

POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRYCZNY

Instytut Automatyki, Robotyki i Inżynierii Informatycznej

Rafał Isbrandt 131768

Konrad Grzelczyk 131452

Stanisław Jajor 131769

Kacper Karman 131777

Dokumentacja projektu na Podstawy Teleinformatyki

Malowanie obrazów biorąc pod uwagę ruch ciała

PT_BeMyBrush

[Github](#)¹

Czerwiec 2019

¹ https://github.com/rav97/PT_BeMyBrush

SPIS TREŚCI

1. CHARAKTERYSTYKA OGÓLNA PROJEKTU	3
1.1. Dlaczego ten temat?	3
1.2. Podział prac.	3
1.3. Harmonogram prac.	4
2. WYMAGANIA	5
2.1. Wymagania funkcjonalne	5
2.2. Wymagania нефункционалне	6
3. NARZĘDZIA, ŚRODOWISKO, BIBLIOTEKI	6
3.1. Środowisko	6
3.2. Narzędzia	6
3.4. Biblioteki	7
4. ARCHITEKTURA ROZWIĄZANIA	7
5. ZAŁOŻENIA IMPLEMENTACYJNE	8
5.1. Kalibracja obrazu	8
5.1.1 Dodany moduł kalibracji obrazu	9
5.2. Okno główne programu i interfejs	10
5.2.1 Okno główne - dzielona obsługa interfejsu	11
5.2.2 Okno główne - obsługa interfejsu za pomocą wskaźnika	12
5.3. Wykrywanie wskaźnika	14
5.3.1 Wykrywanie wskaźnika na podstawie barwy i kształtu	14
5.3.2 Wykrywanie wskaźnika na podstawie większościowego obszaru danej barwy	14
5.4. Rysowanie	15
5.5. Szczegółowy opis elementów interfejsu modelu "wskaźnik"	16
5.5.1 Nowe płótno	16
5.5.2 Zapis	16
5.5.3 Rysowanie	17
5.5.4 Gumka	17
5.5.5 Wybór grubości pędzla	17
5.5.6 Wybór koloru pędzla	18
5.5.7 Wyjście z aplikacji	18

6. NAPOTKANE PROBLEMY	18
6.1. Wykrywanie wskaźnika	18
6.2. Zmienność oświetlenia	19
6.3. Intuicyjność rysowania	19
6.4. Przerywanie rysowania	20
6.5. Wyświetlanie dwóch obrazów	21
6.6. Opracowanie interfejsu użytkownika	21
6.7. Sygnalizacja interfejsu	22
7. PERSPEKTYWY ROZWOJU	23
8. INSTRUKCJA OBSŁUGI	24
8.1. Wstęp	24
8.2. Wskaźnik	24
8.3. Kalibracja	25
8.3. Rysowanie	27
8.4. Interfejs użytkownika	28
8.4. Kończenie pracy aplikacji	30

1. CHARAKTERYSTYKA OGÓLNA PROJEKTU

Celem projektu jest opracowanie oraz implementacja aplikacji opartej o systemy wizyjne, której zadaniem ma być przechwycenie ruchu ciała/obiektu i odzwierciedlenie go w postaci obrazu.

1.1. Dlaczego ten temat?

Temat został wybrany ponieważ wszyscy członkowie grupy posiadają już pewne doświadczenie w zakresie przetwarzania obrazów i systemów wizyjnych. Projekt ten pozwoli na wykorzystanie obecnej wiedzy oraz na zdobycie nowych umiejętności i poznanie nowych technologii oraz rozwiązań.

1.2. Podział prac.

Podział prac zależny będzie od wybranej technologii i dostępnych narzędzi. W początkowej fazie projektu zakłada się pracę wspólną i ewentualnie w późniejszej jego fazie podział na szczegółowe zadania, zgodnie z wiedzą i umiejętnościami poszczególnych członków grupy.

Poniższa tabela przedstawia członków grupy wraz z zadaniami, które wykonywali w czasie pracy nad projektem.

Osoba	Zadania
Rafał Isbrandt	<ul style="list-style-type: none">• Utworzenie wirtualnego środowiska.• Dostęp do kamery i wyświetlanie obrazu.• Wykrywanie konturu i środka ciężkości wskaźnika.• Tworzenie "płótna" o zadanym kolorze.• Rysowanie.• Projekt i implementacja interfejsu.

Konrad Grzelczyk	<ul style="list-style-type: none"> • Dostęp do kamery i wyświetlanie obrazu. • Wykrywanie konturu i środka ciężkości wskaźnika. • Opracowanie kalibracji wykrywania. • Zapisywanie i odczyt ustawień konfiguracyjnych. • Upgrade wskaźnika.
Stanisław Jajor	<ul style="list-style-type: none"> • Dostęp do kamery i wyświetlanie obrazu. • Wykrywanie konturu i środka ciężkości wskaźnika. • Zapisywanie namalowanych obrazów. • Optymalizacja kodu. • Eksport do formatu EXE.
Kacper Karman	<ul style="list-style-type: none"> • Wykonanie wskaźników. • Dostęp do kamery i wyświetlanie obrazu. • Wsparcie duchowe członków zespołu

1.3. Harmonogram prac.

Okres	Założenie
20.03.2019 - 03.04.2019	<ul style="list-style-type: none"> • Ustalenie technologii i podział prac • Pozyskanie technologii • Zapoznanie się z technologiami • Ustalenie harmonogramu • Specyfikacja wymagań i funkcjonalności
03.04.2019 - 17.04.2019	<ul style="list-style-type: none"> • Szkielet projektu i pierwsze metody • Dostęp do kamery i wyświetlanie obrazu • Przygotowanie wskaźnika/wskaźników • Wykrywanie konturu wskaźnika

	<ul style="list-style-type: none"> • Wyznaczanie środka ciężkości konturu • Uzupełnianie dokumentacji
17.04.2019 - 15.05.2019	<ul style="list-style-type: none"> • Metody odpowiedzialne za rysowanie • Detekcja i deskrypcja • Opracowanie i implementacja gestów sterujących aplikacją • Uzupełnianie dokumentacji
15.05.2019 - 29.05.2019	<ul style="list-style-type: none"> • Nadrabianie ewentualnych zaległości • Walidacja rozwiązań • Opracowanie interfejsu • Uzupełnianie dokumentacji i instrukcji obsługi
29.05.2019 - 12.06.2019	<ul style="list-style-type: none"> • Przygotowanie projektu do oddania

2. WYMAGANIA

Wobec projektu sprecyzowane zostały pewne wymagania, które muszą zostać zrealizowane w sposób satysfakcjonujący aby zapewnić użytkownikowi możliwie najlepiej dopracowany produkt. Wymagania te dzieli się na funkcjonalne, czyli dotyczące bezpośrednio funkcji wykonywanych przez system oraz нефункционалне, czyli odpowiedzialne za określone zachowania systemu.

2.1. Wymagania funkcjonalne

- Użytkownik "maluje" przy użyciu dedykowanego wskaźnika,
- Aplikacja przechwytuje pozycję wskaźnika i przenosi ją na obraz w postaci "naniesienia pędzla",
- Użytkownik może wybrać kolor pędzla,

- Użytkownik może wybrać rozmiar pędzla,
- Użytkownik poprzez odpowiedni gest inicjuje i przerywa proces malowania,
- Użytkownik ma możliwość zapisania utworzonego obrazu,

2.2. Wymagania нефunkcjonalne

- Konieczność posiadania kamery internetowej,
- Konieczność odpowiedniego oświetlenia,
- Konieczność posiadania wskaźnika (lub jego zamiennika)

3. NARZĘDZIA, ŚRODOWISKO, BIBLIOTEKI

3.1. Środowisko

- **PyCharm Community Edition 2017.3.4** - zintegrowane środowisko programistyczne dla języka Python. Zostało wybrane ze względu na jego znajomość i pewność, że wybrane narzędzia i biblioteki są obsługiwane.

3.2. Narzędzia

- **Kamera internetowa** - pozwoli na przechwycenie obrazu użytkownika, który po odpowiedniej analizie będzie interpretowany na ruch wirtualnego pędzla.
- **Github** - system obsługi repozytorium i kontroli wersji.
- **Facebook Messenger** - internetowy komunikator wykorzystany do koordynacji prac w sposób zdalny.

3.4. Biblioteki

- **OpenCV-python (3.4.0)** - największa i najbardziej znana biblioteka służąca do przetwarzania obrazów i obsługi kamery.
- **Imutils (0.5.2)** - pomocnicza biblioteka do wykonywania operacji na OpenCV.
- **Scikit-image (0.13.1)** - dodatkowa biblioteka dedykowana do przetwarzania obrazów.
- **Scikit-learn (0.19.1)** - biblioteka zapewniająca metody wykorzystywane w uczeniu maszynowym.
- **Numpy (1.14.0)** - biblioteka ułatwiająca wykonywanie skomplikowanych operacji matematycznych.

4. ARCHITEKTURA ROZWIĄZANIA

Struktura aplikacji została oparta o wykorzystanie pliku źródłowego, zawierającego funkcje i metody niezbędne do wykonania założonych zadań, plik konfiguracyjny przechowujący preferowane ustawienia użytkownika oraz pliki graficzne służące do przedstawienia interfejsu. Aplikacja wykorzystuje szereg bibliotek, które pozwalają efektywnie wykonywać zadania konieczne do uzyskania efektu końcowego. Wykorzystanie odpowiednio sparametryzowanych funkcji zawartych w bibliotekach jest podstawą języka Python, w którym aplikacja została utworzona. Działanie aplikacji zostało oparte o autorskie metody, które wywoływane z odpowiednimi parametrami modyfikują dane (obraz) oraz przekazują wyniki swoich działań innym funkcjom. Aplikacja w swoim funkcjonowaniu została podzielona na etapy. Pierwszy z nich to etap kalibracji w którym użytkownik dostosowuje obraz otrzymywany z kamery do poprawnego wykrywania wskaźnika. Drugi etap to faza modyfikowania obrazu za pomocą ruchów dłoni. Aplikacja pozwala na nanoszenie zmian,

ich usuwanie oraz zapisywanie obrazu do pliku. Drugi etap programu działa w oparciu o wykorzystanie pętli, dlatego program wykonuje zdefiniowane operacje w kolejności i czasie ustalonym przez użytkownika. Architektura oraz przeznaczenie aplikacji zakłada minimalne wymagania, które muszą zostać spełnione do poprawnego działania produktu. Aby móc korzystać z pełnego zakresu możliwości aplikacji należy posiadać:

- komputer z zainstalowanym systemem operacyjnym Windows,
- komputer z kamerą,
- Wskaźnik lub jego zamiennik.

5. ZAŁOŻENIA IMPLEMENTACYJNE

W procesie wstępnego projektowania oraz tworzenia aplikacji, opracowano założenia implementacyjne. Określają one sposób działania i funkcjonowania podstawowych modułów programu. Dzięki ich przygotowaniu i szczegółowemu omówieniu w początkowej fazie projektu, proces implementacji przebiegł stosunkowo sprawnie. Oczywiście napotkano kilka problemów (więcej: Rozdział 6 "NAPOTKANE PROBLEMY"), ale poprzez solidne przygotowanie w fazie projektowej udało się je rozwiązać bądź modyfikować w sposób zdecydowanie korzystny dla finalnego wyniku prac.

Podstawowe założenia implementacyjne zostały pogrupowane zgodnie z kolejnością chronologiczną występowania w aplikacji.

5.1. Kalibracja obrazu

W pierwotnych założeniach implementacyjnych kalibracja obrazu nie występuje, ponieważ zakładano bezproblemowe rozpoznanie wskaźnika o stałym kolorze. W późniejszym etapie projektowym zdecydowano się jednak na przygotowanie wspomnianego modułu, głównie z powodu dwóch następujących aspektów: wpływu warunków oświetleniowych

na wykrywanie wskaźnika oraz chęci udostępnienia możliwości przygotowania wskaźnika o innym kolorze niż prototyp.

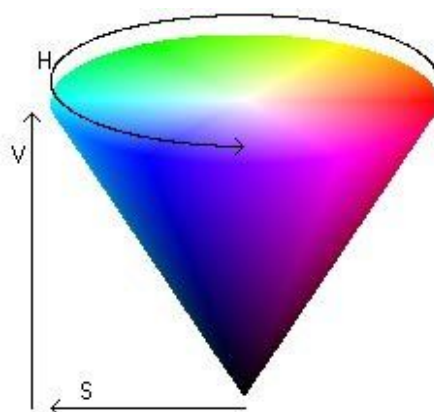
5.1.1 Dodany moduł kalibracji obrazu

Początkowe wartości pobierane są z pliku konfiguracyjnego "Config.txt" (jeśli istnieje). Wartości te pochodzą z ostatniej zatwierdzonej konfiguracji.

```
def ReadFromFile(nr):  
    ParametersFile = open('Resources/Config.txt')  
    try:  
        tekst = ParametersFile.read()  
    finally:  
        ParametersFile.close()  
  
    x = tekst.split('\n')  
    i = 0  
  
    for a in range(6):  
        nr[i] = int(x[i])  
        i += 1
```

Rys.1. Funkcja odpowiedzialna za pobieranie danych z pliku konfiguracyjnego

Moduł powinien umożliwić dopasowanie parametrów wartości progowania (próg górny i próg dolny), zgodnie z modelem przestrzeni barw HSV, do warunków oświetleniowych i koloru wskaźnika.



Rys.2. Model przestrzeni barw HSV

Jednocześnie wymagany jest podgląd aktualnego wykrycia wskaźnika, w celu przedstawiania postępu kalibracji. Pod koniec tego etapu następuje zapis konfiguracji użytkownika w pliku tekstowym oraz przejście (po wciśnięciu klawisza Enter) do właściwego okna programu.

```
def WriteToFile(Ol, Ou):  
    ParametersFile = open("Resources/Config.txt", "w")  
  
    ParametersFile.write(str(Ol[0]) + '\n')  
    ParametersFile.write(str(Ol[1]) + '\n')  
    ParametersFile.write(str(Ol[2]) + '\n')  
    ParametersFile.write(str(Ou[0]) + '\n')  
    ParametersFile.write(str(Ou[1]) + '\n')  
    ParametersFile.write(str(Ou[2]) + '\n')  
  
    ParametersFile.close()
```

Rys.3. Fragment kodu odpowiedzialny za zapis parametrów do pliku konfiguracyjnego

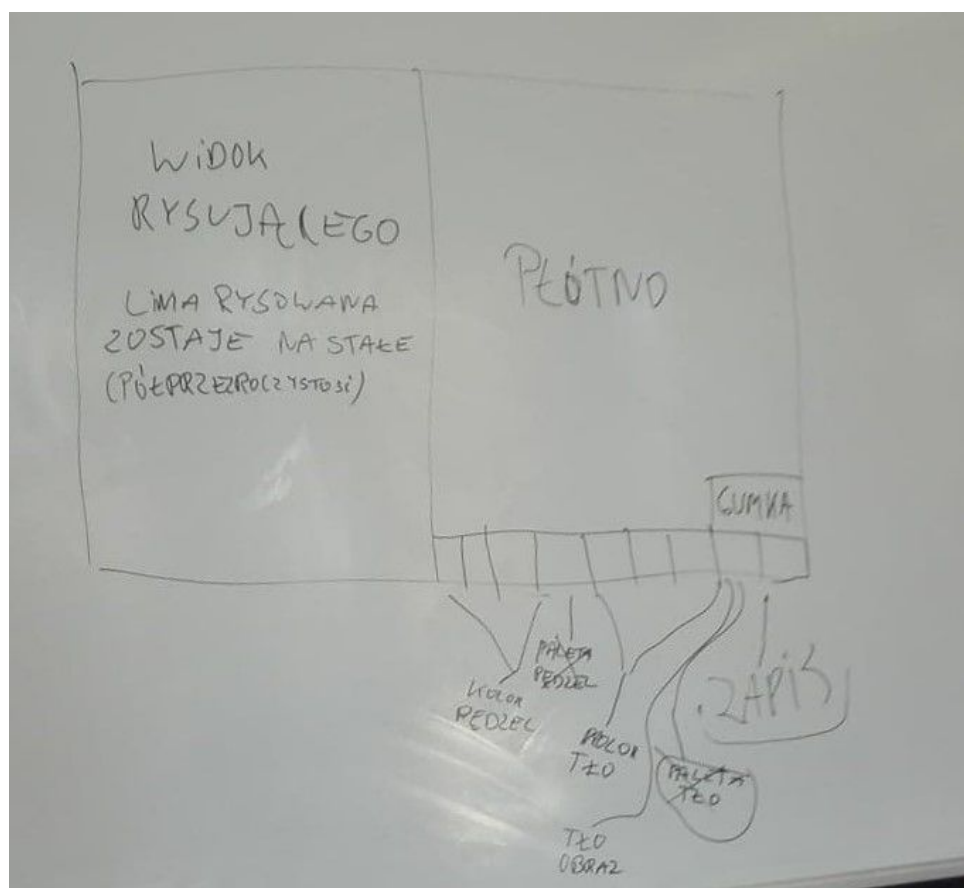
5.2. Okno główne programu i interfejs

Najważniejszym założeniem głównego okna programu jest wyświetlanie użytkownikowi dwóch obrazów jednocześnie, w celu jak najbardziej intuicyjnego tworzenia dzieł. Pierwszym (po lewej stronie) jest widok, obsługującego wskaźnik, bezpośrednio przechwycony z kamery. Na to zostaje nałożony półprzezroczysty ślad pędzla. Drugie okno (po prawej) stanowi płótno, na którym dokonywane są modyfikacje (rysowanie) ruchem wskaźnika. Takie połączenie widoków sprawia, że rysowanie, przerywanie rysowania oraz obsługa opcji dodatkowych jest maksymalnie ułatwiona i usprawniona

W toku tworzenia aplikacji powstały dwa różne założenia obsługi interfejsu programu, a co za tym idzie dwa zupełnie różne modele aplikacji. W obu przypadkach podstawowe założenia pozostają identyczne.

5.2.1 Okno główne - dzielona obsługa interfejsu

Pierwszym projektem okna głównego aplikacji był model "wskaźnik / mysz". Zgodnie z nazwą, w tej implementacji planowano podzielić obsługę programu na dwa oddzielne urządzenia. Rysowanie miałoby się wykonywać tylko za pomocą wskaźnika, a cała reszta obsługi się odbywałaby się z wykorzystaniem myszki komputerowej. Cały dostępny interfejs dla użytkownika planowano umieścić "pod" płótnem. Dużą różnicą w stosunku do modelu "wskaźnik" jest dostęp do palety barw, zarówno przy wyborze koloru pędzla jak i płótna. Przy wykorzystaniu myszy dokładny wybór koloru jest znacznie łatwiejszy. Również sam panel interfejsu w tym przypadku był mniej "inwazyjny" i zajmował zdecydowanie mniejszą część widoku.



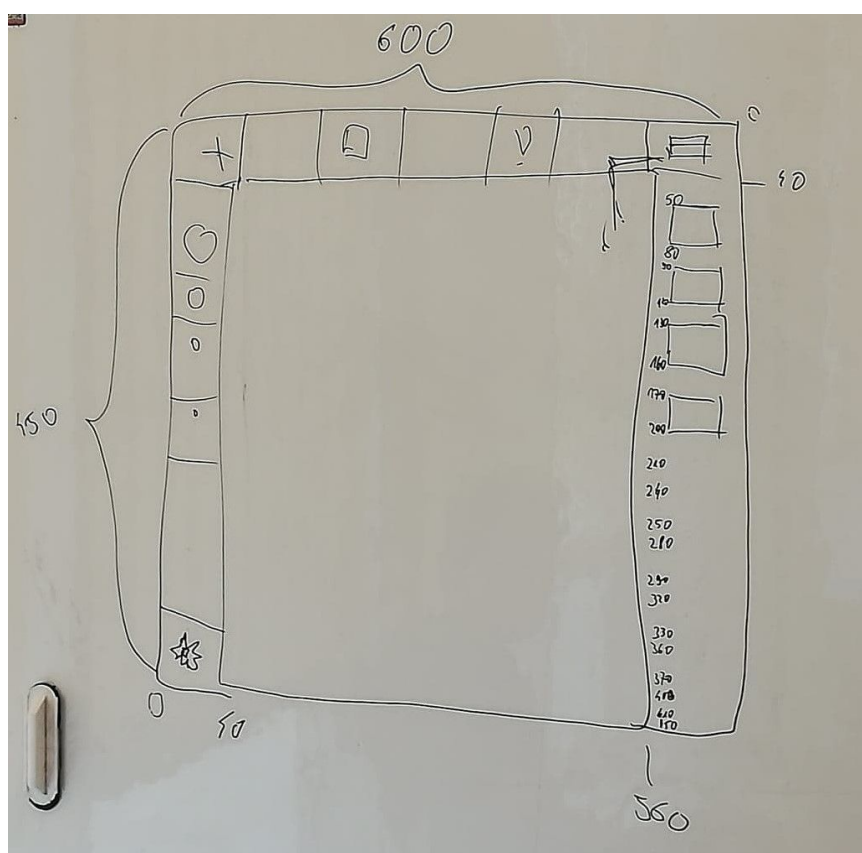
Rys.4. Model projektowy okna głównego w modelu "wskaźnik / mysz"

Ostatecznie z modelu modelu "wskaźnik / mysz" zrezygnowano z powodu trudności występujących przy korzystaniu z aplikacji. Używanie jednocześnie myszki i wskaźnika byłoby uciążliwe oraz znacząco utrudniało przyjemne i łatwe rysowanie.

5.2.2 Okno główne - obsługa interfejsu za pomocą wskaźnika

Drugim modelem, powstałym już po zrezygnowaniu z pierwszego, jest model "wskaźnik". Zakłada on całkowite poleganie na przygotowanym wskaźniku jako narzędziu do obsługi panelu i rysowania. Dzięki temu unika się sytuacji, w której użytkownik musi podchodzić do urządzenia, w celu zmiany parametrów, np. koloru pędzla.

Opracowanie przedstawionego modelu wymusiło znaczące zmiany w widoku głównym programu.



Rys.5. Model projektowy lewej części okna głównego w modelu "wskaźnik"

Przed wszystkim zrezygnowano z umieszczenia panelu użytkownika pod widokiem płótna. Wspomniany element przeniesiono i osadzono wokół lewej części okna głównego (widok rysującego). Dzięki takiemu zabiegowi można korzystać z dostępnych zmian ustawień i obsługi programu (szczegółowo opisano w podrozdziale 5.5 "Szczegółowy opis elementów interfejsu modelu "wskaźnik"").

Zrezygnowano z udostępniania całej palety barw, ponieważ dużo bardziej intuicyjną opcją jest wybór spośród kilku kolorów (rozmieszczonych w ustalonej odległości) za pomocą wskaźnika niż próba dopasowania wymaganej barwy w palecie. Dodano również przycisk wyjścia z aplikacji.

Wybór danej opcji w panelu możliwy jest poprzez najechanie kursorem wskaźnika na ikonę jej odpowiadającą. Przestrzeń wykrycia została ograniczona zgodnie z rozmieszczeniem w panelu.

```
if x < 35:
    if 0 < y < 70:
        canvas = CreateBlankCanvas(frame.shape[1], frame.shape[0], canvas_color[0], canvas_color[1],
                                    canvas_color[2])
    if 70 < y < 135:
        brush_size = 30
    if 135 < y < 195:
        brush_size = 25
    if 195 < y < 255:
        brush_size = 20
    if 255 < y < 315:
        brush_size = 10
    if 315 < y < 375:
        brush_size = 5
    if y > 375:
        break # wyjście
if x > 565:
    if 0 < y < 45:
        nothing(1) # saveFile()
    if 45 < y < 85:
        brush_color = [255, 255, 255] # biały
    if 85 < y < 125:
        brush_color = [0, 0, 0] # czarny
    if 125 < y < 165:
        brush_color = [0, 0, 45] # brązowy
    if 165 < y < 205:
        brush_color = [255, 0, 0] # niebieski
    if 205 < y < 245:
        brush_color = [255, 255, 0] # cyan
    if 245 < y < 285:
        brush_color = [0, 255, 0] # zielony
    if 285 < y < 325:
        brush_color = [0, 0, 255] # czerwony
    if 325 < y < 365:
        brush_color = [0, 128, 255] # pomarańcz
    if 365 < y < 405:
        brush_color = [0, 255, 255] # żółty
    if 405 < y < 450:
        brush_color = [255, 0, 255] # magenta
```

Rys.6. Widok części zakresów (zmiana koloru) przestrzeni wykrywających wskaźnik dla danej opcji

5.3. Wykrywanie wskaźnika

Wykrywanie wskaźnika stanowi najważniejszy element programu, jako warunek konieczny do późniejszej funkcji rysowania. W procesie tworzenia aplikacji istniały jego dwa różne modele. W obu wykorzystuje się operację progowania w przestrzeni barw HSV.

5.3.1 Wykrywanie wskaźnika na podstawie barwy i kształtu

Pierwotnym założeniem było dokonanie operacji progowania w przestrzeni barw HSV, co umożliwiło odfiltrowanie wszystkich barw odmiennych od barwy wskaźnika, a następnie na tak przygotowanym obrazie zastosowanie algorytmu detekcji, przystosowanego do wykrywania okrągłych kształtów. Model ten został odrzucony ze względu na trudności z wykryciem idealnie kulistego kształtu.

5.3.2 Wykrywanie wskaźnika na podstawie większościowego obszaru danej barwy

Ze względu na problemy występujące w modelu opisanym w podrozdziale 5.3.1 zdecydowano się na pozostanie przy wykrywaniu opartym wyłącznie na identyfikowaniu największego obszaru o określonej barwie - barwie wskaźnika. Program następnie określa wielkość okręgu, w którym obszar ten może zostać zamknięty. Środek tego okręgu stanowi końcówkę "pędzla", którym użytkownik będzie mógł rysować.

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv, orangeLowerdrawing, orangeUpperdrawing)

mask = cv2.erode(mask, None, iterations=2)
mask = cv2.dilate(mask, None, iterations=3)

cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)

if len(cnts) > 0:
    c = max(cnts, key=cv2.contourArea)
    ((x, y), radius) = cv2.minEnclosingCircle(c)
```

Rys.7. Fragment kodu odpowiedzialny za wykrycie wskaźnika



Rys.8. Oznaczenie wskaźnika w oknie aplikacji

5.4. Rysowanie

Drugim najważniejszym elementem programu jest prawidłowe rysowanie obrazu na podstawie ruchu wskaźnika. W miejscu wykrycia wskaźnika powstaje punkt, który następnie łączony jest linią z kolejnym wykrytym punktem. Przerwanie rysowania spowodowane jest jego niewykryciem. Po ponownym odsłonięciu piłeczki rysowanie rozpoczyna się od nowego punktu. Uniknięto problemu łączenia się, starego punktu z nowym, za pomocą blokady odległości (więcej w rozdziale 6. "NAPOTKANE PROBLEMY").

```
def DrawLine(ox, oy, nx, ny, thickness, blue, green, red, output):
    cv2.line(output, (int(ox), int(oy)), (int(nx), int(ny)), (blue, green, red), thickness)
```

Rys.9. Fragment kodu przedstawiający funkcję odpowiedzialną za rysowanie

5.5. Szczegółowy opis elementów interfejsu modelu "wskaźnik"

W następującym rozdziale opisano poszczególne elementy interfejsu finalnego widoku głównego programu. Każda z przedstawionych funkcjonalności ma swój zakres wykrycia w oknie podglądu widoku z kamery. Dzięki temu użytkownik może wybrać wymaganą opcję.

5.5.1 Nowe płótno

Domyślnie po uruchomieniu aplikacji użytkownik zaczyna pracę z pustym płótnem. Jeżeli, w którymkolwiek momencie będzie chciał ją rozpocząć od nowa, może wybrać omawianą opcję. Spowoduje to wyczyszczenie płótna i obszaru podglądu programu.

5.5.2 Zapis

Zapis jako niesamowicie istotna funkcjonalność został bardzo dokładnie przemyślany, a następnie przetestowany i zaimplementowany. Po najechniu na wskaźnik tej opcji następuje zapis obrazu do folderu "SavedImages" o nazwie w formacie "Picture[rok + miesiąc + dzień + godzina + minuta + sekunda]". Dodatkowo przygotowano zabezpieczenie przed niepotrzebnym wielokrotnym zapisem. Ponowne wykorzystanie tej opcji nastąpi tylko po wcześniejszym wykryciu znacznika w innym obszarze ekranu. Po wykonaniu operacji zapisu można dalej kontynuować tworzenie dzieła

```
#zapis obrazu do pliku
def SaveFile(cs, image):

    if cs == 1:
        timestr = time.strftime("%Y%m%d_%H%M%S")
        if not os.path.exists("SavedImages"):
            os.mkdir("SavedImages")
        cv2.imwrite("SavedImages/Picture%s.jpg" % timestr, image)
```

Rys.10. Fragment kodu przedstawiający funkcję odpowiedzialną za zapis obrazu

5.5.3 Rysowanie

Domyślnie po uruchomieniu programu aktywna jest funkcja rysowanie. Przebiega zgodnie z podrozdziałem 5.4 "Rysowanie". Należy ją wybrać, np. po wykorzystaniu funkcji gumka w celu ponownej aktywacji tworzenia dzieła.

5.5.4 Gumka

W przypadku chęci poprawy danej części tworzonego dzieła istnieje możliwość wyboru opcji gumka. Zmazywanie odbywa się poprzez zmianę koloru pędzla na zgodny z tłem i tym samym na usunięciu niechcianego fragmentu.

```
if y < 35:
    if 100 < x < 250:
        rubber = 1
    if 350 < x < 500:
        rubber = 0
```

Rys.11. Fragment kodu przedstawiający funkcję odpowiedzialną za włączenie i wyłączenie opcji gumki

5.5.5 Wybór grubości pędzla

Skutkiem wyboru grubości pędzla jest zwiększenie lub zmniejszenie elementu rysującego i tym samym modyfikacja powierzchni rysowania.

```
if brush_size == 30:
    cv2.rectangle(frame, (int(2), int(79)), (int(38), int(132)), (0, 0, 255), thickness=2, lineType=8)
if brush_size == 25:
    cv2.rectangle(frame, (int(2), int(140)), (int(38), int(191)), (0, 0, 255), thickness=2, lineType=8)
if brush_size == 20:
    cv2.rectangle(frame, (int(2), int(200)), (int(38), int(251)), (0, 0, 255), thickness=2, lineType=8)
if brush_size == 10:
    cv2.rectangle(frame, (int(2), int(260)), (int(38), int(312)), (0, 0, 255), thickness=2, lineType=8)
if brush_size == 5:
    cv2.rectangle(frame, (int(2), int(320)), (int(38), int(372)), (0, 0, 255), thickness=2, lineType=8)
```

Rys.12. Fragment kodu przedstawiający funkcję odpowiedzialną za oznaczenie na interfejsie wybranej grubości pędzla

5.5.6 Wybór koloru pędzla

Oczywistym skutkiem wyboru danego koloru jest zmiana barwy rysowania, a także wyświetlenie się w dolnym panelu informacji o aktualnym kolorze. Bardzo istotna funkcjonalność, dzięki której można tworzyć znacznie bardziej zaawansowane dzieła. Zakres obszarów kolorów został przedstawiony na Rys.6.

5.5.7 Wyjście z aplikacji

Po spełnieniu jednego z warunków wyjścia następuje komenda "break". Która zatrzymuje główną pętlę działania programu, co skutkuje zakończeniem jego pracy.

```
if y > 375:  
    break # wyjście
```

Rys.13. Fragment kodu przedstawiający funkcję odpowiedzialną za wyjście z programu

6. NAPOTKANE PROBLEMY

Podczas prac nad projektem napotkano różnego rodzaju problemy związane z implementacją. Każdy z nich musiał zostać rozwiązany, aby możliwe było zaprezentowanie gotowej aplikacji zgodnej z funkcjonalnościami przyjętymi w specyfikacji wymagań.

6.1. Wykrywanie wskaźnika

Pierwszym istotnym problemem, który napotkano była kwestia sposobu w jaki program miał dokonywać operacji wykrywania wskaźnika. Założeniem było dokonanie operacji progowania w przestrzeni barw HSV, co umożliwiło odfiltrowanie wszystkich barw odmiennych od barwy wskaźnika, a następnie na tak przygotowanym obrazie zastosować algorytm detekcji, przystosowany do wykrywania okrągłych kształtów.

Pomysł ten został odrzucony, ponieważ niemożliwe okazało się nałożenie takich progów aby wykryty kształt był idealnie kulisty, dodatkowo wymuszało by to na użytkowniku konieczność trzymania wskaźnika zawsze przodem do kamery. Pozostano zatem przy wykrywaniu opartym wyłącznie na identyfikowaniu największego obszaru o określonej barwie - barwie wskaźnika. Program następnie określa wielkość okręgu, w którym obszar ten może zostać zamknięty. Środek tego okręgu stanowi końcówkę "pędzla", którym użytkownik będzie mógł rysować.

6.2. Zmienność oświetlenia

Problem ten pośrednio wiąże się z poprzednim. Ponieważ wykrywanie oparte jest na rozpoznawaniu barwy wskaźnika, powoduje to znaczne uzależnienie od warunków oświetlenia. Odcień jaki posiada wskaźnik ulegał zmianie w zależności od poziomu oświetlenia w pomieszczeniu. Gdy było zbyt jasno lub zbyt ciemno, aplikacja nie była w stanie poprawnie określić położenia wskaźnika.

W celu rozwiązania problemu wdrożone zostały dwa rozwiązania, częściowo niwelujące wpływ oświetlenia. Pierwszym z nich było wprowadzenie do aplikacji etapu umożliwiającego kalibrację wykrywania. Za każdym uruchomieniem aplikacji użytkownik ręcznie definiuje wartości progowania HSV, dzięki czemu może dostosować je do aktualnych warunków oświetlenia. Drugim rozwiązaniem było zmodyfikowanie samego wskaźnika, wyposażony został w obwód elektryczny z diodą, która go rozświetla. Wskaźnik emitujący własne źródło światła staje się w ten sposób odporny na refleksy światła pochodzącego z zewnątrz, przez co jego barwa pozostaje w stałej wartości.

6.3. Intuicyjność rysowania

Ponieważ założeniem dotyczącym rysowania jest przechwycenie obrazu z kamery, na którego podstawie użytkownik może kontrolować proces rysowania. Konieczne było określenie sposobu, w jaki użytkownik może ten proces wygodnie nadzorować.

Zdecydowano się na podział na dwa obrazy z podglądem. Jeden obraz przedstawia stan płótna na którym przeprowadzane jest rysowanie. Drugi natomiast pokazuje podgląd z kamery, na który nałożony jest prześwitujący podgląd z płótna. Takie rozwiązanie pozwala użytkownikowi na sprawne nadzorowanie procesu rysowania, a także wznowianie rysowania w zadanym punkcie.

6.4. Przerywanie rysowania

Proces rysowania w aplikacji polega na określeniu pozycji środka obszaru o zadanej barwie - wskaźnika. Z każdą przechwyconą klatką obrazu z kamery określany jest punkt poprzedni oraz obecny, następnie program rysuje linię prostą między tymi punktami. Zgodnie z założeniem rysowanie miało zostać przerwane w chwili kiedy wskaźnik przestanie być wykrywany (zostanie zakryty). Takie rozwiązanie sprawiało jednak, że w momencie jak użytkownik będzie chciał wznowić rysowanie w nowym punkcie, program połączy za pomocą linii punkt poprzedni z aktualnym.

W celu rozwiązania problemu w kodzie źródłowym zostało wprowadzone ograniczenie dotyczące możliwości łączenia wykrywanych punktów w linię. Łączenie następuje wyłącznie w przypadku, gdy odległość między poprzednim a aktualnym punktem jest mniejsza niż 25 pikseli obrazu. Jest to rozwiązanie wystarczające dla precyzji rysowania, którą może zapewnić aplikacja.

```

if abs(ox - x) < 25 and abs(oy - y) < 25:
    if rubber:
        DrawLine(ox, oy, x, y, brush_size, canvas_color[0], canvas_color[1], canvas_color[2],
                  canvas)
    else:
        DrawLine(ox, oy, x, y, brush_size, brush_color[0], brush_color[1], brush_color[2],
                  canvas)

```

Rys.14. Warunek umożliwiający wznowienie malowania w innym miejscu.

6.5. Wyświetlanie dwóch obrazów

Założeniem dotyczącym intuicyjności rysowania w aplikacji było przedstawienie użytkownikowi dwóch obrazów. Płótna - na którym powstaje obraz, oraz podglądu z kamery, na który nałożony jest obraz z płótna, co umożliwi użytkownikowi wznowianie rysowania na pożądanym obszarze. OpenCV nie pozwala jednak na wyświetlanie dwóch obrazów w obszarze jednego okna.

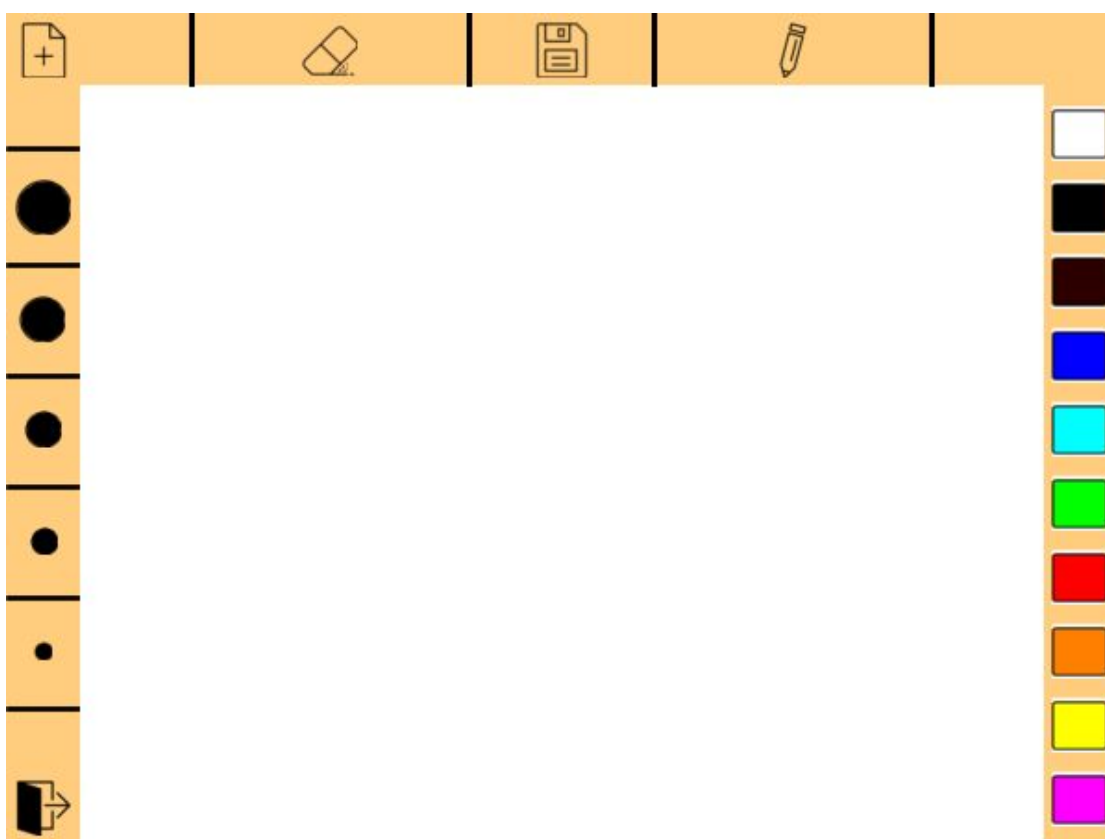
Aby rozwiązać ten problem możliwe były dwa rozwiązania: wyświetlenie każdego obrazu w osobnym oknie lub połączenie dwóch obrazów w jeden, możliwy do wyświetlenia w pojedynczym oknie. Zdecydowano się na drugie rozwiązanie. Przed wyświetleniem okna obrazy są scalane wzdłuż pionowej krawędzi tworząc jeden duży możliwy do wyświetlenia.

6.6. Opracowanie interfejsu użytkownika

Pierwotnym założeniem dotyczącym obsługi aplikacji miało być opracowanie interfejsu użytkownika obsługiwanego przy użyciu myszy. Takie rozwiązanie generowało jednak problemy. Pierwszym z nich był sam problem implementacji takiego rozwiązania. Python nie zapewnia wielu możliwości implementacji interfejsu, ponadto wymagają one działania w osobnej pętli. Takie rozwiązanie kolidowałoby z GUI zapewnianym przez OpenCV, oraz pętlą wykorzystywaną do operacji na obrazie. Drugim problemem był fakt braku wygody w obsłudze takiego programu.

Użytkownik musiałby przerywać rysowanie, aby podejść do komputera, przestawić oczekiwane opcje a następnie ustawić się ponownie na pozycji umożliwiającej rysowanie.

W celu rozwiązania tego problemu zdecydowano się na zastosowanie interfejsu sterowanego, w ten sam sposób jak proces rysowania - poprzez pozycję wskaźnika. Obszar rysowania został ograniczony o górną i boczne krawędzie, w celu umieszczenia nakładki z interfejsem. Użytkownik najeżdżając wskaźnikiem na obszar danego przycisku może wywołać odpowiednią akcję.



Rys.15. Nakładka z interfejsem użytkownika.

6.7. Sygnalizacja interfejsu

Opracowanie interfejsu w sposób opisany powyżej skutkuje kolejnym problemem. Użytkownik powinien zostać poinformowany o aktywności danej funkcjonalności z interfejsu.

Zdecydowano się na skorzystanie z narzędzia dostarczanego przez

OpenCV, w zależności od wybranej opcji z interfejsu, jej "przycisk" otoczony zostaje czerwoną ramką. Ramka ta sygnalizuje użytkownikowi o wybranym trybie rysowania (gumka lub pędzel), aktualnej grubości pędzla oraz zapisaniu obrazu. O wybranym kolorze pędzla informuje kolor okręgu sygnalizującego pozycję wskaźnika.

7. PERSPEKTYWY ROZWOJU

Aplikacja w obecnej postaci ma jeszcze spory potencjał na rozwój. Poniższa lista prezentuje funkcjonalności, o które można wzbogacić aplikację w przyszłości.

Możliwe funkcjonalności do rozwinięcia:

- Obsługa większej ilości wskaźników
- Dostosowanie aplikacji do rozdzielczości obrazu i wyświetlacza
- Wsparcie dla niepełnosprawnych
- Możliwość wyboru koloru tła
- Możliwość wyboru zdjęcia na tło
- Dodanie większej ilości efektów rysowania
- Możliwość wyboru urządzenia wejściowego
- Możliwość zapisu obrazów do różnych formatów i lokalizacji
- Zmiana sposobu detekcji
- Poprawki dotyczące wydajności
- Integralność z czujnikiem Kinect

Zaprezentowane przyszłe funkcjonalności stanowią tylko część możliwego potencjału rozwojowego aplikacji. Nie ma jednak gwarancji, że rozwój projektu będzie kontynuowany.

8. INSTRUKCJA OBSŁUGI

Dziękujemy za korzystanie z aplikacji BeMyBrush. Dokładne zapoznanie się z tą instrukcją pozwoli na wykorzystanie wszystkich możliwości aplikacji.

8.1. Wstęp

Aplikacja BeMyBrush to aplikacja przeznaczona na urządzenia z systemem operacyjnym Windows. Umożliwia ona wykonywanie prostych rysunków za pomocą gestów wykonywanych przy użyciu dedykowanego wskaźnika. Aby możliwe było korzystanie z aplikacji konieczne jest posiadanie następujących elementów:

- Komputera
- Monitora
- Kamery internetowej
- Wskaźnika do obsługi aplikacji
- Kończyn
- Źródła światła

8.2. Wskaźnik

Aby możliwe było korzystanie z programu konieczne jest posiadanie wskaźnika. Zalecana forma wskaźnika to rękawiczna z pomarańczowym punktem w centralnej części dłoni. Taka forma umożliwia łatwe przerywanie i wznowianie rysowania.



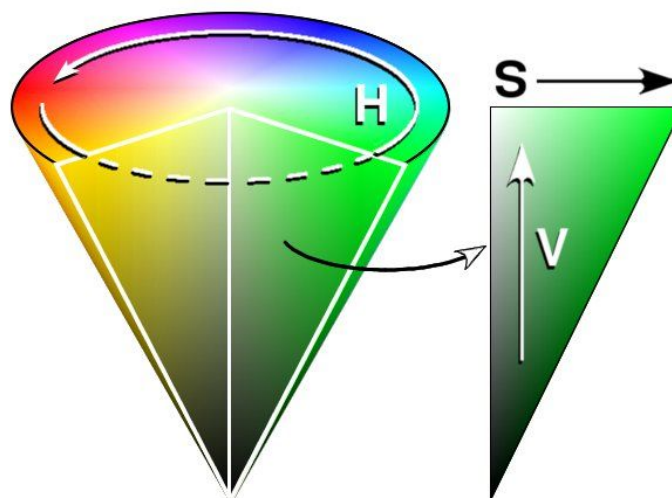
Rys.16. Wskaźnik wykorzystywany przez aplikację.

Program umożliwia zdefiniowanie innego koloru wskaźnika, jednak forma rękawiczki jest zalecana.

Aby móc korzystać z programu, należy założyć wskaźnik na dłoń w taki sposób aby pomarańczowy element był widoczny przez kamerę w komputerze. Optymalna odległość wskaźnika od kamery podczas używania aplikacji wynosi 0.5m – 2.0m. Wskaźnik posiada włącznik uruchamiający diodę która powinna być włączona podczas korzystania z programu.

8.3. Kalibracja

Przy każdym uruchomieniu aplikacji ukazuje się etap kalibracji. Pozwala on na dostosowanie wartości progowania do warunków oświetlenia i koloru wskaźnika. Progowanie polega na zdefiniowaniu górnego oraz dolnego przedziału przestrzeni barw HSV.

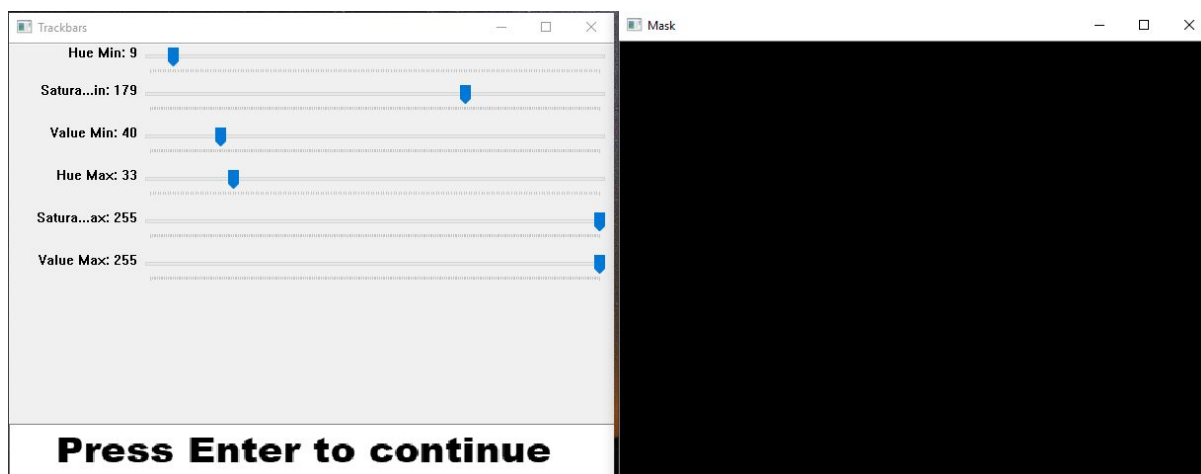


Rys.17. Wizualizacja przestrzeni barw HSV

Definiując progi określamy przedział barw w jakim ma się odbywać wykrywanie. W przypadku pomarańczowego wskaźnika z własnym źródłem światła są to odpowiednio:

- Próg dolny
 - Hue: **9**
 - Saturation: **179**
 - Value: **40**
- Próg górny
 - Hue: **33**
 - Saturation: **255**
 - Value: **255**

Do ustawienia wartości parametrów poszczególnych progów służą suwaki, których pozycja określa wartość danego parametru.



Rys.18. Etap kalibracji

Suwaki należy ustawić w taki sposób aby na ekranie "Mask", w miejscu występowania wskaźnika pojawił się biały punkt. Dla pewności kalibracji należy sprawdzić czy jest on widoczny w pięciu miejscach na ekranie - na środku oraz wszystkich narożnikach.

Jeżeli wskaźnik jest widoczny we wszystkich miejscach ekranu należy nacisnąć klawisz Enter w celu zatwierdzenia kalibracji. Zamknie to okna kalibracji, a następnie uruchomi okno właściwej aplikacji służące do rysowania.

Aplikacja pamięta ustawienia poprzedniej kalibracji, więc jeśli efekt jest zadowalający wystarczy nacisnąć enter.

8.3. Rysowanie

Po przeprowadzeniu kalibracji uruchomi się okno BeMyBrush umożliwiające rysowanie. Okno podzielone jest na dwa obszary: obszar podglądu z interfejsem użytkownika oraz obszar płótna na którym widzimy powstający obraz.





Rys.19. Okno BeMyBrush





Rysowanie odbywa się poprzez poruszanie wskaźnikiem przed kamerą, jest on aktywny zawsze wtedy kiedy znajduje się w polu widzenia kamery. Można go zasłonić i przerwać rysowanie w celu przeniesienia pozycji pędzla na inny fragment malowanego obrazu albo wybrania pozycji z interfejsu. Są na nim dostępne ustawienia grubości pędzla, wybór koloru oraz gumka do mazania, a także opcje czyszczenia płótna, zapisu i wyjścia.

8.4. Interfejs użytkownika

Obsługa interfejsu oparta jest na wykryciu pozycji wskaźnika, po najechaniu wskaźnikiem na pożądaną pozycję interfejsu. Aby uniknąć przypadkowego wybrania opcji z interfejsu obszar wykrywania jest nieco mniejszy niż wymiary ramki.

Interfejs zapewnia dostęp do następujących funkcji:

	Wyczyszczenie płótna w celu namalowania nowego obrazu
	Przełączenie pędzla w tryb gumki. Wybór zostanie zasygnalizowany czerwoną ramką.

	<p>Przełączenie pędzla w tryb malowania. Wybór zostanie zasygnalizowany czerwoną ramką.</p>
	<p>Zapis aktualnego stanu płótna. Wybór zostanie zasygnalizowany czerwoną ramką.</p>
	<p>Wyjście</p>
	<p>Wybór rozmiaru pędzla</p> <ul style="list-style-type: none"> - Grubość 30 pikseli - Grubość 25 pikseli - Grubość 20 pikseli - Grubość 10 pikseli - Grubość 5 pikseli <p>Wybór określonej grubości pędzla zostanie zasygnalizowany zmianą grubości punktu ukazującego pozycję wskaźnika, a także wybrany przycisk zostanie zaznaczony czerwoną ramką.</p>

	<p>Wybór koloru pędzla. Zostanie zasygnalizowany zmianą koloru pętli ukazującej pozycję wskaźnika.</p>
------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------

8.4. Kończenie pracy aplikacji

W celu zakończenia pracy aplikacji należy skorzystać z jednego z poniższych sposobów:

- Najechać wskaźnikiem na ikonę wyjścia w lewym dolnym rogu interfejsu
- Naciśnąć klawisz "Q" na klawiaturze
- Naciśnąć przy użyciu myszy przycisk "X" w oknie BeMyBrush