14.06.19

1. Wojciech Niedbała
2. Patryk Romaniak
3. Albert Millert
4. Michał Mirończuk

Projekt zespołowy - dokumentacja

https://github.com/mirooon/fingero

Malowanie obrazów biorąc pod uwagę ruch ciała

1. Opis projektu

Nasz projekt *Malowanie obrazów biorąc pod uwagę ruch ciała* polegał na stworzeniu aplikacji internetowej, bazującej na wykrywaniu, a także śledzeniu ruchu ręki użytkownika i rysowaniu przy jej pomocy linii. Efektem odwzorowania ścieżki, po której przemieszcza się dłoń jest obraz na płótnie.

Projekt udostępnia możliwość rysowania zwykłym ołówkiem lub pędzlem. Ponadto, do dyspozycji użytkownika udostępnione są narzędzia, pozwalające dostosować styl rysowania, np. paleta kolorów oraz przyciski do zmiany szerokości końcówki, przy pomocy której powstaje rysunek.

Użytkownik na każdym etapie powstawania dzieła ma również możliwość dokonywania korekty. Do tego zadania można wykorzystać białą gumkę.

W sytuacji, gdy użytkownik chce przemyśleć dalszy rozwój pracy, jednocześnie nie chce, by system śledził jego ruchy, ma do dyspozycji przycisk, umożliwiający chwilowe wstrzymanie. Jeśli rysunek nie spełnia oczekiwań autora, z powodzeniem można skorzystać z możliwości jakie oferuje przycisk resetowania płótna; odświeża widok i rozpoczyna śledzenie dłoni na nowo. Po stworzeniu obrazu można go spokojnie zapisać na komputerze, wykorzystując odpowiedni przycisk *Save*.

Zrobiliśmy również rozeznanie, czy istnieją podobne aplikacje korzystające wyłącznie z kamerek internetowych, a nie bardziej zaawansowanych urządzeń (na przykład Kinect) i znaleźliśmy dużo przykładów w Pythonie, jednak były one tworzone typowo jako aplikacje testowe, bez żadnego interfejsu, działające bezpośrednio na okienkach dostarczanych przez OpenCV. Po wypróbowaniu kilku z tych aplikacji, stwierdziliśmy, że działają one bardzo słabo i mocno obciążają komputer (prawdopodobnie twórcy przygotowując filmiki pokazujące możliwości ich programów wybierali jedynie te najlepsze próby, przez co większość artykułów i tutoriali wygląda jakby działała bardzo

dobrze, a w rzeczywistości działa to bardzo różnie. Już sam fakt, że każdy laptop może posiadać inną kamerkę internetować, o różnych parametrach powoduje zmiany w wynikach.

2. Uzasadnienie wybrania tematu

Temat zainteresował nas, ponieważ nie mieliśmy wcześniej styczności z rozpoznawaniem i śledzeniem ruchu rąk. Poprzednie projekty z jakimi mieliśmy styczność w dziedzinie przetwarzania obrazów dotyczyły głównie wykrywania twarzy, bardziej statycznych obiektów oraz zdjęć. Śledzenie ręki wydało nam się zatem odpowiednim wyzwaniem, do którego możemy wymyślić rozwiązanie na kilka sposobów. Każdy z nas miał trochę inną wizję i pomysł na aplikację i wymiana tej wiedzy była interesująca i pozwoliła wybrać najlepszy wspólny pomysł.

Z racji na fakt, że projekt jest na zajęcia z Teleinformatyki, chcieliśmy również wypróbować alternatywne sposoby na przesyłanie informacji od REST, bardziej odpowiadające aplikacjom z potrzebami aktualizacji danych i widoków na żywo, tak żeby każda przetwarzana informacja między stroną a serwerem była dostępna możliwie szybko oraz asynchronicznie.

3. Podział prac

Wojciech Niedbała:

- Przesyłanie obrazu z kamerki internetowej
- Widoki

Patryk Romaniak:

- Możliwość podglądu ścieżki (rysowanie)
- Zapisywanie utworzonego obrazu

Michał Mirończuk:

- Możliwość zmazywania rysunku
- Możliwość wyboru trybu rysowania
- Zmiana stylu, koloru, grubości markera

Albert Millert:

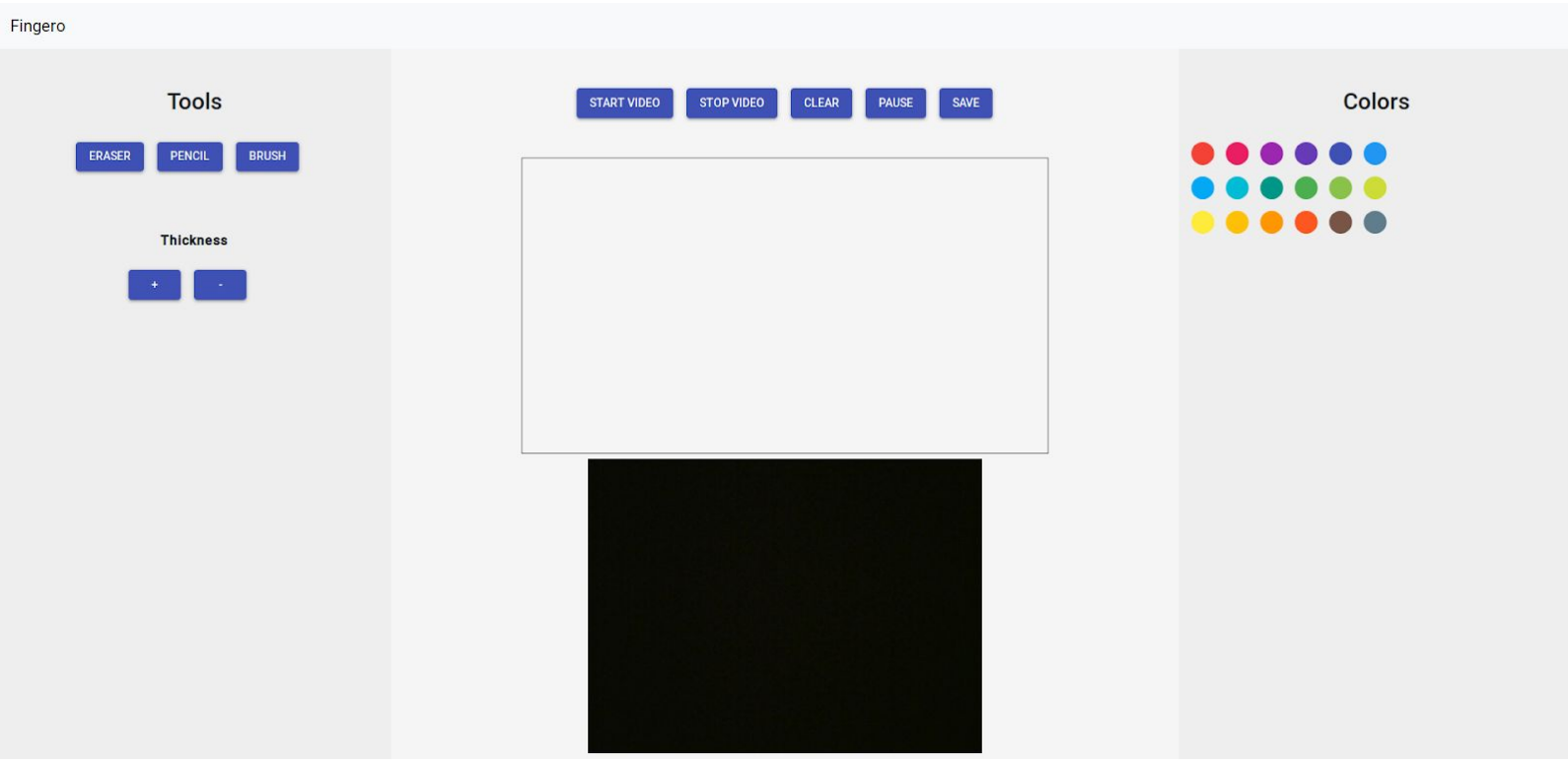
- Przetwarzanie klatek z otrzymanego obrazu
- Rozpoznawanie znacznika
- Śledzenie ruchu

4. Wylistowanie i opisanie funkcjonalności oferowanej przez aplikację

Holistyczny opis aplikacji

Strona główna aplikacji jest utrzymana w minimalistycznej stylistyce i jest wzorowana na kolorystyce zgodnej z Material Design (https://material.io/). Wstępnie próbowaliśmy również stworzyć aplikację mając na uwadze urządzenia mobilne, niestety zbyt wiele elementów różniło się; począwszy od domyślnej, pionowej orientacji telefonu, po sposób działania kamerki internetowej, która odpowiednio działała tylko na niektórych modelach, a na innych nie (co prawdopodobnie było ściśle związane z pozwoleniami udostępnionymi dla przeglądarki na dostęp do przedniej kamery telefonu).

Kwestie designu nie stanowiły dla nas kwestii priorytetowej przy tworzeniu aplikacji, dlatego nie poświęciliśmy jej nadmiernej ilości czasu. Znacznie ważniejsze dla nas było to, żeby zmiana koloru odbywała się natychmiast, nawet podczas rysowania od samego designu wyboru kolorów. Zatem naturalne jest, że w kwestii wyglądu aplikacji można zauważyć wiele niedociągnięć, szczególnie, gdy strona otwierana jest w nietypowych rozdzielczościach.



Rys 1. Widok aplikacji.

Menu

Informatywne menu przedstawia nazwę aplikacji - Fingero.

Trzy panele, stanowiące główną część aplikacji

Pierwszy panel (patrząc od lewej strony) oferuje szereg narzędzi, umożliwiających użytkownikowi wybór trybów, którymi chce wprowadzać zmiany na płótnie (panel środkowy). Do wyboru są gumka (ang. *eraser*), ołówek (ang. *pencil*), pędzel (ang. *brush*). Poniżej można zmieniać grubość pędzla.

Wcześniej wspomniany panel środkowy udostępnia główne operacje w aplikacji dostępne za pomocą przycisków:

- Start Video - odpowiada za uruchomienie wykrywania i śledzenia ręki oraz rysowania. Dodatkowo łączy się przez WebSocket z serwerem i uzyskuje od niego liczbę pokazywanych palców, dzięki czemu można w trakcie rysowania za pomocą gestów zmieniać tryb rysowania,

- Stop Video - zatrzymuje całkowicie wykrywanie, śledzenie i rysowanie oraz zamyka połączenie z serwerem. Po wykonaniu tej operacji możliwe jest jedynie uruchomienia śledzenia od nowa (poprzez Start Video) lub pobranie narysowanego obrazu,
- Clear - czyści całkowicie *canvas,*
- Pause - umożliwia chwilowe zatrzymanie śledzenia palca i rysowania, na przykład w celu przeniesienia ręki w inne miejsce.
- Save pozwala na zapisanie obrazu do pliku *.png.
  Za funkcjonalność zapisywania odpowiada komponent Save.
  Obiekt klasy dziedziczącej po komponencie jest w stanie wyrenderować widok, w tym przypadku przycisk, któremu nadajemy funkcjonalność do zapisu aktualnie rysowanego obrazka.

```
class Save extends Component{
 saveCanvas() {
   const canvasSave = document.getElementById('myCanvas');
   const d =
canvasSave.toDataURL('image/png').replace("image/png",
"image/octet-stream");
   window.location.href = d;
 }

 render(){
   return(
      <Button variant="contained" color="primary"  onClick={
this.saveCanvas }>Save</Button>
   )
 }
}
```

Tę sekcję można rozszerzać o nowe funkcjonalności, takie jak:

- ClearLast - czyszczenie ostatniej narysowanej kreski,
- Manual - przełączenie się na tryb rysowania manualnego - myszką lub gładzikiem, w celu dokonania bardziej

szczegółowych poprawek,
- Random - umożliwiające wybór losowego koloru.

- Poniżej opisanych przycisków znajduje się *canvas* (płótno), na które zostają nanoszone linie rysowane przez użytkownika. Na samym dole panelu widnieje pole, w którym po włączeniu trybu rysowania, zostaje umieszczony obraz z kamery na żywo. Docelowo te dwa obrazy miały być nałożone na siebie, jednak w zależności od warunków oświetleniowych samo płótno bywało bardzo słabo widoczne.

Ostatni panel, wysunięty najbardziej na prawo, umożliwia wybór jednego spośród całej gamy przedstawionych kolorów. Wybrany kolor zmienia się od razu, w przypadku gumki nic nie zmienia.

5. Wybrane technologie wraz z uzasadnieniem dlaczego

Back-end

- Logika serwera została zaimplementowana w języku Golang. Wybraliśmy ten język z powodu jego prostoty oraz podobieństwa do wcześniej nam znanych języków takich jak Python czy JavaScript. Ponadto Go oferuje bogatą obsługę współbieżności za pomocą wbudowanych mechanizmów takich jak goroutines czy kanałów.

- Dzięki temu, że Go nie zawiera klasy Threads, która umożliwiałaby zarządzanie wątkami, Go wyróżnia się na tle tradycyjnych języków. Aby skorzystać z goroutines wystarczy dodać słówko kluczowe go przed wywołaniem funkcji. W dowolnym momencie wykonywania programu jeden wątek wykonuje jeden goroutine; gdy zostanie zablokowany - zostanie zamieniony na inny goroutine, który następnie wykona się w tym wątku. Podejście to jest podobne do działania planisty, ale po pierwsze jest obsługiwane przez środowisko wykonawcze Go, a po drugie - jest to znacznie szybsze. Dzięki takiemu mechanizmowi

można stworzyć współbieżny serwer bez potrzeby opierania się na eventach.

- Tabela poniżej przedstawia podstawowe różnice między podejściem wątkowym a wykorzystującym goroutines:

Wątek	Goroutine
Wątki są zarządzane przez system operacyjny.	Goroutines są zarządzane przez środowisko wykonawcze go i nie mają zależności sprzętowych.
Wielkość stosu wątku wynosi około 1MB	Wielkość stosu goroutine wynosi 2KB
Wielkość stosu jest stałą wartością której nie możemy zmienić.	Wielkość stosu Go jest zarządzana w czasie wykonywania i może wynieść nawet 1 GB, co jest możliwe dzięki zarządzaniu stertą
Komunikacja nie jest prosta wymagana jest odpowiednia synchronizacja.	Wbudowany system bezpiecznej komunikacji kanały, przy ich użyciu o wiele trudniej doprowadzić do zakleszczenia.
Utworzenie wątku kosztuje dość dużo tak samo jak proces jego zamknięcia, ponieważ wątek potrzebuje sporo zasobów które musi zarezerwować bądź oddać systemowi.	Goroutines są tworzone i niszczone przez środowisko wykonawcze Go. Operacje te są bardzo tanie w porównaniu z wątkami, ponieważ środowisko wykonawcze utrzymuje już pulę wątków dla Goroutines. W tym przypadku system operacyjny nie wie o instnieniu goroutines.

- Z uwagi na fakt, że aplikacja w głównej mierze skoncentrowana jest na obróbce obrazu oraz rozpoznawaniu ręki, zdecydowaliśmy się na zrealizowanie tej funkcjonalności z wykorzystaniem biblioteki OpenCV (*Open Source Computer Vision*). Narzędzie jest bezpłatne, open-sourcowe oraz posiada bogatą społeczność ulepszającą produkt.

Za pomocą kamerki internetowej jesteśmy w stanie filtrować obraz, segmentować go oraz wykrywać poszczególne krawędzie, bez implementowania na nowo kodu rozwiązującego powtarzające się zagadnienia, a także bez użycia sztucznej inteligencji, która wymagałaby sporej ilości danych uczących (w tym przypadku zdjęć rąk).

- Aby wykorzystać bibliotekę OpenCV w stu procentach wraz z potencjałem języka Golang posłużyliśmy się paczką GOCV, która ułatwia nam dostęp do biblioteki za pośrednictwem języka stworzonego przez Google.

Front-end

- Po stronie front-endu posłużyliśmy się bardzo wygodną i nowoczesną technologią React - biblioteką, wykorzystywaną do tworzenia klienckich aplikacji internetowych. Pozwala ona tworzyć aplikacje oparte o komponenty, które ustawione są względem siebie w strukturze drzewiastej, gdzie korzeń najczęściej stanowi komponent *App.js*. Za pośrednictwem komponentów mamy możliwość dodawania nowych funkcjonalności w sposób niezależny od siebie, co pozwala na równoległą pracę oraz na łatwiejszą organizację kodu.

- Dodatkową zaletą Reacta jest jego prostota, bariera wejścia dla osoby z minimalną wiedzą z zakresu JavaScript jest niska i pozwala bez większego wysiłku zacząć tworzyć ciekawe projekty. Dzięki udogodnieniom oferowanym przez tę bibliotekę programista ma możliwość przekazywania parametrów do utworzonych komponentów. Poniższy kod stanowi przykład wykorzystania języka JSX w celu zastosowania dynamicznych zmiennych bezpośrednio w znaczniku *Canvas*, w języku HTML.

```
<Canvas
```

```
 width={this.state.width}
 height={this.state.height}
 point={this.state.point}
 mode={this.state.mode}
 thickness={this.state.thickness}
 color={this.state.color}
 reset={this.state.reset}
/>
```

- Dzięki takiemu przekazywaniu danych (*props*) do komponentu *Canvas,* który jest w dużym uogólnieniu naszym wrapperem na element *canvas* w *HTMLu*, każda z tych zmiennych, gdy zostanie zmieniona, komponent *Canvas* sam się zaktualizuje, a następnie wyrenderuje na podstawie zmienionych danych - nie ma potrzeby na manualne wywoływania aktualizacji strony, dzięki czemu kod jest znacznie bardziej przejrzysty, a cała generyczna logika aktualizacji strony jest obsłużona przez *React*.

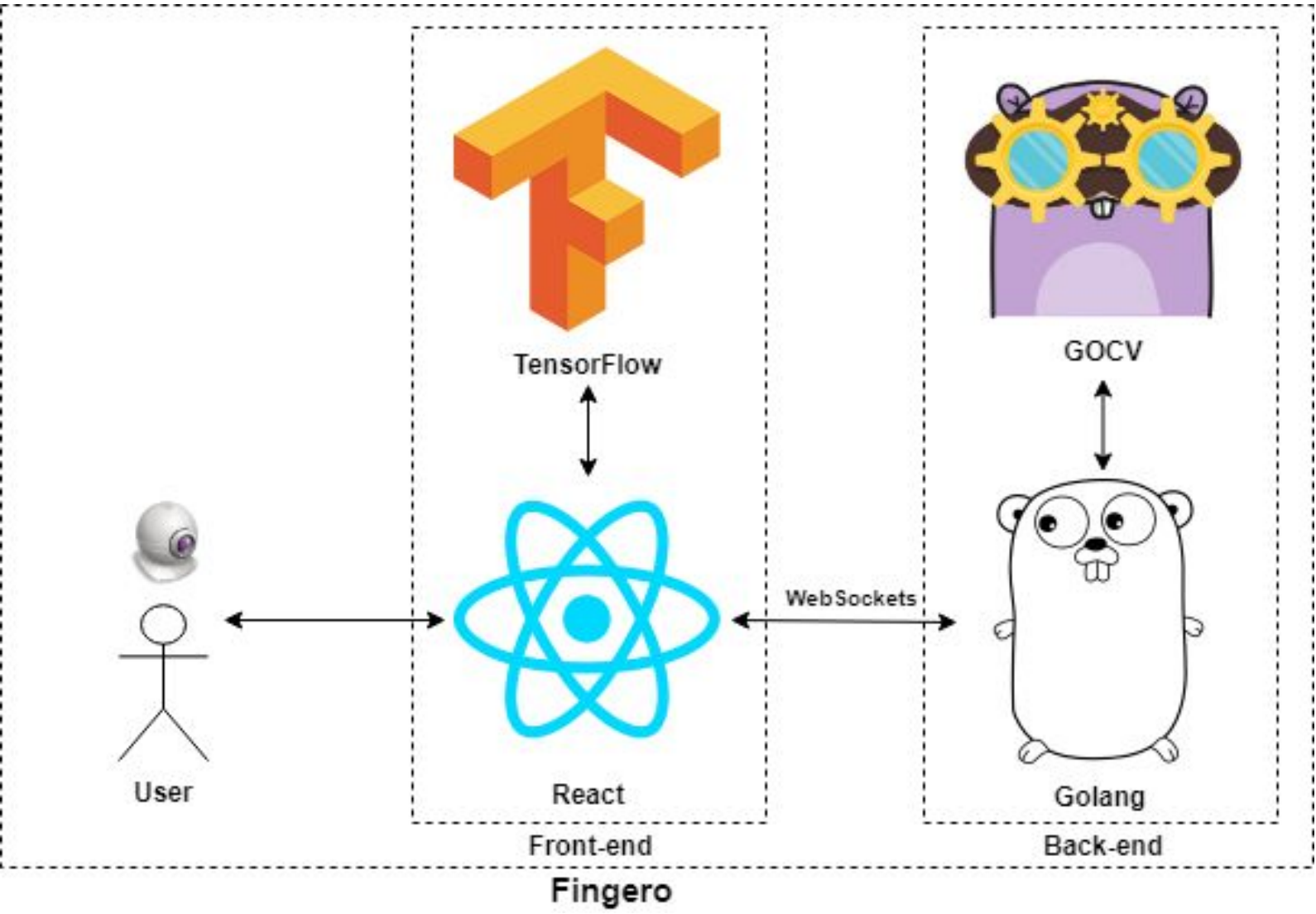
- Użyliśmy również wbudowanej w JavaScript obsługi protokołu WebSockets do wysyłania i odbierania danych z serwerem. Inicjalizacja połączenia następuje, gdy użytkownik zaczyna rysować (poprzez naciśnięcie przycisku *start video*):

```
this.setState(
{ socket: new WebSocket("ws://localhost:8080/ws") },
this.addSocketMethods
);
```

- Kod ten oprócz inicjowania połączenia z serwerem, również ustawia stan zmiennej *socket* na obiekt z połączenia protokołem WebSockets. Do zamknięcia takiego połączenia wystarczy wyciągnąć ten obiekt z całego stanu komponentu głównego, a następnie wywołać funkcję zamykającą połączenie:

```
  const { socket } = this.state;
  socket.close();
```

6. Architektura rozwiązania



Rys 8. Architektura systemu Fingero

- Po stronie serwerowej wykorzystujemy Golang wraz z zainstalowaną biblioteką OpenCV z którego korzysta biblioteka GOCV. Jest on używany do rozpoznawania gestów (ilości pokazanych palców), a następnie odsyłania JSONa z polem count zawierających ilość. ***(Kod z rozpoznawaniem ilości palców + opis)***

- W celu komunikacji z front-endem wykorzystujemy WebSocket. Obraz z kamery internetowej użytkownika jest przesyłany formacie *png*, zakodowany w *Base64*. ***(wysłanie kod frame z fronta na back + opis kodu).*** WebSockets umożliwiają serwerowi i klientowi wysyłanie wiadomości do siebie w dowolnym momencie, po ustanowieniu połączenia, bez wyraźnego żądania jednego lub drugiego. Kontrastuje to z HTTP, który tradycyjnie wiąże się z zasadą wezwanie-odpowiedź - gdzie, aby uzyskać dane, należy je wyraźnie zażądać. Pozwala to na przetwarzanie obrazu w czasie rzeczywistym.

7. Interesujące problemy i rozwiązania ich na jakie się natknęliście

Wykrywanie dłoni na podstawie zakresu HSV obrazu

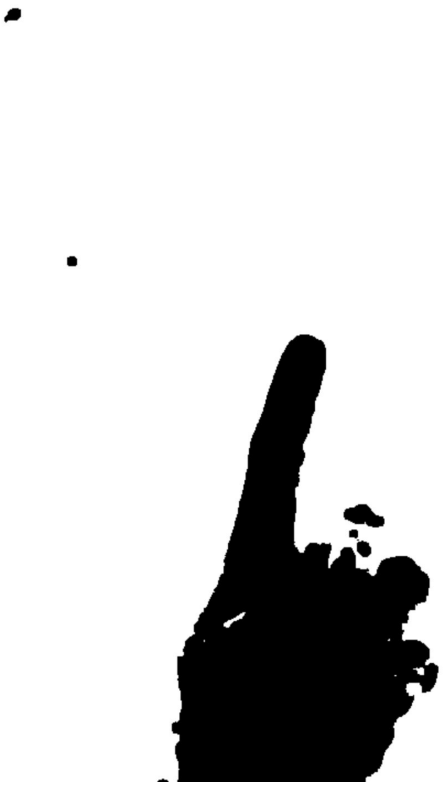
Wczytywany obraz zostaje przekonwertowany z modelu przestrzeni barw BGR na HSV, co pozwala na rozdzielenie go na składowe:

- odcień światła,
- nasycenie koloru,
- jasność (wartość światła białego).

Rys 9. Obraz przekonwertowany z BGR na HSV

- Takie podejście pozwala w prosty sposób zastosować operację progowania obrazu (funkcja *gocv.InRangeWithScalar*), używając

wcześniej zmierzonych i ustalonego zakresu wartości zgodnych z kolorami skóry. Po zastosowaniu operacji, wartości w obrazie, które nie mieszczą się w podanym zakresie, zostają usunięte, a na otrzymanym obrazie możemy wykonać funkcje *gocv.Blur* i *gocv.Threshold,* w celu usunięcia jak największej ilości szumów (na obrazku widoczne głównie w lewym górnym rogu oraz na oknach na środku). Efekt operacji można obserwować na następującym obrazie:



Rys 10. Obraz uzyskany po szeregu operacji progowania oraz odszumiania

- Następnie, na obrazie wykonujemy operacją wykrywania konturów - *gocv.FindContours,* która pozwala oddzielić wszystkie znalezione obiekty (o ile nie są ze sobą ściśle złączone) i zwróci listę list punktów - a konkretnie struktur z polami *int* - X oraz Y (współrzędnymi).
- Kolejnym krokiem jest wybranie jednej listy, odpowiadającej z możliwie największym prawdopodobieństwem palcowi, którym posługuje się użytkownik do rysowania, a nie na przykład głowie, czy innym częściom ciała ludzkiego. Zdecydowaliśmy się na jedno z najbardziej intuicyjnych

i prostych rozwiązań - wybór listy, odpowiadającej kształtowi o największym polu wierząc, że będzie ona przechowywać informacje o oczekiwanym kształcie.

- Kolejnym założeniem, które przyjęliśmy w tej metodzie jest uznanie najwyższego punktu w wybranym kształcie za czubek palca, czyli ręka zawsze musi być wyprostowana. Miało to być rozwiązanie przejściowe w tej metodzie, używane tylko na początku, jednak nigdy nie zostało ulepszone ze względu na to, że całkowicie porzuciliśmy ten pomysł. O decyzji zadecydowało wiele powodów:
  - Wartości HSV mające odpowiadać kolorowi skóry były wpisane w programie, a nie modyfikowane lub kalibrowane, przez co aplikacja działała tylko w dziennych warunkach oświetleniowych.
  - Jako, że zawsze wybieraliśmy listę punktów o największym polu, musieliśmy za każdym razem to pole obliczyć dla każdego obiektu z podanego przedziału HSV (np. głowa, ramię, jasna ściana). Gdy takich obiektów pojawiało się wiele, serwer zauważalnie spowalniał i występowały opóźnienia w renderowaniu rysowanego obrazu na stronie.
  - W przypadku gorszych warunków oświetleniowych występowało dużo szumów i części z nich nie dało się pozbyć, gdyż nachodziły na przykład na rękę, przez co algorytm minimalnie zmieniał co klatkę najwyższy punkt ręki co powodowało skoki oraz ostre linie na rysowanym obrazie.
  - Większość czasu w tym podejściu serwer tracił na enkodowanie i dekodowanie obrazu png do base64.
  - Przy próbach rozwoju algorytmu natrafiliśmy na braki w funkcjach w GOCV, które po prostu nie były jeszcze zaimplementowane w momencie pisania aplikacji. Niewykluczone, że jest to sytuacja przejściowa i biblioteka ta stanie się w końcu kompletna.
  - Algorytm czasem wybierał głowę (podobny kolor skóry), mimo

że ta była oddalona i wyraźnie mniejsza od ręki (prawdopobnie pole z głowy dało się dokładniej policzyć - lista punktów była większa z uwagi na bardziej regularny kształt głowy, przez co i pole mogło być większe).

- Zarówno operacja wykrywania ręki, jak i wykrywania gestów odbywały się na serwerze, przez co był on mocno obciążony, przez co już kilku klientów generowało spore obciążenie i były zauważalne obciążenia - wymiana danych przez WebSockets nie odbywała się w regularnych odstępach czasowych. Był to główny powód późniejszych pomysłów na przeniesienie logiki wykrywania i śledzenia ręki do klienta, żeby serwer mógł się skupić jedynie na wykrywaniu gestów.

Wykrywanie dłoni z wykorzystaniem histogramów koloru skóry

- Metoda, która w teorii miała umożliwić rozpoznawanie dłoni niezależnie od warunków otoczenia - zmieniającego się oświetlenia, koloru skóry osoby testującej oprogramowanie. Histogramy kolorów dostarczają znacznie więcej zróżnicowanych informacji w przeciwieństwie do pojedynczego zakresu wartości jaki oferowała poprzednia metoda.
- Na ekranie w początkowej fazie ukazywane są prostokąty, wskazujące użytkownikowi miejsca, z których pobrane zostaną wartości, na podstawie których następnie liczone są histogramy kolorów.
- Na tym etapie pojawił się główny problem z tym podejściem - brakująca implementacja funkcji *cv2.calcBackProject*, mającej umożliwić wyodrębnienie fragmentu obrazu od reszty, wybierając jedynie wartości z zakresów wyznaczonych przez wyliczone histogramy.
- Kolejnym krokiem było wyznaczenie wszystkich konturów z wyodrębnionego obrazu oraz wybranie jednego - o największej powierzchni. Dla danego konturu obliczane zostają momenty, dzięki którym dzieląc $moment_{10}$ przez $moment_{oo}$ otrzymujemy współrzędną x

centroidu; w sposób analogiczny można wyliczyć punkt współrzędną y - $moment_{01}$ / $moment_{00}$. Przez centroid rozumiemy punkt środkowy kształtu. Dodatkowo, na podstawie największego konturu zostaje wyznaczona otoczka wypukła, która “uwypukla wklęsłe” części konturu, niejako rozszerzając go. Na podstawie otrzymanego kształtu wystarczy wybrać najbardziej odległy od centrum punkt i przyjąć go za czubek palca. Zakładamy, że palec, wykorzystywany do rysowania jest najbardziej wyróżniającym się.

- Etap końcowy stanowi stworzenie listy, przechowującej ostatnie pozycje czubka palca, która zostaje wykorzystana do rysowania.

Wykrywanie dłoni w oparciu o przetrenowany model sieci neuronowej

- Za wykrywanie ręki odpowiada biblioteka TensorFlow.js, która jest została napisana i jest nadal rozwijana poprzez Google i została zaadaptowana do wykorzystania w JavaScript, co jest szczególnie przydatne w użyciu w aplikacji klienckiej, dzięki czemu można znacznie odciążyć serwer. Wygodne jest też wsparcie dla rozszerzeń adaptujących gotowe modele i tworzących predykcje na ich podstawie.

- Rozpoczęcie śledzenia ręki rozpoczyna się od razy po kliknięciu przycisku “*Start Video*”. Odpowiednio wybrany model rozpoznaje rękę i aplikacja jest skłonna do wykonywania operacji pokazywanymi gestami.

Wykrywanie gestów

- Liczba palców oraz przypisany tryb za jaki odpowiada:
  - liczba palców mniejsza od dwóch przypisana została do wyboru pędzla; z uwagi na fakt, że zerowa liczba palców jest uwzględniona w tym przypadku, również defaultowy tryb to pędzel do malowania;
  - dokładnie dwa palce pozwalają zwiększyć grubość pędzla;
  - dowolna liczba, większa od dwóch włącza tryb gumki,

umożliwiając zmazywanie dzieła przygotowanego przez użytkownika.

Renderowanie danych w czasie rzeczywistym

- Poniższy kod przedstawia działanie rysowania w czasie rzeczywistym oraz aktualnie wykorzystywane narzędzie. W celu nałożenia nowego skrawka tworzonego rysunku brana jest pod uwagę ostatnia pozycja palca oraz obecna i na tej podstawie rysowana jest linia. Ponieważ opóźnienia są bardzo małe użytkownik nie widzi najmniejszego opóźnienia związanego z nakładaniem dodatkowej warstwy na obraz. Dodatkowymi parametrami funkcji jest kolor oraz grubość rysowanej linii.

```
updateCanvas = (point, mode, thickness, color) => {
        const ctx = this.refs.canvas.getContext('2d');
        ctx.lineCap = 'round';
        if (Math.abs(point.prevX - point.x) > 75 ||
Math.abs(point.prevY - point.y) > 75) {
            point.x = point.prevX - 1;
            point.y = point.prevY - 1;
        }
        if (mode === "pen") {
            ctx.globalCompositeOperation = "source-over";
            ctx.beginPath();
            ctx.moveTo(point.prevX, point.prevY);
            ctx.lineTo(point.x, point.y);
            ctx.strokeStyle = color;
            ctx.stroke();
        } else if (mode === "brush") {
            ctx.globalCompositeOperation = "source-over";
            ctx.lineWidth = thickness;
            ctx.beginPath();
            ctx.moveTo(point.prevX, point.prevY);
            ctx.lineTo(point.x, point.y);
            ctx.strokeStyle = color;
            ctx.stroke();
        } else if (mode === "eraser") {
            ctx.globalCompositeOperation =
```

```
            "destination-out";
                        ctx.arc(point.x, point.y, thickness, 0,
        Math.PI * 2, false);
                        ctx.fill();
                    }
                }
```

- Nie każda linia może być narysowana zgodnie z zamierzeniem autora dlatego do naszej aplikacji wprowadziliśmy gumkę, którą użytkownik może użyć w celu poprawienia niechcianego elementu.

8. Instrukcja użytkowania aplikacji

Do uruchomienia aplikacji wymagane są:

Golang 1.12

- GoCV 0.19.0
- Gorilla Websocket 1.4.0

OpenCV 4.1.0

npm 6.9.0

W celu uruchomienia aplikacji należy:

- Uruchomić serwer

  >> go run api/main.go
- Uruchomić aplikację

  >> cd ui

  >> npm install

  >> npm start
- Otworzyć aplikację, domyślnie pod adresem http://localhost:3000/

W celu zbudowania aplikacji należy:

- Zbudować serwer

  >> go build api/main.go
- Zbudować stronę

```
>> cd ui
>> npm install
>> npm build
```