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, zarówno 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 (obecnie w fazie implementacji). Nasz program obecnie 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). Dlatego też docelowo zostanie napisany moduł korzystający z alg. Camshift z tej biblioteki.

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 obliczane prawdopodobieństwo przynależności piksela do obiektu 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.

3. KONCEPCJA PROPONOWANEGO ROZWIĄZANIA

- Definiowanie modelu skóry
- Tworzenie obrazu prawdopodobieństwa
- Faza obliczeniowa CamShift
- Wyznaczenie orientacji obiektu

(dalsze etapy nie są na razie dostępne)

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:

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.

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 posiadam obecnie moduł obliczania prawdopodobieństwa dla całej ramki obrazu (nieco inaczej niż w przypadku oryginalnego algorytmu Camshift, gdzie prawdopodobieństwo jest liczone tylko dla regionu w którym się znajduje obiekt). Przykładową wizualizację tablicy prawdopodobieństw widać na rysunku poniżej.

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).

Prawdopodobieństwo przynależności piksela do obiektu jest obliczane przez sprawdzenie

czy wartości kanałów HSV mieszczą się w ustalonych zakresach, jeśli tak to prawdopodobieństwo dla danego piksela przyjmuje się jeden, w przeciwnym wypadku zero. Jeżeli została przeprowadzona kalibracja to dodatkowo do tego prawdopodobieństwa 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

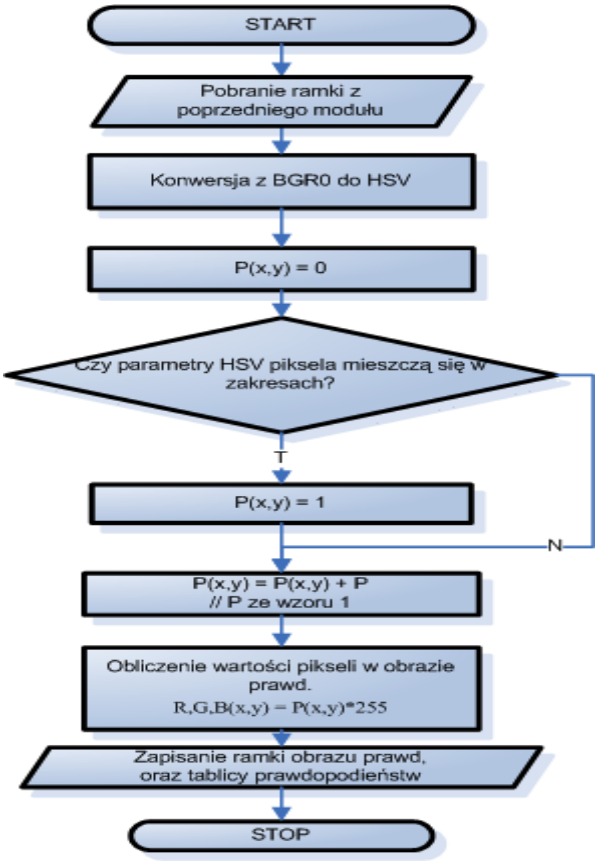
H(Hmax) - maksymalną ilość pikseli o tej samej wartości H

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:



3.3 Śledzenie dłoni – algorytm Camshift

Drugim etapem proponowanej przez nas metody jest śledzenie dłoni oraz obliczenie kąta obrotu dłoni dzięki zastosowaniu algorytmu Camshift.

Po wyznaczeniu tablicy prawdopodobieństw przez moduł do tego stworzony, 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

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

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

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

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

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

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

W pierwszej iteracji obszarem poszukiwań jest cała ramka (w naszym przypadku).

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: $s = \sqrt{\frac{M_{00}}{Max\_Val}}$

Wyznaczane są również rozmiary obiektu:

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

przy czym:

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

Po obliczeniu tych wartości następuje kolejna iteracja, ale dla mniejszego okna poszukiwań o szerokości s i wysokości 1.2*s.

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

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

3.4 Rozpoznawanie gestów

3.5 Generacja poleceń dla systemu operacyjnego

4. WNIOSKI

5. BUDOWA PROGRAMU

5.1 Moduły

5.1.1 Informacje wstępne

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 .

5.1.2 Podstawy działania, parametry.

Ponieważ nie jesteśmy w stanie na etapie pisania programu przewidzieć wszystkich możliwych zastosowań i ścieżek przetwarzania dlatego program sprawdza tylko, czy kolejność ułożenia modułów w torze jest prawidłowa. Do tego celu zdefiniowane zostały podstawowe typy wejść i wyjść z modułów:

```
#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_POSTGEST	Poprzedzający moduł powinien ustalić położenie i wykryć gesty	Moduł oblicza położenie obiektu i jego rotację oraz wykrywa gesty

Każdy moduł może mieć pewne zmienne które mogą być zmieniane z poziomu programu głównego (GUI). Moduł musi taką zmienną „zarejestrować" poprzez makrodefinicję:

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

gdzie:

TYPE - typ parametru (może być: PT_INT, PT_LONG, PT_FLOAT, PT_DOUBLE, PT_STRING, PT_FILENAME);

NAME – nazwa rejestrowanej zmiennej;

DESC – opis zmiennej;

DEF_VAL – domyślna wartość;

Każdy moduł musi mieć zaimplementowaną metodę :

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

ponieważ jest ona zawsze wywoływana, dla każdego modułu. Struktura proc_data zawiera wszystkie niezbędne struktury i zmienne. Mogłoby się wydawać, że skoro wszystkie moduły mają ten sam typ wejścia i wyjścia (w sensie struktury) to kolejność nie powinna mieć znaczenia. Jednak moduły oczekują, że pewne pola tej struktury będą zawierać konkretne dane, stąd też wynika potrzeba dbania o prawidłową kolejność następowania modułów. Dokładniejszy opis struktury proc_data (jak i innych używanych przez nas struktur i typów) znajduje się w pliku „*types.h*" w katalogu „*modules/module_base/src/*".

Każdy moduł musi również posiadać metody zwracające typ wejścia i wyjścia (typy zostały opisane wcześniej):

```
int input_type();
int output_type();
```

oraz metodę zwracającą krótki opis modułu:

```
const char * get_module_description();
```

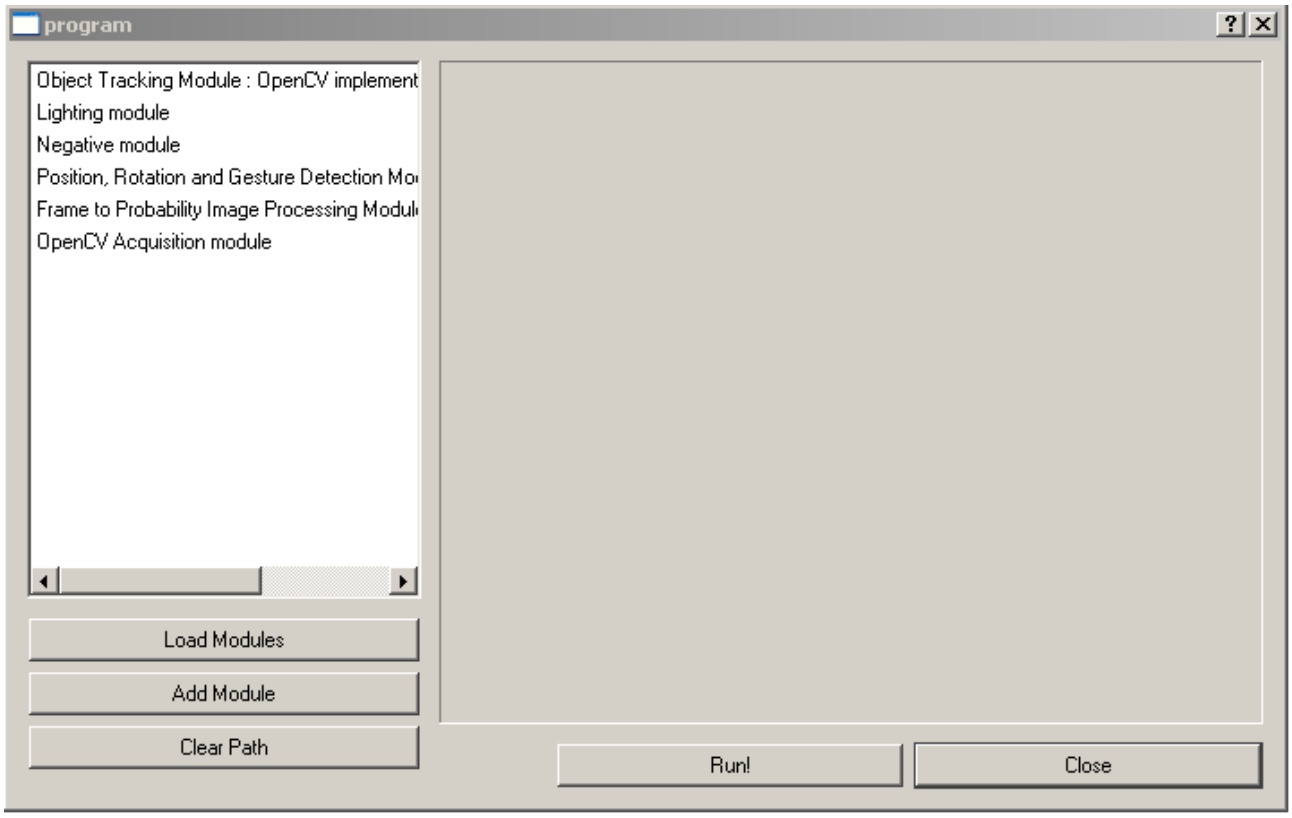
Bardzo ważna jest również inicjalizacja modułu, podczas której są ustalane wartości domyślne niektórych zmiennych oraz tworzone są niezbędne dla działania modułu struktury.

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 pliku „*types.h*".

5.1.3 Rejestrowanie innych zmiennych

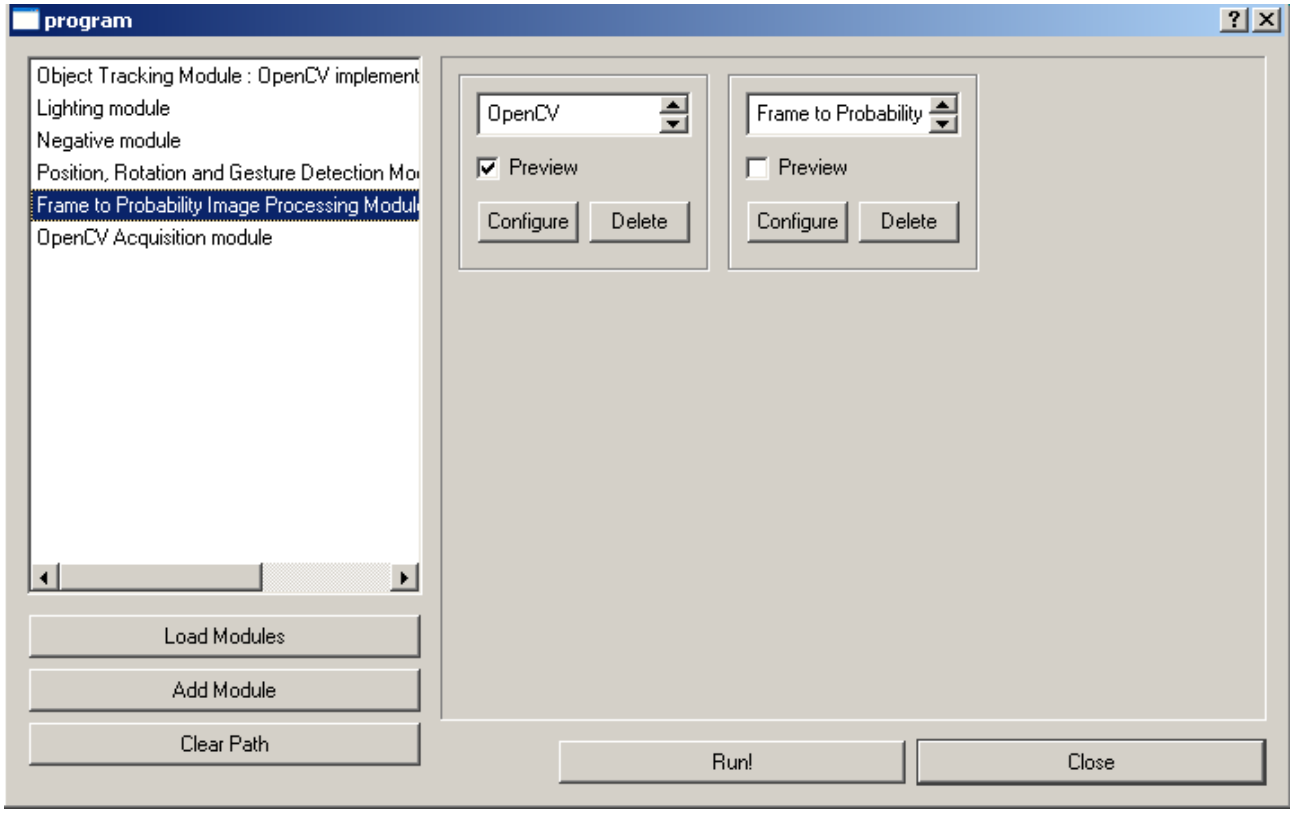
5.1.4 Możliwości debugowania

5.2 Opis okna programu i jego użytkowania

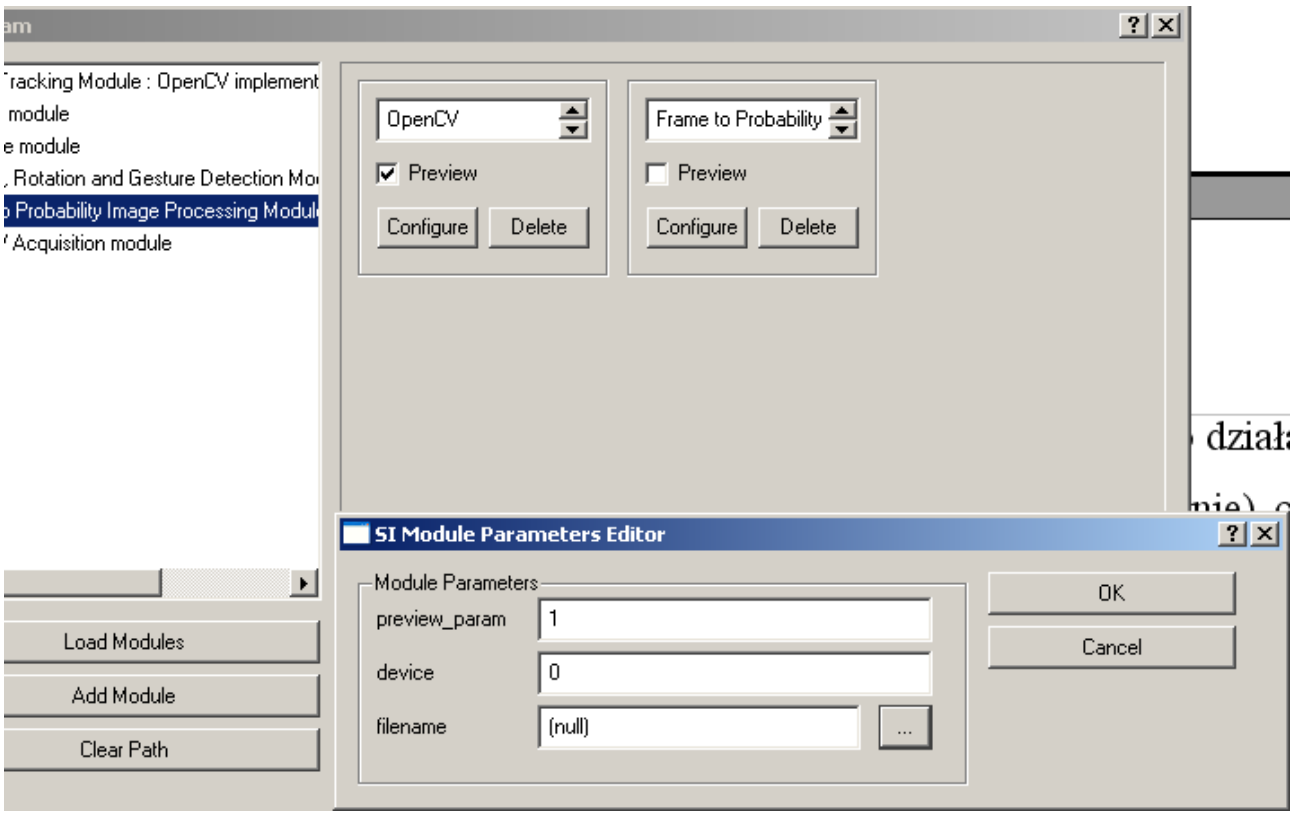


Program jest dość prosty w obsłudze. Główne okno programu widać na zrzucie ekranu znajdującym się powyżej. Widać po lewej stronie listę modułów. Żaden z nich nie jest załadowany, co widać jako puste pole po prawej stronie programu. Przycisk „**Load modules**" służy do wyładowania modułów już załadowanych oraz ponownego ich załadowania. Może być to przydatne bądź to ponownego odczytania katalogu z bibliotekami, lub też do przywrócenia ustawień

domyślnych dla wszystkich modułów. Z kolei zaznaczenie na liście po prawej stronie modułu i kliknięcie przycisku „**Add module**” spowoduje z kolei dodanie modułu to toru przetwarzania (o ile moduł spełnia wymogi odnośnie poprawnej kolejności). Przycisk „**Clear Path**” powoduje zresetowanie toru przetwarzania. Przyciskiem „**Run!**” uruchamiamy proces przetwarzania obrazu, o ile tor przetwarzania jest poprawnie skonstruowany. Przyciskiem „**Close**” zamykamy program.
Na kolejnym zrzucie ekrany widać okno programu z prostym torem przetwarzania:



Jak widać zostały załadowane dwa moduły. Jest to poprawny tor przetwarzania, którego działaniem będzie jedynie obliczenie prawdopodobieństw oraz wygenerowanie (i ew. wyświetlenie) obrazu tego prawdopodobieństwa. Każdy moduł ma swoją nazwę która wyświetla się w polu u góry. Można zaznaczyć czy chcemy oglądać podgląd działania modułu zaznaczając (bądź nie) pole „**Preview**”. Każdy moduł można również skonfigurować, o ile oczywiście zarejestrował on parametry (rejestrowanie parametrów zostało opisane w rozdziale 5.1.2 tego dokumentu). Przykładowe okno konfiguracji wygląda jak na obrazie poniżej:



W tym wypadku można ustawić dwa parametry typu int oraz jeden odpowiedzialny za wybór pliku. Zmiana parametrów podczas przetwarzania obrazu ma wpływ na działanie programu.

6. PROBLEMY NAPOTKANE PODCZAS REALIZACJI PROJEKTU

7. 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