POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRYCZNY
Instytut Automatyki, Robotyki
i Inżynierii Informatycznej

Podstawy teleinformatyki

Malowanie obrazów biorąc pod uwagę ruch ciała - Páint

Grupa L2
Filip Matuszczak 131802
Ewa Dziembowska 131755
Małgorzata Marczyk 131333
Tomasz Jóskowiak 131774

14 czerwca 2019

Spis treści

1 Ogólna charakterystyka projektu

Głównym tematem projekt jest stworzenie aplikacji umożliwiającej rysowanie użytkownikowi za pomocą ruchu ręki. Obraz z kamery będzie przechwytywany i poddawany obróbce w celu rozpoznawania i odzwierciedlania ruchu obiektu służącego do rysowania.
Narzędziem umożliwiającym rysowanie będzie wskaźnik składający się z fioletowej kulki, umieszczonej np. na ołówku. Program przy użyciu biblioteki OpenCV wykrywa filetowy obiekt i wyznacza jego kontury oraz środek. W ten sposób współrzędne środka przesyłane są do modułu odpowiadającego za rysowanie w aplikacji. Aplikacja też pozwala użytkownikowi na rysowanie za pomocą myszki oraz na swobodne przełączanie się między trybem rysowania myszką a rysowania ręką. Dodatkowo użytkownik w każdej chwili ma możliwość zapisania swojej pracy.
Do tego użytkownik ma możliwość wyboru kolorów z dostępnej palety oraz innych narzędzi takich jak ołówek lub gumka. Możliwe jest również zmienianie grubości rysowanej linii. W każdej chwili użytkownik może zmienić sposób rysowania na rysowanie myszką za pomocą odpowiedniego przycisku lub skrótu klawiszowego.

2 Uzasadnienie wyboru

Projekt umożliwiał nam rozwój w technologiach wcześniej nam nieznanych co podniosło nasze umiejętności oraz wiedzę w dziedzinie przetwarzania obrazów oraz projektowania GUI. Poruszał również tematy, które mają częste zastosowania w dzisiejszych technologiach, takie jak, np. poruszanie kursorem za pomocą ruchu ciała. Uznaliśmy także, że efekt końcowy pracy będzie bardzo satysfakcjonujący i sprawi wiele radości użytkownikom. Pozwoli także na aktywne spędzanie czasu zachęcając do ruchu ciała użytkownika. Zachęciło nas też to, że bardzo mało na rynku jest podobnych programów, przez co mogliśmy poczuć się w pewien sposób wyjątkowo.

3 Wymagania

3.1 Funkcjonalne

1. Użytkownik porusza obiektem z fioletową końcówką w kształcie kuli, którego ruch jest odzwierciedlany w aplikacji.
2. Użytkownik może zmienić kolor pędzla.
3. Użytkownik może wymazywać namalowane elementy, wybierając narzędzie gumki.
4. Użytkownik może zapisać stworzony obraz.
5. Użytkownik może wyczyścić cały obraz i zacząć od nowa.
6. Użytkownik może przerwać proces rysowania.
7. Użytkownik może rozpocząć przechwytywanie ruchów.
8. Użytkownik może wybrać metodę rysowania za pomocą myszki lub za pomocą narzędzia.

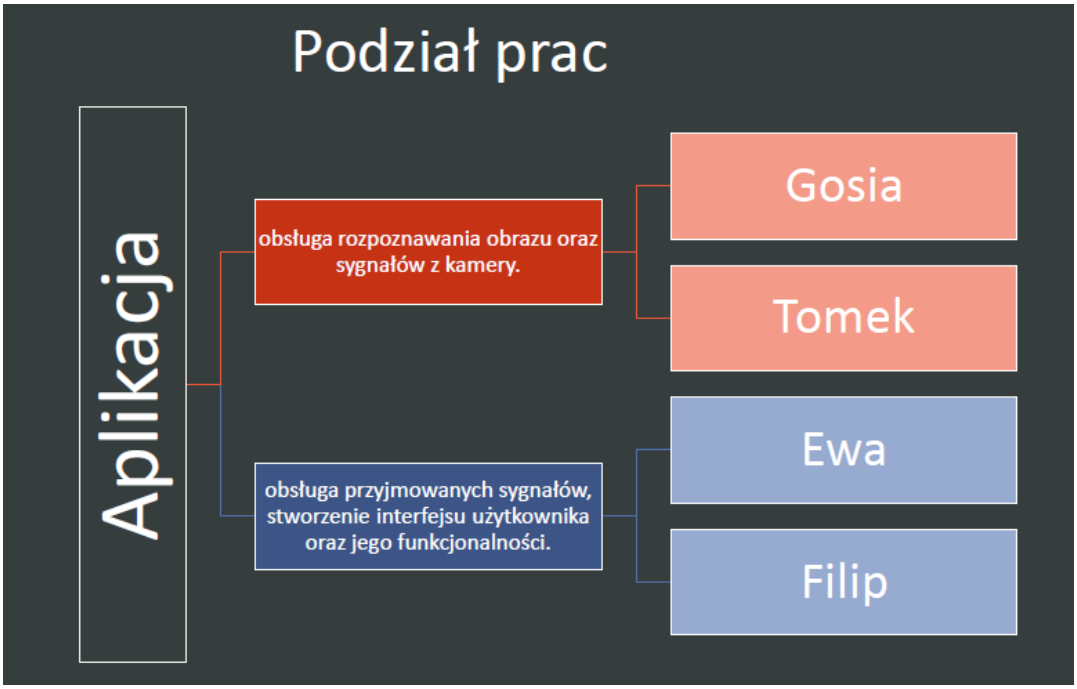
3.2 Niefunkcjonalne

1. Aplikacja musi być napisana w C++.
2. Aplikacja ma działać na systemach operacyjnych Windows 10.
3. Użytkownik musi posiadać kamerę komputerową.
4. Użytkownik powinien posiadać obiekt z fioletową końcówką w kształcie kuli w ręce.
5. Pomieszczenie w którym znajduje się użytkownik musi być dobrze oświetlone.
6. Moduł wykrywający ruch ręki powinien być maksymalnie odporny na zakłócenia.
7. Moduł rysujący powinien działać identycznie w trybie rysowania ruchem ręki i w trybie rysowania myszką.

4 Podział prac

W celu lepszej organizacji czasem oraz zagadnieniami, które należało wykonać nasza grupa postanowiła podzielić się na dwa podzespoły. Zespół złożony z Gosi oraz Tomka miał za zadanie przygotowanie algorytmu rozpoznawania fioletowego, kulistego obiektu. Trzeba było również opracować sposób przesyłania danych dotyczących położenia obiektu. Całość została napisana w języku C++ oraz z wykorzystaniem biblioteki OpenCV.
Zadaniem zespołu składającego się z Ewy oraz Filipa było opracowanie oraz implementacja interfejsu użytkownika wraz z funkcjonalnościami aplikacji umożliwiającymi pełną swobodę wykonywania rysunku. Składała się na to implementacja odpowiednich przycisków oraz ich funkcjonalności. Do tego zaprojektowanie własnych grafik dla konkretnych przycisków ułatwiających ich rozpoznawanie. W tym celu zespół wykorzystał język C++ oraz bibliotekę SFML.

- Gosia oraz Tomek - obsługa rozpoznawania obrazu oraz sygnałów z kamery.
- Ewa oraz Filip - obsługa przyjmowanych sygnałów, stworzenie interfejsu użytkownika oraz jego funkcjonalności.



Rysunek 1: Podział prac przy aplikacji

5 Technologie

Wybór technologi nie sprawił większych problemów, ponieważ istnieje wiele dodatkowych bibliotek dla języka C++. Nasza grupa zdecydowała się na wykorzystanie dwóch konkretnych i popularnie stosowanych w podobnych problemach. Wybrane przez nas biblioteki to OpenCV oraz SFML, które zostaną szerzej opisane w podpunktach tego działu.

5.1 OpenCV

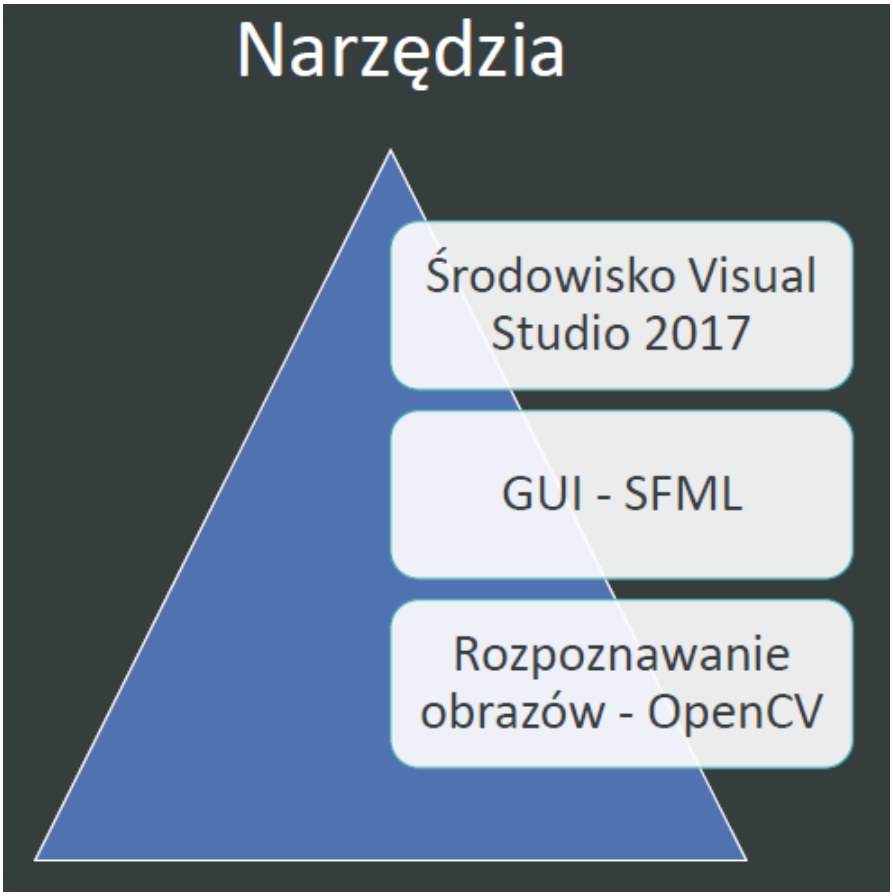
Rysunek 2: OpenCV

Do rozpoznawania obiektu została wykorzystana biblioteka OpenCV w wersji 3.4. Jest to biblioteka wykorzystywana do obróbki obrazu oraz oparta na otwartym kodzie. Obraz przetwarzany jest w czasie rzeczywistym. Uzasadnieniem wyboru przedstawionej biblioteki jest fakt iż oferuje ona wydajne i proste z użyciu metody umożliwiającej przetwarzanie obrazu w czasie rzeczywistym. Do tego obszernie opisana dokumentacja oraz wiele ogólnodostępnych tutoriali również miały wpływ na wybór biblioteki OpenCV.

5.2 SFML

Rysunek 3: SFML

Do utworzenia interfejsu użytkownika została użyta biblioteka SFML. SFML zapewnia prosty interfejs do różnych komponentów, która ułatwia tworzenie gier oraz aplikacji użytkowych. Składa się z pięciu modułów: system, window, graphics, audio i network.

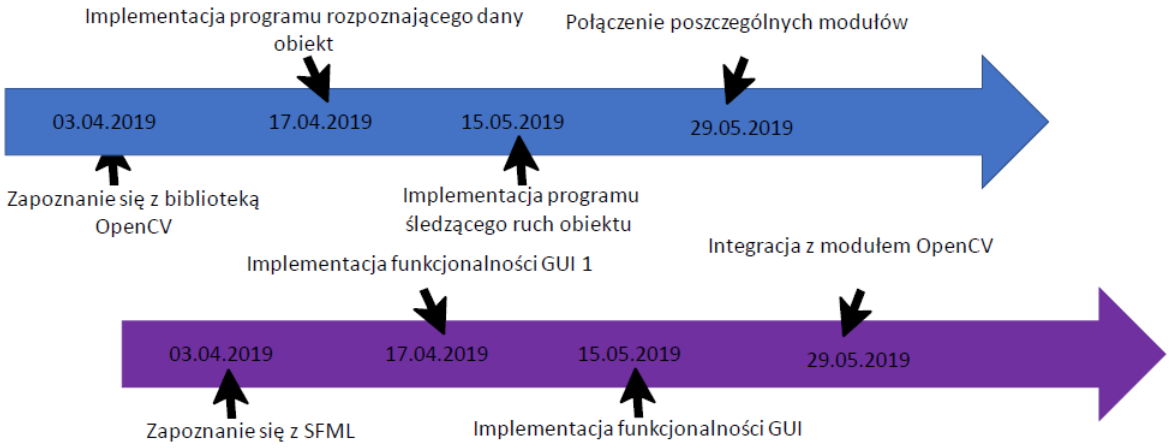


Rysunek 4: Narzędzia wykorzystane przy implementacji aplikacji

6 Oś czasu realizacji zadań

Realizacja wybranego przez nasz zespół projektu była złożona i wymagała dużych nakładów czasowych ze strony członków zespołu. W celu sprawniejszej kontroli nad postępem prac oraz wyglądem projektu każdy z podzespołów wyznaczył terminy realizacji poszczególnych zadań.
Wszystkie terminy pokrywały się z datami zajęć projektowych. W ten sposób wszelkie postępy mogły być zaprezentowane i ocenione przez prowadzącego oraz pozostałe zespoły w celu konsultacji ewentualnych poprawek lub implementacji zaproponowanych rozwiązań.
Zastosowanie podziału zadań projektowych na konkretne terminy czasowe

poskutkowało sprawniejszą implementacja poszczególnych modułów. Do tego kontrola postępu prac oraz jakości produkowanego kodu była łatwiejsza i przyjemniejsza do realizacji.



Rysunek 5: Osie czasu realizacji zadań przez zespoły.

7 Funkcje wykorzystane w implementacji programu

7.1 Interfejs graficzny

1. Window (VideoMode mode, const String &title,
   Uint32 style=Style::Default,
   const ContextSettings &settings=ContextSettings())

   Funkcja tworząca okno interfejsu z odpowiednimy parametrami.

2. void setFramerateLimit (unsigned int limit)

   Funkcja kotroluje częstotliwość odświeżania okna.

3. void setPrimitiveType (PrimitiveType type)

   Ustawia typ dla klasy VertexArrray.

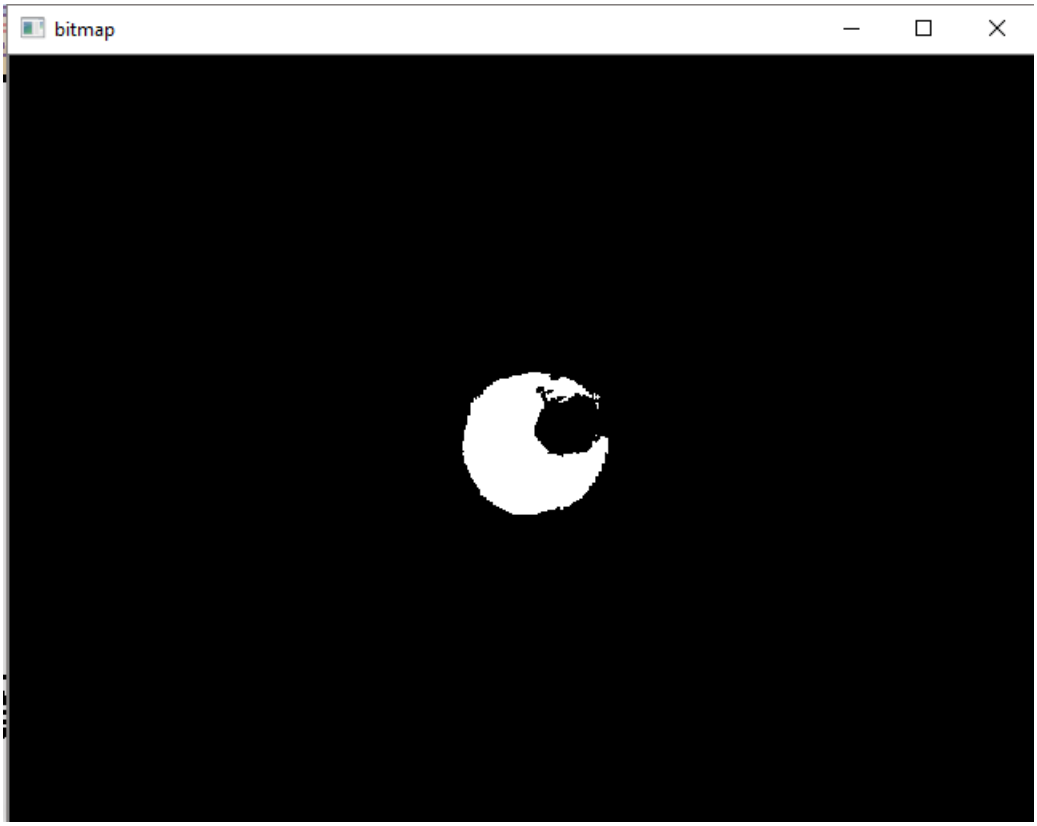
7.2 Metody OpenCV

1. void cvtColor(InputArray src, OutputArray dst, int code)

   Funkcja konwertuje obraz wejściowy (argument src) z jednego standardu opisu obrazu na inny, tzn. (RGB -> HSV, BGR -> RGB). Obraz wyjściowy jest reprezentowany poprzez argument dst, a rodzaj konwersji określa argument code.

2. `void inRange(InputArray src, InputArray lowerb, InputArray upperb, OutputArray dst)`

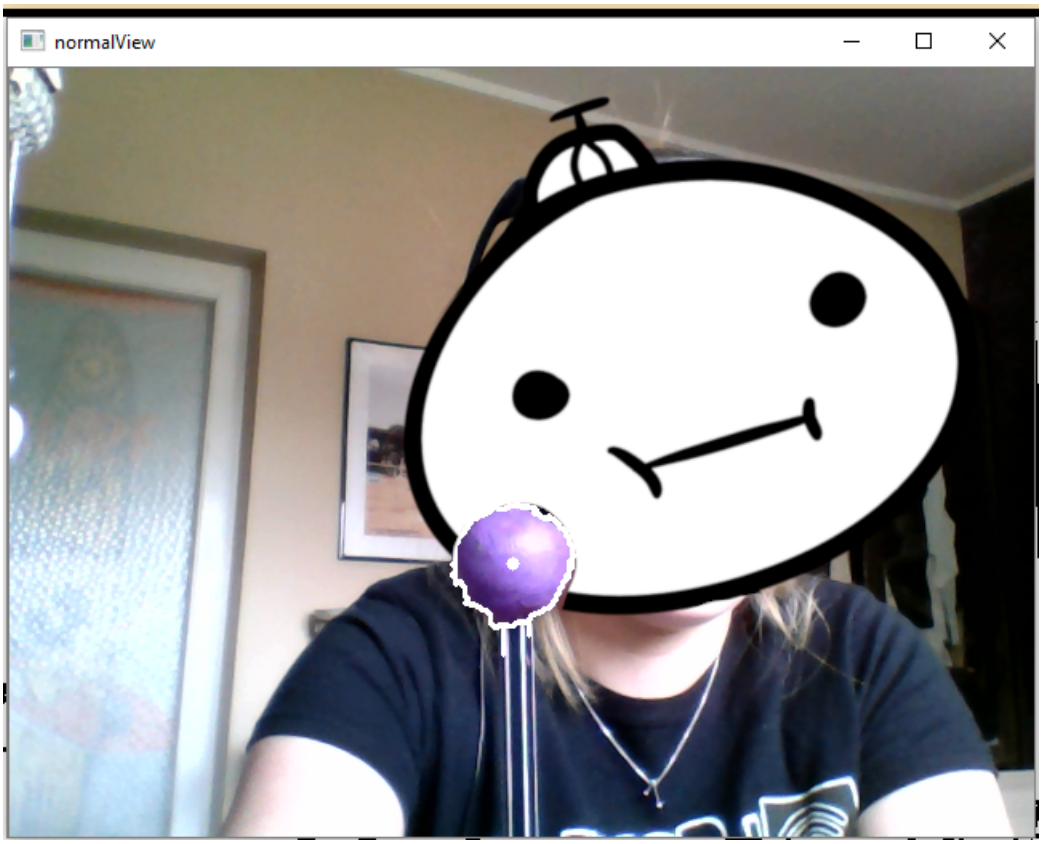
   Funkcja wyszukuje w obrazie wejściowym (argument src) bitów w kolorze z przedziału wyznaczonego za pomocą argumentów lowerboundary i upperboundary. Obraz wyjściowy (argument dst) jest swego rodzaju maską dla zakresu znajdującego się pomiędzy granicami zakresu.



Rysunek 6: Maska z zaznaczonym obietkem

3. `void findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())`

   Funkcja ta znajduje kontury (argument contours) w obrazie wejściowym (argument image). Mode określa formatowanie w jakim zostaną zwrócone kontury, a method sposób znajdowania konturów.



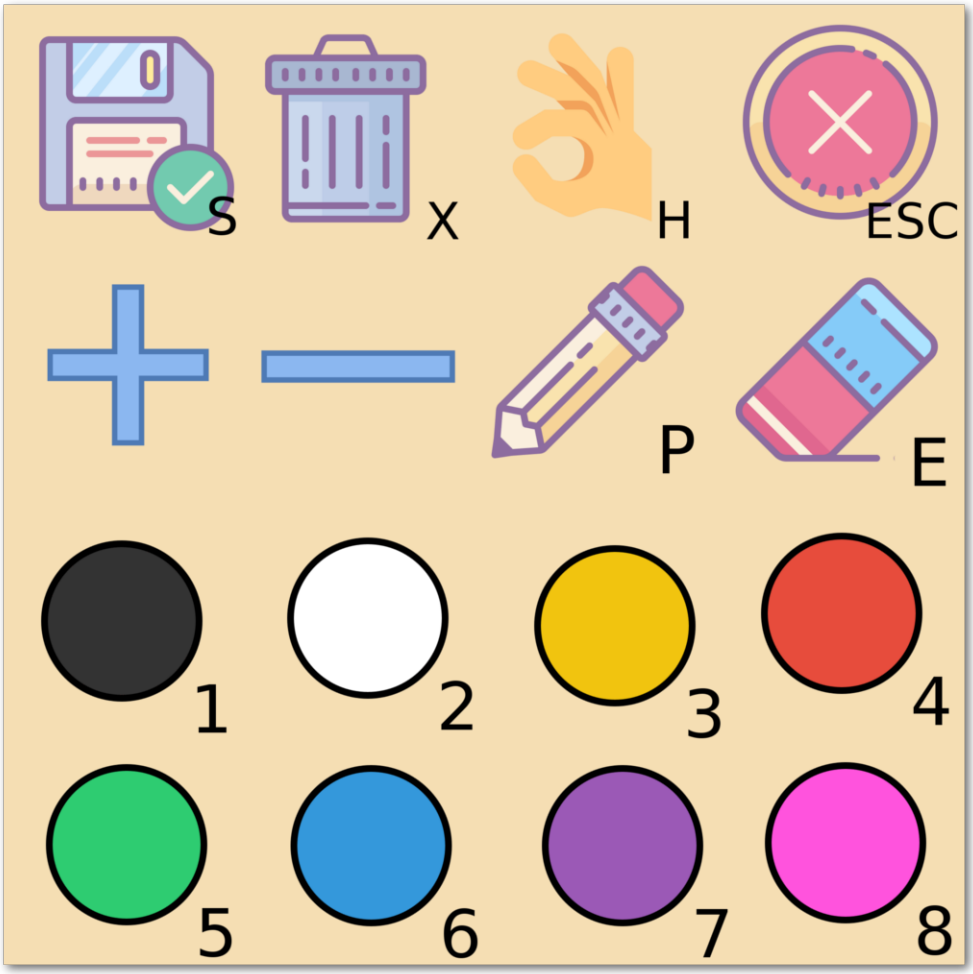
Rysunek 7: Zaznaczenie konturów wykrywanego obiektu

8 Interfejs

Interfejs programu dzieli się na dwie części:

8.1 Pasek narzędzi

Na pasku znajduje się szesnaście przycisków, które można aktywować za pomocą przycisku myszy, lub skrótu klawiszowego. Każda ikona posiada literę lub liczbę, wskazującą który klawisz odpowiada za daną opcje.

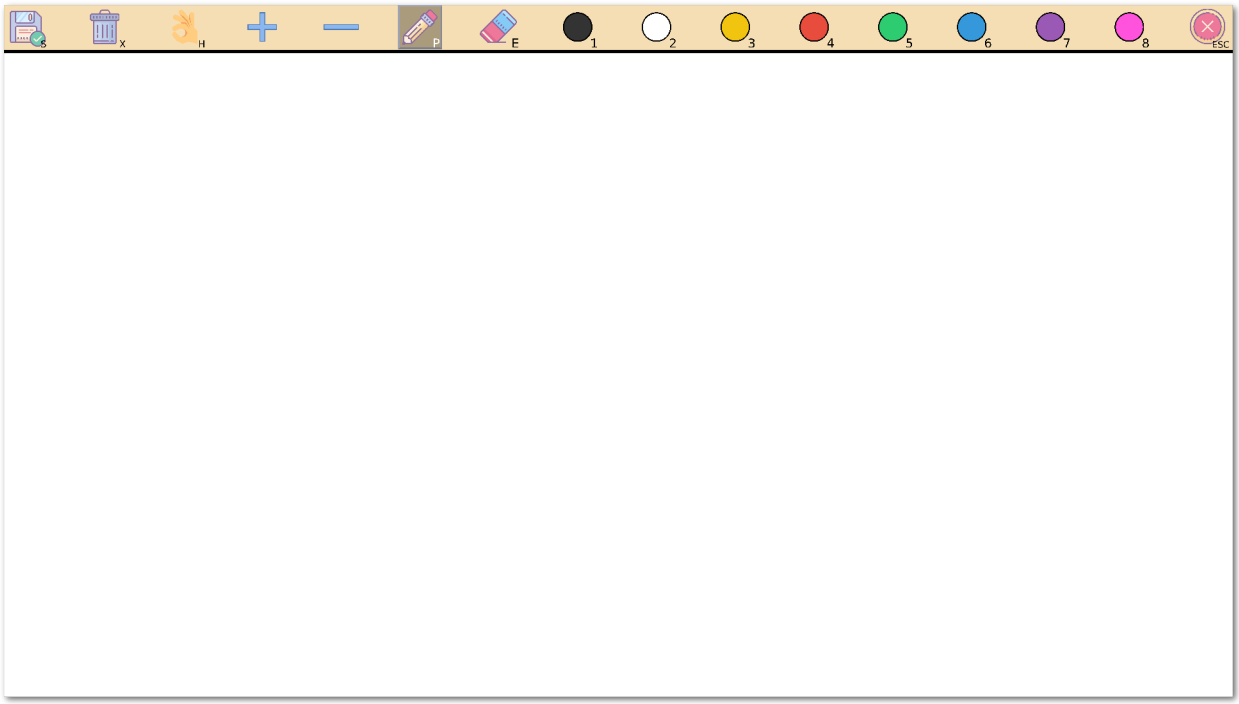


Rysunek 8: Dostępne ikony programu

- Dyskietka (S) - zapisanie aktualnego rysunku w pliku „screen.png”.
- Kosz (X) - wyczyszczenie całego płótna i umożliwia rozpoczęcie pracy od nowa.
- Dłoń (H) - rozpoczęcie bądź zakończenie przechwytywania ruchu z kamery.
- Plus (+) - powiększenie promienia pędzla.
- Minus (–) - zmniejszenie promienia pędzla.
- Ołówek (P) - wybranie narzędzia rysującego - ołówka.
- Gumka (E) - wybranie narzędzia gumki, wymazującej ołówek.
- Kolory (1-8) - Każda ikona odpowiada za zmianę aktualnego koloru ołówka, zmieniając go w kredkę.
- Wyjście (ESC) - zamknięcie programu.

8.2 Płótno

Płótno jest polem po którym można rysować i którego zawartość może zostać zapisana lub wyczyszczona. Ruch użytkownika zamyka się w granicach płótna, nie pozwalając mu na rysowanie poza nim.



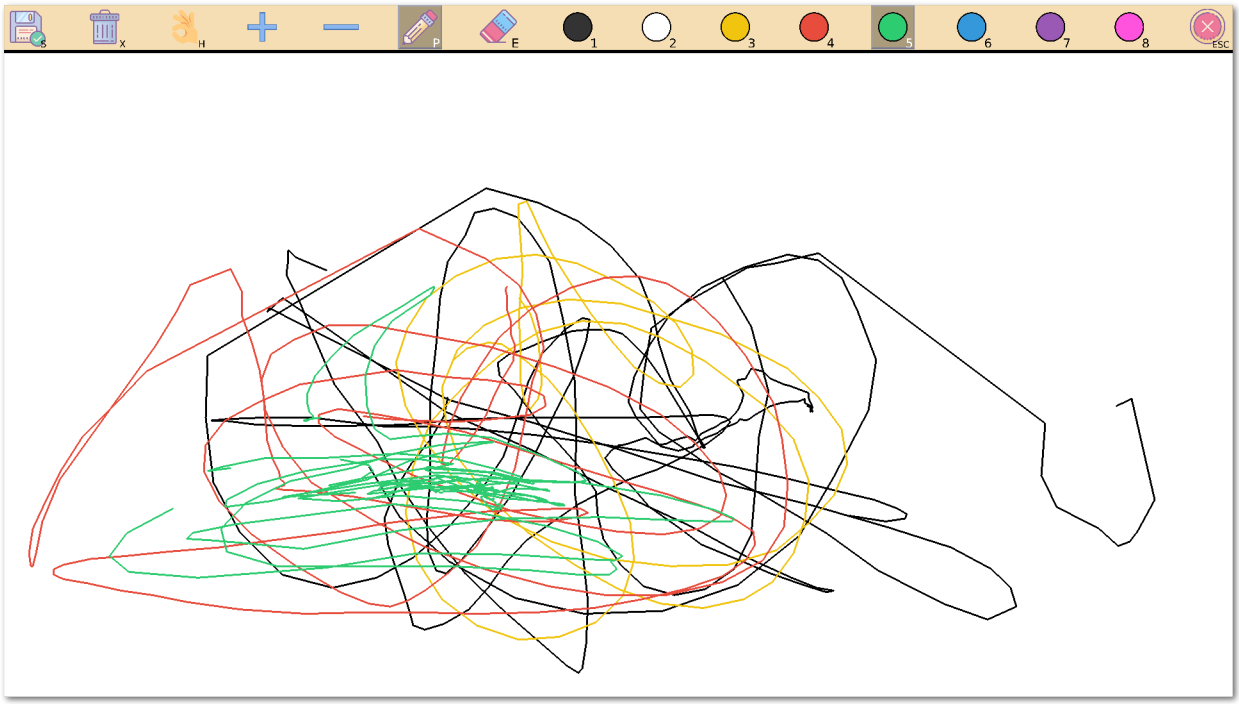
Rysunek 9: Interfejs programu

9 Instrukcja obsługi

Użytkownik może rysować na dwa sposoby: za pomocą myszy bądź ruchu ciała. Po pierwszym uruchomieniu wyświetli się okno gotowe do pracy aplikacji sterowane myszą. Za pomocą kursora można wybierać poszczególne przyciski w pasku narzędzi, które mają wpływ na wielkość oraz kolor ołówka. Dostępna jest także możliwość zapisania bądź usunięcia projektu.

Rysunek 10: Sterowanie programem myszą

Aby rozpocząć przechwytywanie ciała, użytkownik musi wybrać przycisk ręki, bądź wcisnąć klawisz H. Aplikacja posługuje się kamerą domyślną, która jest podłączona do komputera. Po uruchomieniu przechwytywania program stara się odnaleźć wskaźnik w kształcie fioletowej kulki. Ruch wskaźnika, którym porusza użytkownik, odzwierciedlany jest na ekranie, pozostawiając ślad na płótnie.



Rysunek 11: Sterowanie programem ruchem ciała

W trybie przechwytywania obrazu użytkownik nadal ma możliwość korzystania z paska narzędzi, bez przełączania się na sterowanie myszą. W tym celu może posługiwać się skrótami klawiszowymi. Każdy przycisk ma wypisaną nazwę klawisza, do którego jest przypisany. Jeśli użytkownik korzysta z małego, niebieskiego ołówka, to wciśnięcie klawiszy $+$ $+$ 4 w dowolnej kolejności powiększy kursor dwa razy oraz zmieni kolor na czerwony.
Ponownie wybranie przycisku H przełącza z powrotem na tryb sterowania myszą.

10 Problemy

10.1 Napotkane problemy

1. Największym problemem był poprawny wybór i kalibracja koloru wskaźnika.
2. Należało także znaleźć sposób na filtrowanie zakłóceń spowodowanych przez inne obiekty.
3. Algorytm na powiększanie rozmiaru ołówka.
4. Problemy wydajnościowe po zwiększaniu ołówka.
5. Sterowanie interfejsem aplikacji w momencie włączenia przechwytywania kamery.

10.2 Rozwiązania problemów

1. Kolor musiał być kolorem rzadko występującym w otoczeniu, aby uniknąć niepotrzebnych zakłóceń. Ostatecznie decyzja padła na kolor fioletowy ze względu na jego rzadkie występowanie.
2. Filtrowanie zakłóceń zostało rozwiązanie poprzez dodanie dolnej granicy powierzchni konturów, które są brane pod uwagę podczas wyszukiwania konturów.
3. Udało się sformułować poprawny i optymalny algorytm.
4. Regularne zapisywanie stanu okna oraz zwalnianie miejsca w wektorze.
5. Dodanie skrótów klawiszowych.

11 Najciekawsze fragmenty kodu

Listing 1: Decyzja czy rysować przy przechwytywaniu

```
if (confideceLevel > 5)
{
  if (purpleDetector->getX() > 0 &&
    purpleDetector->getY() > 0 &&
    purpleDetector->getX() < window.getSize().x &&
    purpleDetector->getY() < window.getSize().y)
     {

       float x_proportion =
       (float)window.getSize().x /
       (float)purpleDetector->getResolution().x;

       float y_proportion =
       (float)window.getSize().y /
       (float)purpleDetector->getResolution().y;

       lastPointerPos =
  sf::Vector2f(purpleDetector->getX()*x_proportion,
       purpleDetector->getY()*y_proportion);

       draw(lastPointerPos, curr_col, window,
       vertices, size);

    }
}
```

Decyzja czy rysować przy przechwytywaniu jest podejmowana na podstawie zmiennej confidenceLevel. W momencie przechwytywania zwiększamy jej wartość o 1. Gdy 5 razy GUI otrzyma informację, że obiekt został znaleziony, to dopiero wtedy zaczyna rysować. Takie zabezpieczenie w znacznym stopniu maskuje niedoskonałości wykrywania fioletowego koloru i zakłócenia praktycznie w zerowym stopniu wpływają na rysowanie. Gdy detektor zgubi obiekt przynajmniej raz, to zmienna confidenceLevel jest znów ustawiana

na 0. W celu naprawienia różnicy między rozdzielczością kamery a rozdzielczością okna GUI dzielimy rozdzielczość okna przez rozdzielczość kamery i finalną pozycję mnożymy przez uprzednio otrzymaną wartość.

Listing 2: Zapisywanie stanu okna i czyszczenie wektora

```
currentWindow.create(window.getSize().x,
window.getSize().y);
currentWindow.update(window);

  for (auto &vertice : vertices) {
        vertice.clear();
        sf::VertexArray arr;
        arr.setPrimitiveType(sf::LinesStrip);
        vertice.push_back(arr);
        }
```

Z powodów optymalizacyjnych nie możemy sobie pozwolić na trzymanie w wektorze wszystkich linii i musimy go regularnie czyścić. Najpierw tworzona jest tekstura, która zapisuje aktualny stan okna. Później w pętli for czyścimy wszystkie wektory przechowujące linie.

Listing 3: Przechwycenie konturów znajdujących się na obrazie

```
Mat frame, frame_HSV, frame_threshold;

cap >> frame;

flip(frame, frame, +1);

// Convert from BGR to HSV colorspace
   cvtColor(frame, frame_HSV, COLOR_BGR2HSV);

// Detect the object based on HSV Range Values
   inRange(frame_HSV, lowerBoundary, upperBoundary,
    frame_threshold);


std::vector<std::vector<Point>> contours = {};
std::vector<Vec4i> hierarchy = {};
findContours(frame_threshold, contours, hierarchy,
            RETR_EXTERNAL, CHAIN_APPROX_SIMPLE);
```

Obraz przechwytywany przez kamerę jest odbijany horyzontalnie, aby ułatwić zczytywanie kierunku ruchu obiektu. Następnie zmieniana jest przestrzeń kolorów z BGR na HSV. Ułatwia to identyfikowanie przejść pomiędzy barwami. Następnie nakładana jest maska na kolor we wcześniej zdefiniowanym zakresie przestrzeni HSV. Ostatecznie ściągane są kontury, które zostały wykryte po nałożeniu maski.

Listing 4: Znalezienie największego konturu będącego ponad treshholdem

```
if (contours.size() > 0)
 {
  std::sort(contours.begin(), contours.end(),
  compareContourAreas);
  if (fabs(contourArea(
  cv::Mat(contours[contours.size() - 1]))) > 50.0)
   {
   Moments moment =
   moments(contours[contours.size() - 1], false);

   Point2f mass_centre(moment.m10
   / moment.m00, moment.m01 / moment.m00);
   x = mass_centre.x;
   y = mass_centre.y;

   std::cout <<
   fabs(contourArea
          (cv::Mat(contours[contours.size()-1])))
              << std::endl;
   drawContours(frame, contours,
      contours.size() - 1, { 255,255,255 },
        2, 8, hierarchy, 0, Point());
   circle(frame, mass_centre,
      4, { 255,255,255 }, -1, 8, 0);
   }
 }
```

Otrzymane kontury są później sortowane według ich powierzchni. Następnie brany jest największy z konturów i sprawdzane jest czy jego powierzchnia wynosi więcej niż dolna granica przyjmowania konturów. Jeśli kontur spełnia powyższy warunek, wyliczane są jego momenty. Następnie na podstawie momentów określany jest środek wykrytego konturu. Jest on przekazywany do GUI, który wykorzystuje współrzędne do późniejszego rysowanie.

12 Implementacja przycisków

12.1 Klasa Button

Wszystkie przyciski w projekcie dziedziczą z klasy abstrakcyjnej Button. Klasa Button posiada metodę abstrakcyjną action(), która definiuje, co powinno się dziać w momencie kliknięcia w przycisk. Każdy przycisk posiada 3 stany - normal, hovered i clicked. Przy każdym z tych stanów zmienia się wygląd przycisku. Wygląd przycisków zrealizowano za pomocą klasy z biblioteki SFML - sf::Texture. Tekstury ładowane są z plików. W celu odróżniania stanów wprowadzono zmienna typu enum przyjmującą analogicznie 3 stany wartości. Button posiada także skalę, pozwalającą na modyfikację wielkości przycisku.

Listing 5: Implementacja klasy Button

```
class Button {
public:
        enum state {state_clicked,
        state_hovered,
        state_normal};
        Button(sf::Texture* normal,
        sf::Texture* clicked,
        sf::Texture* hovered,
        sf::Vector2f location,
        float scale);
        void checkHover(sf::Vector2f mousePos);
        void checkNormal(sf::Vector2f mousePos);
        bool checkClicked(sf::Vector2f mousePos);
        void setState(state);
        state getState();
        sf::Sprite* getSprite();
        virtual void action() = 0;
        virtual sf::Color getColor() = 0;
        bool isClicked();
private:
        sf::Sprite normal;
        sf::Sprite clicked;
        sf::Sprite hovered;
        sf::Sprite* currentSpr;
        float scale;
        state current;
};
```

12.2 Inicjalizacja klasy Button

Przy inicjalizacji przycisku należało podać 3 ścieżki do obrazków obrazujących 3 stany przycisku. Dodatkowo należało też podać jego pozycję, jego rozmiar oraz wskaźnik na okno programu. Umieszczenie przycisku w znacznym stopniu zależne jest od rozmiaru okna i rozdzielczości monitora.

Listing 6: Przykładowa inicjalizacja przycisku wyjścia

```
ExitButton getExit(
int i,
int windowSize,
sf::Window * window)
{
        Texture* texture_normal = new Texture();
        Texture* texture_hover = new Texture();
        Texture* texture_clicked = new Texture();

        float x = windowSize / divider * i;
        float y = 0;

        loadTexture(*texture_normal,
        "./icons/exit_free.png");
        loadTexture(*texture_hover,
        "./icons/exit_hover.png");
        loadTexture(*texture_clicked,
        "./icons/exit_hover.png");

        return ExitButton(
        texture_normal,
        texture_clicked,
        texture_hover,
        sf::Vector2f(x, y),
        scale,
        window);
}
```

13 Rysowanie w SFML

13.1 Ogólny opis rysowania

Projekty w bibliotece SFML są wyświetlane z pewną częstotliwością. Częstotliwość odświeżania sprawia, że rysowanie za pomocą szybkich ruchów myszki/narzędzia było mocno problematyczne do odwzorowania. Nie można było zastosować najprostszego rozwiązania, czyli rysowanie danego kształtu tam, gdzie aktualnie znajduje się myszka, gdyż przy szybkich ruchach (takich, które przekraczały szybkość odświeżania) pojawiały się przerwy w linii rysowania. Najlepszym rozwiązaniem okazało się zastosowanie konkretnego typu prymitywnego obiektu Vertex - LineStripe'a. Jego zasada jest bardzo prosta - przyjmuje on 2 punkty i rysuje między nimi linie. W momencie, gdy ruch naszej ręki przegania prędkość odświeżania ekranu, zostanie narysowana prosta linia. Jest to pewnego rodzaju uproszczenie, które przez użytkownika w żadnym stopniu nie jest zauważalne, a w znacznym stopniu upraszcza obsługiwanie rysowania ręką.

13.2 Problemy podczas implementacji

Problem pojawił się także zwiększaniu grubości ołówka. Ostatecznie została podjęta decyzja o przechowywaniu w wektorze paru zbiorów Vertexów typu LineStrip. Upraszczając, zwiększając rozmiar pędzla, zwiększamy ilość LineStripe'ów w wektorze typu VertexArray typu LineStrip. Następstwem takiego rozwiązania były problemy wydajnościowe, wraz z rysowaniem, do pamięci podręcznej ładowane było coraz więcej danych, przez co w programie zauważalne było coraz większe opóźnienie.

13.3 Rozwiązanie problemu wydajnościowego

Rozwiązanie problemu wydajnościowego okazało się dosyć prozaiczne. Zdecydowano się na regularne zapisywanie stanu ekranu przy jednoczesnym zwalnianiu Vectora VertexArray'ów. Momentalnie naprawiło to problemy wydajnościowe.

`sf::LineStrip`	A set of connected lines. The end vertex of one line is used as the start vertex of the next one.	

Rysunek 12: Fragment dokumentacji SFML mówiący o LineStrip

14 Zapisywanie obrazu do pliku

Zapisywanie aktualnej pracy użytkownika polega na przechowaniu okna programu w teksturze, która następnie jest zapisywana w formacie png. W takim jednak wypadku na zdjęciu wciąż widoczny jest górny pasek narzędzi. Biblioteka SFML nie zezwala na takie modyfikowanie tekstury w trakcie jej użytkowania (po utworzeniu), które pozwoliłoby łatwo wyciąć górny panel. Z tego powodu wykonywany jest podwójny zapis. Pełny obraz programu zostaje zapisany, aby następnie go otworzyć i wyciąć zbędne fragmenty, pozostawiając wyłącznie płótno zawierające prace użytkownika. Tak zmodyfikowany obraz jest zapisywany ponownie na miejsce poprzedniego. Największym minusem tego rozwiązania jest wydłużony czas zapisu.

15 Integracja modułu rysowania z modułem wykrywania ruchów ręki

Realizując integracje modułu rysowania z modułem wykrywania ruchów ręki należało przyjąć parę założeń:

- Moduł wykrywający ruch nie był doskonały i działał mocno niedeterministycznie,
- Program rysujący próbował przy każdym obiegu pętli łączyć ze sobą poprzednie i aktualne położenie kursora,
- Moduł wykrywający ruch dość mocno spowalniał prace komputera i całego programu.

Biorąc pod uwagę powyższe założenia, integracja obu modułów nie była tak trywialna, jak z góry założono. Jeden z mechanizmów, jaki został wymyślony, to wprowadzanie autorskiego algorytmu maksymalnej prędkości (nazwa własna). Polega on na wprowadzeniu zmiennej confidenceLevel. Gdy moduł wykrywający ruch ręki wykryje narzędzie z fioletową kulką, zwiększa confidenceLevel o 1. Gdy zmienna confidenceLevel przekroczy pewną wartość, dopiero wtedy rysowanie się rozpoczyna. W znacznym stopniu ogranicza to niedoskonałość modułu wykrywającego, który co jakiś czas mógł przesyłać pewne zakłócenia. W momencie, gdy narzędzie chociaż na krótki przestanie być wykrywane, confidenceLevel zostaje z powrotem ustawiony na 0 i cały algorytm powtarza się.

16 Podsumowanie

Projekt udało się zrealizować według wszystkich początkowych założeń.
Wszystkie wymagania funkcjonalne zostały spełnione. Pomimo wielu problemów, które zostały opisane powyżej, program działa bardzo dobrze. Uwarunkowania sprzętowe (np. kamera) mają wpływ na działanie programu. Wydajność programu dzięki zastosowaniu języka C++ okazała się bardzo zadowalająca, przez co projekt ma bardzo dobre wrażenia użytkownika i sprawia, że chce się go częściej używać. Wykorzystane w projekcie biblioteki OpenCV oraz SFML okazały się idealne do realizacji poszczególnych zadań.

Posiadały bogaty wybór klas, funkcji czy tez metod wbudowanych. Dodatkowo możliwość zapisywania swoich dzieł sprawia, że przy drobnych szlifach projekt mógłby stać się szanowanym produktem typu freeware.