Politechnika Poznańska

Wydział Elektryczny

Informatyka

Malowanie obrazów biorąc pod uwagę ruch ciała: Kinart

Autorzy:

Małgorzata Hanyż

Damian Szkudlarek

Cezary Waligóra

14 czerwca 2019

Spis treści

1 Opis aplikacji

Kinart to aplikacja, która ma na celu rozwijanie wyobraźni i przelewanie obrazów wykreowanych w głowie na cyfrowe płótno. Wszystko to za sprawą kilku machnięć dłonią. Urządzenie ***Microsoft Kinect*** pozwala użytkownikowi na interakcję z wirtualnym płótnem bez konieczności używania myszki, poprzez interfejs wykorzystujący gesty wykonywane przy pomocy dłoni. Jest to idealne rozwiązanie dla:

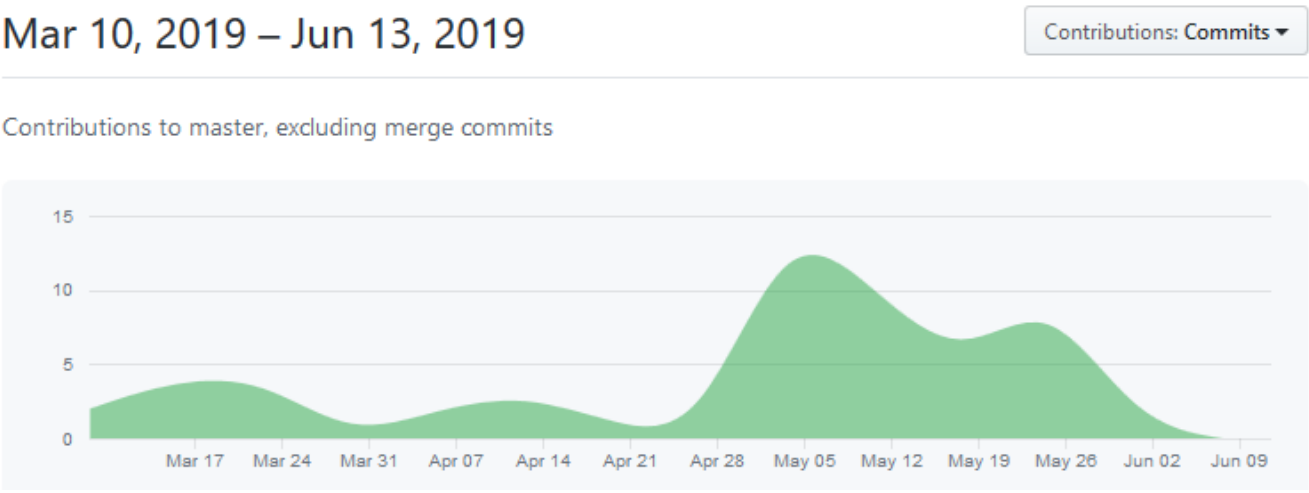
- **dzieci**, które chcą rysować i jednocześnie pozostać w ruchu,
- **studentów**, którzy chcą odstresować się po sesji,
- **emerytów**, którzy chcą grać w kalambury podczas coniedzielnych spotkań.

2 Motywacja

Temat *Malowanie obrazów biorąc pod uwagę ruch ciała* został wybrany przez nas, ponieważ w przeszłości mieliśmy już do czynienia z przetwarzaniem obrazów i mogliśmy wykorzystać wcześniej zdobyte doświadczenia w nowym projekcie. Wychodzimy z założenia, że powinno się zajmować rzeczami, których tematyka nas interesuje, wtedy chętniej i częściej będziemy podchodzić do danej czynności. Temat ten wydał nam się jednym z najciekawszych, tj. przypadł każdemu z członków zespołu do gustu najbardziej z listy dostępnych tematów. Dodatkowo zachęcił nas brak zainteresowania ze strony innych zespołów - jest to projekt unikalny w naszej grupie. Kolejną motywacją była chęć zapoznania się z API *Kinecta*, którego Cezary jest właścicielem; chcieliśmy zobaczyć jak wygląda proces połączenia i użycia funkcji tego urządzenia w programach *pythonowych*.

3 System kontroli wersji

Projekt był tworzony od 10 marca do 5 czerwca 2019 roku. W początkowej fazie projektu, powstawał on raczej niezależnie od platformy GitHub z racji wykorzystania rozwiązania Google Collaboratory. Aktywność w repozytorium *githubowym* osiągnęła maksimum w maju, kiedy udało nam się połączyć wszystkie moduły i w końcu każdy z członków był uzależniony od częstotliwości aktualizacji modułów partnerów.
Link do repozytorium: https://github.com/szkudlarekdamian/Kinart/



Rysunek 1: Aktywność w repozytorium GitHub.

4 Wymagania techniczne

4.1 Wymagania sprzętowe

Do uruchomienia programu niezbędny jest komputer ze skonfigurowanym środowiskiem oraz urządzenie *Kinect Sensor for XBOX360*.
Parametry komputera, na którym uruchamiany był program, które są jednocześnie rekomendowanymi przez nas wymaganiami:

- CPU - Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
- GPU - NVIDIA GeForce 940M
- Pamięć RAM - 8GB

Jako kamerę wykorzystaliśmy urządzenie *Kinect Sensor* przeznaczone do konsoli XBOX360 w wersji oznaczonej numerem 1473. Do uruchomienia programu niezbędny jest także dołączany w zestawie z Kinectem kabel łączący urządzenie z komputerem oraz zasilaniem.

4.2 Wymagane oprogramowanie

Do uruchomienia programu koniecznie jest zainstalowanie sterownika libfreenect wraz z interfejsem do języka python. Pozostałe wymagania (wraz z wersjami):

- python (3.6.7)
- numpy (1.16.2)
- opencv-contrib (4.1.0.25)
- tkinter (8.6)
- PIL (1.1.7)

5 Podział prac

5.1 Małgorzata Hanyż

Do zadań Pani Małgorzaty należało:

- znalezienie biblioteki języka python umożliwiającej stworzenie odpowiedniego dla naszej aplikacji interfejsu użytkownika,
- stworzenie programu typu Paint do rysowania obrazów za pomocą myszki,
- dostosowanie programu do rysowania na podstawie współrzędnych,
- dodawanie kolejnych funkcjonalności takich jak: zapis obrazów, zmiana koloru pędzla, możliwość wymazywania za pomocą gumki czy przerywanie rysowania w celu wyboru opcji z interfejsu.

5.2 Damian Szkudlarek

Do zadań Pana Damiana należało:

- przetworzenie obrazu (mapy głębi) z sensorów *Kinect* w celu odfiltrowania nieistotnych elementów otoczenia aktora,
- zaprojektowanie algorytmu, którego zadaniem jest śledzenie dłoni i zwracanie jej współrzędnych,
- zbudowanie prostego interfejsu, który służy do inicjalizowania początkowego położenia dłoni,

- zaprojektowanie nieskomplikowanego systemu, który cyklicznie sprawdza obecność dłoni w obszarze inicjalizacji - po to, aby rozpocząć działanie modułu śledzącego,
- zaprogramowanie funkcji rozpoznającej gest otwartej dłoni lub zaciśniętej pięści.

5.3 Cezary Waligóra

Do zadań Pana Cezarego należało:

- wybór odpowiedniej technologii do zaczytywania obrazu z sensora Kinect,
- przygotowanie środowiska ze wszystkimi bibliotekami i sterownikami,
- połączenie ze sobą wszystkich modułów (GUI, pobierania danych z sensora i przetwarzania obrazu),
- testowanie nowych funkcjonalności,
- wykorzystanie informacji o rozpoznanym geście do przerywania i wznawiania rysowania.

6 Funkcjonalności Kinart

Aplikacja umożliwia tworzenie własnych rysunków. Wzorowaliśmy się na programie Paint. Do głównych funkcjonalności należą:

- kalibracja w celu dokładnego śledzenia dłoni - odbywa się poprzez umieszczenie dłoni na 5 sekund w zaznaczonym na jednym z ekranów obszarze,
- tworzenie linii imitujących pociągnięcia pędzla na podstawie aktualnego położenia dłoni - jeśli dłoń znajduje się w pozycji otwartej, to pozycja dłoni wyznaczana z każdej kolejnej klatki przedłuża rysowaną linię,
- wymazywanie fragmentów obrazu za pomocą gumki - pozwala na pozbycie się zbędnych części rysunku,
- zmiana koloru farby - do wyboru czarny, niebieski, czerwony oraz zielony, wyboru dokonuje się przez najechanie pięścią na prostokąt z odpowiednim kolorem,

- czyszczenie całego płótna - powoduje usunięcie wszystkich naniesionych odcinków i pozostawienie tylko białego tła,
- zapisywanie utworzonych obrazów do plików graficznych.

7 Wybrane technologie

7.1 Python

Do implementacji programu wykorzystaliśmy język **python**. Kierowaliśmy się przede wszystkim faktem, że wszyscy go znaliśmy i używaliśmy na innych zajęciach lub w prywatnych projektach. Python jest wspierany przez wiele bibliotek, które rozszerzają jego funkcjonalności. Bardzo popularną w przypadku różnego rodzaju operacji matematycznych jest biblioteka *NumPy*, której też użyliśmy w pewnym stopniu do działań matematycznych na składowych obrazu.

7.2 OpenCV

Biblioteka *OpenCV* została przez nas wykorzystana do analizowana klatek pobieranych z urządzenia *Kinect*. Wybraliśmy tę bibliotekę, ponieważ jest to jeden z najpopularniejszych zbiorów funkcji z zakresu przetwarzania obrazu. Była ona wykorzystywana przez nas wcześniej na zajęciach z przedmiotu Przetwarzanie obrazu i systemy wizyjne, a także w innych realizowanych przez nas projektach.
Z biblioteki tej można korzystać na różnych systemach operacyjnych, co znacznie ułatwiło nam pracę, ponieważ używaliśmy zarówno systemu Windows, jak i Linux.
Zadania przetwarzania obrazu jakie musieliśmy zrealizować to:

- rysowanie kształtów, np. kwadrat przestrzeni inicjalizującej śledzenie dłoni, okrąg w jego środku,
- nakładanie tekstu na obraz w celach diagnostycznych,
- usuwanie szumów przy pomocy wypełniania zalewowego i operacji morfologicznych,
- wyszukiwanie i zaznaczanie konturów,

- znalezienie wypukłości (ang. *convex*) na dłoni - w uproszczonym modelu wypukłości powinny symbolizować palce, które można zliczać, a co za tym idzie rozpoznawać gesty,
- śledzenie obiektu, a dokładniej lokalizowanie dłoni na kolejnych klatkach obrazu pobieranych z sensorów urządzenia *Kinect*. Biblioteka *OpenCV* dostarcza klasę *Tracker*, która bardzo dobrze nadawała się do realizacji tego celu.

7.3 TkInter

TkInter to biblioteka do kreowania aplikacji okienkowych. Pozwala na projektowanie graficznych intefejsów użytkownika. Do tworzenia elementów graficznych wykorzystuje tzw. *widgety*. Wybraliśmy ją dlatego, że jest bardzo prosta w obsłudze, przejrzysta i przede wszystkim wystarczająca dla potrzeb projektu. Dodatkowo jest to jedna z najbardziej popularnych bibliotek *pythonowych* do tworzenia interfejsów, dzięki czemu jest wiele materiałów, które pomogły nam w zapoznaniu się z biblioteką i rozwiązywaniu błędów.
Każdy *widget* można dowolnie rekonfigurować w czasie działania programu. Można zmieniać wartości parametrów, czy też rozmiar i położenie obiektu. Jest to duża zaleta przy tworzeniu animacji. *Widgety* tworzą strukturę hierarchiczną, konieczne jest aby przy ich tworzeniu podać nazwę obiektu-rodzica, którego rolę pełni okno, w którym ma się znajdować dany *widget*. Biblioteka *TkInter* umożliwia także powiązanie zdarzeń z *widgetami*. Realizuje się to za pomocą metody *bind*.

7.4 Kinect

Kinect to urządzenie, które jest dodatkowym komponentem do kosoli Xbox360. Pozwala ono na interakcje z konsolą bez konieczności trzymania w dłoniach żadnych czujników ani kontrolerów. Ponieważ rozwiązanie to jest wykorzystywane w grach wideo, musi się cechować wysoką jakością zwracanych informacji, aby zapewnić odpowiedni poziom rozgrywki.
Metoda tworzenia mapy głębi, z której korzysta Kinect, nosi nazwę światła strukturalnego. W urządzeniu znajdują się 2 specjalnie przeznaczone do tego celu podzespoły: emiter podczerwieni oraz kamera podczerwieni. Pierwszy z nich wyświetla zbiór punktów, które są nanoszone na wszystko, co znajduje się przed *Kinectem*. Ponieważ naświetlenie odbywa się w podczerwieni, nie jest zauważalne dla człowieka. Drugi z komponentów analizuje ułożenie punktów na zarejestro-

wanym obrazie i na tej podstawie określa odległość elementów obrazu od sensora. Wybierając to urządzenie kierowaliśmy się przede wszystkim tym, że pozwala ono na uzyskanie mapy głębi otoczenia będącego w zasięgu sensora. Fakt ten był dla nas istotny z trzech powodów:

1. uniezależnienie od warunków - podstawowym problemem klasycznych metod przetwarzania jest ich słabe uogólnienie. Stosunkowo łatwo jest stworzyć program działający w określonych warunkach przy konkretnym oświetleniu. Niestety nawet niewielka zmiana otoczenia może uniemożliwić poprawne przetwarzanie obrazu. *Kinect* jest przystosowany do pracy w zmiennych warunkach, a zwracana mapa głebi ma zbliżoną skuteczność, nawet przy skrajnej zmianie warunków,

2. uwzględnienie odległości - aby praca z programem była intuicyjna, obraz powinien powstawać przez poruszanie wyciągniętą ręką, co imituje nanoszenie farby za pomocą pędzla na płótnie znajdującym się przed użytkownikiem. Pomiar odległości jest dużo łatwiejszy przy wykorzystaniu mapy głębi, gdzie każdy kolor punkt obrazu reprezentuje jego odległość od urządzenia,

3. filtrowanie obrazu - położenie w przestrzeni jest przedstawiane w skali szarości, co umożliwia łatwe odfiltrowanie w całym obrazie elementów znajdujących się w określonej odległości i usunięcie z niego tych fragmentów, które są poza określonym przedziałem wartości.

7.5 Libfreenect

Libfreenect to sterownik o otwartym kodzie źródłowym, który jest częścią projektu OpenKinect. Chociaż jest on napisany w języku C++, to udostępnia interfejsy (ang. *wrappers*) w innych językach, między innymi w języku python. Pozwala on na wykorzystanie funkcji umożliwiających odczyt obrazu z sensora Kinect. Wybraliśmy go, ponieważ zapewniał wymagane przez nas funkcjonalności, a do tego był stosunkowo dobrze udokumentowany, co wynika z jego względnie dużej popularności wśród tego typu sterowników.

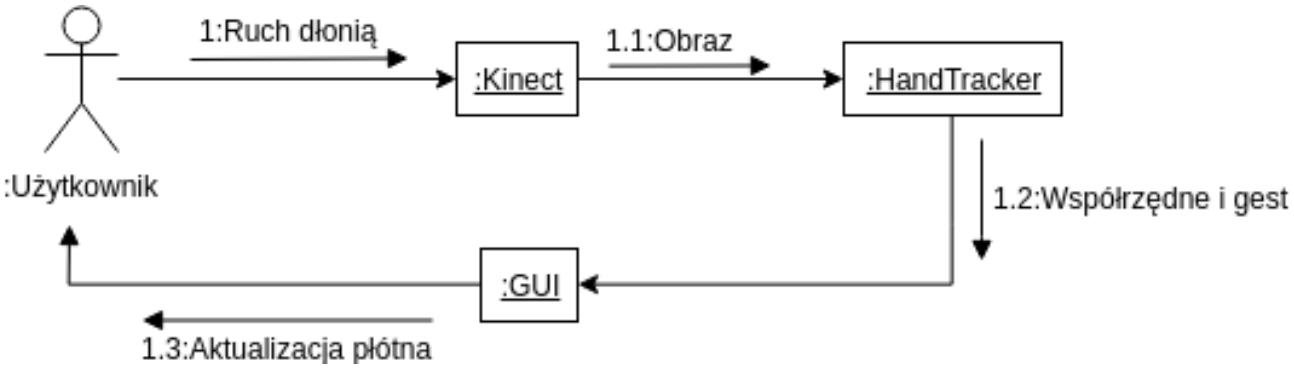
8 Architektura

8.1 Oprogramowanie

Program składa się z 4 głównych modułów:

- kinect_video_recorder.py - udostępnia funkcję do pobierania kolejnych klatek z *Kinecta*,
- trackHand.py - udostępnia klasę służącą do przetwarzania kolejnych klatek obrazu, inicjalizowania dłoni i jej śledzenia,
- GUI.py - odpowiada za interfejs graficzny. Posiada funkcje umożliwiające rysowanie na płótnie oraz obsługę interfejsu na podstawie współrzędnych zwracanych przez *tracker*,
- mainArt.py - łączy wszystkie moduły oraz wywołuje główną pętlę programu.

8.2 Przepływ informacji



Rysunek 2: Diagram pokazuje, jak program po kolei przetwarza informacje. Ruch użytkownika jest przekazywany do modułu przetwarzania obrazu, który zwraca współrzędne dłoni oraz rozpoznany gest. Na podstawie tych informacji następuje aktualizacja obrazu płótna, które jest wyświetlane użytkownikowi

8.3 Moduł połączenia z Kinectem

Udostępnia funkcje do pobierania danych z sensorów Kinecta.

get_depth()

Opis:

Podstawowa funkcja, która pobiera mapę głębi z Kinecta. Wykorzystuje metodę z zaimportowanego modułu freenect, stanowiącym interfejs sterownika libfreenect dla języka python.

Parametry:

Brak.

Wartość zwracana:

Obraz z mapy głębi reprezentowany przez dwuwymiarową tablicę numpy ośmiobitowych liczb całkowitych.

get_video()

Opis:

Funkcja zwracająca obraz z sensora RGB znajdującego się w urządzeniu. Była przez nas wykorzystywana do celów testowych, aby móc łatwo porównać mapę głębi z kolorowym widokiem liczb całkowitych

Parametry:

Brak.

Wartość zwracana:

Obraz z kamery reprezentowany przez dwuwymiarową tablicę numpy ośmiobitowych liczb całkowitych.

8.4 Moduł przetwarzania obrazu

8.4.1 Informacje wstępne

Podstawą modułu przetwarzania obrazu jest klasa ***HandTracker***.
Konstruktor tej klasy przyjmuje jeden argument - źródło obrazu.
Źródłem obrazu może być strumień wideo z *Kinecta* lub alternatywnie plik wideo o dowolnym rozszerzeniu (będący zapisem wcześniej wspomnianego strumienia wideo).

8.4.2 Atrybuty

Atrybutami klasy *HandTracker* są:

- *cap* - obiekt zwracany przez funkcję cv2.VideoCapture(); pozwala na odczytywanie kolejnych klatek ze źródła obrazu,
- *height* - wysokość klatki,
- *width* - szerokość klatki,
- *handInitialized* - flaga informująca o tym, czy dłoń została już wykryta,
- *tracker* - instancja *trackera* typu CSRT z biblioteki OpenCV,
- *thread* - wątek, który służy do cyklicznego sprawdzania obecności dłoni w obszarze inicjalizacji,
- *stopFlag* - obiekt typu Event, który zatrzymuje lub uruchamia licznik czasu z powyższego wątku,
- *x1* - współrzędna x lewego górnego rogu obszaru inicjalizacji,
- *x2* - współrzędna x prawego dolnego rogu obszaru inicjalizacji,
- *y1* - współrzędna y lewego górnego rogu obszaru inicjalizacji,
- *y2* - współrzędna y prawego dolnego rogu obszaru inicjalizacji,
- *centerPoint* - krotka przechowująca współrzędne x i y punktu centralnego (domyślnie środek obszaru inicjalizacji).

8.4.3 Metody

---

kinectOpened()

Opis:

Służy do sprawdzenia czy źródło obrazu zostało poprawnie otwarte lub czy wciąż jest otwarte.

Parametry:

Brak.

Wartość zwracana:

Wartość zero-jedynkowa mówiącą czy źródło obrazu zostało poprawnie otwarte lub czy wciąż jest otwarte.

Uwagi:

Źródło obrazu może nie zostać poprawnie otwarte w przypadku błędnego parametru wejściowego w konstruktorze klasy lub nierozpoznanych lokalnie kodeków.
Źródło obrazu jest otwarte dopóki dostarcza klatki. Przykładowo zamykane jest w przypadku końca filmu.

---

getNextFrame()

Opis:

Służy do pobrania kolejnej klatki z *Kinecta* lub pliku wideo.

Parametry:

Brak.

Wartość zwracana:

Referencja do kolejnej klatki lub wartość *None* w przypadku, gdy kolejna klatka nie istnieje.

getFrameWithInitBox(frame, drawHand)

Opis:

Modyfikuje kopię obrazu wejściowego poprzez narysowanie na niej kształtu reprezentującego obszar inicjalizacji dłoni.

Parametry:

frame - obraz wejściowy, na który chcemy nanieść kszałty,

drawHand - parametr *boolowski*, którego prawdziwość spowoduje narysowanie w obszarze inicjalizacji dłoni, pomocniczego obrysu dłoni.

Wartość zwracana:

Obraz będący kopią obrazu wejściowego *frame* z narysowanymi kształtami prostokąta - zgodnie z atrybutami *x1,x2,y1,y2* oraz koła - na podstawie atrybutu *centerPoint*.

filterDepth(frame, closestDistance, furthestDistance)

Opis:

Prosty filtr szumów - wygasza z obrazu *frame* wszystkie piksele znajdujące się poza zakresem zdefniowanym przez parametry *closestDistance* i *furhestDistance*.

Parametry:

frame - obraz wejściowy, w odcieniach szarości, który chcemy poddać filtracji,

closestDistance - minimalna jasność piksela, aby nie został wygaszony,

furthestDistance - maksymalna jasność piksela, aby nie został wygaszony.

Wartość zwracana:

Odfiltrowany obraz w odcieniach szarości.

getNeighbourhoodROI(frame, centerPoint, side)

Opis:

Wycina obszar obrazu (*Rectangle of interest*) w postaci kwadratu o boku *side* pikseli oraz środku w punkcie *centerPoint*.

Parametry:

frame - obraz wejściowy, z którego ROI chcemy wyciąć,

centerPoint - krotka reprezentująca współrzędne x i y punktu środka ROI,

side - długość boku kwadratu - ROI - podana w pikselach.

Wartość zwracana:

Obraz, będący wycinkiem wyprodukowanym na podstawie parametrów metody.

getValueOfCenter(frame)

Opis:

Pobiera aktualną średnią wartość pikseli w określonym otoczeniu punktu na obrazie, wskazywanym przez atrybut *centerPoint*.

Parametry:

frame - obraz wejściowy, z którego wartość chcemy odczytać.

Wartość zwracana:

Liczba całkowita dodatnia typu *uint8*, będąca medianą wartości pikseli otoczenia punktu.

Uwagi:

Otoczenie punktu jest wycinkiem na obrazie - użyto metody *getNeighbourhoodROI(frame, centerPoint, side)*, gdzie obrazem wejściowym jest ten sam obraz, punktem centralnym jest atrybut *centerPoint*, a bok jest ustalony przez użytkownika w pliku konfiguracyjnym - oznaczony jako *MEDIAN_SQUARE_SIDE* i domyślnie wynosi 30 pikseli.

findHandContour(frame, drawContour)

Opis:

Wyszukuje na obrazie największy kontur (może go narysować) oraz prostokąt go obejmujący (ang. *bounding box*).

Parametry:

frame - obraz wejściowy, na którym chcemy odszukać kontur,

drawContour - parametr *boolowski*, który decyduje o tym, czy na obrazie ma zostać narysowany znaleziony kontur. Domyślnie wartością tego parametru jest fałsz.

Wartość zwracana:

Jeśli wartością parametru *drawHand* jest prawda, wówczas zwracane są:

1. kopia obrazu wejściowego z uwydatnionym konturem,
2. największy znaleziony kontur, zwrócony przez *cv2.findContours()*,
3. prostokąt obejmujący wyżej opisany kontur, wyznaczony przez *cv2.boundingRect()*.

Jeśli wartością parametru *drawHand* jest fałsz, wówczas zwracane są te same wartości z wykluczeniem pierwszego - obrazu.

filterFrame(frame, center, condition, kernelSize, iterations)

Opis:

Filtr obrazu, który wygasza wszystkie piksele x, które nie należą do określonego parametrami przedziału. Spełnia zależność:

$$\mu = median(center),$$

$$\forall x \notin (median - condition; median + condition), x = 0$$

Na tak przefiltrowanym obrazie dodatkowo przeprowadzane jest morfologiczne domknięcie.

Parametry:

frame - obraz wejściowy, który chcemy poddać filtracji,

center - wycinek obrazu, z którego medianę należy policzyć,

condition - liczba całkowita, która jest granicą błędu wartości piksela, braną pod uwagę przy wygaszeniu,

kernelSize - rozmiar *kernela* podawany na wejście morfologicznego domknięcia *cv2.morphologyEx()*,

iterations - liczba iteracji domknięcia podawana na wejście funkcji *cv2.morphologyEx()*.

Wartość zwracana:

Obraz oczyszczony z szumów filtrem i morfologicznym domknięciem.

Uwagi:

Wycinek obrazu *center* może być uzyskany z użyciem metody *getNeighbourhoodROI(frame, centerPoint, side)*

---

initTracker(frame)

Opis:

Odpowiada za rozpoczęcie śledzenia ruchu dłoni, między kolejnymi klatkami. Wycina stały obszar (część środka dłoni) ze środka obszaru inicjalizacji dłoni, po czym filtruje obraz na podstawie tego wycinka. Filtrowany obraz jest wypełniany zalewowo w punkcie wskazywanym przez atrybut *centerPoint*. Na obrazie wynikowym wyszukiwany jest największy kontur - kontur dłoni, który wchodzi na wejście *trackera* CSRT.

Parametry:

frame - obraz wejściowy, z dłonią w środku obszaru inicjalizacji.

Wartość zwracana:

Brak.

Uwagi:

Użyte zostały wyżej wymienione metody: *getNeighbourhoodROI()*,*filterFrame()*, *findHandContour()*, a także *cv2.floodFill()*.

---

gestureRecognition(frame, p1, p2, offset)

Opis:

Służy do rozpoznawania gestu dłoni na wycinku obrazu. Algorytm rozpoznaje w prosty sposób liczbę palców, opierając się na defektach wypukłości zwracanych przez funkcję *cv2.convexityDefects()* oraz zależnościach kątowych między nimi występującymi.

Parametry:

frame - obraz wejściowy,

p1 - krotka reprezentująca współrzędne lewego górnego rogu obszaru z dłonią,

p2 - krotka reprezentująca współrzędne prawego dolnego rogu obszaru z dłonią,

offset - dodatkowy margines dodawany do obszaru definiowanego przez p1 i p2.

Wartość zwracana:

Kontur z dłonią oraz rodzaj gestu.

Pięść oznaczona jest jako 0, a otwarta dłoń jako 1.

Uwagi:

Ze względu na często rozmyte kontury dłoni oraz dużą odległość od kamery, udaje się rozróżniać jedynie dwa najbardziej różne od siebie gesty - przy czym złączone palce mogą być rozpoznane jako pięść.

---

trackHand(frame)

Opis:

Realizuje aktualizację *trackera* filtrowanym odległościowo (*filterDepth()*) obrazem. Rysuje na ob-

razie *bounding box* dłoni zwrócony przez *tracker* oraz w postaci tekstu współrzędne środka dłoni i wynik klasyfikacji wykonanego gestu.

Parametry:

frame - obraz wejściowy.

Wartość zwracana:

Kopia obrazu z narysowanymi szczegółami, współrzędne środka dłoni oraz wynik klasyfikacji wykonanego gestu.

8.4.4 Czasowa inicjalizacja dłoni

Moduł wzbogacony został również o wątek (jego instancja jest jednym z atrybutów klasy *HandTracker*. Wątek opiera swoje działanie na podstawie:

- dwóch wartości liczbowych w pliku konfiguracyjnym - odpowiadającym za częstość sprawdzania obecności dłoni oraz liczby koniecznych sprawdzeń z pozytywnym rezultatem; domyślnie wątek sprawdza co sekundę czy dłoń znajduje się w obszarze, a po pięciosekundowej obecności dłoni w obszarze inicjalizacji, zostaje wywołana metoda initTracker(),
- **flagi** *stopped* - atrybut *stopFlag* z klasy *HandTracker*, który zatrzymuje wątek po zainicjalizowaniu dłoni lub uruchamia go w razie potrzeby,
- **licznika** *found* - który zlicza ile razy w określonym czasie, dłoń znajdowała się w obszarze inicjalizacji,
- **flagi** *isRunning* - która informuje o tym, czy wątek akutalnie sprawdza obecność występowania dłoni w obszarze inicjalizacji.

8.5 Moduł graficznego interfejsu

8.5.1 Informacje wstępne

Moduł graficznego interfejsu zawiera klasę *Kinart* która kreuje interfejs graficzny użytkownika i implementuje metody potrzebne do tworzenia obrazu na płótnie. Konstruktor klasy nie przyjmuje

atrybutów. Używa ona biblioteki *TkInter* do tworzenia okna interfejsu graficznego oraz dodatkowo biblioteki *Pillow* (w skrócie *PIL*) w celu możliwości zapisania utworzonego obrazu do pliku typu *png*.

8.5.2 Atrybuty

Atrybutami klasy *Kinart* są:

- *active_button* - obiekt aktualnie wybranego przycisku interfejsu,
- *color* - aktualny kolor rysowania,
- *eraser_on* - zmienna mówiąca o tym czy jest wybrana opcja wymazywania obrazu,
- *line_width* - rozmiar pędzla,
- *old_x* - ostatnia otrzymana współrzędna szerokości,
- *old_y* - ostatnia otrzymana współrzędna wysokości,
- *old_dot* - ostatni stworzony obiekt kropki podążającej za dłonią użytkownika w momencie przerwania rysowania,
- *root* - okno interfejsu graficznego,
- *reset_button*, *eraser_button*, *green_button*, *red_button*, *blue_button*, *black_button*, *save_button* - obiekty przycisków interfejsu klasy *Button*,
- *painting* - obiekt płótna typu *Canvas* interfejsu,
- *image* - obiekt płótna typu *PIL.Image* klasy *Pillow* służący do zapisania stworzonego obrazu,
- *draw* - obiekt płótna typu *ImageDraw* klasy *Pillow* służący do zapisania stworzonego obrazu.

8.5.3 Metody

---

setup()

Opis:

Metoda przypisująca wartości początkowe zmiennym klasy. Jest wywoływana przy starcie aplikacji.

Parametry:

Brak.

Wartość zwracana:

Brak.

---

updateCoords(x, y)

Opis:

Metoda odpowiedzialna za rysowanie oraz wymazywanie obrazu na płótnie. Jeśli nie jest aktywny tryb wymazywania tworzy ona linię prostą od początku poprzednich współrzędnych do tych podanych jako parametry, korzystając ze zmiennych, np.: *line_width* i *color*. Następnie nowe współrzędne stają się tymi starymi. Jeśli nie istnieją poprzednie współrzędne tworzenie linii jest pomijane. W trybie wymazywania metoda działa w dokładnie taki sam sposób, z tym że do rysowania używa koloru białego. Jest ona wywoływana w momencie gdy użytkownik ma otwartą dłoń.

Parametry:

x - współrzędna szerokości zwracana przez *tracker*,

y - współrzędna wysokości zwracana przez *tracker*.

Wartość zwracana:

Brak.

Uwagi:

Parametry muszą mieć wartości całkowite.

---

resetCoords()

Opis:

Metoda przerywająca rysowanie bądź wymazywanie. Wywoływana w momencie zmiany gestu użytkownika na zamkniętą dłoń.

Parametry:

Brak.

Wartość zwracana:

Brak.

---

createDot(x, y)

Opis:

Metoda służąca do tworzenia kropki podążającej za ręką rysującego. Tworzy ona okrągły wskaźnik w miejscu podanych w parametrach współrzędnych jednocześnie usuwając poprzedni, o ile istnieje. Nowo stworzona kropka staje się starą. Metoda wywoływana gdy użytkownik ma zamkniętą dłoń.

Parametry:

x - współrzędna szerokości zwracana przez *tracker*,

y - współrzędna wysokości zwracana przez *tracker*.

Wartość zwracana:

Brak.

Uwagi:

Parametry muszą mieć wartości całkowite.

---

resetDot()

Opis:

Metoda usuwająca wskaźnik kropki podążający za ręką. Wywoływana jest w momencie gdy użytkownik otworzy dłoń w celu rysowania.

Parametry:

Brak.

Wartość zwracana:

Brak.

resetCanvas()

Opis:

Metoda resetuje namalowane linie na płótnie oraz zeruje obiekty służące do zapisywania obrazu. Wywoływana jest w momencie użycia przycisku *RESET*. Z pomocą funkcji *activate_button()* zmienia odpowiedni przycisk.

Parametry:

Brak.

Wartość zwracana:

Brak.

use_eraser()

Opis:

Metoda umożliwia zmianę trybu na wymazywanie obrazu, zmieniając wartość zmiennej *eraser_on* na *True*. W tym celu wywołuje funkcję *activate_button()*, która uaktualnia odpowiedni przycisk

interfejsu.

Parametry:

Brak.

Wartość zwracana:

Brak.

---

green_color(), red_color(), blue_color(), black_color()

Opis:

Metody odpowiadają za zmianę aktywnego koloru rysowania. W tym celu przypisują do zmiennej *color* odpowiadający swojej nazwie kolor oraz wywołują funkcję *activate_button()* zmieniającą aktywny przycisk w interfejsie.

Parametry:

Brak.

Wartość zwracana:

Brak.

---

activate_button(button, eraser_mode=False),

Opis:

Metoda odpowiada za zmianę aktywnego przycisku i jego animację w interfejsie. Przypisuje aktualny przycisk wykorzystując przekazany parametr *button* oraz uaktualnia stan zmiennej *eraser_on* przypisując jej parametr *eraser_mode*.

Parametry:

button - obiekt *Button*, który ma być aktywny,

eraser_mode - informacja czy ma zostać uaktywniony tryb wymazywania. Parametr nie jest wymagany przy wywoływaniu metody. Przyjmuje on domyślną wartość *False*.

Wartość zwracana:

Brak.

---

save()

Opis:

Metoda umożliwia zapisanie stworzonego dzieła do pliku typu *png*. Wykorzystuje do tego funkcję *save()* biblioteki *Pillow*. Używa funkcji *activate_button()* do zmiany odpowiedniego przycisku interfejsu.

Parametry:

Brak.

Wartość zwracana:

Brak.

8.6 Main

W tym module zdefiniowana jest klasa Gesture oraz główna pętla programu wraz z wywoływanymi w niej funkcjami.

8.6.1 Gesture

Klasa ta odpowiada za analizę ostatnich gestów, w celu wykrycia pojedynczych błędów. Jej atrybutami są:
currentGesture - zapisujące numer obecnego gestu (0 - zaciśnięta pięść, 1 - otwarta dłoń),
gestureHistory - lista o długości x podanej w konstruktorze, w której zapisywane są oznaczenia ostatnich odczytanych gestów.

Udostępnia ona 2 metody, getGesture zwracającą obecny gest, oraz metodę checkGesture opisaną poniżej.

---

checkGesture(newGesture)

Opis:

Metoda najpierw dodaje na koniec listy gestureHistory nowy gest jednocześnie usuwając jej pierwszy element, a następnie sprawdza, czy lista zawiera tylko te same gesty (tylko wartości 1 lub albo tylko wartości 0). Jeśli warunek jest spełniony, to pierwszy element z listy (wszystkie są takie same) staje się obecnym gestem zapisanym w zmiennej currentGesture.

Parametry:

newGesture - oznaczenie nowego pobranego gestu (0 - zaciśnięta pięść, 1 - otwarta dłoń).

Wartość zwracana:

Brak.

Uwagi:

W przypadku wykrycia, że na liście gestureHistory znajdują się więcej niż dwie różne wartości funkcja wyrzuca wyjątek.

8.6.2 Główna pętla programu i funkcje w niej wywoływane

Po uruchomieniu głównej pętli programu odczytywane są kolejne klatki mapy głębi pobranej z Kinecta, a następnie, po krótkiej kalibracji, inicjalizowany jest moduł śledzenia dłoni. Od tego momentu na każdej kolejnej klatce lokalizowana jest dłoń oraz rozpoznawany jest jej gest (otwarta dłoń lub zaciśnięta pięść). Następnie na podstawie tych informacji wywoływane są odpowiednie metody z modułu GUI.

Do funkcji zdefiniowanych w tym module należą:

---

initializeHandWithFrame(hT, frame)

Opis:

Funkcja wywołuje odpowiednie metody klasy handTracker inicjujące śledzenie dłoni

Parametry:

hT - instancja klasy handTracker

frame - obraz mapy głębi jako tablica numpy

Wartość zwracana:

Brak.

Uwagi:

Funkcja dodatkowo umieszcza na obrazie podanym jako parametr frame informację o obecnej odległości dłoni od urządzenia.

---

writeDistanceInfoOnFrame(frame, text)

Opis:

Funkcja nanosi na obraz podany jako parametr frame napis podany jako parametr text

Parametry:

frame - obraz mapy głębi jako tablica numpy

text - informacja, która zostanie naniesiona na obraz

Wartość zwracana:

Brak.

---

checkCoordsCorrectness(coords)

Opis:

Funkcja sprawdza, czy pobrane koordynaty nie wychodzą poza maksymalne wymiary obrazu

Parametry:

coords - koordynaty jako para liczb

Wartość zwracana:

Wartość logiczna True lub False oznaczająca, czy współrzędne są poprawne

---

useInterfaceButton(paint, coords)

Opis:

Funkcja aktywuje odpowiedni element interfejsu użytkownika na podstawie podanych współrzędnych

Parametry:

paint - instancja klasy Kinart z modułu GUI coords - koordynaty jako para liczb

Wartość zwracana:

Brak.

---

paintAndinteract(paint, coords, gest)

Opis:

Funkcja sprawdza poprawność współrzędnych, a następnie na podstawie gestu wybiera, czy aktywować któryś z przycisków, nanieść nową linię czy przesunąć wskaźnik położenia dłoni.

Parametry:

paint - instancja klasy Kinart z modułu GUI

coords - koordynaty jako para liczb

gest - wartość gestu rozpoznanego na obrazie (0 lub 1)

Wartość zwracana:

Brak.

---

rescale_coords(coords, scale=2)

Opis:

Funkcja zmienia wartość współrzędnych według podanej skali. Dodatkowo punkt (0,0) jest przesuwany tak, aby środek obrazu przed i po przeskalowaniu przypadał w tym samym miejscu.

Parametry:

coords - koordynaty jako para liczb

scale - skala, według której zmienione zostaną współrzędne, domyślnie 2 co oznacza zmniejszenie każdego z wymiarów dwukrotnie

Wartość zwracana:

przeskalowane wartości jako krotka liczb.

9 Napotkane problemy

9.1 Rysowanie na podstawie kolejnych zwróconych punktów

Jednym z problemów, które należało rozwiązać, było tworzenie rysunków na podstawie kolejnych punktów, które były wyznaczane na podstawie położenia dłoni. W internecie można znaleźć rozwiązania polegające na nanoszeniu na obraz kolejnych współrzędnych. Sprawia to jednak, że rysunki mają postać zbioru punktów, a nie ciągłych linii imitujących pociągnięcia pędzla. Rozwiązanie zaproponowane przez Małgorzatę polegało na tworzeniu kolejnych odcinków linii przez łączenie nowego punktu z poprzednim, jeśli jest on zapisany w pamięci. W przeciwnym wypadku nowy punkt stawał się punktem początkowym kolejnej linii.

9.2 Automatyczne zawieszanie sensora

Problem ten był związany z wersją Kinecta. Pomimo tego, ze odczytywane były z niego kolejne klatki, to niedługo po uruchomieniu sensor zawieszał się. Było to komunikowane poprzez wyświetlenie informacji:

"*USB device disappeared, cancelling stream 81 :(USB camera marked dead, stopping streams*"

Odpowiedzialny był za to specjalny fragment jądra systemu, który w ramach oszczędzania energii zawiesza te urządzenia podłączone przez USB, które uznaje za bezczynne[1]. W tym przypadku była to błędna decyzja, ponieważ z sensora stale były pobierane kolejne klatki obrazu. Rozwiązanie polega na wpisaniu (jako użytkownik root) odpowiedniej wartości w pliku /sys/module/usbcore/parameters/autosuspend. Plik ten zawiera tylko jedną liczbę całkowitą, która określa, po ilu sekundach nieaktywności urządzenie zostanie zawieszone. Domyślnie wynosi ona 2. Aby zablokować automatyczne zawieszanie (co było porządane w naszym przypadku), należało w tym pliku umieścić liczbę -1. Można to zrobić komendą:

"echo -1 > /sys/module/usbcore/parameters/autosuspend".

9.3 Śledzenie dłoni

Problematyczne początkowo było ustalenie sposobu śledzenia dłoni. Opieramy pracę naszej aplikacji na urządzeniu *Kinect*. Obraz z kamery nie jest wystarczająco dobrej jakości, aby zastosować nauczanie maszynowe do detekcji dłoni, a mapa głębi ze względu na duży dystans dłoni od sensorów ma niestabilny kształt.

Ostatecznie wybranym rozwiązaniem było ustalenie obszaru roboczego, służącego do wskazania systemowi, gdzie dłoń się znajduje i jak wygląda. Następnie obraz był odfiltrowywany i podawany na wejście różnej klasy *trackerów*.

Aby móc wykorzystać klasę *tracker* z *OpenCV* konieczne jest podanie początkowego położenia dłoni. Położenie dłoni w kolejnych chwilach wyznaczane jest przez aktualizację *trackera* (znającego poprzednie położenie dłoni i jej cechy) nowym obrazem.

Problemem było ustalenie, który z licznych typów *trackera* nada się najlepiej do śledzenia ruchu dłoni o zmiennej szybkości. Problem udało się rozwiązać porównując wyniki śledzenia w takich samych, dostosowanych do zastosowań aplikacji, warunkach.

[1] https://www.kernel.org/doc/Documentation/usb/power-management.txt

9.4 Korzystanie z interfejsu bez rysowania

W początkowych wersjach programu niemożliwe było przerwanie rysowania. Oznaczało to, że jeśli użytkownik chciał wybrać któryś z przycisków interfejsu, aby np. zmienić kolor, to musiał namalować linię od miejsca w którym się obecnie znajduje, do danego przycisku. Aby to poprawić, wprowadziliśmy obsługę dwóch gestów - otwartej dłoni oraz zaciśniętej pięści. Jeżeli użytkownik ma otwartą dłoń, to przesuwając rękę rysuje linię. Po zaciśnięciu pięści linia nie powstaje i można najechać ręką na przycisk interfejsu lub przesunąć wskaźnik w inne miejsce, aby rozpocząć malowanie od tego punktu.

9.5 Zakłócenia przy odczytywaniu gestów

Zaprojektowany przez Damiana system rozróżniania gestów działał bardzo dobrze, jednak co pewien czas na jednej lub kilku klatkach pod rząd odczyt był niepoprawny. Powodowało to, że w trybie malowania (otwarta dłoń) linie nie były ciągłe, ale pojawiały się w nich przerwy związane z omawianym błędem. Analogicznie po przejściu w tryb sterowania (zaciśnięta pięść) pojedyncze błędne odczyty gestu powodowały powstawanie niechcianych odcinków.
Wprowadzone przez nas rozwiązanie polegało na analizie ostatnich gestów (domyślnie pięciu). Są one zapisywane w liście, która funkcjonuje jako kolejka FIFO (ang. *First In First Out*). Otwarta dłoń jest oznaczana jako liczba 1, a zaciśnięta pięść jako 0. Gest jest zmieniany tylko wtedy, gdy wszystkie gesty na liście mają tę samą wartość. Dzięki temu pojedyncze błędy nie wpływają na działanie programu.

9.6 Rysowanie kursorem myszki

Jednym z problemów z którymi spotkaliśmy się podczas tworzenia aplikacji, było przemyślenie jak ma powstać obraz na ekranie interfejsu. Początkowo padł pomysł aby wykorzystać możliwość przypisywania zdarzeń do *widgetów* aplikacji, oferowaną przez bibliotekę *TkInter*. Rozwiązanie polegało na przypisaniu wciśnięcia lewego przycisku myszy do funkcji która będzie tworzyć linie na płótnie. Wydawało się ono być dobre, lecz przechwytywanie rzeczywistego kursora komputera okazało się dosyć problematyczne, niewygodne i nie działało do końca tak jakbyśmy chcieli. Kolejnym pomysłem było stworzenie „osobistego" kursora aplikacji. Niestety nie udało się znaleźć odpowiedniej do tego biblioteki i okazało się to niemożliwe. Ostatecznie zrezygnowaliśmy z

tego typu rozwiązania i skupiliśmy się na bezpośrednim rysowaniu na płótnie z wykorzystaniem współrzędnych z *trackera*.

10 Instrukcja użytkowania aplikacji

Rysunek 3: Zrzut ekranu wykonany w trakcie korzystania z programu. W oknie po lewej stronie widzimy obecny rysunek wraz z dostępnymi opcjami. Są to (od lewej): czyszczenie całego płótna, aktywacja gumki, zmiana koloru na zielony, zmiana koloru na czerwony, zmiana koloru na niebieski, zmiana koloru na czarny oraz na końcu zapisanie obrazu do pliku. Po prawej stronie widzimy użytkownika tworzącego przedstawione dzieło. U góry tego okna możemy zobaczyć, że rozpoznany obecnie gest to otwarta dłoń (co jest zgodne z prawdą). Obok wyświetlone są znalezione współrzędne środka dłoni. Położenie dłoni oznaczone jest prostokątem.

1. Podłącz odpowiednio urządzenie *Kinect* do komputera.
2. Uruchom plik *mainArt.py*.
3. Ustaw się w odpowiedniej odległości sensora Kinect, sugerując się napisem na ekranie śledzenia dłoni.
4. Umieść rękę w czworokącie wyświetlonym na ekranie w celu inicjalizacji modułu śledzącego.
5. Poczekaj aż inicjalizacja zakończy się powodzeniem.

6. W celu rysowania poruszaj otwartą dłonią.

7. W celu przerwania rysowania zamknij dłoń.

8. Aby wybrać konkretny przycisk najedź na niego zaciśniętą dłonią, a następnie otwórz i zamknij dłoń.

9. Aby przerwać śledzenie dłoni, wystarczy zmienić jej odległość względem sensora.

10. Aby ponownie rozpocząć używanie interfejsu należy od nowa zainicjalizować dłoń postępując zgodnie z punktami 3,4 i 5.

11 Plik konfiguracyjny

Aplikacja jest wyposażona w plik konfiguracyjny **config.py**, w którym można zmieniać parametry modułu śledzącego dłoń.

11.1 Style obszarów inicjalizacji i śledzenia

1. BOUNDING_BOX_COLOR_INIT = (177, 187, 223)
   Kolor kwadratu wspomagającego inicjalizację dłoni - oznaczony jako InitBox

2. CENTER_POINT_COLOR_INIT = (118, 100, 245)
   Kolor środka kwadratu wspomagającego inicjalizację dłoni

3. BOUNDING_BOX_COLOR_TRACKING = (173, 245, 145)
   Kolor kwadratu, wspomagającego wizualizację śledzenia dłoni - oznaczony jako TrackingBox

4. CENTER_POINT_COLOR_TRACKING = (239, 237, 191)
   Kolor środka kwadratu, wspomagającego wizualizację śledzenia dłoni

5. BOUNDING_BOX_BORDER_WIDTH = 4
   Szerokość obramowania, InitBox i TrackingBox

11.2 Inicjalizacja dłoni

1. CHECKING_FOR_HAND_INTERVAL = 1
   Interwał (w sekundach), z jakim sprawdzana będzie obecność dłoni w InitBox

2. HOW_MANY_TIMES_HAND_MUST_BE_FOUND = 5

   Ile razy z rzędu musi zostać znaleziona dłoń w InitBox (powiązane z interwałem powyżej)

3. MINIMUM_VALUE_TO_CONSIDER_HAND = 14

   Minimalna wartość odległości z sensorów Kinect, aby obiekt znalazł się w InitBox (im niższa wartość, tym bliżej musimy być kamery)

4. MAXIMUM_VALUE_TO_CONSIDER_HAND = 17 Maksymalna wartość odległości z sensorów Kinect, aby obiekt znalazł się w InitBox (im wyższa wartość, tym dalej musimy być kamery)

5. MEDIAN_SQUARE_SIDE = 30 Bok kwadratowego wycinka obrazu, z którego zostaje obliczona mediana, przy inicjalizacji dłoni

11.3 Śledzenie dłoni

1. CLOSEST_DISTANCE = 12

   Minimalna wartość odległości z sensorów Kinect, która nie jest usuwana z obrazu

2. FURTHEST_DISTANCE = 31

   Maksymalna wartość odległości z sensorów Kinect, która nie jest usuwana z obrazu

11.4 Diagnostyka

1. SHOW_INIT_BOX_DURING_TRACKING = False

   Czy InitBox ma być wyświetlany w trakcie śledzenia dłoni

2. SHOW_INIT_HAND_CONTOUR = True

   Czy pomocniczy kontur dłoni ma być wyświetlany w InitBox