



Przedmiot: Sztuczna inteligencja i sensoryka.

PR063HCI4

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

Spis treści

1. ABSTRAKT.....	1
2. WSTĘP.....	2
3. KONCEPCJA PROPONOWANEGO ROZWIĄZANIA.....	3
3.1 Definiowanie modelu skóry.....	4
3.1.1 Kalibracja.....	4
3.2 Tworzenie obrazu prawdopodobieństwa.....	5
3.2.1 Informacje wstępne; obliczanie prawdopodobieństw.....	5
3.3 Śledzenie dłoni – algorytm Camshift.....	7
3.4 Rozpoznawanie gestów.....	8
3.5 Generacja poleceń dla systemu operacyjnego.....	8
4. WNIOSKI.....	8
5. BUDOWA PROGRAMU (Opis API bibliotek).....	8
5.1 Informacje ogólne.....	8
5.2 Funkcje i metody.....	8
5.3 Typy i struktury.....	10
5.4 Zdefiniowane nazwy.....	11
5.5 Ważniejsze pliki.....	13
6. OBSŁUGA PROGRAMU.....	13
7. PROBLEMY NAPOTKANE PODCZAS REALIZACJI PROJEKTU	14
8. LITERATURA.....	14

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 przechwytyjące), 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, wynalezionej 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 przechwytyjących 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 dłonie. 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
- 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:



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 posiada 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

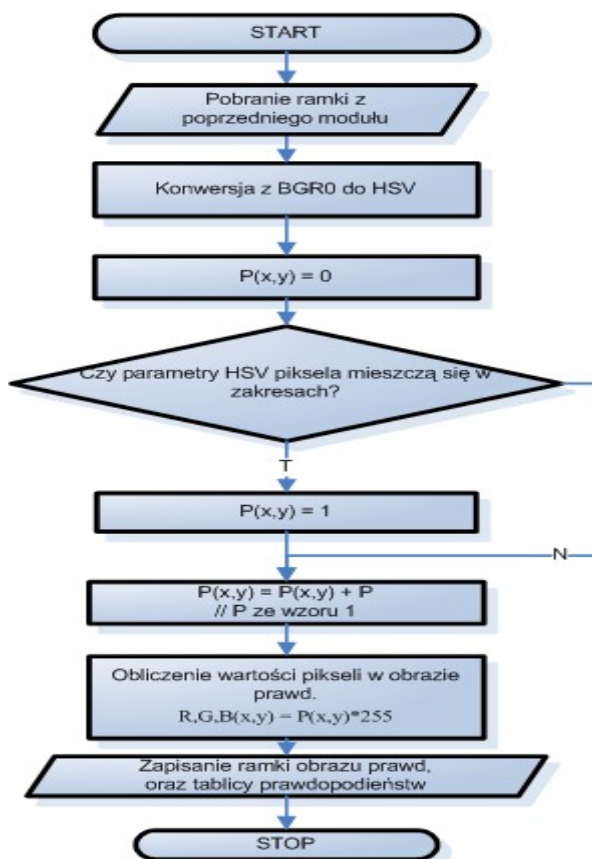
$H(H_{\max})$ - maksymalną ilość pikseli o tej samej wartości H

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:

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

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

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

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

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

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

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:

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

Wyznaczane są również rozmiary obiektu:

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

przy czym:

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

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

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 (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.

5.2 Funkcje i metody

int *init(PropertyMgr * pm)*

Metoda inicjalizacji modułu, podczas której są ustalane wartości domyślne niektórych zmiennych oraz tworzone są niezbędne dla działania modułu struktury. Wywoływana każdorazowo przy uruchamianiu symulacji. Ponadto podczas może zostać ustawiony tzw. Menadżer Zmiennych (PropertyMgr). Służy on do rejestrowania w

programie dowolnych zmiennych, które mogą być używane przez inne moduły. Menadżer został zaimplementowany, aby umożliwić tworzenie modułów, które nie zostały przewidziane podczas pisania projektu. Aby go móc używać należy go zainicjalizować makrem *USE_PROPERTY_MGR(pm)*.

Metoda powinna zwrócić jeden z kodów stanu.

void *free()*

Metoda służąca do zwolnienia zasobów stworzonych przy inicjalizacji modułu bądź jego działaniu. Ponadto w tym miejscu powinno zwolnić się zajmowane urządzenia, lub pliki. Funkcja nic nie zwraca.

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

Najważniejsza metoda, wywoływana dla każdego modułu dla każdej ramki obrazu. To w tej metodzie wykonuje się wszystkie elementy przetwarzania obrazu i detekcji obiektów. Metoda pobiera strukturę *proc_data* i po przetworzeniu powinna zwrócić również nową, przetworzoną strukturę. Dzięki temu możliwe jest przetwarzanie obrazu w kolejnych modułach. Dodatkowo należy wpisać do zmiennej *result* status wykonania. 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 tylko dla modułów które wymagają kalibracji (zaznaczenia jakiegoś obszaru na obrazie). Jako parametry przekazywane są współrzędne lewego górnego wierzchołka prostokąta oraz jego wysokość i szerokość. Funkcja nic nie zwraca.

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 wejść/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 <>), 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)).

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 wejscowego
    frame_data * frame;       /// ramka obrazu
    pd_data * position;       /// informacje o pozycji kursowa, gestach
    float * prob;             /// obraz prawdopodobienstwa
    float max_prob;           /// maksymalna wartosc prawdopodobienstwa
    void * user_data0;        /// niezdefiniowane
    void * user_data1;        /// j/w
};
```

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_LMBCCLICK    0x01 /// wciśnięty lewy klawisz myszki
#define GESTURE_RMBCCLICK    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_POSTGEST         0x03
```

Interpretację tych typów przedstawia tabela:

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

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

#define ST_UNDEFINED         0xFF
```

5.5 Ważniejsze pliki

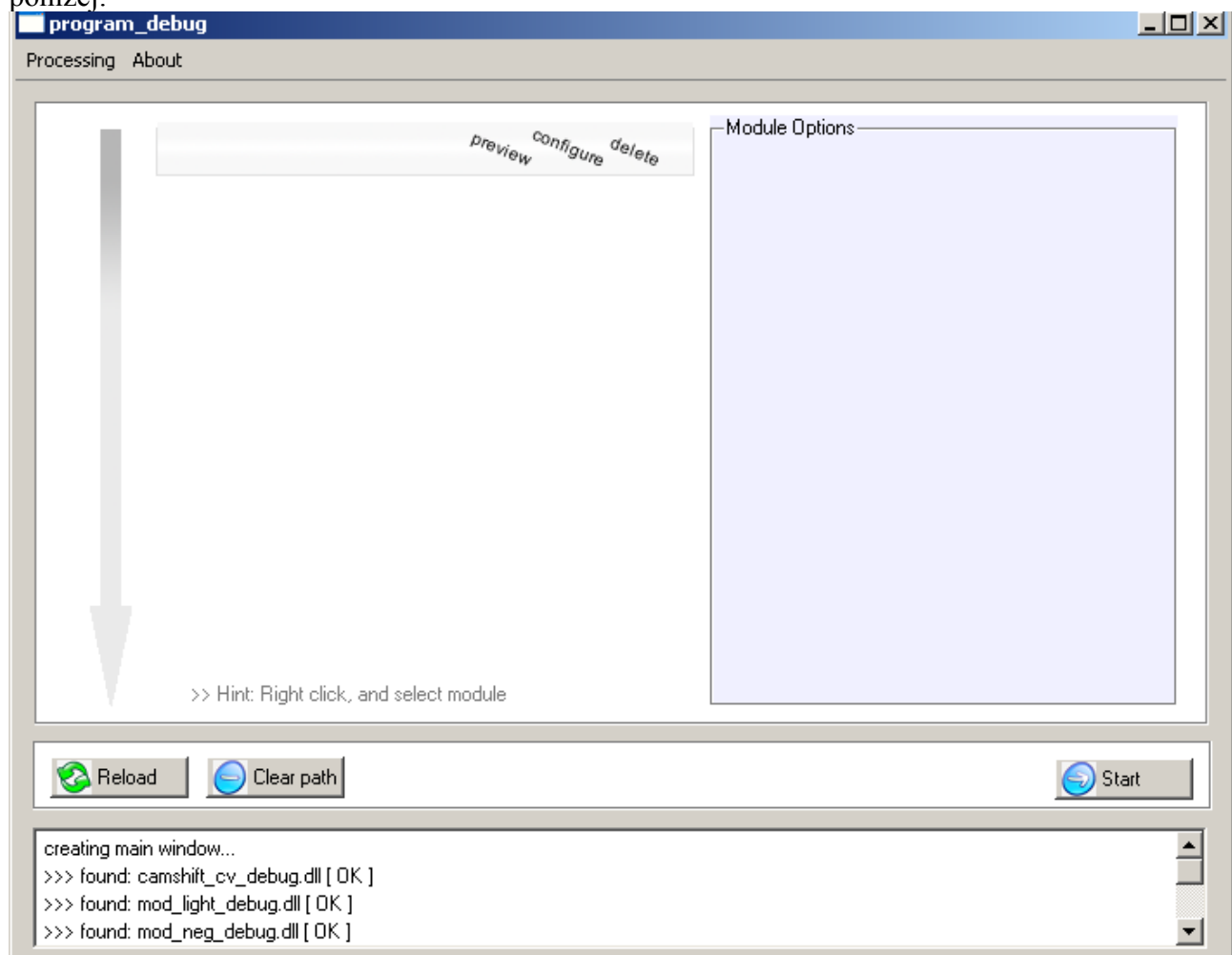
Aby móc napisać jakikolwiek moduł należy do kodu dołączyć kilka plików nagłówkowych:

- *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:

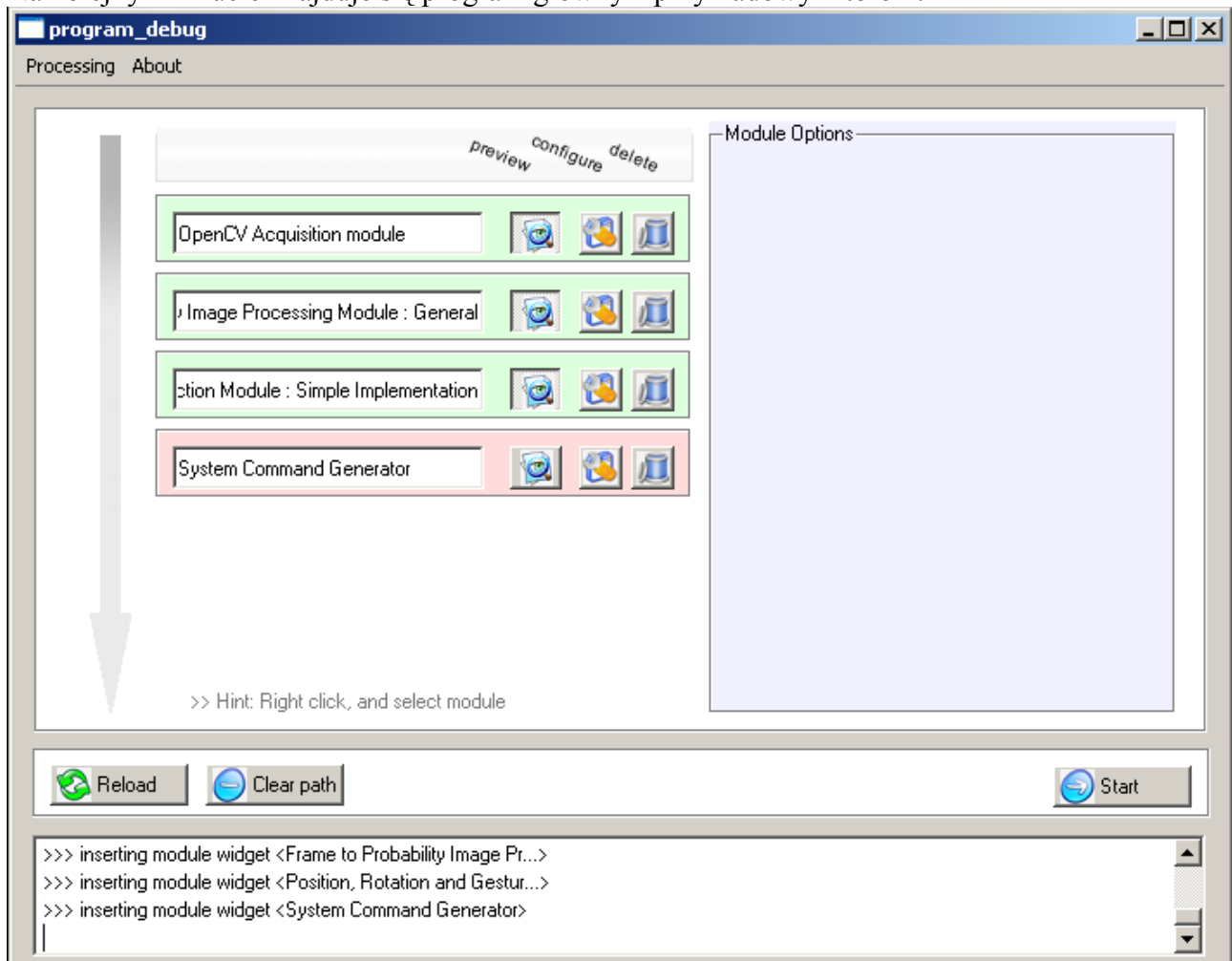


Na oknie głównym widać podstawowe elementy:

- 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.

Na kolejnym rzucie znajduje się program główny z przykładowym torem:



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

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>