Baraniecki Karol Byczko Maciej	Prowadzący: Dr inż. Dominik Żelazny	Numer ćwiczenia laboratoria 12
PT 16:30 TP	Temat ćwiczenia: Obsługa kamery USB	Ocena:
Grupa: D	Data wykonania: 14 stycznia 2022	

1 Zagadnienia do opracowania

1. Znajomość podstawowych funkcji i zasad korzystania z WIN32 API (pojęcie HWND, tworzenie okien i ich obsługa, w szczególności GDI - HDC, funkcja BitBlt)
2. USB w Windows (standard USB, Interface HID - ogólnie)
3. Zasada działania kamery USB
4. Metody obsługi kamery USB (AVICAP32.DLL, TWAIN, WIA 1.0, WIA 2.0)
5. Sposób wykorzystania bibliotek DLL w aplikacji tworzonej w środowisku Visual Studio 2005 lub 2008
6. Poznanie API32 biblioteki AVICAP32.DLL (podstawowe funkcje i stałe)
7. Poznanie API do WIA

2 Zadania do wykonania

1. Korzystając z przykładowej aplikacji stwierdzić obecność i poprawność kamery podłączonej do portu USB komputera (aplikacja testowa)

Rysunek 1: Screenshot obrazu z kamery w aplikacji testowej

2. Połącz się z urządzeniem i za pomocą odpowiednich komunikatów łączących się z driverami kamery - skonfiguruj ją.

   - Za pomocą programu powinno dać się zmieniać opcje kamery (rozdzielczość obrazu, nasycenie, kontrast, ew. zoom, sterowanie kamera etc.)
     - a/A - Alpha +/-
     - b/B - Beta +/-
     - z/Z - Zoom +/-
   - Zapisz obraz z kamery w dowolnym formacie (wskazany JPG)

```
1 os.makedirs("screens", exist_ok=True)
2 filename = datetime.datetime.isoformat(datetime.datetime.now()).replace(":","")
3 cv.imwrite(f"screens/{filename}.png", frame)
```

   Za pomocą klawisza "s" robimy zdjęcie kamery i obraz zapisujemy w folderze screens, w nazwie pliku jest zawarta data oraz godzina zrobienia zdjęcia.

- Zapisz obraz z kamery w postaci filmu AVI

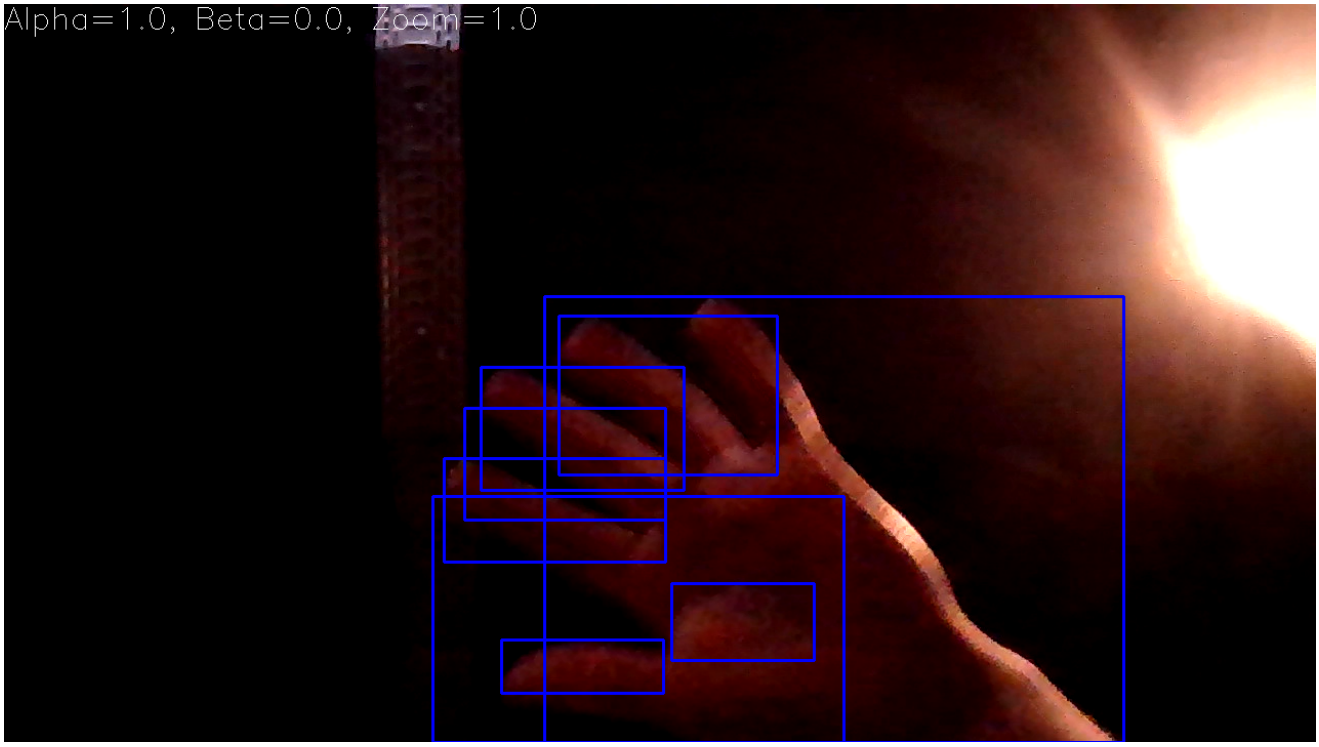
```
recording = False
recorder = cv.VideoWriter("Film.avi", 0, 20, (width, height))
```

Za pomocą klawisza "r" uruchamiamy nagrywanie, ponowne naciśnięcie klawisza "r" zatrzymuje nagrywanie.

3. Rozbuduj program o:

- stwórz prosty detektor ruchu - poprzez analizę obrazu z kamery w czasie rzeczywistym (wystarczy sprawdzać zmiany koloru kilku punktów (pikseli), ćwiczenie można rozwinąć o najprostsze algorytmy wykrywające krawędzie etc.)

```
# motion detection
if detect:
  if prev_frame is not None: # don't compare with empty frame

    # change to grayscale
    prev_frame = cv.cvtColor(prev_frame, cv.COLOR_BGR2GRAY)
    frame_copy = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

    # remove sharp edges
    prev_frame = cv.GaussianBlur(prev_frame, (21, 21), 0)
    frame_copy = cv.GaussianBlur(frame_copy, (21, 21), 0)

    # compare frames
    delta_frame = cv.absdiff(prev_frame, frame_copy)

    # mark different pixels
    thresh_frame = cv.threshold(delta_frame, 5, 255, cv.THRESH_BINARY)[1]
    cv.dilate(thresh_frame, None, iterations=2)

    contours, _ = cv.findContours(thresh_frame, cv.RETR_EXTERNAL, cv.
    CHAIN_APPROX_SIMPLE)

    # create rectangles on captured motion
    rectangles = []
    for c in contours:
      if cv.contourArea(c) < 1 / 750 * width * height:
        continue
      rectangles.append(cv.boundingRect(c))

    frame_copy = frame.copy()
    for rect in rectangles:
      x = rect[0]
      y = rect[1]
      w = rect[2]
      h = rect[3]
      cv.rectangle(frame_copy, (x, y), (x + w, y + h), (255, 0, 0), 2)
    cv.imshow(title, frame_copy)

prev_frame = frame
```



Rysunek 2: Przykładowe zdjęcie wykonane za pomocą programu wraz z wykryciem ruchu

3 Wnioski

Wszystkie podpunkty były w miarę proste ponieważ biblioteka OpenCV w pythonie oferuje wszystkie wymagane funkcjonalności jako funkcje do bezpośredniego użycia. Jedynie wykrywanie ruchu było trochę bardziej skomplikowane ponieważ wymagało od nas wykonania obliczeń, aby zmniejszyć ilość potrzebnej mocy obliczeniowej to zamieniliśmy obraz na skalę szarości i usunęliśmy ostre krawędzie aby wykrywanie ruchu było łatwiejsze, lecz nie jest to narzędzie idealne.