Laboratorium z przedmiotu Systemy wbudowane (SW)

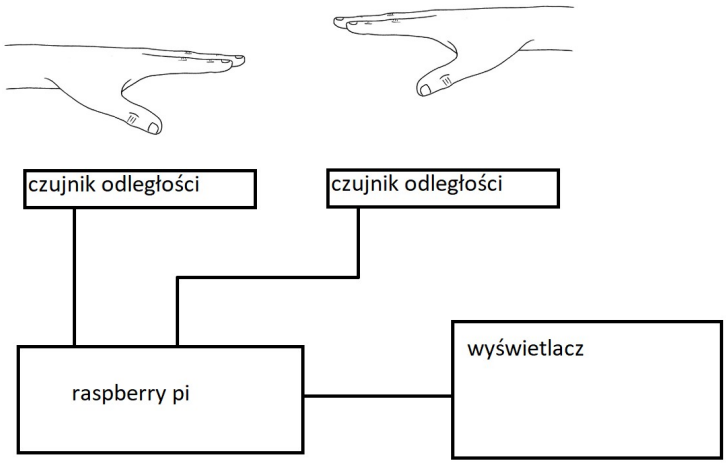
Karta projektu – zadanie 7

Nazwa projektu:
PONG using distance meters and analog display

Prowadzący:	*Autorzy:*	*Grupa dziekańska:*	*I1.2*
Ariel Antonowicz	148071 148099	***Ocena:***	

Cel projektu:
Utworzenie systemu wbudowanego służącego do grania w PONG.
Gracze będą sterować pozycjami paletek przez ruszanie dłońmi nad czujnikami odległości.
Piłka oraz paletki będą wyświetlane na wyświetlaczu przy użyciu sygnału analogowego.

Schemat:



Wykorzystana platforma sprzętowa, czujniki pomiarowe, elementy wykonawcze:

2 Czujniki odległości HC-SR04
Raspberry Pi 3B/4
Monitor ze złączem analogowym

Cel i zakres projektu

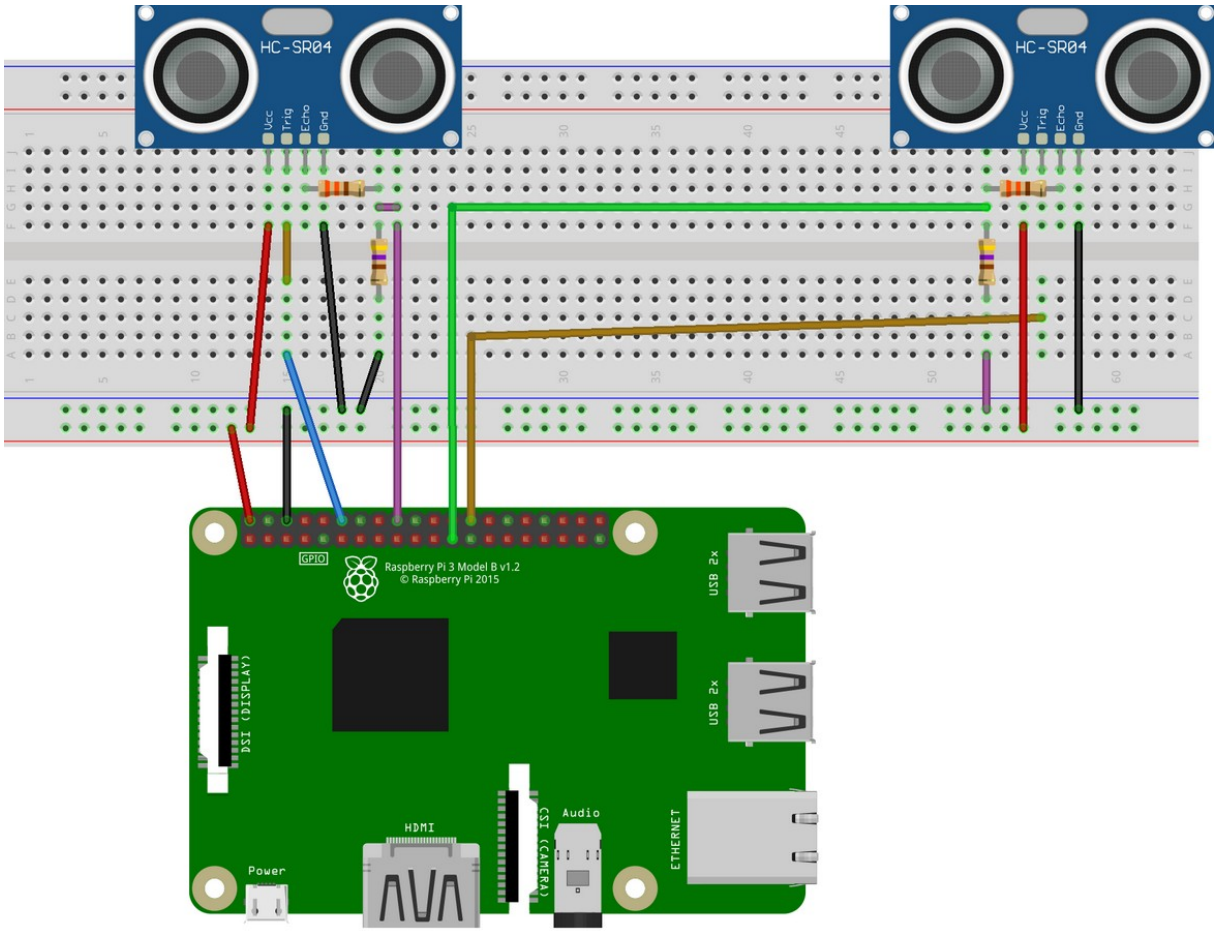
Celem projektu było przygotowanie systemu wbudowanego służącego do gry PONG sterowanej przy użyciu ultradźwiękowych czujników odległości. Dodatkowo interfejs gry wyświetlany miał być na wyświetlaczu sterowanym sygnałem analogowym generowanym programowo.

Projekt objął przygotowanie programu dla Raspberry pico, który na port szeregowy przyjmuje pozycje paletek oraz piłki w sposób asynchroniczny a następnie generuje sygnał analogowy dla złącza VGA monitora. Dane dla modułu raspberry pico są generowane przez skrypt uruchomiony na raspberry pi 4b. Zadaniem tego modułu jest sczytywanie odległości z dwóch czujników, obliczanie pozycji piłki na planszy (i całej logiki gry), serializacja tych danych i wysyłanie ich w pętli na łącze szeregowe do raspberry pico.

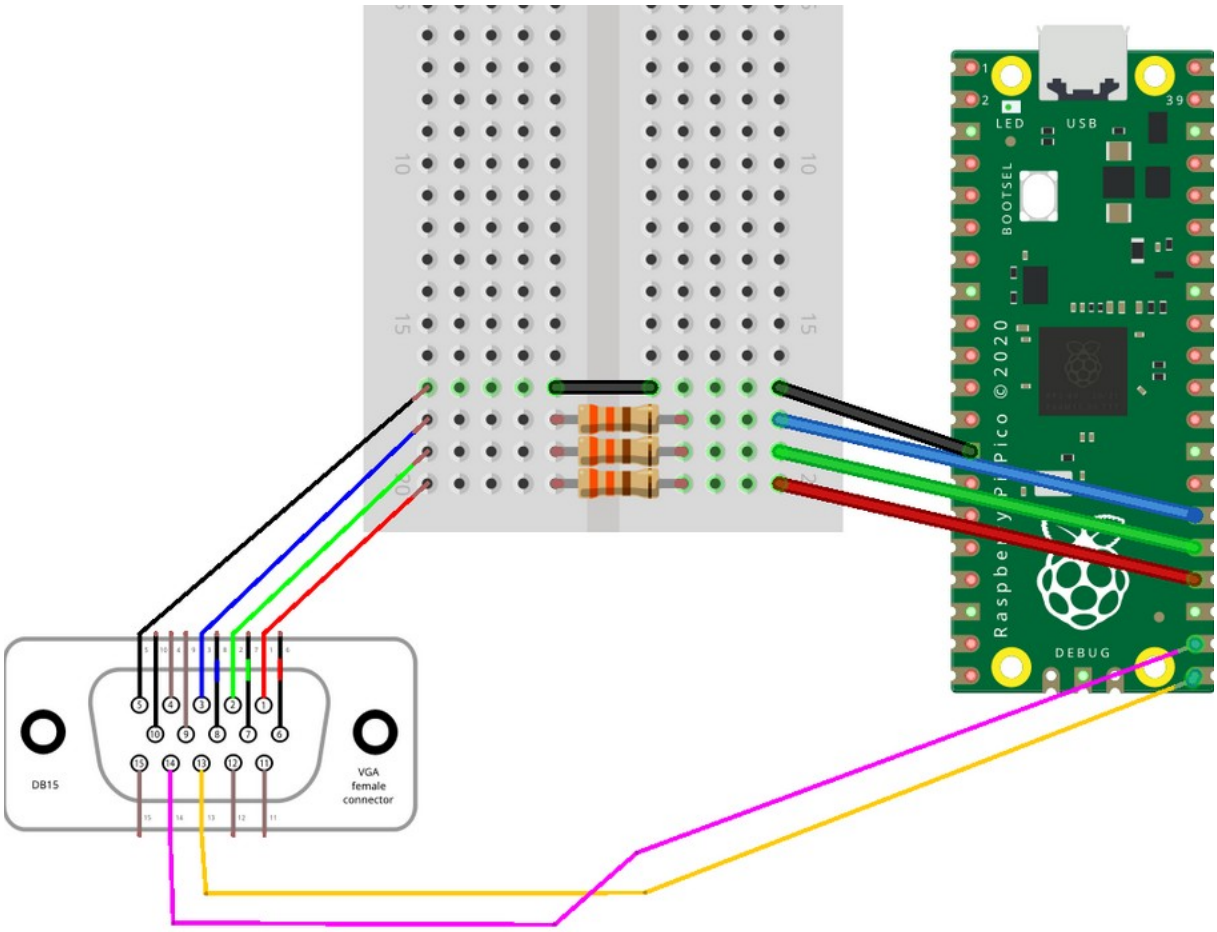
Schemat ideowy

Na stronie tytułowej

Schemat połączeniowy



Rys 1. Schemat połączenia czujników do raspberry pi 4b



Rys 2. Schemat połączenia raspberry pico do złącza vga

Powyższe dwa układy są połączone ze sobą złączem microUSB.

Projekt a realizacja

Wynik końcowy naszej pracy nie jest całkowicie zgodny z naszymi początkowymi założeniami. Po pierwsze, zakładaliśmy użycie jedynie modułu raspberry pi 4b. Ze względu na małą ilość materiałów dotyczących „ręcznego” sterowania sygnałem analogowym VGA przy użyciu tego modułu, zdecydowaliśmy się użyć raspberry pico jako „pośrednika” między raspberry pi 4 a wyświetlaczem. Kolejną rzeczą która nie zgadza się z naszymi oczekiwaniami to płynność wyświetlanego obrazu. Użycie ultradźwiękowych czujników odległości, komunikacji szeregowej oraz asynchroniczny charakter komunikacji między modułami raspberry sprawił, że wyświetlany obraz jest powolny, a gra toczy się stosunkowo wolno. Mimo wszystko jesteśmy zadowoleni z rezultatu naszej pracy.

Projekt można by usprawnić przez zsynchronizowanie wysyłania danych z raspberry pi 4 i ich odczyt na raspberry pico, pomogłoby to uzyskać stałą liczbę klatek na sekundę, co mogłoby wpłynąć na poprawienie płynności obrazu.

Fragmenty kodu źródłowego raspberry pico

```
int a=0, b=0, c=0,d=0;
int oa=0, ob=0, oc=0, od=0;
while (true) {
    scanf("%d %d %d %d\n", &a, &b, &c, &d);
    printf("\n");

    for (i=0; i<640; i++) {
        for (j=0; j<480; j++) {
            if ((i >= 20 && i <= 60) && (j > a && j < a + 120)){
                drawPixel(i, j, BLUE); // rysowanie lewej paletki
            }
            else if((i >= 580 && i <= 620) && (j > b && j < b + 120)) {
                drawPixel(i, j, RED); // rysowanie prawej paletki
            }
            else if ( (i>=c && i<=c+40) && (j>=d && j<d+40) ) {
                drawPixel(i, j, WHITE); // rysowanie piłki
            }
            else {
                drawPixel(i, j, BLACK); // tło planszy
            }
        }
    }
}
```

Rys 3. Pętla główna modułu, jej zadaniem jest sczytanie danych z portu szeregowego i wygenerowanie sygnału dla VGA

```
void drawPixel(int x, int y, char color) {
    // Range checks
    if (x > 639) x = 639 ;
    if (x < 0) x = 0 ;
    if (y < 0) y = 0 ;
    if (y > 479) y = 479 ;

    // Oblicz numer piksela na podstawie dwuwymiarowych współrzędnych?
    int pixel = ((640 * y) + x) ;

    // Jeden bajt koduje dwa piksele
    // Sprawdzamy czy piksel, który chcemy narysować
    // jest reprezentowany przez pierwszą czy drugą połowę bajtu
    if (pixel & 1) {
        char mask = 0x0f;
        char other = vga_data_array[pixel>>1] & mask;
        vga_data_array[pixel>>1] = other | (color << 3);
    } else {
        char mask = 0xf0;
        char other = vga_data_array[pixel>>1] & mask;
        vga_data_array[pixel>>1] = color | other;
    }
}
```

Rys 4. Funkcja drawPixel – vga_data_array jest buforem bezpośredniego dostępu do pamięci do PIO

```
int rgb_chan_0 = 0;
int rgb_chan_1 = 1;

dma_channel_config c0 = dma_channel_get_default_config(rgb_chan_0);
channel_config_set_transfer_data_size(&c0, DMA_SIZE_8);
channel_config_set_read_increment(&c0, true);
channel_config_set_write_increment(&c0, false);
channel_config_set_dreq(&c0, DREQ_PIO0_TX2) ;
channel_config_set_chain_to(&c0, rgb_chan_1);

dma_channel_configure(
    rgb_chan_0,
    &c0,
    &pio->txf[rgb_sm],
    &vga_data_array,
    TXCOUNT,
    false
);

dma_channel_config c1 = dma_channel_get_default_config(rgb_chan_1);
channel_config_set_transfer_data_size(&c1, DMA_SIZE_32);
channel_config_set_read_increment(&c1, false);
channel_config_set_write_increment(&c1, false);
channel_config_set_chain_to(&c1, rgb_chan_0);

dma_channel_configure(
    rgb_chan_1,
    &c1,
    &dma_hw->ch[rgb_chan_0].read_addr,
    &address_pointer,
    1,
    false
);

pio_sm_put_blocking(pio, hsync_sm, H_ACTIVE);
pio_sm_put_blocking(pio, vsync_sm, V_ACTIVE);
pio_sm_put_blocking(pio, rgb_sm, RGB_ACTIVE);


pio_enable_sm_mask_in_sync(pio, ((1u << hsync_sm) | (1u << vsync_sm) | (1u << rgb_sm)));

dma_start_channel_mask((1u << rgb_chan_0)) ;
```

Rys 5. Inicjalizacja kanałów bezpośredniego dostępu do pamięci

Fragmenty kodu raspberry pi 4

```
try:
    while True:
        dist1 = distance(ECHO_1, TRIGGER_1)
        while dist1 == -1:
            # Pobieraj odczyt z pierwszego czujnika dopóki nie będzie on poprawny
            dist1 = distance(ECHO_1, TRIGGER_1)

        dist2 = distance(ECHO_2, TRIGGER_2)
        while dist2 == -1:
            # Pobieraj odczyt z drugiego czujnika dopóki nie będzie on poprawny
            dist2 = distance(ECHO_2, TRIGGER_2)
        # aktualizuj pozycję paletek i piłki
        update_ball()
        move_paddle1(dist1)
        move_paddle2(dist2)
        print(score)
        # wypisz dane na port szeregowy
        ser.write(bytearray("{} {} {} {}\n".format(
            int(paddle2pos), int(paddle1pos),
            int(ballpos[0]),int(ballpos[1])),
            "utf-8"))
        # czekaj aż moduł pico potwierdzi odebranie danych
        _ = ser.readline()

except KeyboardInterrupt:
    GPIO.cleanup()
```

Rys 6. Pętla główna modułu, w każdej iteracji, pobiera ona odległość z obu czujników, oblicza pozycję piłki i paletek oraz wysyła dane na łącze szeregowe

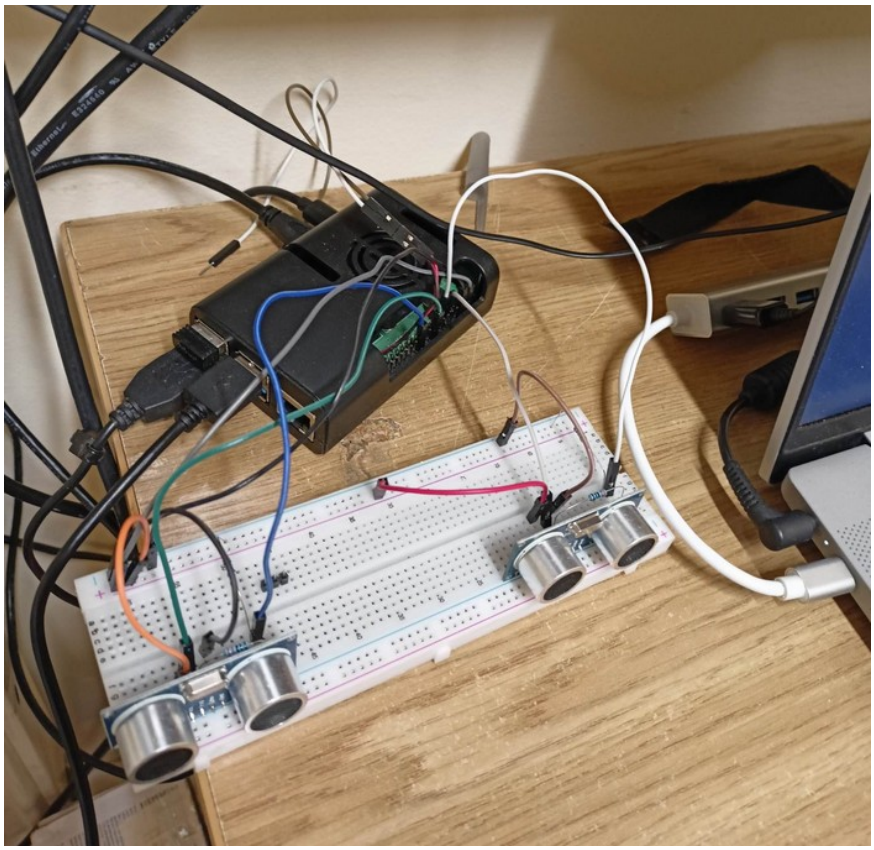
```python
def distance(echo, trigger):
    # wyślij sygnał dźwiękowy
    GPIO.output(trigger, GPIO.HIGH)
    time.sleep(0.00001)
    GPIO.output(trigger, GPIO.LOW)

    StartTime = time.time()
    StopTime = time.time()

    i = 0
    while GPIO.input(echo) == 0:
        StartTime = time.time()
        i+=1
        if i > 1000:
            return -1;

    i = 0
    while GPIO.input(echo) == 1:
        StopTime = time.time()
        i+=1
        if i > 1000:
            return -1;
```

Rys 7. Funkcja distance (analogiczna jak ta, użyta na zajęciach laboratoryjnych)

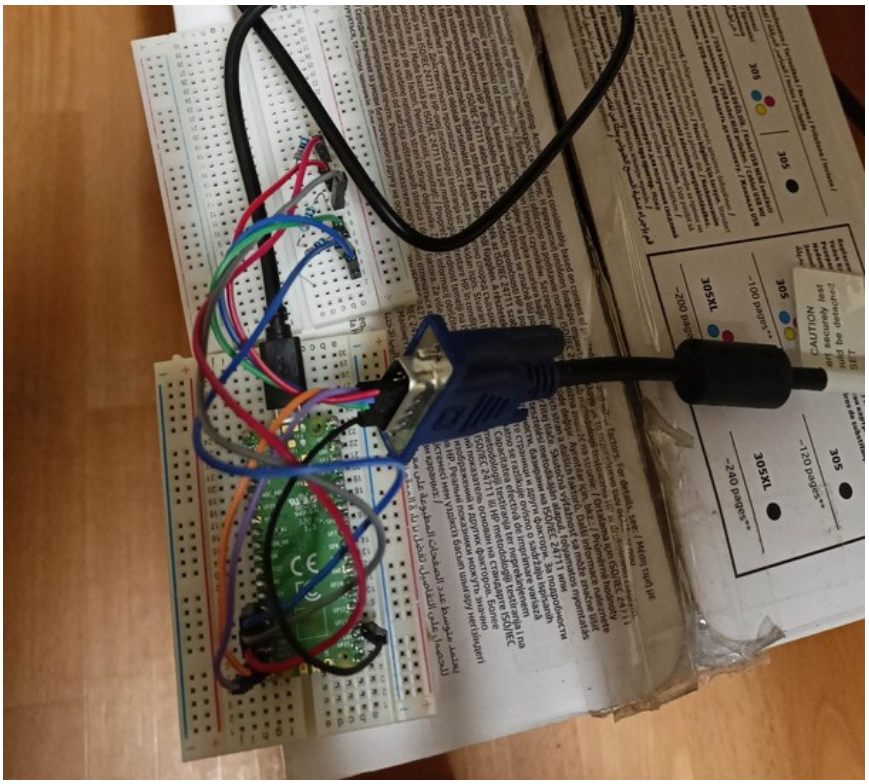
```python
ser = serial.Serial(
        port='/dev/ttyACM0', # raspberry pico podłączone do złącza usb
        baudrate=115200,
        parity=serial.PARITY_NONE, # brak kontroli parzystości
        stopbits=serial.STOPBITS_ONE, # jedynka jako stop bit
        bytesize=serial.EIGHTBITS, # ośmiobitowe bajty
        timeout=1 # timeout po sekundzie
)
```

Rys 8. Inicjalizacja połączenia szeregowego

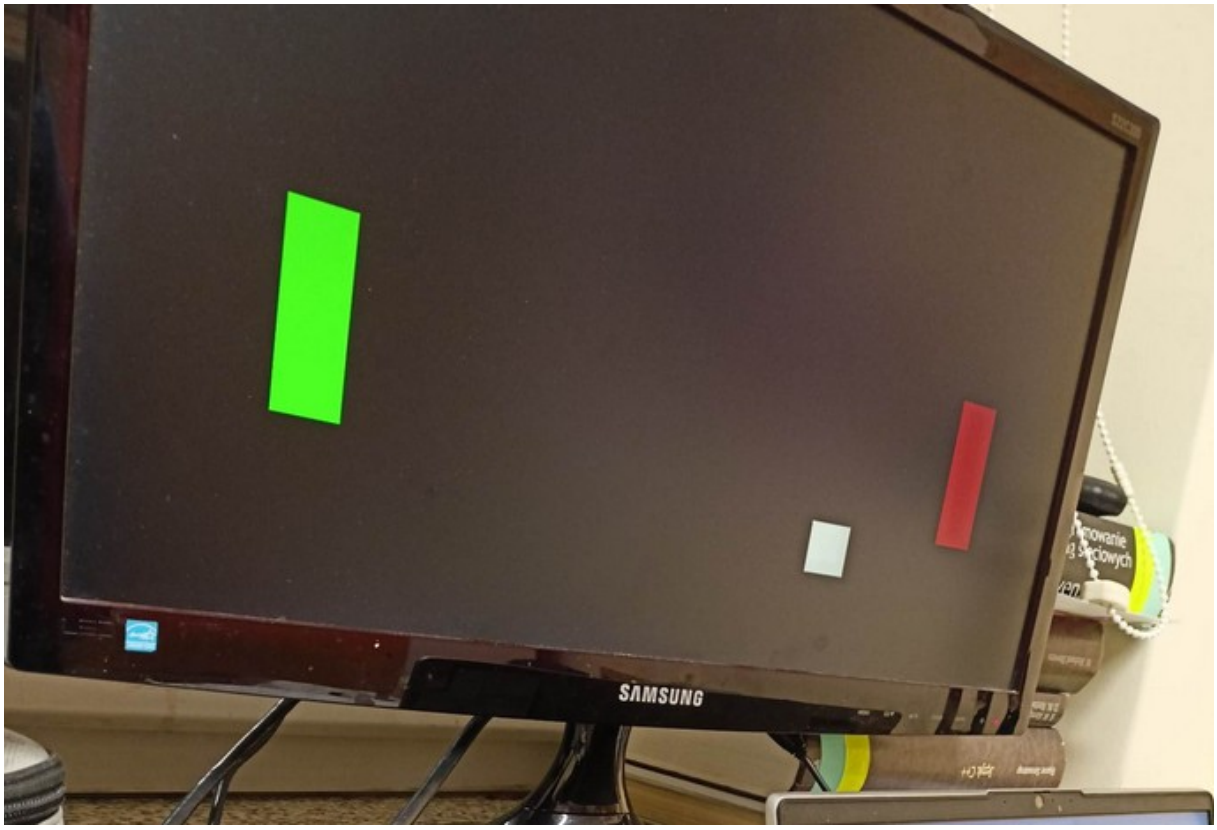
Zdjęcia



Rys 9. Połączenie raspberry pi 4 i sensorów



Rys 10. Połączenie raspberry pico ze złączem VGA



Rys 11. Efekt końcowy

Podsumowanie

Podsumowując, projekt udało się zrealizować zachowując główne jego założenia. Mimo problemów z wydajnością, produkt nadaje się do użytku.