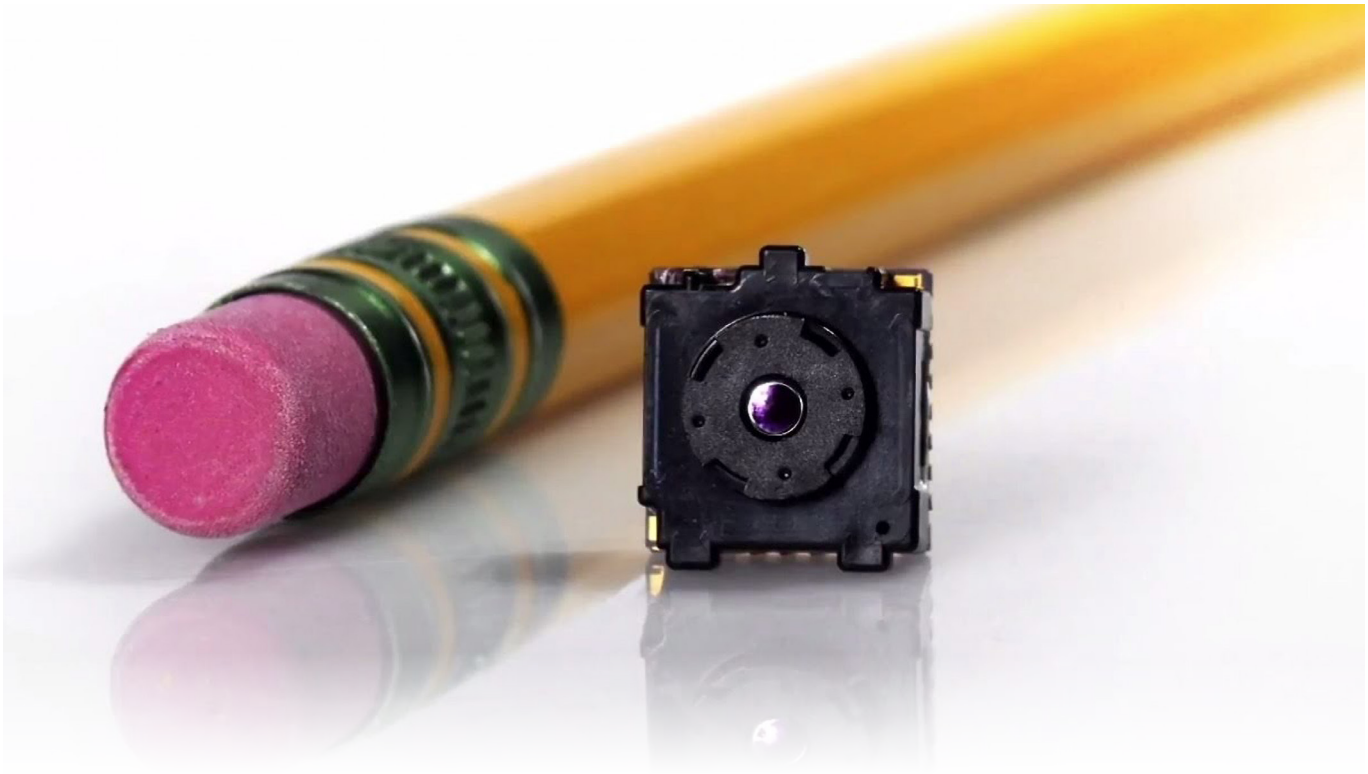


Kamera termiczna oparta na Raspberry Pi

Obserwacja świata w podczerwieni to fascynująca sprawa. Niestety nasze oczy nie rejestrują tego fragmentu pasma promieniowania elektromagnetycznego, więc skazani jesteśmy na wykorzystywanie specjalnych kamer. Urządzenia te są zazwyczaj dosyć drogie, dlatego też studenci z politechniki w Michigan postanowili skonstruować własne, otwarte urządzenie tego typu. Grupa kierowana przez dr. Joshue Pearce'a opracowała otwarty projekt kamery termalnej, wykorzystującej komputer jednopłytkowy Raspberry Pi i moduł kamery podczerwonej FLIR Lepton.

Obrazowanie rozkładu temperatury – termografia – ma wiele zastosowań w różnych dziedzinach: przemyśle, nauce czy medycynie. Termografia polega na obrazowaniu w zakresie średniej podczerwieni, to jest od 9 do 14 μm. Każde ciało emituje promieniowanie elektromagnetyczne, najczęściej w zakresie podczerwieni, o długości fali odwrotnie proporcjonalnej do jego temperatury. Emisja z maksimum przypadającym na zakres średniej podczerwieni odpowiada temperaturom najczęściej spotykanym w naszym otoczeniu.

Urządzenia do obrazowania w podczerwieni są zazwyczaj bardzo drogie. Aby jednak skorzystać z tej technologii na uczelni, zespół dr. Pearce'a skonstruował własną kamerę termiczną na bazie modułu kamery podczerwonej Lepton firmy FLIR. Jest to ekonomiczny moduł do zastosowań w sprzęcie przenośnym i konsumenckim.

Elementy składowe układu

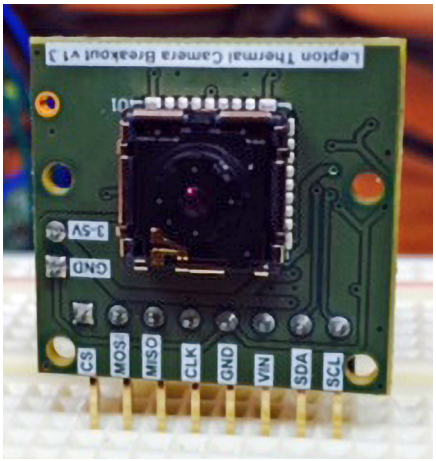
Autorzy użyli pierwszej generacji komputerka Raspberry Pi – wersji B+, do budowy opisanego urządzenia. Jednak każda inna wersja Raspberry będzie odpowiednia do tej kamery i napisanego oprogramowania, wymagając jedynie minimalnych poprawek. Wszystkie elementy składowe zebrano w **tabeli 1.**

Przygotowywanie karty SD

Na początek musimy przygotować odpowiednio kartę SD dla komputera Raspberry Pi, aby mógł poprawnie działać. Do tego potrzebny będzie nam zwykły komputer PC, wyposażony w czytnik kart SD. W pierwszej kolejności musimy sformatować kartę i umieścić na niej instalator systemu operacyjnego dla Raspberry Pi. Do formatowania karty możemy użyć jednego z dedykowanych do tego celu programów. Autor konstrukcji wskazuje program Formatter for Windows i opis w artykule dotyczy tego programu.

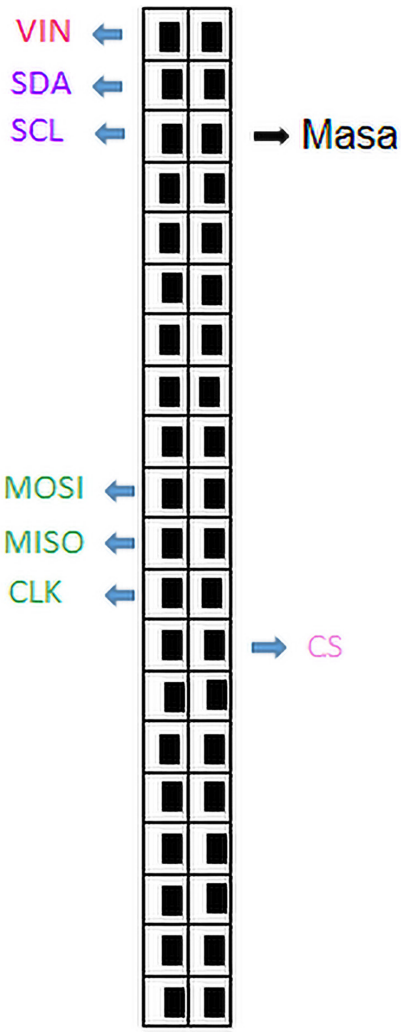
Po pobraniu i zainstalowaniu programu możemy w komputerze umieścić kartę SD i uruchomić zainstalowany program. Następnie wybieramy opcję FORMAT SIZE ADJUSTMENT, czyli formatowanie z dostosowaniem wielkości karty SD. Jeśli opcja jest zaznaczona, to możemy kliknąć OK i przejść dalej. W polu „Drive" wybieramy literę dysku skojarzoną z naszą kartą SD. Jeśli w komputerze zainstalowany mamy tylko jeden czytnik kart, z jedną kartą, to oprogramowanie automatycznie wykryje literę dysku, pod którą znajduje się karta. Po ustawieniu wszystkich parametrów możemy nacisnąć przycisk Format i potwierdzić to, naciskając OK w kolejnych okienkach, które będą się pojawiać. Po chwili oczekiwania karta microSD zostanie sformatowana.

Kolejnym krokiem jest instalacja systemu operacyjnego z wykorzystaniem pakietu NOOBS. Należy pobrać najnowszą wersję



Rysunek 1. Moduł FLIR Lepton na płytce z wyprowadzeniami wszystkich sygnałów

pakietu w postaci pliku ZIP (w chwili pisania artykułu jest to 3.0.1). NOOBS najprościej pobrać z oficjalnej witryny ***www.raspberrypi.org***. Po pobraniu paczki zawartość musimy wypakować na czystą kartę microSD, którą sformatowaliśmy w poprzednim kroku. Możemy teraz włożyć kartę microSD do Raspberry Pi, podłączyć myszkę i klawiaturę oraz ekran (np. telewizor poprzez port HDMI). Po uruchomieniu systemu postępujemy zgodnie z poleceniami na ekranie, instalując system operacyjny Raspbian.



Rysunek 2. Schemat wyprowadzeń 40-pinowego złącza Raspberry Pi. Oznaczono piny, które trzeba podłączyć do modułu kamery

Tabela 1. Elementy potrzebne do budowy kamery termicznej z modułem FLIR Lepton

Moduł kamery termowizyjnej	FLIR Lepton
Raspberry Pi	Model B+ lub wyższy
Zasilacz 5 V	Z wtyczką microUSB i wydajnością prądową nie mniejszą niż 700 mA
Karta microSD	Minimalna pojemność 8 GB, rekomendowane 32 GB, jeśli chcemy nagrywać więcej filmów w podczerwieni. Karta powinna być co najmniej klasy 4
Klawiatura	Z USB
Myszka	Z USB
Kabel HDMI	
Połączenie z siecią	Kablowe połączenie Ethernet lub połączenie bezprzewodowe, po dołożeniu do komputera karty Wi-Fi na USB

Podłączenie FLIR Lepton do Raspberry Pi

Kolejnym krokiem jest podłączenie do komputera modułu kamery FLIR Lepton. Warto kupić moduł Lepton w postaci modułu deweloperskiego – od razu przylutowanego do płytki drukowanej, z wyprowadzonymi wszystkimi potrzebnymi sygnałami na złączach szpilkowych. W ten sposób możemy połączyć moduł kamery z komputerem przy użyciu płytki stykowej lub zworek. Moduł z kamerą FLIR Lepton pokazano na **rysunku 1.** Moduł z kamerą potrzebuje do komunikacji dwóch interfejsów szeregowych – SPI do przesyłania danych oraz I²C do przesyłania informacji kontrolnych. Komputer Raspberry Pi wyposażony jest w oba te interfejsy oraz duże wsparcie programistyczne do tworzenia oprogramowania w postaci gotowych bibliotek do języków takich jak C i Python.

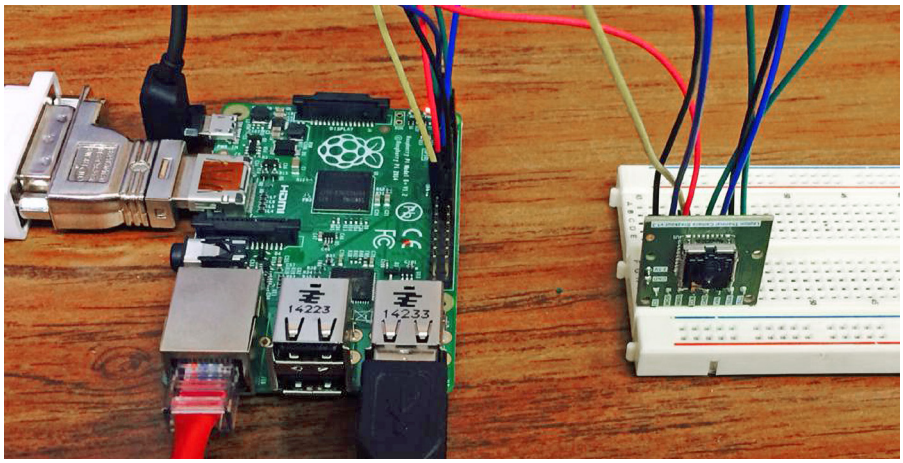
Raspberry Pi ma 40-pinowe złącze, na którym wyprowadzono m.in. linie interfejsów SPI oraz I²C, a także zasilanie i masę. Na **rysunku 2** pokazano, gdzie znajdziemy poszczególne sygnały SPI (MOSI, MISO, CLK i CS) oraz I²C (SDA i SCL). Łączymy je w wybrany przez siebie sposób z odpowiadającymi sygnałami po stronie modułu z kamerą termowizyjną. Pamiętajmy, aby dołączać moduł do kamery po odłączeniu zasilacza od Raspberry Pi. Na **rysunku 3** pokazano podłączony moduł z kamerą na płytce stykowej. Po podłączeniu wszystkiego do komputera Raspberry Pi możemy ponownie uruchomić komputer (podłączyć zasilanie do portu microUSB) i przejść do konfiguracji.

W pierwszej kolejności musimy podłączyć Internet – poprzez kabel (Ethernet) lub Wi-Fi. W zależności od tego, w jaki sposób podłączamy nasz moduł Raspberry Pi do Internetu, sposób konfiguracji będzie inny. Po podłączeniu do sieci musimy zaktualizować wszystkie komponenty systemu. W tym celu należy uruchomić terminal i po kolei wpisać komendy:

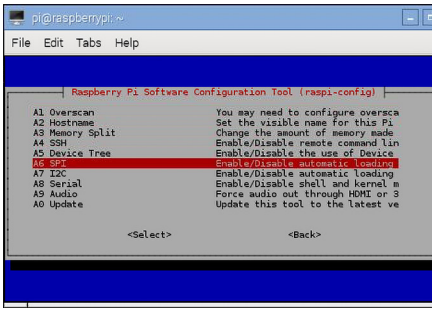
```
sudo apt-get update
sudo apt-get dist-upgrade
```

Pozwoli to zaktualizować informację na temat najnowszych wersji pakietów w programie APT, który służy do instalacji oprogramowania na systemie Raspbian, a następnie zaktualizować wszystkie pakiety w naszym systemie operacyjnym do najaktualniejszych wersji. Dla zachowania bezpieczeństwa systemu powinniśmy co jakiś czas taką czynność powtarzać.

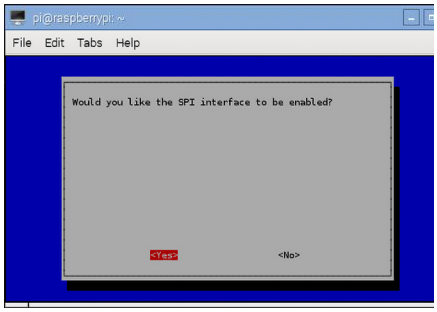
Kolejnym krokiem, po aktualizacji systemu, jest konfiguracja sprzętowych portów Raspberry Pi – SPI oraz I²C. Domyślnie,



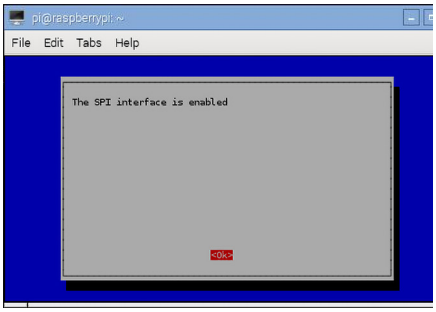
Fotografia 3. Moduł FLIR Lepton na płytce stykowej, połączony z Raspberry Pi



Rysunek 4a. Opcja SPI w raspi-config



Rysunek 4b. Okno potwierdzenia uruchomienia interfejsu SPI



Rysunek 4c. Okno z informacją o uruchomieniu SPI

po instalacji świeżej kopii systemu operacyjnego interfejsy te są wyłączone. Aby je uruchomić, w terminalu wpisujemy:

```
sudo raspi-config
```

Uruchomi się narzędzie do konfiguracji naszego komputera. Wybieramy „Advanced Options" a tam wybieramy SPI, potwierdzamy, naciskając YES i klikamy w kolejnym oknie OK (patrz **rysunek 4**). Następnie potwierdzamy chęć automatycznego ładowania modułu jądra ze sterownikiem interfejsu SPI i potwierdzamy nasz wybór (patrz **rysunek 5**). Analogicznie postępujemy z ustawieniami dla I²C. Teraz w menu głównym konfiguratora wybieramy opcję *finish* i resetujemy komputer (konfigurator zapyta nas, czy chcemy go ponownie uruchomić).

Instalacja potrzebnego oprogramowania

Po skonfigurowaniu komputera możemy przystąpić do instalacji potrzebnego oprogramowania. W pierwszej kolejności musimy zainstalować bibliotekę Qt wraz z zestawem narzędzi deweloperskich do tworzenia graficznego interfejsu użytkownika. Aby zainstalować pakiet narzędzi deweloperskich, w terminalu wpisujemy komendę:

```
sudo apt-get install qt4-dev-tools
```

Podczas instalacji zostaniemy zapytani, czy chcemy ją kontynuować. Potwierdzamy, naciskając klawisz „y", co pozwoli APT pobrać i następnie zainstalować ten pakiet wraz z wszystkimi jego zależnościami i wymaganymi bibliotekami. Kolejnym krokiem jest pobranie oprogramowania kamery, któremu przyjrzymy się dokładniej w dalszej części artykułu. Oprogramowanie, w postaci pliku .zip możemy pobrać z repozytorium na GitHubie. Archiwum należy wypakować, by uzyskać dostęp do plików. W tym celu, w folderze, w którym chcemy trzymać pliki tego projektu, wprowadzamy w terminalu następujące komendy:

```
wget https://github.com/groupgets/
LeptonModule/archive/master.zip
unzip master.zip
```

Możemy także skorzystać z wbudowanego w system klienta gita i sklonować repozytorium na Raspberry Pi, wpisując w docelowym folderze w terminalu komendę:

```
git clone https://github.com/
groupgets/LeptonModule.git
```

Następnie możemy przejść do folderu *master/raspberrypi_video/leptonSDKEmb32PUB* i skompilować gotowy program, wpisując w terminalu po kolei:

```
make
cd /home/pi/master/
raspberrypi_video
qmake && make
sudo ./raspberrypi_video
```

gdzie */home/pi/master/raspberrypi_video* to folder, w którym znajduje się pobrane przez nas wcześniej repozytorium. W ten sposób skompilowaliśmy i uruchomiliśmy oprogramowanie dedykowane do obsługi kamery FLIR Lepton. Jednakże, jako profesjonaliści, musimy zajrzeć do środka kodu i przeanalizować najważniejsze jego bloki, aby zrozumieć, jak działa ten software.

Listing 1. Plik *main.cpp*

```
#include <QApplication>
#include <QThread>
#include <QMutex>
#include <QMessageBox>

#include <QColor>
#include <QLabel>
#include <QtDebug>
#include <QString>
#include <QPushButton>

#include „LeptonThread.h"
#include „MyLabel.h"

int main( int argc, char **argv )
{
          // utworzenie nowej aplikacji
          QApplication a( argc, argv );

          QWidget *myWidget = new Qwidget;
          // ustawienie pozycji (400, 300) i wielkości (340, 290) okna
          myWidget->setGeometry(400, 300, 340, 290);

          // utworzenie miejsca na obraz
          QImage myImage;
          myImage = QImage(320, 240, QImage::Format_RGB888);
          QRgb red = qRgb(255,0,0); // definicja koloru czerwonego w RGB
          // górny lewy róg aplikacji wypełniony jest na czerwono
          for(int i=0;i<80;i++) {
                    for(int j=0;j<60;j++) {
                              myImage.setPixel(i, j, red);
                    }
          }

          // utworzenie nowej etykiety i ustawienie jej obrazu w oknie
          MyLabel myLabel(myWidget);
          //  ustawienie pozycji (10, 10) i wielkości (320, 240) w oknie
          myLabel.setGeometry(10, 10, 320, 240);
          myLabel.setPixmap(QPixmap::fromImage(myImage));

          // utworzenie przycisku FFC
          QPushButton *button1 = new QPushButton(„Perform FFC", myWidget);
          button1->setGeometry(320/2-50, 290-35, 100, 30);

          // utworzenie osobnego wątku do zbierania danych po SPI
          // gdy wątek zada sygnał updateImage, etykieta zaktualizuje się
          LeptonThread *thread = new LeptonThread();
          QObject::connect(thread, SIGNAL(updateImage(QImage)), &myLabel,
          SLOT(setImage(QImage)));

          // połączenie przycisku FFC z wątkiem wykonującym akcję
          QObject::connect(button1, SIGNAL(clicked()), thread, SLOT(performFFC()));
          thread->start();

          myWidget->show();

          return a.exec();
}
```

Kod programu do obsługi kamery FLIR Lepton

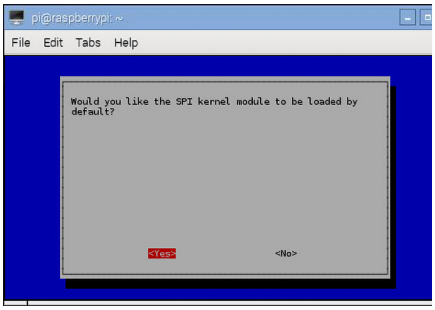
Pierwszym plikiem, jakiemu warto się przyjrzeć, jest *main.cpp*, zaprezentowany na **listingu 1**. Program korzysta z dwóch plików nagłówkowych *LeptonThread.h* oraz *MyLabel.h*, oprócz oczywiście plików nagłówkowych samego Qt. W pierwszym z nich znajduje się kod obsługujący kamerę Lepton – definicja wątku pobierającego dane z kamery, a także funkcje, pozwalające

na wydawaniu urządzeniu komend, takich jak wykonanie FFC. W drugim pliku natomiast znajduje się definicja etykiety Qt z obrazem, co umożliwia jego prezentację, jednakże analiza tego pliku nie jest tutaj niezbędna, gdyż to, co nas najbardziej interesuje, to komunikacja z kamerą – framework i sposób prezentacji danych, to kwestia wtórna.

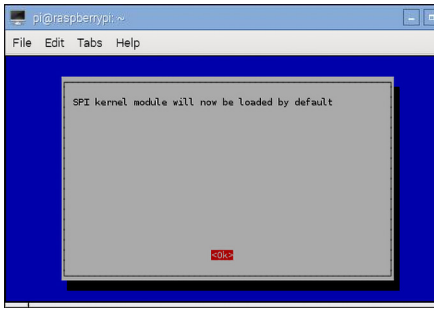
Przesył danych z kamery odbywa się poprzez interfejs SPI. W pliku *LeptonThread.cpp* znajduje się m.in. definicja metody *run()* z obiektu wątku do obsługi kamery Lepton. Definicję tej funkcji pokazano na **listingu 2**. Metoda *run()* po uruchomieniu inicjalizuje interfejs SPI, a następnie odczytuje pewną liczbę pakietów (PACKETS_PER_FRAME zdefiniowane w pliku jako 60). W przypadku wystąpienia tzw. drop packet, informującego o zatrzymaniu lub braku transmisji, licznik pakietów zostaje zresetowany do zera. 750 takich resetów powoduje reset całego systemu, włącznie z ponowną inicjalizacją portu SPI w układzie. Po poprawnym odebraniu wszystkich pakietów ramki wektor *results* przekazywany jest do dalszej funkcji, która ustawia każdy piksel obrazka *myImage* metodą *setPixel(kolumna, rząd, kolor)*. Jednocześnie obrazek jest normalizowany tak, aby optymalnie pokryć całą przestrzeń barwną. Iterując po wszystkich pikselach, algorytm odnajduje najmniejszą i największą wartość z kamery (odpowiednio *minValue* i *maxValue*). Na podstawie tych wartości wyznaczana jest rozpiętość temperatury (*diff* = *maxValue* – *minValue*), która po podzieleniu przez nią liczby 255 (liczba kolorów w obrazie) daje

Listing 2. Definicja metody *run()*

```
void LeptonThread::run()
{
	// utwórz początkowy obraz
	myImage = QImage(80, 60, QImage::Format_RGB888);

	// otwórz port SPI
	SpiOpenPort(0);

	while(true) {

		// odczytywanie pakietów danych z kamery poprzez SPI
		int resets = 0;
		for(int j=0;j<PACKETS_PER_FRAME;j++) {
			// jeżeli odebrany pakiet jest typu „drop”, musimy zresetować komunikację oraz
			// ustawić j = -1, aby w kolejnej iteracji po inkrementacji było znowu zerem
			read(spi_cs0_fd, result+sizeof(uint8_t)*PACKET_SIZE*j, sizeof(uint8_t)*PACKET_SIZE);
			int packetNumber = result[j*PACKET_SIZE+1];
			if(packetNumber != j) {
				j = -1;
				resets += 1;
				usleep(1000);
				// 750 resetów to arbitralny limit, jako że nie powinno się nigdy wydarzyć, aby
				// przerwa w transmisji była tak długa dla tej częstotliwości czytania z kamery
				// Czytając z kamery częściej, okazać się może, że limit ten zostanie jednakże
				// łatwo przekroczony, co zostanie od razu oznaczone jako utrata synchronizacji
				if(resets == 750) {
					SpiClosePort(0);
					usleep(750000);
					SpiOpenPort(0);
				}
			}
		}
		if(resets >= 30) {
			qDebug() << „Zakończono odczyt, resetuje: „ << resets;
		}

		frameBuffer = (uint16_t *)result;
		int row, column;
		uint16_t value;
		uint16_t minValue = 65535;
		uint16_t maxValue = 0;

		for(int i=0;i<FRAME_SIZE_UINT16;i++) {
			// pomijamy pierwsze 4 bajty każdej ramki (to nagłówekj) - są to dwie wartości typu uint16_t
			if(i % PACKET_SIZE_UINT16 < 2) {
				continue;
			}

			// zamiana MSB z LSB
			int temp = result[i*2];
			result[i*2] = result[i*2+1];
			result[i*2+1] = temp;

			value = frameBuffer[i];
			if(value > maxValue) {
				maxValue = value;
			}
			if(value < minValue) {
				minValue = value;
			}
			column = i % PACKET_SIZE_UINT16 - 2;
			row = i / PACKET_SIZE_UINT16 ;
		}

		float diff = maxValue - minValue; // rozpiętość wartości
		float scale = 255/diff; // skala do normalizacji danych na obrazie
		QRgb color;
		for(int i=0;i<FRAME_SIZE_UINT16;i++) {
			if(i % PACKET_SIZE_UINT16 < 2) {
				continue;
			}
			value = (frameBuffer[i] - minValue) * scale;
			const int *colormap = colormap_ironblack; // mapa kolorów wykorzystana do obrazka
			color = qRgb(colormap[3*value], colormap[3*value+1], colormap[3*value+2]);
			column = (i % PACKET_SIZE_UINT16 ) - 2;
			row = i / PACKET_SIZE_UINT16;
			myImage.setPixel(column, row, color);
		}

		// emisja sygnału dla interfejsu w Qt, który mówi, że można aktualizować obrazek
		emit updateImage(myImage);

	}

	// finalnie - zamknięcie portu SPI
	SpiClosePort(0);
}
```



Rysunek 5a. Okno z pytaniem o uruchamianie modułu jądra z obsługą SPI przy starcie systemu



Rysunek 5b. Potwierdzenie domyślnego ładowania modułu

Listing 3. Definicja m.in. funkcji *lepton_perform_ffc()* przy wykorzystaniu funkcji z SDK FLIR Lepton

```
#include „Lepton_I2C.h”
// blbioteki oficjalnego SDK do kamer Lepton
#include „leptonSDKEmb32PUB/LEPTON_SDK.h”
#include „leptonSDKEmb32PUB/LEPTON_SYS.h”
#include „leptonSDKEmb32PUB/LEPTON_Types.h”

bool _connected;

LEP_CAMERA_PORT_DESC_T _port;

// funkcja do połączenia się z kamerą
int lepton_connect() {
          LEP_OpenPort(1, LEP_CCI_TWI, 400, &_port);
          _connected = true;
          return 0;
}

// funkcja uruchamiająca normalizację FFC w kamerze
void lepton_perform_ffc() {
          if(!_connected) {
                    lepton_connect();
          }
          LEP_RunSysFFCNormalization(&_port);
}

// tutaj dodać można więcej funkcji z SDK
```

nam skalę (*scale = 255/diff*) użytą do normalizacji (wartość tego piksela:
`value = (frameBuffer[i] - minValue) * scale`,
gdzie *frameBuffer* to bufor całej ramki obrazu). Gdy już cały obraz zostanie znormalizowany i załadowany do *myImage*, wyemitowany zostaje sygnał dla interfejsu Qt, informujący o tym, że obraz wyświetlany na ekranie może zostać zaktualizowany.

W *LeptonThread.cpp* znajdziemy także odwołanie do funkcji *lepton_perform_ffc()*. Służy ona do uruchomienia funkcji FFC – rekalibracji sensora, mającej na celu poprawienie jednorodności jego pracy. FFC uruchamiać należy w momencie, gdy obraz, jaki widzimy, przypomina środkowy lub prawy obraz, pokazany na **rysunku 6**. Implementacja funkcji *lepton_perform_ffc()* znajduje się w pliku *Lepton_I2C.cpp*, który pokazany jest na **listingu 3**. Funkcja *lepton_perform_ffc()* uruchamia jedynie połączenie z kamerą Lepton (poprzez interfejs I²C) i wykorzystuje funkcję z SDK, które dostarcza FLIR, wraz z samą kamerą. W pliku *Lepton_I2C.cpp* oraz *Lepton_I2C.h* zaimplementować można oczywiście i inne funkcje, jakie dostępne są w SDK.

Podsumowanie

W powyższym artykule przedstawiliśmy ciekawy projekt amatorskiej kamery termowizyjnej, opartej na sensorze FLIR Lepton. Sensor ten nie jest tanim elementem – w zależności od wersji i tego, czy jest zainstalowany na gotowej płytce PCB, kosztuje od kilkuset, do nawet 1,5 tysiąca złotych – to i tak jest o wiele tańsze rozwiązanie, niż dedykowana kamera termowizyjna. Dodatkowo, dzięki standardowym interfejsom – SPI oraz I²C – moduł ten z łatwością może zostać podłączony do dowolnego mikrokontrolera, płytki rozwojowej czy komputera jednopłytkowego. Ogromna elastyczność modułu i łatwość jego oprogramowania, dzięki dedykowanemu SDK, jest jego ogromną zaletą.

Nikodem Czechowski

Źródła:

1. ***http://bit.ly/2YIg7FN***
2. ***http://bit.ly/32gyrIg***

Rysunek 6a. Obraz o wysokiej jednorodności (normalny)

Rysunek 6b. Obraz ziarnisty (szum o dużej częstotliwości przestrzennej)

Rysunek 6c. Obraz plamisty (szum o małej częstotliwości przestrzennej)