Temat:	Malowanie obrazów biorąc pod uwagę ruch ciała		
Wykonali:	Aleksandra Czerniawska Michał Klempka Wojciech Duda	Grupa:	TI I2

SPIS TREŚCI

1. Wstęp

Cyfrowe przetwarzanie obrazu to gałąź informatyki która jest rozwijana od lat 60 XX wieku. Przez ostatnie lata, dało się zauważyć spory rozwój w tej dziedzinie. Przy przetwarzaniu pojedynczego obrazu czas przetwarzania obrazu nie gra zbyt dużej roli, jednak przy przetwarzaniu obrazu w czasie rzeczywistym sytuacja ta diametralnie się zmienia: czas przetwarzania jest ważnym czynnikiem który wpływa na wydajność aplikacji. Dzisiejsza moc komputerów pozwala jendak na wykonanie wielu obliczeń w czasie rzeczywistym, co pozwala na całkiem niezłą analizę otrzymywanego obrazu, nawet na komputerach osobistych, czy smartfonach. Dlatego też technologia przetwarzania obrazu w czasie rzeczywistym, cieszy się coraz większą popularnością. Możliwe stają się rozwiązania, które zmieniają dotychczasowy sposób obsługi programów (klawiatura oraz mysz), na bardziej naturalny dla człowieka - korzystając przy tym z niedrogich wbudowanych kamer, które są obecne w prawie każdym nowym laptopie, czy komórce. W naszym projekcie staraliśmy się umożliwić kontrolę, prostego programu do rysowania, przetwarzając obraz użytkownika w czasie rzeczywistym.

2. Opis zadania

Zaimplementowanie aplikacji wychwytującej obraz z kamery, umożliwiającej malowanie na podstawie ruchów, oraz gestów użytkownika. Aplikacja ma za zadanie określać wykonywane ruchy przez użytkownika i przekształcać je na obraz wyświetlany na ekranie, który można potem zapisać do pliku .jpg.

3. Uzasadnienie podjęcia tematu

Spośród przedstawionych tematów, temat rysowania za pomocą ruchu ciała wykrywanego przez kamerę wydał się nam najbardziej interesujący. Powody dlaczego uważamy ten temat za warty zainteresowania:

- Możliwości rozwoju podobnych projektów, wraz z rozwojem technologi przetwarzania obrazu, duży potencjał omawianej technologii
- Powszechność kamer, prawie każdy posiada kamerkę internetową

- Wiele możliwości użycia podobnych technologii - przykładowo:
    - możliwość wykorzystania w VR (oculus, google carboard) jako kontroler,
    - wykorzystanie do gry zręczościowej (gry sportowe, takie jak np. na konsolę Wii, korzystające z kamery)
- Możliwość wykorzystania przez osoby niepełnosprawne, które mają trudności z posługiwaniem się klawiaturą i myszą (malowanie za pomocą śledzenia ruchu głowy, bądź w bardziej zaawanasowanych wersjach ruchu gałek ocznych)

4. Zastosowane metodyki pracy w grupie

Połączenie Agile - Metodyki zwinnej z programowaniem ekstremalnym Scrum. Opracowaliśmy harmonogram z uwzględnieniem dwutygodniowych sprintów.

Użyliśmy systemu kontroli wersji Git, wybraliśmy serwis hostingowy Github który umożliwia trzymanie repozytoriów Gita. Po ukończeniu każdego sprintu, łączyliśmy powstałe moduły za pomocą tego narzędzia.

W zespole komunikowaliśmy się głównie przez portal społecznościowy Facebook, ale komunikacja odbywała się również na Uczelni.

5. Harmonogram i podział pracy:

Działanie:	Osoby:			T1	T2	T3	T4	T5	T6
	Aleksandra Czerniawska	Michał Klempka	Wojciech Duda						
Wybór tematu									
Wybór narzędzi, oraz ich konfiguracja									
Ustalenie metody pracy									
Zapoznanie się z metodami przechwycenia obrazu z kamerki w OpenCV									
Wstępny podział zadań									
Konfiguracja Visual Studio z OpenCV									
Implementacja pierwszej wersji wykrywania obiektów za pomocą detekcji kolorów i nasycenia									
Umożliwienie rysowania w interfejsie OpenCV									
Rysowanie okręgów w miejscach, gdzie wykrywamy obiekt, oznaczanie krawędzi									
Zapoznanie się z możliwościami tworzenia interfejsu w QT, konfiguracja środowiska									
Poprawiony podział zadań									

Spotkanie organizacyjne - zmiana sposobu wykrywania na kaskady na Haar'a									
Implementacja wykrywania zamkniętej dłoni przez kaskady Haar'a									
Interfejs zmiany grubości pędzla, oraz resetowania płótna									
Interfejs zmiany kolorów pędzla									
Interfejs zmiany grubości pędzla									
Detekcja koloru skóry									
Zapis do pliku									
Dodanie większej ilości kaskad, możliwość przełączania się pomiędzy różnymi trybami za pomocą klawiatury									
Testy i wybranie optymalnych kaskad									
Poprawki wykrywania częsci ciała									
Dodanie opcji gumki									
Prace nad interfejsem QT									
Poprawki interfejsu, zapisywanie, korzystając tylko z wykrywania kamerą									
Wstępna refaktoryzacja kodu									

Końcowa refaktoryzacja kodu									
Ukończenie dokumentacji									
Poprawki i testy, spotkanie organizacyjne									
Oddanie ukończonego projektu									

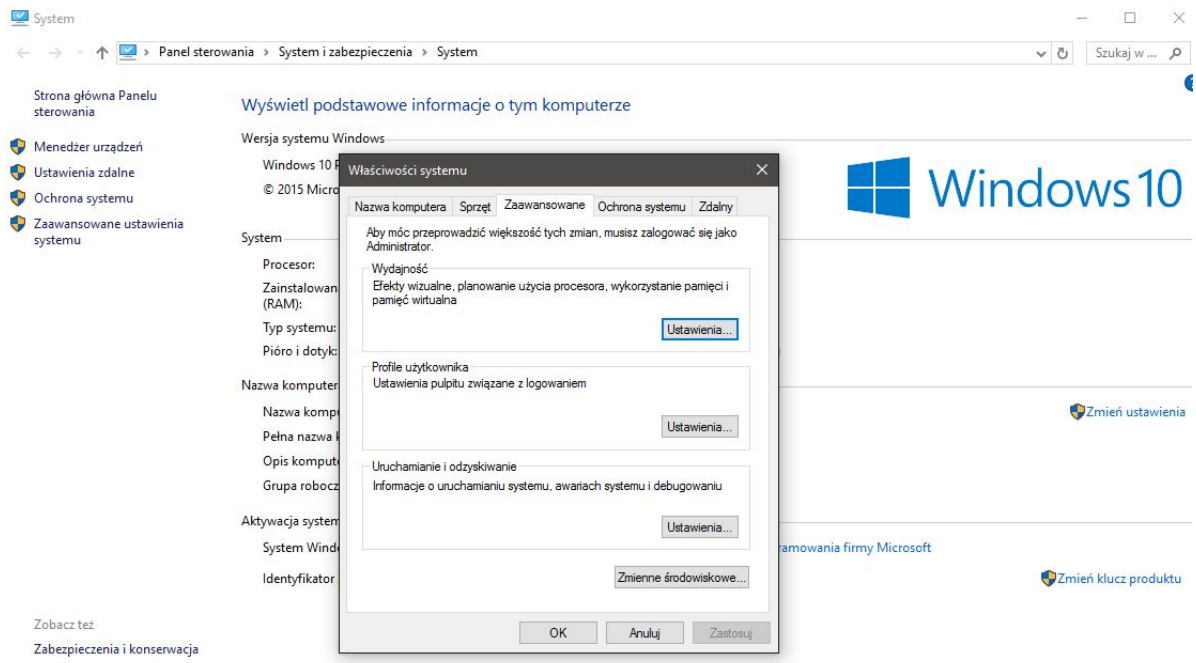
6. Wybrane technologie

- OpenCV 2.4.12
- Język C++
- Środowisko Visual Studio 2013
- QT 5.6 wraz z Qt Creator 3.6.1
- Kaskady Haar'a do wykrywania położenia interesujących nas częsci ciała

7. Konfiguracja Visual Studio 2013 z OpenCV

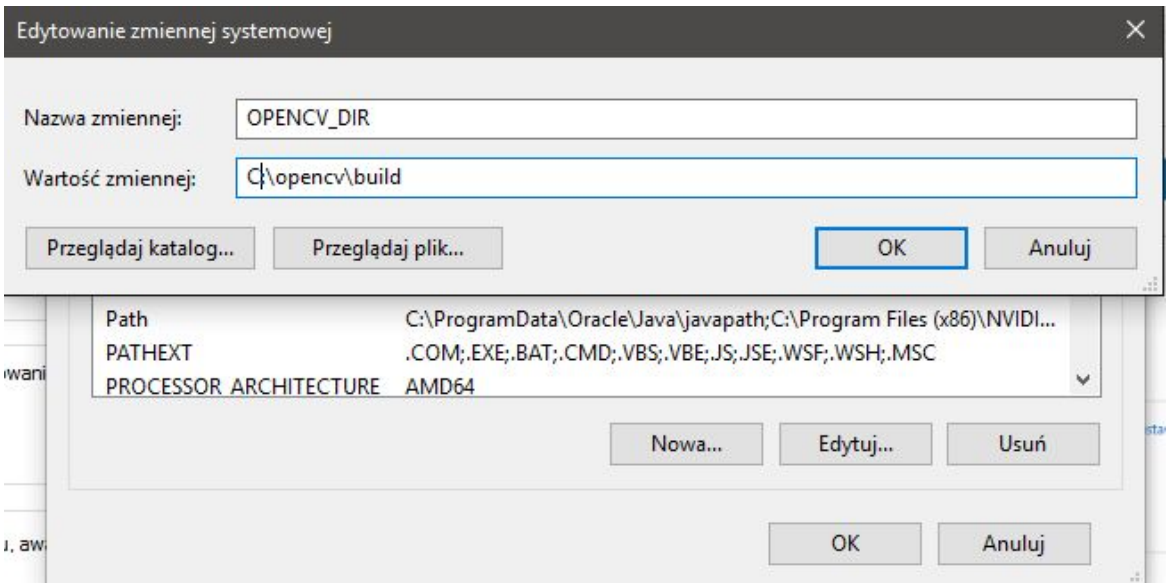
Konfiguracja OpenCV z Visual Studio na systemie Windows, wbrew pozorom potrafiła sprawić trochę trudności, dlatego przedstawimy krótką instrukcję jak bezproblemowo wykonać tę konfigurację na każdej maszynie:

1) Zaczynamy od pobrania potrzebnej nam wersji OpenCV ze strony: http://opencv.org/downloads.html
2) Wypakowujemy pobrane pliki do nowego folderu opencv na dysku C:\
3) Dla późniejszej wygody możemy utworzyć ścieżkę dostępu (PATH), w tym celu udajemy się do: zaawansowanych właściwości systemu i wybieramy zmienne środowiskowe



Obraz 1. Zmienne środowiskowe.

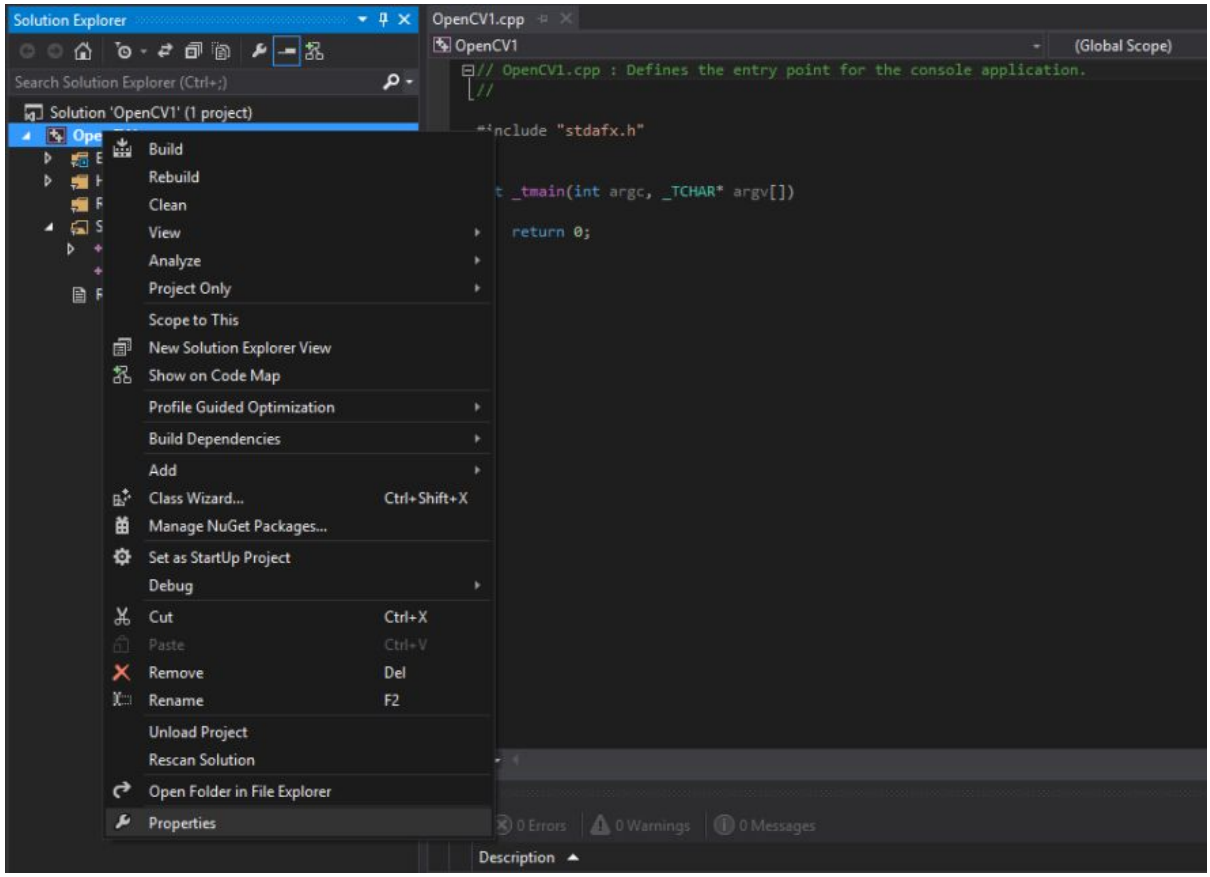
4) Dodajemy nową zmienną środowiskową OPENCV_DIR, o wartości: C:\opencv\build, możemy też dodać taką zmienną dla danego użytkownika, w niektórych konfiguracjach rozwiązuje to problem ze znalezieniem ścieżki.



Obraz 2. Tworzenie zmiennej środowiskowej.

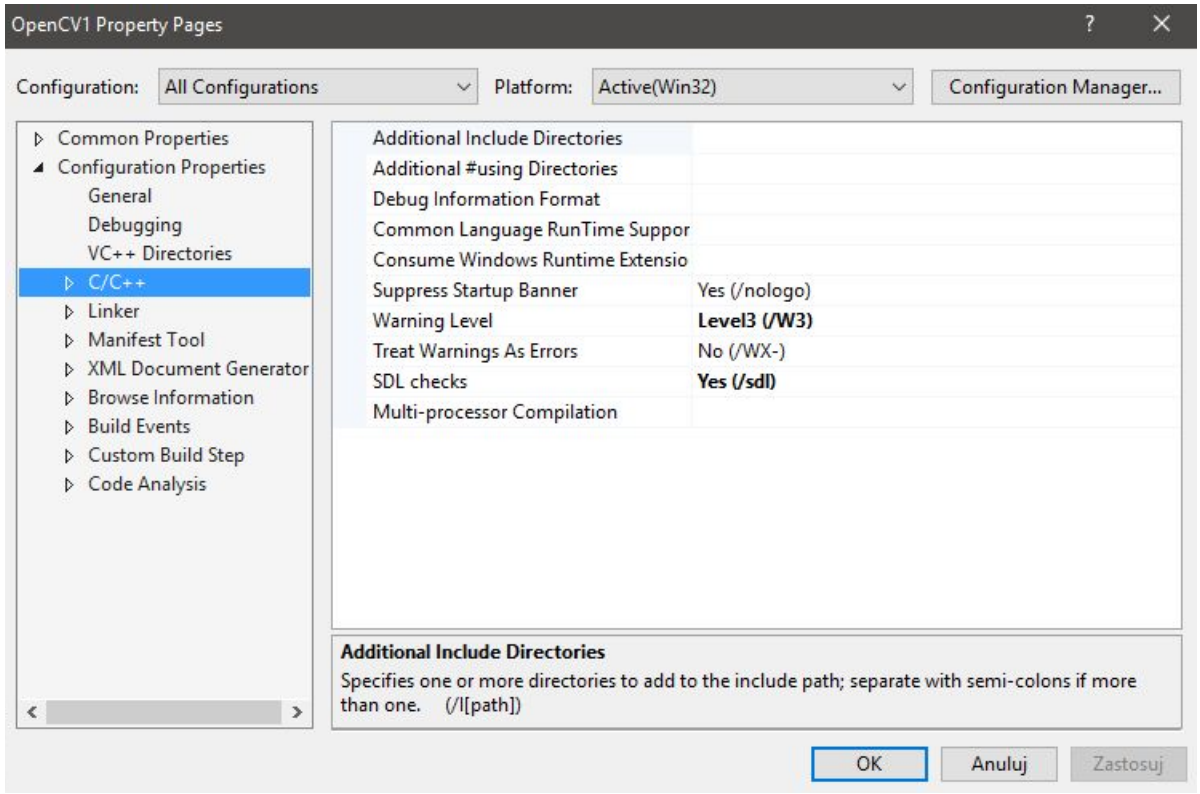
5) Tworzymy nowy projekt C++ w Visual Studio

6) Klikamy prawym przyciskiem na nasz projekt w Solution Explorer i wybieramy Properties:



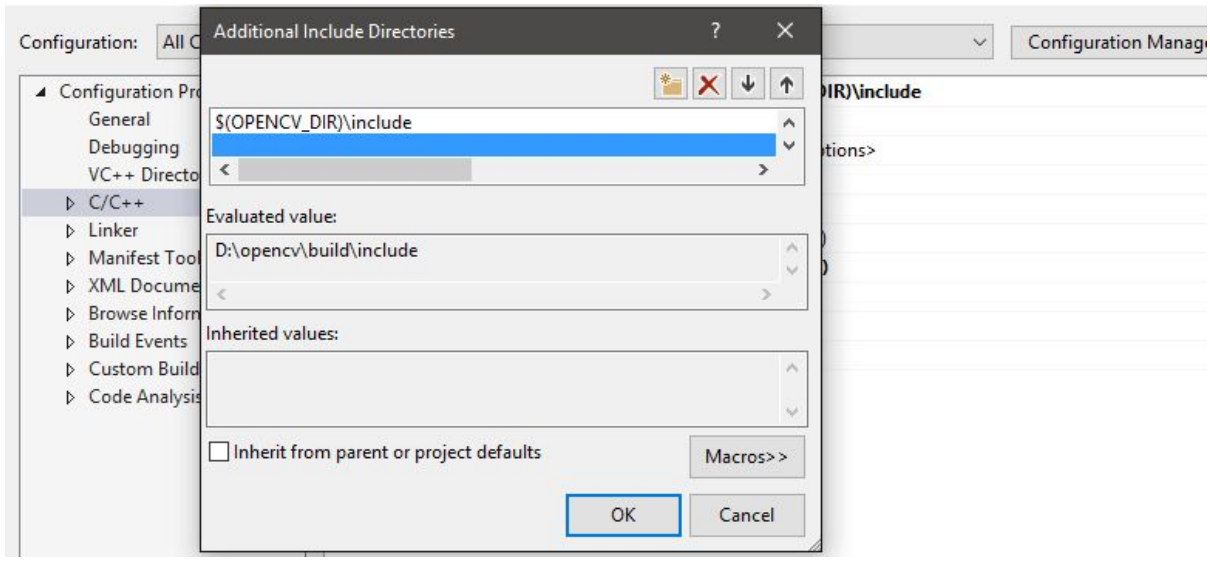
Obraz 3. Właściwości projektu.

7) Ustawiamy konfigurację na Wszystkie konfiguracje (All configurations), jeśli korzystamy z OpenCV w wersji 64 bit to zmieniamy platformę na 64bit, jeśli nie zostawiamy tak jak w obrazku niżej. W naszym wypadku korzystaliśmy z wersji 32bit, oraz kompilatora vc12, co jest ważne w punktach niżej.



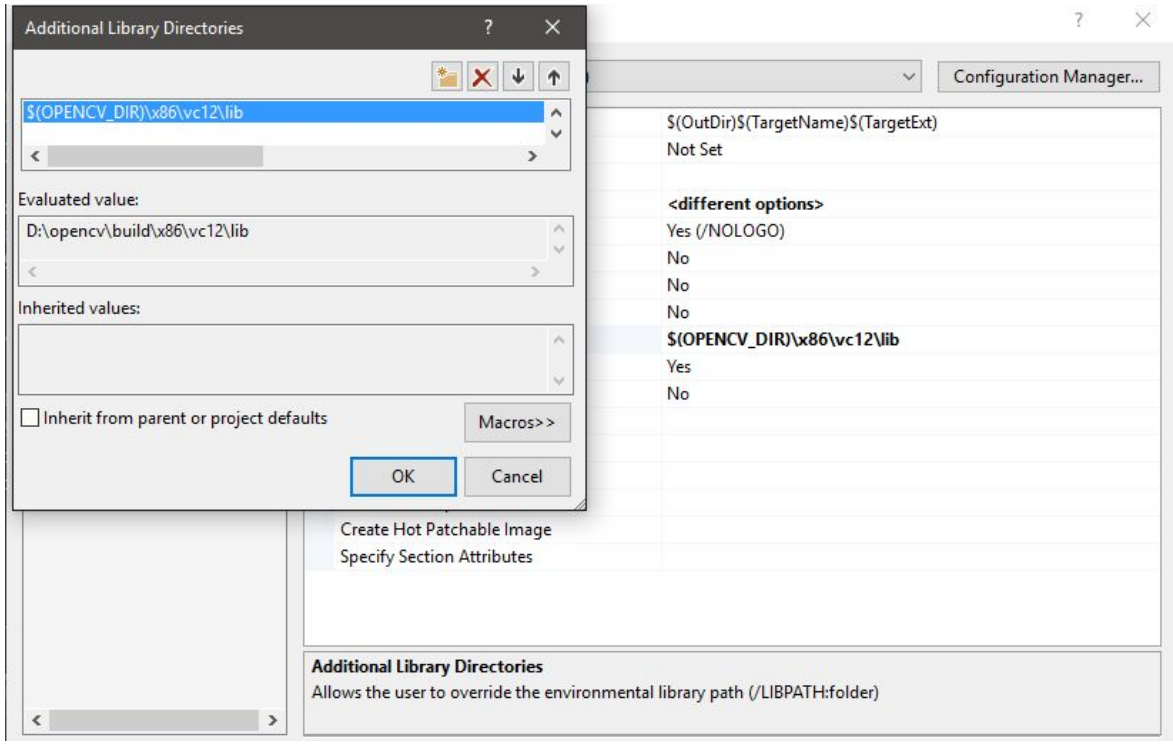
Obraz 4. Konfiguracja platformy.

8) Przechodzimy do zakładki C/C++ / General / Additional Include Directories, wskazujemy ścieżkę do folderów: $(OPENCV_DIR)\include



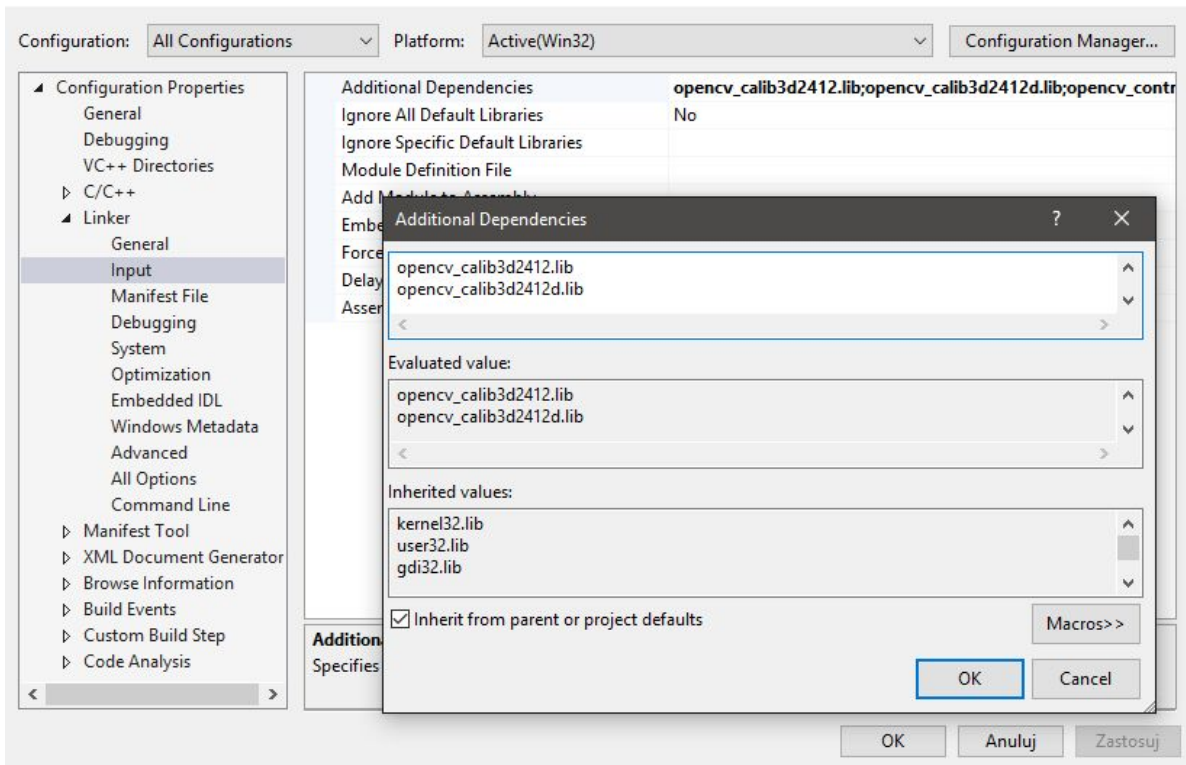
Obraz 5. Additional Include Directories.

9) Przechodzimy do zakładki Linker / General / Additional Library Directories i podajemy ścieżkę: $(OPENCV_DIR)\x86\vc12\lib



Obraz 6. Additional Library Directories.

10)Przechodzimy do zakładki Linker / Input / Additional Dependencies (podajemy wszystkie pliki z lokalizacji C:\opencv\build\x86\vc12\lib\)



Obraz 7. Additional Dependencies.

Lista plików, które dodajemy do Additional Dependencies:

- opencv_calib3d2412.lib
- opencv_calib3d2412d.lib
- opencv_contrib2412.lib
- opencv_contrib2412d.lib
- opencv_core2412.lib
- opencv_core2412d.lib
- opencv_features2d2412.lib
- opencv_features2d2412d.lib
- opencv_flann2412.lib
- opencv_flann2412d.lib
- opencv_gpu2412.lib
- opencv_gpu2412d.lib
- opencv_highgui2412.lib
- opencv_highgui2412d.lib
- opencv_imgproc2412.lib
- opencv_imgproc2412d.lib
- opencv_legacy2412.lib
- opencv_legacy2412d.lib
- opencv_ml2412.lib
- opencv_ml2412d.lib
- opencv_nonfree2412.lib
- opencv_nonfree2412d.lib

Pliki z liteką ‘d’ na końcu, dodajemy dla trybu Debug, bez dla Release.

11) Po takiej konfiguracji możemy dodać do projektu potrzebne nam bilbioteki takie jak np. <opencv2/opencv.hpp>, projekt uruchamiamy w trybie release.

8. Konfiguracja QT 5.6 z OpenCV

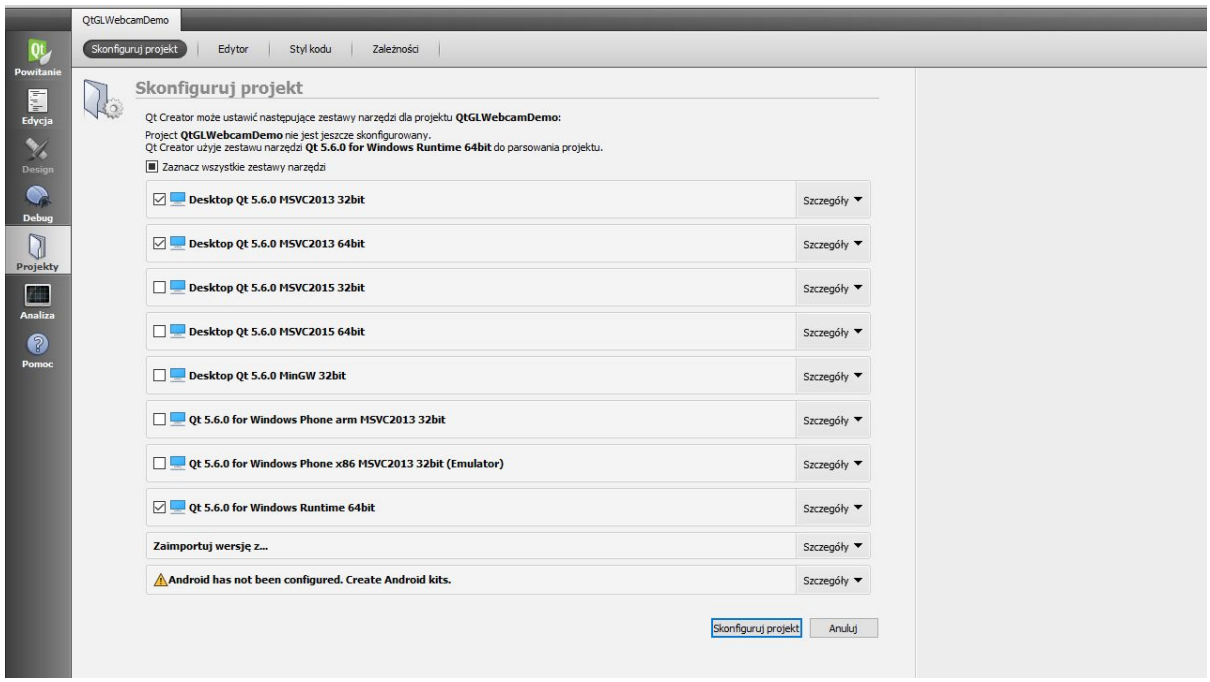
QT posiada wiele przydatnych narzędzi do tworzenia interfejsu, co skłoniło nas do próby przeportowania obecnych implementacji z Visual Studio, do QT Creatora, co niestety nie zostało ukończone, ale udało się nam skonfigurować QT do współpracy z OpenCV, oraz do wyświetlania obrazu w specjalnym widgecie, przedstawimy krótką instrukcję jak skonfigurować QT do tego zadania.

1) Tak samo jak dla Visual Studio, zaczynamy od pobrania potrzebnej nam wersji OpenCV ze strony: http://opencv.org/downloads.html

2) Wypakowujemy pobrane pliki do nowego folderu opencv na dysku C:\

3) Pobieramy demo projektu ze skonfigurowanym widgetem OpenCV dla QT ze strony: https://github.com/Myzhar/QtOpenCVViewerGl

4) Uruchamiamy pobrany projekt, po uruchomieniu konfigurujemy, z wybranymi zestawami narzędzi, w naszym wypadku jest to MSVC2013 32bit



Obraz 8. Konfiguracja projektu w QT Creator.

5) Edytujemy plik NazaProjektu.pro dodając ścieżkę do bibliotek:
OPENCV_PATH = C:/opencv
LIBS_PATH = "$$OPENCV_PATH/build/x86/vc12/lib"

6) Dodajemy nazwy bibliotek w takim formacie:

```
CONFIG(debug, debug|release) {
  LIBS     += -L$$LIBS_PATH \
              -lopencv_core2412d \
              -lopencv_highgui2412d \
              -lopencv_calib3d2412d \
              -lopencv_contrib2412d \
              -lopencv_core2412d \
              -lopencv_features2d2412d \
              -lopencv_flann2412d \
```

```
        -lopencv_gpu2412d \
        -lopencv_highgui2412d \
        -lopencv_imgproc2412d \
        -lopencv_legacy2412d \
        -lopencv_ml2412d \
        -lopencv_nonfree2412d \
        -lopencv_objdetect2412d \
        -lopencv_ocl2412d \
        -lopencv_photo2412d \
        -lopencv_stitching2412d \
        -lopencv_superres2412d \
        -lopencv_ts2412d \
        -lopencv_video2412d \
        -lopencv_videostab2412d
}
```

Przykład plików dla konfiguracji debug.

7) Po zakończeniu edycji pliku .pro uruchamiamy qmake, oraz rebuildujemy nasz projekt. Jeśli wszystko zostało wykonanie poprawnie możemy rozpocząć pracę, nad interfejsem.

9. Kaskada Haar’a

Wykrywanie dłoni w OpenCV realizowane jest za pomocą klasyfikatora kaskad Haar'a (Haar Cascade Classifier). Wykrywacz ten, na podanym obrazku przeprowadza analizę każdej części obrazka i klasyfikuje ją jako zawierającą dłoń bądź nie. Dłonie na obrazku mogą być większe lub mniejsze, co w praktyce oznacza, że algorytm klasyfikatora musi być wykonany kilkakrotnie na danym obrazku, aby w końcu mógł wyodrębnić poszukiwane cechy. Klasyfikator działa na podstawie danych zapisanych w pliku XML, w którym to znajdują się definicje poszukiwanych obiektów.

Biblioteka OpenCV udostępnia wiele wytrenowanych wcześniej klasyfikatorów. Zapisywane są one w formacie XML. W wykorzystywanej przez nas wersji biblioteki znajdują się między innymi:

haarcascade_frontalface_default.xml
haarcascade_mcs_eyepair_big.xml
haarcascade_mcs_nose.xml
haarcascade_frontalface_alt.xml

Powyższe klasyfikatory wytrenowane są w wyszukiwaniu twarzy oraz jej części składowych takich jak uszy oczy itd. W naszej implementacji użyliśmy również kaskad wykrywających otwartą dłoń, oraz pięść.

Do wytrenowania własnego klasyfikatora Haar'a potrzebne są dwa zbiory danych. Pierwszy zbiór to pozytywy czyli obrazy przedstawiające tylko interesujący nas obiekt. Kolejny to zbór negatywny - przykłady obrazów przedstawiających rzeczy inne niż interesujący nas obiekt - które reprezentują tło obiektu. Zbiory uczące powinny zawierać kilka tysięcy przykładów. Zgromadzenie takich zbiorów może wymagać bardzo dużo czasu, ale przykładowo dla twarzy istnieją gotowe bazy danych. Do budowania zbiorów uczących służy polecenie opencv_createsamples.
Następnym krokiem jest trening klasyfikatora. Służy do tego polecenie opencv_haartraining.

Po tej operacji można utworzyć plik XML. Służy do tego polecenie convert_cascade, znajdujące się w katalogu. Tak przygotowany klasyfikator, może być już użyty w programie.

10. Przypadki użycia oraz diagramy UML

“Diagram 1: przypadki użycia” jest to diagram UML przestawiający funkcjonalności naszego systemu.

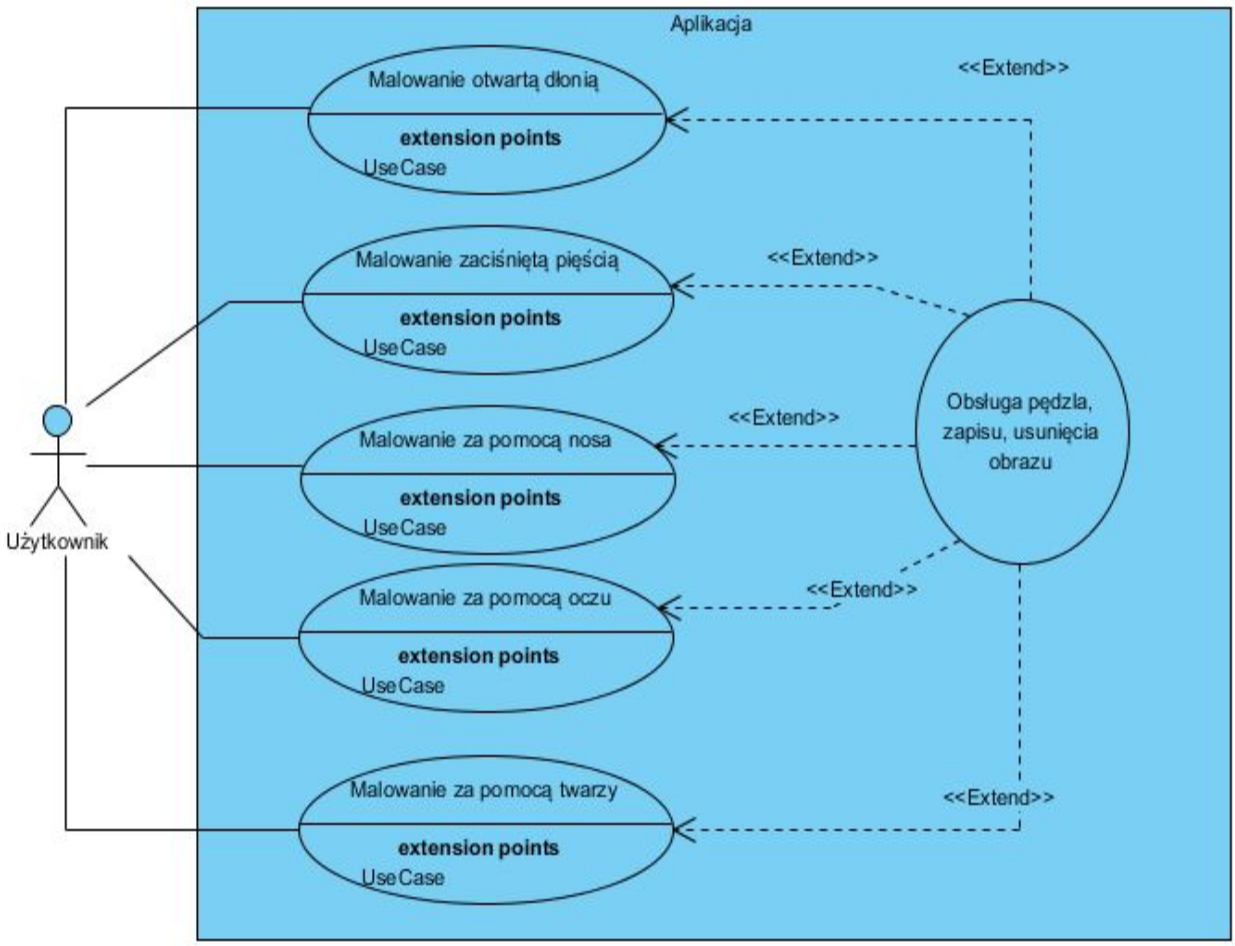


Diagram 1. Przypadki użycia.

Diagram 2: czynności opisuje kolejne kroki jakie może wykonać użytkownik podczas korzystania z aplikacji.

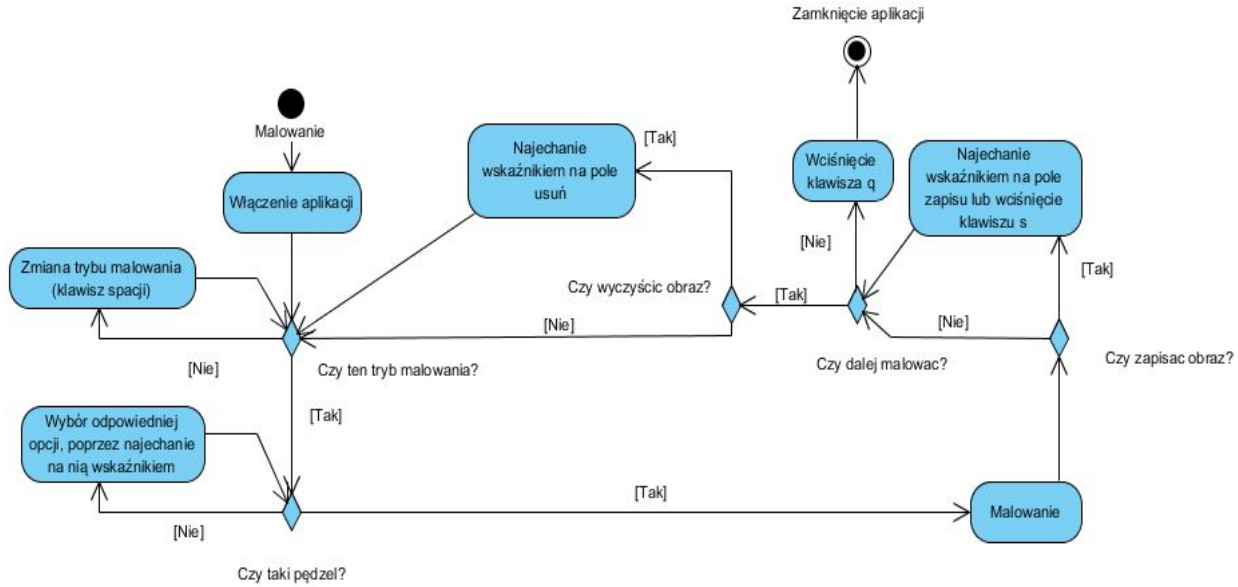


Diagram 2. czynności.

11. Wykorzystane biblioteki

W projekcie wykorzystujemy następujące biblioteki:

- opencv2/opencv.hpp
- iostream
- vector
- algorithm
- iomanip
- ctime

12. Kluczowe fragmenty kodu wraz z opisem

Funkcje:

- **void CreateTextButton(Mat& frame,Point p1, Point p2, Scalar s, std::string text, Point text_p)**
  funkcja za pomocą której tworzone są przyciski z tekstem, jako element interfejsu

- **void CreateLineButton(Mat& frame, Point r_p1, Point r_p2, Point l_p1, Point l_p2, int lineType)**
  funkcja za pomocą której tworzone są przyciski z linią o różnej grubości, jako element interfejsu

- **void DrawCircles(Mat& frame, std::vector<Rect>& hands, cv::Rect& maxRect, int& posX, int& posY)**
  funkcja wykorzystywana do rysowania kółka na wykrytym obiekcie

- **void DisplayMessage(Mat& frame, Point p1, std::string text)**
  funkcja do wyświetlania komunikatów w głównym oknie programu

- **void SaveFile(Mat& frame)**
  funkcja która zapisuje do pliku obecny obraz

- **int main(int argc, char *argv[])**
  główna funkcja programu w której znajduje się m. in. główna pętla przetwarzania obrazu oraz wykrywanie obiektów

Kluczowa metoda:

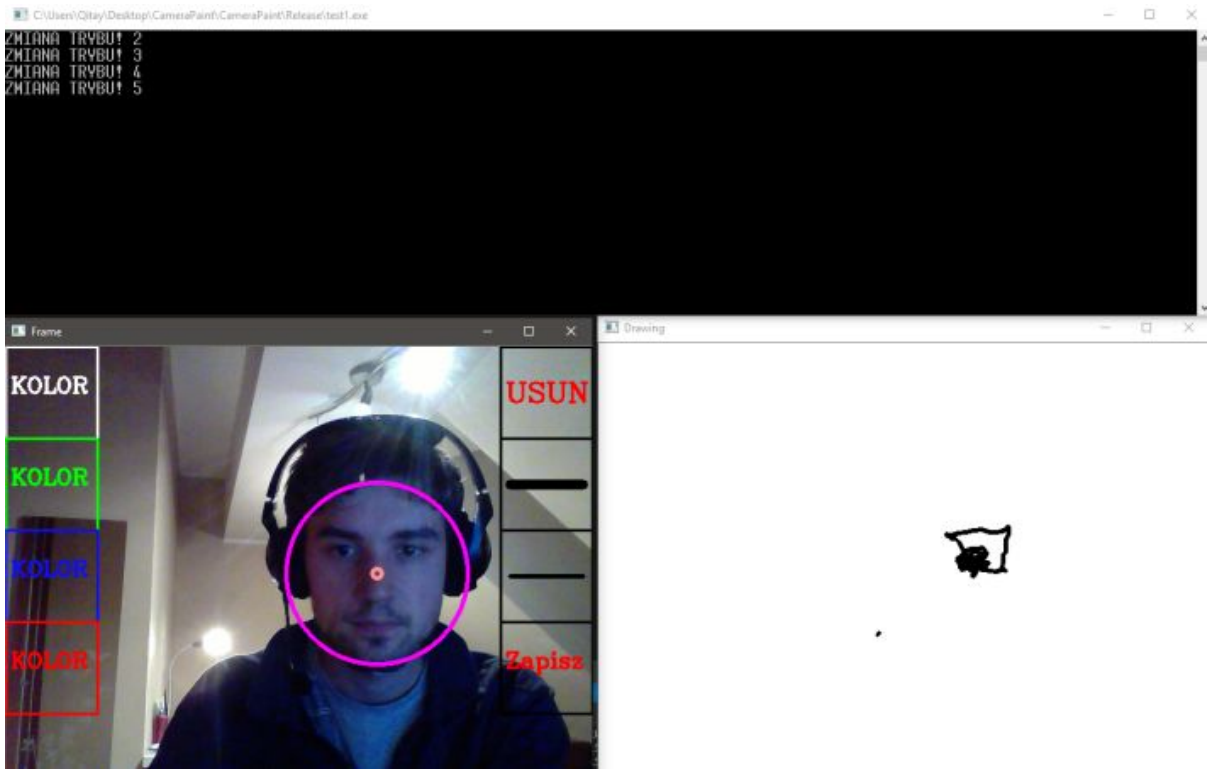
hand_cascade.**detectMultiScale**(detection, hands, 1.1, 2, 0 | CV_HAAR_FIND_BIGGEST_OBJECT, Size(50, 50), Size(300, 300));
Wykrywa obiekty o różnych rozmiarach w obrazie wejściowym. Wykryte obiekty są zwracane w postaci listy prostokątów.

13. Instrukcja obsługi, opis interfejsu

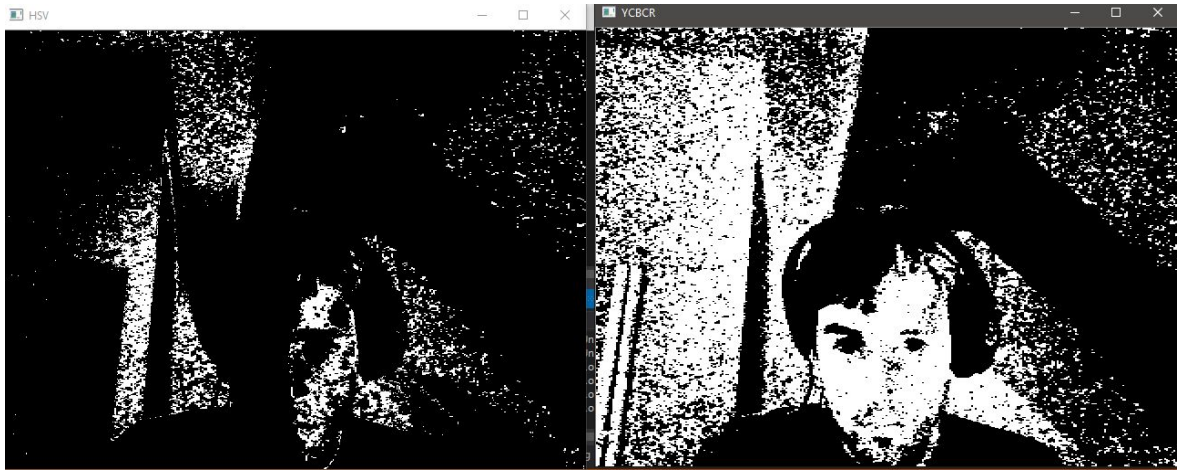
Aplikacja składa się z trzech okien - Przechwytywanego obrazu z interfejsem użytkownika, płótna na którym rysujemy, oraz konsoli. Kropka pośrodku okręgu wskazuje, gdzie obecnie znajduje wskaźnik, którym uruchamiamy wszystkie funkcje programu, oraz którym rysujemy. dwa dodatkowe okna pojawiają się gdy wciśniemy klawisz g - uruchomimy wtedy wykrywanie koloru skóry tak jak widać na obrazie 10. Program można obsługiwać klawiszami:

- zmiana wykrywanego obiektu - spacja
- zapisanie obrazu - ‘s’
- uruchomienie wykrywania koloru skóry - ‘g’
- wyjście z aplikacji - ‘q’
- wyłączenie trybu rysowania - ‘h’

Aktualne działania wyświetlane są w oknie konsoli.



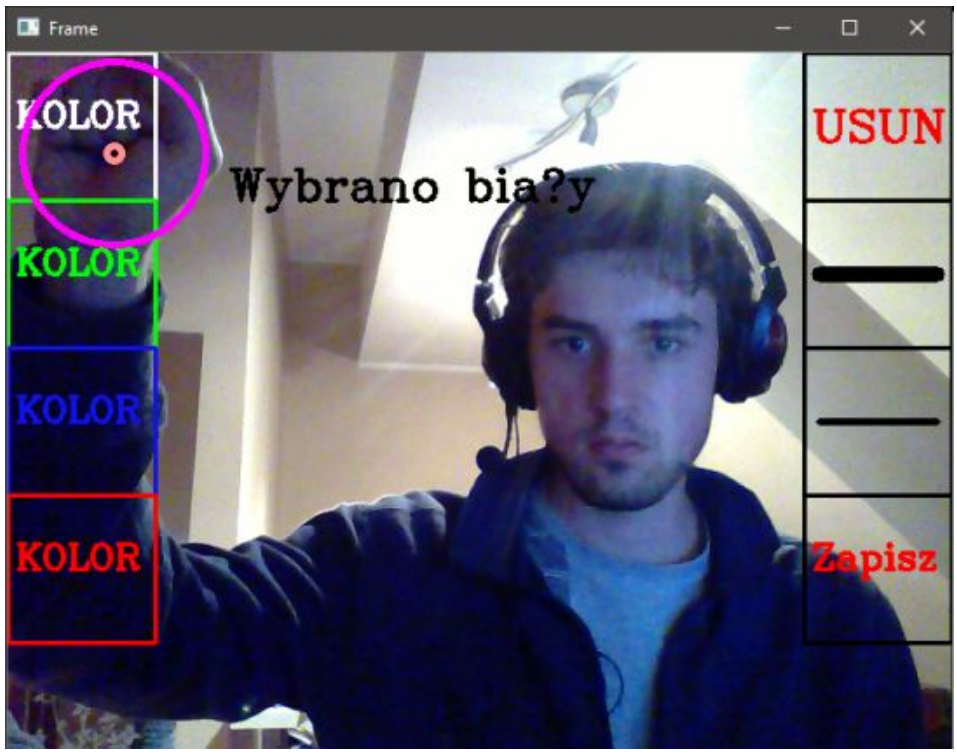
Obraz 9. Cały interfejs.



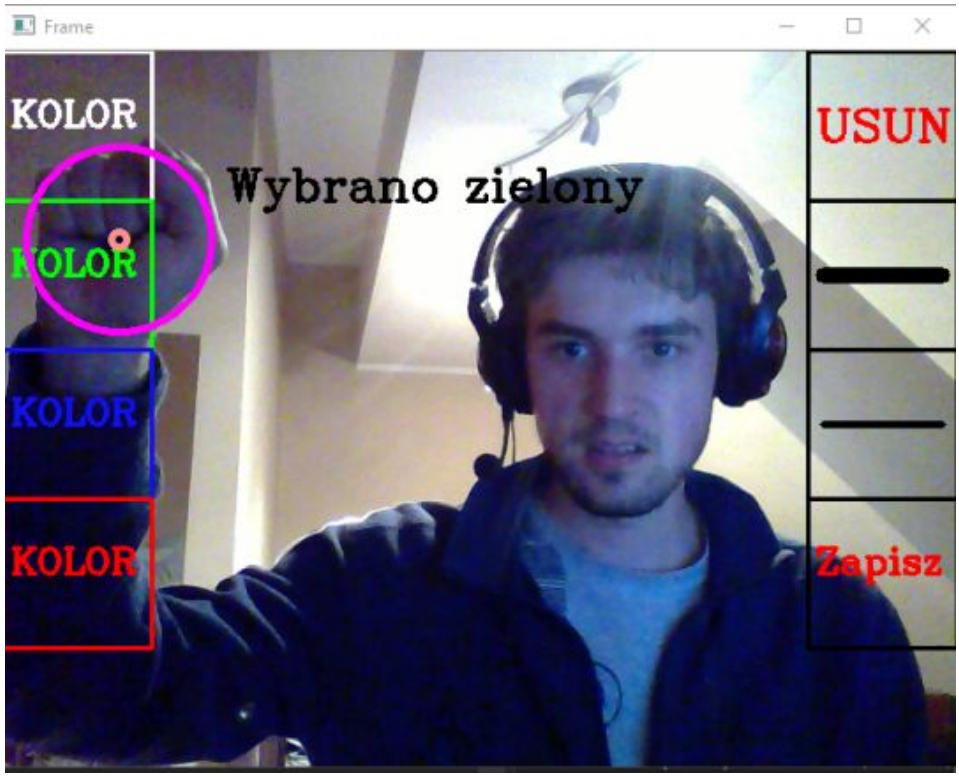
Obraz 10. Dodatkowe okna wykrawania koloru skóry (beżowa ściana nie pomaga w wykrywaniu koloru skóry).

Wybieranie koloru

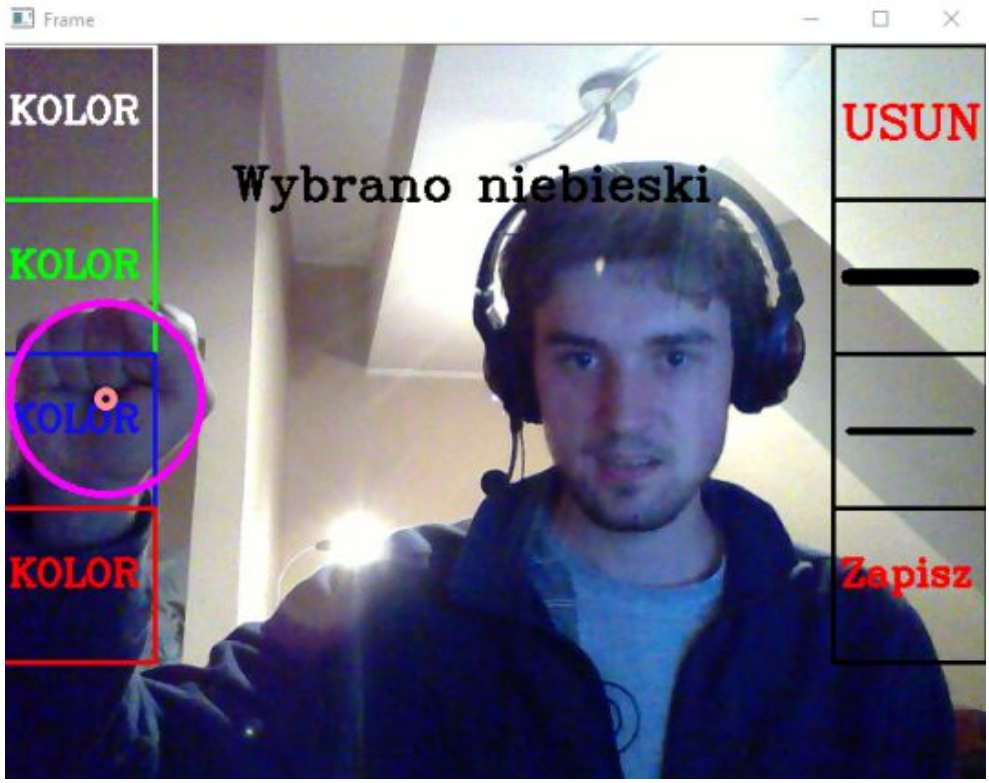
Na obrazach od 9 do 12 mamy pokazane przykłady jak wybierać konkretne kolory pędzla. Na ekranie po lewej znajdują się kratki czterech kolorów, każda z nich zmienia kolor pędzla na kolor napisu. Po wybraniu koloru na ekranie pośrodku wyświetli się komunikat potwierdzający wykonaną czynność .



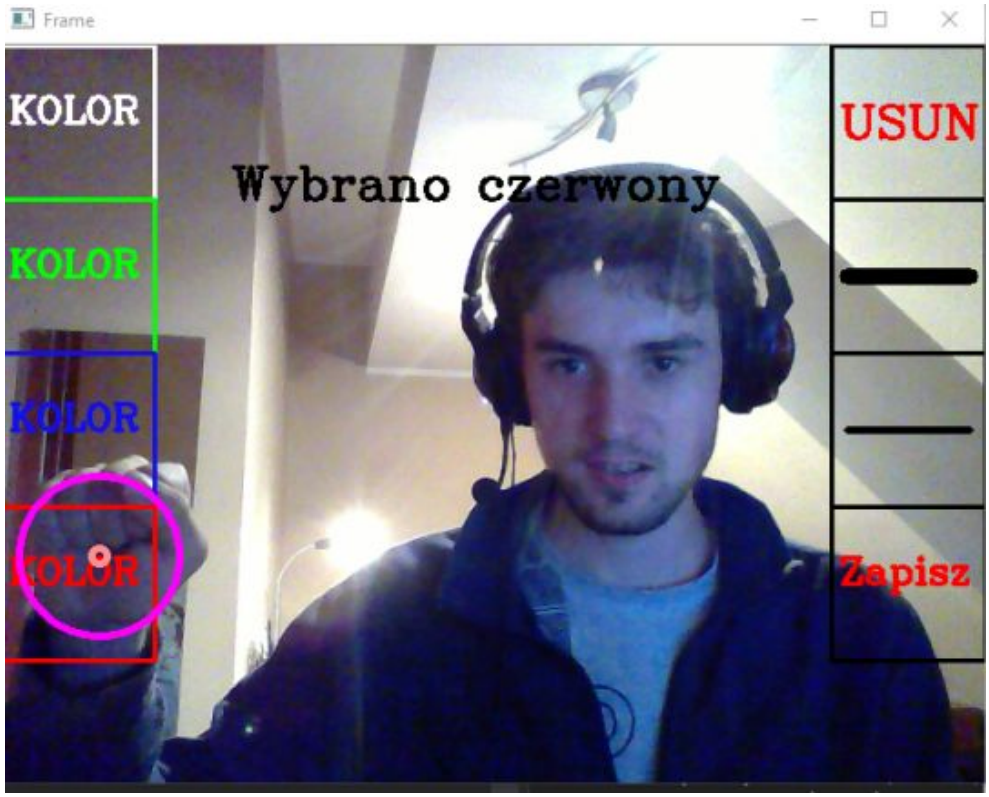
Obraz 11. Wybieranie koloru białego - gumki.



Obraz 12. Wybieranie koloru zielonego.



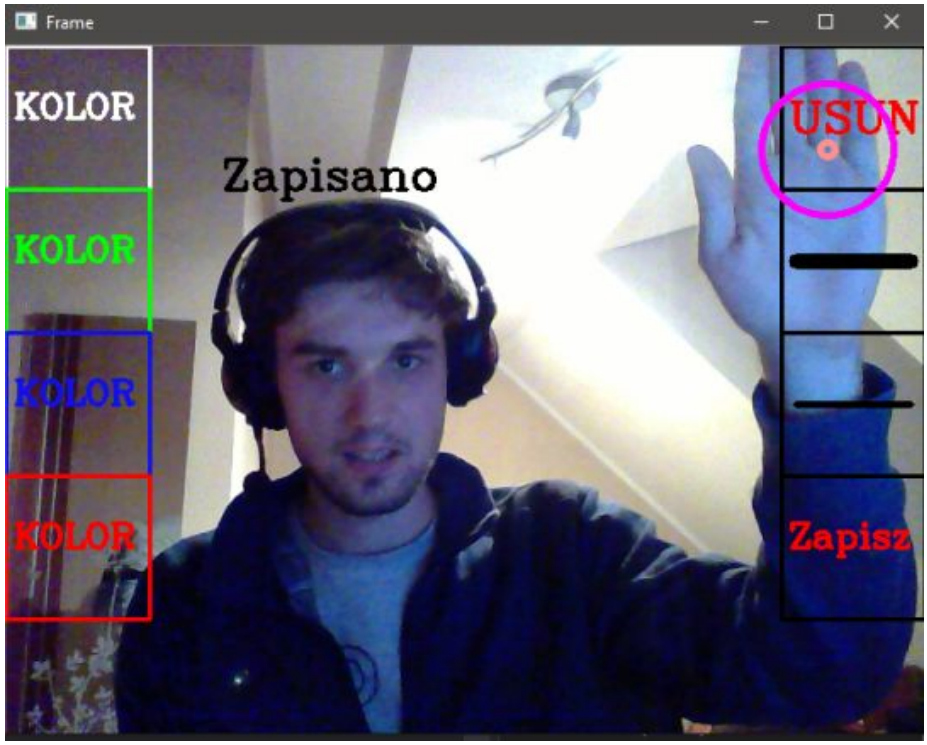
Obraz 13. Wybranie koloru niebieskiego.



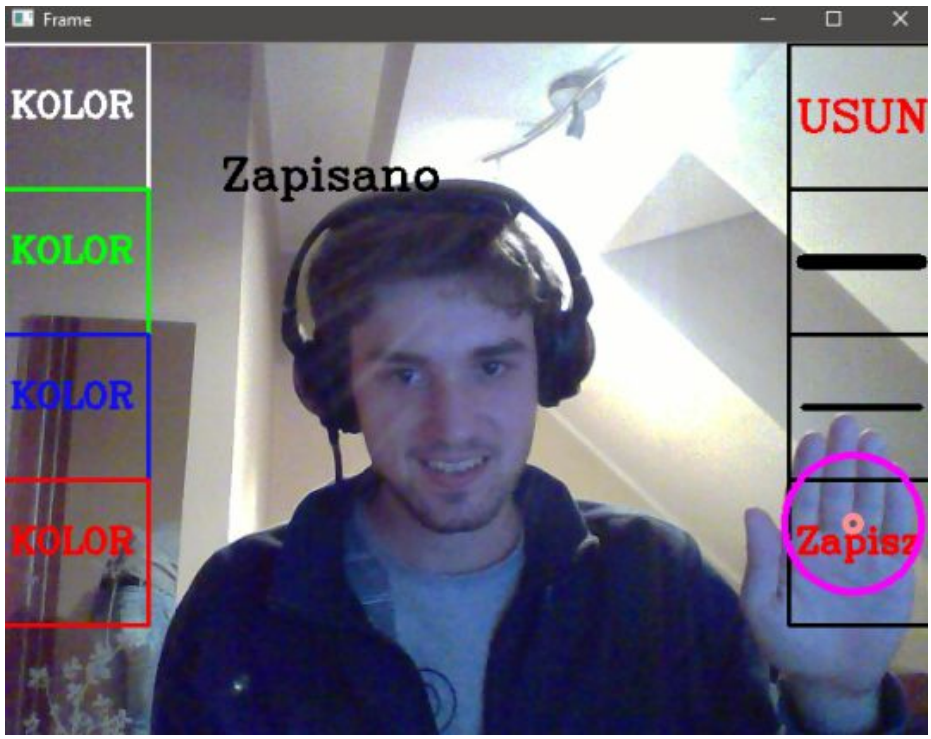
Obraz 14. Wybranie koloru czerwonego.

Czyszczenie płótna, oraz zapis pliku

Po prawej stronie znajduje się pole “USUN”, po najechaniu na to pole program czyści płótno i usuwa nasze postępy. Jak widać na obrazie 13. kratka ta też zapisuje nasz malunek do pliku, na wypadek przypadkowego najechania na tą funkcję. Jak widać na obrazie 14 ostatni napis “Zapisz” powoduje zapisanie aktualnych malunków do pliku.



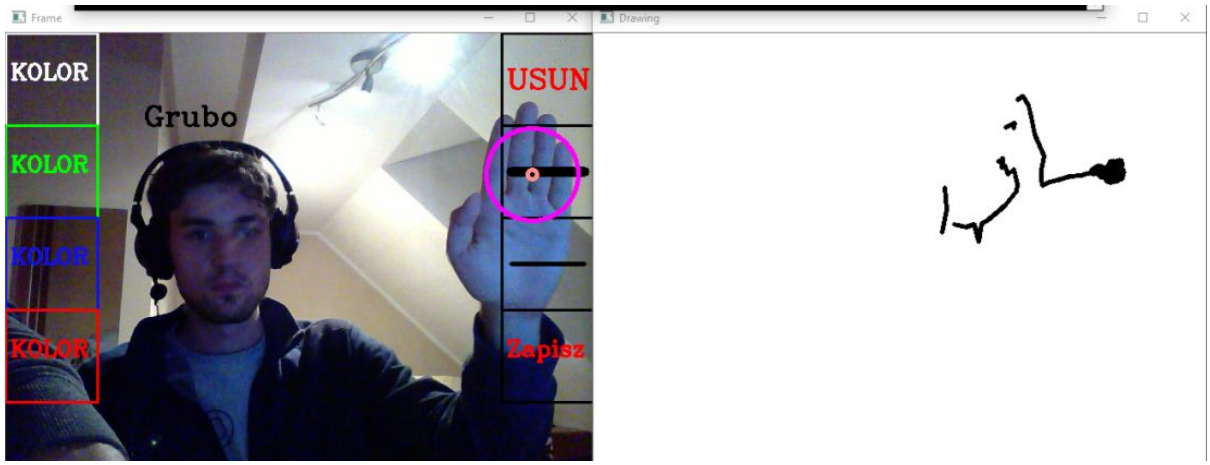
Obraz 15. Czyszczenie obrazu.



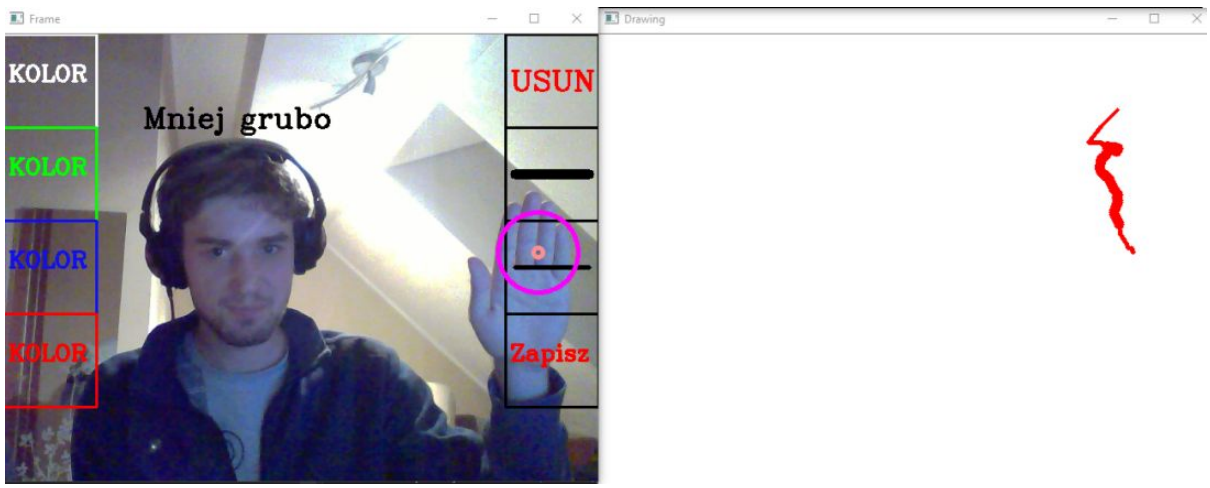
Obraz 16. Zapisanie obrazu.

Zmiana grubości pędzla

Na obrazach 15 oraz 16, po środku znajdują się dwie kreski powodują one zwiększenie lub zmniejszenie grubości pędzla, po prawej pokazany jest ekran, który wyświetla utworzone malunki użytkownika, oraz pokazuje różnicę pomiędzy Grubym pędzlem i tym standardowym.



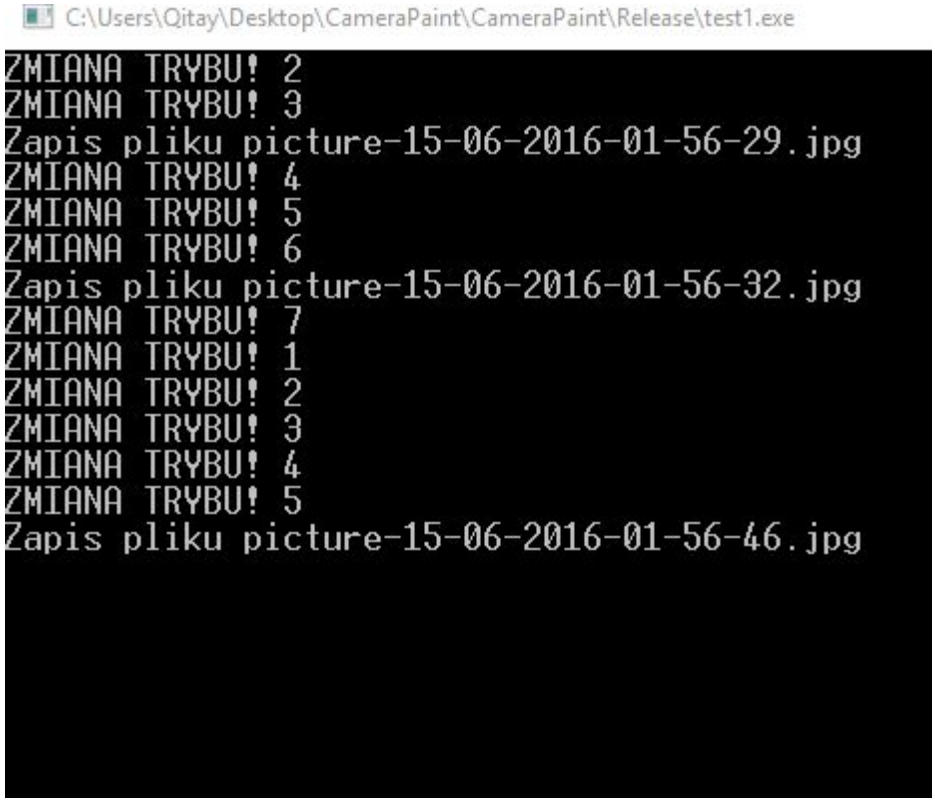
Obraz 17. Gruby pędzel.



Obraz 18. Mniej gruby pędzel.

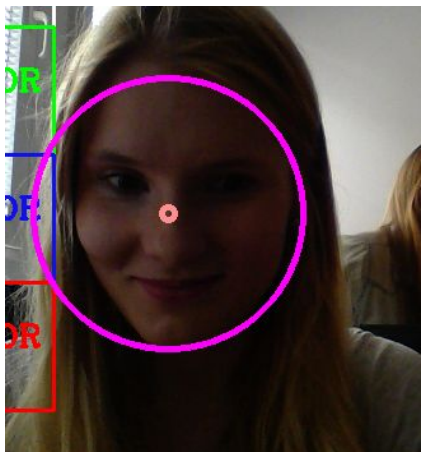
Różne tryby wykrywania części ciała

Przy pomocy klawisza spacja można przełączać się między różnymi kaskadami haara, użyliśmy w sumie 7 kaskad - dwóch dla wykrywania zamkniętej dłoni, dwóch dla wykrywania otwartej dłoni, jednej dla całej głowy, jednej dla pary oczu i jednej dla nosa. Zmiana trybu powoduje wyświetlenie informacji w konsoli (obraz 19).



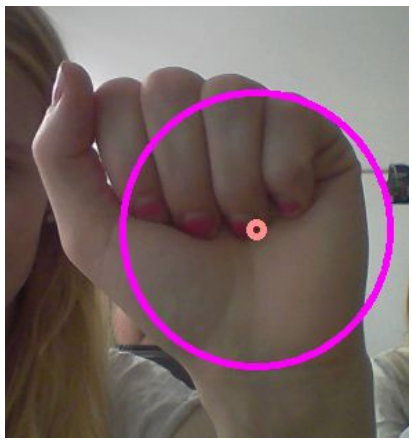
Obraz 19. Konsola przy zmianach trybów wykrywania

.



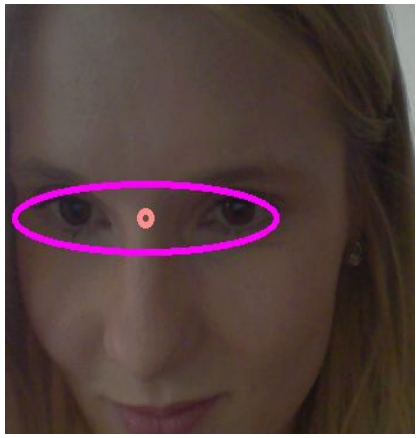
Obraz 20. Tryb 5

Obraz 21. Tryb 7



Obraz 22. Tryby 1,2

Obraz 23. Tryby 3,4



Obraz 24. Tryb 6

14. Podsumowanie

Wykonane cele:

- Przechwytywanie obrazu z kamery
- Wykrywanie obiektów za pomocą nasycenia, koloru
- Malowanie na podstawie ruchu wykrytych obiektów
- Obsługa podstawowych funkcji pędzla
- Implementacja wykrywania, za pomocą kaskady Haara’a
- Zapisywanie obrazu do pliku
- Opracowanie interfejsu aplikacji
- Wykrywanie zaciśniętej dłoni, otwartej dłoni, oczu, nosa, twarzy

Możliwe kierunki rozwoju aplikacji:

- Usprawnienie detekcji części ciała (błędy w rozróżnianiu oraz wykrywania części ciała)
- Skompresowanie liczby okien do jednego okna
- Dodanie nowych wykrywanych części ciała
- Zwiększenie palety kolorów pędzla
- Usprawnienie interfejsu
- Dodanie opcji zmiany rozdzielczości aplikacji
- Możliwość malowania na już istniejącej grafice
- Obsługa za pomocą gestów dłoni
- Upłynnienie ruchów

Praca zespołowa:

Praca zespołowa w naszej grupie była przyjemna, jedynie czasami mieliśmy problemy z komunikacją między sobą poprzez obowiązki pozauczelnianie. Ciężko było znaleźć moment, kiedy cała nasza trójka miałaby wolny czas. Liczne dyskusje jednak pozwoliły nam, na osiągnięcie kompromisu, doprowadzając ostatecznie do prawidłowej komunikacji oraz rozwiazania wszelkie sporów. Bardzo przydatnym okazał się system kontroli wersji Git, oraz scrumchat, który realizowaliśmy za pomocą komunikacji przez facebooka.

Dzięki pracy zespołowej poznaliśmy różnorodne punkty widzenia oraz rozwiązania danych problemów. Nasze osobowości pomimo różnorodności charakterów dopełniały się tworząc kreatywny owoc pracy. Nauczyliśmy się pracy w zespole, oraz wagi odpowiedniego podziału obowiązków. Pomimo wszelkich trudności udało nam się osiągnąć cel, określony w założeniach projektu.

Obraz 25. Przykładowy rysunek stworzony za pomocą naszego programu