SPRAWOZDANIE Z LABORATORIUM SYSTEMÓW WIZYJNYCH

Sterowanie chwytakiem za pomocą gestów dłoni

1 Cel projektu

Należy opracować język porozumiewania się z robotem za pomocą gestów dłoni. Powinien on zawierać przynajmniej następujące komendy:

- zamknij chwytak (dwa palce zwarte)
- otwórz chwytak (dwa palce zwarte)
- przesuń o skok w lewo (palec w prawo pod kątem 45 stopni)
- przesuń o skok w prawo (palec w prawo pod kątem 45 stopni)
- stop (palce schowane)

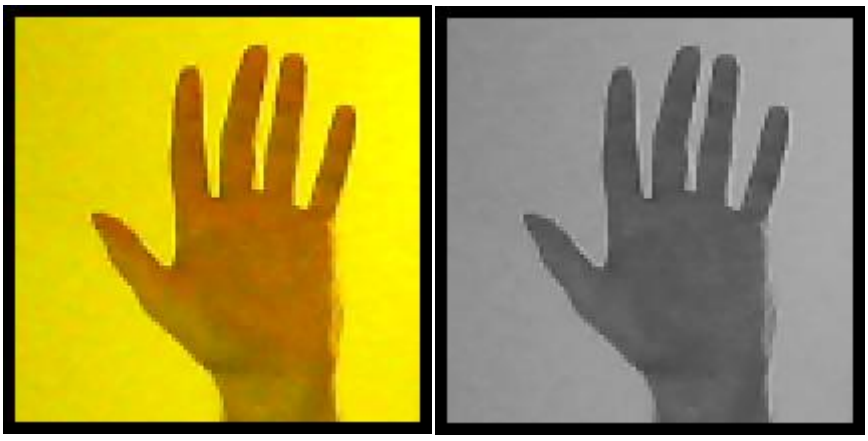
Rozpoznanie komendy w najprostszym przypadku sygnalizowane komunikatem słownym lub jako sterowanie modelem graficznym (symbol szczęk robota przesuwających się lewo/prawo). Pozycja dłoni może być ustalona (względem np. wyrysowanej siatki), tło do wyboru (np. czarne). Program powinien pracować z kamerą rzeczywistą.

2 Zamiana obrazu na czarno - biały

Obraz czarno-biały (w odcieniach szarości) to obraz, w którym zawartość każdej składowej (R, G, B) jest jednakowa, czyli składowe są w stosunku 1:1:1. Zatem, aby uzyskać obraz w odcieniach szarości należy tak go przekształcić, żeby dla każdego piksela obrazu odpowiednio do stopnia jasności piksela udział poszczególnych składowych był taki sam. W tym celu początkowo odczytywany jest piksel obrazu o danym kolorze z odpowiednią zawartością każdej składowej podstawowej i wyróżnione zostają składowe RGB.

```
mono_color_temp = ( rgb_temp.red() +  rgb_temp.green() + rgb_temp.blue() ) / 3
```

Kolejny krok stanowi sumowanie jasności każdej barwy podstawowej oraz podział uzyskanej sumy przez 3 (bez reszty). Po powyższej operacji trzeba dokonać syntezy koloru z trzech składowych. Operacje wykonujemy w pętli dla każdego piksela obrazu.



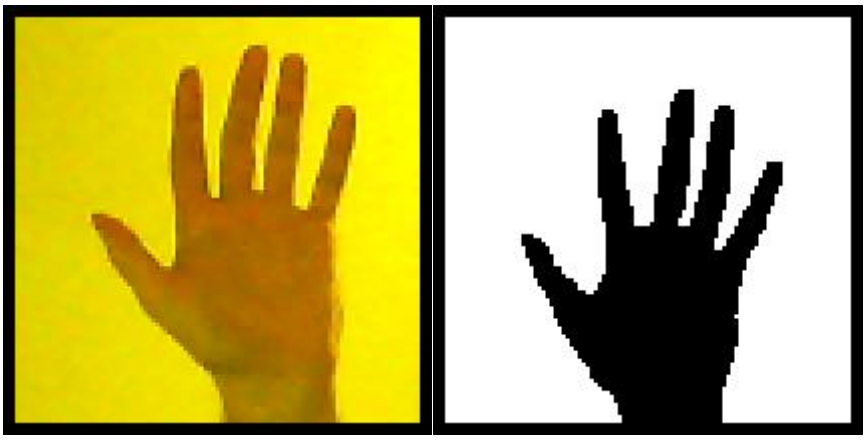
Rys. 2.1 Zamiana obrazu kolorowego na czarno - biały

3 Binaryzacja

Binaryzacja polega na zamianie wartości pikseli z odcieni szarości albo kolorów na wartości 0 lub 1. Celem tego procesu jest radykalna redukcja informacji zawartej w obrazie.

```
    if ( !ui->real->isChecked() and ui->reverse->isChecked() ){
        (ui->thresholding_value->value() <= mono_color_temp ) ? rgb_temp.setRgb( 0, 0, 0 ):
rgb_temp.setRgb( 255, 255, 255 );
    }

    if ( !ui->real->isChecked() and !ui->reverse->isChecked() ){
        (ui->thresholding_value->value() <= mono_color_temp ) ? rgb_temp.setRgb( 255, 255, 255 ):
rgb_temp.setRgb( 0, 0, 0 );
    }

    if ( ui->real->isChecked() and ui->reverse->isChecked() and ui->thresholding_value->value() <=
mono_color_temp ){
        rgb_temp.setRgb( 0, 0, 0 );
    }

    if ( ui->real->isChecked() and !ui->reverse->isChecked() and ui->thresholding_value->value() >
mono_color_temp ){
        rgb_temp.setRgb( 0, 0, 0 );
    }

    rgb_current = rgb_temp;
}
```

Zazwyczaj realizowana jest przez progowanie, polegające na ustaleniu wartości progowej, poniżej której piksele obrazu klasyfikowane są jako piksele obiektu, natomiast pozostałe piksele klasyfikowane są jako piksele tła.



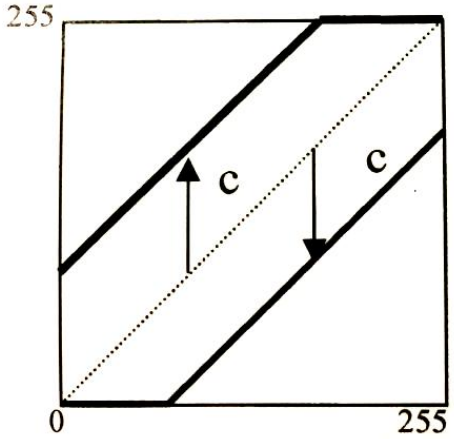
Rys. 3.1 Operacja binaryzacji

4 Korekcja jasności obrazu

Zmiana jasności obrazu sprowadza się do wykonania przekształcenia obrazu przy pomocy odpowiednio przygotowanej tablicy LUT (*ang. lookup table*). Współczynniki w takiej tablicy wyznaczane są według następującego wzoru:

$$LUT(i) = \begin{cases} 0 & if \quad i+c < 0 \\ i+c & if \quad 0 \le i+c \le i_{max} \\ i_{max} & if \quad i+c > i_{max} \end{cases}$$

gdzie i_{max} oznacza maksymalną dopuszczalną wartość składowej piksela obrazu. Jeżeli wartość stałej c, jest większa od 0, to nastąpi zwiększenie jasności obrazu. W przeciwnym wypadku, gdy wartość c jest mniejsza od 0, nastąpi zmniejszenie jasności obrazu. Wykresy powyższych krzywych przedstawione są poniżej.



Rys. 4.1 Krzywa korekcji jasności obrazu

Jak można zauważyć zmiana jasności sprowadza się do dodania do wartości wszystkich składowych RGB obrazu pewnej stałej c. Jeżeli wynik jest większy od zakresu, to należy przyjąć wartość maksymalną, natomiast jeśli jest on mniejszy od zakresu, wówczas należy założyć wartość minimalną. Zwiększanie jasności obrazu powoduje przesunięcie histogramu obrazu w prawo, zaś zmniejszenie jasności przesunięcie w lewo.

Rys. 4.2 Korekcja jasności obrazu. (Od lewej) zmniejszona, oryginalna, zwiększona jasność

5 Korekcja kontrastu

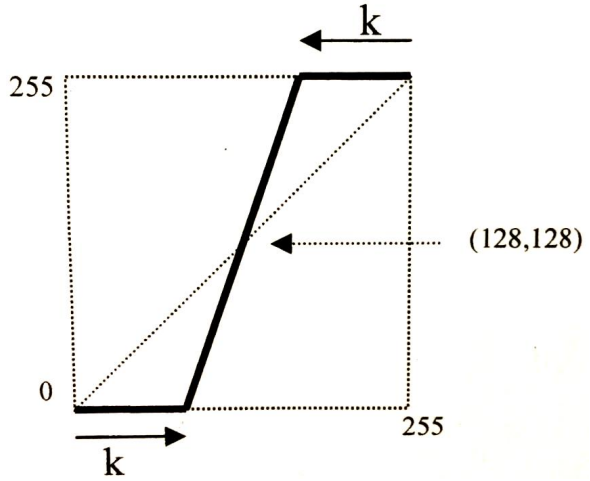
Podobnie jak przy binaryzacji zmiana kontrastu obrazu realizowana jest przy pomocy odpowiednio przygotowanej tablicy LUT. Współczynniki w tej tablicy wyznacza się w sposób następujący:

$$LUT(i) = \begin{cases} 0 & if \quad k(i-\frac{i_{max}}{2}) + \frac{i_{max}}{2} < 0 \\ k(i-\frac{i_{max}}{2}) + \frac{i_{max}}{2} & if \quad 0 \le k(i-\frac{i_{max}}{2}) + \frac{i_{max}}{2} \le i_{max} \\ i_{max} & if \quad k(i-\frac{i_{max}}{2}) + \frac{i_{max}}{2} > i_{max} \end{cases}$$

gdzie i_{max} oznacza maksymalną dopuszczalną wartość składowej RGB piksela obrazu.

```
k = ui->correction_contrast->value();

for ( int i = 0; i<256; i++){
   if (i <= k ) LUT[i] = 0;
   if (i >= 255 - k) LUT[i] = 255;
   if (((255 / ((255 - k) - k)) * i - (255 / ((255 - k) - k)) * k <= 255 ) and
      ((255 / ((255 - k) - k)) * i - (255 / ((255 - k) - k)) * k >= 0) and
      (i > k) and (i < 255 - k)) {
      LUT[i] = (255 / ((255 - k) - k)) * i - (255 / ((255 - k) - k)) * k;
   }
}
```

Jeżeli wartość stałej *k*, czyli współczynnika kierunkowego prostej, jest większa od 1, to nastąpi zwiększenie kontrastu obrazu. W przeciwnym wypadku, gdy wartość *k* jest mniejsza od 1, nastąpi zmniejszenie kontrastu obrazu. Poniżej wykres opisanej krzywej.



Rys. 5.1 Krzywa korekcji kontrastu obrazu

6 Negatyw obrazu

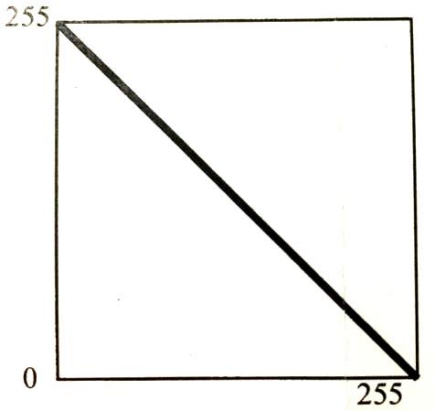
Do uzyskania negatywu obrazu wykorzystuje się odpowiednio skonstruowaną tablicę LUT. Dla tego przypadku współczynniki tej tablicy obliczamy według poniższej formuły:

$$LUT(i) = i_{max} - i$$

gdzie i_{max} oznacza maksymalną dopuszczalną wartość składowej RGB piksela obrazu.

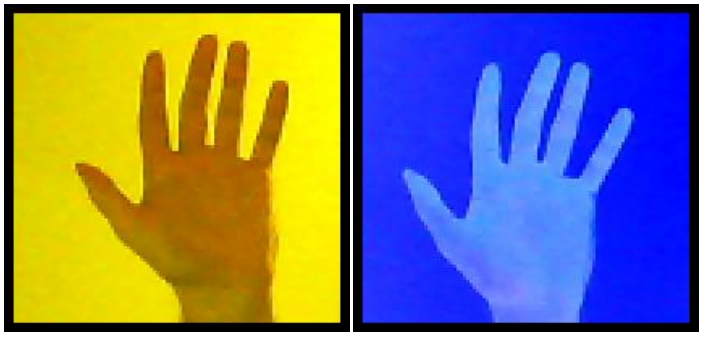
```
for ( int i = 0; i<256; i++){
   LUT[ i ] = 255 - i;
}
```

Wykres operacji negatywu obrazu kolorowego będzie zatem wyglądał tak:



Rys. 6.1 Krzywa obrazująca negatyw obrazu

Realizację negatywu obrazu sprowadza się bowiem do odjęcia od maksymalnej dopuszczalnej wartości składowej RGB obrazu. Negatyw to dopełnienie obrazu.



Rys. 6.2 Operacja negatywu na obrazie

7 Wykrywanie obiektu

Po ustawieniu odpowiednich filtrów zapewniających eliminację szumów związanych z tłem na którym znajduje się obiekt oraz przeprowadzeniu operacji progowania (binaryzacji) możemy przystąpić do wykrycia obiektu. Algorytm który został tutaj zastosowany jest stosunkowo prosty i nie jest odporny na szumy, więc obiekt który chcemy wykryć musi być na jednolitym tle, ponieważ sam algorytm detekcji obiektu nie ma funkcji eliminacji szumów. Detekcja obiektu jest przeprowadzana za pomocą dwóch pętli *for*, które przeszukują cały obraz zaczynając od lewego górnego piksela skończywszy na dolnym prawym. Po napotkaniu piksela o innym kolorze niż tło są aktualizowane zmienne *xmin*, *ymin*, *xmax* oraz *ymax*, które po wykonaniu całej funkcji detekcji obiektu zawierają współrzędne prostokąta stanowiący obrys wykrytego obiektu.

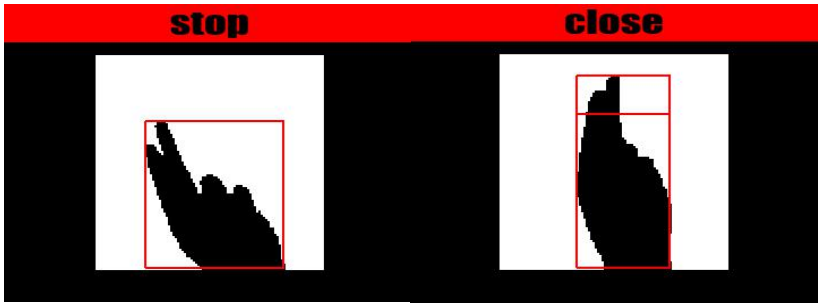
```
If ( rgb_temp.black() ) {
   if( x > xmax ) xmax = x;
   if( x < xmin ) xmin = x;
   if( y > ymax ) ymax = y;
   if( y < ymin ) ymin = y;
}
```

Poprawność tego algorytmu zależy w dużym stopniu od wcześniejszej filtracji przeprowadzonej ręcznie przez użytkownika za pomocą zmiany jasności / kontrastu oryginalnego obrazu oraz korekcji RGB, w taki sposób aby zapewnić jak najlepszą binaryzację tj. widoczny sam detal bez dodatkowych szumów tła.

Rys. 7.1 Szumy tła

8 Rozpoznanie gestów dłoni

Kolejnym etapem jest wykrycie gestów dłoni. W tym algorytmie zostały przyjęte założenia że dłoń jest umieszczona w orientacji pionowej w taki sposób, że w analizowanym obszarze jest widoczna dłoń od nadgarstka do końca palców. Tylko takie usytuowanie dłoni przed kamerą gwarantuje poprawne działanie algorytmu.

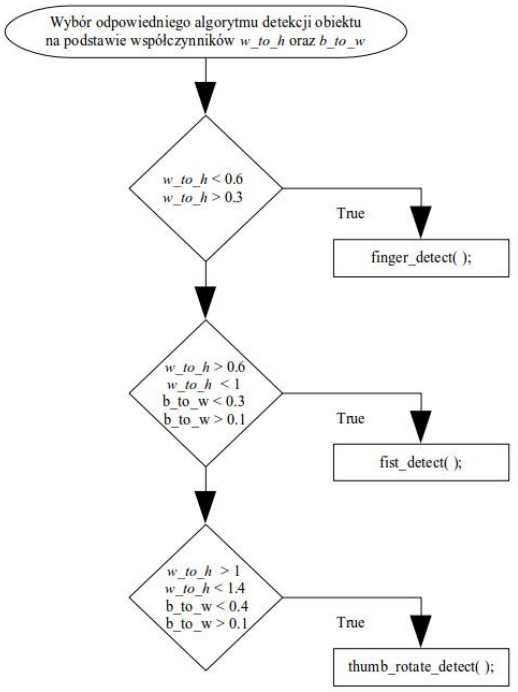


Rys. 8.1 Zła i prawidłowa orientacja dłoni

Cała funkcja mająca za zadanie rozpoznanie gestów dłoni i zwrócenia odpowiedniej wartości obrazującą dany gest jest rozłożona na kilka mniejszych funkcji, które są wykonywane na podstawie dwóch współczynników *w_to_h* oraz *b_to_w*, które stanowią odpowiednio stosunek szerokości wykrytego obiektu do jego wysokości oraz stosunek ilości czarnych do białych pikseli na wykrytym obszarze.

```
white_pixel_count ? b_to_w =(double)(black_pixel_count)/(double)(white_pixel_count) : b_to_w = 0;
( (ymax - ymin) != 0 ) ? w_to_h = (double)( xmax - xmin ) / (double)( ymax - ymin ) : w_to_h = 0;
```

Poniżej został przedstawiony schemat blokowy wyboru odpowiedniej funkcji na podstawie podanych wyżej współczynników.

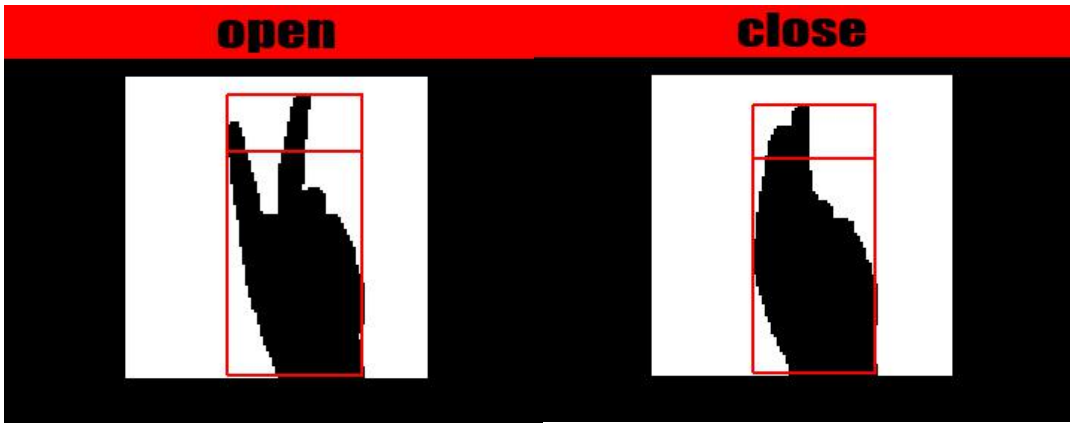


Rys. 8.2 Schemat blokowy wyboru odpowiedniego algorytmu

Funkcja *finger_detect( )* ma za zadanie wykrycie czy palec wskazujący oraz środkowy są ze sobą złączone, czy rozwarte stanowiąc gest victorii. Realizacja jest stosunkowo prosta i nie wymaga dużej ilości operacji, więc nadaje się do analizy obrazu w czasie rzeczywistym nawet gdy nie dysponujemy dużą mocą obliczeniową jednostki centralnej CPU. Funkcja analizuje tylko jedną linię obrazu na wysokości stanowiącej 80% całej wysokości obiektu. Rozpoznanie gestu jest dokonywane na podstawie ilości zmian koloru pikseli z białych na czarne.

```
    finger = 0;
    bool lock = false;
    for ( int temp_x = 2*xmin; temp_x <= 2*xmax; temp_x++ ){
       if ( lock == false and image_3.pixelColor( temp_x, 2*(finger_detect_line) ).black() ){
          lock = true;
          finger++;

       }
       if( lock == true and !image_3.pixelColor( temp_x, 2*(finger_detect_line) ).black() ){
          lock = false;
       }
    }
    detected_gesture = finger;
 }else{
    finger = 0;
 }
```

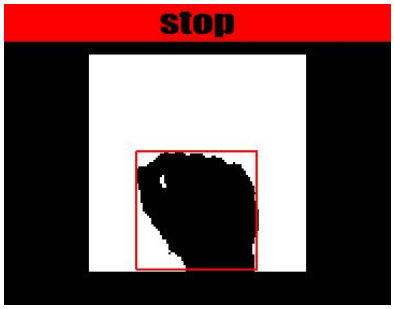
Jeżeli palce są złączone to taka zmiana nastąpi tylko raz, jeśli palce są rozłożone to zmiana taka wystąpi dwa razy. Na tej podstawie funkcja zwraca wynik 1 (palce złożone) lub 2 (gest victorii).



Rys. 8.3 Przykład działania funkcji finger_detect()

Kolejną funkcją jest *fist_detect( )*, która jest bardzo prostą funkcją i bazuje tylko na współczynnikach *w_to_h* oraz *b_to_w*. Jeśli współczynniki mieszczą się w odpowiednim przedziale to funkcja zwraca wartość równą 3. Oznacza to, że został wykryty gest pięści.

```
if ( w_to_h > 0.6 && w_to_h < 1 && b_to_w < 0.3 && b_to_w > 0.1){
   detected_gesture = 3;
}
```

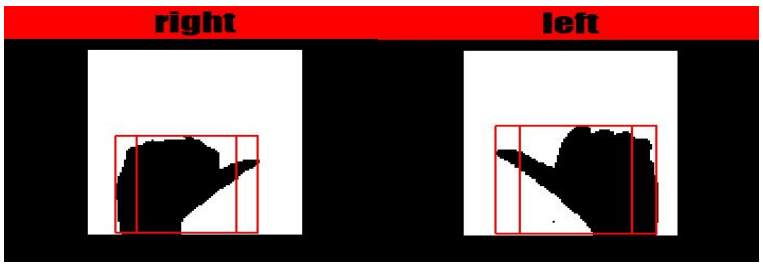


Rys. 8.4 Rozpoznanie gestu pięści

Ostatnią funkcja użytą do rozpoznania gestów jest *thumb_rotate_detect( )*, której calem jest wykrycie orientacji kciuka. Działanie tej funkcji jest zbliżone do funkcji *finger_detect( ),* z tą różnicą że nie badamy ilości zmian koloru pikseli a sumę czarnych pikseli. Algorytm bazuje na ilości czarnych pikseli z lewej i prawej strony wykrytego obiektu.

```
      for ( int temp_y_l = 2*ymin; temp_y_l <= 2*ymax; temp_y_l++ ){
         ( image_3.pixelColor( 2*(thumb_rotate_detect_line_l), temp_y_l ).black() ) ?
temp_count_black_l++ : temp_count_black_l ;
      }
      for ( int temp_y_r = 2*ymin; temp_y_r <= 2*ymax; temp_y_r   ++ ){
         ( image_3.pixelColor( 2*(thumb_rotate_detect_line_r), temp_y_r ).black() ) ?
temp_count_black_r++ : temp_count_black_r;
      }

      temp_count_black_l > temp_count_black_r ? detected_gesture = 4 : detected_gesture = 5;
   }
```

Funkcja analizuje dwie kolumny obrazu prowadząc dwie proste pionowe na szerokości stanowiącej 30% oraz 70% całej szerokości obrazu, sumując ilość czarnych pikseli. Na podstawie tych dwóch wyników jest określana orientacja kciuka. Jeśli zagęszczenie czarnych pikseli jest większe z lewej strony w stosunku do prawej oznacza orientację kciuka w prawą stronę, w przeciwnym wypadku kciuk jest zorientowany w lewą stronę. Funkcja zwraca dwie wartości 4 oraz 5 co oznaczają odpowiednio kciuk zwrócony w prawo i kciuk zwrócony w lewo.

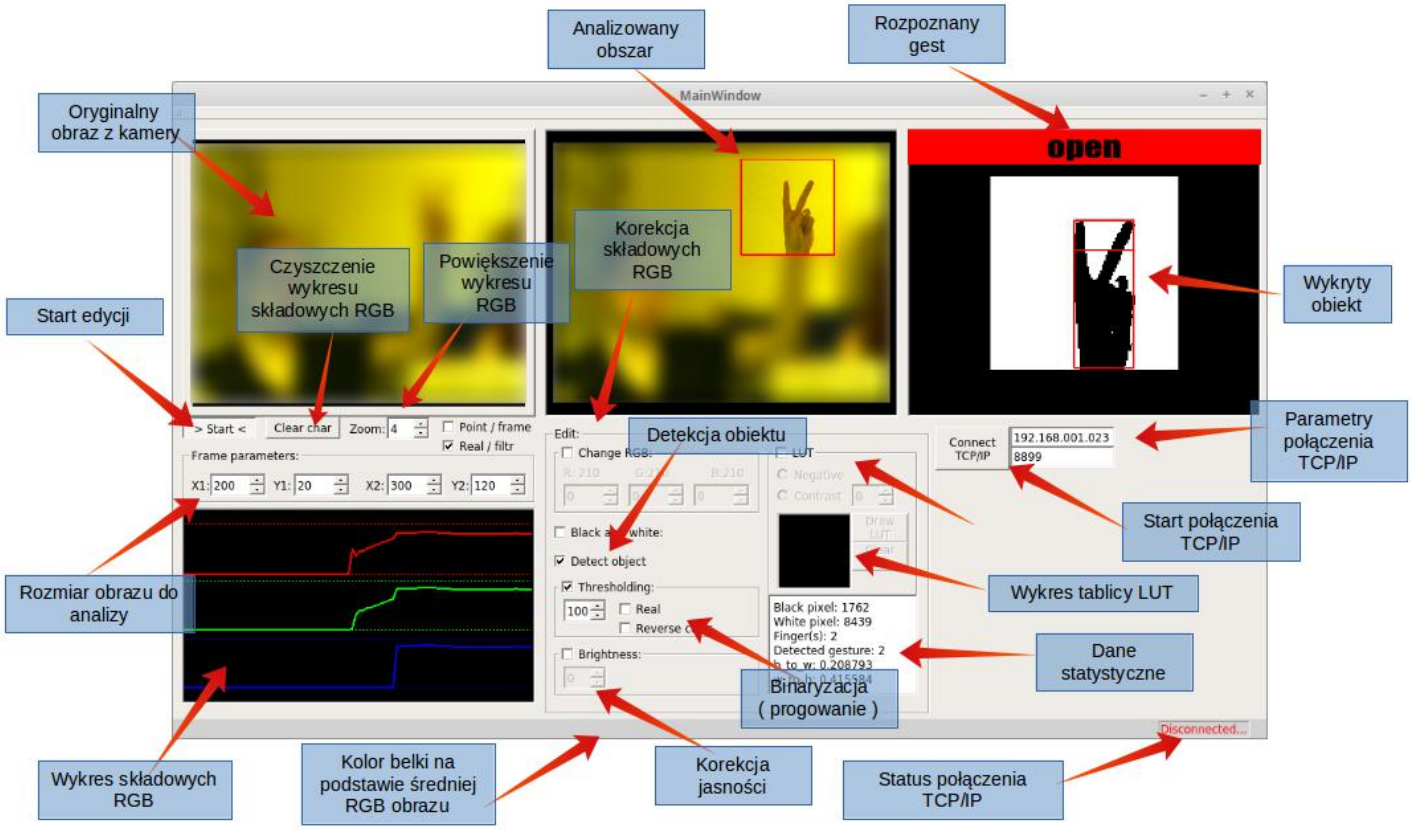


Rys. 8.5 Wykrycie orientacji kciuka

9 Fizyczna realizacja

Cała aplikacja docelowo została stworzona do pracy z rzeczywistym chwytakiem, gdzie za pomocą gestów dłoni użytkownik może sterować orientacją chwytaka. Odpowiednio dla rozłożonych palców chwytak ma być otwarty, po złączeniu palców chwytak ma zostać zamknięty. Orientacja kciuka jest wykorzystywana do zmiany położenia chwytaka. Po zwróceniu kciuka w lewą stronę chwytak ma poruszać się do góry, zaś po zwróceniu w prawą stronę ma poruszać się w dół. Pięść symbolizuje komendę stop, czyli zatrzymanie chwytaka.

Poniżej został przedstawiony graficzny interfejs aplikacji napisanej w środowisku programistycznym *QtCreator* realizujące wszystkie wyżej wymienione funkcje.



Przykład użycia programu we współpracy z chwytakiem poruszającym się w płaszczyźnie góra - dół: https://www.youtube.com/watch?v=K2m1moGp3d8

10 Podsumowanie

W projekcie zostały zrealizowane wszystkie zadania jakie zostały postawione przez prowadzącego. Algorytmy użyte w projekcie są bardzo przejrzyste dzięki czemu mogą być dalej rozwijane przez innych użytkowników. Na poprawność działania algorytmu kluczową rolę odgrywa odpowiednia filtracja oryginalnego obrazu przez użytkownika stosują odpowiedniego rodzaju filtry korekcyjne w celu zapewnienia najlepszej binaryzacji obrazu. Mając do dyspozycji narzędzia, którymi możemy korygować wszystkie parametry i składowe obrazu w prosty sposób możemy uzyskać zadowalający nas efekt.

Dalsza optymalizacja algorytmu może być zrealizowana poprzez automatyczną zmianę współczynników *b_to_w* oraz *w_to_h* w zależności od aktualnej sceny i czynników zewnętrznych np. przy słabszym oświetleniu.