

# **Projekt monitoringu**

## **Technika Mikroprocesorowa 2013/2014**

**Autorzy:**

Cieśla Łukasz

Lichoń Tomasz



**A G H**

## **Spis treści:**

- 1. Cel**
- 2. Wykorzystany sprzęt**
- 3. Plan**
- 4. Harmonogram prac**
- 5. Idea działania programu**
- 6. Opis podstawowych funkcjonalności**
  - 1. Zbieranie danych z kamery**
  - 2. Wysyłanie maili z załącznikami filmowymi**
  - 3. Porównywanie zdjęć**
  - 4. Kompresja bitmap to JPEG**
  - 5. Kompresja JPEG to M-JPEG**
  - 6. Oczekiwanie na ruch**
  - 7. Nagrywanie i monitorowanie zmian**
  - 7. Wydajność**
  - 8. Wizja rozwoju**
  - 9. Podsumowanie**
- Bibliografia**

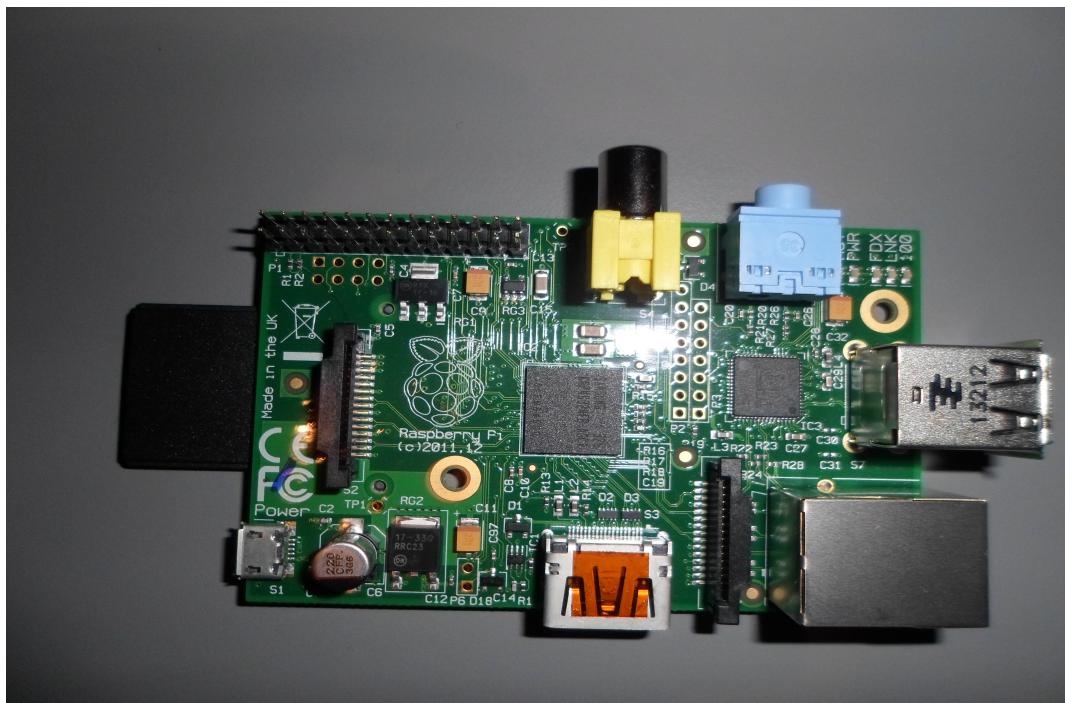
## **1. Cel:**

Projektowane urządzenie ma za zadanie pełnić rolę monitoringu antywłamaniowego. W momencie kiedy wykryje ono ruch w kadrze kamery, rozpocznie nagrywanie i prześle jego zapis na zdefiniowany adres mailowy. Po zniknięciu zauważonego wcześniej obiektu, urządzenie kończy nagrywanie i powraca do początkowego stanu oczekiwania, tak aby na skrzynkę elektroniczną trafiały wyłącznie istotne nagrania.

## **2. Wykorzystany sprzęt:**

Projekt działa w oparciu o komputer Raspberry Pi model B rev 2.0, wyposażony w złącze Ethernet i port USB do którego podłączono kamerę internetową, może być to dowolna kamera kompatybilna z Raspberry, listę takich kamer można znaleźć na stronie [http://elinux.org/RPi\\_USB\\_Webcams](http://elinux.org/RPi_USB_Webcams). W naszym przypadku była to „Playstation Eye for PS2”. Rolę pamięci pełni karta SD.

Wyszczególniony komputer został wybrany głównie ze względu na niewielki rozmiar, stosunkowo niską cenę, oraz łatwość integracji z zewnętrznymi urządzeniami (jak np. kamera USB).



Rys1. Komputer Raspberry Pi v2



Rys2. Kamerka internetowa Playstation Eye for PS2

### 3. Plan:

System jest programowany w języku C. Podstawę stanowi algorytm porównujący 2 zdjęcia zrobione w pewnym odstępie czasu i decydujący, czy zmiana wyglądu pomieszczenia jest na tyle istotna aby rozpocząć nagrywanie (lub też zakończyć gdy nie dzieje się już nic ciekawego). Kolejny etap to nagrywanie widoku z kamery, kompresja obrazu do formatu jpeg, po czym składanie klatek do avi i przesyłanie fragmentów nagrań pod wskazany adres e-mail. Duże wyzwanie stanowi optymalne zaprojektowanie i zbalansowanie powyższych algorytmów, tak aby dysponowana moc obliczeniowa oraz przepustowość łącza pozwoliły na szybkie wysyłanie i nagrywanie obrazu (co jest konieczne ze względu na ograniczenia pamięciowe).

## **4. Harmonogram prac:**

- Zakup sprzętu
- Integracja wszystkich podzespołów
- Przygotowanie repozytorium pod kody źródłowe
- Przegląd literatury w celu zdobycia informacji na temat przetwarzania obrazu i zaprojektowanie głównego algorytmu
- Przegląd dostępnych bibliotek języka C pozwalających na pozyskiwanie bitmapy z matrycy kamery
- Implementacja i testy zasadniczej funkcjonalności:
  - metody pozwalające na obsługę kamery (takePhoto(), startRecording(), stopRecording())
  - funkcja logiczna porównująca obrazy i decydująca o nagrywaniu (isDifferent(photo1, photo2))
  - implementacja algorytmu kompresji do formatu jpeg
  - implementacja algorytmu kompresji do formatu avi
  - przetestowanie poprawności powyższych funkcji
- Scalenie stworzonych funkcji w główny program
- Dobór odpowiednich parametrów programu, tak aby wykrywać tylko istotne ruchy w otoczeniu
- Integracja i analiza wydajności całości projektu
- Dokonanie niezbędnych zmian w celu poprawy wydajności, tak aby system mógł działać w czasie rzeczywistym
- Wizja rozwoju projektu

## **5. Idea działania programu**

Zadaniem kamery jest robienie zdjęć do analizy i nagrywanie filmów w momencie wykrycia ruchu. O częstotliwości tworzonych zdjęć decyduje główny algorytm. Do pobierania danych z kamery użyto biblioteki OpenCV2. Bitmapy będą pobierane z kamery i dzięki zaimplementowanemu algorytmowi kompresji jpeg będą zapisywane w folderze */tmp/pictures*, skąd później pobierać je będzie moduł odpowiedzialny za kompresję avi i zamieni on je na film który zapisze w katalogu */tmp/videos*, po skończonym zapisie film zostanie wysłany na skrzynkę mailową, a program powróci do normalnej pracy.

## **6. Opis podstawowych funkcjonalności**

- Zbieranie danych z kamery**

*IplImage\* takePicture();*

Pobieranie danych z kamery opakowane jest w powyższą funkcję. Korzysta ona bezpośrednio z biblioteki OpenCV przez co jest bardzo wydajna i pozwala na sczytywanie nawet do kilkudziesięciu klatek na sekundę. Zwraca ona strukturę IplImage, która zawiera w sobie surowe dane w postaci RGB, oraz rozmiar zdjęcia. W kwestii użycia OpenCV do pobrania danych pomocny okazał się przykład ze strony <http://ceit.uq.edu.au/content/quick-example-video-capture-using-opencv>

- Wysyłanie maili z załącznikami filmowymi**

*void sendVideo(char\* path);*

Funkcjonalność związana z wysyłaniem filmów zaszyta została w funkcji sendVideo, która jako argument przyjmuje ścieżkę do filmu który chcemy wysłać, wysyła go ona

na maila który został stworzony na potrzeby projektu i na stałe wpisany w pliku nagłówkowym email.h (adres może być zamieniony na dowolny inny). Przesyłanie maili odbywa się z pomocą programu mutt, w naszym przypadku skonfigurowano go do współpracy z serwerem poczty gmail, w oparciu o następujący tutorial <http://lifehacker.com/5574557/how-to-use-the-fast-and-powerful-mutt-email-client-with-gmail>, mutt wymaga stworzenia folderów `~/.mutt/cache/headers`, `~/.mutt/cache/bodies`, `~/.mutt/certificates`, oraz obecności pliku konfiguracyjnego `.muttrc`. Tworzenie folderów i skopiowanie pliku konfiguracyjnego odbywa się w funkcji init modułu email która wywoływana jest po starcie programu. Gdy init wykona się poprawnie można w bardzo prosty sposób wysłać dowolny film używając z poziomu programu polecenia:

```
system(echo \"title\" | mutt -s \"msg\" -a attachementPath -- receiver@address)
```

## ● Porównywanie zdjęć

```
int isDifferent(char* first, char* second)
```

Funkcja porównująca 2 zdjęcia przyjmuje wskaźniki na dane obu z nich (składające się z wysokość\*szerokość\*3 bajtów) i zwraca 1 jeśli są według niej różne, a 0 w przeciwnym przypadku. Dodatkowo wypisuje na standardowe wyjście informacje diagnostyczne o tym, w ilu procentach różniły się przekazane jej zdjęcia.

Sam algorytm jest bardzo prosty, przebiega ona po wszystkich pikselach i zlicza pary które różnią się o co najmniej PIXEL\_VALUE\_EPSILON, następnie wylicza jaki procent całości pikseli one stanowią, i jeśli jest on większy niż MINIMAL\_DIFFERENCE\_PERCENTAGE to zwraca 1. Podane tutaj Parametry zdefiniowane zostały w pliku nagłówkowym basicWebcamFunctionality.h. Podczas testów najlepiej sprawdzały się PIXEL\_VALUE\_EPSILON=30, MINIMAL\_DIFFERENCE\_PERCENTAGE=10. Są one jednak mocno zależne od konkretnej kamery, jakości światła oraz tego jak duży obszar ma być monitorowany,

dlatego zalecamy indywidualne dobieranie wyżej wymienionych wartości, zależnie od przypadku.

## ● Kompresja bitmap to JPEG

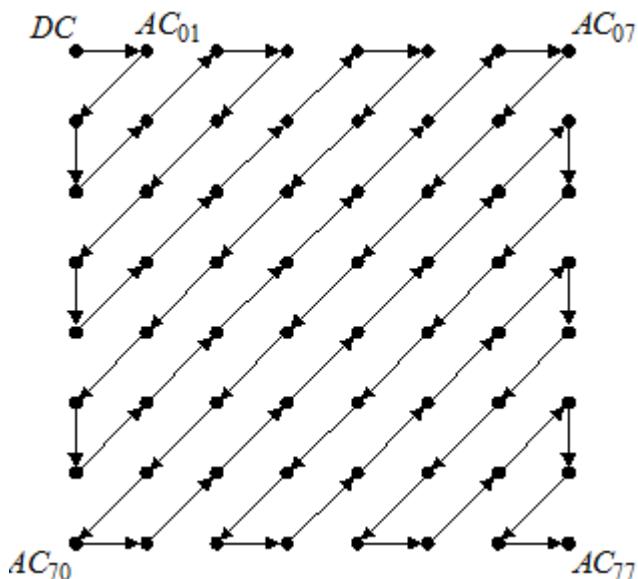
```
void jpegEncoderInit();  
void jpegEncoderTeardown();  
void storeToFile(IplImage* image,char* path);
```

Cały kod związany z kompresją został umieszczony w podkatalogu BitmapToJpeg, którego głównym plikiem udostępniającym powyższy interfejs naszemu programowi jest *BitmapToJpeg.c*. JpegEncoderInit odpowiada za alokację niezbędnych buforów, które zwalniane poprzez jpegEncoderTeardown. Funkcja *storeToFile()* jest używana przez główny program podczas nagrywania filmu, cyklicznie wywołującą on ją dla kolejnych bitmap aby zapisać je na dysku pod wskazaną nazwą w formacie jpeg (w celu późniejszego przetworzenia ich na film).

Aby stworzyć plik niezbędne jest uzupełnienie struktury nagłówka o pola charakteryzujące konkretne zdjęcie, jak na przykład wysokość i szerokość. Po stworzeniu nagłówków przechodzimy do właściwej części. Kompresja JPEG jest algorytmem stratnym (oznacza to, że podczas działania programu pewna część pierwotnych danych nie pozwoli się nigdy odzyskać). Głównymi krokami algorytmu, zaimplementowanymi w jpegEncoderze są:

- skonwertowanie RGB na luminację (intuicyjnie jasność) i 2 kanały barw. Warto wspomnieć, że dzieje się tak ponieważ ludzkie oko lepiej dostrzega niewielkie zmiany jasności niż barw. Następuje redukcja części pikseli kanałów barwy.
- kanały dzielimy na kwadratowe bloki o boku 8, po czym wykonujemy na nich dyskretną transformację kosinusową (DSC). W jej wyniku otrzymujemy częstotliwość zmian wewnętrz ka dego bloku oraz średni  warto  ka dego z nich. Obie z powy szych liczb s  zmiennoprzecinkowe. (W tym momencie nie tracimy żadnych danych, g d  proces mo na odwróci )

- Aby poprawić współczynnik kompresji zastępujemy dla każdego bloku jego średnią wartość przez różnicę wobec poprzedniej wartości, po czym następuje zamiana liczb zmiennoprzecinkowych na całkowite (na tym etapie bezpowrotnie tracimy część danych).
- Współczynniki transformaty porządkujemy zygzakowo, zamieniając macierz na wektor (jak na rysunku poniżej) po czym kompresujemy wartości niezerowe algorytmem Huffmana.



Rys3. Zig-zag order

## ● Kompresja JPEG to M-JPEG

```
void makeMovie(int amount, char* picturesPath, char* videoPath, int fps);
```

Całość funkcjonalności potrzebnej głównemu programowi zawiera powyższa funkcja, jako argument przyjmuje ona liczbę zdjęć z których należy stworzyć film, katalog w którym się one znajdują(mają nazwy „0”, „1” itd.. aż do amount), ścieżkę pod którą należy zapisać stworzony film, oraz liczba klatek na sekundę z jaką powinno zostać zapisane nagranie. Film w formacie avi zapisujemy dzieląc dane na części, każda z nich posiada swój identyfikator. Każdy z fragmentów składa się z

dwóch składowych: hdrl – nagłówek zawierający metadane mówiące o pliku video (rozmiar obrazu, ilość klatek) oraz movi – czyli właściwe dane wizualne. Zapis danych umożliwia proces kodowania, ponadto wykorzystujemy MJPEG jako format kompresji (aby łatwo przejść od stworzonych zdjęć do końcowego filmu). W formacie tym każda klatka skompresowana jest jako oddzielny obraz JPEG, z których każdy kolejny wyświetlany jest po poprzedniku po upłynięciu odpowiedniej przerwy. Ważne jest też to, że w formacie mjepg każda klatka posiada osobną kompresję niezależną od pozostałych, przez co cały algorytm nawet dla bardzo długich filmów nie wymaga wygórowanej pamięci operacyjnej, co jest istotne w przypadku jeśli uruchamiamy program na raspberry pi, lub tym bardziej (w przyszłości) na procesorze jednoukładowym.

## ● Oczekiwanie na ruch

Po uruchomieniu program wchodzi w nieskończoną pętlę wykrywania ruchu. Robi on zdjęcia cyklicznie co pewien interwał czasu (podczas testów najlepszą wartością interwału okazało się być 200ms). Jeśli w którymkolwiek momencie 2 występujące zdjęcia różnią się od siebie (według funkcji isDifferent), program rozpoczyna nagrywanie i wchodzi w kolejną pętlę związaną z nagrywaniem.

## ● Nagrywanie i monitorowanie zmian

Podczas nagrywania podobnie jak w oczekiwaniu na ruch robimy cyklicznie zdjęcia, z tym że każde z nich zapisujemy na karcie pamięci w celu późniejszego skonwertowania ich do filmu. Oprócz tego system cały czas monitoruje czy obraz nadal się zmienia, robi to jednak z dużo mniejszą częstotliwością. Analizowane jest tylko co któreś zdjęcie, zależnie od zdefiniowanego parametru NTH\_FRAME\_OF\_RECORD, obecnie wartość ta ustawiona jest na 2. Jeśli kolejnych EQL\_IN\_ROW\_TO\_STOP\_RECORDING (domyślnie 5) zdjęć z rzędu nie różni się znacząco, przerywamy nagrywanie, ze stworzonych zdjęć robimy film, który następnie wysyłamy mailem.

## **7. Wydajność**

Wydajność komputera raspberry pi nie pozwala na zapis i konwersję szybszą niż 2 zdjęcia na sekundę, co przekłada się na niewielki fps tworzonego filmu, nie jest to jednak problemem ponieważ kamery bezpieczeństwa kładą większy nacisk na niewielki rozmiar niż na jakość nagrania. Dodatkowym czynnikiem ograniczającym jest wysyłanie filmu, które zatrzymuje główną pętlę na pewien czas, jest to związane z połączeniem z serwerem pocztowym i uploadem filmu, może jednak ono być łatwo zamienione na zapis w dołączonej pamięci fizycznej.

## **8. Wizja rozwoju**

W pierwszej kolejności należy pomyśleć nad cache'owaniem tworzonych zdjęć w celu zwiększenia wydajności podczas nagrywania (zapis każdego na karcie pamięci jest zbyt czasochłonny). Dodatkowo należałoby pomyśleć nad zmianą poczty elektronicznej na inne miejsce docelowe dla filmów, jak na przykład dedykowany serwer z szybkim łączem pozwalającym na uniknięcie opóźnień podczas uploadu plików. Ponadto system jest łatwo skalowalny na większą liczbę kamer, każda z nich może posiadać swój niezależny kontroler zainstalowanym naszym systemem.

## **9. Podsumowanie**

Realizacja projektu okazała się trudnym lecz interesującym wyzwaniem, całość implementacji poza pozyskiwaniem zdjęć i obsługą poczty napisana została w czystym C bez użycia żadnych bibliotek zewnętrznych, przez co jest przenośna na wszelkiego rodzaju urządzenie posiadające możliwość zaprogramowania w języku C, w tym układy FPGA.

## Bibliografia:

- [http://elinux.org/RPi\\_USB\\_Webcams](http://elinux.org/RPi_USB_Webcams)
- <http://malinowepi.pl/>
- <http://opencv.org/>
- <http://ceit.uq.edu.au/content/quick-example-video-capture-using-opencv>
- <http://lifehacker.com/5574557/how-to-use-the-fast-and-powerful-mutt-email-client-with-gmail>
- <http://en.wikipedia.org/wiki/JPEG>
- <http://en.wikipedia.org/wiki/Mjpeg>