ULB BRUFACE BRUSSELS FACULTY OF ENGINEERING VUB

ELEC-Y415

Heart rate detection by Photoplethysmography (PPG)

Software and Engineering for Embedded System

Authors:

Bao HAN
Anthony Salim

Professor:

Bruno Tiago da Silva Gomes

May 2024

1 Introduction

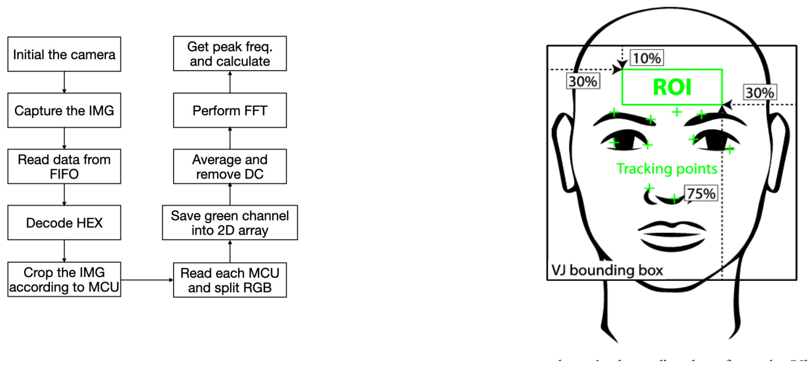


Figure 1: Plan of the project

This project utilizes Photoplethysmography (PPG), an optical method that measures changes in blood volume within peripheral circulation, to detect a person's heart rate. PPG is not only non-invasive and cost-effective but also provides critical cardiovascular data through the absorption of light by the skin.

Widely used in clinical settings, PPG devices monitor variables such as heart rate, blood oxygen saturation, blood pressure, and cardiac output.

The method involves illuminating the skin with an LED and measuring the light that is either reflected or transmitted by the skin using a photodiode, as changes in blood volume affect light absorption. This project specifically analyzes color variations in a targeted facial region, using the sensitivity of the green channel in RGB images to effectively capture the blood volume pulse.

Figure 1 illustrates the comprehensive plan for extracting and detecting heart rate, detailing the various steps involved. These steps are further elaborated upon in Sections 3, 4, and 5.

2 Hardware Components

SPI Connection	
Arducam OV5642	Arduino Nano 33 BLE
CS	D10
MOSI	D11
MISO	D12
SCK	D13
GND	GND
VCC	5V
SDA	A4
SCL	A5

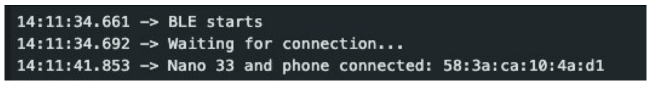
This project utilizes the Arduino Nano 33 BLE and the Arducam OV5642. The initial setup involves connecting the Arduino to the camera via an SPI connection. The table above delineates the pin connections between the Arduino and the camera, each serving specific functions:

1. **CS (Chip Select)**: Links to D10 on Arduino to select the camera among multiple SPI devices.
2. **MOSI (Master Out Slave In)**: Connected to D11, sends data from Arduino to the camera.
3. **MISO (Master In Slave Out)**: Attached to D12, sends data from the camera to Arduino.
4. **SCK (Serial Clock)**: Connected to D13, provides the clock signal for SPI communication.
5. **GND (Ground)**: Completes the circuit by connecting to Arduino's ground.
6. **VCC (Power Supply)**: Powers the camera via Arduino's 5V output.
7. **SDA (Serial Data)**: Linked to A4 for I2C communication to configure the camera.
8. **SCL (Serial Clock)**: Attached to A5, provides the I2C clock signal.

Control over the Arduino and Arducam is facilitated through a Bluetooth connection, with a mobile phone application sending commands to capture images.

3 Capturing the Image

3.1 Bluetooth

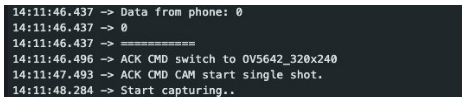


```
14:11:34.661 -> BLE starts
14:11:34.692 -> Waiting for connection...
14:11:41.853 -> Nano 33 and phone connected: 58:3a:ca:10:4a:d1
```

Figure 2: Bluetooth connection succeed

The Arduino is controlled via Bluetooth using the **ArduinoBLE** library. By sending `0x00` from the phone to the Arduino, the image setting is configured to JPEG with a resolution of 320x240. The UUID of the BLE is set to *2A37* for the heart rate characteristic.

3.2 Taking the Image



```
14:11:46.437 -> Data from phone: 0
14:11:46.437 -> 0
14:11:46.437 -> ===========
14:11:46.496 -> ACK CMD switch to OV5642_320x240
14:11:47.493 -> ACK CMD CAM start single shot.
14:11:48.284 -> Start capturing..
```

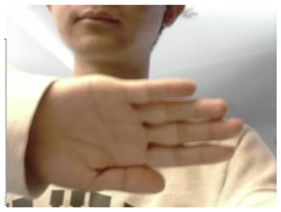


Figure 3: Arducam OV5642 starts to capture image

Upon receiving `0x00`, the Arducam start capturing an image (as shown in figure 3) and compresses it to JPEG format, storing the data in the FIFO buffer. The compressed image data is read one byte at a time using **myCAM.read_fifo()** until the end marker `0xFFD9` is detected. The image data starts with `0xFFD8` and ends with `0xFFD9`, marking the boundaries of the JPEG file.

There is another way to read the data, which is the burst modes. In this mode multiple bytes can be transmitted during one SPI transmission, which is way faster than the normal read FIFO method. However, one also needs to pay attention that this method may consume more power during the transmission. In our case, the size of captured image is not that big (only 320×240 and JPEG format), therefore it is not necessary to use burst mode (burst mode is more favorable in large-size BMP image transmission).

4 Image Processing

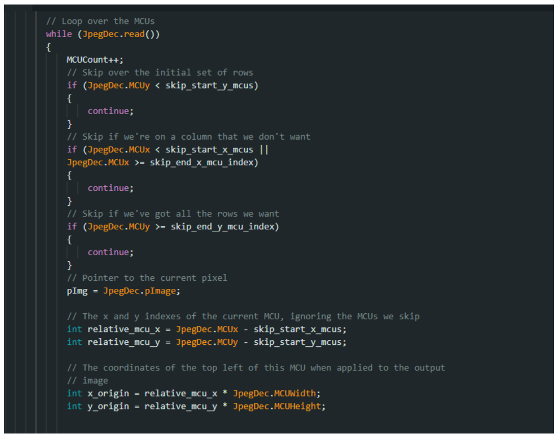
This section outlines the core steps involved in processing images from capture to the extraction of useful data for heart rate detection using Photoplethysmography (PPG). These steps include decoding JPEG images, cropping to focus on specific regions, and extracting the green channel which is crucial for detecting blood volume changes.

4.1 From JPEG to MCU

The **JPEGDecoder** library decodes the JPEG image into Minimum Coded Units (MCUs). Each MCU, typically 16x8 pixels for a 320x240 image, represents the smallest unit processed together. The function **JpegDec.decodeArray()** is crucial for handling JPEG compression and enables efficient image data manipulation.

4.2 Cropping the Image

To minimize computational load and enhance measurement accuracy, the image is cropped to focus on areas with prominent blood flow signals, such as the cheeks. This selective cropping to a 16x8 pixel Region of Interest (ROI), representing one MCU, ensures precision in subsequent analyses.



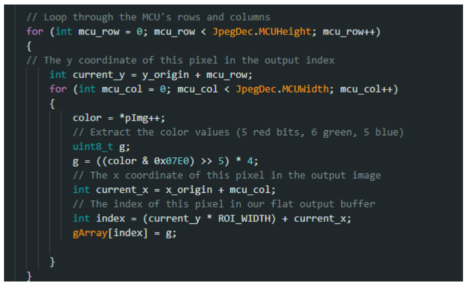
```
// Loop over the MCUs
while (JpegDec.read())
{
    MCUCount++;
    // Skip over the initial set of rows
    if (JpegDec.MCUy < skip_start_y_mcus)
    {
        continue;
    }
    // Skip if we're on a column that we don't want
    if (JpegDec.MCUx < skip_start_x_mcus ||
    JpegDec.MCUx >= skip_end_x_mcu_index)
    {
        continue;
    }
    // Skip if we've got all the rows we want
    if (JpegDec.MCUy >= skip_end_y_mcu_index)
    {
        continue;
    }
    // Pointer to the current pixel
    pImg = JpegDec.pImage;

    // The x and y indexes of the current MCU, ignoring the MCUs we skip
    int relative_mcu_x = JpegDec.MCUx - skip_start_x_mcus;
    int relative_mcu_y = JpegDec.MCUy - skip_start_y_mcus;

    // The coordinates of the top left of this MCU when applied to the output
    // image
    int x_origin = relative_mcu_x * JpegDec.MCUWidth;
    int y_origin = relative_mcu_y * JpegDec.MCUHeight;
```

Figure 4: Illustration of MCUs being skipped outside the ROI

MCUs outside the designated ROI are skipped, and the green channel is extracted from those within for further processing, as detailed in Section 4.3.

4.3 Extracting the Green Channel

The green channel is crucial for PPG as it best captures blood volume changes due to its sensitivity to light absorption differences caused by blood flow.



```
// Loop through the MCU's rows and columns
for (int mcu_row = 0; mcu_row < JpegDec.MCUHeight; mcu_row++)
{
// The y coordinate of this pixel in the output index
    int current_y = y_origin + mcu_row;
    for (int mcu_col = 0; mcu_col < JpegDec.MCUWidth; mcu_col++)
    {
        color = *pImg++;
        // Extract the color values (5 red bits, 6 green, 5 blue)
        uint8_t g;
        g = ((color & 0x07E0) >> 5) * 4;
        // The x coordinate of this pixel in the output image
        int current_x = x_origin + mcu_col;
        // The index of this pixel in our flat output buffer
        int index = (current_y * ROI_WIDTH) + current_x;
        gArray[index] = g;

    }
}
```

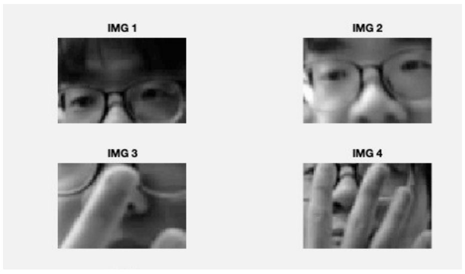


Figure 5: Image Green Channel

In RGB565 format, the green channel is extracted using a mask (0x07E0), shifted right by 5 bits and scaled up by a factor of 4 to enhance the signal quality, covering a full 8-bit range (0-255).

- **Masking**: Applying `0x07E0` retains only the green bits.
- **Shifting**: Shifting the masked value right by 5 bits aligns it to the base.
- **Scaling**: Multiplying by 4 adjusts the range from 0-63 to 0-255.

5 Detecting the Heart Rate

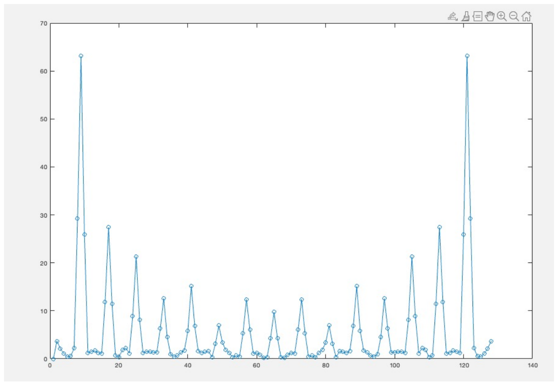


Figure 6: FFT of a signal

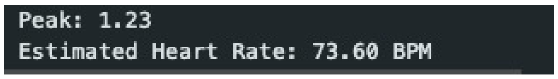
This project leverages Photoplethysmography (PPG) to measure heart rate by detecting changes in blood volume within peripheral circulation.
The heart rate detection process begins by transforming the green channel data of multiple images captured by the camera (7 in the project) to the frequency domain using Fast Fourier Transform (FFT). Prior to the FFT, the signal undergoes several preprocessing steps to enhance clarity and reduce noise:

1. **Removing the DC Component:** This step involves averaging the green channel values per frame to isolate changes due to heartbeats. Subtracting the mean removes biases from static conditions like ambient lighting, enhancing the quality of the signal for FFT analysis.

2. **High Pass Filter:** Applied to eliminate low-frequency noise and environmental artifacts that do not correlate with heart rate, this filter ensures that only frequencies above a specific cutoff (1 Hz) are analyzed. This could further improve the accuracy of the measurement.

3. **Windowing:** Prior to FFT, windowing reduces spectral leakage by smoothing the signal edges. This technique helps to accurately delineate the frequency spectrum, essential for precise heart rate detection.

By performing these step, you're preparing the data for more accurate frequency analysis, focusing on the changes due to the heart's pulsation rather than gradual shifts in baseline or other slow changes not related to the heart rate.



```
Peak: 1.23
Estimated Heart Rate: 73.60 BPM
```

Figure 7: Heart Rate per Minute

After processing, the peak frequency (in Hz) identified from the FFT corresponds to the heart rate, which is converted to beats per minute (bpm) by multiplying by 60. The results, as shown in Figure 7, demonstrate the PPG method's effectiveness in heart rate monitoring.

Due to the resolution of the camera, variable room lighting conditions and the inaccurate focus, the results may not always be 100% accurate.

6 Procedure

Step1: Searching for the Arduino nano 33 BLE:

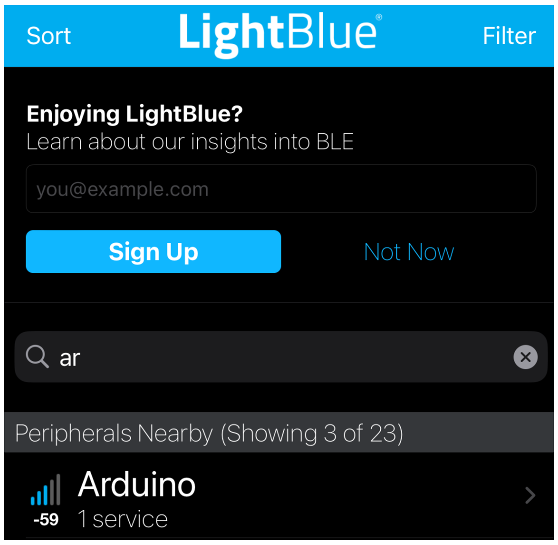
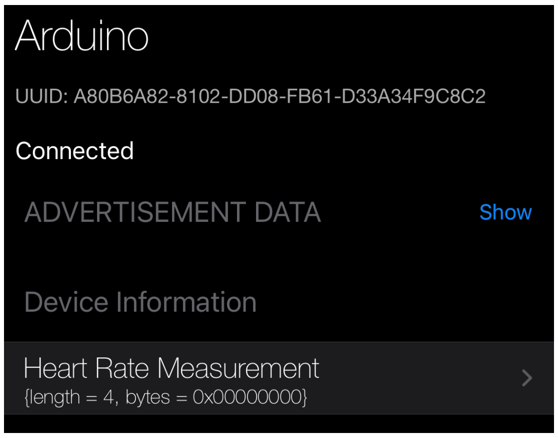


Figure 8: Searching for Arduino

Step2: Connect with the Service:


Arduino

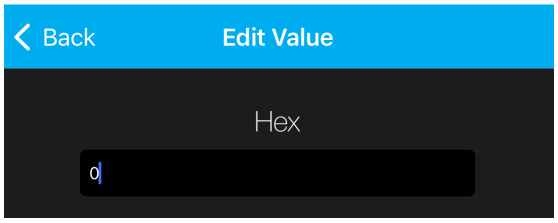
UUID: A80B6A82-8102-DD08-FB61-D33A34F9C8C2

Connected

Heart Rate Measurement
{length = 4, bytes = 0x00000000}

Figure 9: Choosing the service

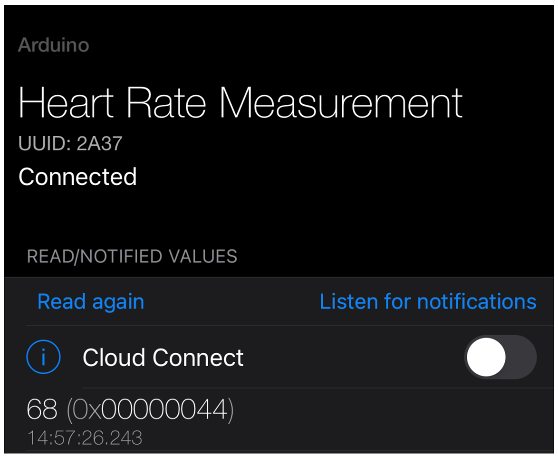
Step3: Input 0 to begin the capture:



0

Figure 10: Starting capturing

Step4: Read the output:



Arduino

Heart Rate Measurement

UUID: 2A37

Connected

READ/NOTIFIED VALUES

68 (0x00000044)

14:57:26.243

Figure 11: Getting result, which is 68 BPM

Finally, to see the accuracy of the measurement, the heart rate detector on the iWatch is used, the result is shown as below:

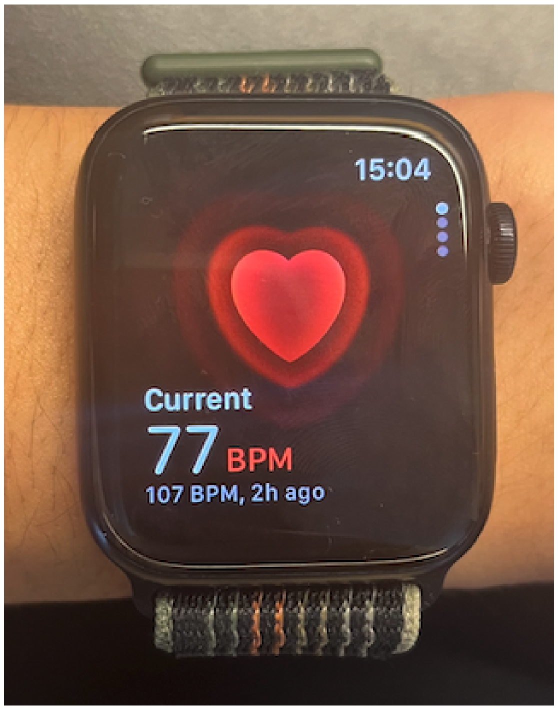


Figure 12: Result from iWatch