AKADEMIA GÓRNICZO-HUTNICZA
im. St. Staszica w Krakowie
WEAIiE, Katedra Automatyki

Laboratorium Biocybernetyki

Przedmiot: Sztuczna inteligencja i sensoryka.

PR063HCI4

Temat projektu: -HCI4- **Wirtualna myszka - detekcja i śledzenie dłoni przy pomocy algorytmu „camshift".**

Spis treści

Wykonali: Andrzej Jasiński, Tomasz Huczek III rok AiR
Konsultant: Jaromir Przybyło

1. ABSTRAKT

Celem naszego projektu jest wykrycie a następnie śledzenie i wyznaczenie orientacji poruszającej się dłoni, a następnie identyfikacja zestawu podstawowych gestów i wygenerowanie poleceń dla systemu operacyjnego. Program ma posiadać budowę modułową, tak, aby możliwe było użycie różnych rozwiązań na poszczególnych etapach algorytmu, bez konieczności pisania programu od podstaw. Detekcja i śledzenie dłoni jest wykonywana za pomocą algorytmu CamShift, przy pomocy zmodyfikowanego algorytmu autorstwa naszych poprzedników (projekt z ubiegłego roku). Wykrywanie gestów jest oparte o algorytm z poprzedniego roku i nasz własny. Akwizycja obrazu jest dokonywana zarówno z plików wideo jak i urządzeń przechwytywania obrazu. Obecnie akwizycja jest możliwa tylko za pomocą biblioteki OpenCV, jednak dzięki elastycznej budowie modułowej istnieje możliwość użycia innych metod przechwytywania obrazu (np. DirectShow, lub Video For Windows).

Projekt jest wykonany całkowicie w języku C++, w środowisku Microsoft Visual Studio (wersja 2003). Interfejs użytkownika (GUI) oparty jest na bibliotece Qt 3. Program działa na zasadzie ładowanych dynamicznie bibliotek DLL, odpowiedzialnych za poszczególne etapy przetwarzania. Tor przetwarzania jest bardzo elastyczny, możliwe jest stworzenie dowolnego toru, istotne jest tylko, aby na początku był moduł akwizycji obrazu oraz następne moduły miały zgodność co do typów wejść-wyjść. W obecnej wersji programu kolejnymi etapami przetwarzania są: akwizycja obrazu z jednego z możliwych źródeł (plik lub urządzenie przechwytujące), obliczanie prawdopodobieństw przynależności do obiektu dla każdego piksela, obliczanie pozycji i kąta obrotu.

Wykrywanie dłoni (obiektu) odbywa się na podstawie analizy obrazu w formacie HSV, dlatego bardzo ważne jest odpowiednie ustawienie parametrów granicznych H S i V, tak aby dłoń była poprawnie wykrywana i śledzona. Niezbędna jest również kalibracja, aby ustawić parametry programu zależnie od panujących warunków.

Słowa kluczowe: camshift, opencv, RGB, HSV, Probability Image

2. WSTĘP

Problem sprzężenia komputera z człowiekiem jest znany i rozważany już od bardzo dawna. Mimo bardzo dużego postępu technologicznego, wciąż do sterowania kursorem na ekranie monitora używamy urządzenia zwanego myszką, wynalezionego już w latach 60 ubiegłego wieku. Pomimo rozwoju myszki i stosowania nowych technologii sama idea pozostała ta sama. Celem tego projektu jest próba zastosowania zupełnie innej metody sterowania komputerem. Dzięki ciągłemu wzrostowi

mocy obliczeniowej procesorów oraz ogólnej dostępności urządzeń przechwytujących obraz (kamery internetowe, karty telewizyjne połączone z kamerą, kamery DV połączone przez porty USB czy FireWire) rodzi się pomysł sterowania komputerem za pomocą informacji wizyjnej. Nasz projekt jest kolejną próbą realizacji tego pomysłu. Doświadczenia naszych poprzedników dowodzą, że jest to możliwe. W naszym programie korzystaliśmy z fragmentów kodu oraz niektórych algorytmów opisanych w poprzednich projektach.

Podstawowym problemem przy realizacji sterowania komputerem za pomocą obrazu jest sama detekcja dłoni i odróżnienie jej jednoznacznie i precyzyjnie od otoczenia. Bardzo przydatna okazuje się konwersja z formatu RGB do palety HSV, ponieważ kolor skóry dłoni białego człowieka mieści się zazwyczaj w przedziale 10 – 35 wartości H. Oczywiście przy zmianach oświetlenia, lub niedoskonałości kamery, w oparciu tylko o ten próg nie zawsze uda się wykryć poprawnie dłoń. Dlatego tak ważna jest kalibracja. Nasz program korzysta z algorytmu CamShift, zmodyfikowanego przez naszych poprzedników. Dokonali oni modyfikacji algorytmu, aby poprawić jego skuteczność. Jednocześnie nie zdecydowali się oni na zastosowanie algorytmu dostępnego w bibliotece OpenCV ze względu na jego nieskuteczność, jednak nasze obserwacje dowodzą, że implementacja w OpenCV działa poprawnie (aktualna wersja 1.0 zawiera dużo usprawnień i być może stąd wynika różnica).

Algorytm CamShift jest modyfikacją algorytmu MeanShift i w odróżnieniu od niego nie analizuje całej ramki aby znaleźć obiekt, tylko pewien fragment zwany oknem poszukiwań, w którym prawdopodobieństwo wystąpienia obiektu jest największe. Dzięki temu jest to algorytm szybszy niż MeanShift.

Algorytm zastosowany w naszym programie jest oparty na rezultatach projektów z poprzednich lat.

Najpierw jest wykonywana binaryzacja obrazu na podstawie analizy składowych HSV ramki obrazu. Skóra zazwyczaj mieści się w przedziale 10-35 H, 20-255 V. Oczywiście przy różnych warunkach oświetleniowych wartości te ulegną zmianie i dlatego też niezbędna jest kalibracja. Następnie na podstawie histogramu obliczane jest prawdopodobieństwo (Wzór 1) przynależności piksela do obiektu.

3. KONCEPCJA PROPONOWANEGO ROZWIĄZANIA

- Definiowanie modelu skóry
- Tworzenie obrazu prawdopodobieństwa
- Faza obliczeniowa CamShift
- Wyznaczenie orientacji obiektu
- Generacja poleceń dla systemu operacyjnego

Należy oczywiście zaznaczyć, że w związku z bardzo elastyczną budową programu jest to tylko jedno z możliwych rozwiązań. Dzięki uniwersalności programu można w zupełnie inny sposób dokonywać detekcji i śledzenia dłoni, niekoniecznie w oparciu o model skóry HSV i algorytm CamShift.

Program jest zbudowany na zasadzie ładowanych dynamicznie bibliotek DLL (modułów). Każdy moduł (biblioteka DLL) musi określić jaki moduł powinien go poprzedzać i jaki być po nim. Dzięki temu program nie musi narzucać z góry pewnego toku postępowania, jedynie określa pewną kolejność wynikającą z zastosowanych modułów.

3.1 Definiowanie modelu skóry.

3.1.1 Kalibracja

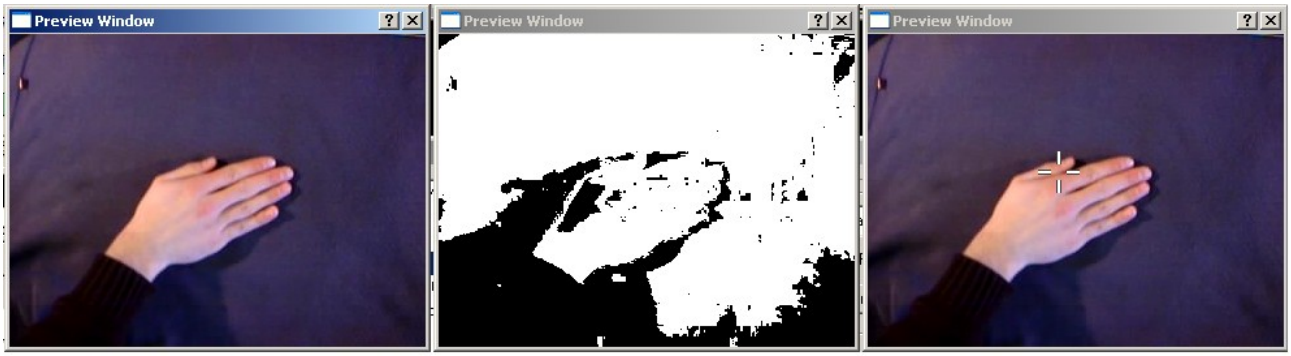
Pierwszym krokiem śledzenia dłoni jest prawidłowe zdefiniowanie modelu skóry. W naszym przypadku model skóry jest określony przez wartości kanałów dla poszczególnych pikseli. Stosujemy system HSV ze względu na oddzielenie barw (kanał H) od jasności i nasycenia. Doświadczenia poprzednich projektów pokazują, że skóra ludzka mieści się w przedziale 10-35 wartości Hue, 20-255 V (jasność) oraz powyżej 30 S (nasycenie - saturation). Jednak w naszym programie wartość minimalna nasycenia wynosi z reguły 0.

Jest oczywiste, że przy różnych warunkach oświetleniowych, lub nawet identycznych ale przy różnych urządzeniach akwizycji obrazu, wartości progowe ulegną zmianie, nieraz na tyle dużej, że program nie będzie w stanie znaleźć dłoni. Dlatego tak ważna jest kalibracja, która skoryguje nastawienia zakresów HSV. Po kalibracji wyliczany jest histogram zaznaczonego obiektu oraz wartość Hue dla której histogram przyjmuje największą wartość (czyli powtarza się najczęściej). Histogram i wartość maksymalna Hue są w dalszych częściach algorytmu przy obliczaniu obrazu prawdopodobieństwa przynależności piksela do obiektu (dłoni).

Kalibrację należy przeprowadzić uważnie, tak aby zaznaczyć jak największy możliwy obszar śledzonego obiektu, ale żeby nie zaznaczyć przypadkiem tła. Poprawne wykonanie kalibracji umożliwi dokładne śledzenie obiektu, a to czy kalibracja została wykonana poprawnie najlepiej widać na podglądzie obrazu prawdopodobieństwa. Przykłady dobrze i źle wykonanej kalibracji znajdują się poniżej:

Ilustracja 1: Przykład poprawnej kalibracji (środkowe okno)

Widać, że parametry są poprawnie dobrane, na obrazie prawdopodobieństwa obiekt jest bardzo wyraźny a kursor wskazuje faktycznie na dłoń (pewna niedokładność wynika z faktu, że palce są rozłożone). Na kolejnym obrazie widać przykład źle dobranych parametrów.



Ilustracja 2: Przykład źle wykonanej kalibracji

Tutaj widać przykład złego zaznaczenia obiektu (zaznaczono również fragment tła). Piksele tła są teraz widziane jako obiekt, co uniemożliwia śledzenie dłoni, mimo że jest ona jakoś widoczna, to z punktu widzenia liczenia momentów nie ma większego znaczenia co się objawia tym, że kursor nie podąża za dłonią, stojąc w jednym miejscu. Tak więc uważna kalibracja jest podstawą poprawnego działania całego programu.

3.2 Tworzenie obrazu prawdopodobieństwa

3.2.1 Informacje wstępne; obliczanie prawdopodobieństw

Program posiada moduł obliczania prawdopodobieństwa tylko dla regionu w którym znajduje się obiekt. Przykładową wizualizację tablicy prawdopodobieństw widać na rysunku poniżej.

Ilustracja 3: Przykładowa wizualizacja tablicy prawdopodobieństwa

Jasne punkty oznaczają duże prawdopodobieństwo czyli przynależność piksela do obiektu, czarne przynależność do tła (prawdopodobieństwo równe 0). Obliczone prawdopodobieństwa są przechowywane w tablicy jednowymiarowej, a wartości prawdopodobieństw mogą być większe od jedności (zastosowaliśmy metodę zaproponowaną przez autorów projektu z ubiegłego roku).

Pierwszym krokiem wykrycia obiektu jest binaryzacja. Na podstawie analizy kanałów HSV do tablicy prawdopodobieństw wpisywana jest wartość „1" jeśli piksel mieści się w ustalonych progach dla kanałów H, S i V, w przeciwnym wypadku do tablicy wpisywana jest liczba „0". Jeżeli została przeprowadzona kalibracja to dodatkowo do tej wartości dodaje się wartość histogramu dla składowej H piksela podzieloną przez maksymalną ilość pikseli o tej samej wartości H, wyraża się to wzorem:

$$P = \frac{H(L(x,y))}{H(H_{\max})}$$

Wzór 1:

gdzie H(L(x,y)) - wartość histogramu dla składowej H piksela

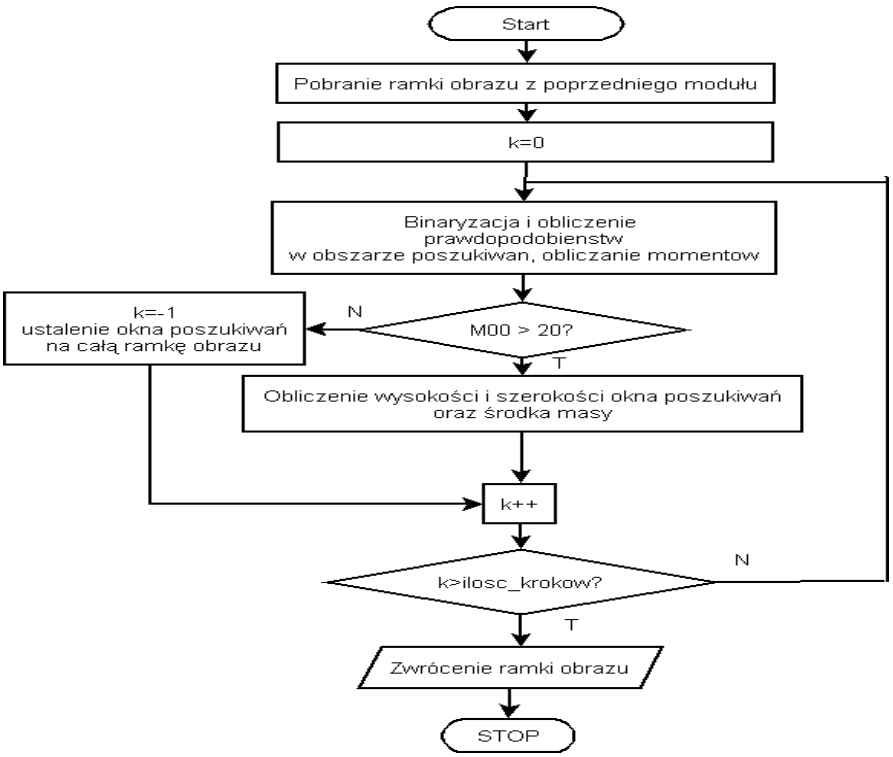
H(Hmax) - maksymalną ilość pikseli o tej samej wartości H

Przez taki sposób obliczania prawdopodobieństw jest możliwe że będą one niekiedy większe od jedynki, jednak na wizualizacji nie widać tego, ze względu na sposób przedstawiania prawdopodobieństwa:

$$R,G,B(x,y) = P(x,y)*255;$$

w przypadku wykroczenia poza 255, wartość składowych piksela jest obcinana do 255 (jako, że obraz jest reprezentowany w 8 bitach na kanał).

Schemat algorytmu przygotowanego przez nas modułu znajduje się poniżej:



Ilustracja 4: Schemat blokowy algorytmu obliczania prawdopodobieństwa przynależnośći piksela do obiektu

Po wyznaczeniu tablicy prawdopodobieństw następuje obliczenie momentów zerowego, pierwszego, oraz drugiego rzędu ze wzorów:

$M_{00} = \sum_{x \in O} \sum_{y \in O} P(x,y)$ - moment zerowego rzędu

Wzór 2:

$M_{10} = \sum_{x \in O} \sum_{y \in O} xP(x,y)$ - moment pierwszego rzędu dla współrzędnej x

Wzór 3:

$M_{01} = \sum_{x \in O} \sum_{y \in O} yP(x,y)$ - moment pierwszego rzędu dla współrzędnej y

Wzór 4:

$M_{20} = \sum_{x \in O} \sum_{y \in O} x^2 P(x,y)$ - moment drugiego rzędu dla x

Wzór 5:

$M_{02} = \sum_{x \in O} \sum_{y \in O} y^2 P(x,y)$ - moment drugiego rzędu dla y

Wzór 6:

$M_{11} = \sum_{x \in O} \sum_{y \in O} xyP(x,y)$ - moment drugiego rzędu dla xy

Wzór 7:

gdzie x,y to współrzędne piksela, O – obszar poszukiwań, P(x,y) – prawdopodobieństwo przynależności piksela do obiektu.

Dla pierwszej ramki analizowana jest cały obszar później tylko obszar poszukiwań. W przypadku wyjscia obiektu poza obszar poszukiwan (np. Zbyt szybkie poruszenie dlonią) nastąpi analiza całej ramki i ponowne obliczenie obszaru poszukiwań.

Po obliczeniu momentów następuje obliczenie środka masy obiektu i nowego obszaru poszukiwań. Wzory na kolejne współrzędne:

$x_c = \frac{M_{10}}{M_{00}}$ $y_c = \frac{M_{01}}{M_{00}}$ oraz na nowy rozmiar okna poszukiwań: $s = \sqrt{\frac{M_{00}}{Max\_Val}}$

Wzór 8: *Wzór 9:* *Wzór 10:*

Po obliczeniu tych wartości następuje kolejna iteracja, ale dla innego okna poszukiwań o szerokości 2*s i wysokości 2.4*s (środkiem tego prostokąta są współrzędne xc i yc).

3.3 Śledzenie dłoni – algorytm Camshift

Następnym krokiem jest wyznaczenie wielkości oraz orientacji obietku. Poniżej znajdują się wzory na wysokość oraz szerokość obiektu.

$h = 2 \cdot \sqrt{\frac{(a+c) + \sqrt{b^2 + (a-c)^2}}{2}}$ $w = 2 \cdot \sqrt{\frac{(a+c) - \sqrt{b^2 + (a-c)^2}}{2}}$

Wzór 11: wzór na wysokość obiektu *Wzór 12: wzór na szerekość obiektu*

przy czym:

$a = \frac{M_{20}}{M_{00}} - x_c^2$ $b = 2 \cdot \frac{M_{11}}{M_{00}} - x_c \cdot y_c$ $c = \frac{M_{02}}{M_{00}} - y_c^2$

Wzór 13: *Wzór 14:* *Wzór 15:*

Również jest wyliczany ponownie środek masy.

Kolejnym krokiem jest wyznaczenie orientacji obiektu. W tym celu stosujemy wzór z dokumentacji algorytmu Camshift:

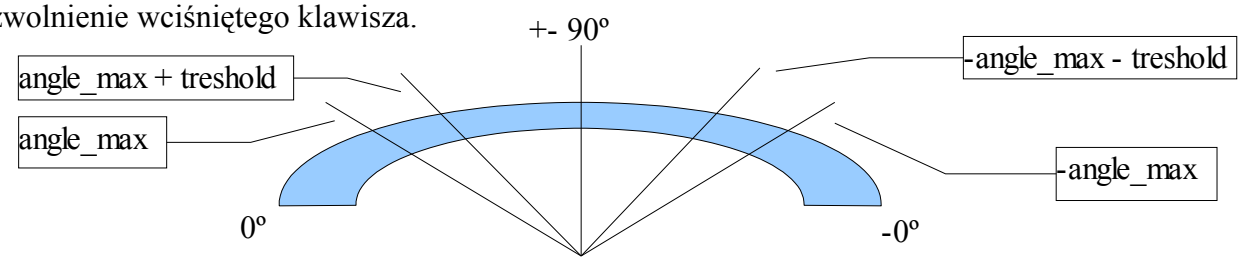
$$\theta = \frac{arctg\left(\frac{2\cdot\left(\frac{M_{11}}{M_{00}} - x_c y_c\right)}{\left(\frac{M_{20}}{M_{00}} - x_c^{\,2}\right) - \left(\frac{M_{02}}{M_{00}} - y_c^{\,2}\right)}\right)}{2}$$

Wzór 16: Wzór na orientację obiektu

Wzór ten daje bardzo dobre wyniki, co pozwala na implementację gestów.

3.4 Rozpoznawanie gestów

Nasz program potrafi rozpoznać kliknięcie prawym oraz lewym klawiszem na podstawie analizy kąta obrotu dłoni. Jeżeli wartość kąta przekroczy pewną, ustalaną w programie wartość, to jest to interpretowane jako wcisniecie klawisza myszy (to którego jest uzależnione od znaku kąta – wartości ujemne oznaczają prawy klawisz, dodatnie lewy). Gdy dłoń powraca do położenia pionowego wówczas, po przekroczeniu wartości granicznej + wartość nieczułości następuje zwolnienie wciśniętego klawisza.



Kąty pomiędzy zerem a wartością angle_max są interpretowane jako lewy klawisz myszki, a pomiędzy -0 stopni a -angle_max jako prawy klawisz. Wartość treshold jest dodawana (odejmowana), aby zabezpieczyć się przed przypadkowym puszczeniem przycisku myszy, gdy nastąpi mała zmiana kąta.

Dodatkowo, program jest w stanie wykryć zmianę wielkości dłoni i zinterpretować to jako kliknięcie trzecim klawiszem myszki. Wykrycie zmiany rozmiarów następuje przez porównanie wielkości początkowej dłoni (mierzonej dla pierwszej ramki obrazu i każdorazowo po kalibracji). Jeżeli wartość aktualna wysokości dłoni będzie mniejsza o pewną wartość (ustalana w programie) to zostanie to zinterpretowane jako wciśnięcie trzeciego klawisza myszki. Jednak ze względu na to, że podczas zaciskania dłoni gwałtowanie zmieniają się wartości kąta orientacji obiektu (wysokość staje się szerokością) używanie trzeciego klawisza myszy jest dość problematyczne. Należy uważnie dobrać parametry sterujące wykryciem zaciśnięcia myszki tak, aby zapobiec „kliknięciom" wynikającym ze zmiany orientacji. Dlatego też, emulacja trzeciego klawisza myszy jest domyślnie wyłączona.

3.5 Generacja poleceń dla systemu operacyjnego

Na podstawie rozpoznanych gestów i wyznaczonego położenia dłoni do systemu wysyłane są odpowiednie polecenia, sterujące wskaźnikiem myszy oraz emulujące wciśnięcia i zwolnienia klawiszy myszy. Aby uruchomić sterowanie wskaźnikiem należy wciśnać i puścić klaiwsz F12. Ponowne użycie tego klawisza spowoduje wyłączenie sterowania wskaźnikiem. Są możliwe dwa typy sterowania: pozycyjny oraz prędkościowy.

Sterowanie pozycyjne polega na przeliczeniu pozycji obiektu na klatce obrazu na położenie wskaźnika na ekranie komputera. Aby móc to zrobić najpierw jest sprawdzana aktualna rozdzielczość ekranu a następnie obliczane dwa współczynniki:

$$Xaspect = \frac{szer_{ekranu}}{szer_{filmu}}$$
$$Yaspect = \frac{wysokosc_{ekranu}}{wysokosc_{filmu}}$$

Wzór 17:

Następnie pozycja x i y obiektu w obrazie jest mnożona przez odpowiedni współczynnik i tak otrzymana pozycja wskaźnika jest wysyłana wprost do systemu, ustawiając kursor myszy w odpowiednim miejscu.

Sterowanie prędkościowe działa nieco inaczej. Jeżeli środek obiektu znajduje się w obszarze „neutralnym", w centralnej części ramki obrazu, to wskaźnik się nie porusza. Pryz wyjsciu obiektu poza ograniczenia następuje przemieszczenie kursora myszy z prędkością proporcjonalną do odległości obiektu od środka ekranu w zadanym kierunku.

3.6 Proponowane modyfikacje

Podczas realizacji projektu zawuażyliśmy, że pojawiają się problemy z wykonywaniem kliknięć myszką. Przyczyną jest to, ze podczas obrotu dłoni w momencie próby wykonywania kliknięcia przesuwa się również środek ciężkości, co z kolei powodowało duże przemieszczenie wskaźnika myszki na ekranie. Utrudniało to dość mocno używanie przycisków myszki. Aby nieco zminimalizować wpływ tego zjawiska postanowiliśmy dokonać pewnej modyfikacji, polegającej na przesunięciu środka ciężkości na początek obiektu. Zostało to zrealizowane w dość prosty sposób:

$$dx = sign(u) \cdot \frac{\cos(u) \cdot h}{2}$$
$$dy = \left| \left(\frac{\sin(u) \cdot h}{2} \right) \right|$$

Wzór 18: Wzór na przesunięcie wzdłuż osi x i y

gdzie:

u – kąt obrotu;

sign(u) – znak kąta obrotu;

h – wysokość obiektu;

Następnie od współrzędnej yc (y środka masy) jest odejmowana wartość dy, a od współrzędnej xc jest dodawana lub odejmowana (zależnie od znaku) wartość dx. Dzięki temu zabiegowi przesunięcie wskaźnika podczas obortu dłoni nadal występuje, ale jest dużo mniejsze. Warunkiem aby ta metoda działała sprawnie jest dobre skalibrowanie programu. W naszym projekcie należy właczyć opcję „Move center" w module Posdetect.

4. WNIOSKI

Nasz program przetestowaliśmy na labolatorium zestawiając następujące stanowisko:

- komputer klasy PC z systemem Windows XP
- kamera podłączona do karty telewizyjnej (framegrabbera)

Dodatkowo na komputerze było zainstalowanych kilka programów. Przedewszystkim VirtualDub aby móc wybrać źródło sygnału (S-Video, Video Composite lub Tuner) oraz parametry takie jak ilość ramek na sekundę (najlepiej wybrać 25). Jeszcze jednym programem którego używaliśmy był CamStudio do nagrywania zawartości pulpitu i kursora.

Po tych wstępnych przygotowaniach wykonaliśmy kilka testów:

- sterowanie ręką i obsługa prostego programu napisanego przez nas (Kalibrator i Tester)
- test sterowania prędkościowego na wyżej opisanych programach
- próba sterowania systemem Windows (otwieranie okien na pulpicie, przesuwanie okienek)
- test działania 3-go klawisza (w przeglądarce internetowej)
- sprawdzenie czy jest możliwe sterowanie komputerem ręką np. w rękawiczce

Programy Kalibrator i Tester mierzą czas pomiędzy kolejnymi kliknięciami na przyciskach, dodatkowo Kalibrator oblicza parametry a i b, na podstawie których można obliczyć czas wymagany na klikniecie oraz ocenić skuteczność sterowania.

Liczby a i b są parametrami równania opisującego czas poruszania się Fitt`s Law:

$$MT = a + b\log_2\left(\frac{2A}{W} + 1\right)$$

Wzór 19: Wzór na szacowany czas ruchu

przyyjmując jako A (amplitudę – odległości) kolejno 180, 180, 270 i szerokość 240, 120, 60 otrzymaliśmy kolejne punkty na osi rzędnych: 1,3219, 2, 3,32. Następnie mierząc czasy kliknięc kolejnych przycisków można było metodą regresji liniowej obliczyć współczynniki a i b.

Współczynnik 1/b określa wydajność sterowania.

Oto zestawienie czasów uzyskanych w programie Kalibrator (czasy mierzone są między puszczeniem przycisku Start a puszczeniem przycisku Btn 1-3 i w takiej też kolejności są w tabelce)

Nazwa pliku video	Metoda Sterowania	Czasy [ms]	a	b
test_speed_prob_01.avi	Prędkościowe	3093, 3125, 6390	0,30244	1,76113
test_speed_prob_02.avi	Prędkościowe	2203, 5484, 9625	-2,27212	3,63168
test_vel_01.avi	Prędkościowe	3328, 5046, 4906	2,95108	0,66629
test_vel_02.avi	Prędkościowe	2688, 3625, 3109	2,85694	0,12811
test_vel_03.avi	Prędkościowe	2496, 3313, 3141	2,37176	0.27209
test_posl_mc_01.avi	Pozycyjne – wprost +MC	2594, 3484, 3906	1,97748	0,60982
test_posl_mc_02.avi	Pozycyjne – wprost + MC	2234, 2781, 4672	0,46516	1,248
test_posl_mc_03.avi	Pozycyjne – wprost +MC	1735, 3500, 6640	-1,44958	2,44192
test_posl_01.avi	Pozycyjne – wprost	15953, 4531, 8781	15,63139	-2,65346
test_posl_02.avi	Pozycyjne – wprost	10672, 8750, 7266	12,49125	-1,62342
test_posl_03.avi	Pozycyjne – wprost	5391, 4938, 8875	2,16806	1,91151

Komentarz: Steowanie prędkościowe oznacze sterowanie gdzie prędkość przesuwu kursora jest uzależniona od oddalenia od środka ekranu, sterowanie Pozycyjne – wprost oznacza, że pozycja wskaźnika myszy jest ustanalana wprost z pozycji dłoni na obrazie z kamery. Sterowanie z MC (Move Center) oznacza, że była włączona ta opcja – czyli przesunięcie środka ciężkości w dół

(patrz rozdz. 3.6).

Dla porównania czasy uzyskane w programie Kalibrator dla zwykłej myszki (Logitech Cordless Optical Mouse): 140, 181, 341 (ms). Parametry a=-0,07070; b=0,17507.

Wyniki uzyskane w programie Tester. Ostatnia kolumna prezentuje czasy wykonania kliknięć pomiędzy centralnym przyciskiem, a przyciskami znajdującymi się na okręgu:

Nazwa pliku video	Typ Sterowania	Czasy [s]
test_b_vel_01.avi	Prędkościowe	4,391; 6,672; 4,891; 4,219; 3,453; 3,578; 4,094; 4,766; 3,328
test_b_vel_02.avi	Prędkościowe	6,438; 3,578; 5,204; 6,531; 4,985; 3,359; 3,078; 3,609; 6,219
test_b_vel_03.avi	Prędkościowe	3,641; 4,062; 4,344; 3,391; 4,937; 4,391; 4,125; 5,172; 4,063
ver_2_vel_test_01.avi	Prędkościowe (w rękawiczce)	5,422; 4,422; 3,687; 3,734; 3,657; 4,015; 5,395; 2,656; 2,344

Sterowania pozycyjnego wprost nie wykonaliśmy ze względu na dużą trudność z klikaniem w przyciski.

Dodatkowo wykonaliśmy kilka testów aby ocenić, czy możliwe jest praktyczne sterowanie systemem operacyjnym:

Nazwa pliku video	Akcje wykonywane
sterowanie_systemem.avi	Ogólne akcje – otwarcie katalogu z pulpitu, przesuwanie oknem, zaznaczenei kilku obiektów w katalogu
sterowanie_systemem_3klawisz.avi	Test działania trzeciego klawisza w programie Internet Explorer
sterowanie_systemem_menu_kontekstowe.avi	Wywołanie menu kontekstowego dla katalogu, pokazanie okna jego właściwości
poruszanie_kursorem.avi	Poruszanie wskaźnikiem, bez żadnych kliknięć

Analizując powyższe tabelki można dojść do następujących wniosków:

- sterowanie prędkościowe wydaje się bardziej precyzyjne niż sterowanie wprost pozycją. Wynika to ze strefy nieczułości, dzięki temu przesunięcia wskaźnika wynikające z obrotu dłoni mogą zostać zignorowane, co pozwala na klikanie małych obiektach (m.in. Dlatego właśnie systemem sterowaliśmy tylko prędkościowo)
- sterowanie wprost pozycją z obrazu video jest bardziej intuicyjne, oraz pozwala na szybsze przesunięcie wskaźnika myszy. Niestety są dość duże problemy z klikaniem, gdyż wskaźnik wykonuje znaczny ruch przy obrocie dłoni. Problem ten nieco niweluje przesunięcie środka ciężkości, ale mimo to jest problemem kliknięcie na małe obiekty
- porównując uzyskane czasy z przeciętną myszką można zauważyć jak wielką ma ona przewagę (różnica jest aż o jeden rząd wielkości!!). O ile w sterowaniu dłonią samo nakierowanie wskaźnika można wykonać względnie szybko, o tyle nie ma mowy wykonania kliknięcia w porównywalnym tempie.
- Sterowanie systemem jest możliwe, choć nieco uciążliwe i raczej z zastosowaniem metody prędkościowej (większa dokładność).
- Przy poprawnym ustawieniu parametrów programu również trzeci klawisz będzie działać poprawnie (co widać na przykładzie filmu z Internet Explorerem – strona może być przewijana dłonią)

Podsumowując, sterowanie komputerem przy użyciu dłoni jest możliwe. Nasz projekt to potwierdza. Jednak zastąpienie myszki w codziennym użytkowaniu jest raczej nierealne, ze względu na zbyt małą dokładność przy sterowaniu informacją z sygnału video. Sterowanie dłonią może natomiast mieć zastosowanie w grach, w których gracz ma za zadanie wykonać jakąś czynność czy gest za pomocą dłoni (lub ręki czy głowy). W takich programach nie jest wymagana bardzo duża dokładność, co pozwala na zastosowanie przedstawionej przez nas metody.

5. BUDOWA PROGRAMU (Opis API bibliotek)

5.1 Informacje ogólne

Ze względu na bardzo dużą elastyczność naszego programu oraz możliwość dodawania w zasadzie dowolnych modułów przetwarzania obrazu, niezbędne jest omówienie chociaż podstaw działania programu oraz zasady budowy modułów i komunikacji między nimi. Program został stworzony w środowisku Visual Studio C++ 2003, i najlepiej korzystać z niego przy tworzeniu nowych modułów. Moduły są ładowanymi dynamicznie bibliotekami DLL .

Bardzo ważną cechą naszego projektu jest to, że ramka obrazu ma format BGR0 (ze względu na zastosowanie biblioteki Qt), i jest tablicą jednowymiarową o wymiarze ***height*width*depth*** , gdzie height jest wysokością obrazu, width jego szerokością, a depth ilością kanałów (u nas 4). Wartości kanałów są przechowywane jako typ char bez znaku (8 bitów). Opis sposobu przechowywania obrazu również jest zawarty w rozdziale 5.3.

Projekt pisaliśmy tak, aby możliwe było generowanie dokumentacji kodu programem doxygen. Odpowiedni plik konfiguracyjny tego programu znajduje się w katalogu doc. Została również dołączona dokumentacja wygenerowana tymże programem. Zawiera ona bardzo dużo przydatnych inforamcji z opisami zmiennych włącznie. Dodatkowo schematycznie rozrysowuje połączenia między klasami.

5.2 Funkcje i metody (dziedzicznone po module_base)

Moduły są bibliotekami dll, które posiadają funkcję eksportujacą obiekty klas reprezentujących dane moduły. Każda biblioteka dll stanowiąca nowy moduł musi mieć zaimplementowaną funkcję w postaci:

```
extern "C" __declspec(dllexport) moduleBase * export_module();
```

która powinna zwracać wskaźnik do utworzonego w pamięci gotowego do użycia modułu, który dziedziczy po klasie *moduleBase*. Klasa moduleBase zawiera deklaracje metod pozwalających na komunikację się z programem zarządzającym modułami. Część z tych metod musi być zaimplementowana w nowo tworzonym module, część jest nieobowiązkowa (zależna od potrzeb modułu), a część wogóle nie jest istotna z punktu widzenia tworzenia nowych modułów, ponieważ jest wykorzystywana jedynie przez system zarządzający modułami niejako *w tle,* poprzez wykorzystanie odpowiednich makrodefinicji. Poniżej zostały opisane najważniejsze metody, które są niezbędne, bądź przydatne dla programisty tworzącego nowe rozszerzenia.

5.2.1 Metody klasy moduleBase

---

int *init(PropertyMgr * pm)*

Metoda inicjalizacji modułu, podczas której mogą być ustalane wartości domyślne niektórych zmiennych oraz tworzone są niezbędne dla działania modułu struktury. Wywoływana każdorazowo przy uruchamianiu symulacji. Podczas inicjalizacji może zostać ustawiony tzw. Menadżer Zmiennych (PropertyMgr – patrz rozdz. 5.2.2). Metoda powinna zwrócić jeden z kodów stanu.

---

void *free()*

Metoda wywoływana przy zakończeniu przetwarzania obrazu, pozwala na zwolnienie niepotrzebnych już zasobów używanych przez moduł. Metoda jest wywoływana każdorazowo przy wyłączeniu przetwarzania obrazu z kamery, bądź orzy zakończeniu odtwarzania pliku wideo. Dodatkowo metoda wywoływana jest przy zwalnianiu modułów (przy zakończeniu programu, bądź przy przeładowywaniu modułów).

---

*proc_data * process_frame(proc_data * prev_frame, int * result)*

Najważniejsza metoda, wywoływana dla każdego modułu dla każdej przetwarzanej ramki obrazu. To w tej metodzie wykonuje się wszystkie elementy przetwarzania obrazu i detekcji obiektów. Metoda otrzymuje jako argument strukturę proc_data i po przetworzeniu powinna zwrócić również nową, uzupełnioną w odpowiednie dane strukturę, która zostanie przekazana kolejnemu modułowi w torze przetwarzania. Dzięki temu możliwe jest przetwarzanie obrazu w kolejnych modułach zachowując maksymalną elastyczność. Zmienna result powinna po wyjściu z metody mieć wartość kodu statusu (ST_OK, badź status błędu w przypadku niepowodzenia). Opis struktury proc_data oraz kody statusów znajdują się dalej, w rozdz. 5.3 oraz 5.4

---

void *mouse_select(int sx, int sy, int sw, int sh)*

Implementacja tej metody nie jest obligatoryjna, i w zasadzie jest ona zaimplementowana tylko na potrzeby modułów które wymagają kalibracji (zaznaczenia jakiegoś obszaru na obrazie). Wywoływana jest, gdy na którymkolwiek oknie podglądu użytkownik zaznaczy jakis obszar. Jako parametry przekazywane są współrzędne lewego

górnego wierzchołka prostokąta oraz jego wysokość i szerokość. Metoda ta jest wywoływana dla wszystkich modułów, niezależnie od tego, na którym oknie podglądu został zaznaczony fragment obrazu.

---

int *input_type()*

Metoda zwracająca typ wejścia modułu. Opisy typów wejść/wyjść w rozdz. 5.4

---

int *output_type()*

Metoda zwracająca typ wyjścia z modułu. Opisy typów wejśc/wyjść w rozdz. 5.4

---

const char ** get_module_description()*

Metoda zwracająca opis modułu, który jest następnie wyświetlany w oknie głównym programu.

Konstruktor

W konstruktorze można zarejestrować zmienne do konfiguracji modułu poprzez makro

#define REG_PARAM(TYPE, NAME, DESC, DEF_VAL)

gdzie TYPE jest rodzajem parametru (opis w rozdziale 5.4), NAME nazwą rejestrowanego parametru, DESC opisem, widocznym w oknie konfiguracji, DEF_VAL wartością domyślną (w przypadku używania typów RANGE należy dodatkowo zamiast określenia wprost wartości domyślnej, określić wartość i zakresy przez użyć konstruktora klasy *int_range* lub *float_range* z parametrami (value, min, max) - patrz rozdz. 5.4). Makro to wywołuje metody modułu moduleBase, zapisując wszystkie parametry, które następnie poprzez odpowiedni interfejs są udostępniane programowi zarządzającemu modułami.

5.2.2 Komunikacja modułów – klasa PropertyMgr

Moduły mogą komunikować się wzajemnie, jeżeli tylko podczas implemetacji danych modułów zaplanujemy taką możliwość, oraz gdy w torze przetwarzania znajdą się dwa moduły, które mogą komunikować się między sobą. Do tego celu została napisana klasa PropertyMgr. Udostępnia ona następujące metody:

```
int register_property( std::string name, void * value );
```

metoda ta służy do zarejestrowania zmiennej/parametru w naszym menadżerze. *name* to nazwa parametru, która jednoznacznie identyfikuje parametr.

```
void * get_property( std::string name );
```

metoda ta służy do wyszukiwania zarejestrowanego wcześniej parametru. Zwraca wskaźnik na ten parametr, bądź *NULL* w przypadku niepowodzenia. *name* to nazwa szukanego parametru.

Menadżera można wykorzystać w sytuacji, gdy chcemy w jednym z modułów wykorzystać pewne dane, lub zmienić parametry innego modułu, które nie są dostępne globalnie w programie. Aby móc go używać należy go zainicjalizować makrem *USE_PROPERTY_MGR(pm)* w metodzie *init* dziedziczonej po *moduleBase.*

5.3 Typy i struktury

W programie zostało zdefiniowanych przez nas kilka nowych typów danych (struktur). Są one niezbędne do poprawnego działania programu. Taka ilość struktur jest niezbędna ze względu na uniwersalność projektu.

Struktura przechowująca informacje o ramce obrazu

```
struct frame_data
{
	unsigned int  width;		/// szerokosc obrazu w pixelach
	unsigned int  height;		/// wysokosc obrazu w pixelach
	unsigned int  depth;		/// Ilosc bajtow na pixel - dla RGB 3
	/** Tablica wartosci pikseli, jedno wymiarowowa o dlugosci
	width*height*depth
	    Jeden piksel to 4 kolejne elementy w kolejnoœci G B R 0
	    P1          |P2         |P3         .....
	    [0][1][2][3][4][5][6][7][8][0][A][B]..... itd
	    |G  B  R  0 |G  B  R  0 |G  B  R  0|
	*/

	unsigned char * bits;		/// wskaznik do pixeli obrazu
};
```

Struktura do opisu położenia obiektu

```
struct pd_data
{
	int x,y;		/// pozycja myszy
	int gesture;		/// gest (definicje powyzej)
	float angle;		/// kat
};
```

Struktura opisuje dane które są zwracane przez moduły, lub przekazywane na wejście modułów.

```
struct proc_data
{
	frame_data  * input_frame;	/// ramka obrazu z modulu wejsciowego
	frame_data  * frame;		/// ramka obrazu odpowiadajaca biezacemu
					/// modulowi
	pd_data     * position;		/// informacje o pozycji kursowa, gestach
	float       * prob;		/// obraz prawdopodobienstwa
	float         max_prob;		/// maksymalna wartosc prawdopodobienstwa
	float       * moments;		/// tablica momentów
};
```

Struktura przechowuje informacje o histogramie

```
struct hist_data
{
	int   * hist_vals;/// tablica wartosci histogramu
	int   h_size;	/// wielkość tablicy
	int   histMaxVal; /// maksymalna wartość w histogramie
	float maxV,
	      minV,
	      minS;	/// kolejno minimalne i maksymalna jasność i minimalne
			/// nasycenie
};
```

5.4 Zdefiniowane nazwy

W programie zostało zdefiniowanych kilka nazw dla kodów statusów, nazw wejść i wyjść itp.

Gesty:

```
#define GESTURE_NULL			0x00 /// brak wciśniętego klawisza myszy
#define GESTURE_LMBCLICK		0x01 /// wciśnięty lewy klawisz myszki
#define GESTURE_RMBCLICK		0x02 /// wciśnięty prawy klawisz myszki
#define GESTURE_LMBDBCLICK		0x03 /// podwójne kliknięcie lewym klawiszem
```

Typy wejść i wyjść

```
#define MT_NONE                 0x00
#define MT_FRAME                0x01
#define MT_PROBDATA             0x02
#define MT_POSGEST              0x03
```

Interpretację tych typów przedstawia tabelka:

Typ wejścia(wyjścia)	Ustawiony na wejściu	Ustawiony na wyjściu
MT_NONE	Moduł jest pierwszym w torze.	Moduł jest ostatnim w torze.
MT_FRAME	Poprzedzający moduł powinien być modułem akwizycji, bądź przetwarzania obrazu.	Moduł albo pobiera ramkę z urządzenia (pliku) bądź też przetwarza ją (np. rozjaśnia)
MT_PROBDATA	Poprzedzający moduł powinien obliczać prawdopodobieństwa przynależności pikseli do tła.	Moduł oblicza prawdopodobieństwa
MT_POSGEST	Poprzedzający moduł powinien ustalić położenie i wykryć gesty	Moduł oblicza położenie obiektu i jego rotację oraz wykrywa gesty

---

Kody stanu

```
#define ST_OK                   0x00
#define ST_ALLOC_ERROR          0x01
#define ST_OPEN_ERROR           0x02
#define ST_WRITE_ERROR          0x03

#define ST_DEVICE_NOT_FOUND     0x04
#define ST_FRAME_ERROR          0x05
#define ST_ALREADY_EXISTS       0x06

#define ST_MISSING_DLL          0x07
#define ST_UNKNOWN_MODULE       0x08
#define ST_EXPORT_ERROR         0x09
#define ST_OUT_OF_RANGE         0x0A
#define ST_WINDOW_CLOSED        0x0B
```

Parametry modułów

```
#define PT_INT          0x01
#define PT_LONG         0x02
#define PT_BOOL         0x03
#define PT_FLOAT        0x04
#define PT_DOUBLE       0x05
#define PT_STRING       0x06
#define PT_FILENAME     0x07
#define PT_INT_RANGE    0x08
#define PT_FLOAT_RANGE  0x09
#define PT_PREVIEW      0x0A
```

Dodatkowe typy parametrów

`class preview`

Typ parametru odpowiadający wartości PT_PREVIEW. Parametr ten jest prezentowany w oknie opcji jako wartość do tylko do odczytu, odświeżana w trakcie przetwarzania z wybraną częstotliwością. (można wykorzystać np. do śledzenia wartości kąta, bądź innej zmiennej)

`class int_range`

Typ parametru odpowiadający wartości PT_INT_RANGE. Parametr ten jest prezentowany w oknie opcji jako suwak, z mozliwością łatwej zmiany wartości parametru. Takie rozwiązanie wymaga jednak dodatkowych informacji takich jak wartość minimalna oraz maksymalna. Wszystkie te informacje są przechowywane własnie w klasie int_range. W momencie gdy chcemy stworzyć w module parametr typu PT_INT_RANGE, deklarujemy zmienną typu int_range (zamiast int), w inicjalizacji parametru korzystamy z konstruktora klasy:

```
int_range::int_range( int value, int min, int max );
```

W programie modułu możemy korzystać ze zmiennej typu int_range w taki sam sposób jak ze zmiennej typu int, ponieważ definiuje ona odpowiednie operatory.

`class float_range`

Typ parametru odpowiadający wartości PT_FLOAT_RANGE. Parametr jest prezentowany w taki sam sposób jak powyższy PT_INT_RANGE, z tą różnicą, że dotyczy wartości zmienno przcinkowych.

5.5 Omówienie zaimplementowanych modułów

5.5.1 videoacq_opencv.dll

Moduł służy do akwizycji obrazu z pliku lub urządzenia przechwytującego video. W metodzie inicjalizującej (*init*) otwiera plik wejściowy, bądź inicjalizuje urządzenie video. W wypadku błędu (brak pliku, bądź niemozliwe znalezienie urządzenia wideo) zwraca kod błędu. W metodze zwalniającej zasoby (*free*) zamyka plik bądź urządzenie wideo. W metodze przetwarzania ramki obrazu (*process_frame*) odczytuje ramkę obrazu z pliku bądź z urządzenia wideo, kowertuje do odpowiedniego formatu (patrz rozdz. 5.3) a następnie odpowiednio uzupełnia strukturę *proc_data* i zwraca do niej wskaźnik przekazując do kolejnych modułów.

5.5.2 prob_image.dll

Moduł służy do obliczania prawdopodbieństw przynależności pikseli do tła poprzez binaryzację oraz obliczenie prawdopodobieństwa ze wzroru(1). W zwracanej strukturze *proc_data* znajdują się obliczone prawdopodobieństwa (pole *float *prob*)oraz momenty(pole *float *moments*). Opis struktury *proc_data* znajduje się w rozdziale 5.3.

5.5.3 posdetect.dll

Moduł na podstawie obliczonej wcześniej tablicy prawdopodobieństw (wyliczonej w poprzednim

module) wyznacza położenie, orientację oraz rozpoznaje podstawowe gesty(wcisniecie prawego, lewego i środkowego klawisza myszy). Moduł zwraca strukturę *proc_data* zawierającą ramkę obrazu z nałożonym markerem oraz informację o położeniu obiektu (pole *pd_data * position*).

5.5.4 mod_steer.dll

Moduł ma za zadanie wygenerować polecenia dla systemu operacyjnego. Ze struktury *proc_data* która jest jego parametrem (otrzymanym z poprzedniego moduły w torze przetwarzania) pobiera informację o położeniu obiektu i na tej podstawie generuje polecenia. Moduł wykorzystuje funkcje WinAPI do generacji kliknięć myszką, sterowania kursorem myszy, oraz do rejestrowania stanu klawisza uruchamiającego i zatrzymującego sterowanie systemem operacyjnym poprzez nasz program (klawisz F12).

5.5.5 Dodakowe moduły

Wraz z programem zamieszczamy dwa dodatkowe moduły nie mające większego znaczenia w działaniu programu, ale na ich podstawie można w prosty sposób prześledzić działanie modułów, i w prosty sposób zapoznać się z ideą ich działania ponieważ nie są tak skomplikowane jak pozostałe moduły napisane w ramach naszego projektu. Te moduły to *mod_light.dll* oraz *mod_neg.dll.* Oba są modułami które na wejście przyjmują ramkę obrazu i jako wyjście zwracają również ramkę obrazu. Pierwszy z nich służy do rozjaśniania obrazu, a drugi tworzy na wyjściu negatyw obrazu wejściowego. Dla modułu *mod_light.dll* jest dodatkowo stworzony parametr odpowiadający współczynnikowi rozjaśnienia, który może być zmieniany z poziomu programu w opcjach modułu.

5.6 Działanie programu – zarządzanie modułami

Program zarządzający (zwany dalej *programem*) ma za zadanie załadować moduły, umożliwić ich konfigurację, oraz udostepnić możliwość połączenia ich w jeden tor przetwarzania. Po uruchomieniu program przeszukuje katalog *modules* w poszukiwaniu plików *dll.* Każdy plik dll jest ładowany, po czym następuje sprawdzenie, czy dana biblioteka jest właściwym modułem naszego systemu (sprawdzenie następuje poprzez wyszukanie funkcji eksportującej – patrz rozdz. 5.2).

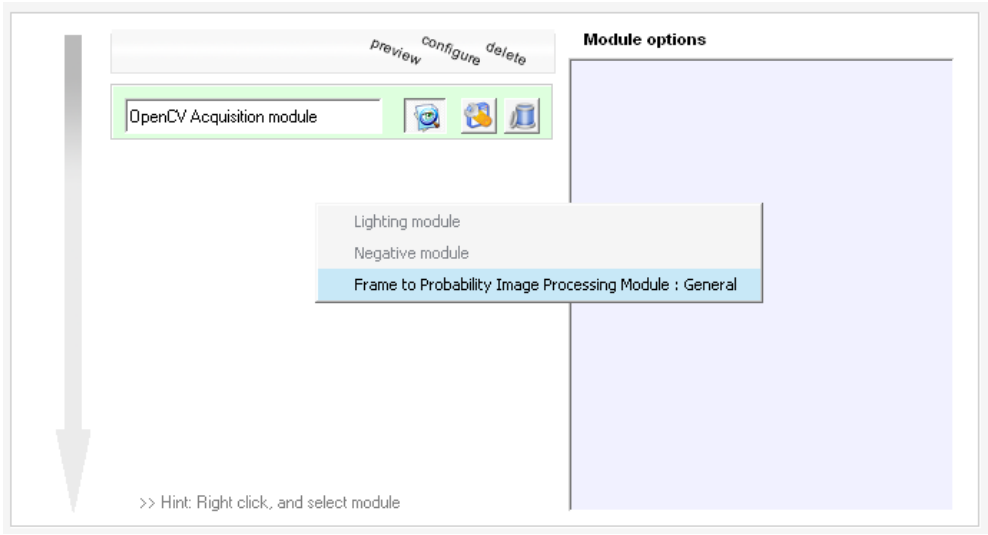


Illustration 5: Wybór właściwego modułu

Po uruchomieniu programu mamy mozliwość stworzenia toru przetwarzania, poprzez wybranie kolejnych modułów. Przy próbie wybrania modułu następuje sprawdzenie całej listy załadowanych modułów, i wyszukaniu tych, które mogą znajdować się na wybranej pozycji toru przetwarzania. (więcej o typach wejścia oraz wyjścia modułów w rozdz. 5.4)

Na rysunku nr 5 mamy przestawione wstawianie nowego modułu do toru przetwarzania. W torze znajduje się już moduł akwizycji wideo (*OpenCV Acquisition*). Przy próbie wyboru kolejnego modułu rozwija nam się lista mozliwych modułów do wybrania – selekcja dokonywana jest na podstawie typu wyjścia ostatniego modułu w torze, oraz typów wjesciowych wszystkich załadowanych modułów. W efekcie na liście znajdują się tylko te moduły, które mają zgodne wejście z modułem poprzedającym je w torze przetwarzania.

Każdy moduł po wstawieniu do toru przetwarzania widoczny jest w oknie w postaci kontrolki zawerającej nazwę modułu, oraz kilka przycisków oznaczonych kolejno: *preview*, *configure*, *delete*. Pierwszy z nich oznacza, czy dla danego modułu ma być otworzone okno podglądu. Dodatkowo, gdy opcja ta jest wybrana kontrolka modułu podswietla się na zielono, w przeciwnym wypadku podświetlenie jest koloru czerwonego. Kolejny przycisk pokazuje w oknie *Module options* dostępne opcje modułu, poprzez wykorzystanie interfejsu zdefiniowanego w klasie *moduleBase.* W zależności od typu parametru pokazuje się odpowiednia kontrolka umożliwająca zmianę jego wartości, bądź tylko śledzenie zmian podczas przetwarzania (patrz ilustracja nr 7). Zmiany parametrów dokonują się bezpośrednio (czyli w momencie wpisania wartości do pola tekstowego parametr modułu zmienia się od razu, nawet podczas przetwarzania obrazu). Ostatni przycisk oznaczony jako *delete* służy do usuwania modułów z toru przetwarzania (uwaga – można usuwać tylko ostatni moduł z toru).

W momencie rozpoczęcia przetwarzania wszystkie moduły w torze zostają zainicjalizowane poprzez wywołanie metody *init* klasy *moduleBase.* Następnie następuje przetwarzanie. Poczynając od pierwszego modułu w torze, struktura *proc_data* jest przekazywana przez wszystkie moduły po kolei.

Po zakończeniu bądź po przerwaniu przetwarzania wszytkie moduły znajdujące się w torze przetwarzania zostają zamykane poprzez wywołanie metody *free* (klasa *moduleBase*).

5.7 Ważniejsze pliki

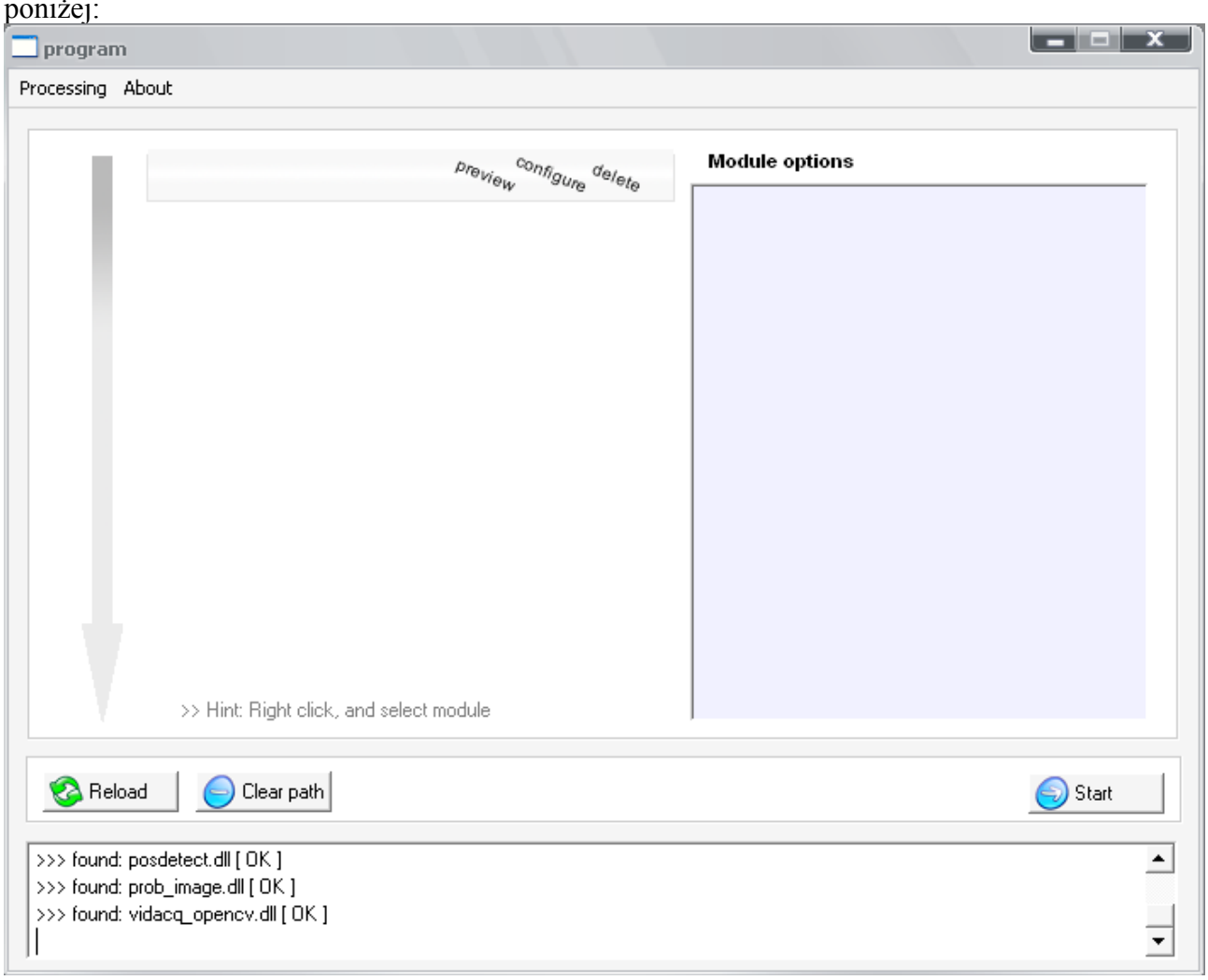
Aby móc napisać jakikolwiek moduł należy do kodu dołączyć kilka plików nagłówkowych:

- *module_base.h* – Zawiera definicję klasy podstawowej modułów, oraz kilka użytecznych makrodefinicji. Dołączenie tego pliku musi wystąpić w pliku nagłówkowym tworzonego modułu.
- *status_codes.h* – Należy go dołączyć do pliku z kodem modułu (cpp). Zawiera definicje kodów stanu wykonania programu;
- *types.h* – Zawiera definicje struktur używanych w programie;

W tych plikach znajdują się również opisy struktur oraz sposób ich użycia.

6. OBSŁUGA PROGRAMU

Program jest stosunkowo prosty w obsłudze. Okno główne programu znajduje się na obrazie poniżej:

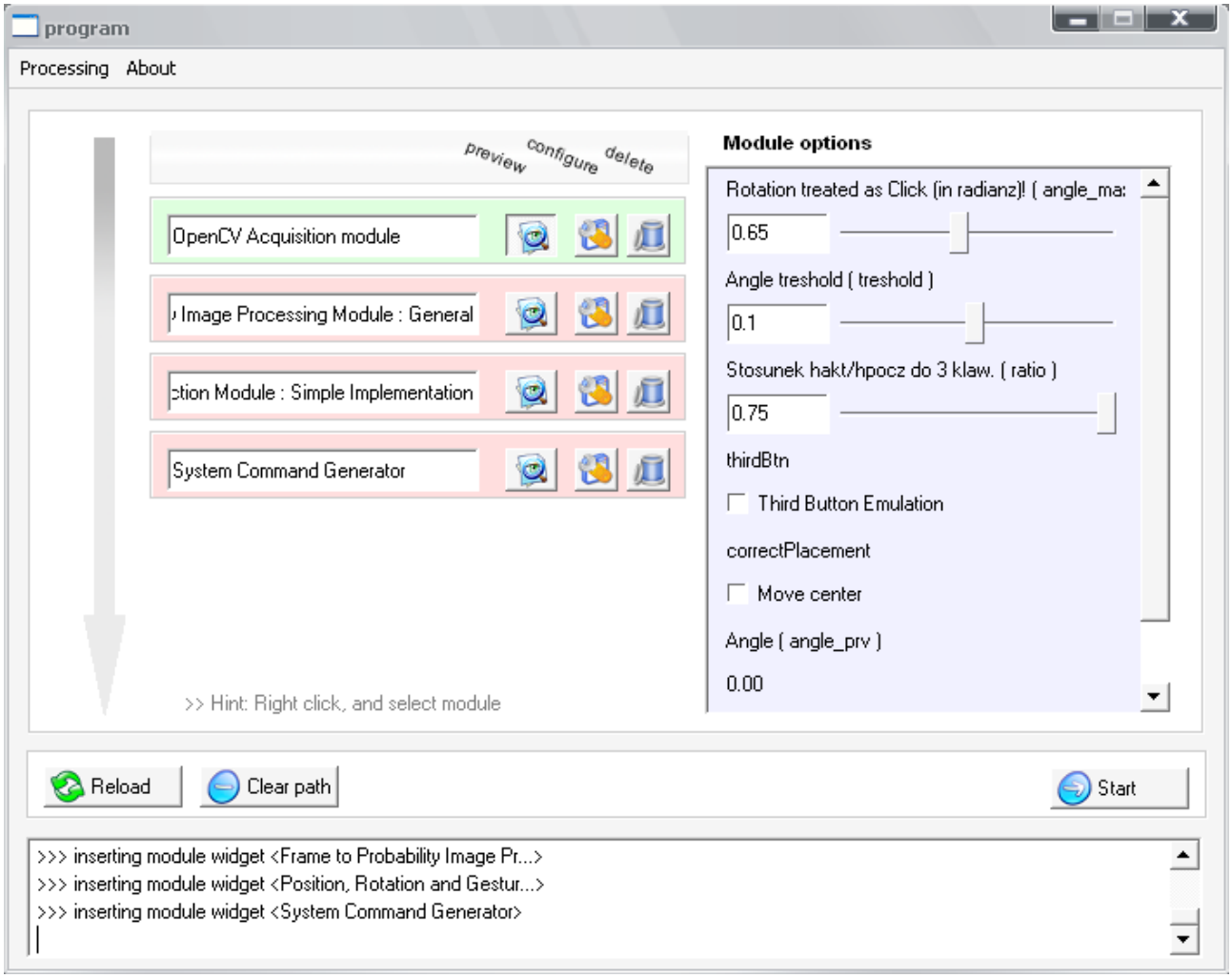


Ilustracja 6: Główne okno programu

Na oknie głównym widać podstawowe elementy:

- Przycisk „*Reload*” służący do przeładowania modułów;
- Przycisk „*Clear Path*” do zresetowania toru przetwarzania;
- Przycisk „*Start*” uruchamia symulację. Po uruchomieniu służy również do zatrzymywania przetwarzania.
- Białe pole toru przetwarzania. Klikając prawym klawiszem myszy można wybrać moduł który zostanie umieszczony w torze. Program sam dba o prawidłową kolejność modułów.
- Po prawej, na fioletowym tle znajduje się pole w którym ustawia się parametry wybranego modułu;
- Na samym dole znajduje się konsola, informująca o stanie pracy programu.
- Po dodaniu modułu do toru, obok jego opisu znajdują się trzy przyciski: pierwszy (Preview) służy do wybierania, czy ma zostać pokazany podgląd z przetwarzania obrazu w module, drugi (Config) służy do ustawiania parametrów modułu. Po kliknięciu na nim, po prawej stronie pokazują się możliwe do ustawiania parametry.
- Przyciskiem „Delete” można usunąć ostatni moduł z toru;

Na kolejnym zrzucie znajduje się program głowny z przykładowym torem:



Ilustracja 7: Okno programu z dodanymi modułami

Widać tu, że zostały umieszczone kolejno moduły: akwizycji obrazu, obliczania prawdopodobieństwa, detekcji obiektu oraz generacji poleceń dla systemu operacyjnego. Przetwarzanie obrazu następuje z „góry do dołu” tj. w tym wypadku od akwizycji do generacji poleceń dla systemu operacyjnego.

7. PROBLEMY NAPOTKANE PODCZAS REALIZACJI PROJEKTU

Podczas realizacji programu natknęliśmy się na kilka problemów, jednak wszystkie zostały rozwiązane. Podstawowym wyzwaniem było stworzenie elastycznego systemu modułów, tak aby można było szybko tworzyć dowolne nowe rozszerzenia.

Pewien problem wystąpił podczas implementacji wykrywania kąta obrotu i wielkości obiektu. Zastosowane rozwiązania z ubiegłego roku okazały się niewystarczające i dość problematyczne. Jednak wzory, znalezione w dokumentacji OpenCV oraz artykule [2] pomogły rozwiązać trudności. Przydatne również okazało się spojrzenie do kodów źródłowych biblioteki OpenCV.

8. LITERATURA

1. Robert Laganière: Programming computer vision applications
   http://www.site.uottawa.ca/~laganier/tutorial/opencv+directshow/cvision.htm
2. Gary R. Bradski: Computer Vision Face Tracking For Use in a Perceptual User Interface
   http://isa.umh.es/pfc/rmvision/opencvdocs/papers/camshift.pdf