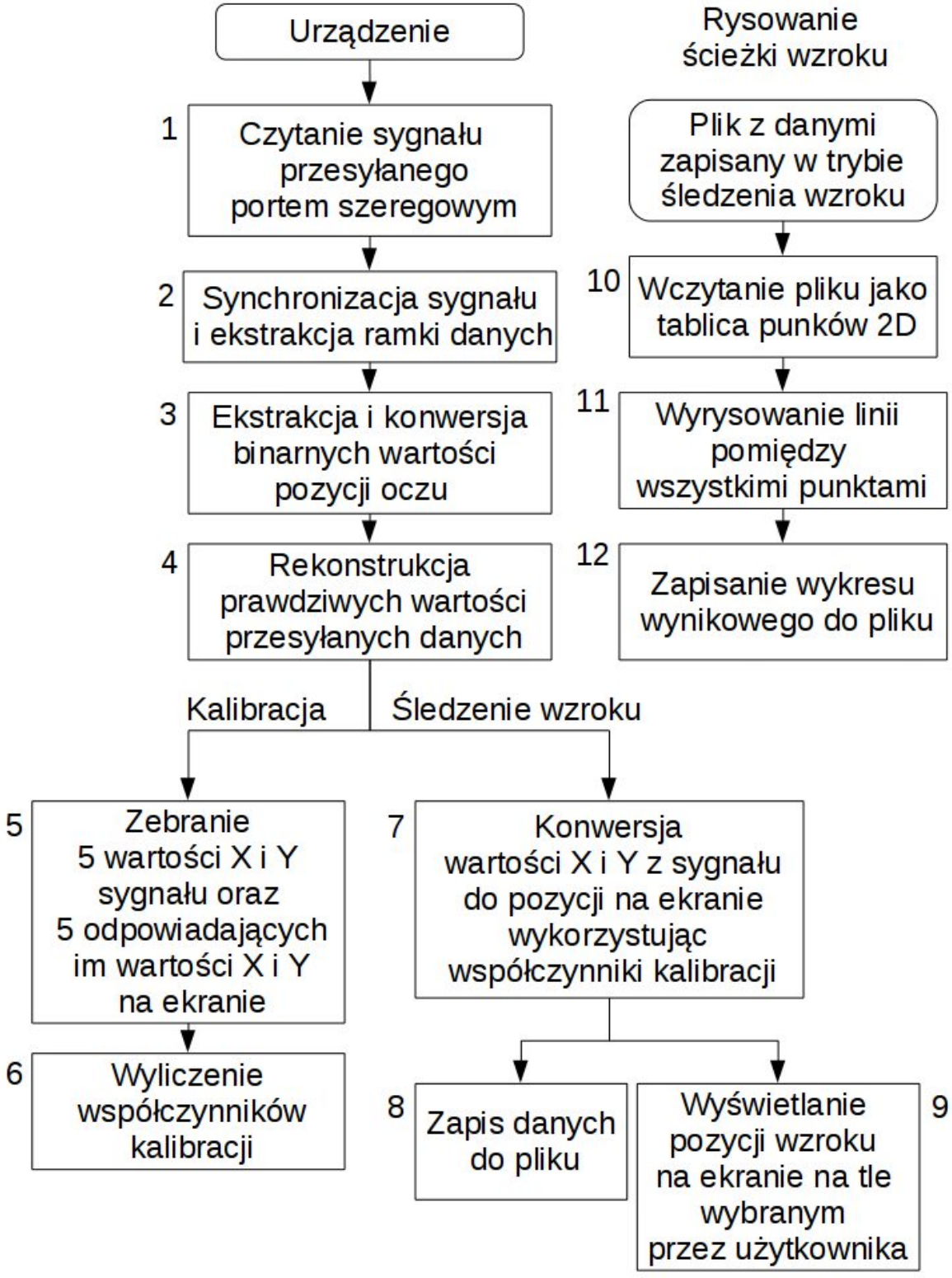
Zasada działania programu EyeTracker



Rys. 1 Schemat trzech trybów działania programu: kalibracja, śledzenie wzroku, rysowanie ścieżki wzroku.

Blok 1. Czytanie sygnału przesyłanego portem szeregowym
Klasy: *DataHandler*, *SerialPortReader*

Połączenie z urządzeniem jest możliwe dzięki sterownikowi do modułu FTDI urządzenia tworzącego wirtualny port szeregowy z danymi wysyłanymi przez urządzenie za pomocą portu USB. Dane z portu są sczytywane z wykorzystaniem klasy *QSerialPort*, metodą asynchroniczną (przykład implementacji: http://doc.qt.io/qt-5/qtserialport-creaderasync-example.html).

Dane potrzebne do nawiązania połaczenia zaczerpnięto z dokumentacji urządzenia (załącznik 1).

Blok 2. Synchronizacja sygnału i ekstrakcja ramki danych
Klasy: *SerialPortReader*

Na podstawie danych o przesyłanym sygnale (załącznik 1) zaimplementowano algorytm synchronizacji i ekstrakcja ramki, który polega na buforowaniu sygnału, szukanie w nim indeksu bajtów markera ramki (3 puste bajty na początku ramki i jeden o wartości FF na końcu), a następnie wycięcie z bufora danych o długości ramki (56 bajtów) z początkiem zadanym przez znaleziony indeks i przekazanie jej do dalszego przetwarzania. Metoda implemetująca algorytm to *handleReadyRead* w klasie *SerialPortReader*.

Blok 3. Ekstrakcja i konwersja binarnych wartości pozycji oczu
Klasa: *DataHandler*

Ekstrakcja danych z ramki odbywa się przez czytanie ramki bajt bo bajcie, sprawdzaniu poszczególnych bitów bajtu z użyciem maski bitowej i wpisywanie danego bajtu do wcześniej stworzonej struktury zawierającej pola odpowiadające danym w ramce na podstawie ich numeru kolejności w ramce (załącznik 1).

Następnie zapisane dane binarne w polach typu QString strukury ramki są konwertowane do typu int z użyciem metody *toInt*.

Blok 4. Rekonstrukcja prawdziwych wartości przesyłanych danych
Klasa: *DataHandler*

Wartości są przesyłane przez urządzenie jako przesuwający się przedział o wartości początka i końca ograniczonym przez rozmiar bitowy liczby (12 bitów), co daje nam zakres od 0 do 4096. Z tego powodu wymagany jest algorytm rekonstruujący wykrywający przeskoki tego przedziału, ich kierunek i wartość przesunięcia, a następnie dodający globalnie wyliczone przesunięcie do przesyłanych danych, tak aby odtworzyć ich pierwotną wartość. Wiecej na temat problemu i implemetacji algorytmu w dokumentacji urządzenia (załącznik 2).

Blok 5. Zebranie 5 wartości XY sygnału oraz 5 odpowiadających im wartości XY na ekranie
Klasy: *CalibrationHandler*, *Calibration*

Zebranie 5 wartości X i Y sygnału polega na zapisaniu aktualnie przesyłanych danych do tablicy w momencie wciskania przez użytkownika klawisza spacji. Pięć korespondujących punktów na ekranie jest wyliczanych na podstawie danych o rozdzielczości ekranu i ustalonej pozycji punktów kalibracyjnych na planszy kalibracyjnej, tj. 4 punkty w rogach ekranu oddalone o 100px od najbliższych krawędzi oraz jeden punkt w centrum.

Blok 6. Wylicanie współczynników kalibracji
Klasa: *Calibration*

Algorytm wyliczający współczynniki kalibracji oparto na algorytmie używanym do kalibracji ekranów dotykowych. Opis algorytmu w artykule „Calibration in touch-screen systems" autorstwa Wendy'ego Fanga i Tony'ego Changa.

Blok 7. Konwersja wartości XY z sygnału do pozycji na ekranie wykorzystując współczynniki kalibracji
Klasa: *Calibration*

Dzięki wyliczonym podczas kalibracji współczynnikom, klasy *SessionHandler i EyePointerWidget* mogą konwertować dane z urządzenia na pozycję na ekranie, wykorzystując wskaźnik do klasy *Calibration* i jej metodę *getPointOnScreen*.

Blok 8. Zapis danych do pliku
Klasa: *SessionHandler*

Dane zapisywane są do formatu CSV z 6 liniowym nagłówkiem o postaci:

```
sep=,
Session:, %1
Date:, %1
Screen width [px]:, %1
Screen height [px]:, %1
Time [ms], Eye position X [px], Eye position Y [px]
```

Pierwsza linia wymusza w programie Excel wykorzystanie przecinka jako separatora kolumn, co zapewnia jego prawidłowe otwarcie w polskiej wersji tego programu. Kolejne linie opisują dane i warunki badania (rozdzielczość ekranu). Pod nagłówkiem wpisywane są 3 kolumny danych zawierające czas i pozycję XY na ekranie.

Blok 9. Wyświetlanie pozycji wzroku na ekranie, na tle wybranym przez użytkownika
Klasy: *SessionHandler*, *EyePointerWidget*

Pozycja rysowana jest w postaci czerwonej kropki dzięki wykorzystaniu klasy QPainter. Tło wyświetlane jest z użyciem klasy QWebEngineView umozliwiającej ładowanie jako tło strony internetowej bądź pliku html offline.

Blok 10. Wczytanie pliku jako tablica punktów 2D
Klasa: *DataPlotter*

Dane sczytywane są z pliku linia po linii z omnięciem 6 pierwszych linii nagłówka, a następnie rozdzielane na poszczególne kolumny używając przecinka jako separatora kolumn. Istotne dane do rysowania (pozycja XY na ekranie) są wysyłane i zapisywane w buforze klasy rysującej *PlotWidget*.

Blok 11. Wyrysowanie linii pomiędzy wszystkimi punktami
Klasa: *PlotWidget*

Wykorzystując klasę QPainter rysowane są linie łączące punkty sczytane z pliku, tworzące ścieżkę wzroku na ekranie. W tle wyświetlane jest z użyciem klasy QwebEngineView te same tło które ustawione jest w przypadku trybu śledzenia wzroku.

Blok 12. Zapisanie wykresu wynikowego do pliku
Klasa: *DataPlotter*

W momencie wyrysowania wykresu cały obraz wraz z tłem jest renderowany i zapisywany jako plik o formacie PNG z uzyciem klasy QPixmap.

Załącznik 1 – Dokumentacja sposobu przesyłania danych przez urządzenie

5. Serial communication details

The JAZZ-novo system provides data to the host PC using the USB serial communication. The USB interfaces of the JAZZ-novo components utilizes the FTDI® chips. The drivers for these interfaces can be found on CD disc provided with the system.

The parameters for establishing the communication for serial transmission are following

Baud rate: 300000 bits/s

Parity: None

Byte size: 8

Stop bits: 1

The JAZZ-novo system transmit data in packets using the following format.

1 packet (56 bytes) = 2 frames 28 bytes each

	0	1	2	3	4	5	6	7	8	9	10	11
frame 1	0	0	0	eye_y_0	eye_x_0	acc_x_0	acc_y_0	gyro_x_0	gyro_y_0			

12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
mic_0		mic_1		mic_2		mic_3		mic_4		mic_5		mic_6		mic_7	C1	eye_b	COUNTER

	28	29	30	31	32	33	34	35	36	37	38	39
frame 2	pul_l	pul_R	eye_y_1	eye_x_1	acc_x_1	acc_y_1	gyro_x_1	gyro_y_1				

40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
mic_8		mic_9		mic_10		mic_11		mic_12		mic_13		mic_14		mic_15	C2	CRC	C3

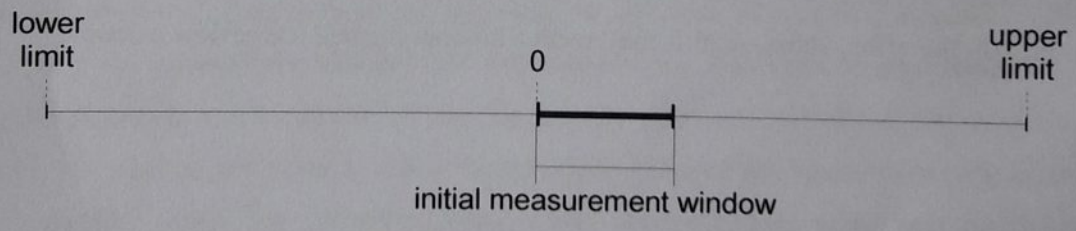
Each packet contains two frames of 28 bytes each. The packets are generated with the 500 Hz frequency (2 ms per packet).

Załącznik 2 – Dokumentacja implemetacji algorytmu rekostrukcji sygnału

7. Measurement window – reconstruction algorithm

Both eye position signals and pulsoxymetry signals are being measured in zoomed, movable measurement windows. Data signal samples are sent to the host PC using the 12 bits lengths format. Each sample has the offset which has to be added to the sample to obtain the sample value in the whole possible measurement range. Normally this is done automatically by the JazzRecorder software so the user does not have to take care about it. This chapter explain the reconstruction algorithm used on the PC side in case the raw data samples has to be decoded in the user's custom application.

At the start-up Jazz attempts to set the window to best-fit signal in it. After balancing, each eye position signal and pulsoxymetry signal will start in the middle of the measurement window. Movable measurement window allows tracing *Eye position* or *Pulsoxymetry* signals when they leaves measurement range. We may imagine the whole possible range of the measured singal with the sensor window moving along this range allowing to see the part of the range.

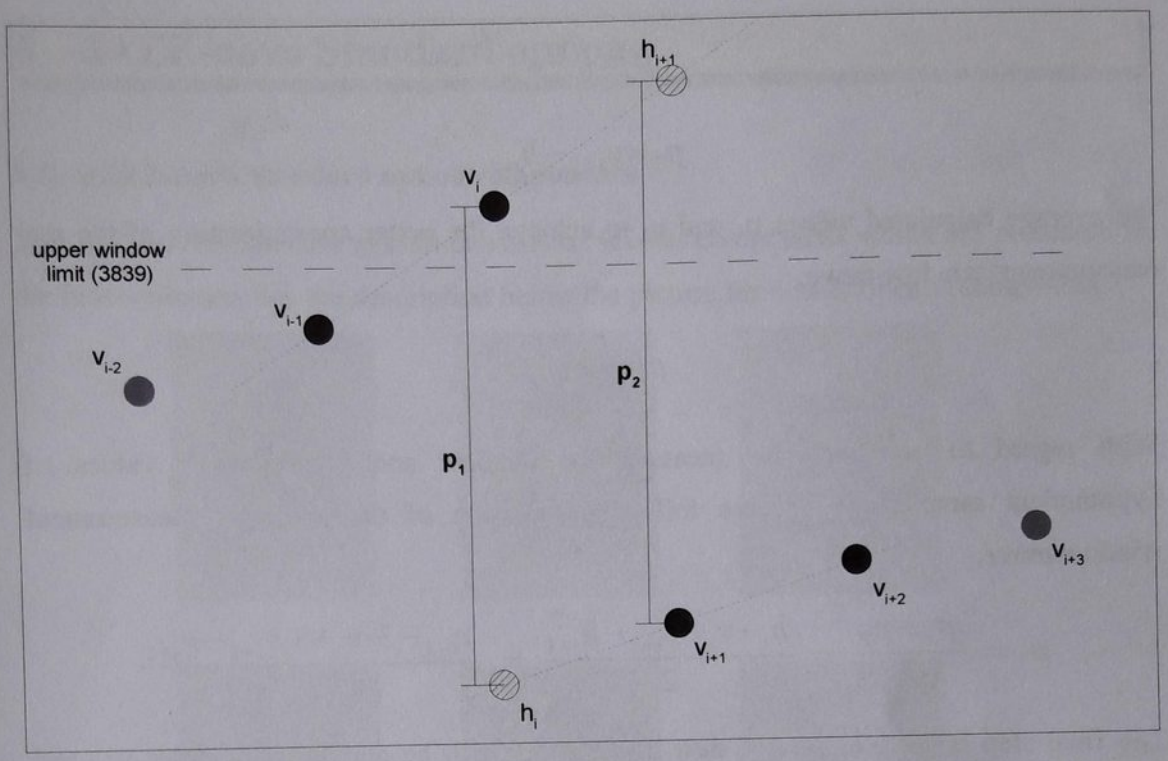


The sample values coming from the sensor to PC are fitted into the 12 bit (4096) window. We may imagine that the sensor has deducted from each sample value the offset which is actually the left window border value. Thus to obtain real sample value we have to add to it the actual value of the window left border. At the very beginning after the device is switched on we may assume that the offset is 0. Thus every sample value after adding the

offset will have the same value as before.

Let imagine that the signal value is coming close to the right border of the actual measurement window (i.e. 4095 value). When the value of the sample exceed EFF value in hexadecimal code (what is 3839 in decimal code) the next sample offset is moved to fit this sample somewhere in the measurement window. However due to the hardware implementation of the jump it is impossible to get to know the exact value of the jump in the PC software. This value has to be deducted from the following samples value. This is the moment where the reconstruction algorithm comes to play.

Basically the approach is following. Calculate hypothetical next sample value after the sample which has crossed the upper jump border EFF using the linear approximation from previous two samples (as if there was no moving window). Calculate hypothetical previous sample value to the first sample in new window using also linear approximation. Calculate first size of jump using difference between sample value from first step and sample value of the first sample in the new window. Calculate the second size of the jump using difference between last sample value in the old window and sample value from the second step. Average both jumps sizes to get the jump size closest to the real jump size and modify the offset value. As this may seem a bit complicated lets review example.



As you can see on the above figure the v_i if the first value above the upper jump border. The next samples, v_{i+1} and v_{i+2}, are measured in the new, moved, measurement window (have new offset). According to our algorithm we calculate hypothetical values of sample v_{i+1} as if the measurement window had not been moved. This value is represented on the figure as sample h_{i+1}. In the similar way we calculate hypothetical value of the sample v_i in the new measurement window. This value is represented on the figure as sample h_i. Both hypothetical values are calculated using linear approximation using two consecutive samples.

$$h_{i+1} = v_i + (v_i - v_{i-1})$$

$$h_i = v_{i+1} - (v_{i+2} - v_{i+1})$$

Now we can calculate the approximation of the measurement window move based on the two pairs of samples (h_i , v_i) and (h_{i+1} , v_{i+1}).

$$p_1 = h_i - v_i$$

$$p_2 = v_{i+1} - h_{i+1}$$

We average calculated values p_1 and p_2 to achieve the better approximation of the real measurement window move.

$$p = \frac{p_1 + p_2}{2}$$

With regard to equations for measurement window and equations of values of hypothetical samples we receive following equation of the averaged measurement window move.

$$p = \frac{p_1 + p_2}{2} = \frac{h_i - v_i + v_{i+1} - h_{i+1}}{2} = \frac{v_{i-1} - 3 \cdot v_i + 3 \cdot v_{i+1} - v_{i+2}}{2}$$

The final step is to calculate the new offset which will be applied to samples v_{i+1} and following. To get new offset value we have to **deduct** calculated value p from the previous offset value (let's remind here that the initial offset is 0).

$$NewOffset = OldOffset - p$$

As the move value is negative in the presented example the new offset will be greater from the old one what fits our expectation. After adding the new offset to the v_{i+1} value it will land somewhere near the h_i value. Exactly the same procedure can be used for the situation when sample value goes below the lower jump border (0FF hexadecimal or 256 decimal).

Notice! All calculations must be made using values without offset applied - just as they come from the sensor in the 12 bit (4096) range. The old offset is valid for v_i sample and all preceding, and the new offset if valid for v_{i+1} sample and all following.