

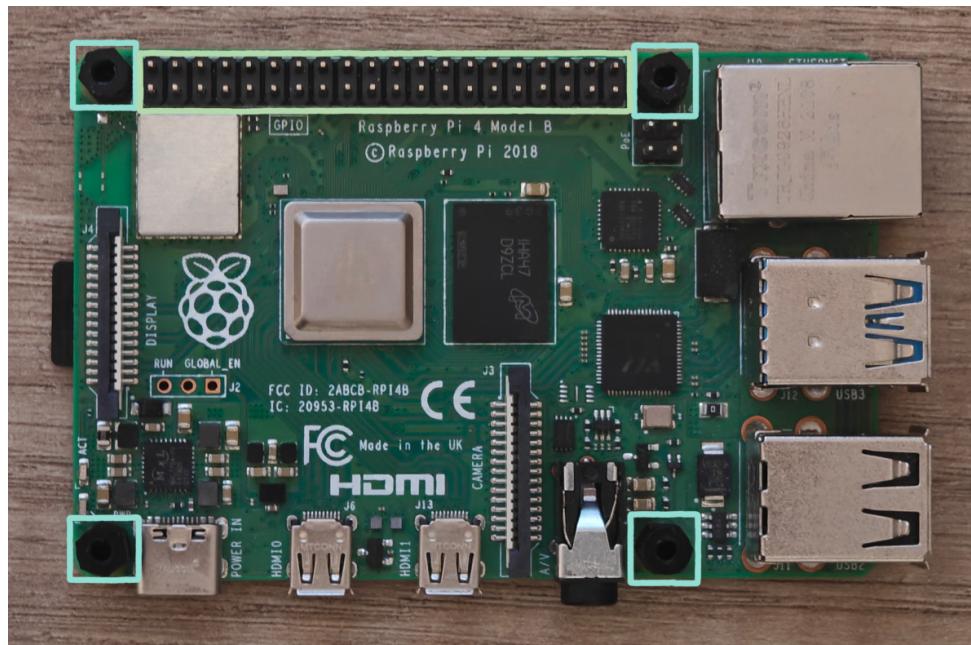
1. Informacje wstępne

Dokument przygotowany jest do druku dwustronnego w kolorze – w takim formacie każde laboratorium zajmuje jedną kartkę, opisując połączenia na przedniej stronie, a rozwiązania zadań (2.3, 3.3, 4.3, 5.3, 6.3) na tylnej.

1.1 Raspberry Pi

Na poniższym rysunku [1.1] płytka *Raspberry Pi* kolorem zielonym zaznaczone jest złącze *GPIO*, wszystkie dalsze rysunki będą w tej samej orientacji – z tym złączem po górnej stronie rysunku. Kolorem jasnoniebieskim zaznaczone są złączki mocujące moduły podrzędne, można wykorzystać je wraz z plastikowymi śrubkami do bezpieczniejszego przymocowania modułów.

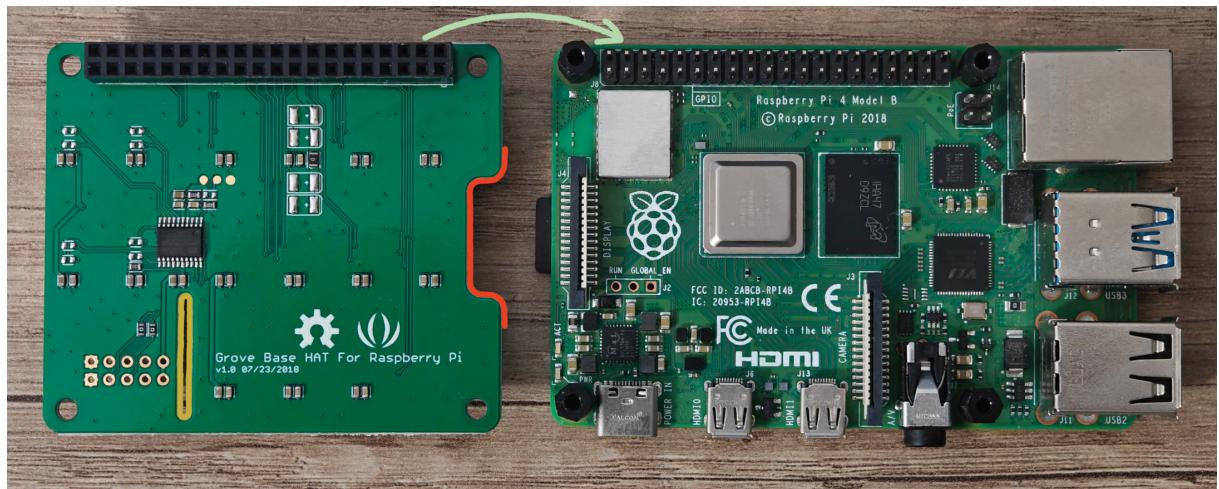
Moduły mocuje się wciskając znajdujące się po ich dolnej stronie panel pasujący do interfejsu *GPIO* i opcjonalnie przykręcając złączki. Moduły są zaprojektowane w taki sposób, że w prawidłowej orientacji nie wystają poza granice płytki *Raspberry Pi*.



Rysunek 1.1: Płytki *Raspberry Pi*

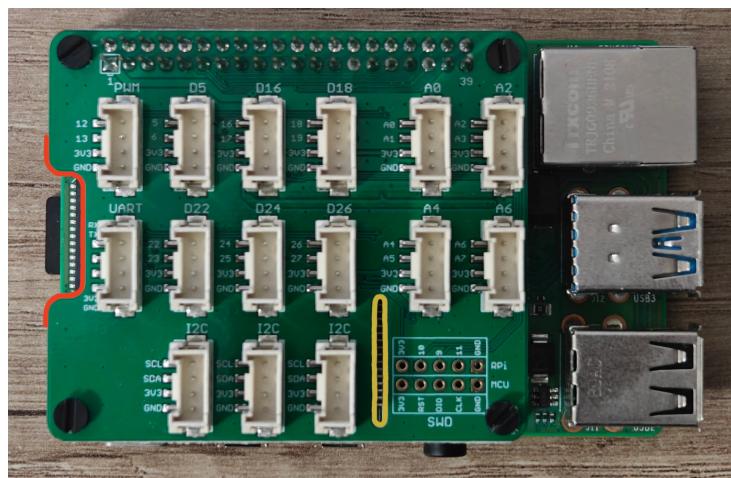
1.2 Grove Base HAT

Grove Base HAT jest wykorzystany w dwóch zadaniach. *Grove Base HAT* odwrócony tak, aby widzieć jego port *GPIO* obok portu *GPIO* płytki [1.2], ma wcięcie (zaznaczone kolorem czerwonym) po prawej stronie oraz szczelinę (kolor żółty) po lewej dolnej stronie. Montaż w obu przypadkach jest identyczny, a różni się jedynie przymocowaniem rozszerzeń do modułu Grove.



Rysunek 1.2: Moduł *Grove Base HAT* obok płytki

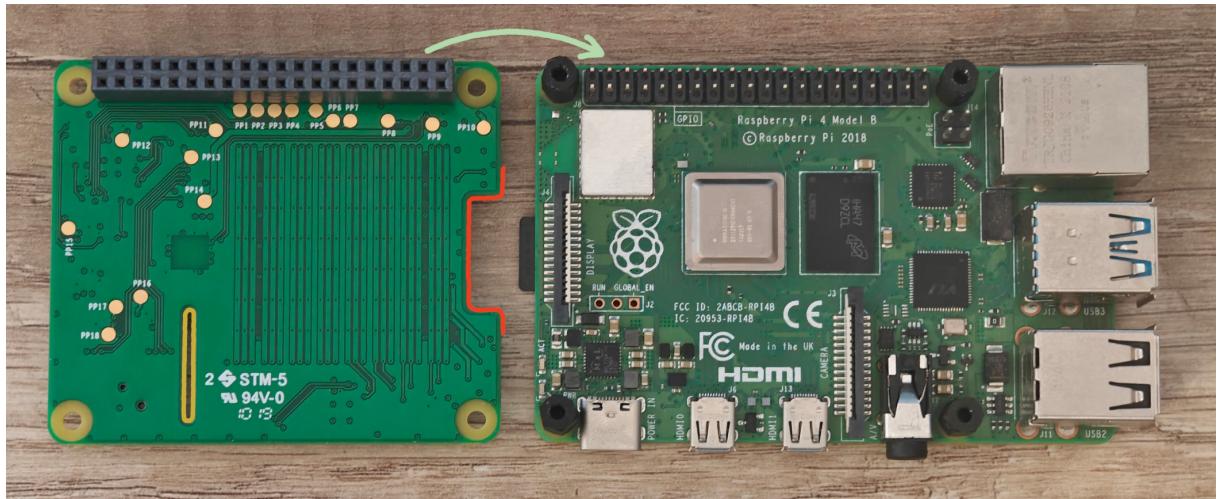
Przymocowanie sprawi, że to wcięcie będzie po stronie lewej, a szczelina po prawej, ale nadal na dole [1.3].



Rysunek 1.3: Przymocowany moduł *Grove Base HAT*

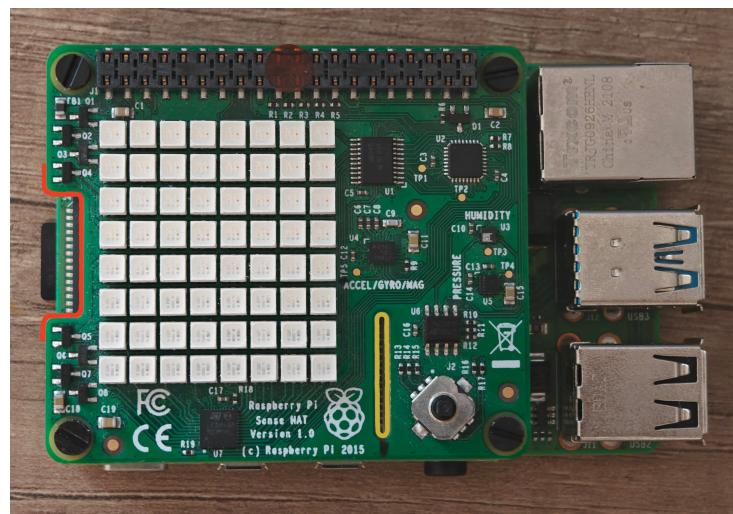
2. Laboratorium I

W tym laboratorium wykorzystano moduł *Sense HAT*. Na rysunku [2.1] użyto te same kolory pomocnicze, co w przypadku *Grove Base HAT*.



Rysunek 2.1: Moduł *Sense HAT* obok płytki

Przymocowanie sprawi, że to wcięcie będzie po stronie lewej, a szczelina po prawej, ale nadal na dole [2.2].



Rysunek 2.2: Przymocowany moduł *Sense HAT*

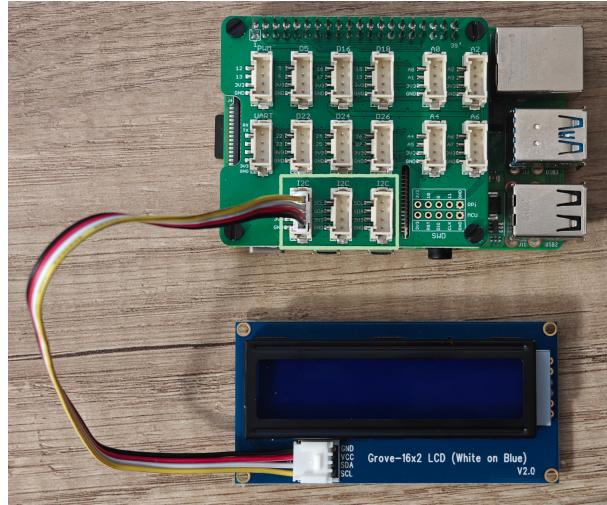
```
# Z2.  
def set_imu():  
    sense.set_imu_config(True, True, False)  
  
# Z3.  
def get_yaw():  
    return sense.get_orientation_radians()['yaw']  
  
# Z4.  
def draw_compass(yaw):  
    sense.set_pixels(calculate_leds(yaw))
```

Rysunek 2.3: Rozwiązańa laboratorium I

3. Laboratorium II

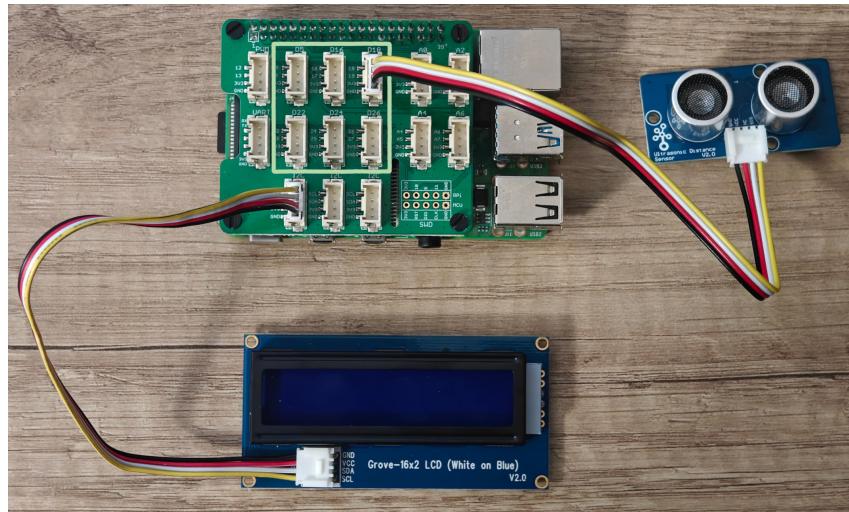
Laboratorium wykorzystuje *Grove Base HAT*, podłącz najpierw ten moduł.

Wyświetlacz LCD wykorzystuje port **I^2C** . Laboratorium zawiera zadanie odczytania podłączonego numeru portu – wykorzystaj losowy z portów I^2C (kolor jasnozielony) [3.1].



Rysunek 3.1: Montaż rozszerzenia – wyświetlacz LCD

Czujnik ultradźwiękowy wykorzystuje port **cyfrowy**. Laboratorium zawiera zadanie odczytania podłączonego numeru portu – wykorzystaj losowy z portów cyfrowych [3.2].



Rysunek 3.2: Montaż rozszerzenia – moduł ultradźwiękowy

```

# Z1.
LCD_address = 0x3E

# Z2.
# w zależności od połączeń fizycznych
Sonar_port = '5, 16, 18, 22, 24, albo 26'

# Z3.
sonar = GroveUltrasonicRanger(Sonar_port)
display = GroveDisplay(LCD_address)

# Z4.
def time_to_dist(t: float) -> float:
    sekundy = t / 10**6
    metry = sekundy * 343 # m/s
    centymetry = metry * 100
    jedna_strona = centymetry / 2
    return jedna_strona

# Z5.
def show_measurement(val: float, prev: float):
    display.setCursor(0, 0)
    display.write(f'cur: {val:.2f} cm')
    display.setCursor(1, 0)
    display.write(f'prv: {prev:.2f} cm')

async def z6():
    # Z6.
    previous = 0
    while True:
        current = time_to_dist(sonar.get_time())
        show_measurement(current, previous)
        previous = current
        await sleep(1)

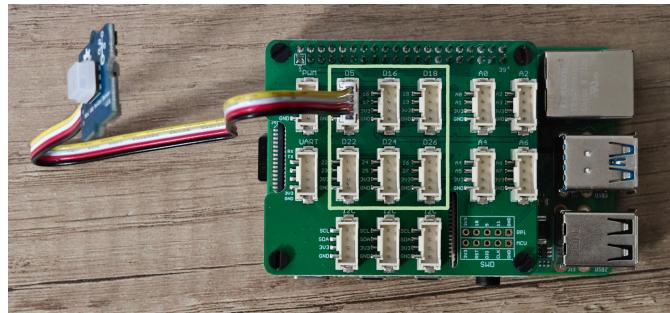
```

Rysunek 3.3: Rozwiązania laboratorium II

4. Laboratorium III

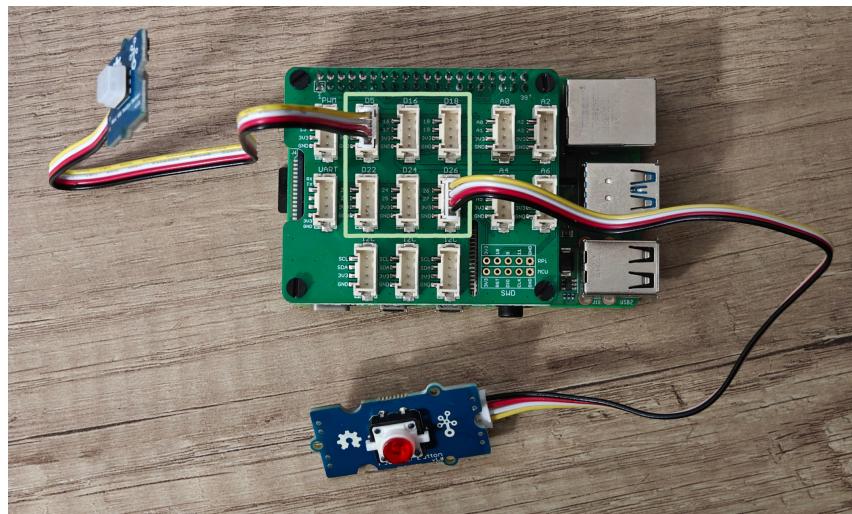
Laboratorium wykorzystuje *Grove Base HAT* – podłącz najpierw ten moduł.

Czujnik ruchu wykorzystuje port **cyfrowy**. Laboratorium zawiera zadanie odczytania podłączonego numeru portu – wykorzystaj losowy z portów cyfrowych [4.1].



Rysunek 4.1: Montaż rozszerzenia – czujnik ruchu

Przycisk LED wykorzystuje port **cyfrowy**. Laboratorium zawiera zadanie odczytania podłączonego numeru portu – wykorzystaj losowy z portów cyfrowych [4.2].



Rysunek 4.2: Montaż rozszerzenia – przycisk LED

```
# Z1.  
# w zależności od połączeń fizycznych  
sensor_port = '5, 16, 18, 22, 24, albo 26'  
led_pin = '5, 16, 18, 22, 24, albo 26, ale nie ten sam co sensor_port'  
btn_pin = 'led_pin + 1'  
  
# Z2.  
def motion_detected():  
    if motion.flag:  
        btn.led = True  
  
# Z3.  
def btn_pressed():  
    if btn.pressed:  
        btn.led = False  
  
async def z4():  
    # Z4.  
    while True:  
        motion_detected()  
        btn_pressed()  
        await sleep(2)
```

Rysunek 4.3: Rozwiązania laboratorium III

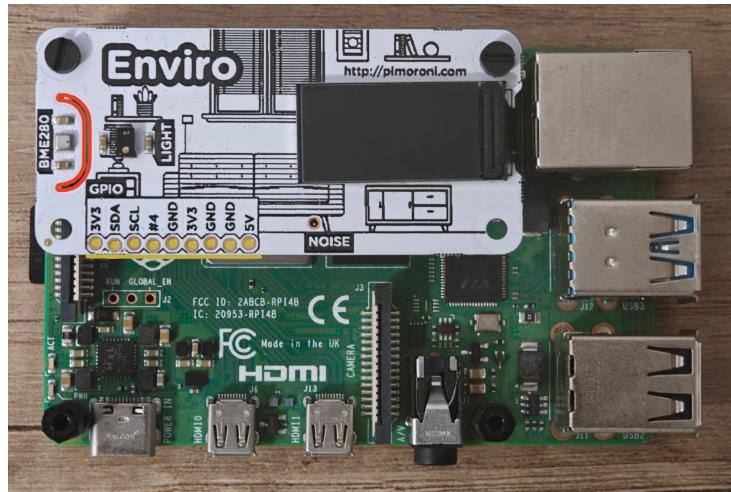
5. Laboratorium IV

Moduł *Enviro HAT* nie ma widocznego na poprzednio montowanych modułach wcięcia, ale w podobny sposób można wykorzystać cięcie odgradzające czujnik temperatury od większej części modułu (kolor czerwony) wraz z otworami wyprowadzającymi *GPIO* (kolor żółty) [5.1].



Rysunek 5.1: Moduł *Enviro HAT* obok płytki

Przymocowanie sprawi, że to cięcie będzie po stronie lewej, a otwory nadal po dolnej stronie [5.2].



Rysunek 5.2: Przymocowany moduł *Enviro HAT*

```

# Z1.
def update_sensors():
    bme280.update_sensor()
    ltr559.update_sensor()

# Z2.
metrics = [
    Metric('CPU', '°C', lambda: get_cpu_temperature()),
    # dodaj linijkę metryk dla temperatury, ciśnienia, wilgotności,
    # → bliskości, jasności, głośności SPL
    Metric('<temperatura>', '°C', lambda: compensate_temperature(bme280.
        → temperature)),
    Metric('<ciśnienie>', 'Pa', lambda: bme280.pressure),
    Metric('<wilgotność>', '%', lambda: bme280.humidity),
    Metric('<bliskość>', 'mm', lambda: proximity_to_mm(ltr559.
        → get_proximity(passive=True))),
    Metric('<jasność>', 'lux', lambda: ltr559.get_lux(passive=True)),
    Metric('<głośność>', 'SPL', lambda: sph0645lm4h_b.spl()),
]
]

# Z3.
def send_message():
    if values['<bliskość>'][-1] < 0.1:
        post(f'https://ntfy.sh/{ntfy_topic}', json={
            k: float(v[-1]) for k, v in values.items()
        })

# Z4.
async def loop():
    update_sensors()
    for i in data_loop(metrics):
        update_sensors()
        insert_vals(metrics)
        display_text(i)
        send_message()
        await sleep(1)

```

Rysunek 5.3: Rozwiązania laboratorium IV

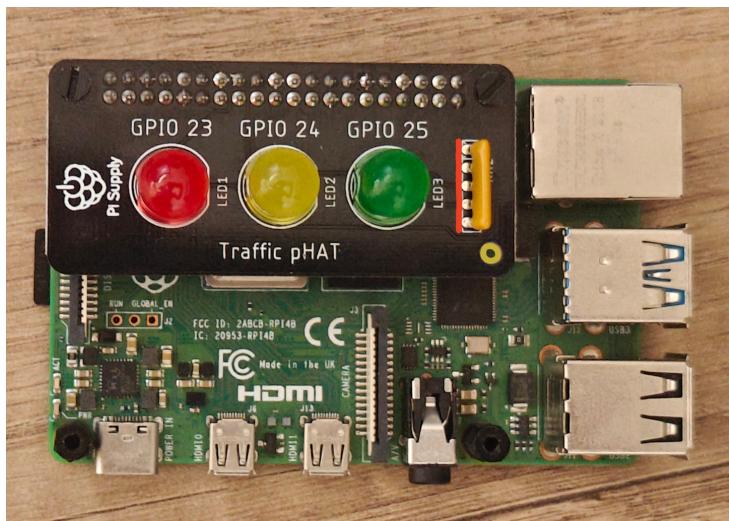
6. Laboratorium V

Moduł *Traffic pHAT* również różni się od poprzednich modułów, ale w podobny sposób do pomocy w orientacji można wykorzystać złącza elementu oznaczonego *RN2* (kolor czerwony) i dolny otwór montażowy (kolor żółty) [6.1].



Rysunek 6.1: Moduł *Traffic pHAT* obok płytki

Po przymocowaniu modułu zaznaczone elementy będą jak w poprzednich modułach odbiciem lustrzanym stanu początkowego [6.2].



Rysunek 6.2: Przymocowany moduł *Traffic pHAT*

```

# Z1.
def gpio_config():
    GPIO.setmode(GPIO.BOARD)

# Z2.
led_red = 16
led_yellow = 18
led_green = 22

# Z3.
def port_config():
    GPIO.setup(led_red, GPIO.OUT)
    GPIO.setup(led_yellow, GPIO.OUT)
    GPIO.setup(led_green, GPIO.OUT)

# Z4.
async def red_to_green():
    GPIO.output(led_yellow, GPIO.HIGH)
    await sleep(2)
    GPIO.output(led_green, GPIO.HIGH)
    GPIO.output(led_red, GPIO.LOW)
    GPIO.output(led_yellow, GPIO.LOW)

async def green_to_red():
    GPIO.output(led_yellow, GPIO.HIGH)
    GPIO.output(led_green, GPIO.LOW)
    await sleep(2)
    GPIO.output(led_red, GPIO.HIGH)
    GPIO.output(led_yellow, GPIO.LOW)

# Z5.
"curl https://ntfy.sh/<identyfikator> -d 'zaliczono'"

# Z6.
from requests import post, get

async def sender():
    for i in range(3):
        await green_to_red()
        await sleep(2)
        post(f'https://ntfy.sh/{ntfy_topic}', data=b'rg')
        await sleep(10)
        post(f'https://ntfy.sh/{ntfy_topic}', data=b'gr')
        await sleep(4)
        await red_to_green()
        await sleep(10)
    post(f'https://ntfy.sh/{ntfy_topic}', data=b'stop')

async def receiver():
    for line in get('https://ntfy.sh/<identyfikator płytka nadrzędnej>/
    ↪ raw', stream=True).iter_lines():
        match line:
            case b'rg':
                await red_to_green()
            case b'gr':
                await green_to_red()
            case b'stop':
                break

```

Rysunek 6.3: Rozwiązań laboratorium V