AKADEMIA GÓRNICZO-HUTNICZA

im. St. Staszica w Krakowie

WEAIiE, Katedra Automatyki

Laboratorium Biocybernetyki

MM123_ASSISTIVE

Przedmiot: Interfejsy Multimodalne

Temat projektu: **ASSISTIVE**

Spis treści:

Wykonali: Daniel Jagielski, Tomasz Kmiecik, Marcin Marzyński

3 rok Informatyki Stosowanej

konsultant: *Jaromir Przybyło*

Wersja 1.0

Kraków, czerwiec 2012.

1. Abstrakt

Celem projektu było stworzenie oprogramowania, które dzięki wykorzystaniu analizy informacji wizyjnej, umożliwiłoby zastąpienie klasycznych metod sterowania urządzeniami (np. telewizorem, komputerem). Obraz jest akwizowany przez kamerę umieszczoną nad użytkownikiem siedzącym na wózku inwalidzkim w taki sposób, że obie dłonie osoby niepełnosprawnej są doskonale widoczne. Zadaniem aplikacji jest wykrycie ruchu obu rąk, przy czym ruch prawej dłoni pozwala na wskazanie odpowiedniego kierunku, a gesty lewej dłoni na dokonanie akcji wyłączenie.

Do stworzenia aplikacji wykorzystaliśmy bibliotekę OpenCV. Obraz potrzebny do testów uzyskaliśmy przy pomocy laboratoryjnej kamery Logitech.

W rezultacie naszych działań otrzymaliśmy

- wykrywanie gestów pokazywania kierunków (ruchy w lewo lub w prawo) wykonywanych prawą ręką, na podstawie algorytmu Camshift.

- wykrywanie gestu potwierdzenia (zaciśnięcie i otwarcie dłoni)

- pokazywanie obrazka z aktualnie zaznaczoną pozycją menu

- przechodzenie po menu

- wywołanie testowych akcji dla poszczególnych stanów

Stworzona przez nas aplikacja w przyszłości może zostać wzbogacona o podpięcie konkretnych akcji systemowych (zmianę piosenek, uruchamianie programów zewnętrznych), kalibracja w trybie rzeczywistym (zamiast na uprzednio przygotowanym materiale), konfigurowanie opcji menu.

Słowa kluczowe: OpenCV, prawa ręka , lewa ręka, CamShift, Back Projection

2. Wstęp

a) Cele i założenia projektu.

Celem projektu było stworzenie oprogramowania, które dzięki wykorzystaniu analizy informacji wizyjnej, umożliwiłoby zastąpienie klasycznych metod sterowania urządzeniami (np. telewizorem, komputerem). Obraz jest akwizowany przez kamerę umieszczoną nad użytkownikiem siedzącym na wózku inwalidzkim w taki sposób, że obie dłonie osoby niepełnosprawnej są doskonale widoczne. Zadaniem aplikacji jest wykrycie ruchu obu rąk, przy czym ruch prawej dłoni pozwala na wskazanie odpowiedniego kierunku, a gesty lewej dłoni na dokonanie akcji wyłączenie. Sposób ten można porównać do obsługi starych telefonów komórkowych przy pomocy joysticka, gdzie wciśnięcie joysticka powodowało akcję, a ruchy gałką pozwalały na nawigację po jednowymiarowym menu. Powyżej omówione podejście nie jest popularne na świecie, stosuje się raczej wykrywanie ruchu gałek ocznych lub wykrywanie ruchów głowy.

b) Zarys ogólny proponowanego rozwiązania.

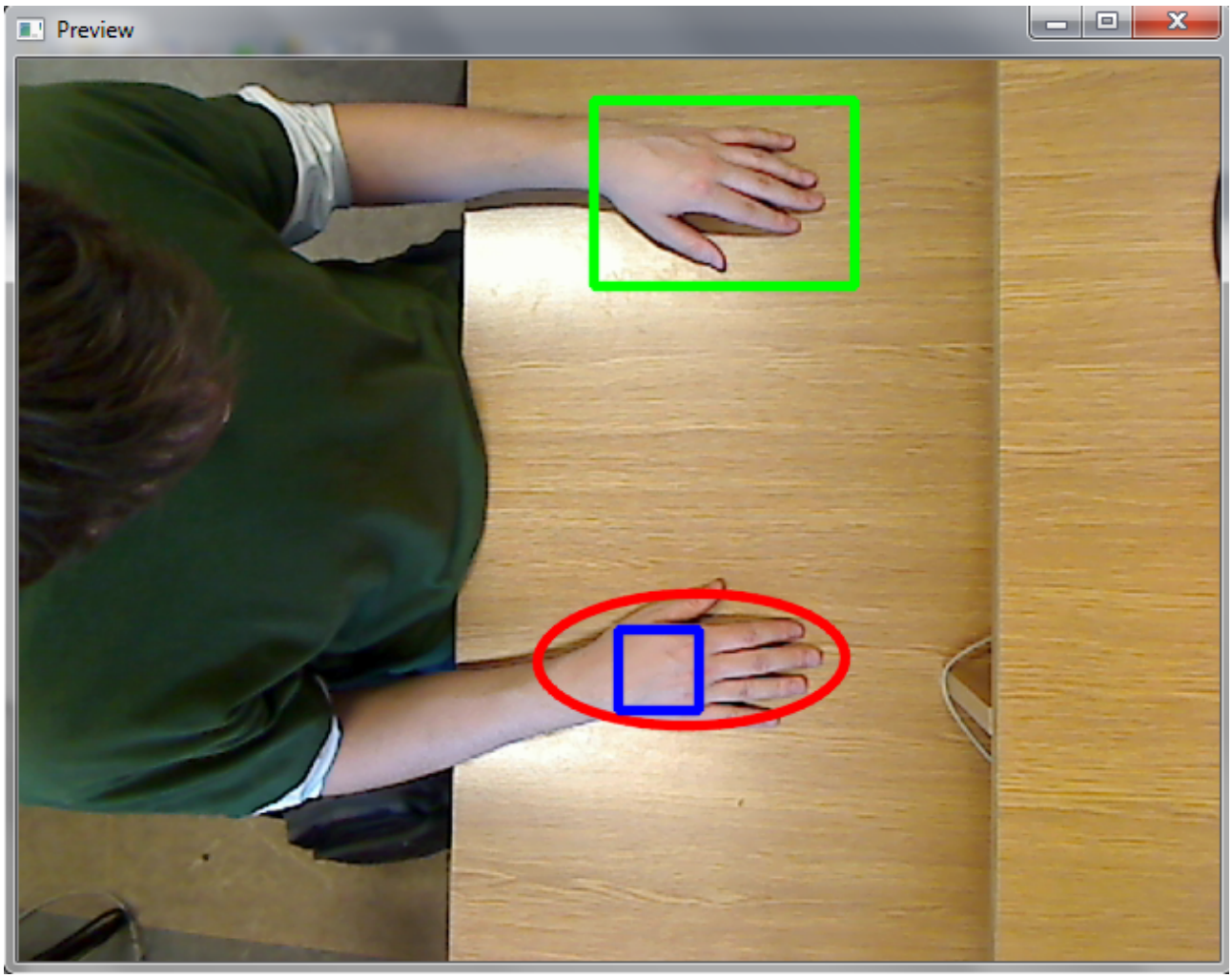
Pierwszym etapem jest kalibracja, wykrywanie położenia obu rąk. Należy przydzielić odpowiednie gesty do odpowiedniej dłoni. Następnie należy dokonać implementacji rozpoznawania gestów obu dłoni w języku C++ przy pomocy algorytmu camshift z biblioteki OpenCV.

Kolejnym etapem będzie stworzenie sterowania maszyny stanowej, przy pomocy której będziemy sterować naszą aplikacją. Ostatnim etapem będzie stworzenie wizualizacji dla naszego projektu i podpięcie pod nią konkretnych akcji.

.

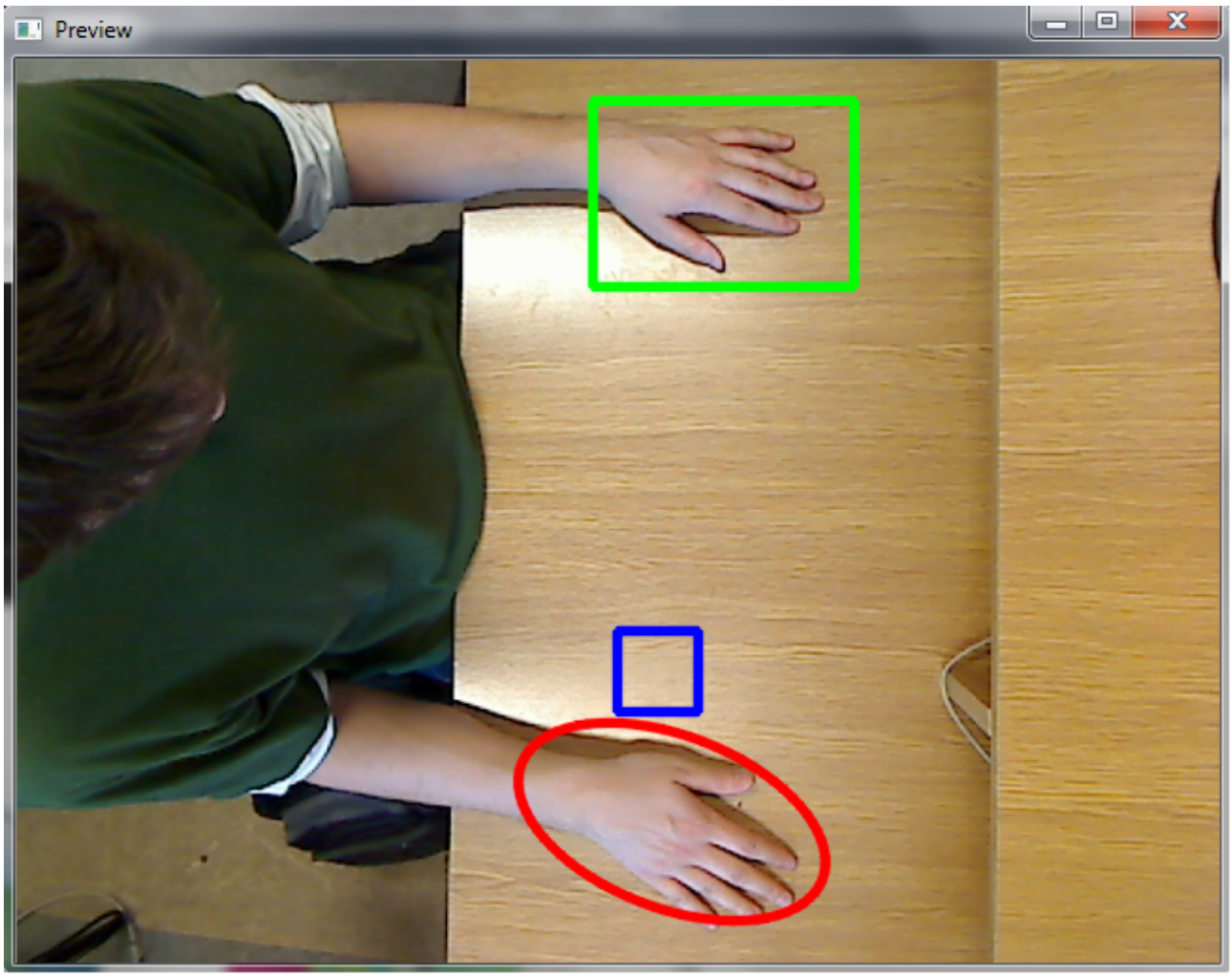
3. Koncepcja proponowanego rozwiązania

Pracę rozpoczęto od przydzielenia konkretnych gestów pod konkretną dłoń. Założono, że prawa ręka będzie odpowiedzialna za wykonywanie ruchu, przemieszczanie się po menu, natomiast lewa ręka miałaby być odpowiedzialna za wykonywanie akcji. Wymyślono kilka propozycji tego gestu, wybór padł na zaciśniecie pięści ze względu na łatwość implementacji takiego rozwiązania w języku c++. Następnie zaimplementowano wykrywanie prawej ręki – dokonano tego przy użyciu algorytmu camshift z biblioteki OpenCV. Położenie dla ręki jest z kolei ustawione „na sztywno” w związku z tym, że nie zmienia swojego położenia w ciągu działania programu.



Podgląd widoku użytkownika z kamery

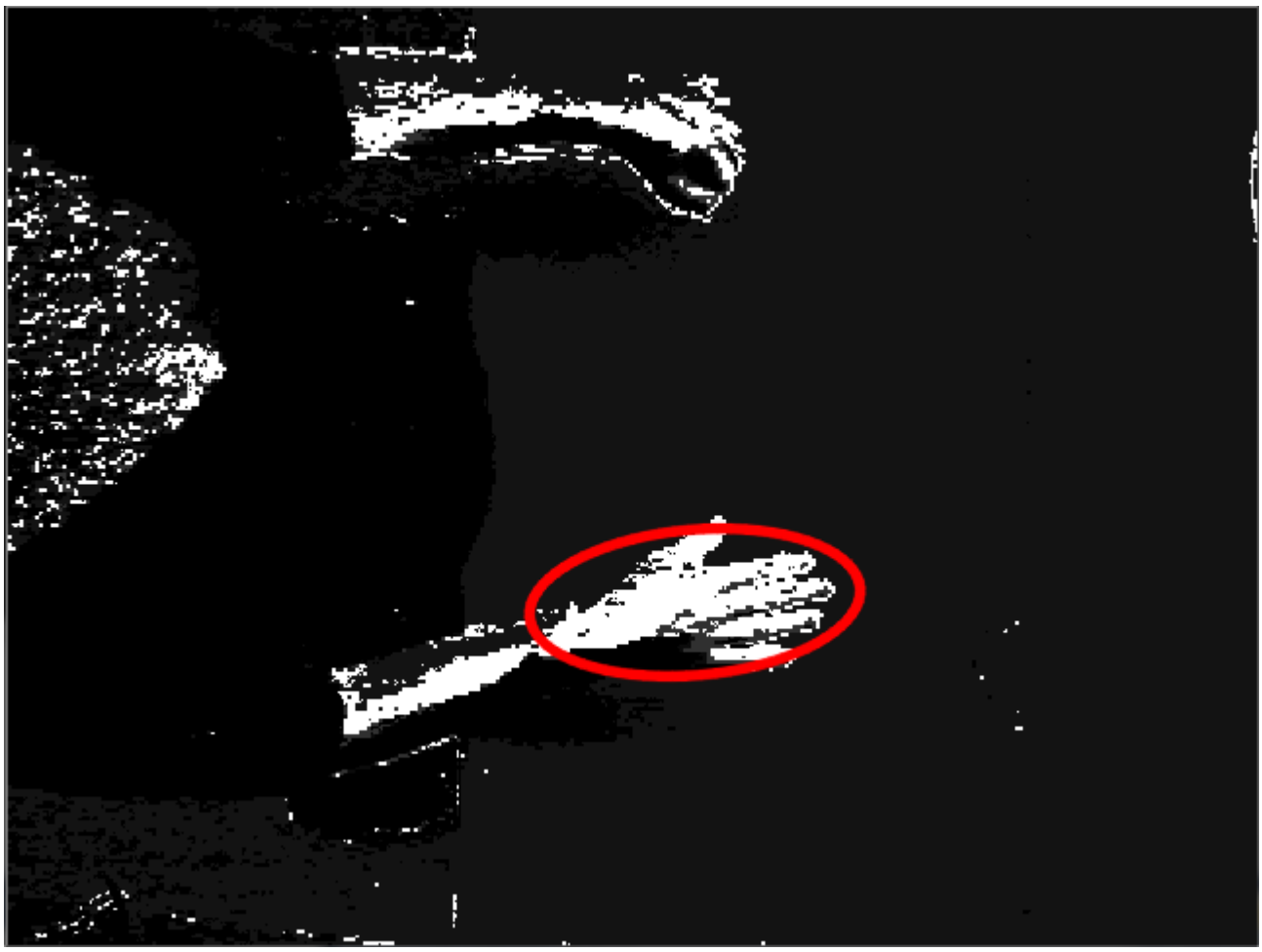
Zielona i niebieska ramka to ramki kalibracyjne - użytkownik musi trzymać prawą rękę tak, aby skóra jego dłoni wypełniała niebieską ramkę w całości. Na tej podstawie są wyznaczane parametry początkowe dla algorytmu camshift. Czerwona elipsa pokazuje położenie prawej ręki (wynik śledzenia algorytmu). Lewa ręka użytkownika musi znajdować się wewnątrz zielonej ramki. Dla wygody użytkownika można wcześniej oznaczyć markerem miejsce na stole, gdzie spoczywa lewa dłoń. Nie jest to jednak konieczne, gdyż użytkownik na bieżąco widzi podgląd z kamery z zaznaczonym obszarem w którym powinien trzymać dłoń.



Podgląd widoku użytkownika po zmianie położenia dłoni

Jak widać na załączonym obrazku – po wykryciu przez camshift niebieski kwadrat pozostaje na miejscu, natomiast czerwona elipsa podąża obrazując śledzenie ruchu prawej dłoni.

Dodatkowo wyliczony Back Projection [3] przez algorytm camshift używamy do wykrywania lewej ręki. Liczymy ilość dopasowanych punktów wewnątrz zielonego prostokąta. Jeśli spadnie poniżej krytycznego poziomu oznacza to, że zmniejszyło się pole powierzchni ręki (np. została zaciśnięta w pięść) i należy wykonać daną akcję. Zarówno śledzenie ręki jak i zaciskanie z punktu widzenia Back Projection zostały pokazane na dołączonym obrazku.

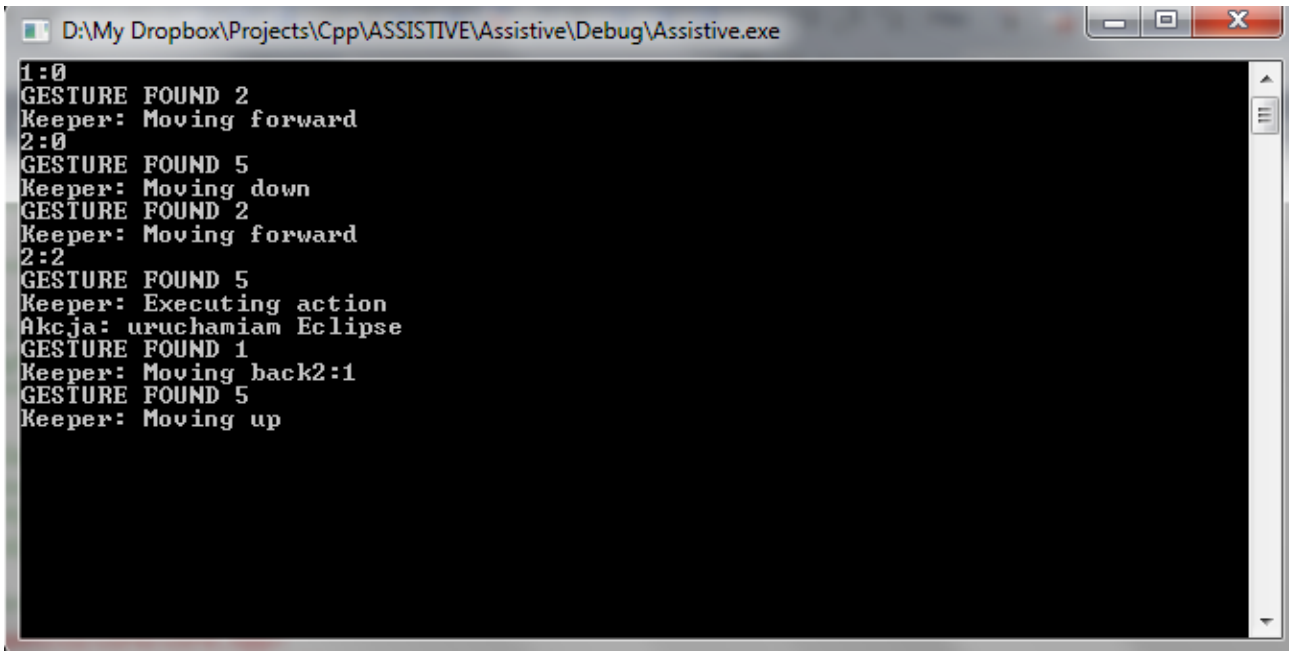


Działanie Back Projection

Następnie należało stworzyć maszynę stanową. Stworzyliśmy poziomy w obrębie, których możemy poruszać się w lewo i w prawo, po wybraniu/akceptacji przenosimy się w głąb struktury,. Tutaj znajdują się kolejne opcje i możemy wrócić z powrotem na poziom wyżej, poziom niżej lub wybrać daną usługę w zależności od potrzeby użytkownika.

Informacje o sterowaniu maszyną stanów menu ("Keeper: ...") oraz efekty wykonywania kodu podpiętego pod akcje menu zostają przedstawione na konsoli w następującym formacie - X:Y (X – oznacza pozycje w pierwszym poziomie menu, Y – oznacza pozycję w obrębie tego menu, gdzie 0 to element główny, 1 to element back, czyli powrotu do poziomu głównego menu a inne liczby to akcje podpięte pod to menu) lub zwykłym opisem słownym, jeśli jest to usługa.

W konsoli możemy także obserwować pomocnicze informacje pozwalające śledzić przebieg programu. Wyświetlane są informacje o znalezionych gestach "GESTURE FOUND", numery gestów można porównać z nazwami stałych zdefiniowanych w pliku Gesture.hpp.



Konsola

Na koniec należało dokonać wizualizacji poczynionych prac; stworzono zatem następujące menu, po którym kursorem poruszamy się poprzez wykonywanie gestów.

Menu – poszczególne ikony odpowiadają za kolejny poziom zagnieżdżenia/opcję

4. Rezultaty i wnioski

W wyniku prac otrzymano działający program pozwalający na wykrywanie gestów prawej dłoni – ruchów w lewo i w prawo. Udało się także wykryć gesty lewej dłoni – jedynie jako gest zaciśnięcia pięści. Jednakże należałoby zastanowić się nad innymi rozwiązaniami, ze względu na to, że nie każdy niepełnosprawny może być w stanie zacisnąć pięść. Kolejnym ograniczeniem jest konieczność umieszczania lewej dłoni w jednym, konkretnym miejscu.

Stworzono także prostą maszynę stanową, którą można wykorzystać nie tylko przy obsłudze komputera, ale także innych urządzeń. Zrealizowano obsługę tylko uprzednio przygotowanych sekwencji video, należy zastanowić się nad sposobem przetwarzania video w czasie rzeczywistym.

Utworzono prostą wizualizację w formie menu zawierającego duże ikony reprezentujące możliwe wybory; wielkość owych ikon jest istotna, ze względu na osoby niepełnosprawne z dużą wadą wzroku. Umieszczono również wywołanie testowych akcji dla poszczególnych stanów, np. "Uruchomienie eclipse".

5. Podsumowanie

Realizacja podstawowych założeń projektu przy pomocy biblioteki OpenCV nie stanowiła problemu. Zastosowanie algorytmu camshift pozwoliło na znaczne skrócenie prac nad projektem. Zagadnienie zaczęło się komplikować jednak przy wykrywaniu ustawienia lewej ręki i rozróżnianiu jej gestów. Zarzucono przez to możliwość cofania w menu. Nie udało nam się stworzyć wersji release projektu w Visual Studio.
Ten projekt można rozbudować w przyszłości o podpięcie konkretnych akcji systemowych (zmiana piosenek, uruchamianie programów zewnętrznych) lub też nawet o obsługę innych urządzeń niż komputer. Można także dodać kalibrację w czasie rzeczywistym, a także menu, które pozwalałoby na dowolną konfigurację i ustawianie opcji w zależności od potrzeb użytkownika.

6. Literatura

[1] Laganiere Robert "OpenCV 2 Computer Vision Application Programming Cookbook"
[2] Bradski Gary, Kaehler Adrian "Learning OpenCV"
[3] OpenCV Back Projection Tutorial
http://opencv.itseez.com/doc/tutorials/imgproc/histograms/back_projection/back_projection.html
[dostęp: 15 V 2012]

7. DODATEK A: Opis opracowanych narzędzi i metody postępowania

Do pracy z projektem ASSISTIVE wymagane jest zainstalowanie Visual Studio 2010 oraz biblioteki OpenCV w wersji 2.2 dla Visual Studio.

Sposób tworzenia aplikacji oparliśmy na opisie zawartym w książce "OpenCV 2 Computer Vision Application Programming Cookbook" [1]. Znajduje się w rozdziale 1, na stronach 11-18.

Plik źródłowy z filmem musi znajdować się w konkretnym katalogu, w przeciwnym przypadku program nie zadziała. Domyślna linia to : ("E:/assistive/marcin/gest.avi"). Możemy zmienić ścieżkę dostępu w pliku Assistive.cpp w tej linii : videoInput.open("E:/assistive/marcin/gest.avi");

8. DODATEK C. Opis informatyczny procedur

Do realizacji zadania zastosowaliśmy środowisko programistyczne Visual Studio 2010 firmy Microsoft. By zbudować projekt używamy opcji Debug, ze względu na znane i nierozwiązane problemy z opcją Run. Musimy także pamiętać o dołączeniu biblioteki OpenCv w wersji 2.2 specjalnie stworzonej dla Visual Studio , bez której projekt nie ruszy.

Poniżej zostaną przedstawione najważniejsze funkcje:

Funkcja służąca do przetwarzania każdej ramki

```
/*****************************************************************************/
/* Autorzy:          Marcin Marzyński, Daniel Jagielski, Tomasz Kmiecik     */
/* Kierunek:  III rok Informatyka Stosowana, EAIiE, AGH                     */
/* Data modyfikacji: 10-06-2012                                             */
/*****************************************************************************/
/*
                                                                  */
/* Funkcja odpowiedzialna za dokonywanie konwersji na HSV, obliczanie histogramu
śledzonego obrazu, wykrywanie gestów Wykorzystuje bibliotekę OpenCV do przetwarzania
obrazów.                                  */
/* Do śledzenia prawej ręki wykorzystywany jest algorytm CamShift (implementacja */
/* z OpenCV). Gesty lewej ręki liczone są na podstawie backprojection obrazu
*/  /* Dołączone biblioteki:
*/
/* #include <opencv2/core/core.hpp>                                          */
/* #include <opencv2/highgui/highgui.hpp>                                    */
/* #include <opencv2/video/tracking.hpp>                                     */
/* #include <opencv2/imgproc/imgproc.hpp>                                    */
/* #include <opencv2/highgui/highgui.hpp>                                    */
/* Funkcja przyjmuje : Mat image – obraz ramki, zwraca: void                 */
/*****************************************************************************/



void VideoProcessor::processFrame(Mat image)
{
       frameNo++;
       //Mat hsv, hue, mask;
       int vmin = 10, vmax = 256, smin = 30, hsize = 16;
       int minDelta = 50;

       float hranges[] = {0,180};
    const float* phranges = hranges;
```

```
    //konwersja kolorów z RGB do HSV
    cvtColor(image, hsv, CV_BGR2HSV);

    inRange(hsv, Scalar(0, smin, MIN(vmin,vmax)),
                 Scalar(180, 256, MAX(vmin, vmax)), mask);

    int ch[] = {0, 0};
    hue.create(hsv.size(), hsv.depth());
    mixChannels(&hsv, 1, &hue, 1, ch, 1);

    if(mFirstFrame)
    {
          //obliczenie histogramu śledzonego obszaru
          Mat roi(hue, mSelection), maskroi(mask, mSelection);
     calcHist(&roi, 1, 0, maskroi, mHist, 1, &hsize, &phranges);
          normalize(mHist, mHist, 0, 255, CV_MINMAX);
          mFirstFrame = false;
          trackWindow = mSelection;
    }

    calcBackProject(&hue, 1, 0, mHist, backproj, &phranges);
backproj &= mask;

    mLeftGestureReady = false;

    Mat myBP;
    //cvtColor(backproj, myBP, CV_HSV2BGR);
    backproj.copyTo(myBP);
    myBP = backproj(mLeftSelection);
    double s = sum(myBP)[0];
    if(s < 700000)
    {
          mLeftGestureActive = true;
          //std::cout << frameNo << " BOOM " << s << std::endl;
    } else
    {
          //jezeli był aktywy to znaczy, że to koniec zaciskania ręki
          if(mLeftGestureActive)
          {
                mLeftGesture.type = Gesture::GESTURE_LHAND_MOVE;
                mLeftGestureReady = true;
          }
          mLeftGestureActive = false;
    }

mTrackBox = CamShift(backproj, trackWindow,
                     TermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10, 1 ));

    mGestureReady = false;

    //jezeli jeszcze nie zapamietano polozenia srodka to teraz zapamietuje
    if(!mFixedTrackBox.size.width)
    {
          mFixedTrackBox = mTrackBox;
    }

    int delta = mFixedTrackBox.center.y - mTrackBox.center.y;
    if(delta >= minDelta)
    {
          //gest lewy
          if(mGesture.type != Gesture::GESTURE_RHAND_LEFT)
          {
                mGesture.type = Gesture::GESTURE_RHAND_LEFT;
                mGestureReady = true;
          }
```

```
	}
	else if((-1*delta) >= minDelta)
	{
		//gest prawy
		if(mGesture.type != Gesture::GESTURE_RHAND_RIGHT)
		{
			mGesture.type = Gesture::GESTURE_RHAND_RIGHT;
			mGestureReady = true;
		}
	}
	else
	{
		mGesture.type = Gesture::GESTURE_UNKNOWN;
	}

}
```

Główna funkcja programu

```
/*****************************************************************************/
/* Autorzy:		Marcin Marzyński, Daniel Jagielski, Tomasz Kmiecik
			*/
/* Kierunek:  III rok Informatyka Stosowana, EAIiE, AGH
			*/
/* Data modyfikacji: 10-06-2012
					*/
/*****************************************************************************/
/*
								*/
/* Główna funkcja aplikacji, jednocześnie jej kontroler.
			*/
/* Inicjalizuje wszystkie obiekty, w tym okna aplikacji. Następnie odczytuje kolejne*/
/* ramki z filmu i przekazuje je do obiektu VideoProcessor, który odpowiedzialny   */
/* jest za śledzenie obiektów na ekranie i rozpoznawanie wykonywanych przez nie    */
/* gestów.                                                                         */

int main(int argc, char* argv[])
{
	//inicjalizacja video
	VideoCapture videoInput;
	videoInput.open("E:/assistive/marcin/gest.avi");

	if(!videoInput.isOpened())
	{
		cout << "***Could not initialize capturing...***\n";
		std::system("pause");
	 return 0;
	}

	//utworzenie okienek
	cv::namedWindow(PREVIEW_WINDOW_ID);
	cv::namedWindow(MENU_WINDOW_ID);

	VideoProcessor videoProcessor;
	StateKeeper stateKeeper;
	cout << stateKeeper.getMainPos() << ":" << stateKeeper.getSubPos() << endl;

	//pętla odczytywania klatek i wysyłania ich do przetwarzania
	for(;;)
	{
		Mat frame;
		videoInput >> frame;
		if(frame.empty()) {
```

```
                        cout << "Escaping loop \n";
                        break;
                }

                frame.copyTo(image);

                videoProcessor.processFrame(image);
                if(videoProcessor.isGestureReady())
                {
                        Gesture gesture = videoProcessor.getGesture();
                        cout << "GESTURE FOUND " << gesture.type << endl;
                        switch(gesture.type)
                        {
                        case Gesture::GESTURE_RHAND_LEFT:
                                cout << "Keeper: Moving back";
                                stateKeeper.moveBackwards();
                                cout << stateKeeper.getMainPos() << ":" <<
stateKeeper.getSubPos() << endl;
                                break;
                        case Gesture::GESTURE_RHAND_RIGHT:
                                cout << "Keeper: Moving forward" << endl;
                                stateKeeper.moveForward();
                                cout << stateKeeper.getMainPos() << ":" <<
stateKeeper.getSubPos() << endl;
                                break;
                        default:
                                cout << "Unknown" << endl;
                        }
                }

                if(videoProcessor.isLeftGestureReady())
                {
                        Gesture gesture = videoProcessor.getLeftGesture();
                        cout << "GESTURE FOUND " << gesture.type << endl;
                        State* state = stateKeeper.GetState();
                        switch(gesture.type)
                        {
                        case Gesture::GESTURE_LHAND_MOVE:
                                if(state->getSubPos() == 1)
                                {
                                        cout << "Keeper: Moving up" << endl;
                                        stateKeeper.moveUp();
                                }
                                else if(state->getSubPos() == 0)
                                {
                                        cout << "Keeper: Moving down" << endl;
                                        stateKeeper.moveDown();
                                }
                                else
                                {
                                        cout << "Keeper: Executing action" << endl;
                                        stateKeeper.GetState()->func();
                                }
                                break;
                        default:
                                cout << "Unknown" << endl;
                                break;
                        }
                }

                //wyswietlenie stanu menu
                imshow(MENU_WINDOW_ID, stateKeeper.GetState()->getImg());

                //rysowanie podgladu i markerow na ramce
                ellipse( image, videoProcessor.getTrackBox(), Scalar(0,0,255), 3,
```

```
CV_AA );
                rectangle(image, videoProcessor.getSelection(), Scalar(255, 0, 0), 3,
CV_AA);
                rectangle(image, videoProcessor.getLeftSelection(), Scalar(0, 255, 0),
3, CV_AA);
                imshow(PREVIEW_WINDOW_ID, image);
                char c = (char)waitKey(5);
                if(c == 27)
                        break;
        }

        system("pause");
        return 0;
}
```

Zmiana stanu w maszynie stanowej

```
/*****************************************************************************/
/*
/*void StateKeeper::ChangeState
          /* Przeznaczenie
/* Funkcja zmienia obecny stan . Używana jest do zmiany pozycji w menu
/* kontrolowanym przez aplikację
/*  Argumenty funkcji:                                                       */
/*      (I)int value1 – oznacza pozycję w poziomie                           */
/*      (I)int value2 – oznacza pozycję opcji                                */
/* Funkcja zwraca void                                                       */
/* Używane zmienne:                                                          */
/*  vector<State> stany                                                      */
/*  int mainpos – określa obecną pozycję w poziomie                          */
/*  int subPos – określa obecną opcję                                        */
                                                                           */
/*****************************************************************************/

void StateKeeper::ChangeState(int value1, int value2)
{
// iteracja po wektorze stanów
        for(std::vector<State>::iterator it = stany.begin(); it != stany.end(); ++it)
        {
// sprawdzamy czy nowy stan istnieje
                if ( ( value1 == it->getMainPos() ) && ( value2 == it->getSubPos() ))
                {
// zmieniamy obecny stan na nowy
                        mainPos = value1;
                        subPos = value2;
                }

        }
}
```

9. DODATEK D. Spis zawartości dołączonych nośników (dyskietek, CD ROMu)

- Gest.avi – film źródłowy

- SRC - postacie źródłowe stworzonych procedur wraz z projektem, plik główny assistive.cpp
- DOC – Raport_ASSISTIVE.doc, Raport_ASSISTIVE.pdf, assistive.mp4 (przykładowe działanie program)