którego wartość jest kopiowana z obrazu źródłowego do wyjściowego. Ten algorytm jest głównie wykorzystywany, gdy najważniejszym czynnikiem jest czas działania funkcji lub zdjęcie składa się głównie z pionowych lub poziomych linii, ponieważ w innym wypadku fotografia zostaje mocno postrzępiona przez brak użycia wygładzania krawędzi.

Interpolacja dwuliniowa, która została wykorzystana w projekcie inżynierskim, nosi nazwę odnoszącą się do jądra interpolacji, którego kształt jest funkcją liniową. W grafice komputerowej jest to złożenie dwóch interpolacji liniowych – jedna dla szerokości zdjęcia i druga dla wysokości. Do obliczeń w algorytmie brane są pod uwagę wartości czterech sąsiadujących pikseli, które stykają się bokami z danym pikselem. Rezultatem są wygładzone krawędzie, jednak ze względu na liniową funkcję jądra, rozmycie przeprowadzone jest tylko w kierunku pionowym i poziomym, więc odwzorowanie przy bardziej skomplikowanych kształtach jest problematyczne. Zasadę działania interpolacji dwuliniowej przedstawiono na rysunku 2.1, którą opisano wzorem 2.1, gdzie z_a, z_b, z_c, z_d, to wartości zapisane w danym pikselu odpowiednio dla sąsiadujących pikseli obrazu źródłowego w punkcie A, B, C i D, a funkcja z(x,y) to obraz wyjściowy:



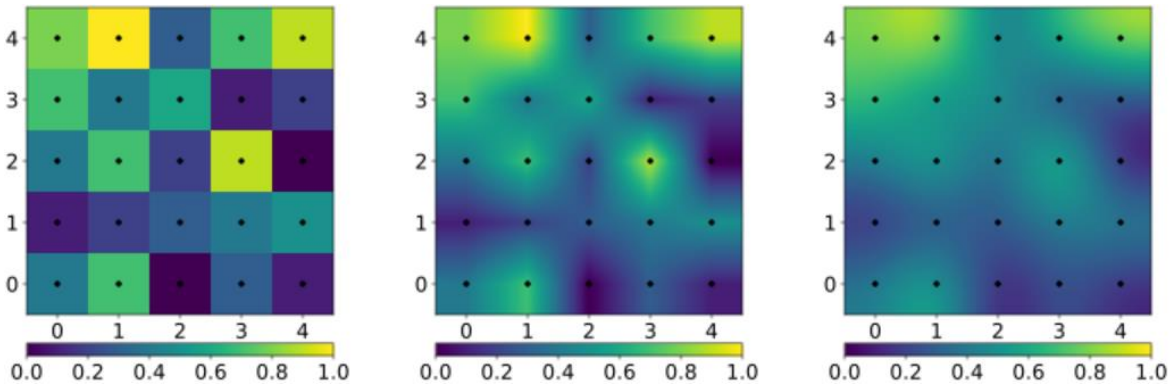
Rys. 2.1. Zasada działania interpolacji dwuliniowej.

$$z[x,y] = z_a(1-x)(1-y) + z_b x(1-y) + z_c xy + z_d(1-x)y \tag{2.1}$$

Interpolacja dwusześcienna jest domyślną opcją w większości programów graficznych głównie ze względu na łagodniejsze generowanie krawędzi niż w interpolacji liniowej, ponieważ obejmuje ona większy zakres sprawdzanego obszaru i inaczej go uśrednia. W tej interpolacji piksel poddawany jest najpierw metodzie „najbliższego sąsiada", a następnie nowy piksel oraz bezpośrednio z nim sąsiadujące są arytmetycznie uśredniane, aby uzyskać wyjściowy piksel. Interpolację dwusześcienną (4-punktową) określa się wzorem 2.2, gdzie funkcja z(x,y) to obraz wyjściowy i g(x,y) to obraz po zastosowaniu metody „najbliższego sąsiada" [3]:

$$z[x,y] = \sum_{i=x-1}^{x+1} \sum_{j=y-1}^{y+1} g[i,j]/9 \tag{2.2}$$

Poniżej (rys. 2.2) przedstawiono efekt interpolacji tego samego obrazu przez opisywane wyżej metody.



Rys. 2.2. (od lewej) Metoda „najbliższego sąsiada", Interpolacja liniowa i interpolacja dwusześcienna.

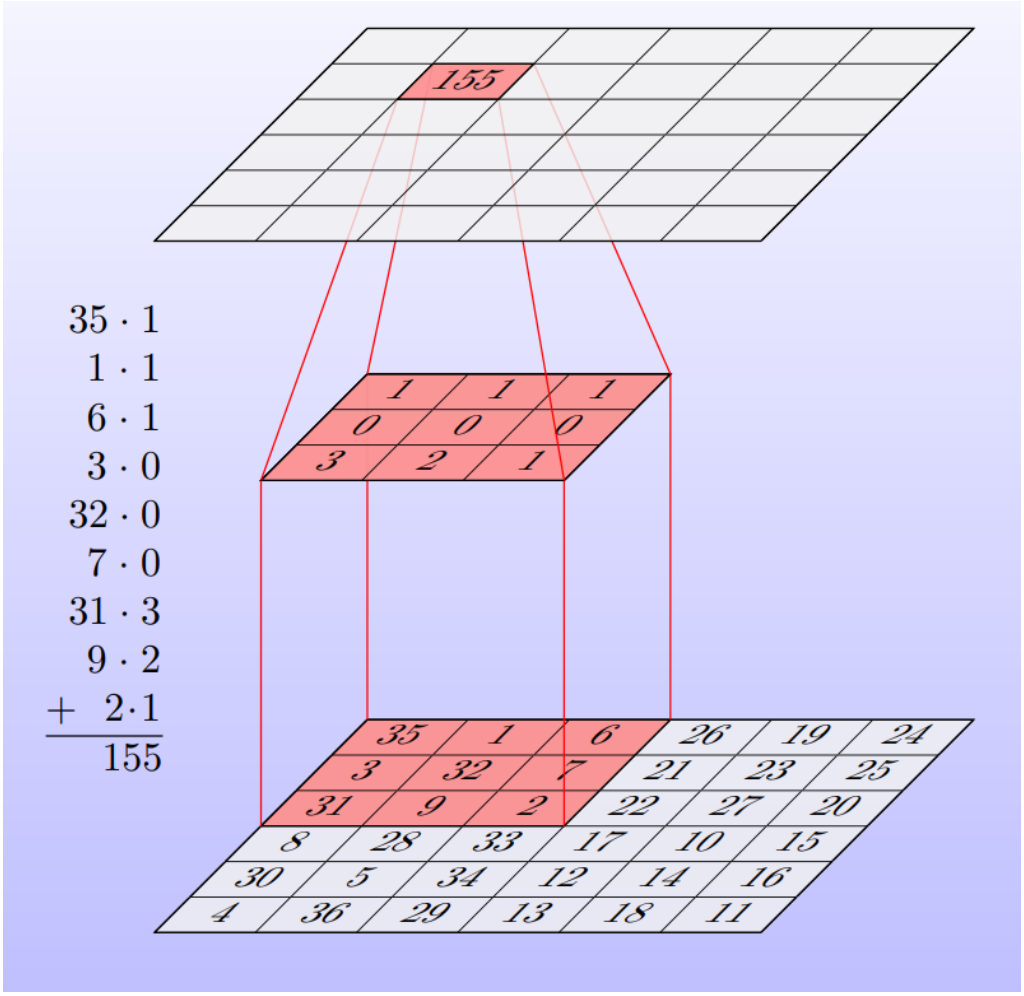
Przekształcenia bezkontekstowe (punktowe) to operacje wykonywane wyłącznie na pojedynczym punkcie danego obrazu, bez uwzględnienia innych otaczających go pikseli. Z tego względu obliczenia takich przekształceń wykonuje się szybko i relatywnie łatwo. Do takich operacji można zaliczyć: odwrócenie kolorów (negatyw), zmiana jasności czy korekcja barw. Przeprowadzane modyfikacje są zawsze odwracalne, jeśli wprowadzający zmiany algorytm wykorzystuje funkcję ściśle monotoniczną. W innym przypadku, część danych zawartych na zdjęciu może zostać bezpowrotnie utracona [2]. W projekcie inżynierskim wykorzystano dwa algorytmy wykorzystujące ten typ przekształceń: mapa cieplna, która oblicza wartość bezwzględną z różnicy wartości średniej kolorów (dla koloru czerwonego, zielonego i niebieskiego) pomiędzy dwoma obrazami, a także przekształcenie czarno-białe zdjęcia.

Utworzenie obrazu w odcieniach szarości wiąże się z ustawieniem jednakowej wartości na wszystkich trzech kanałach barw w pikselu i może się odbywać na kilka różnych (punktowych) sposobów. Najprostsza, ale też rzadko używana, jest metoda na arytmetyczne uśrednienie barw z trzech kanałów RGB. Inną możliwością jest zastosowanie desaturacji, która polega na uśrednieniu wartości maksymalnej dodanej do wartości minimalnej pojawiających się w trzech

kanałach. Innym podejściem jest tzw. dekompozycja, w której odcień szarości jest ustawiony na podstawie maksymalnej wartości koloru z trzech kanałów lub na minimalnej wartości. Jest również możliwość ręcznego wybrania dominującego kanału, na podstawie którego inne barwy będą przyjmowały jego wartość. Jednak najczęstszą wybieraną metodą jest ważone uśrednianie barw do uzyskania czarno-białego obrazu. Powodem tego jest różnica w postrzeganiu barw przez ludzkie oko. Człowiek rozróżnia bardziej kolory zielone od innych, dlatego w projekcie inżynierskim wykorzystano tą metodę, która jest również używana w radiotelekomunikacji oraz przez programy do zaawanasowanej obróbki fotografii [4]. Istnieje pewna rozbieżność w przyjmowania wagi dla poszczególnych kanałów barw. Te wartości oscylują między 0,59-0,71 dla koloru zielonego, 0,21-0,30 dla koloru czerwonego i 0,07-0,11 dla koloru niebieskiego.

Przekształcenia kontekstowe wykonywane przez filtry (inne nazwy to: jądro przekształcenia, maska, macierz), które modyfikują konkretne piksele zdjęcia w zależności od wartości ich samych oraz ich otoczenia, są bardzo złożone, gdyż dla każdego pojedynczego punktu, algorytm musi sprawdzić ustalony obszar sięgający od kilku do nawet kilkudziesięciu pikseli w zależności od wybranego zastosowania. Zazwyczaj maski algorytmów są proste i regularne, najczęściej złożone z liczb całkowitych, więc nie obciążają procesora. Co więcej, takie jądro przekształcenia może być nakładane jednocześnie na wszystkie piksele w obrazie, jeśli wykorzysta się do tego programowanie wielowątkowe. Problemem jest niemożność wykonania operacji filtracji dla pikseli położonych przy brzegach fotografii. Takie przekształcenia są bardzo często wykorzystywane, głównie do redukowania szumów powstających w obrazie lub wyostrzenia pewnych elementów na przetwarzanym obrazie. Wyróżnia się filtry liniowe, które filtrują obraz maskami z liniową kombinacją liczb i są z reguły prostsze w wykonania, a także filtry nieliniowe, posiadające nieliniową kombinację cyfr w swoich jądrach przekształceń.

W filtrach liniowych wykonuje się operację dyskretnego splotu funkcji, zwaną także konwolucją. W przypadku analizy obrazu wykonywany jest splot zdjęcia wejściowego zapisanego w postaci macierzy z macierzą filtru z pominięciem brzegów. Na rysunku 2.3 przedstawiono zasadę realizacji algorytmu oraz zaprezentowano równanie opisujące dyskretny splot dwuwymiarowych tablic (2.3):



Rys. 2.3. Zasada działania dyskretnego splotu macierzy [5].

$$h[x,y] = \sum_{i=x-b}^{x+b} \sum_{j=x-b}^{y+b} f[i,j] * k[x-i, y-j] \tag{2.3}$$

gdzie:

- f(x, y) – obraz wejściowy,
- h(x, y) – obraz wyjściowy,
- k(x, y) – jądro (kernel) przekształcenia,
- b – odległość środkowego piksela maski od granicy macierzy.

W projekcie inżynierskim wykorzystano filtr liniowy (górnoprzepustowy) do realizacji algorytmu detekcji krawędzi. W pierwszej kolejności przekształca obraz źródłowy do odcieni szarości. Następnie, oblicza gradient intensywności dla dwóch kierunków, ponieważ w najprostszym rozumieniu krawędź to linia oddzielająca obszary o różnej jasności. Jeśli algorytm wykryje gwałtowną zmianę to wynik na obrazie wyjściowym będzie duży, a duża wartość ustawiona na wszystkich kanałach barw oznacza kolor biały. Jako maski zastosowano operatory

Sobela, Prewitta i Kirscha, wszystkie dla dwóch różnych kierunków. Poniżej przedstawiono wartości macierzy dla operatorów: Sobela w wymiarze „x” (2.4) i wymiarze „y” (2.5), Prewitta w wymiarze „x” (2.6) i w wymiarze „y” (2.7) oraz Kirscha w wymiarze „x” (2.8) i wymiarze „y” (2.9).

$$xSobel = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$ySobel = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.5}$$

$$xPrewitt = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$yPrewitt = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{2.7}$$

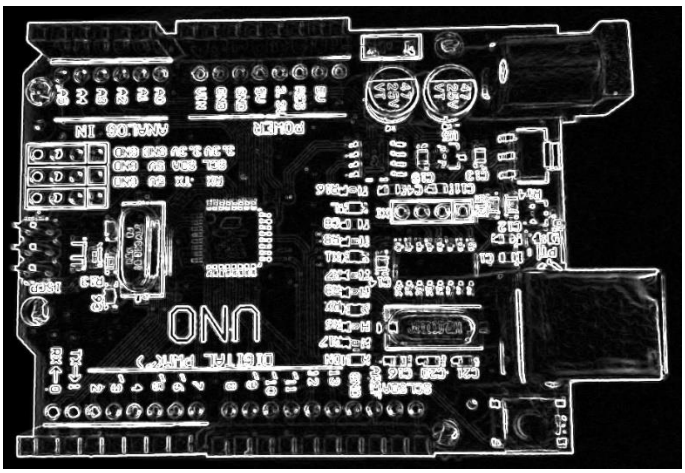
$$xKirsch = \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \tag{2.8}$$

$$yKirsch = \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \tag{2.9}$$

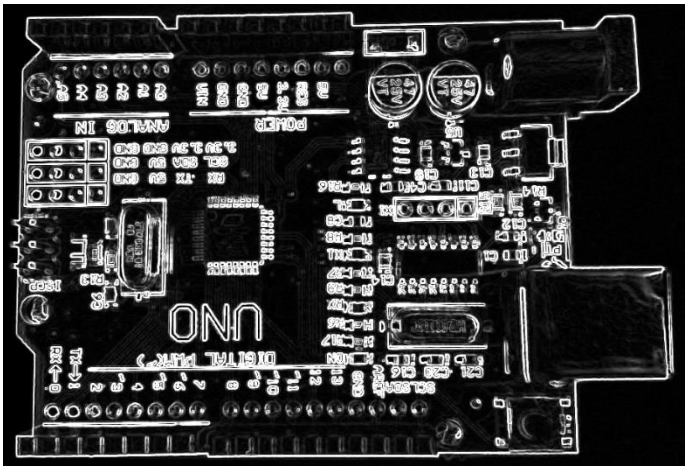
Ze względu na wykorzystanie dwóch masek otrzymano również dwa obrazy wyjściowe. Do kombinowania tych obrazów h_x i h_y użyto formuły Euklidesowej (2.10), aby uzyskać ostateczny obraz wyjściowy h:

$$h[x,y] = \sqrt{(h_x[x,y])^2 + (h_y[x,y])^2} \tag{2.10}$$

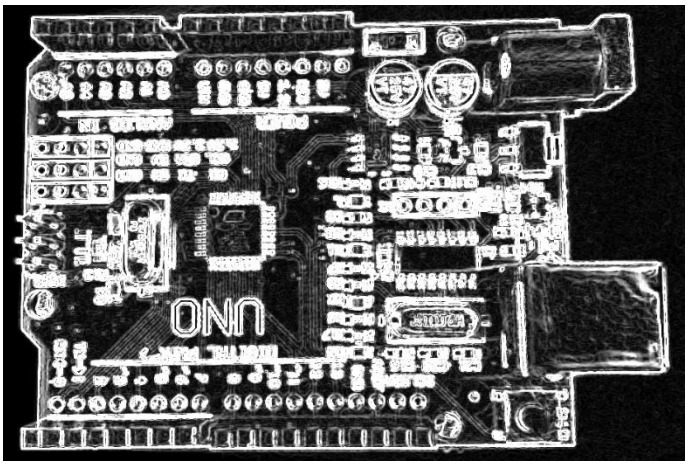
Poniżej przedstawiono wyniki nałożenia maski Sobela (rys. 2.4), Prewitta (rys. 2.5) i Kirsha (rys. 2.6) na to samo zdjęcie wejściowe.



Rys. 2.4. Przedstawienie działania detekcji krawędzi z wykorzystaniem operatora Sobela.

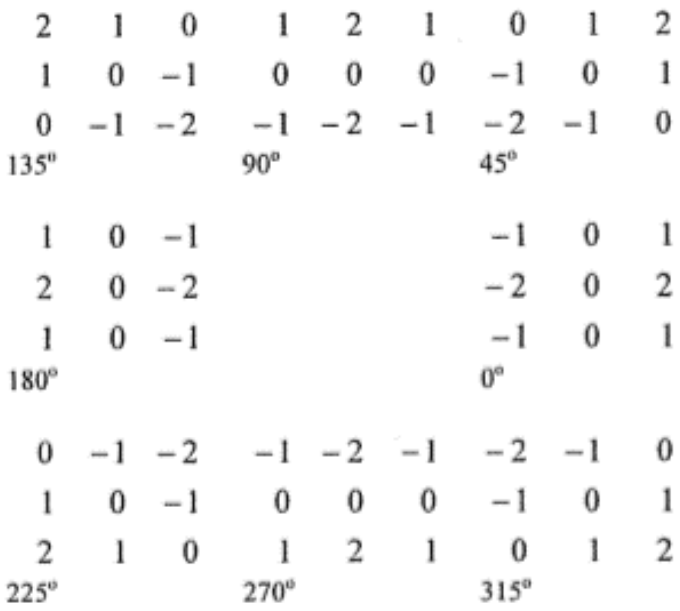


Rys. 2.5. Przedstawienie działania detekcji krawędzi z wykorzystaniem operatora Prewitta.



Rys. 2.6. Przedstawienie działania detekcji krawędzi z wykorzystaniem operatora Kirscha.

Algorytm w programie obliczenia od lewej do prawej dla wymiaru „x” oraz od góry do dołu dla wymiaru „y”, z tego względu wszystkie powyższe maski są wykorzystane dla przesunięcia 0° i 270°, jednak istnieje możliwość stworzenia tzw. kompasu Sobela (rys. 2.7) lub Kirscha, składającego się z przesuniętych macierzy o wielokrotność 45°.



Rys. 2.7. Operatory Sobela dla różnych kierunków.

Bardzo zawansowane metody przetwarzania obrazu wykorzystują niekiedy kilka różnych typów przekształceń jednocześnie. Przykład może być algorytm upraszczający liczbę kolorów stworzony na potrzeby projektu inżynierskiego. Na początku wykonuje on operację przekształcenia punktowego, polegającego na dzieleniu bez reszty wartości koloru danego piksela przez współczynnik upraszczających, a następnie ponowne wymnożenie przez niego. Drugim etapem jest przekształcenie kontaktowe, które sprawdza otoczenie danego piksela i jeśli w nim jest co najmniej 55% pikseli tego samego koloru to całe otoczenie zmienia na ten kolor.

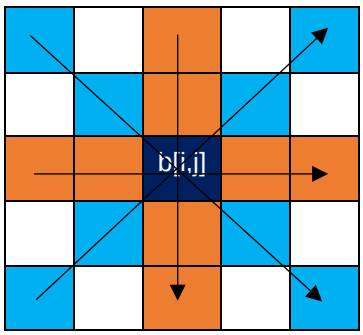
Innym przekształceniem wykorzystywanym w przetwarzaniu obrazów, które nie zostało zaimplementowane do żadnego algorytmu projektu inżynierskiego, jest przekształcenie widmowe. Jest ono pod wieloma względami bardzo podobne do przekształceń kontekstowych, jednak tutaj nie piksel poddawany jest różnym operacją, a w istocie cały obraz. W tej metodzie tworzone jest za pomocą Dyskretnej Transformacji Fouriera DTF dwuwymiarowe widmo obrazu i następnie jest poddawane modyfikacji, na przykład usuwaniu składowych o wysokich częstotliwościach. Przetworzone widmo można następnie za pomocą odwrotnej transformacji Fouriera przywrócić do pierwotnej formy. Taka metoda pozwala na bardziej precyzyjną manipulację danymi, ale wiąże się to z zwiększonym obciążeniem procesora [2]. Jest ona rzadziej stosowana od filtrów kontekstowych, ponieważ jest trudniejsza w implementacji w kodzie programu, a także wymaga minimalnej wiedzy z zakresu cyfrowego przetwarzania sygnałów.

2.2 Metody analizy obrazu.

Techniki rozpoznawania obrazu są coraz powszechniej stosowane praktycznie w każdej dziedzinie. Takie metody przede wszystkim pozwalają zaoszczędzić czas w porównaniu do metod „analogowych”. Rozpoznawanie tekstu na przykład na tablicach rejestracyjny pozwala zautomatyzować proces na obszarach parkingowych, a w zaawansowanych aplikacjach tłumacza na smartfony można zaoszczędzić czas robiąc zdjęcie danego tekstu i w wygodny sposób przetłumacz bez konieczności ręcznego wpisywania. Innym przykładem jest wykorzystanie rozpoznawania obrazu do zabezpieczenia poufnych danych przez zmapowanie twarzy użytkownika, a nawet palca, ponieważ optyczne czytniki papilarne również zapisują dane w postaci zdjęcia do późniejszej analizy.

W projekcie inżynierskim skupiono się na zagadnieniu rozpoznawania obrazu w warunkach przemysłowych płytek drukowanych. Tutaj metody rozpoznawania obrazu są najmniej skomplikowane, ponieważ warunki testowe takie jak natężenie oświetlenia czy temperatura barwowa są w przybliżeniu niezmienne. Często algorytmy wykonuje się wewnątrz maszyn, które lutują komponenty elektroniczne na laminacie płytki, gdzie jest mały wpływ warunków zewnętrznych na wynik analizy.

Pierwszą funkcją rozpoznawania obrazu stworzoną na potrzeby projektu inżynierskiego, jest algorytm rozpoznający tło. W tym przypadku założono, że tło obszaru testowego, gdzie sprawdzane są płytki jest białe. Funkcja sprawdza w ośmiu kierunkach do granicy ustalonego obszaru sprawdzania czy kolor na dwóch zdjęciach w danym punkcie jest zbliżony do białego. Jeśli przynajmniej 90% sprawdzonych pikseli jest w przybliżeniu białych to wtedy sprawdzany piksel może zostać rozważony jako tło. Zasada działania pokazano na rysunku 2.8, gdzie kolorem ciemnoniebieskim zaznaczono piksel sprawdzany, białym piksele nie sprawdzane, jasnoniebieskim i pomarańczowym piksele sprawdzane przez algorytm z zaznaczonymi kierunkami działania funkcji.



Rys. 2.8. Zasada działania metody rozpoznawania tła.

Piksel środkowy nie jest sprawdzany. Należy zauważyć, że zaprezentowano działanie przy masce sprawdzającej o wymiarach 5x5. W przygotowanej funkcji zastosowano maskę o rozmiarze 15x15. Dla algorytmu detekcji krawędzi zastosowano praktycznie identyczną wersję, ale przeznaczoną do sprawdzania czarnego tła, gdyż taki kolor jest ustawiony w przypadku nie wykrycia krawędzi na badanym zdjęciu.

Sam algorytm do przeprowadzenia analizy testowanego obrazu do wzorcowego jest bardzo prosty. Polega na porównaniu „piksel w piksel” wartości dla trzech kanałów barw. Obliczana jest wartość bezwzględna z różnicy w pikselach pomiędzy jednym a drugim zdjęciem. Kolejny etap to sprawdzenie czy algorytm rozpoznający tło nie zwrócił dla tych współrzędnych informacji o wykryciu tła. Jeżeli w tym miejscu nie ma tła to kolejnym krokiem jest sprawdzenie czy wartość różnicy nie przekracza ustalonej wartości tolerancji. Wynik jest pozytywny, gdy tolerancja mieści się w ustalonym zakresie, a negatywny jak go przekracza. W między czasie zliczana jest ilość pikseli dobrych, złych i tła, a po sprawdzeniu wszystkich pikseli w danym zdjęciu obliczany jest wynik przeprowadzonej próby.

Dla łatwiejszej reprezentacji wyników analizy stworzono również funkcję, która przedstawia na „mapie wynikowej” piksele, które przeszły weryfikację pozytywnie na zielono, negatywnie na czerwono, a piksele tła są pokazane na niebiesko.

Metoda w specyficznych warunkach zbliżonych do tych panujących na linii produkcyjnej jest w stanie poradzić sobie dość dokładnie z rozpoznawaniem płytek PCB. W fabrykach podzespołów elektronicznych takie rozwiązanie mogłoby być stosowane na każdym etapie lutowania komponentów przez maszyny w celu weryfikacji poprawności procesu. Oprócz zgrubnej analizy, należałoby ustawić punkty szczególnych zainteresowań w miejscach, w których komponent został umieszczony, gdzie próg zdania powinien być odpowiednio większy lub zmodyfikowany pod kątem wykrycia za dużych przesunięć.

Na rysunku 2.9 pokazano wynik rozpoznawania w przypadku usunięcia rezystora z miejsca zaznaczonego pomarańczowym kwadratem.

Rys. 2.9. Reakcja algorytmu rozpoznawania obrazu na usunięcie z płytki PCB rezystora.

W takim wypadku, jeśli ustalony zostałby w tym miejscu punkt szczególnego zainteresowania to algorytm wykryłby za dużą różnicę w kolorach na tym obszarze, co pozwoliłoby na powiadomienie o tym fakcie pracownika obsługującego urządzenie lutujące.

Innym bardziej złożonym podejściem do rozpoznawania obrazu jest segmentacja. Głównym założeniem jest wyodrębnienie z pełnej informacji zawartej na zdjęciu tej części, która jest istotna dla konkretnego zastosowania. Im mniejsza jest dana informacja tym łatwiej ją przeanalizować, dlatego w segmentacji dąży się do dużych uproszczeń badanego zdjęcia. Należy zauważyć, że implementacja sposobu analizy obrazu zależy od przeznaczenia czy roli jaką będzie spełniał, przykładowo algorytm na rozpoznawanie twarzy może być zupełnie inny od rozpoznawania tekstu. Algorytmów o zastosowaniu ogólnym jest stosunkowo niewiele w porównaniu do liczby metod przetwarzania obrazu.

Kilka przykładowych metod opierających się na dzieleniu informacji to: segmentacja przez podział obszaru, rozrost obszaru czy wykrycie krawędzi. W metodzie przez podział obrazu porównywana jest z ustaloną wartością progową, wybieraną przeważnie na podstawie histogramu, wartość jasność każdego elementu znajdującego się na fotografii, a następnie dzieli się je na te o przekroczonej wartości i nie przekroczonej. W metodzie krawędziowej najpierw są wyszukiwane różnice w jasności pomiędzy obszarami. W tym celu stosuje się operator gradientu, a na otrzymanym gradiencie wykonuje się operację progowania. Następnie odrzucane są krawędzie, które nie tworzą bryły zamkniętej. W segmentacji przez rozrost obszaru nie poszukuje się różnic między wartościami obiektów na obrazie, lecz obszarów, w których grupa elementów ma podobną jasność. W ten sposób z pojedynczego piksela obraz rozrasta się poprzez sprawdzanie kolejnych sąsiednich pikseli, ale istnieje metoda, która na samym początku dzieli obraz na początkowe obszary i obliczenia są wykonywane do momentu otrzymania pełnej jednolitości obszarów [2].

Istnieją również sposoby czysto obliczeniowe do analizy obrazu. Taką metodą – jedną z prostszych – jest wyznaczenie pola powierzchni wykrytego obiektu, które może posłużyć jako wstępna metoda weryfikacji w algorytmie i jeśli obiekt przejdzie ją pozytywnie to może być sprawdzany dalej przez bardziej skomplikowane algorytmy. Jest możliwość również obliczania długości krawędzi badanego elementu do wstępnego rozpoznania obrazu, jednak jest to trudniejsze w realizacji ze względu na trudność w przybliżeniu ciągłej krzywej przez dyskretną kombinację pikseli danego zdjęcia.

Mocno rozwijającymi się technologiami analizy obrazu są te oparte o sztuczną inteligencję i głębokie uczenie maszynowe w sieciach neuronowych. Rozpoznawanie danych może być oparte o wyszukiwanie podobnych zapytań w chmurze, dlatego takie algorytmy są bardzo dobre do analizy generycznych obiektów na zdjęciach, takich jak samochody, rowery czy psy. Ciągły rozwój sztucznej inteligencji może doprowadzić do sytuacji, w której będzie istnieć jeden ogólny sposób rozpoznawania obiektów, dzięki możliwości samouczenia się i dostępu do milionów danych w Internecie.

3 PROJEKT – WARSTWA SPRZĘTOWA

3.1 Założenia techniczne.

Celem pracy jest projekt urządzenia do przechwytywania i analizy otrzymanego obrazu z modułu obiektywu. Zdecydowano się na model Raspberry Pi 3B wraz z wyświetlaczem dotykowym LCD, który posłuży do wyświetlania interfejsu użytkownika. Taki wybór jest podyktowany faktem, iż jest to, obok Arduino, jedna z najpopularniejszych platform wykorzystywanych do programowania prostych aplikacji i urządzeń. Programy na tę platformę najczęściej wykonuję się w języku programowania Python na system operacyjny Linux, jednak w tym przypadku zastosowano system operacyjny Windows 10 IoT Core i język programowania C# (.NET) opracowane przez firmę Microsoft.

Jako urządzenie do przechwytywania obrazu wybrano kamerę LifeCam HD-5000 firmy Microsoft, która jest przymocowana na stałe do ramy na wysokości 106 mm, co pozwala na badanie obiektów o maksymalnych wymiarach około 12x7,5 cm. Komunikuje się ona z Raspberry Pi za pomocą interfejsu USB. Aplikację przygotowano do badań płytek drukowanych typu Arduino. Najważniejszym celem jest przygotowanie możliwie jednakowych warunków testowych. W tym celu doświetlono scenerię za pomocą dwóch lampek LED podłączonych do gniazda USB o mocy 1,2 W, a także zamontowano w odpowiednich miejscach w ramie kołki, aby uniemożliwić niepożądany obrót płytki i ustalić dokładne miejsce do przeprowadzania prób. W interfejsie użytkownika można wybrać pomiędzy dwoma przygotowanymi wcześniej modelami (oryginalne Arduino UNO Rev. 3 i chiński odpowiednik płytki UNO), a także zapisać do pamięci własne zdjęcie wzorcowe płytki, które może posłużyć do późniejszego porównywania.

Dodatkowo użytkownik ma możliwość ustalenia tolerancji oraz progu zdania testu, a także wyświetlenia szczegółowych wyników oraz przetworzonych zdjęć. Ostatecznie przygotowano 3 algorytmy przetwarzające obraz oraz jeden do ich porównywania.

Projekt może zostać rozszerzony do pełnego modelu pokazowego do rozpoznawania obrazu jak w zakładach produkcji elementów elektronicznych poprzez zastosowanie taśmociągu i czujników zbliżeniowych.

3.2 Komputery jednopłytkowe – porównanie wybranych rozwiązań.

Znalezienie platformy spełniającej wymagane kryteria nie jest trudnym zadaniem. Na rynku istnieje wiele firm, które produkują komputery typu SBC. Znaleźć można konstrukcje zarówno bardzo kompaktowe, oferujące mniejszą ilość GPIO i złącz komunikacyjnych, ale też takie posiadające relatywnie dużą moc obliczeniową w swojej klasie lub całe zestawy startowe, posiadające wiele modułów rozszerzeń. Projekt z założenia zakłada użycie platformy z rodziny Raspberry Pi, ale opisano również krótko rozwiązania konkurencyjne. Ma to na celu zobrazowanie obecnego rynku komputerów jednopłytkowych oraz pokazanie różnych podejść przez różnych producentów.

Jednym z nowszych zaprezentowanych rozwiązań jest platforma NVIDIA Jetson, a konkretnie model Nano. Płytka ta jest głównie ukierunkowana na projekty związane z robotyką

samosterowaną, a także do aplikacji związanych ze sztuczną inteligencją i sieciami neuronowymi za sprawą dedykowanych modułów obliczeniowych AI typu SoM (*System on Module*), dzięki czemu moc obliczeniowa tego minikomputera może wzrosnąć nawet do 472 GFLOPS. Sercem układu jest czterordzeniowy procesor NVIDIA Tegra X1 (ARM Cortex A57) o taktowaniu 1,43 GHz oraz wydajny układ graficzny NVIDIA Maxwell z 128 rdzeniami CUDA o taktowaniu 921 MHz [6]. Poza tym Jetson Nano posiada 40 interfejsów GPIO (w tym SDIO, SPI, I^2C i I^2S). Platforma działa w oparciu o autorski system Linux oraz przygotowano rozbudowane biblioteki i narzędzie SDK. Charakterystyczną cechą tej platformy jest możliwość wyboru trybu pracy między ekonomicznym, w którymi pobór mocy wynosi ok. 5 W, oraz wydajnościowym, w którym zwiększa częstotliwość zegara procesora i układu graficznego, przez co pobór mocy wzrasta do ok. 10 W. Ze względu na dużą moc obliczeniową i możliwości wykorzystania nowoczesnych technologii płytka od NVIDII nie jest tanią propozycją, ponieważ jest wyceniona na około 500 zł.

Innym popularnym producentem komputerów jednopłytkowych jest AAEON, który w swojej ofercie posiada platformę UP. Najciekawszym modelem jest UP Board, który również kosztuje około 500 zł. Podobnie jak NVIDIA, firma AAEON swoje produkty przede wszystkim oferuje dla rozwiązań z zakresu automatyki i robotyki oraz sztucznej inteligencji, gdzie posiada dedykowane moduły obliczeniowe od firmy Intel. Zasadniczą różnicą jest rodzaj wykorzystanego procesora. W Raspberry Pi oraz Jetson zastosowano procesory z rdzeniem ARM, natomiast w UP Board wykorzystany jest procesor o architekturze x86, a konkretnie czterordzeniowy Intel Atom x5-z8350 o taktowaniu 1,92 GHz [7]. Z tego względu możliwe na tej płytce jest zainstalowanie każdej wersji Windowsa 10, a także większości dystrybucji Linuxa (w tym Ubuntu i Debian). Minikomputer posiada pełnoprawny układ graficzny Intel HD 400, więc możliwe jest zainstalowanie API graficznych, takich jak DirectX 12 i OpenGL 4.2. W przeciwieństwie do Raspberry Pi posiada wbudowaną pamięć eMMC, na której można zainstalować system operacyjny. Za obsługę interfejsu GPIO (w tym SDIO, SPI, I^2C i I^2S) odpowiada mikrokontroler Altera Max V wraz z 8-bitowym ADC. Pobór mocy jest relatywnie duży (około 20 W), głównie przez wykorzystanie procesora x86, które są znacznie mniej energooszczędne od procesorów ARM.

Najpopularniejszą platformą komputerów typu SBC jest Raspberry Pi. Platforma ta zapewnia kompleksowe rozwiązania pod względem programowym i sprzętowym. Dodatkowo posiada bardzo dużą bazę użytkowników oraz bibliotek, dzięki którym jest możliwość rozszerzenia możliwości układu o nowe funkcje. Zgodnie z założeniami projektowymi należy wybrać platformę z rodziny Raspberry Pi.

Istnieje kilka wersji płytek różniących się od siebie parametrami, dostępnymi wejściami/wyjściami oraz mocą obliczeniową, ale zdecydowano się na model 3B, głównie ze względu na kompatybilność z system operacyjnym Windows 10 IoT Core, na którym oparto działania całego urządzenia.

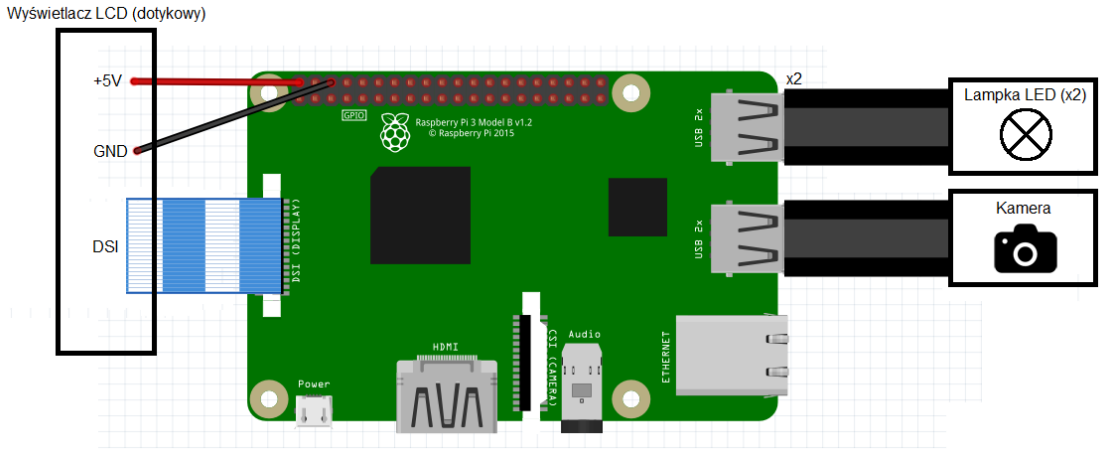
Sercem układu jest SoC firmy Broadcom BCM2837 posiadający czterordzeniowy procesor ARM Cortex-A53 o taktowaniu 1,2 GHz oraz 1 GB pamięci SDRAM [8]. Na wyposażeniu znajdują się 4 porty USB 2.0, jedno gniazdo HDMI i 3,5 mm jack, a także porty do podłączenia

wyświetlaczy (DSI) oraz kamer (CSI). Oprócz tego Raspberry Pi posiada 40 złącz GPIO (w tym UART, SPI i I^2C). Do komunikacji sieciowej można wykorzystać port Ethernet, moduł WiFi oraz Bluetooth 4.1. Niewątpliwą zaletą tej płytki jest jej cena, która wynosi około 150 zł i z tego względu do projektu wybrano to rozwiązanie, ponieważ nie wymaga on dużej mocy obliczeniowej, gdyż najwięcej zasobów będzie wykorzystywanych na przesyłanie obrazu z kamery. Ta platforma posiada dedykowaną dystrybucję systemu operacyjnego Linux o nazwie Raspbian, będącą zmodyfikowaną wersją dystrybucji Debian. Ponadto system jest zaopatrzony w kompilator języka programowania Python wersji 3.x. Dodatkowo jest możliwość zainstalowania innych systemów operacyjnych, takich jak Ubuntu MATE, Android lub wspominany wcześniej Windows 10 IoT Core, który umożliwia kompilowania programów stworzonych na platformie .NET w językach programowania Visual Basic, XAML, C++, C# lub JavaScript [9]. Wadą tego rozwiązania jest brak zainstalowanej pamięci trwałej na laminacie płytki (Raspberry Pi posiada tylko slot karty SD do przechowywania plików) i wszystkie dane (w tym system operacyjny) muszą być zapisywane na karcie SD, która jest mniej trwała i bardziej wrażliwa na utratę danych. Wszystkie urządzenia z tej platformy posiadają bardzo słaby procesor graficzny, który uniemożliwia instalowanie zaawansowanych API graficznych, takich jak DirectX od Microsoftu. Widok omawianej płytki jest przedstawiony na rysunku 3.1.

Rys. 3.1. Widok minikomputera Raspberry Pi 3B.

3.3 Projekt układu.

Praca inżynierska zakłada zaprojektowanie układu umożliwiającego weryfikację płytek drukowanych za pomocą kamery. Układ ponadto pozwala użytkownikowi samemu wybrać tolerancję i próg zdania testu, a także wyświetla w aplikacji szczegółowy raport dotyczący wyniku próby. W tym celu zainstalowano dotykowy wyświetlacz LCD. Dwie lampki LED dodatkowo doświetlają obszar testowy, aby spróbować stworzyć jednakowe warunki oświetleniowe. Na rysunku 3.2 przedstawiono poglądowy schemat układu.



Rys. 3.2. Schemat poglądowy układu

Głównym elementem urządzenia jest minikomputer Raspberry Pi 3B. Zasilanie może odbywać się na dwa różne sposoby. Pierwszy sposób odbywa się przy pomocy złącza MicroUSB przy napięciu 5 V. Jest to opcja najprostsza i najczęściej wykorzystywana. Drugi sposób wykorzystuje złącze GPIO „5V" do zasilenia układu napięciem 5 V. W nowszych wersjach płytek, tj. 3B+ oraz 4B, można zasilić płytkę przez PoE (ang. *Power over Ethernet*) za pomocą specjalnej nakładki. W tym projekcie zdecydowano się wykorzystać złącze MicroUSB i podłączyć przewód do zasilacza 3,1 A.

Kolejną istotną częścią projektu jest dotykowy wyświetlacz LCD. Zdecydowano, iż zostanie wykorzystany oficjalny wyświetlacz stworzony przez Raspberry Pi Foundation – Raspberry Pi 7" Touchscreen Display [10]. Posiada on 7-calowy ekran dotykowy (pojemnościowy) wykonany w technologii LCD. Maksymalna rozdzielczość wynosi 800x480 pikseli, a częstotliwość odświeżania wynosi 60 Hz. Na rynku dostępne są również kompaktowe wyświetlacze wykorzystujące złącze HDMI, o innych przekątnych i rozdzielczości, a także z rezystancyjnymi ekranami dotykowymi. Zestaw składa się z panelu LCD, płytki-adaptera, przewodów oraz elementów montażowych. Również tutaj zasilanie może zostać doprowadzone dwoma różnymi metodami. Pierwszą metodą jest podłączenie przewodów pomiędzy złączami „5V" i „GND" adaptera i minikomputera. Opcjonalną drugą metodą jest podłączenie przy pomocy złącza na płytce-adapterze MicroUSB przy napięciu 5 V. Ekran LCD jest podłączany do ekranu adaptera za pomocą dwóch przewodów taśmowych. Komunikacja z Raspberry Pi 3B może następować poprzez port DSI lub przez magistralę I^2C, ale w tym celu należy podłączyć przewody pomiędzy złącza „SDA" (ang. *Serial Data Line*) i „SCL" (ang. *Serial Clock Line*) adaptera i Raspberry Pi. Wadą tego rozwiązanie jest przekazanie pełnej kontroli nad tą linią magistrali dla procesora graficznego. Na potrzeby tego projektu skorzystano z rozwiązania zasilenia adaptera z minikomputera, natomiast do komunikacji wykorzystano port DSI. Podglądowy widok takiego połączenia wyświetlacza z Raspberry Pi pokazano na rysunku 3.3.

Rys. 3.3. Widok ekranu dotykowego z adapterem połączonego z Raspberry Pi.

Do zwiększenia wytrzymałości mechanicznej i zabezpieczenia urządzeń przed czynnikami zewnętrznymi wykorzystano obudowę ASM-1900035-21. Ta obudowa jest przeznaczona konkretnie do spasowania oryginalnego wyświetlacza LCD z płytką Raspberry Pi. Posiada wycięcia na wszystkie dostępne porty, zdejmowaną tylną klapkę w celu umożliwienia dostępu do złącz GPIO, a także miejsce na dedykowaną kamerę. Poglądowy widok wyświetlacza, adaptera i minikomputera pokazano na rysunku 3.4.

Rys. 3.4. Raspberry Pi oraz wyświetlacz w obudowie ASM-1900035-21.

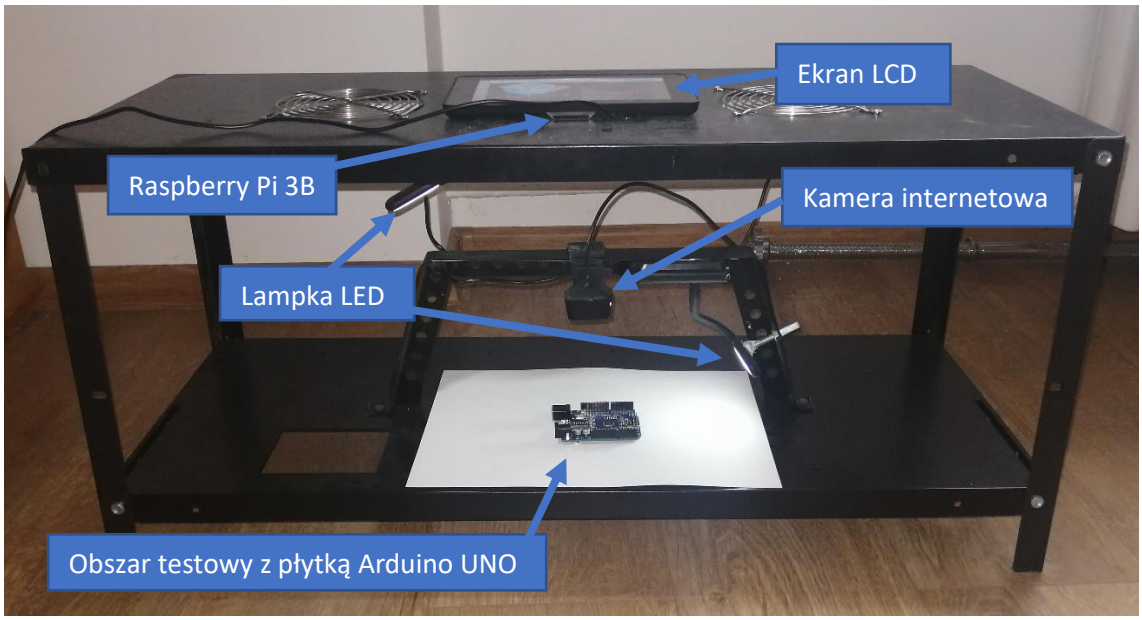
Do przechwytywania obrazu wykorzystano kamerę internetową LifeCam HD-5000 od firmy Microsoft. Posiada ona matrycę wykonaną w technologii CMOS i pozwala na wykonywanie zdjęć maksymalnie w rozdzielczości 1280x800 pikseli [11]. Do komunikacji i zasilania wykorzystuje interfejs USB 2.0. Wybrano takie rozwiązanie głównie ze względu na niską cenę (ok. 50 zł) oraz na wbudowaną nóżkę, którą łatwo można zamontować na ramie. Poglądowe zdjęcie kamery internetowej pokazano na rysunku 3.5.

Rys. 3.5. Kamera internetowa Microsoft LifeCam HD-5000.

Do doświetlenia scenerii wykorzystano dwie lampki LED, które zostały zasilane z dwóch portów USB płytki Raspberry Pi. Dzięki zastosowaniu LEDów do obiektywu kamery wpada więcej światła przez co obraz staje jaśniejszy i jest mniej szumów. Poglądowe zdjęcie zainstalowanych lampek pokazano na rysunku 3.6.

Rys. 3.6. Lampka LED podłączana do portu USB.

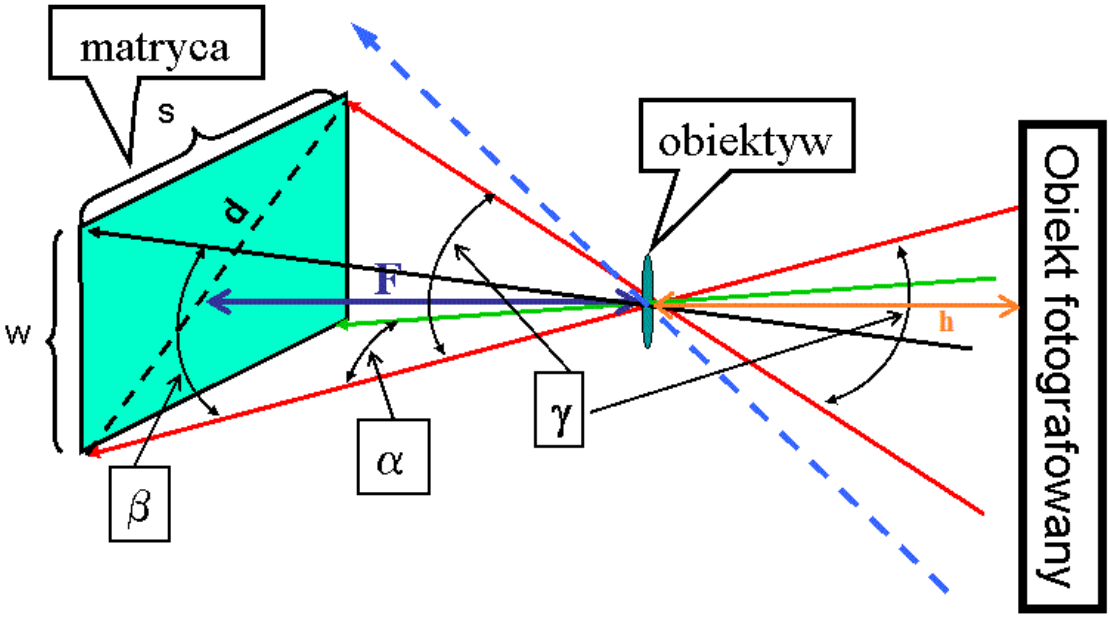
Wszystkie powyższe elementy zamontowano na stalowej ramie. Wyświetlacz zainstalowano na górze, aby użytkownik miał łatwy dostęp do niego. Pod nim znajduje się kamera internetowa zamontowana na ramie tak, aby była na wysokości 106 mm od obszaru badanego. W obszarze badanym występują wystające kołki do umieszczania płytek Arduino, a cały obszar pokryto matową białą kartką papieru, ponieważ tworzy lepsze warunki oświetleniowe od czarnej błyszczącej powierzchni ramy. Widok stanowiska badawczego przedstawiono na rysunku 3.7.



Rys. 3.7. Widok całego urządzenia do analizy wizyjnej płytek drukowanych.

3.4 Dokładność pomiarowa urządzenia.

Istotnym parametrem określającym urządzenie do rozpoznawania obrazu jest dokładność z jaką może wykonywać algorytmy. Wielkościami opisującymi tą dokładność są: przekątna matrycy światłoczułej, ogniskowa obiektywu, rozdzielczość wykonywanego zdjęcia oraz wysokość od obiektywu do obiektu badanego. Poniżej przedstawiono poglądowy schemat kamery (rys. 3.8), wzory do obliczenia kątów widzenia kamery (3.1-3), wzory na obszar obejmujący przez podgląd kamery (3.4-5), a także wzór określający dokładność w pikselach na mm^2 projektowanego urządzenia (3.6). Z obliczeń wynika, że w wykonywanym przez urządzenie zdjęciu, na jeden milimetr przypada około 10,5 pikseli:



Rys. 3.8. Schemat poglądowy

$$\alpha = 2 * arctg\left(\frac{s}{2 * F}\right) = 2 * arctg\left(\frac{3{,}2}{2 * 2{,}8}\right) = 59{,}49° \quad (3.1)$$

$$\beta = 2 * arctg\left(\frac{w}{2 * F}\right) = 2 * arctg\left(\frac{2}{2 * 2{,}8}\right) = 39{,}31° \quad (3.2)$$

$$\gamma = 2 * arctg\left(\frac{d}{2 * F}\right) = 2 * arctg\left(\frac{3{,}7}{2 * 2{,}8}\right) = 66{,}91° \quad (3.3)$$

$$x = 2 * h * tg\left(\frac{\beta}{2}\right) = 2 * 106 * tg\left(\frac{39{,}31°}{2}\right) = 75{,}72\ [mm] \quad (3.4)$$

$$y = 2 * h * tg\left(\frac{\alpha}{2}\right) = 2 * 106 * tg\left(\frac{59{,}49°}{2}\right) = 121{,}14\ [mm] \quad (3.5)$$

$$k = \frac{a * b}{x * y} = \frac{1280 * 800}{121{,}14 * 75{,}72} = 111{,}63\ \left[\frac{px}{mm^2}\right] \quad (3.6)$$

gdzie:

- s, w, d – odpowiednio szerokość, wysokość i przekątna matrycy kamery HD-5000;
- F – ogniskowa obiektywu;
- α, β, γ – kąt widzenia obiektywu odpowiednio horyzontalny, wertykalny i ukośny;
- x, y – odpowiednio długość i szerokość jaką obejmuję podgląd kamery;
- a, b – rozdzielczość wykonywanego zdjęcia odpowiednio pionowa i pozioma;
- k – współczynnik dokładności wykonywanego zdjęcia.

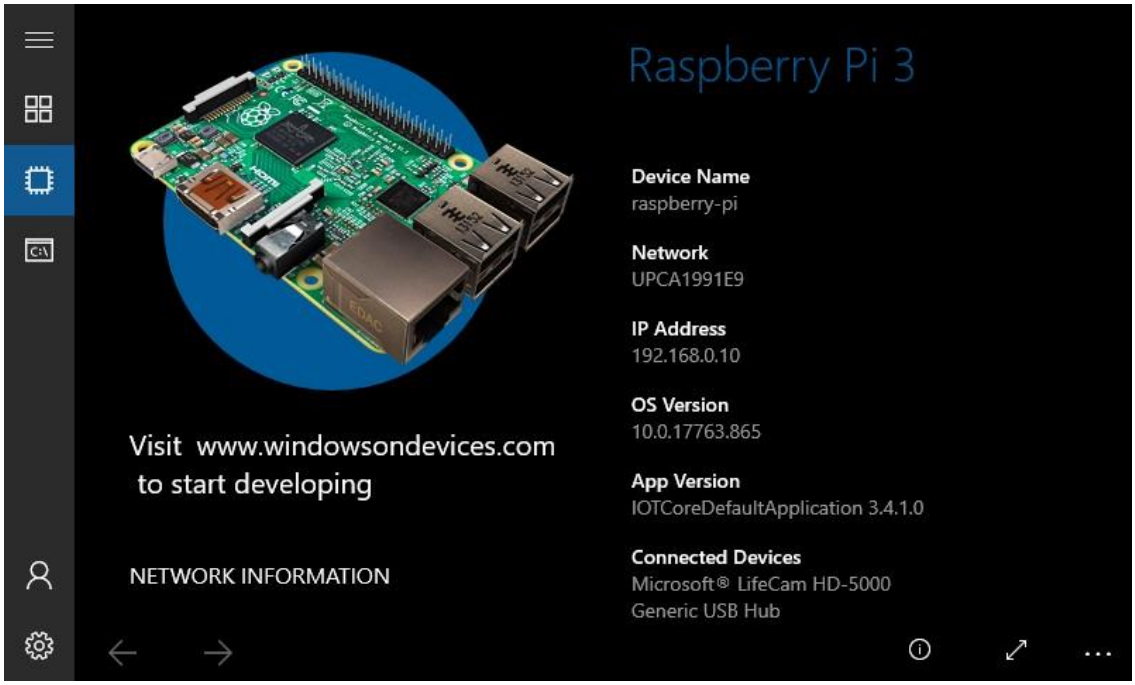
Dane na temat ogniskowej i przekątnej matrycy OmniVision OV9712-1D, wykorzystanej w kamerze internetowej Microsoft LifeCam HD-5000 zaczerpnięto z karty katalogowej [11].

4 PROJEKT – WARSTWA PROGRAMOWA

4.1 System operacyjny.

Firma Microsoft posiada szeroką gamę systemów operacyjnych, programów, środowisk graficznych, interfejsów, narzędzi deweloperskich, języków programowania czy bibliotek. Wśród nich jest OS pod nazwą Windows 10 IoT Core przeznaczony przede wszystkim do komputerów jednopłytkowych. Oficjalnie wspiera między innymi minikomputery z rodziny Raspberry Pi, Qualcomm DragonBoard, Intel MinnowBoard, NXP i.MX czy AAEON Up [12].

Zdecydowano się użyć wyżej wymienionego systemu operacyjnego w obecnie najnowszej dostępnej wersji (17763), który został zainstalowany na karcie SD. Jest to minimalistyczna wersja desktopowego Windowsa 10, różniąca się przede wszystkim brakiem pulpitu i eksploratora plików oraz koniecznością instalacji programów stworzonych w interfejsie UWP (w wersji dla procesorów z rdzeniem ARM). Widok strony głównej systemu Windows 10 IoT Core wygląda jak na rysunku 4.1.



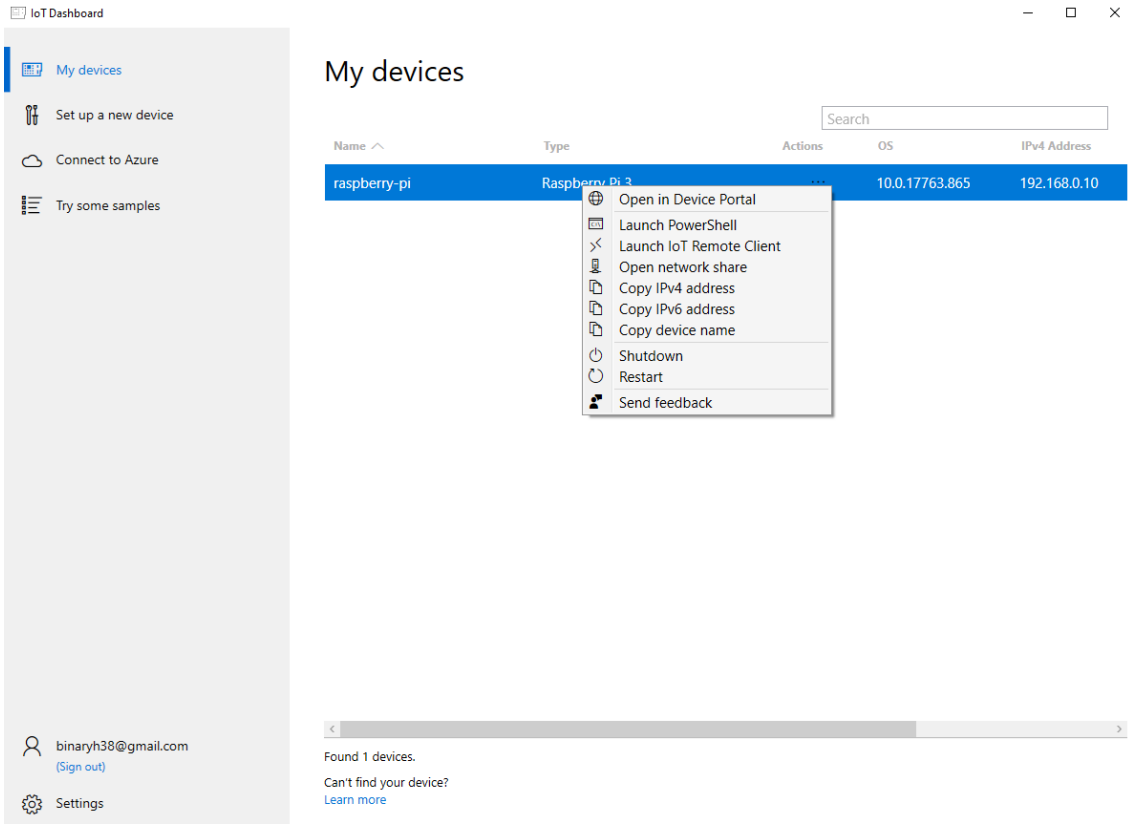
Rys. 4.1. Domyślna aplikacja systemu Windows 10 IoT Core.

System nie posiada zbyt wiele preinstalowanych aplikacji. Wśród nich można znaleźć przeglądarkę internetową, przeglądarkę zdjęć, odtwarzacz muzyki, pogodynkę, szkicownik, konsolę poleceń, program aktualizacyjny i ustawienia. Do nawigacji w systemie może posłużyć ekran dotykowy (OS posiada w budowaną klawiaturę ekranową) lub standardowa klawiatura i myszka.

Zasadniczą różnicą w działaniu tego systemu operacyjnego od innych jest podział aplikacji na „Background” i „Foreground”. Aplikacje typu „Background” są to podstawowe funkcje działające w tle, takie jak na przykład WiFi czy host odpowiadający za pamięć karty SD, a także usługi połączenia z chmurą, natomiast typu „Foreground” to programy wyświetlane bezpośrednio na ekranie, czyli te uruchomione przez użytkownika. W przeciwieństwie do systemów Windows

10 i Linux nie można mieć otwartych w procesie dwóch aplikacji „Foreground” i nie jest możliwym przełączanie się w tle między nimi. Jeżeli program zostanie zamknięty, zostanie wyczyszczony z pamięci procesu, inaczej pisząc, nie ma funkcji minimalizacji aplikacji.

Firma Microsoft przygotowała kilka programów, ułatwiających zarządzanie urządzeniami opartymi na ich rozwiązaniach. Pierwszą taką aplikacją jest Windows IoT Dashboard, za pomocą której można sterować urządzeniami wyposażonymi w system Windows IoT. W zakładce „My devices” (rys. 4.2) można odnaleźć urządzenia podłączone do tej samej sieci i uruchomić kilka funkcji, takich jak na przykład włączenie konsoli poleceń Powershell na komputerze stacjonarnym i uzyskanie dostępu do zaawansowanych ustawień urządzenia, włączenie klienta do przekazywania obrazu z płytki i sterowania z poziomu klawiatury i myszki komputera (niedostępne na Raspberry Pi 3B ze względu na brak sterowników do wbudowanej karty graficznej), włączenie możliwości wymiany plików za pomocą udostępniania sieciowego, a także wyłączanie i resetowania urządzenia.

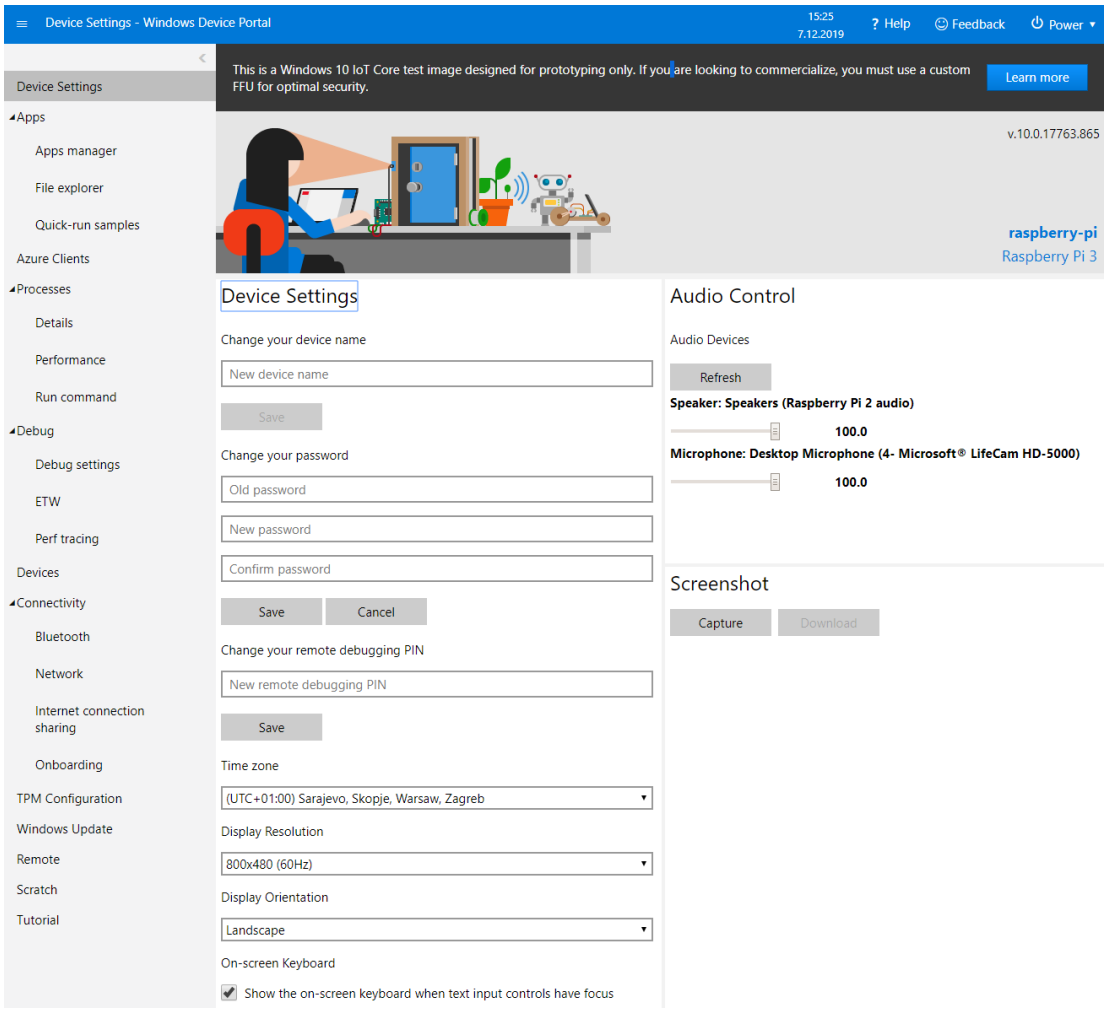


Rys. 4.2. Widok okna „My devices”.

Kolejna zakładka programu to „Set up a new device”. Jak sama nazwa wskazuje, służy ona do pierwszej konfiguracji urządzenia. W niej można znaleźć opcje do wybrania typu urządzenia i wersji systemu jaką chcemy pobrać i zainstalować na karcie SD, która później należy włożyć do slotu minikomputera. Dodatkowymi możliwościami są możliwość zainstalowania niestandardowego obrazu systemu operacyjnego lub skonfigurowanie połączenia WiFi jeszcze przed instalacją.

Ostatnia zakładka, która zostanie opisana to „Connect to Azure”. Za jej pomocą możemy podłączyć urządzenie do serwisu Microsoft Azure. Jest to płatna platforma posiadająca zbiór usług do wykonywania zadań obliczeniowych hostowanych aplikacji oraz tworzenia i zarządzania zasobami w chmurze [13]. Producent oferuje szereg różnych funkcji, takich jak na przykład tworzenie wirtualnych maszyn lub wypożyczanie serwerów od Microsoftu, przechowywanie danych w chmurze, zarzadzanie bazami danych typu SQL, tworzenie i sterowanie aplikacjami w chmurze, implementacja uczenia maszynowego dla rozwiązań użytkownika, a także zarządzanie urządzeniami Internetu rzeczy. Jeżeli płytka zostanie podłączona do tego serwisu to może odbierać dane, na przykład z czujników umieszczonych w całym budynku, podłączonych do jednej sieci WiFi, aby na ich podstawie uruchamiać określone funkcje w przygotowanym programie. Takim przykładem jest możliwość sterowania kominkiem w domku jednorodzinnym za pomocą czujników czadu, wilgotności czy temperatury umieszczonych w różnych jego sekcjach bez konieczności podłączenia ich przewodami do urządzenia.

Innym podobnym programem do zarządzania urządzeniem poprzez sieć bezprzewodową jest „Windows Device Portal”. Aby uzyskać do niego dostęp należy wpisać w przeglądarkę internetową adres IP urządzenia podłączonego do tej samej sieci i zalogować się ustawionym podczas instalacji hasłem. Za jego pomocą można kontrolować urządzenie w bardziej zaawansowany sposób niż przez Windows IoT Dashboard. Stroną startową jest „Device Settings”, gdzie jest możliwość ustawienia podstawowych parametrów urządzenia czy zrobić zrzut ekranu (rys. 4.3).



Rys. 4.3. Strona startowa Windows Device Portal.

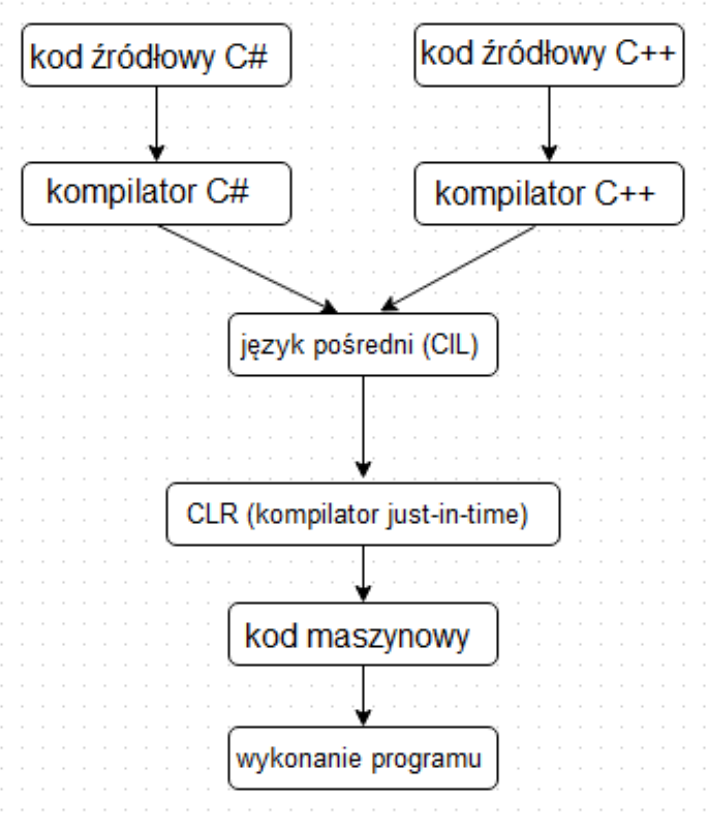
Przydatną funkcją tej aplikacji jest opcja do podglądu, przesłania lub pobrania plików znajdującej się na urządzeniu poprzez zakładkę „File explorer”. Należy zauważyć, że system operacyjny Windows 10 IoT Core nie posiada eksploratora plików i jedyną wbudowaną możliwością manipulacji plików jest właśnie poprzez ten moduł. Innymi ciekawymi funkcjami są „Apps manager”, który pozwala na zdalne instalowanie i odinstalowywanie aplikacji, a także „Performance”, w którym jest możliwość podglądu używanych procesów oraz monitorowania zużycia procesora i pamięci operacyjnej. W trybie bezczynności Raspberry Pi zużywała około połowy z dostępnego 1 GB pamięci RAM na potrzeby systemowe.

Udostępniony przez firmę Microsoft obraz Windows 10 IoT Core dla poszczególnych minikomputerów jest dostępny tylko do zastosowań hobbystycznych i edukacyjnych. Aby skomercjalizować projekt należy stworzyć własny obraz systemu operacyjnego, który pozbawiony jest rozwiązań chronionych prawami autorskimi [12]. Istnieje kilka takich produktów gotowych do sprzedaży opartych o ten system operacyjny. Jednym z takich projektów jest robot „Misty II” stworzony przez Misty Robotics, który może uruchamiać lokalnie uczenie maszynowe w głębokiej sieci neuronowej [14]. Innym przykładem jest termostat GLAS od firmy Johnson Controls, który dodatkowo wykorzystuje inne technologie od Microsoft, takie jak na przykład bazę danych Azure Cosmos, usługę IoT Hub czy zapisywanie danych w chmurze [15].

4.2 Oprogramowanie.

Możliwość wyboru środowiska programistycznego zostało ograniczone, z powodu wykorzystania systemu operacyjnego od Microsoftu na płytce Raspberry Pi. Linux, w przeciwieństwie do Windows 10 IoT Core, pozwala na pisanie programów w praktycznie każdym możliwym języku programowania, jak na przykład Python, C, Java czy JavaScript. Jednak zastosowanie frameworku (środowiska programistycznego) od Microsoftu pozwala na łatwe sprzężenie kodu programu z różnymi innymi usługami od tej firmy.

Zastosowane środowisko programistyczne to .NET Framework w wersji 4.7.x. Obsługuje ono języki programowania stworzone przez Microsoft: C#, C++/CLI, Visual Basic, F#, J#, JScript.NET (bazujący na JavaScript); a także inne języki jak Delphi.NET, Fortran, Perl czy Python. Sama platforma składa się ze środowiska uruchomieniowego CLR (ang. *Common Language Runtime*), języka pośredniego CIL (ang. *Common Intermediate Language*), kompilatora JIT (ang. *Just-In-Time*), standardowych bibliotek klas oraz kompilatorów dla poszczególnych języków wysokiego poziomu (w standardzie znajdują się kompilatory dla języków stworzonych przez Microsoft). Zaletą tego frameworku jest to, że można w projekcie programu łączyć ze sobą kody źródłowe innych języków programowania, ponieważ po skomplikowaniu przez odpowiedni dla języka kompilator, a następnie przetworzony na pośredni język programowania, który jest odpowiednikiem assemblera w językach niższego poziomu. Kod wyrażony w CIL jest następnie wdrażany przez środowisko uruchomieniowe CLR, które składa się z zestawu standardowego typów danych oraz standardowego formatu metadanych, który opisuje te typy danych. Kolejnym krokiem jest kompilacja do kodu maszynowego określonego programu, z tą różnicą, że kompilacja metody klasy wykonuje się w momencie, gdy program pierwszy raz się do niej odwoła. Tworzony wtedy jest tymczasowy fragment kodu na czas ładowania konkretnego modułu, który następnie zostaje zastąpiony przez skompilowany kod. Taka forma kompilacji nazywa się kompilacją w locie (JIT). Powstały kod maszynowy, który jest zapisany w systemie binarnym, komputer może odczytać i wykonać znajdujące się w nim instrukcję [16]. Cały opisywany proces pokazano na rysunku 4.4.



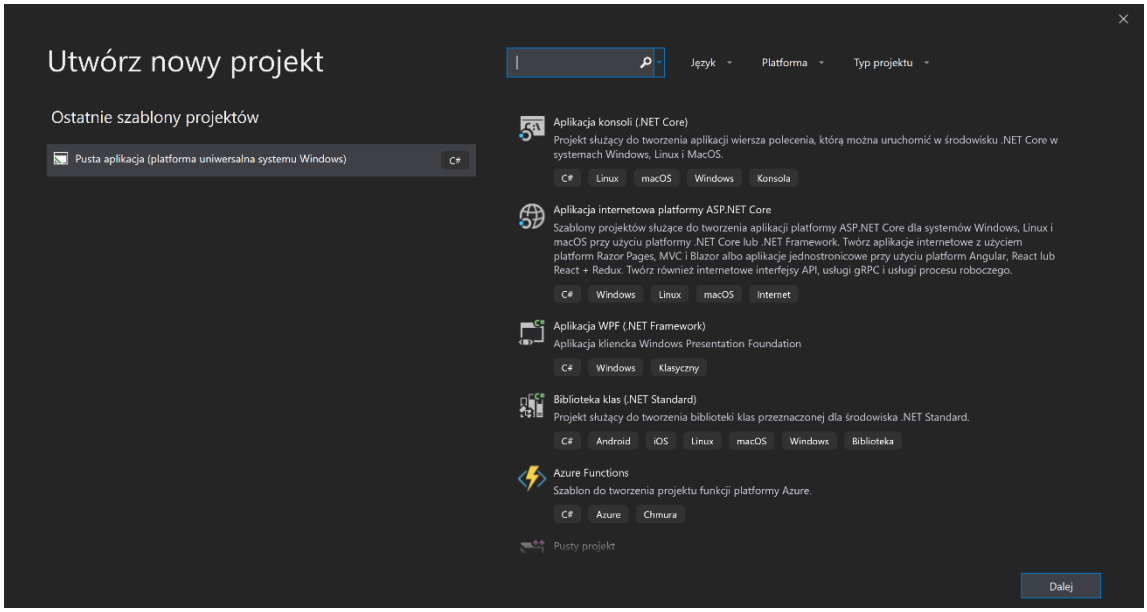
Rys. 4.4. Zasada kompilacji programu w .NET Framework.

Framework posiada wiele obsługiwanych w standardzie modeli aplikacji, takich jak na przykład Windows Forms, Windows Presentation Foundation (WPF) czy ASP.NET, który jest przeznaczony do projektowania stron internetowych. Dodatkowo istnieją rozszerzone implementacje tego środowiska programistycznego: Mono, w skład którego wchodzi Xamarin, za pomocą którego można tworzyć aplikacje na systemy Android oraz iOS; .NET Core, dzięki któremu jest możliwość budowania aplikacji konsolowych, ASP.NET Core oraz Universal Windows Platform (UWP).

Windows IoT Core wykorzystuje model aplikacji UWP, dlatego całe oprogramowanie stworzone do projektu wykorzystuje tą technologię. Jego niewątpliwą zaletą jest możliwość wdrażania tej samej aplikacji na różne systemy operacyjne wykorzystujące zarówno procesory x86 i ARM. Wspierane są: Windows 10 Desktop, Windows 10 on ARM, Windows 10 IoT, Windows 10 Mobile, Xbox One System Software oraz Windows Mixed Reality. Z tego powodu udostępniono paczkę instalacyjną przygotowanej aplikacji dla systemu Windows 10 wraz z instrukcją [dodatek A].

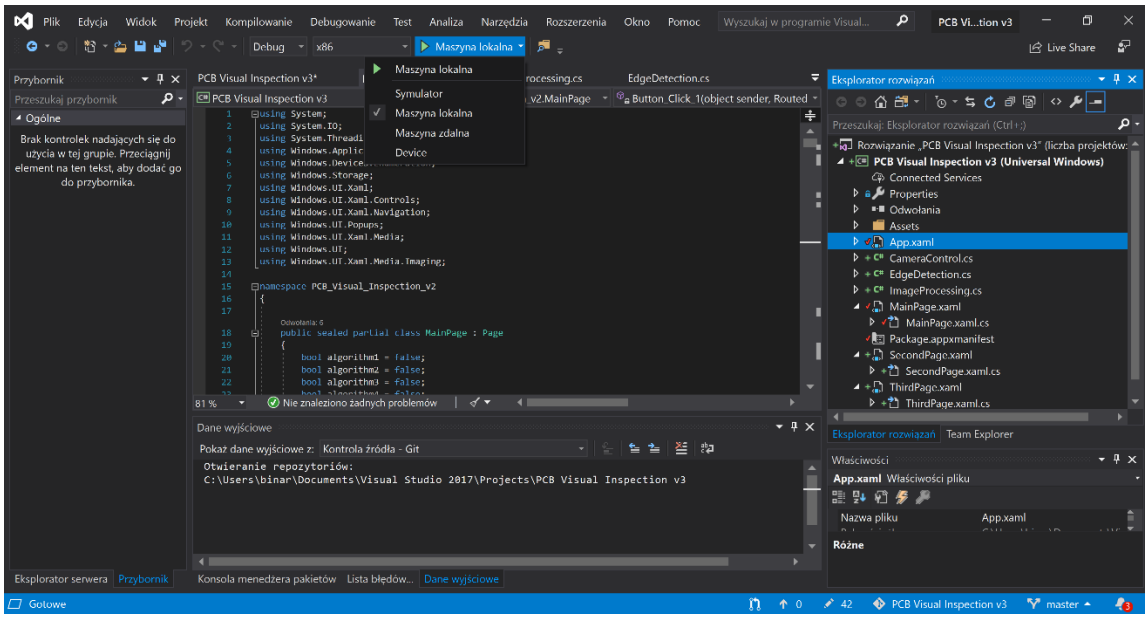
Programem wykorzystanym do tworzenia projektu i kompilacji programu jest Microsoft Visual Studio 2019 w wersji Community. Jest to zintegrowane środowisko programistyczne IDE (ang. *Integrated Development Enviroment*), za pomocą którego można pisać aplikacje na komputery, strony internetowe czy smartfony, w językach programowania obsługiwanych przez .NET Framework. To narzędzie pozwala na programowanie warstwy wizualnej aplikacji (tzw. „Frontend”) z użyciem XAML, HTML lub CSS, a także warstwy programowej (tzw. „Backend”).

Pierwszą stroną podczas otwierania programu jest wybór istniejącego projektu lub utworzenie nowego. W przypadku wybrania drugiej opcji pojawi się okno z wyborem dostępnych typów aplikacji do utworzenia (rys. 4.5). Przy każdym typie widnieje tag, który informuje jaki język programowania jest wykorzystany czy jakie technologie i platformy docelowe są wskazane. Za pomocą tych tagów można również szybko znaleźć w wyszukiwarce żądany typ projektu. W projekcie inżynierskim zastosowano szkic pustej aplikacji opartej o platformę uniwersalną systemu Windows w języku programowania C#.



Rys. 4.5. Okno tworzenia nowego projektu w Microsoft Visual Studio 2019 Community.

Okno główne zostało pokazane na rysunku 4.6. W środkowej części widnieje edytor tekstowy, a pod nim konsola debuggera. W prawej części jest „Eksplorator rozwiązań”, w którym znajdują się wszystkie użyte pliki i zasoby w danym projekcie. Zainstalowany jest również zdalny debugger, który pozwala na wdrażanie aplikacji do innych komputerów lub urządzeń IoT za pośrednictwem sieci bezprzewodowej. W górnej części programu widnieje opcja do wyboru architektury procesora dla kompilowanej aplikacji oraz docelowe urządzenie, na które ma być wgrany program (maszyna lokalna, maszyna zdalna lub symulator). Ważniejszymi zakładkami w eksploratorze rozwiązań są: „Properties”, gdzie można zmieniać ustawienia zaawansowana kompilatora i projektu, a także docelową wersję systemu Windows dla aplikacji. W zakładce „Package.appxmanifest” ustawia się nazwę aplikacji, jej opis, ikony w menu start, wymagane uprawnienia czy opcje tworzenia paczki instalacyjnej; „Odwołania”, w których, jak sama nazwa wskazuje, można dodawać odwołania do rozszerzeń zainstalowanych bibliotek oraz pobierać pakiety stworzone przez innych programistów, które rozszerzają funkcje platformy. W projekcie inżynierskim wykorzystano tylko domyślne pakiety stworzone i udostępnione przez firmę Microsoft. Dodatkowym atutem programu Visual Studio jest kompilator w czasie rzeczywistym, który na bieżąca sprawdza poprawność wpisywanego kodu oraz pokazuje podpowiedzi uzupełniające lub sugestie dotyczące zmiany niepoprawnej struktury wykorzystanej funkcji w programie.



Rys. 4.6. Główne okno programu Microsoft Visual Studio 2019 Community.

4.3 Koncepcja działania programu.

Program składa się z głównych funkcji. Pierwszą z nich jest odbieranie danych wysyłanych z kamery i zapis ich w postaci zdjęcia w pamięci lokalnej urządzenia. Drugim elementem jest przetworzenie tych zdjęć przez odpowiednie algorytmy. Ostatnia składowa aplikacji to porównanie wykonanej fotografii z obrazem wzorcowym danej płytki.

W tym celu stworzono trzy strony interfejsu programu głównego, gdzie każda strona składa się pliku *xaml.cs z kodem źródłowym w języku C# i pliku *.xaml z kodem w języku XAML do projektowania interfejsu użytkownika, a także trzy klasy. Najważniejszą stroną jest strona z podglądem kamery (MainPage.xaml.cs), na której wybiera się algorytmy, wykonuje zdjęcie oraz wyświetla wynik przeprowadzonej próby. Na kolejnej stronie (SecondPage.xaml.cs) znajdują się ustawienia parametrów testowych oraz wybór płytki do analizy wizyjnej. Raport z wykonanego testu wyświetlany jest na osobnej stronie (ThirdPage.xaml.cs), która wyświetla wszystkie zrobione i przetworzone zdjęcia oraz szczegółowy wynik procentowy dla konkretnego algorytmu. W klasach umieszczony jest zbiór metod do realizacji określonych zadań związanych z przetwarzaniem obrazu (ImageProcessing.cs), wykrywaniem krawędzi na zdjęciu (EdgeDetection.cs) oraz kontrolowaniem modułu kamery (CameraControl.cs).

Podczas załadowania strony głównej aplikacji pierwszy raz zostają stworzone nowe obiekty z wyżej wymienionych klas oraz klasy do przesyłania parametrów między stronami (ColorTolerance do obierania z ustawień, Pictures wysyła do raportu). Ich deklaracja w funkcji MainPage wygląda następująco:

```
ColorTolerance tolerance = new ColorTolerance();
Pictures images = new Pictures();
ImageProcessing ProcessImage = new ImageProcessing();
CameraControl Camera = new CameraControl();
EdgeDetection EdgeDetector = new EdgeDetection();
```

Po przejściu przez użykownika do podglądu kamery wywoływana zostaje metoda Initialize z klasy CameraCapture [dodatek B], do której przysyłana jest zmienna typu integer z żądanym kątem obrotu obrazu – 180 dla tego programu. Podgląd kamery wyświetla się poprzez przypisanie pola Preview do elementu wyświetlającego strumień wideo w kodzie XAML strony głównej. W tej klasie przygotowano jeszcze metody TakePicture, dzięki której można zapisać wykonane zdjęcie do pliku, a także Dispose, mająca zadanie przerwania przesyłania podglądu kamery w momencie, gdy nie jest to potrzebne.

Warto zauważyć, że wykorzystano programowanie obiektowe, które pozwala na tworzenie obiektów o tych samych parametrach jako różne od siebie instancje. Dla przykładu, jeżeli w projekcie zainstalowane byłyby dwie kamery, to można stworzyć dwa obiekty na bazie CameraControl o nazwie Camera1 i Camera2 do wyświetlania dwóch różnych podglądów. Zaletą takiego rozwiązania jest możliwość sterowania osobno pierwszą kamerą i drugą, na przykład do zrobienia zdjęcia wywołać można Camera1.TakePicture lub analogicznie postąpić dla Camera2. Poza tym, wiele metod wykorzystanych w aplikacji wykorzystuje programowanie asynchroniczne, które jest przydatne w momencie oczekiwania na urządzenie wejścia/wyjścia lub w trakcie wykonywania mocno obciążających procesor obliczeń [17]. Zaletą metody asynchronicznej jest to, że w momencie oczekiwania na dłuższą operację, jak na przykład pobieranie pliku z serwera, nie blokuje ona dalszego działania kodu i interfejsu użytkownika, ponieważ w przypadku metody synchronicznej program będzie czekać do momentu, aż żądaną informację uzyska. Funkcja asynchroniczna posiada wbudowaną zmienną informującą o tym, czy dane zadanie zostało wykonane, a następnie informuje program główny o możliwości otrzymania oczekiwanej zmiennej. Inaczej pisząc, jest możliwość, aby aplikacja wykonywała dwa lub więcej operacji naraz, pomimo że struktura programu dalej pozostaje jednowątkowa. Nie można w tym przypadku mylić programowania asynchronicznego od równoległego (współbieżnego), które do każdej nowej operacji tworzy nowy wątek.

W klasie ImageProcessing [dodatek C] zaimplementowano metody do zbierania danych o obrazie i ich przetwarzania. Łącznie napisano 11 metod, które są wykorzystywane wewnętrznie w tej klasie oraz w kodzie programu głównego.

Pierwszą opisaną metodą jest GetPixel. Służy ona do przetworzenia obrazu w ciąg bajtów zawierający informację o kolorach. W przesłanych zmiennych są też wartości domyśle dla scale i scaled_image_location, przyjmujące zadane wartości w momencie nie przesłania wartości w momencie ich wywołania w funkcji głównej. Te zmienne są wykorzystane w metodzie Resize do zmniejszenia rozmiaru zdjęcia. Jeden piksel jest równy czterem bajtom. Następnie ten ciąg bajtów zapisuje do osobnych tabeli dla odpowiedniego koloru.

Opcjonalnie można wykorzystać tablice trójwymiarowe do zapisu odpowiednich kolorów piksela, jednak w projekcie inżynierskim zdecydowano się na większą przejrzystość, poprzez wydzielenie tablic z nazwami barw.

Dla algorytmu drugiego (uproszczenie liczby kolorów), opisanego w podrozdziale 2.1, przygotowano metodę SetPixel. Przesłane do niej zostają tablice z kolorami pierwotnego zdjęcia oraz współczynnik upraszczający. Zmniejsza ona liczbę pikseli oraz zapisuje przetworzone kolory do odpowiednich tablic dwuwymiarowych. Następnym krokiem jest zamiana trzech tablic dwuwymiarowych z barwami na jednowymiarowy ciąg bajtów i zapisanie go do nowo utworzonego pliku.

Kolejną metodą użytą w klasie ImageProcessing jest algorytm do porównywania obrazów ComparePictures oraz odfiltrowywania tła. Do funkcji zostają przesłane dwuwymiarowe tablice kolorów dla dwóch zdjęć i także wartości tolerancji barwy. Jest to najbardziej złożony algorytm ze wszystkich przygotowanych. Na koniec funkcji wywoływane są metody HeatMap i ScoreMap, które tworzą nowe obrazy w oparciu o przesłane zmienne. HeatMap korzysta ze tablicy difference, w której są wartości bezwzględne różnicy między pikselami o tej samej pozycji, natomiast ScoreMap korzysta ze sprawdzonych pikseli pod kątem zgodności z ustawioną tolerancją i odfiltrowanym tłem. Dodatkowo stworzono funkcję TryToByte, która zwraca liczbę z przedziału od 0 do 255, a jeżeli przekroczy liczba przekroczy te granice, to zostaję zwrócona wartość maksymalna lub minimalna. Ostatnim krokiem jest przetworzenie tablic do ciągów bajtowych i utworzenie z nich nowego obrazu z odpowiednimi parametrami.

Metody SetPixel, HeatMap oraz ScoreMap korzystają z zapisu ciągów bajtów do pliku. Taką możliwość zapewnia funkcja SaveToFile, która zapisuje nowo utworzony plik do folderu „Obrazy” lub „Obrazy/Z aparatu” w zależności od wersji systemu Windows.

Ostatnią klasą przygotowaną na potrzeby aplikacji jest EdgeDetection [dodatek D], w której znajdują się gotowe macierze dla operatorów Sobela, Prewitta i Kirscha dla dwóch wymiarów (dwóch kierunków) oraz metoda ConvolutionFilter, tworząca obraz z wyróżnionymi krawędziami. Jest to programowa implementacja algorytmu trzeciego (krawędziowego) opisanego w podrozdziale 2.1. Głównym celem funkcji jest przekształcenie zdjęcia do odcieni szarości zgodnie z zaleceniem BT.709 Sektora Radiokomunikacji Międzynarodowego Związku Telekomunikacyjnego ITU-R (ang. *International Telecomunication Union – Radiocommunication Sector*) [18], zastosowano korekcję ważoną barw, ze względu na inne postrzeganie barw przez ludzkie oko. Z tego powodu najbardziej ważonym kolorem w algorytmie jest zielony z przelicznikiem 71,52%, natomiast kolor czerwony ma wagę 21,26%, a niebieski 7,22%. Kolejnym krokiem jest nałożenie maski danego operatora dla obraz źródłowy i znalezienie pikseli, w których występuje największa różnica wartości w danych barwach, a następnie zapisanie ciągu bajtowego do nowego obrazu.

Wszystkie powyższe metody zostają wywołane w programie głównym w momencie, gdy użytkownik naciśnie przycisk „Wykonaj test” [dodatek E]. Pierwszym krokiem jest odczyt parametrów przesyłanych ze strony „Ustawienia”, dotyczących algorytmów i sposobu przeprowadzenia próby. Następnie, jeżeli podgląd kamery jest aktywny, wykonane zostaje zdjęcie płytki na obszarze testowym, która w dalszym etapie będzie porównywana. Po zakończeniu operacji zapisywania fotografii, zainicjalizowane zostają funkcję GetPixel dla zdjęcia testowego i odpowiedniego wzorcowego, żeby później przesłać zwrócone tabele kolorów do

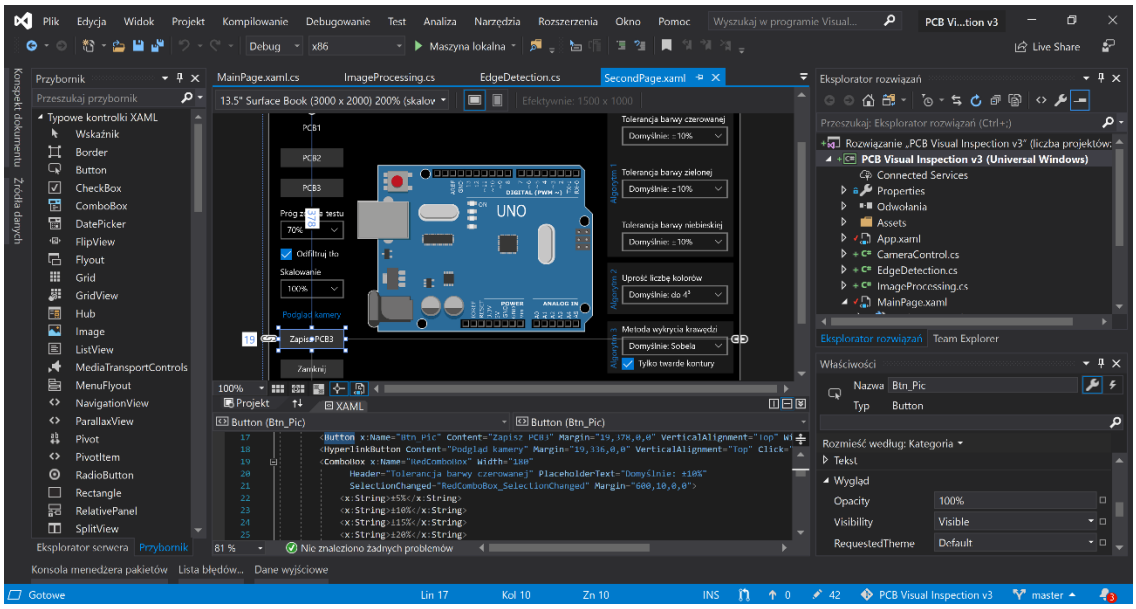
metody ComparePictures, która porównuje przesłane zmienne. Jeżeli algorytm upraszczający jest zaznaczony to wykona się również metoda SetPixel przetwarzająca dane zdjęcia oraz GetPixel i ComparePictures. Jeśli znacznik algorytmu 3 (krawędziowego) jest aktywny to zamiast SetPixel zostanie wykonana metoda ConvolutionFilter, która zwraca tworzy nowe zdjęcie z wyszczególnionymi krawędziami, a następnie porównana w ten sposób co w poprzednich algorytmach.

Warto zauważyć, że wyłączanie podglądu w trakcie działania tej funkcji ma na celu zwolnienie zużycia zasobów procesora, aby mógł całą moc obliczeniową skupić na obliczaniu algorytmów. W trakcie przesyłania widoku obrazu z kamery, Raspberry Pi 3B wykorzystanie procesora wynosi około 50%, a po jego wyłączeniu zużycie spada do około 10%. Zwiększenie szybkości działania programu zapewnia również skorzystanie z metod asynchronicznych, za pomocą operatorów „async" i „await", które przy długich dla procesora obliczeniach nie blokują innych operacji, które mogą działać w tle Ogólnie rzecz biorąc, napisana w kodzie programu głównego funkcja działająca po naciśnięciu przycisku „Wykonaj test", jest napisana w bardzo prosty i długi sposób, ale zawiera wszystkie możliwe do wystąpienia przypadki.

4.4 Interfejs użytkownika.

Program Microsoft Visual Studio 2019 posiada wbudowane narzędzie do tworzenia interfejsu użytkownika i jest oparte na języku programowania XAML. W ten sposób można dodawać obiekty do strony, takie jak na przykład przyciski, zdjęcia czy pola tekstowe.

Poniżej przestawiono edytor XAML (rys. 4.7). W środkowej części znajdują się dwa elementy. Dolny to edytor tekstowy, w którym kod może być wpisywany ręcznie lub generowany automatycznie przez wstawianie obiektów z przybornika i zmienianie ich parametrów w karcie „Właściwości", natomiast górny to projekt strony, gdzie reprezentowany jest widok i parametry wszystkich wstawionych do edytora obiektów. Po lewej stronie znajduje się karta „Przybornik", zawierająca wszystkie możliwe elementy, nazywane kontrolkami XAML, do wstawienia w projekcie strony. Obiekty zastosowane w SecondPage.xaml to: Button, CheckBox, ComboBox, Image i HyperlinkButton. W prawej dolnej części widnieje karta „Właściwości", dzięki której można ustawiać parametry wstawianych kontrolek. Przykładowo, w przycisku jest możliwość zmiany między innymi jego koloru, tekstu, przezroczystości czy interakcji.

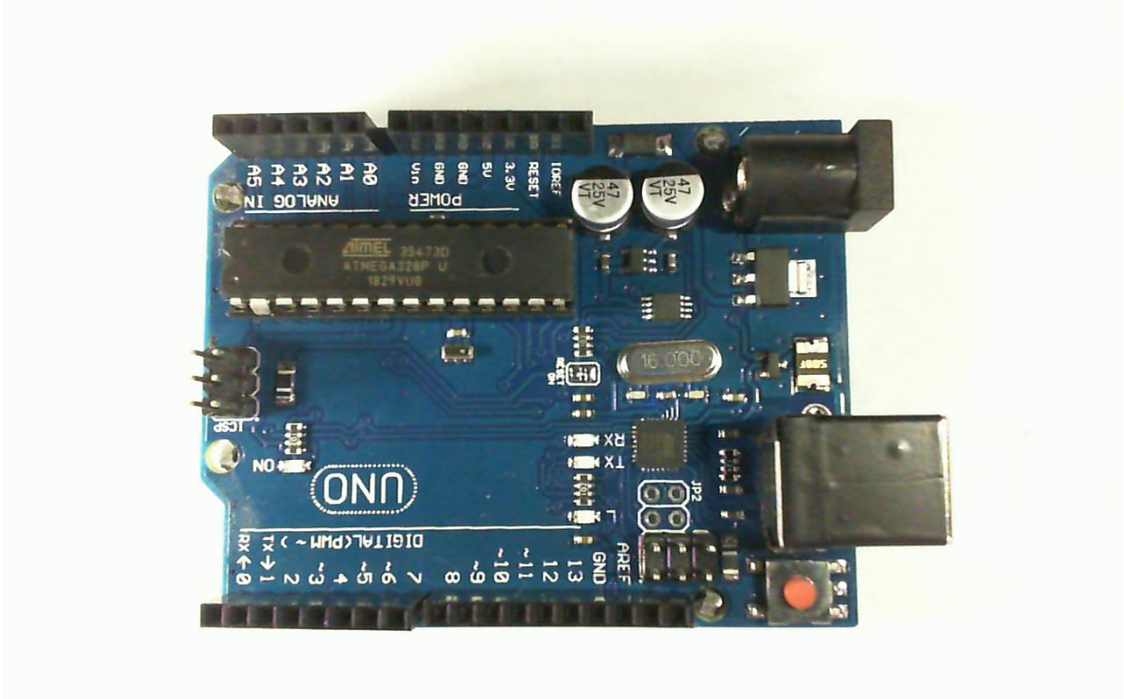


Rys. 4.7. Okno edycji strony XAML w Microsoft Visual Studio 2019 Community.

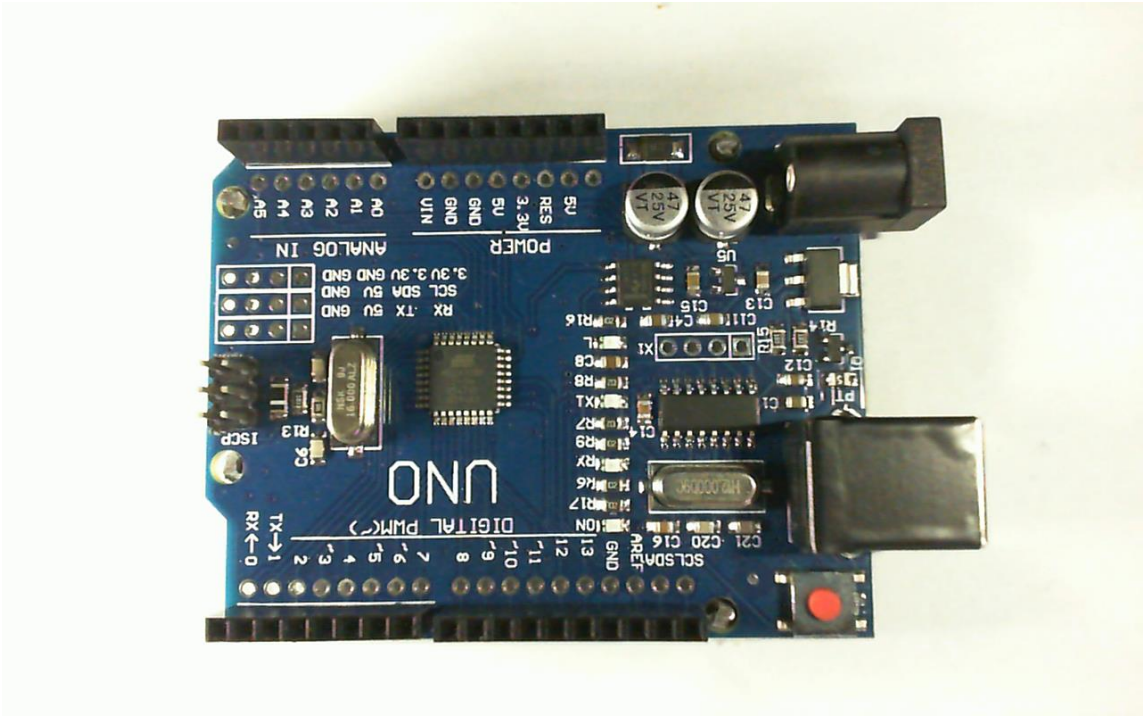
Dodatkowo, wszystkie utworzone kontrolki XAML mogą być ściśle powiązane z kodem programowania C# danej strony. Dostępną funkcją jest tworzenie zdarzeń na konkretny rodzaj akcji, na przykład na przyciśniecie przycisku, aby wykonać jakąś część kodu. Innym rodzajem sprzężenia tych dwóch edytorów jest możliwość odwoływania się w kodzie C# do konkretnej kontrolki i programowe zmienianie koloru czy tekstu widniejącego na niej. Do kodu głównego z edytora XAML zostaje przesłany obiekt wywołujący dane zdarzenie, na przykład przyciśnięcie prawego przycisku myszy, a także zmienna z klasy EventArgs, która zawiera informacje o stanie i danych skojarzonych ze zdarzeniem kierowanym. Takie zdarzenie jest wywoływane w momencie zmiany wyświetlanego elementu strony takiego jak ComboBox, który reaguje na wybranie innego parametru z rozwijalnej listy.

5 PRZYKŁAD DZIAŁANIA URZĄDZENIA

Z założenia urządzenie ma służyć do kontroli wizyjnej płytek drukowanych. Do przeprowadzenia testów jego poprawności wykorzystano płytki Arduino UNO Rev3 jako PCB1 (rys. 5.1) oraz UNO Board jako PCB2 (rys. 5.2), ale użytkownik ma też możliwość zapisania własnej płytki drukowanej jako PCB3 i wykonywania na niej analizy wizyjnej.

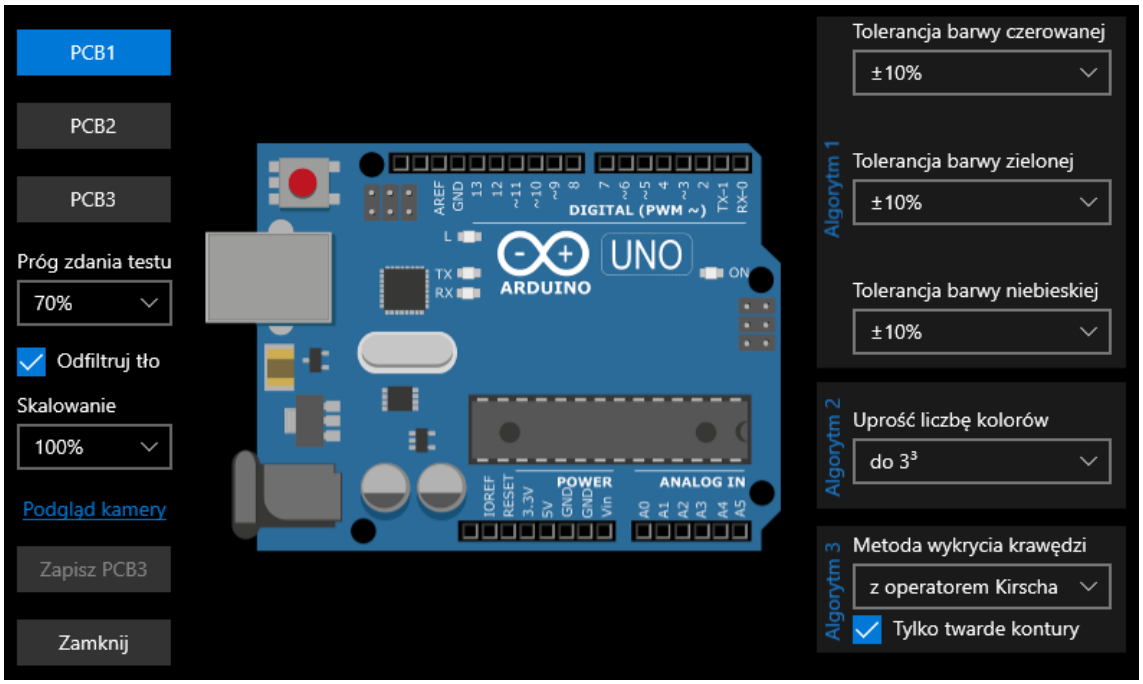


Rys. 5.1. Zdjęcie wzorcowe PCB1 (Arduino UNO Rev3).



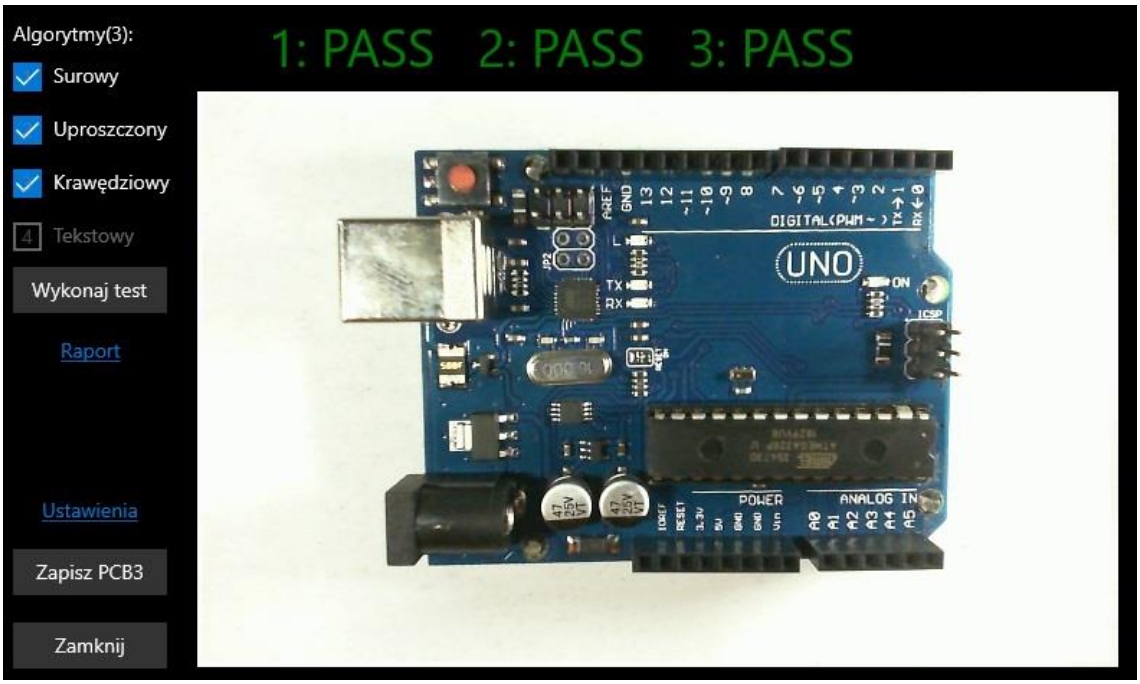
Rys. 5.2. Zdjęcie wzorcowe PCB2 (UNO Board).

Po uruchomieniu programu jako pierwsze pojawia się okno SecondPage z ustawieniami parametrów dla przeprowadzonych prób (rys. 5.3). Tutaj po prawej stronie można ustawić dla algorytmu pierwszego (surowego) tolerancję dla trzech barw: ±5%, ±10%, ±15% i ±20%; gdzie wartość ±10% jest domyślną przy włączeniu programu. Algorytm drugi (uproszczony) zmniejsza liczbę kolorów z 256^3 do 3^3 (domyślnie), 4^3 lub 5^3. Warto zauważyć, że wtedy algorytm w trakcie porównywania zdjęć nie stosuje tolerancji dla poszczególnych kolorów, a opiera się wyłącznie na bezpośrednim porównaniu pikseli przetworzonych przez ten algorytm obrazów. Dla trzeciego algorytmu (krawędziowego) można wybrać, które macierze zostaną wykorzystane, pomiędzy operatorem Sobela, Prewitta oraz Kirscha, który jest ustawiony jako domyślny. Dodatkowo można włączyć opcję twardego konturu, który pozostawia na przetworzonym zdjęciu tylko krawędzie w kolorze białym. Po lewej stronie widnieją przyciski do wyboru testowanej płytki, których obraz poglądowy na środku okna strony również się zmienia. Oprócz tego jest możliwość wyznaczenia progu zdania testu przez algorytmy w granicach od 50% do 90% co 10%. Pole wyboru „Odfiltruj tło" pozwala na przełączenie działania funkcji CheckBackground w metodzie ComparePictures. Program pozwala też na zmniejszenie sprawdzanego i wzorcowego zdjęcia o 50% lub 25% za pomocą opcji „Skalowanie". Hiperłącze „Podgląd kamery" przenosi użytkownika do głównej strony aplikacji MainPage i przesyła ustawione parametry za pomocą klasy ColorTolerance. Przycisk „Zapisz PCB3" jest uaktywniony w momencie przyciśnięcia przycisku „PCB3" i pozwala on na przejście do strony z podglądem kamery, gdzie jest możliwość zapisania zdjęcia jako fotografii wzorcowej PCB3. Ostatnim elementem na tej stronie jest przycisk „Zamknij" i jak sama nazwa wskazuje jego naciśnięcie powoduje wyjście z programu. Aplikacje na system operacyjny Windows 10 IoT Core nie posiadają w prawym górnym rogu przycisków do zamykania, minimalizacji czy maksymalizacji, więc programy na tą platformę powinny mieć dedykowany przycisk do wyłączenia procesu.



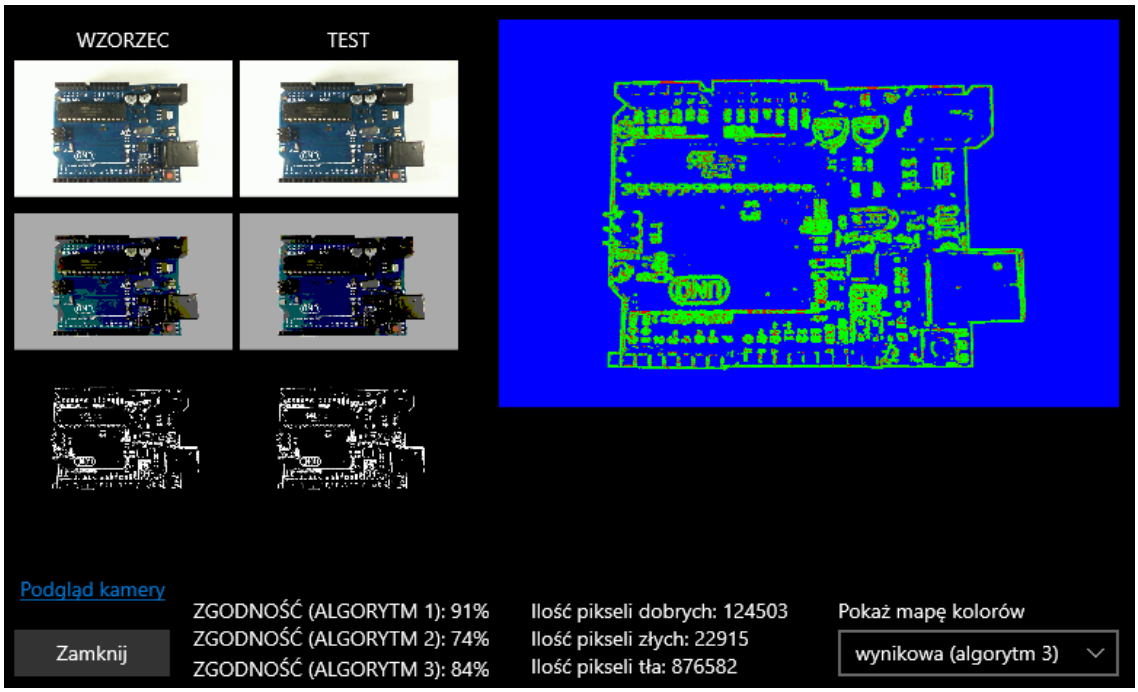
Rys. 5.3. Okno „Ustawienia" w aplikacji projektu inżynierskiego.

Kolejnym oknem jest MainPage (rys. 5.4). W środkowej części znajduje się podgląd obrazu z kamery. Po lewej stronie są pola wyboru algorytmów, które będą realizowane. Pod nimi umieszczony został przycisk „Wykonaj test”, który rozpoczyna proces weryfikacji wizyjnej, a po skończonej operacji wyświetlony zostaje w górnej części blok tekstowy z informacją o wyniku próby dla danego algorytmu. Jeśli wszystkie zwrócą pozytywny wynik to kolor tekstu będzie zielony, jak wszystkie będą negatywne to będzie czerwony, a w każdym innym przypadku będzie żółty. Niżej widnieją dwa hiperłącza: „Raport” przenosi użytkownika do strony z wynikami testu i przetworzonymi zdjęciami; „Ustawienia” powraca do strony SecondPage opisanej powyżej. Przycisk „Zapisz PCB3” wykonuje zdjęcie oraz ustawia je jako obraz wzorcowy dla płytki PCB3. Warto zauważyć, że podgląd kamery jest odwrócony o 180°, jednak wykonywane zdjęcia już nie są. Nie wpływa to na funkcjonowanie aplikacji, ponieważ jeśli wszystkie zdjęcia wzorcowe i testowe są w taki sposób ustawione, to sposób sprawdzanie będzie taki sam.



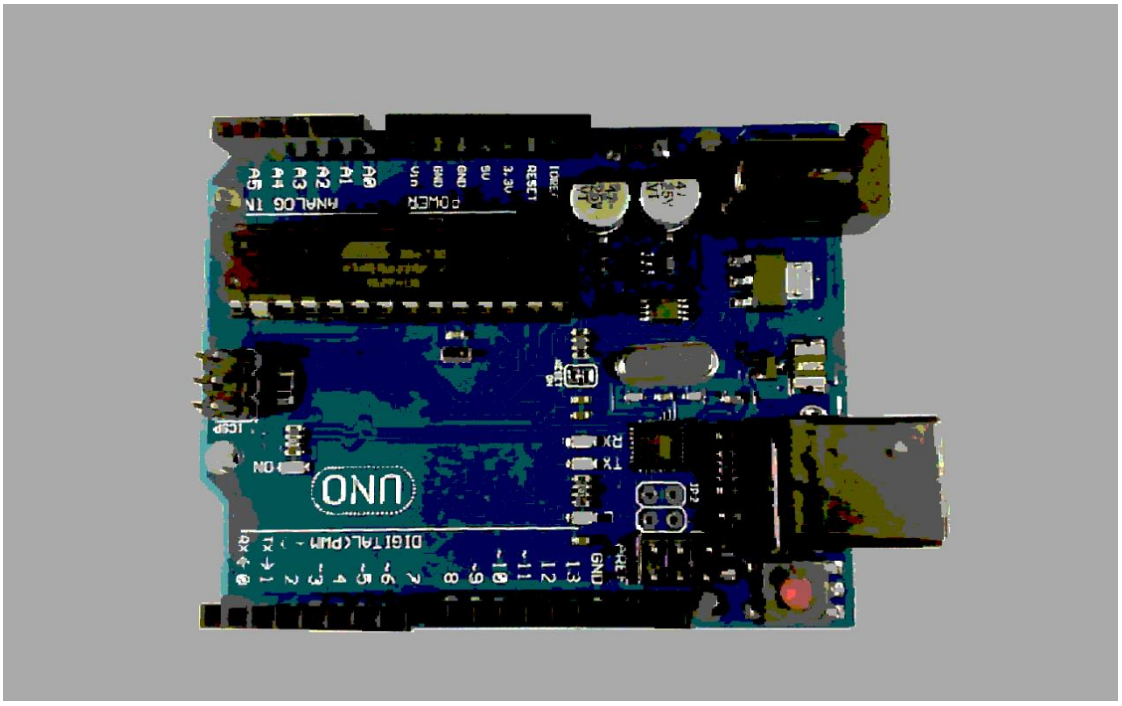
Rys. 5.4. Okno „Podgląd kamery” w aplikacji projektu inżynierskiego.

Ostatnim oknem jest ThirdPage (rys. 5.5). Na samym dole znajdują się bloki tekstowe z procentowymi wynikami testów i ilością pikseli przyporządkowaną do odpowiedniego parametru. Te dane są przesyłane z MainPage za pomocą klasy Pictures. Oprócz tego, w dolnej części okna jest też hiperłącze, które przenosi użytkownika do podglądu kamery, a także lista z wyborem wyświetlenia typu mapy kolorów. Dostępne są dwa rodzaje tego typu obrazów: cieplna, która reprezentuje wartość bezwzględną z różnicy pomiędzy kolorami piksela w porównywanych obrazach; wynikowa dla wszystkich trzech algorytmów, gdzie pokazane są zgodności pikseli w odniesieniu do tolerancji i opcjonalnie na niebiesko przedstawione jest odfiltrowane białe (dla algorytmu pierwszego i drugiego) lub czarne tło (dla algorytmu trzeciego). Ta strona zawiera także wszystkie wykonane i porównywane zdjęcia, a dodatkowo użytkownik ma możliwość powiększenia dowolnego z nich, gdy naciśnie myszką lub palcem na nie.



Rys. 5.5. Okno „Raport” w aplikacji projektu inżynierskiego.

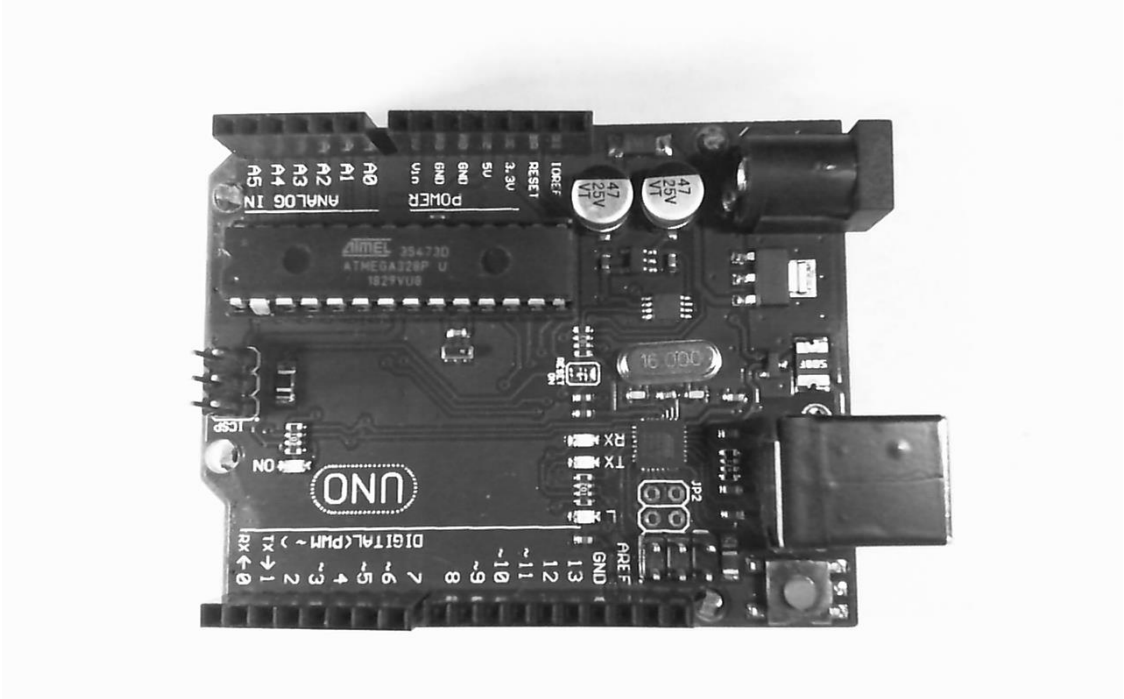
Algorytm drugi (uproszczony) zmniejsza liczbę kolorów do ustalonej wartości. Poniżej przedstawiono obraz (rys. 5.6), na którym widać przetworzone przez algorytm zdjęcie z ponad 16,5 mln kolorów do 27.



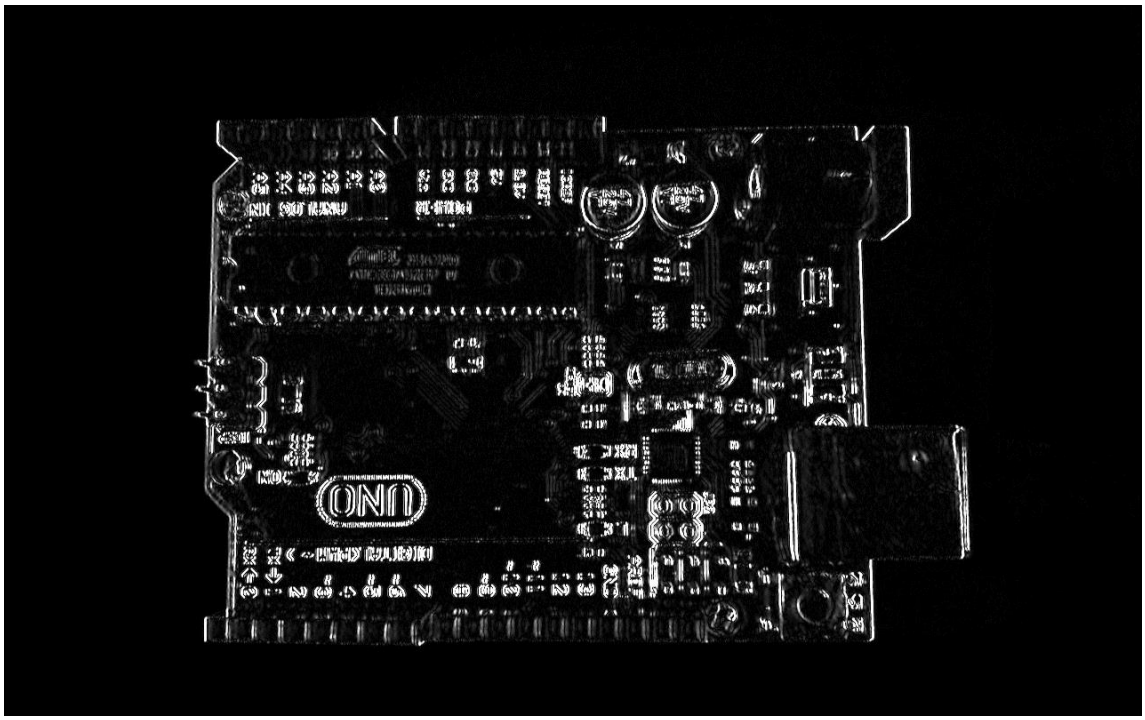
Rys. 5.6. Uproszczone do 3^3 kolorów zdjęcie płytki wzorcowej PCB1.

Trzeci algorytm (krawędziowy) działa stopniowo. Na samym początku konwertuje obraz do odcieni szarość (rys. 5.7), dzięki czemu działanie funkcji jest bardziej precyzyjne. Następnie nakładane są na obraz maski wybranego w programie operatora. Jednocześnie wykonywana jest konwolucja źródłowego piksela z macierzą dla wymiaru „x” (rys. 5.8) oraz z macierzą dla wymiaru

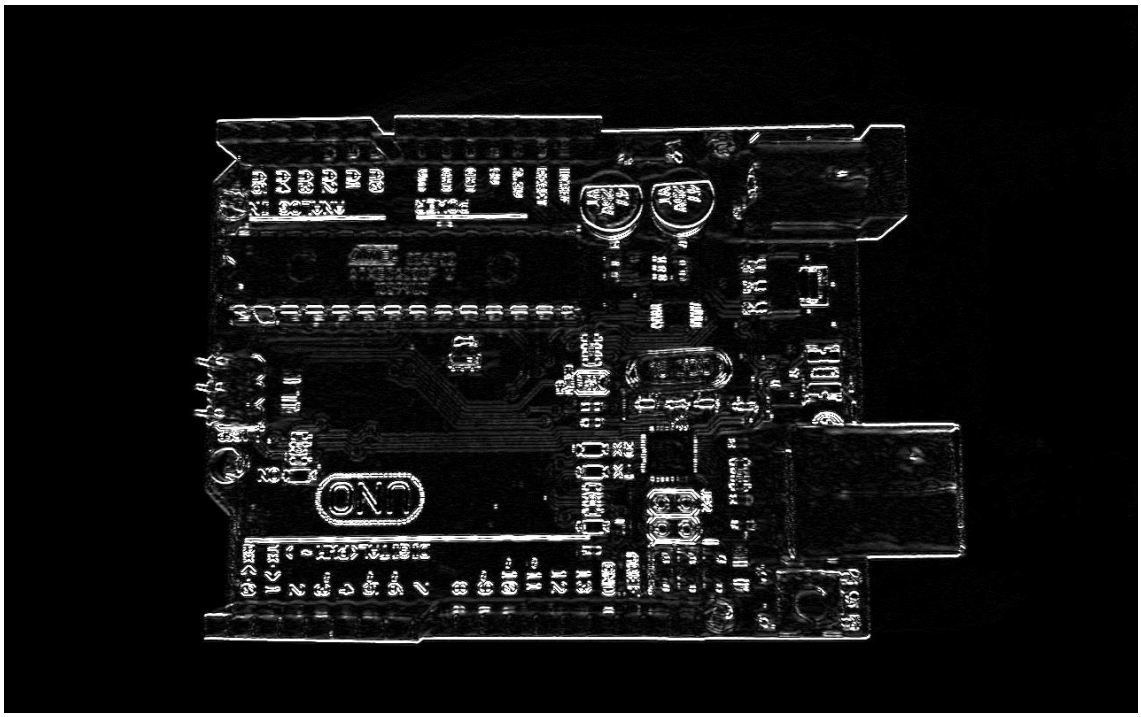
„y” (rys. 5.9), a następnie wykonana zostaje suma kwadratów otrzymanego piksela z dwóch wymiarów (rys. 5.10). Dodatkowo, jeśli włączone zostały twarde kontury to funkcja odfiltruje od otrzymanego zdjęcia kolory inne niż biały (rys. 5.11).



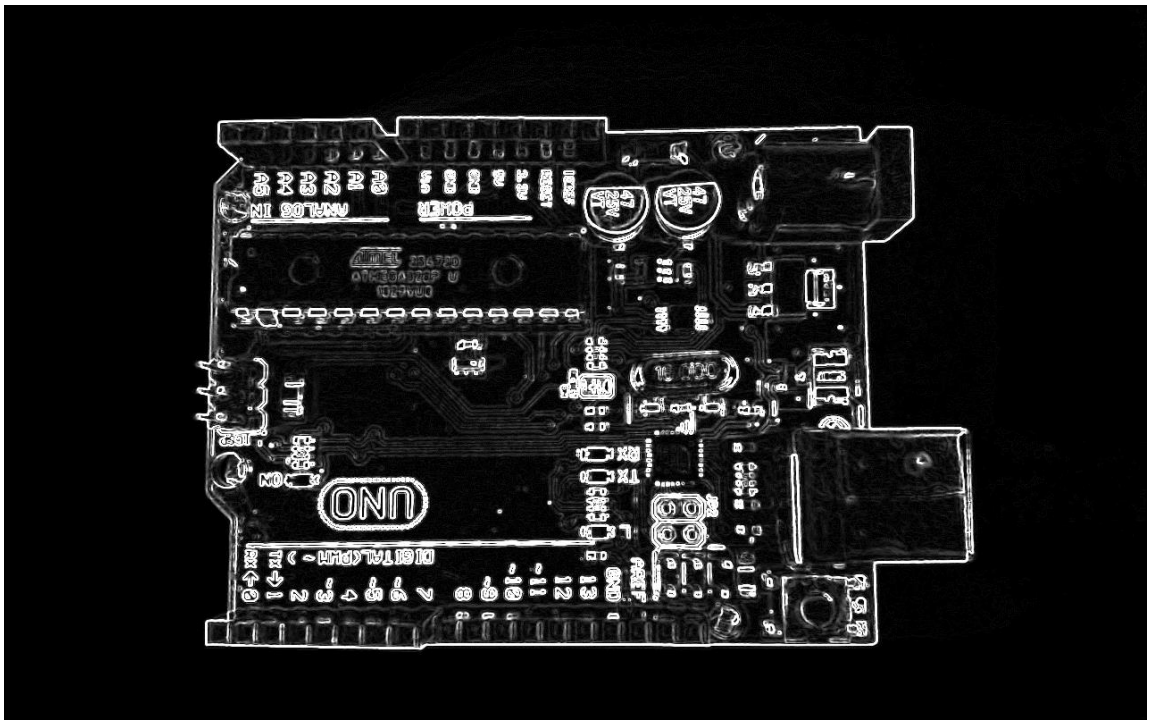
Rys. 5.7. Czarno-białe zdjęcie płytki wzorcowej PCB1.



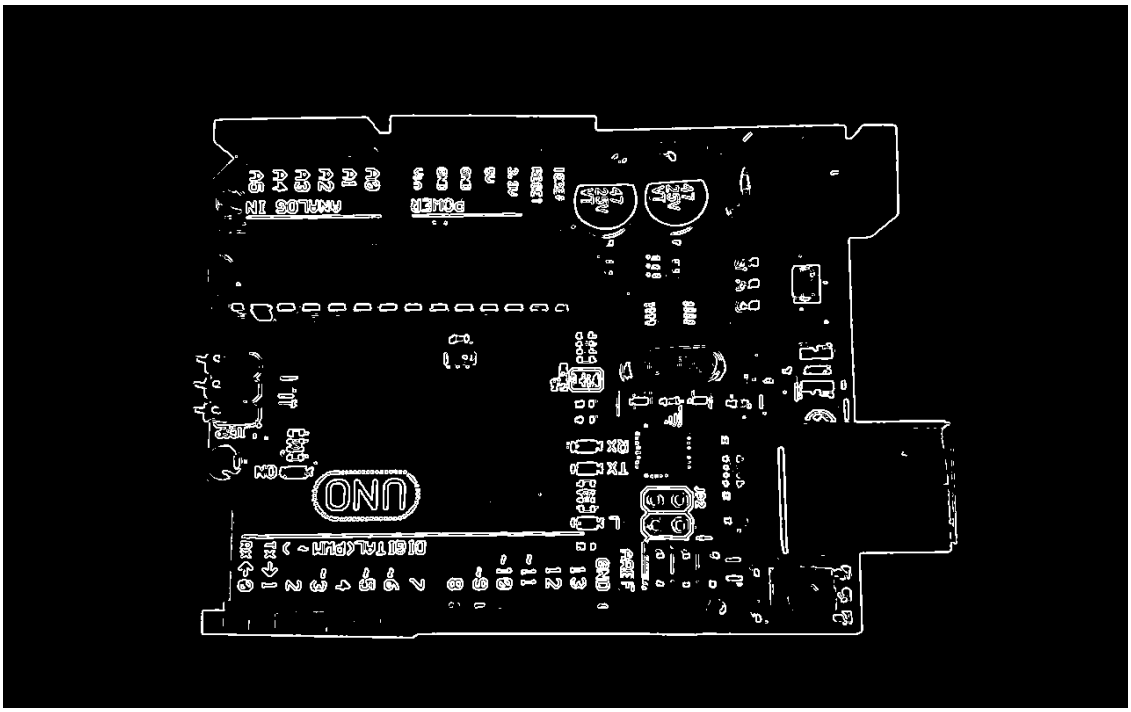
Rys. 5.8. Zdjęcie płytki wzorcowej PCB1 po nałożeniu operatora Sobela w wymiarze „x”.



Rys. 5.9. Zdjęcie płytki wzorcowej PCB1 po nałożeniu operatora Sobela w wymiarze „y”.



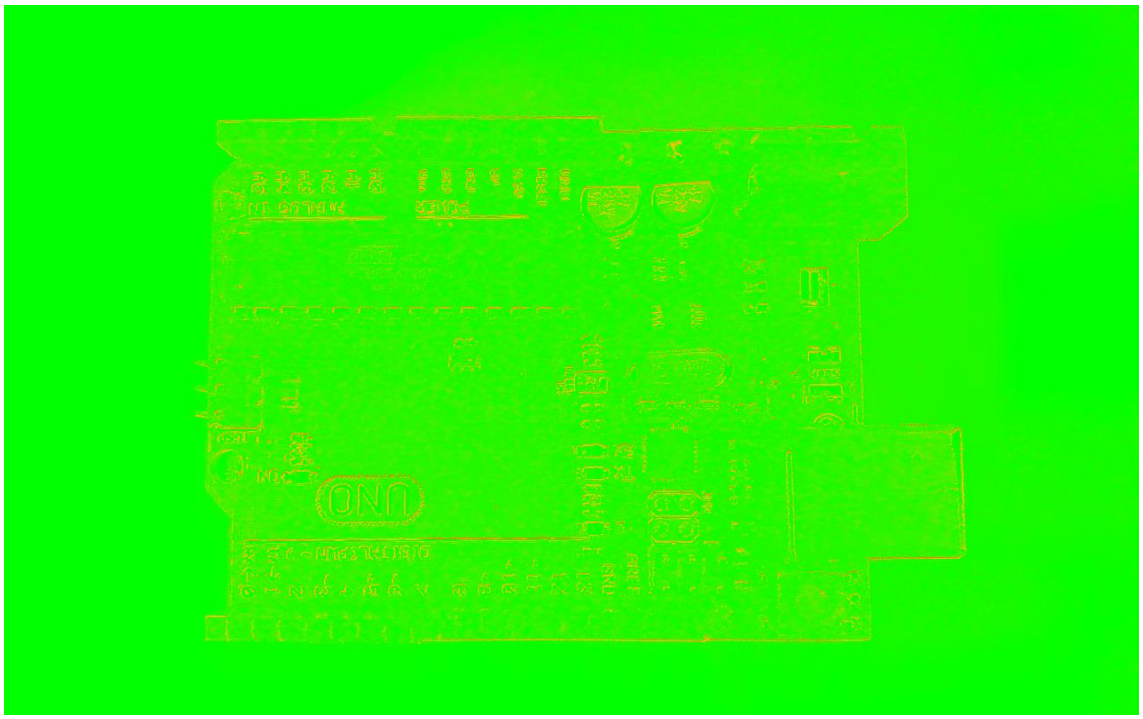
Rys. 5.10. Zdjęcie płytki wzorcowej PCB1 po nałożeniu operatora Sobela bez twardych konturów.



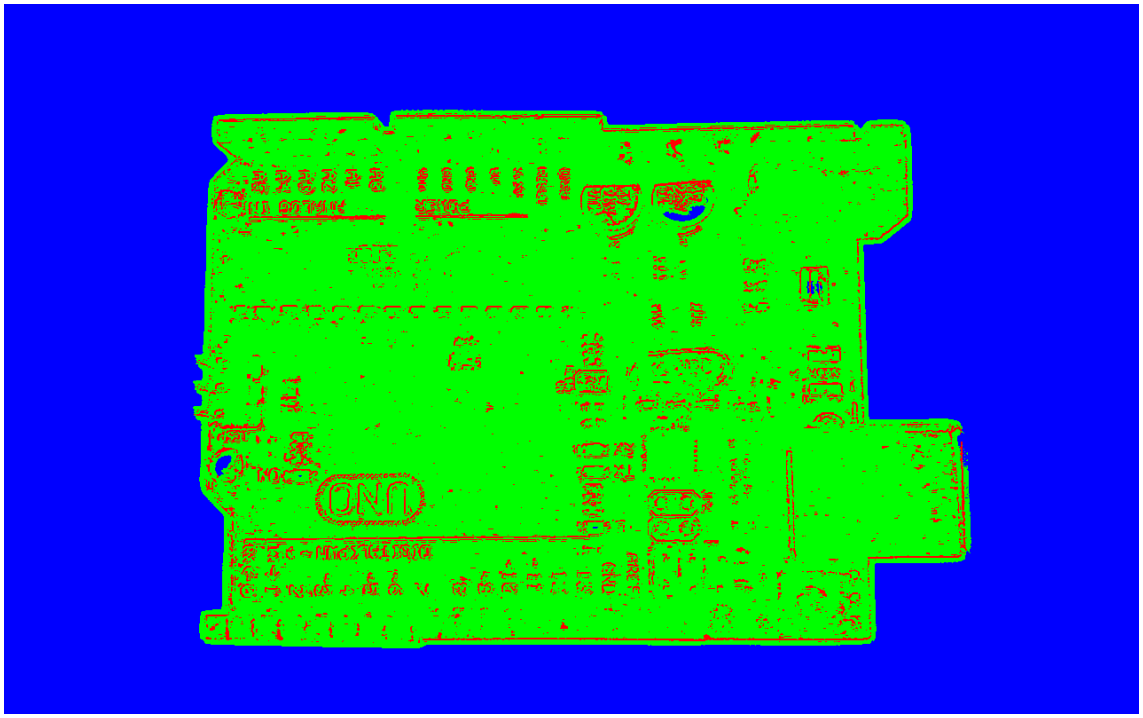
Rys. 5.11. Zdjęcie płytki wzorcowej PCB1 po nałożeniu operatora Sobela z twardymi konturami.

Analizując powyższe obrazy można stwierdzić, iż algorytm krawędziowy dość dobrze spełnia swoje zadanie. Co więcej, jest on też bardzo uniwersalny i może być zastosowany do każdego rozwiązania. Jedynym problemem może być odbijające się od powierzchni płytki światło, które niekiedy może wpływać na poprawność detekcji krawędzi jak na rysunku 5.11, widać, że po lewej stronie obrazu kontur testowanej płytki zanika.

Ostatnimi zaimplementowanymi metodami przetwarzania obrazu w programie to mapa cieplna (rys. 5.12) oraz mapa wynikowa (rys. 5.13). Mapa cieplna jest początkowo cała zielona, a następnie, jeśli jest różnica między pikselami płytki wzorcowej a badanej, to na każdy 1 punkt różnicy dodaje 5 koloru czerwonego i odejmuje 1 kolor zielony od piksela mapy cieplnej. Mapa wynikowa to trójkolorowy obraz przedstawiający w kolorze zielonym zgodne piksele, w kolorze czerwonym – niezgodne, a w niebieskim – tło.



Rys. 5.12. Mapa cieplna między zdjęciem wzorcowym PCB1 i przykładowo testowanym.



Rys. 5.13. Mapa wynikowa algorytmu pierwszego (surowego) między zdjęciem wzorcowym PCB1 i przykładowo testowanym.

Błędy w rozpoznawaniu obrazu wynikają z braku możliwości ustawienia parametrów kamery internetowej, ponieważ nie posiada ona sterownika dla systemu Windows 10. Kamera automatycznie wyostrza obraz i zmienia kontrast, co jest właściwie jedynym większym powodem rozbieżności przyprowadzanych prób. Gdyby istniała możliwość na stałe dobrania odpowiednich wartości tego modelu kamerki, to po implementacji programowej wyniki pomiarowej byłyby o wiele dokładniejsze. Innym czynnikiem wpływającym na testy może być wpływ światła zewnętrznego. Jednak można go wyeliminować stosując zamkniętą obudowę i tym samym tworząc ciemnię, aby warunki oświetleniowe były jeszcze bardziej do siebie zbliżone.

Funkcja odfiltrowująca tło CheckBackground jest zaraz po TakePicture najdłużej wykonującą się. Z tego względu wprowadzono opcję skalującą zdjęcie, aby skrócić czas jej działania, jeśli użytkownikowi zależy na szybkości działania programu. Za pomocą metody Stopwatch zmierzono czas działania funkcji CheckBackground w zależności od rozdzielczości sprawdzanego zdjęcia, a wyniki przedstawiono w poniższej tabeli (tab. 5.1):

Tabela 5.1. Czas działania funkcji CheckBackground w zależności od rozdzielczości obrazu.

Lp.	Czas działania funkcji [ms]					
	Skalowanie 100% rozdzielczość 1280x800		Skalowanie 50% rozdzielczość 640x400		Skalowanie 25% rozdzielczość 320x200	
	Algorytm 1	Algorytm 2	Algorytm 1	Algorytm 2	Algorytm 1	Algorytm 2
1	991,239	1013,238	237,528	229,990	54,489	55,872
2	969,397	968,482	235,807	228,936	54,490	54,125
3	965,094	961,176	238,895	229,858	54,624	55,105
4	990,176	1018,623	232,123	230,152	54,995	54,487
5	1163,667	1013,037	235,552	227,978	54,667	54,964
Śr.	1015,915	994,911	235,981	229,383	54,653	54,911

6 PODSUMOWANIE

Celem pracy był projekt urządzenia do kontroli wizyjnej płytek drukowanych za pomocą wybranych algorytmów. Postawione mu zadania z zakresu rozpoznawania obrazu zostały wykonane w stopniu pozwalającym na wychwycenie różnic między płytkami, a metodykę realizacji analizy przedstawiono w powyższych rozdziałach.

Zaprojektowany w ramach pracy inżynierskiej układ składający się z komputera jednopłytkowego Raspberry Pi 3B, kamery internetowej Microsoft LifeCam HD-5000, dotykowego wyświetlacza LCD, dwóch lampek LED oraz stelażu stanowiska badawczego, spełnia postawione mu wymagania. Testy aplikacji przeprowadzono dla trzech różnych płytek PCB przy jak najbardziej zbliżonych warunkach oświetleniowych. Wykonane serie próbnych działań programu dla wszystkich możliwych do ustawienia parametrów (tolerancję barw, uproszczenie liczby kolorów, wykrywanie krawędziowe za pomocą różnych operatorów z lub bez twardych konturów, próg zdawalność testu, przeskalowanie rozdzielczości obrazu, odfiltrowywanie tła) potwierdzają poprawność działania algorytmów.

Projekt można rozwinąć na kilka sposobów. Pierwszym z nich jest rozwinięcie konstrukcji do pełnego stanowiska testowego jak w fabrykach podzespołów elektronicznych. Wymagałoby to użycia taśmociągu sterowanego silnikami krokowymi i czujnikami zbliżeniowymi podłączanymi do wyprowadzeń GPIO płytki Raspberry Pi. Opcjonalnie można wyposażyć stanowisko w skaner kodów kreskowych, który czytałby je z testowanych płytek, a następnie zapisywałby logi odnośnie danego testu do systemu SAP. Inną możliwością ulepszenia funkcjonalności układu byłoby zainstalowanie lepszej kamery, która pozwalałaby na ustawienie parametrów przybliżenia, kontrastu czy ostrości. Jeszcze inną funkcją rozszerzającą mogłoby być zaprogramowanie algorytmu rozpoznawania tekstu lub wykorzystującego głębokie uczenie maszynowe i sztuczną inteligencję, który sam dobierałby odpowiednie macierze filtrów do danego typu obrazów. Powyższe pomysły mogą stanowić rozwinięcie projektu w ramach pracy magisterskiej.

WYKAZ LITERATURY

1. W. Malina, M. Smiatacz. *Metody cyfrowego przetwarzania obrazów*. Warszawa: Akademicka Oficyna Wydawnicza EXIT, 2005.
2. R. Tadeusiewicz, P. Korohoda. *Komputerowa analiza i przetwarzanie obrazów*. Kraków: Wydawnictwo Postępu Telekomunikacji, 1997.
3. Z. Świerczyński, P. Rokita. "Podwyższanie rozdzielczości obrazów cyfrowych z wykorzystaniem informacji o krawędziach". Biuletyn Instytutu Automatyki i Robotyki, nr 25, 2008, s. 117-120.
4. C. Poynton. *Digital Video and HDTV Algorithms and Interfaces*. San Francisco: Morgan Kaufmann Publishers, 2003.
5. J. Ratajczak. "Cyfrowe przetwarzanie obrazów i sygnałów". Laboratorium Robotyki Politechniki Wrocławskiej, wykład 5, 2015, s. 56.
6. Oficjalna strona płytki NVIDIA Jetson Nano: https://www.nvidia.com/pl-pl/autonomous-machines/embedded-systems/jetson-nano/
7. Oficjalna strona płytki AAEON UP Board: https://www.aaeon.com/en/p/up-board-computer-board-for-professional-makers
8. Oficjalna strona płytki Raspberry Pi 3B: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
9. Oficjalna dokumentacja interfejsu UWP: https://docs.microsoft.com/pl-pl/windows/uwp/
10. Dokumentacja techniczna wyświetlacza LCD Raspberry Pi Touch Display: https://www.raspberrypi.org/documentation/hardware/display/
11. Dokumentacja techniczna kamery internetowej Microsoft LifeCam HD-5000: https://media.flixcar.com/f360cdn/Microsoft-124674720-TDS_LifeCamHD-5000.pdf
12. Lista urządzeń posiadająca gotowy obraz systemu Windows 10 IoT Core: https://docs.microsoft.com/en-us/windows/iot-core/tutorials/quickstarter/prototypeboards
13. Oficjalna strona Microsoft Azure: https://azure.microsoft.com/pl-pl/
14. Oficjalna strona firmy Misty Robotics: https://www.mistyrobotics.com/
15. Oficjalna strona produktu GLAS firmy Johnson Controls: https://glas.johnsoncontrols.com/
16. A. Troelsen, *Język C# 2010 i platforma .NET 4*. Warszawa: Wydawnictwo Naukowe PWN, 2011
17. Programowanie asynchroniczne w języku C#: https://docs.microsoft.com/pl-pl/dotnet/csharp/async
18. Zalecenie nr 709 Sektora Radiotelekomunikacji Międzynarodowego Związku Telekomunikacji: https://www.itu.int/rec/R-REC-BT.709-6-201506-I/en

WYKAZ RYSUNKÓW

Dodatek A: Zawartość płyty CD

Na płycie CD zawarto:

- pracę inżynierską w wersji cyfrowej edytowalnej oraz nieedytowalnej (Skórski_Arkadiusz_168879_2019.docx, Skórski_Arkadiusz_168879_2019.pdf),
- paczkę instalacyjną dla Windows 10 i instrukcję instalacji aplikacji (folder „Instalator”),
- karty katalogowe, noty aplikacyjne oraz inne materiały pomocnicze ogólnodostępne wykorzystane w pracy (folder „Materiały pomocnicze”),
- przykładowe zdjęcia do zaprezentowania działania programu (folder „Zdjęcia”),
- krótki film prezentujący działanie układu (Skórski_Arkadiusz_168879_2019.mp4).

Dodatek B: Kod źródłowy klasy CameraControl

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Windows.Devices.Enumeration;
using Windows.System.Display;
using Windows.Media.Capture;
using Windows.Storage;
using Windows.Media.MediaProperties;
using Windows.UI.Xaml.Controls;
using Windows.Media.Devices;

namespace PCB_Visual_Inspection_v2
{
    class CameraControl
    {
        // ten obiekt daje możliwość przechwytywania obrazu z modułu kamery
        public MediaCapture Capture;

        // ten obiekt pozwala decydować kiedy uśpić działanie kamery
        private readonly DisplayRequest RequestDisplay = new DisplayRequest();

        // zapożyczone z: https://msdn.microsoft.com/en-
us/library/windows/apps/xaml/hh868174.aspx
        private static readonly Guid RotationKey = new Guid("C380465D-2271-428C-9B83-
ECEA3B4A85C1");

        // zbiorcze informacje o dostępnych urządzeniach
        DeviceInformationCollection Devices;
        public string ErrorCode;

        // ten obiekt jest elementem interfejsu użytkownika
        // i umożliwia wyświetlenie obrazu z kamery
        public CaptureElement Preview { get; set; }

        public async Task Initialize(int rotation_offset)
        {

            // zbiera informacje o dostępnych urządzeniach umożliwiających przechwyt wideo
            Devices = await DeviceInformation.FindAllAsync(DeviceClass.VideoCapture);

            // jeśli nie istnieje bierzący podgląd kamery oraz jeśli kamer wideo jest więcej niż
 0
            // to wykonaj operacji - jeśli warunki nie spełnione to ustaw kod błedu
            if (Capture == null && Devices.Count > 0)
            {
                // wybiera pierwsze napotkane urządzenia do przechwycenia obrazu
                var preferredDevice = Devices.FirstOrDefault();

                // tworzy nowy obiekt z klasy MediaCapture do przechwytywania obrazu
                Capture = new MediaCapture();

                // ustawia aby ekran nie usypiał się w momencie bezczynności
                RequestDisplay.RequestActive();

                // inicjalizacja podglądu kamery
                await Capture.InitializeAsync(
                    new MediaCaptureInitializationSettings
                    {
                        VideoDeviceId = preferredDevice.Id
                    });

                // zapisywanie do tabeli jednowymiarowej wszystkich dostępnych rozdzielczości ka
mery
                var resolutions = Capture.VideoDeviceController.GetAvailableMediaStreamPropertie
s(MediaStreamType.Photo).ToList();

                // ustawia typ zapisywanych danych jako zdjęcia oraz ustawia rozdzielczość
                await Capture.VideoDeviceController.SetMediaStreamPropertiesAsync(MediaStreamTyp
e.Photo, resolutions[47]); // 5 - 1280x720; 47 - 1280x800;
```

```
                // ustawia źródło wyświetlania jako strumień danych z obiektu przechwytującego o
braz
                Preview.Source = Capture;

                // rozpoczęcie przesyłu obrazu do podglądu
                await Capture.StartPreviewAsync();

                // ustawienie rotacji obrazu o wartość zadaną w funkcji
                var rotation_settings = Capture.VideoDeviceController.GetMediaStreamProperties(M
ediaStreamType.VideoPreview);
                rotation_settings.Properties.Add(RotationKey, rotation_offset);
                await Capture.SetEncodingPropertiesAsync(MediaStreamType.VideoPreview, rotation_
settings, null);
            }
            else
            {
                ErrorCode = "Nie znaleziono kamery";
            }
        }

        public async Task TakePicture(string image_name)
        {

            // stworzenie pliku, do którego będzie zapisane zdjęcie i podanie jego ścieżki zapis
u
            var file = await KnownFolders.PicturesLibrary.CreateFileAsync(image_name, CreationCo
llisionOption.ReplaceExisting);

            // ustawienie typu zapisywanego zdjęcia - w tym wypadku obrazu typu bitmapa
            ImageEncodingProperties Picture = ImageEncodingProperties.CreateBmp();

            // zaktualizwanie pliku o treść przesłaną w zdjęciu
            await Capture.CapturePhotoToStorageFileAsync(Picture, file);
        }

        public void Dispose()
        {
            // funkcja służąca do wyłączenia przechwytywania obrazu
            // w momencie gdy nie jesteśmy na stronie z podglądem kamery
            if (Capture != null)
            {
                Capture.Dispose();
                Capture = null;
            }

            if (Preview.Source != null)
            {
                Preview.Source.Dispose();
                Preview.Source = null;
            }
        }

        public async void Focus() // nieużywane
        {
            // spróbuj ustawić kontrast
            Capture.VideoDeviceController.Contrast.TrySetAuto(true);

            // zwraca informację czy urządzenia wspiera focus
            if (Capture.VideoDeviceController.FocusControl.Supported == true)
            {
                // uzyskaj kontrolę nad focusem od obiektu Capture
                var focus_control = Capture.VideoDeviceController.FocusControl;

                // ustawia ostrość
                var focus_range = focus_control.SupportedFocusRanges.Contains(AutoFocusRange.Ful
lRange) ? AutoFocusRange.FullRange : focus_control.SupportedFocusRanges.FirstOrDefault();

                // ustawia tryb
                var focus_mode = focus_control.SupportedFocusModes.Contains(FocusMode.Single) ?
FocusMode.Single : focus_control.SupportedFocusModes.FirstOrDefault();

                // konfiguracja focusa z powyższymy parametrami
                focus_control.Configure(
                    new FocusSettings
```

```
                    {
                        Mode = focus_mode,
                        AutoFocusRange = focus_range
                    });

                // zaczekaj aż skończy wyostrzać obraz
                await focus_control.FocusAsync();
            }
        }

    }
}
```

Dodatek C: Kod źródłowy klasy ImageProcessing

```
using System;
using System.IO;
using System.Threading.Tasks;
using Windows.Storage;
using Windows.UI.Xaml.Media.Imaging;
using Windows.Graphics.Imaging;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Storage.Streams;
using System.Diagnostics;


namespace PCB_Visual_Inspection_v2
{
    public class ImageProcessing
    {
        public byte[,] RedColorArray = new byte[0, 0];
        public byte[,] GreenColorArray = new byte[0, 0];
        public byte[,] BlueColorArray = new byte[0, 0];
        public byte[,] AlphaColorArray = new byte[0, 0];
        public uint GoodPixelCount = 0;
        public uint BadPixelCount = 0;
        public uint BackgroundPixelCount = 0;
        public double Score = 0;
        public string ErrorCode;
        public bool[,] BackgroundArray = new bool[0, 0];

        public int ImageWidth { get; set; }
        public int ImageHeight { get; set; }
        public int MaxLength = 0;

        public async Task GetPixel(StorageFolder storage_location, string image_location, byte s
cale = 100, string scaled_image_location = "Resized.BMP")
        {
            // jeśli skalowanie aktywne to wykonaj funcje Resize i zmień lokalizacje docelową pl
iku
            // na ten przed chwilą zeskalowany
            if (scale != 100)
            {
                await Resize(storage_location, image_location, scale, scaled_image_location);
                image_location = scaled_image_location;
            }

            // dekodowanie strumienia danych z konkretnego pliku (przesłanego w zmiennych)
            StorageFile file = await storage_location.GetFileAsync(image_location);
            Stream image_stream = await file.OpenStreamForReadAsync();
            BitmapDecoder decoder = await BitmapDecoder.CreateAsync(image_stream.AsRandomAccessS
tream());

            // zapis wymiarów obrazu do zmiennych
            ImageHeight = Convert.ToInt32(decoder.PixelHeight);
            ImageWidth = Convert.ToInt32(decoder.PixelWidth);
            MaxLength = Math.Max(ImageHeight, ImageWidth);

            // stworzenie ciągów bajtów do przechowywania informacji o pikseli
            // 1 piksel = 4 bajty (dla każdego kanału A,R,G,B)
            var data = await decoder.GetPixelDataAsync();
            byte[] pixel_array = data.DetachPixelData();

            Array.Resize(ref pixel_array, MaxLength * MaxLength * 4);

            RedColorArray = new byte[MaxLength, MaxLength];
            GreenColorArray = new byte[MaxLength, MaxLength];
            BlueColorArray = new byte[MaxLength, MaxLength];
            AlphaColorArray = new byte[MaxLength, MaxLength];

            // 1 bajt to kolor niebieski, 2 - zielony, 3 - czerowny, 4 - kanał alfa
            // ten ciąg 4 bajtów to jeden piksel
            uint k = 0;
            for (int i = 0; i < MaxLength; i++)
            {
                for (int j = 0; j < MaxLength; j++)
```

```
                {
                    BlueColorArray[i, j] = pixel_array[k]; // kolor niebieski
                    GreenColorArray[i, j] = pixel_array[k + 1]; // kolor zielony
                    RedColorArray[i, j] = pixel_array[k+2]; // kolor czerwony
                    AlphaColorArray[i, j] = pixel_array[k + 3]; // kanał alfa
                    k += 4;
                }
            }
        }

        public async Task SetPixel(byte[,] r_color, byte[,] g_color, byte[,] b_color, string fil
e_name, byte simplifier)
        {
            // zapis wymiarów obrazu do zmiennych
            int width = r_color.GetLength(0);
            int height = r_color.GetLength(1);

            // tutaj wykonywane jest uproszczenie liczby kolorów w postaci dzielenia bez reszty
            // i ponownego przemnożenia przez dane uproszczenie
            if (simplifier > 1)
            {
                for (int i = 0; i < width; i++)
                {
                    for (int j = 0; j < height; j++)
                    {
                        b_color[i, j] = Convert.ToByte(b_color[i, j] / simplifier);
                        b_color[i, j] *= simplifier;
                        g_color[i, j] = Convert.ToByte(g_color[i, j] / simplifier);
                        g_color[i, j] *= simplifier;
                        r_color[i, j] = Convert.ToByte(r_color[i, j] / simplifier);
                        r_color[i, j] *= simplifier;
                    }
                }
                byte t = 192;
                sbyte border=2; // odległość środkowego piksela do granicy maski sprawdzającej
                byte threshold=55; // próg przejścia
                bool is_background = false;

                // ustawienie wykrywania tła dla poszczególnych uproszczeń
                if (simplifier == 85)
                    t = 170;
                if (simplifier == 64)
                    t = 192;
                else
                    t = 204;

                for (int i = 0; i < width; i=i+border+1)
                {
                    for (int j = 0; j < height; j=j+border+1)
                    {
                        // algorytm zaczyna się po deadzone tab aby nie wyjść po zakres tabeli
                        if (((i > 1 * border) && (i < width - 1 * border)) && ((j > 1 * border)
&& (j < height - 1 * border)))
                        {
                            uint goodpixel_counter = 0, badpixel_counter = 0;

                            // sprawdzanie czy dany piksel jest tłem
                            if (r_color[i, j] == t && g_color[i, j] == t && b_color[i, j] == t)
                                is_background = true;
                            else
                                is_background = false;

                            // jesli piksel nie jestem tłem to wykonuje algorytm sprawdzający w
postaci krzyża-efektywnie 8 kierunków
                            // i nalicza ile wyników jest pozytywnych - jeśli powyżej zadanej wa
rtości
                            // to algorytm wszystkie w okolicy o długości maski piksele ustawia
jako ten ze środka maski
                            if (is_background == false)
                            {
                                for (int k = (-border); k <= border; k++)
                                {
                                    if (k == 0)
                                        continue;
```

```
                                    if (r_color[i + k, j + k] == r_color[i, j] && g_color[i + k,
 j + k] == g_color[i, j] && b_color[i + k, j + k] == b_color[i, j])
                                        goodpixel_counter++;
                                    else badpixel_counter++;

                                    if (r_color[i + k, j - k] == r_color[i, j] && g_color[i + k,
 j - k] == g_color[i, j] && b_color[i + k, j - k] == b_color[i, j])
                                        goodpixel_counter++;
                                    else badpixel_counter++;

                                    if (r_color[i + k, j + 0] == r_color[i, j] && g_color[i + k,
 j + 0] == g_color[i, j] && b_color[i + k, j + 0] == b_color[i, j])
                                        goodpixel_counter++;
                                    else badpixel_counter++;

                                    if (r_color[i + k, j + k] == r_color[i, j] && g_color[i + k,
 j + k] == g_color[i, j] && b_color[i + k, j + k] == b_color[i, j])
                                        goodpixel_counter++;
                                    else badpixel_counter++;
                                }
                                if (100 * goodpixel_counter / (badpixel_counter + goodpixel_coun
ter) > threshold)
                                {
                                    for (int x = -(border); x <= border; x++)
                                    {
                                        for (int y = -(border); y <= border; y++)
                                        {
                                            if (x == 0 & y == 0)
                                                continue;
                                            r_color[i + x, j + y] = r_color[i, j];
                                            g_color[i + x, j + y] = g_color[i, j];
                                            b_color[i + x, j + y] = b_color[i, j];
                                        }
                                    }
                                }
                            }
                        }
                    }
                    }
            }

            // WriteableBitmap używa formatu BGRA - 4 bajty na piksel
            byte[] image_array = new byte[width * height * 4];

            BitmapAlphaMode alpha_mode = BitmapAlphaMode.Ignore;

            // zapis do ciągów bajtów odpowiedniej tabeli dwuwymiarowej z kolorem
            for (int i = 0; i < width; i++)
            {
                for (int j = 0; j < height; j++)
                {
                    image_array[4 * (j + width * i) + 0] = b_color[i,j]; // kolor czerwony
                    image_array[4 * (j + width * i) + 1] = g_color[i,j]; // kolor zielony
                    image_array[4 * (j + width * i) + 2] = r_color[i,j]; // kolor czerwony
                }
            }

            // zapisanie danych do pliku z odpowiednimi ustawieniami
            await SaveToFile(image_array, file_name, CreationCollisionOption.ReplaceExisting, Bi
tmapPixelFormat.Bgra8, alpha_mode);
        }

        public async Task ComparePictures(byte[,] r_master, byte[,] g_master, byte[,] b_master,
byte[,] r_slave, byte[,] g_slave, byte[,] b_slave , string heatmap_file_name, string scoremap_fi
le_name, byte rt, byte gt, byte bt, byte filter_background = 1, byte edge_threshold = 7, bool is
_simplified = false, bool black_background = false)
        {
            // sprawdzanie czy tablice mają te same długości - jeśli nie to jest nie zgodności r
ozdzielczości obrazów
            if ((r_master.Length + b_master.Length + g_master.Length) == (b_slave.Length + g_sla
ve.Length + r_slave.Length))
            {
                // zapis wymiarów obrazu do zmiennych
                int width = r_master.GetLength(0);
```

```
                int height = r_master.GetLength(1);

                // deklaracja nowych tablic dwuwymiarowych
                BackgroundArray = new bool[width, height];
                byte[,] r_difference = new byte[width, height];
                byte[,] g_difference = new byte[width, height];
                byte[,] b_difference = new byte[width, height];
                byte[,] difference = new byte[width, height];
                byte[,] score = new byte[width, height];

                // resetowania liczników i wyniku
                GoodPixelCount = 0;
                BadPixelCount = 0;
                BackgroundPixelCount = 0;
                Score = 0;

                // funkcja sprawdzająca tło
                if (filter_background == 1)
                    CheckBackground(r_master, g_master, b_master, r_slave, g_slave, b_slave, edg
e_threshold,is_simplified, black_background);

                // obliczanie wartości bezwzględnej z różnicy pomiędzy danymi kolorami w obrazac
h
                    for (int i = 0; i < ImageHeight; i++)
                {
                    for (int j = 0; j < ImageWidth; j++)
                    {
                        r_difference[i, j] = (byte)Math.Abs(r_master[i, j] - r_slave[i, j]);
                        g_difference[i, j] = (byte)Math.Abs(g_master[i, j] - g_slave[i, j]);
                        b_difference[i, j] = (byte)Math.Abs(b_master[i, j] - b_slave[i, j]);
                        difference[i, j] = (byte)((r_difference[i, j] + g_difference[i, j] + b_d
ifference[i, j]) / 3);

                        // jeżeli z funkcji CheckBackground wykryło, że w tym miejsu jest piksel
 to przypisz wartość dwa do tablicy score
                    if (BackgroundArray[i,j] == true)
                        { score[i, j] = 2; BackgroundPixelCount++; }

                    // jeżeli różnica kolorów mieści się w tolerancji to przypisz wartość jeden
do tablicy score
                    else if (r_difference[i, j] <= rt && g_difference[i, j] <= gt && b_differenc
e[i, j] <= bt)
                        { score[i, j] = 1; GoodPixelCount++; }

                    // jeśli różnica ponad tolerancje przypisz wartość 0 do tablicy score
                    else
                        { score[i, j] = 0; BadPixelCount++; }
                    }
                }

                // oblicza wynik
                Score = ((100*GoodPixelCount) / (GoodPixelCount + BadPixelCount));

                // funkcje do stworzenia mapy cieplnej i wynikowej
                await HeatMap(difference, heatmap_file_name);
                await ScoreMap(score, scoremap_file_name);
            }
            else
                ErrorCode = "Nie zgodność rozdzielczości obrazów"; // można zastosować dodatkowo
 ShowMessage
        }

        private void CheckBackground(byte[,] r_master, byte[,] g_master, byte[,] b_master, byte[
,] r_slave, byte[,] g_slave, byte[,] b_slave, byte edge_threshold, bool is_simplified, bool blac
k_background)
        {
            // zapis wymiarów obrazu do zmiennych
            int width = r_master.GetLength(0);
            int height = r_master.GetLength(1);

            // ustawienie tolerancji tła
            byte tr = 192;
            if (is_simplified == true)
                tr = 170;
```

```
            byte tg = 170;
            byte tb = 128;
            double threshold = 90; // próg przejścia

            // start funkcji mierzącej szybkość działania
            var watch = new Stopwatch();
            watch.Start();

            // jeżeli tło jest czarne - używane dla algorytm detekcji krawędzi
            if (black_background == true)
            {
                byte t = 128;
                for (int i = 0; i < width; i++)
                {
                    for (int j = 0; j < height; j++)
                    {

                        // algorytm zaczyna się po deadzone tab aby nie wyjść po zakres tabeli
                        if (((i > 1 * edge_threshold) && (i < width - 1 * edge_threshold)) && ((
j > 1 * edge_threshold) && (j < height - 1 * edge_threshold)))
                        {
                            uint background_counter = 0;
                            uint foreground_counter = 0;

                            // wykonanie algorytmu sprawdzającego w postaci krzyża-
efektywnie 8 kierunków
                            // i nalicza ile wyników ile jest tła jeśli poszczególny kolor jest
poniżej tolerancji tła
                            // to algorytm ustawia sprawdzany piksel jako tło
                            for (int k = (-edge_threshold); k <= edge_threshold; k++)
                            {
                                if (k == 0)
                                    continue;
                                if (r_master[i + k, j + k] < t && r_slave[i + k, j + k] < t && g
_master[i + k, j + k] < t && g_slave[i + k, j + k] < t && b_master[i + k, j + k] < t && b_slave[
i + k, j + k] < t)
                                    background_counter++;
                                else foreground_counter++;

                                if (r_master[i + k, j - k] < t && r_slave[i + k, j - k] < t && g
_master[i + k, j - k] < t && g_slave[i + k, j - k] < t && b_master[i + k, j + k] < t && b_slave[
i + k, j - k] < t)
                                    background_counter++;
                                else foreground_counter++;

                                if (r_master[i + k, j + 0] < t && r_slave[i + k, j + 0] < t && g
_master[i + k, j + 0] < t && g_slave[i + k, j + 0] < t && b_master[i + k, j + 0] < t && b_slave[
i + k, j + 0] < t)
                                    background_counter++;
                                else foreground_counter++;

                                if (r_master[i + 0, j + k] < t && r_slave[i + 0, j + k] < t && g
_master[i + 0, j + k] < t && g_slave[i + 0, j + k] < t && b_master[i + 0, j + k] < t && b_slave[
i + 0, j + k] < t)
                                    background_counter++;
                                else foreground_counter++;
                            }
                            if (100 * background_counter / (foreground_counter + background_coun
ter) > threshold)
                                BackgroundArray[i, j] = true;
                            else
                                BackgroundArray[i, j] = false;
                        }
                        else
                            BackgroundArray[i, j] = true;
                    }
                }
            }

            // algorytm dla tła białego - jedyna różnica jest w znaku
        // nalicza ile wyników ile jest tła jeśli poszczególny kolor jest powyżej tolerancji tła
            else
            {
                for (int i = 0; i < width; i++)
```

```
            {
                for (int j = 0; j < height; j++)
                {

                    if (((i > 1 * edge_threshold) && (i < width - 1 * edge_threshold)) && ((
j > 1 * edge_threshold) && (j < height - 1 * edge_threshold)))
                    {
                        uint background_counter = 0;
                        uint foreground_counter = 0;

                        for (int k = (-edge_threshold); k <= edge_threshold; k++)
                        {
                            if (k == 0)
                                continue;
                            if (r_master[i + k, j + k] >= tr && r_slave[i + k, j + k] >= tr
&& g_master[i + k, j + k] >= tg && g_slave[i + k, j + k] >= tg && b_master[i + k, j + k] >= tb &
& b_slave[i + k, j + k] >= tb)
                                background_counter++;
                            else foreground_counter++;

                            if (r_master[i + k, j - k] >= tr && r_slave[i + k, j - k] >= tr
&& g_master[i + k, j - k] >= tg && g_slave[i + k, j - k] >= tg && b_master[i + k, j + k] >= tb &
& b_slave[i + k, j - k] >= tb)
                                background_counter++;
                            else foreground_counter++;

                            if (r_master[i + k, j + 0] >= tr && r_slave[i + k, j + 0] >= tr
&& g_master[i + k, j + 0] >= tg && g_slave[i + k, j + 0] >= tg && b_master[i + k, j + 0] >= tb &
& b_slave[i + k, j + 0] >= tb)
                                background_counter++;
                            else foreground_counter++;

                            if (r_master[i + 0, j + k] >= tr && r_slave[i + 0, j + k] >= tr
&& g_master[i + 0, j + k] >= tg && g_slave[i + 0, j + k] >= tg && b_master[i + 0, j + k] >= tb &
& b_slave[i + 0, j + k] >= tb)
                                background_counter++;
                            else foreground_counter++;
                        }
                        if (100 * background_counter / (foreground_counter + background_coun
ter) > threshold)
                            BackgroundArray[i, j] = true;
                        else
                            BackgroundArray[i, j] = false;
                    }
                    else
                        BackgroundArray[i, j] = true;
                }
            }
        }

        // koniec funkcji sprawdzającej szybkość działania i wyświetlenie wartości w debuggerze
            watch.Stop();
            System.Diagnostics.Debug.WriteLine("Time elapsed: " + watch.Elapsed);
        }

        private async Task HeatMap(byte[,] difference , string heatmap_file_name)
        {
            // zapis wymiarów obrazu do zmiennych
            int width = difference.GetLength(0);
            int height = difference.GetLength(1);

            // stworzenie tablicy jednowymiarowej o długości (rozdzielczość*4)
            // ponieważ każdy piksel to 4 bajty - odpowiednio B, G, R, A
            byte[] image_array = new byte[width * height * 4];

            // im większa różnica między pikselami to odejmuje od zielonej mapy kolor zielony z
mnożnikiem 1, a dodaje
// kolor czerwony z mnożnikiem 5. TryToByte jest po to aby nie wyjść poza zakres bajta (0...255)
            for (int i = 0; i < width; i++)
            {
                for (int j = 0; j < height; j++)
                {
                        image_array[4 * (j + MaxLength * i) + 0] = 0;
```

```
                        image_array[4 * (j + MaxLength * i) + 1] = TryToByte(255, (difference[i,
 j]), 1, false);
                        image_array[4 * (j + MaxLength * i) + 2] = TryToByte(0, (difference[i, j
]), 5, true);
                        image_array[4 * (j + MaxLength * i) + 3] = 255;
                }
            }

            // zapisanie danych do pliku z odpowiednimi ustawieniami
            await SaveToFile(image_array, heatmap_file_name, CreationCollisionOption.ReplaceExis
ting, BitmapPixelFormat.Bgra8,BitmapAlphaMode.Ignore);
        }

        private byte TryToByte(byte var1, byte var2, byte multiplier, bool is_addition)
        {
            // zwraca wartość między o a 255
            int val = 0;
            if (is_addition == false)
                val = var1 - multiplier * var2;
            else
                val = var1 + multiplier * var2;
            if (val < 0)
                return 0;
            else if (val > 255)
                return 255;
            else
                return Convert.ToByte(val);
        }

        public async Task ScoreMap(byte[,] score , string scoremap_file_name)
        {
            // zapis wymiarów obrazu do zmiennych
            int width = score.GetLength(0);
            int height = score.GetLength(1);

            // stworzenie tablicy jednowymiarowej o długości (rozdzielczość*4)
            // ponieważ każdy piksel to 4 bajty - odpowiednio B, G, R, A
            byte[] image_array = new byte[width * height * 4];

            // jeśli score[i,j] = 1 (czyli zgodny piksel) to ustaw kolor zielony
            // jeśli score[i,j] = 0 (czyli niezgodny piksel) to ustaw kolor czerwony
            // jeśli score[i,j] = 2 (czyli piksel jest tłem) to ustaw kolor niebieski
            for (int i = 0; i < width; i++)
            {
                for (int j = 0; j < height; j++)
                {
                    if (score[i, j] == 1)
                    {
                        image_array[4 * (j + width * i)+0] = 0;
                        image_array[4 * (j + width * i) + 1] = 255;
                        image_array[4 * (j + width * i) + 2] = 0;
                        image_array[4 * (j + width * i) + 3] = 255;
                    }
                    else if (score[i, j] == 0)
                    {
                        image_array[4 * (j + width * i) + 0] = 0;
                        image_array[4 * (j + width * i) + 1] = 0;
                        image_array[4 * (j + width * i) + 2] = 255;
                        image_array[4 * (j + width * i) + 3] = 255;
                    }
                    else
                    {
                        image_array[4 * (j + width * i) + 0] = 255;
                        image_array[4 * (j + width * i) + 1] = 0;
                        image_array[4 * (j + width * i) + 2] = 0;
                        image_array[4 * (j + width * i) + 3] = 255;
                    }
                }
            }

            // zapisanie danych do pliku z odpowiednimi ustawieniami
            await SaveToFile(image_array, scoremap_file_name, CreationCollisionOption.ReplaceExi
sting, BitmapPixelFormat.Bgra8, BitmapAlphaMode.Ignore);
        }
```

```
        public async Task<StorageFile> SaveToFile(byte[] image_array, string file_name, Creation
CollisionOption collision, BitmapPixelFormat image_format, BitmapAlphaMode alpha_mode)
        {
            // stworzenie nowej bitmapy
            WriteableBitmap image = new WriteableBitmap(ImageWidth, ImageHeight);

            // użycie using upewnia ze strumień danych zostanie zakończony po skończonej operacj
i
            // przypisanie do strumienia danych ciągu bajtów o długości wszystkich pikseli * 4 (
bo 4 bajty na piksel)
            using (Stream image_stream = image.PixelBuffer.AsStream())
            {
                await image_stream.WriteAsync(image_array, 0, ImageHeight * ImageWidth * 4);
            }

            // stworzenie pliku w folderze Obrazy o zadanej nazwie i parametrze
            // w przypadku wystąpienia pliku o tej samej nazwie
            var file = await KnownFolders.PicturesLibrary.CreateFileAsync(file_name, collision);

            // użycie using upewnia ze strumień danych zostanie zakończony po skończonej operacj
i
            // przypisanie do strumienia danych operacji otwarcia pliku
            using (IRandomAccessStream image_stream = await file.OpenAsync(FileAccessMode.ReadWr
ite))
            {
                // enkodowanie zdjęcia z utworzonego strumienia danych
                BitmapEncoder encoder = await BitmapEncoder.CreateAsync(BitmapEncoder.BmpEncoder
Id, image_stream);
                Stream pixel_stream = image.PixelBuffer.AsStream();

                // stworzenie ciągu bajtów pikseli o długości strumienia danych
                byte[] pixel_array = new byte[pixel_stream.Length];

                // odczyt danych
                await pixel_stream.ReadAsync(pixel_array, 0, pixel_array.Length);

                // enkodowania obrazu o poniższych parametrach
                encoder.SetPixelData(image_format, alpha_mode, // format i kanał alfa
                                     (uint)image.PixelWidth, // szerokość obrazu
                                     (uint)image.PixelHeight, // wysokość obrazu
                                     96.0, // DPI w szerokości
                                     96.0, // DPI w wysokości
                                     pixel_array); // strumień bajtów do zapisu
                await encoder.FlushAsync(); // zakończenie enkodowania
            }
            return file; // zwrócenie pliku
        }

        public async Task Resize(StorageFolder storage_location, string image_location, byte sca
le, string new_image_location)
        {
            // odczytanie istniejącego pliku o zadanej nazwie i lokalizacji
            StorageFile file = await storage_location.GetFileAsync(image_location);

            // użycie using upewnia ze strumień danych zostanie zakończony po skończonej operacj
i
            // przypisanie do strumienia danych operacji otwarcia pliku
            using (IRandomAccessStream image_stream = await file.OpenAsync(FileAccessMode.ReadWr
ite))
            {
                // dekodowanie zdjęcia do oczytania jego wymiarów
                var decoder = await BitmapDecoder.CreateAsync(image_stream);

                // żądana rozdzielczość
                int target_width = (int)decoder.PixelWidth * scale / 100;
                int target_height = (int)decoder.PixelHeight * scale / 100;

                // stworzenie nowego strumienia pamięci
                var memory_stream = new InMemoryRandomAccessStream();

                // enkodowanie zdjęcia z utworzonego strumienia pamięci
                BitmapEncoder encoder = await BitmapEncoder.CreateForTranscodingAsync(memory_str
eam, decoder);
```

```
                // obliczanie współczynnika skalującego
                double ratio_width = (double)target_width / decoder.PixelWidth;
                double ratio_height = (double)target_height / decoder.PixelHeight;
                double ratio_scale = Math.Min(ratio_width, ratio_height);

                // zabezpieczenie przed zerową żądaną rozdzielczością
                if (target_width == 0)
                    ratio_scale = ratio_height;

                if (target_height == 0)
                    ratio_scale = ratio_width;

                // przeskalowane wymiary zdjęcia
                uint scaled_height = (uint)Math.Floor(decoder.PixelHeight * ratio_scale);
                uint scaled_width = (uint)Math.Floor(decoder.PixelWidth * ratio_scale);

                // interpolacja liniowa bitmapy do przeskalowanych wymiarów
                encoder.BitmapTransform.InterpolationMode = BitmapInterpolationMode.Linear;
                encoder.BitmapTransform.ScaledHeight = scaled_height;
                encoder.BitmapTransform.ScaledWidth = scaled_width;

                // czyści bufory i strumienia i przygotowanie do zapisu
                await encoder.FlushAsync();

                // startowa pozycja strumienia pamięci
                memory_stream.Seek(0);

                // zapis do tablicy pikseli z strumienia pamięci o zadanej długości ARGB
                var pixel_array = new byte[memory_stream.Size];

                // odczyt do strumienia pamięci parametrów przeskalowanego obrazu
                await memory_stream.ReadAsync(pixel_array.AsBuffer(), (uint)memory_stream.Size,
InputStreamOptions.None);

                // zapis zdjęcia do nowej żądanej lokalizacji (nazwy)
                StorageFile new_file = await storage_location.CreateFileAsync(new_image_location
, CreationCollisionOption.ReplaceExisting);
                await FileIO.WriteBytesAsync(new_file, pixel_array);
            }
        }

  public async Task Rotate(StorageFolder storage_location, string image_location, uint rotation)
        {
            // funkcja do obracania obrazu o 90, 180 lub 270 stopni
            StorageFile file = await storage_location.GetFileAsync(image_location);
            using (IRandomAccessStream image_stream = await file.OpenAsync(FileAccessMode.ReadWr
ite),
                                                memory_stream = new InMemoryRandomAccessStream())
            {
                BitmapDecoder decoder = await BitmapDecoder.CreateAsync(image_stream);
                BitmapEncoder encoder = await BitmapEncoder.CreateForTranscodingAsync(memory_str
eam, decoder);
                if (rotation == 90)
                    encoder.BitmapTransform.Rotation = BitmapRotation.Clockwise90Degrees;
                else if (rotation == 180)
                    encoder.BitmapTransform.Rotation = BitmapRotation.Clockwise180Degrees;
                else if (rotation == 270)
                    encoder.BitmapTransform.Rotation = BitmapRotation.Clockwise270Degrees;
                else
                {
                    ErrorCode = "Można tylko obracać o 90, 180 i 270 stopni";
                    goto end_method;
                }
                await encoder.FlushAsync();
                memory_stream.Seek(0);
                image_stream.Seek(0);
                image_stream.Size = 0;
                await RandomAccessStream.CopyAsync(memory_stream, image_stream);
            end_method:;
            }
        }
    }
}
```

Dodatek D: Kod źródłowy klasy EdgeDetection

```
using System;
using System.Threading.Tasks;
using Windows.Storage;
using Windows.Graphics.Imaging;
using System.IO;
using Windows.UI.Xaml.Media.Imaging;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Storage.Streams;

namespace PCB_Visual_Inspection_v2
{
    class EdgeDetection
    {
        // gotowe macierze dla operatorów Sobela, Prewitta i Kirscha

        private static  double[,] xSobel
        {
            get
            {
                return new double[,]
                {
                    { -1, 0, 1 },
                    { -2, 0, 2 },
                    { -1, 0, 1 }
                };
            }
        }

        private static double[,] ySobel
        {
            get
            {
                return new double[,]
                {
                    {  1,  2,  1 },
                    {  0,  0,  0 },
                    { -1, -2, -1 }
                };
            }
        }

        private static double[,] xPrewitt
        {
            get
            {
                return new double[,]
                { { -1,  0,  1, },
                  { -1,  0,  1, },
                  { -1,  0,  1, }, };
            }
        }

        private static double[,] yPrewitt
        {
            get
            {
                return new double[,]
                { {  1,  1,  1, },
                  {  0,  0,  0, },
                  { -1, -1, -1, }, };
            }
        }

        private static double[,] xKirsch
        {
            get
            {
                return new double[,]
                { {  5,  5,  5, },
                  { -3,  0, -3, },
                  { -3, -3, -3, }, };
```

```
            }
        }

        private static double[,] yKirsch
        {
            get
            {
                return new double[,]
                { {  5, -3, -3, },
                  {  5,  0, -3, },
                  {  5, -3, -3, }, };
            }
        }

        public async Task ConvolutionFilter(StorageFolder storage_location, string image_locatio
n, byte edge_method, byte only_strong_edges, string output_file_name, bool grayscale = true)
        {
            // jądra przekształcenia dla danego kierunku
            double[,] xkernel = new double[3, 3];
            double[,] ykernel = new double[3, 3];

            // w zależności od żądanej metody - przypisz odpowiednie macierze
            if (edge_method == 0)
            {
                xkernel = xSobel;
                ykernel = ySobel;
            }
            else if (edge_method == 1)
            {
                xkernel = xPrewitt;
                ykernel = yPrewitt;
            }
            else if (edge_method == 2)
            {
                xkernel = xKirsch;
                ykernel = yKirsch;
            }

            // dekodowanie strumienia danych z konkretnego pliku (przesłanego w zmiennych)
            StorageFile file = await storage_location.GetFileAsync(image_location);
            Stream image_stream = await file.OpenStreamForReadAsync();
            BitmapDecoder decoder = await BitmapDecoder.CreateAsync(image_stream.AsRandomAccessS
tream());

            // zapis wymiarów obrazu do zmiennych
            int height = Convert.ToInt32(decoder.PixelHeight);
            int width = Convert.ToInt32(decoder.PixelWidth);

            // stworzenie ciągów bajtów do przechowywania informacji o pikseli
            // 1 piksel = 4 bajty (dla każdego kanału A,R,G,B)
            var data = await decoder.GetPixelDataAsync();
            byte[] pixel_array = data.DetachPixelData();
            byte[] result_array = new byte[height * width * 4];

            // przekształcenie obrazu do odcieni szarości ITU-R BT.709 "Luma coefficients"
            if (grayscale == true)
            {
                float rgb = 0;
                for (int i = 0; i < pixel_array.Length; i += 4)
                {
                    rgb = pixel_array[i] * 0.2126f;
                    rgb += pixel_array[i + 1] * 0.7152f;
                    rgb += pixel_array[i + 2] * 0.0722f;
                    pixel_array[i] = (byte)rgb;
                    pixel_array[i + 1] = pixel_array[i];
                    pixel_array[i + 2] = pixel_array[i];
                    pixel_array[i + 3] = 255;
                }
            }

            // stworzenie zmiennych dla każdego koloru piksela i dla każdego jądra przekształcen
ia (w tym algorytmie 2 kierunki)
            double xr = 0.0;
            double xg = 0.0;
```

```
                double xb = 0.0;
                double yr = 0.0;
                double yg = 0.0;
                double yb = 0.0;

                // końcowe wartości kolorów w danym pikselu
                double r = 0.0;
                double g = 0.0;
                double b = 0.0;

                // border to odległość środkowego piksela macierzy do granicy jądra
                // wszystkie operatory są macierzami 3x3 więc odległość od środka do granicy to 1
                int border = 1;
                int n = 0; // jeden bajt w macierzy jądra przekształcenia
                int k = 0; // jeden bajt w ciągu bajtów obrazu

                // algorytm uwzględnia deadzone tak aby jądro przekształcenia
                // nie wychodziło poza zakres tabeli przy wykonywaniu działań
                for (int j = border; j < height - border; j++)
                {
                    for (int i = border; i < width - border; i++)
                    {
                        // resetowania wartości RGB
                        xr = xg = xb = yr = yg = yb = 0;
                        r = g = b = 0.0;

                        // pozycja środkowego piksela w jądrze przekształcenia w odniesieniu
                        // do jednowymiarowego ciągu bajtów danego zdjęcia
                        k = j * width*4 + i * 4;

                        // nakładanie jądra przekształcenia (filtru, maski) na obraz
                        for (int y = -border; y <= border; y++)
                        {
                            for (int x = -border; x <= border; x++)
                            {
                                n = k + x * 4 + y * width*4;
                                xb += (double)(pixel_array[n]) * xkernel[y + border, x + border];
                                xg += (double)(pixel_array[n + 1]) * xkernel[y + border, x + border]
;
                                xr += (double)(pixel_array[n + 2]) * xkernel[y + border, x + border]
;
                                yb += (double)(pixel_array[n]) * ykernel[y + border, x + border];
                                yg += (double)(pixel_array[n + 1]) * ykernel[y + border, x + border]
;
                                yr += (double)(pixel_array[n + 2]) * ykernel[y + border, x + border]
;
                            }
                        }

                        // obliczanie gradientu dla wartości RGB
                        b = Math.Sqrt((xb * xb) + (yb * yb));
                        g = Math.Sqrt((xg * xg) + (yg * yg));
                        r = Math.Sqrt((xr * xr) + (yr * yr));

                        // ustawienie limitów aby nie przekraczać wartości 1 bajta (0...255)
                        if (b > 255) b = 255;
                        else if (b < 0) b = 0;
                        if (g > 255) g = 255;
                        else if (g < 0) g = 0;
                        if (r > 255) r = 255;
                        else if (r < 0) r = 0;

                        // zapisywanie informacji do nowego ciągu bajtów
                        // jeśli chcemy tylko twarde kontury to zostawiamy w obrazie tylko czysty bi
ały kolor
                        if (only_strong_edges == 1)
                        {
                            if (b < 255 & g < 255 & r <255)
                            {
                                result_array[k] = 0;
                                result_array[k + 1] = 0;
                                result_array[k + 2] = 0;
                                result_array[k + 3] = 255;
                            }
```

```
                    else
                    {
                        result_array[k] = (byte)(b);
                        result_array[k + 1] = (byte)(g);
                        result_array[k + 2] = (byte)(r);
                        result_array[k + 3] = 255;
                    }
                }
                else
                {
                    result_array[k] = (byte)(b);
                    result_array[k + 1] = (byte)(g);
                    result_array[k + 2] = (byte)(r);
                    result_array[k + 3] = 255;
                }
            }
        }

            // stworzenie nowej bitmapy
            WriteableBitmap image = new WriteableBitmap(width, height);

            // użycie using upewnia ze strumień danych zostanie zakończony po skończonej operacj
i
            // przypisanie do strumienia danych ciągu bajtów o długości wszystkich pikseli * 4 (
bo 4 bajty na piksel)
            using (image_stream = image.PixelBuffer.AsStream())
            {
                await image_stream.WriteAsync(result_array, 0, width * height * 4);
            }

            // zapisanie danych do pliku z odpowiednimi ustawieniami
            await SaveToFile(image, output_file_name, CreationCollisionOption.ReplaceExisting, B
itmapPixelFormat.Bgra8, BitmapAlphaMode.Ignore);

        }

        public async Task<StorageFile> SaveToFile(WriteableBitmap image, string file_name, Windo
ws.Storage.CreationCollisionOption collision, BitmapPixelFormat image_format, BitmapAlphaMode al
pha_mode)
        {
            // stworzenie pliku w folderze Obrazy o zadanej nazwie i parametrze
            // w przypadku wystąpienia pliku o tej samej nazwie
            var file = await KnownFolders.PicturesLibrary.CreateFileAsync(file_name, collision);

            // użycie using upewnia ze strumień danych zostanie zakończony po skończonej operacj
i
            // przypisanie do strumienia danych operacji otwarcia pliku
            using (IRandomAccessStream image_stream = await file.OpenAsync(FileAccessMode.ReadWr
ite))
            {
                // enkodowanie zdjęcia z utworzonego strumienia danych
                BitmapEncoder encoder = await BitmapEncoder.CreateAsync(BitmapEncoder.BmpEncoder
Id, image_stream);
                Stream pixel_stream = image.PixelBuffer.AsStream();

                // stworzenie ciągu bajtów pikseli o długości strumienia danych
                byte[] pixel_array = new byte[pixel_stream.Length];

                // odczyt danych
                await pixel_stream.ReadAsync(pixel_array, 0, pixel_array.Length);

                // enkodowania obrazu o poniższych parametrach
                encoder.SetPixelData(image_format, alpha_mode, // format i kanał alfa
                                     (uint)image.PixelWidth, // szerokość obrazu
                                     (uint)image.PixelHeight, // wysokość obrazu
                                     96.0, // DPI w szerokości
                                     96.0, // DPI w wysokości
                                     pixel_array); // strumień bajtów do zapisu
                await encoder.FlushAsync(); // zakończenie enkodowania
            }
            return file; // zwrócenie pliku
        }
    }
}
```

Dodatek E: Funkcja wykonująca test w kodzie programu głównego

```
private async void Button_Click_1(object sender, RoutedEventArgs e) // Wykonaj test
{
    // wyzerowania wyników poszczególnych algorytmów
    score_algorithm1 = 0;
    score_algorithm2 = 0;
    score_algorithm3 = 0;

    // pobranie ostatnich zapisanych parametrów z SecondPage
    byte red_tolerance = tolerance_settings[0];
    byte green_tolerance = tolerance_settings[1];
    byte blue_tolerance = tolerance_settings[2];
    byte which_pcb = tolerance_settings[3];
    byte threshold = tolerance_settings[4];
    byte simplify_ratio = tolerance_settings[5];
    byte edge_method = tolerance_settings[6];
    byte strong_edges = tolerance_settings[7];
    byte filter_background = tolerance_settings[8];
    byte scale = tolerance_settings[9];

    // wyłączenie możliwości wciskania przycisków na czas działania funkcji
    SwitchControls(false);

    // jeżeli zostały przesłane parametry ze strony SeondPage to je przypisz
    // wartość może być null w momencie przejścia ze strony ThirdPage do MainPage
    // dlatego stworzono tabele tolerance_settings, która zapisuje ostatnie przesłane parametry
    if (tolerance != null)
    {
        which_pcb = tolerance.PCB;
        threshold = tolerance.Threshold;
        simplify_ratio = tolerance.Simplifier;
        red_tolerance = tolerance.Red;
        green_tolerance = tolerance.Green;
        blue_tolerance = tolerance.Blue;
        edge_method = tolerance.EdgeMethod;
        strong_edges = tolerance.StrongEdges;
        filter_background = tolerance.FilterBackground;
        scale = tolerance.Scale;
    }

    // jeśli włączony jest podgląd kamery to wykonaj zdj i na czas kontroli wizyjnej podgląd
    // zostaje wyłączony i zastąpiony obrazkiem wskazującym, która płytka PCB jest testowana
    if (Camera.Capture != null)
    {
        await Camera.TakePicture("TEST.bmp");
        Camera.Dispose();
        if (which_pcb == 1)
            TakenPhoto.Source = new BitmapImage(new Uri("ms-appx:///Assets/ArduinoUNO.png"));
        if (which_pcb == 2)
            TakenPhoto.Source = new BitmapImage(new Uri("ms-appx:///Assets/BoardUNO.png"));
        if (which_pcb == 3)
            TakenPhoto.Source = new BitmapImage(new Uri("ms-appx:///Assets/CustomBoard.png"));
        TakenPhoto.Visibility = Visibility.Visible;
    }

    // wywołanie funkcji GetPixel i przypisanie wartości tablic kolorów do tablic płytki test
    await ProcessImage.GetPixel(storage_pictures, "TEST.BMP", scale, "TEST-resized.BMP");
    r_test = ProcessImage.RedColorArray;
    g_test = ProcessImage.GreenColorArray;
    b_test = ProcessImage.BlueColorArray;

    // w zależności od wybranej PCB i skalowania zostają przypisane odpowiednie wartości
    // do tablic kolorów danego PCB, a następnie porównanie tablic test z PCBx.
    if (which_pcb == 1)
    {
        if (scale == 25)
            await ProcessImage.GetPixel(storage_package, @"Assets\PCB1-resized25.BMP");
        else if (scale == 50)
            await ProcessImage.GetPixel(storage_package, @"Assets\PCB1-resized50.BMP");
        else
```

```
            await ProcessImage.GetPixel(storage_package, @"Assets\PCB1.BMP");
        r_pcb1 = ProcessImage.RedColorArray;
        g_pcb1 = ProcessImage.GreenColorArray;
        b_pcb1 = ProcessImage.BlueColorArray;
        await ProcessImage.ComparePictures(r_pcb1, g_pcb1, b_pcb1, r_test, g_test, b_test, "Heat
Map.BMP", "ScoreMap.BMP", red_tolerance, green_tolerance, blue_tolerance, filter_background);
    }
    if (which_pcb == 2)
    {
        if (scale == 25)
            await ProcessImage.GetPixel(storage_package, @"Assets\PCB2-resized25.BMP");
        else if (scale == 50)
            await ProcessImage.GetPixel(storage_package, @"Assets\PCB2-resized50.BMP");
        else
            await ProcessImage.GetPixel(storage_package, @"Assets\PCB2.BMP");
        r_pcb2 = ProcessImage.RedColorArray;
        g_pcb2 = ProcessImage.GreenColorArray;
        b_pcb2 = ProcessImage.BlueColorArray;
        await ProcessImage.ComparePictures(r_pcb2, g_pcb2, b_pcb2, r_test, g_test, b_test, "Heat
Map.BMP", "ScoreMap.BMP", red_tolerance, green_tolerance, blue_tolerance, filter_background);
    }
    if (which_pcb == 3)
    {
        await ProcessImage.GetPixel(storage_pictures, "PCB3.BMP", scale, "PCB3-resized.BMP");
        r_pcb3 = ProcessImage.RedColorArray;
        g_pcb3 = ProcessImage.GreenColorArray;
        b_pcb3 = ProcessImage.BlueColorArray;
        await ProcessImage.ComparePictures(r_pcb3, g_pcb3, b_pcb3, r_test, g_test, b_test, "Heat
Map.BMP", "ScoreMap.BMP", red_tolerance, green_tolerance, blue_tolerance, filter_background);
    }

    // przypisanie wyników z obiektu ProcessImage (klasa ImageProcessing)
    // do pomocniczych później wykorzystanych zmiennych i parametrów dla ThirdPage
    score_algorithm1 = ProcessImage.Score;
    images.ResultBackgroundRaw = ProcessImage.BackgroundPixelCount;
    images.ResultGoodPixelRaw = ProcessImage.GoodPixelCount;
    images.ResultBadPixelRaw = ProcessImage.BadPixelCount;

    // jeśli zaznaczony algorytm uproszczony to wykonaj funkcje SetPixel dla płytki test i PCBx.
    if (algorithm2 == true)
    {
        await ProcessImage.SetPixel(r_test, g_test, b_test, "Slave_Simplified.BMP", simplify_rat
io);

        if (which_pcb == 1)
            await ProcessImage.SetPixel(r_pcb1, g_pcb1, b_pcb1, "Master_Simplified.BMP", simplif
y_ratio);
        if (which_pcb == 2)
            await ProcessImage.SetPixel(r_pcb2, g_pcb2, b_pcb2, "Master_Simplified.BMP", simplif
y_ratio);
        if (which_pcb == 3)
            await ProcessImage.SetPixel(r_pcb3, g_pcb3, b_pcb3, "Master_Simplified.BMP", simplif
y_ratio );

        // przypisywanie do tablic kolorów i późniejsze porównanie ich
        await ProcessImage.GetPixel(storage_pictures, "Slave_Simplified.BMP");
        r_slave_simply = ProcessImage.RedColorArray;
        g_slave_simply = ProcessImage.GreenColorArray;
        b_slave_simply = ProcessImage.BlueColorArray;

        await ProcessImage.GetPixel(storage_pictures, "Master_Simplified.BMP");
        r_master_simply = ProcessImage.RedColorArray;
        g_master_simply = ProcessImage.GreenColorArray;
        b_master_simply = ProcessImage.BlueColorArray;

        await ProcessImage.ComparePictures(r_master_simply, g_master_simply, b_master_simply, r_
slave_simply, g_slave_simply, b_slave_simply, "SimplifiedHeatMap.BMP", "SimplifiedScoreMap.BMP",
 0, 0, 0, filter_background, 7, true);

        // zapis do zmiennych pomocniczych
        score_algorithm2 = ProcessImage.Score;
        images.ResultBackgroundSimplified = ProcessImage.BackgroundPixelCount;
        images.ResultGoodPixelSimplified = ProcessImage.GoodPixelCount;
        images.ResultBadPixelSimplified = ProcessImage.BadPixelCount;
```

```
    }

    // jeśli zaznaczony algorytm krawędziowy to wykonaj fukcje ConvolutionFilter dla płytki test
 i
    // PCBx z odpowiednim skalowaniem
    if (algorithm3 == true)
    {
        if (which_pcb == 1)
        {
            if (scale == 25)
                await EdgeDetector.ConvolutionFilter(storage_package, @"Assets\PCB1-
resized25.bmp", edge_method, strong_edges, "Master_Edge.bmp");
            else if (scale == 50)
                await EdgeDetector.ConvolutionFilter(storage_package, @"Assets\PCB1-
resized50.bmp", edge_method, strong_edges, "Master_Edge.bmp");
            else
                await EdgeDetector.ConvolutionFilter(storage_package, @"Assets\PCB1.bmp", edge_m
ethod, strong_edges, "Master_Edge.bmp");
        }

        if (which_pcb == 2)
        {
            if (scale == 25)
                await EdgeDetector.ConvolutionFilter(storage_package, @"Assets\PCB2-
resized25.bmp", edge_method, strong_edges, "Master_Edge.bmp");
            else if (scale == 50)
                await EdgeDetector.ConvolutionFilter(storage_package, @"Assets\PCB2-
resized50.bmp", edge_method, strong_edges, "Master_Edge.bmp");
            else
                await EdgeDetector.ConvolutionFilter(storage_package, @"Assets\PCB2.bmp", edge_m
ethod, strong_edges, "Master_Edge.bmp");
        }

        if (which_pcb == 3)
        {
            if (scale != 100)
                await EdgeDetector.ConvolutionFilter(storage_pictures, "PCB3-
resized.bmp", edge_method, strong_edges, "Master_Edge.bmp");
            else
                await EdgeDetector.ConvolutionFilter(storage_pictures, "PCB3.bmp", edge_method,
strong_edges, "Master_Edge.bmp");
        }

        // przypisywanie do tablic kolorów płytki PCBx
        await ProcessImage.GetPixel(storage_pictures, "Master_Edge.BMP");
        r_master_edge = ProcessImage.RedColorArray;
        g_master_edge = ProcessImage.GreenColorArray;
        b_master_edge = ProcessImage.BlueColorArray;

        // dla płytki test
        if (scale != 100)
            await EdgeDetector.ConvolutionFilter(storage_pictures, "TEST-
resized.bmp", edge_method, strong_edges, "Slave_Edge.bmp");
        else
            await EdgeDetector.ConvolutionFilter(storage_pictures, "TEST.bmp", edge_method, stro
ng_edges, "Slave_Edge.bmp");

        // przypisywanie do tablic kolorów płytki test
        await ProcessImage.GetPixel(storage_pictures, "Slave_Edge.BMP");
        r_slave_edge = ProcessImage.RedColorArray;
        g_slave_edge = ProcessImage.GreenColorArray;
        b_slave_edge = ProcessImage.BlueColorArray;

        // porównanie tablic kolorów płytki test do PCBx.
        await ProcessImage.ComparePictures(r_master_edge, g_master_edge, b_master_edge, r_slave_
edge, g_slave_edge, b_slave_edge, "EdgeHeatMap.BMP", "EdgeScoreMap.BMP", red_tolerance, green_to
lerance, blue_tolerance, filter_background, 1, false, true);

        // zapis do zmiennych pomocniczych
        score_algorithm3 = ProcessImage.Score;
        images.ResultBackgroundEdge = ProcessImage.BackgroundPixelCount;
        images.ResultGoodPixelEdge = ProcessImage.GoodPixelCount;
        images.ResultBadPixelEdge = ProcessImage.BadPixelCount;
    }
```

```
        // przesyłanie parametrów do obiektu images (klasa Pictures), który później
        // posłuży do przesłania ich do strony ThirdPage
        images.PCB = which_pcb;

        if (algorithm1 == true)
        {
            images.ResultRaw = score_algorithm1;
            images.AlgorithmRaw = true;
        }
        else images.AlgorithmRaw = false;

        if (algorithm2 == true)
        {
            images.AlgorithmSimplified = true;
            images.ResultSimplified = score_algorithm2;
        }
        else images.AlgorithmSimplified = false;

        if (algorithm3 == true)
        {
            images.AlgorithmEdge = true;
            images.ResultEdge = score_algorithm3;
        }
        else images.AlgorithmEdge = false;

        images.IsTested = true;

        // wywołanie funkcji wyświelającej nad podglądem wyników poszczególnych algorytmów
        // i odblokowanie możliwości sterowania przyciskami w programie
        ShowResult(score_algorithm1, score_algorithm2, score_algorithm3, threshold);
        SwitchControls(true);

        // na koniec funkcji - przywrócenie podglądu z kamery
        if (Camera.Capture == null && device_info.Count > 0)
        {
            TakenPhoto.Visibility = Visibility.Collapsed;
            TakenPhoto.Source = null;
            await Camera.Initialize(screen_rotation);
        }
    }
```