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

28 maja 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 poddawnay 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.
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 użutkownik może zmnieć sposób rysowania na rysowanie myszką za pomoca 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, np. poruszanie kursorem za pomocą ruchu ciała.

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.

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.

4 Podział prac

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

5 Technologie

Jako środowisko zostało użyte Visual Studio 2017.

5.1 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 motedy umozliwiające przetwarzanie obrazu w czasie rzeczywistym. Do tego obszernie opisana dokumentacja oraz wiele ogólnodostępnych tutoriali również miały wpływ na wybór biblioteki OpenCV.

5.2 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.

6 Funkcje wykorzystane w implementacji programu

6.1 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.

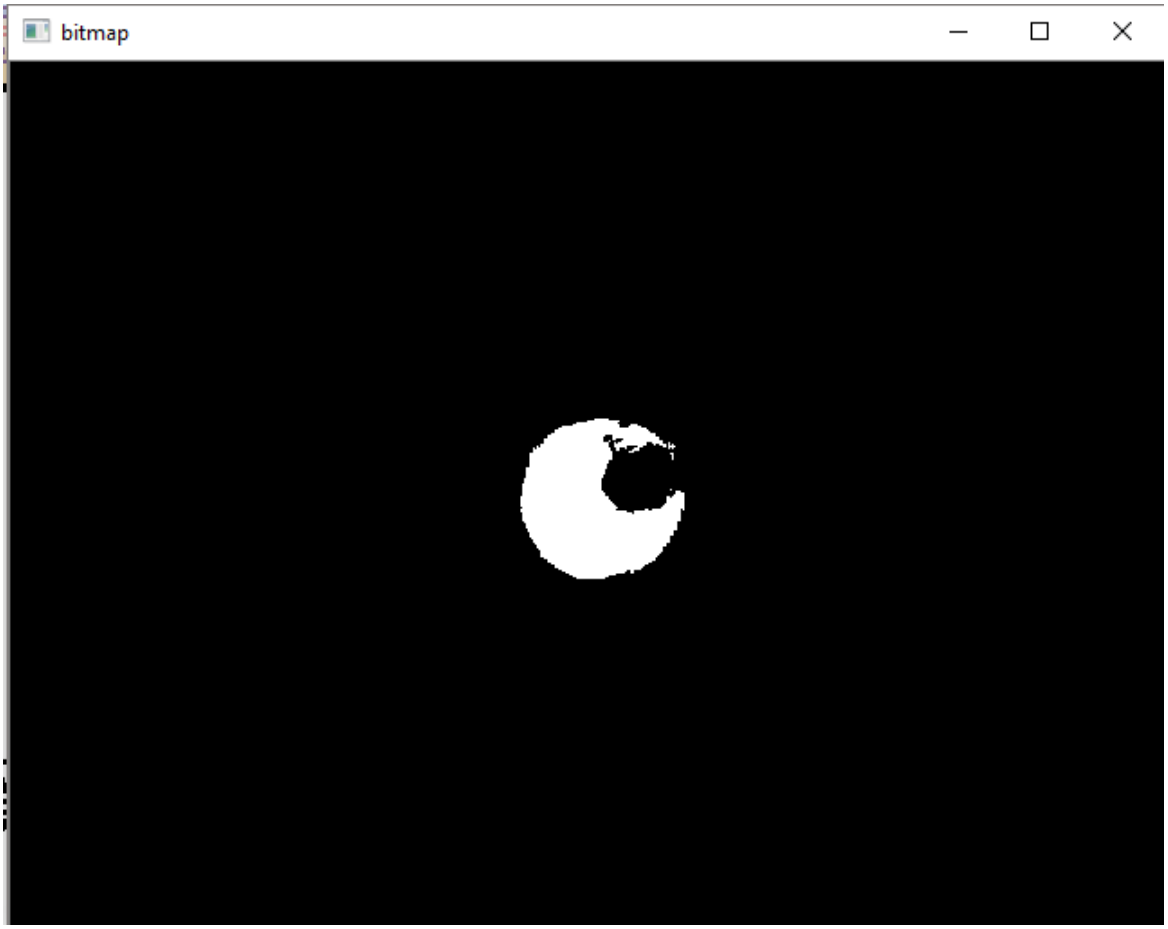
6.2 Metody OpenCV

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)`

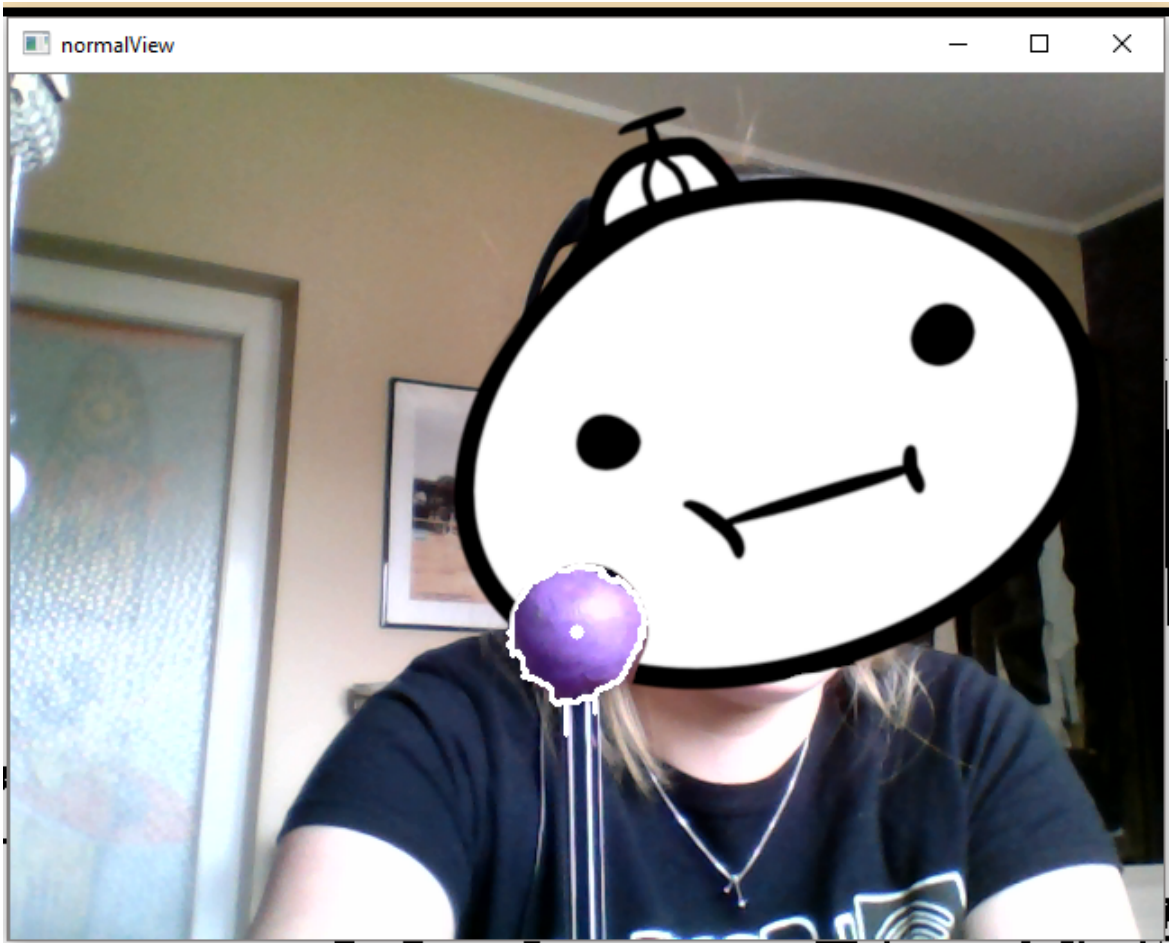
   Funkcja wyszukuje w obrazie wejściowym (argument src) bitów w kolorze z przedziału wyznaczonego za pomocą argumentów lowerboundary i upperboundary. Obraz wyjściowy (argument dst) jest swego rodzaju maską dla zakresu znajdującego się pomiędzy granicami zakresu.



Rysunek 1: 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.



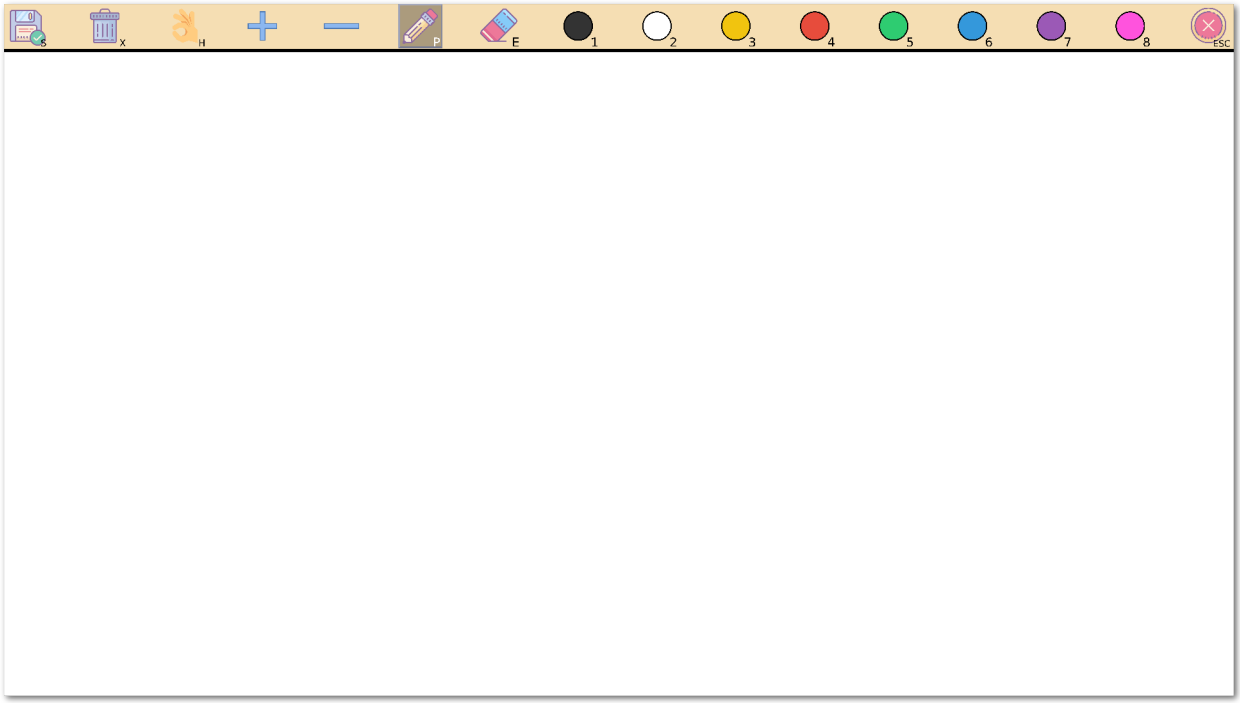
Rysunek 2: Zaznaczenie konturów wykrywanego obiektu

7 Interfejs

Interfejs programu dzieli się na dwie części:

1. Pasek narzędzi
   Na pasku znajduje się szesnaście przycisków, które można aktywować za pomocą przycisku myszy, lub skrótu klawiszego. Każda ikona posiada litere lub liczbę, wskazującą który klawisz odpowiada za daną opcje.

   - 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 zmiane aktualnego koloru ołówka, zmieniając go w kredke.
   - Wyjście (ESC) - zamknięcie programu.



Rysunek 3: Interfejs programu

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

8 Problemy

8.1 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.

8.2 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 sformuować poprawny i optymalny algorytm.
4. Regularne zapisywanie stanu okna oraz zwalnianie miejsca w wektorze.
5. Dodanie skrótów klawiszowych.

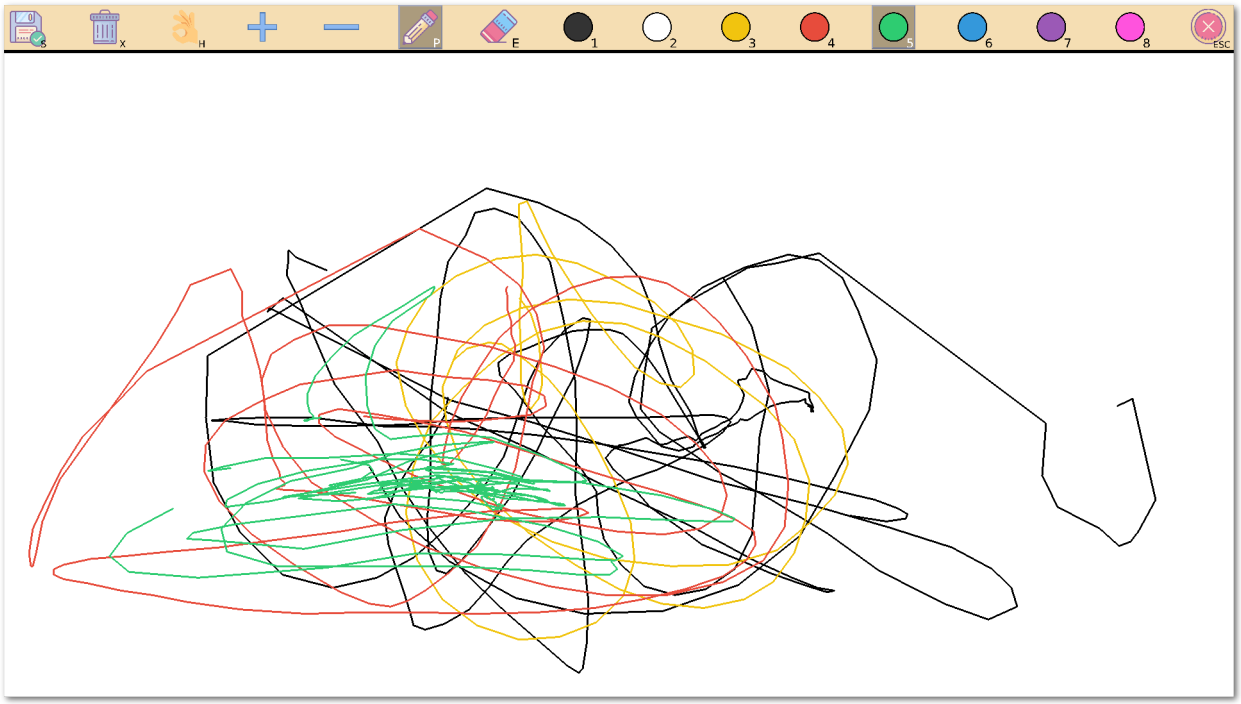
9 Instrukcja obsługi

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 4: 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 5: 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 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 jest tworzona 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 zczytwanie kierunku ruchu obiektu. Następnie zmieniana jest przestrzeń kolorów z BGR na HSV. Ułatwia to identyfikowanie przejść pomiędzy barwami. Nastepnie 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 sprwadzane jest czy jego powierzchnia wynosi więcej niż dolna granica przyjmowania komnturów. Jeśli kontur spełnia powyższy warunek, wyliczane są jego momenty. Nastepnie na podstawie

momentów określany jest środek wykrytego konturu.Jest on przekazywany do GUI, który wykorzystuje współrzędne do późniejszego rysownaie.

11 Podsumowanie

Projekt udało się zrealizować według wszsytkich 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.