

Politechnika Warszawska

W Y D Z I A Ł E L E K T R O N I K I
I T E C H N I K I N F O R M A C Y J N Y C H



Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa magisterska

na kierunku Automatyka i Robotyka

Urządzenie zliczające pszczoły w ulu oparte o zestaw czujników
pojemnościowych

Tomasz Żebrowski

Numer albumu 310371

promotor
prof. dr hab. inż. Paweł D. Domański

WARSZAWA 2026

Urządzenie zliczające pszczoły w ulu oparte o zestaw czujników pojemnościowych

Streszczenie.

W niniejszej pracy przedstawiony został system automatycznego zliczania pszczoł na wejściu ula, który wykorzystuje szereg czujników pojemnościowych. Wzrost pojemności elektrycznej kondensatorów, wywołany wysokim współczynnikiem przenikalności elektrycznej owadów, mierzony jest z wykorzystaniem nowatorskiej metody pomiarowej, ograniczającej do maksimum złożoność i koszt systemu, zapewniając przy tym satysfakcjonującą czułość. Opracowane zostały metody przetwarzania zebranych danych pozwalające precyzyjnie określić chwile pojawiania się pszczoły w czujniku, a także kierunek ich ruchu. Zaprezentowano i przeanalizowano dwa algorytmy detekcji: oparty na automacie stanowym wykrywającym sekwencje skoków sygnału, oraz wykorzystujący korelację krzyżową sygnału z wzorcem. Walidacja rozwiązań, składająca się z testów laboratoryjnych wykorzystujących modele pszczoły, a także testów w warunkach rzeczywistych z wykorzystaniem ręcznie anotowanych danych, pozwoliła potwierdzić poprawne działanie stworzonego systemu. Nadaje się on do szacowania natężenia ruchu zbieraczek na wylotku ula, natomiast stosowalność do aproksymacji rozmiaru kolonii pozostaje ograniczona ze względu na występujące błędy zliczania. Zbudowane urządzenie stanowi dowód skuteczności opracowanych metod, które w wyniku przyszłego rozwoju mogą przynieść jeszcze lepsze efekty.

Słowa kluczowe: pszczelarstwo, czujnik pojemnościowy, przetwarzanie danych

Bee hive monitoring device based on capacitive sensors

Abstract.

This thesis presents an automated bee-counting system situated at the hive entrance, utilizing an array of capacitive sensors. The increase in electrical capacitance, induced by the high relative permittivity of the insects, is measured using a novel methodology designed to minimize system complexity and cost while ensuring satisfactory sensitivity. Data processing methods were developed to precisely determine the instances of bee arrival within the sensor, as well as their direction of movement. Two detection algorithms are presented and analyzed: the first based on a finite-state machine detecting signal step sequences, and the second utilizing cross-correlation between the signal and a reference template. Validation of the proposed solution – comprising laboratory tests using bee models and field trials utilizing manually annotated data – confirmed the functional efficacy of the developed system. The device is suitable for estimating the traffic intensity of foragers at the hive entrance; however, its applicability for approximating total colony size remains limited due to inherent counting errors. The constructed prototype serves as a proof of concept for the developed methods, which, through further refinement, offer potential for enhanced performance.

Keywords: bee counting, capacitive sensing, data analytics

Spis treści

| | |
|--|----|
| 1. Wprowadzenie | 7 |
| 1.1. Wstęp teoretyczny | 7 |
| 1.2. Stosowane metody automatycznego zliczania pszczół | 7 |
| 1.3. Cel i zakres pracy | 8 |
| 2. Budowa urządzenia | 9 |
| 2.1. Założenia projektu | 9 |
| 2.2. Prototypowy czujnik pojemnościowy | 9 |
| 2.2.1. Zasada działania | 9 |
| 2.2.2. Układ pomiarowy | 11 |
| 2.2.3. Realizacja sprzętowa | 15 |
| 2.2.4. Testy | 20 |
| 2.3. Konstrukcja urządzenia | 22 |
| 2.4. Układ elektroniczny | 23 |
| 2.4.1. Dobór komponentów | 23 |
| 2.4.2. Realizacja płytki PCB | 25 |
| 2.5. Oprogramowanie mikrokontrolerów | 28 |
| 2.5.1. Algorytm akwizycji | 29 |
| 2.5.2. Komunikacja między mikrokontrolerami | 32 |
| 2.5.3. Komunikacja ze światem zewnętrznym | 36 |
| 2.6. Testy | 38 |
| 2.7. Montaż urządzenia | 40 |
| 3. Algorytm detekcji | 43 |
| 3.1. Zbiór danych przykładowych | 43 |
| 3.1.1. Testy urządzenia w pasiece | 43 |
| 3.1.2. Ręczna anotacja danych | 46 |
| 3.2. Wstępne przetwarzanie danych | 49 |
| 3.2.1. Filtr uśredniający | 49 |
| 3.2.2. Usuwanie trendu | 52 |
| 3.3. Opis proponowanych algorytmów | 55 |
| 3.3.1. Automat stanowy | 55 |
| 3.3.2. Korelacja wzajemna z wzorcem | 60 |
| 3.4. Wyniki testów | 64 |
| 3.5. Wybór algorytmu | 67 |
| 4. Test integracyjny | 69 |
| 4.1. Implementacja algorytmu na sprzęcie | 69 |
| 4.2. Test zliczania pszczół | 72 |
| 5. Rezultaty pracy w warunkach rzeczywistych | 73 |
| 6. Podsumowanie | 74 |
| Podziękowania | 76 |

| | |
|--------------------------------|----|
| Bibliografia | 77 |
| Wykaz symboli i skrótów | 80 |
| Spis rysunków | 82 |
| Spis tabel | 82 |

1. Wprowadzenie

1.1. Wstęp teoretyczny

Pszczelarstwo jest zajęciem wymagającym wiele pracy, wiedzy i doświadczenia. W dzisiejszych czasach, pszczelarze wspierani są różnorodnym sprzętem, umożliwiającym skuteczniejszą opiekę nad ulami, oraz wydajną produkcję miodu. Ponadto, przez ostatnie lata rozwijane były nowoczesne technologie, które, wykorzystując elektronikę, pozwalają dogłębnie monitorować stan pszczelich kolonii. Rosnące w popularność zaawansowane czujniki zapewniają pszczelarzom precyzyjne dane o zdrowiu pszczół, pozwalające na skuteczniejsze zarządzanie pasieką – z pozytywnymi skutkami zarówno dla owadów, jak i efektywności samego biznesu. Stosowane szeroko rozwiązania pozwalają, jak opisują Hadjur [1] i Danieli [2] wraz z zespołami, na monitorowanie takich parametrów jak: temperatura, wilgotność i stężenie CO₂ wewnętrz ula – których nieprawidłowe poziomy mogą powodować choroby; waga całego gniazda, świadcząca o łącznej ilości pszczół, zebranego miodu, i innych zasobów. Stosuje się również czujniki akustyczne, mierzące częstotliwość, intensywność i barwę wibracji, pozwalające na wykrywanie podwyższonego stresu pszczół, a także początków rojenia się.

Jednym z zadań realizowanym przez systemy elektroniczne, wymienianym przez Meiklego i Holsta [3], jest zliczanie pszczół wchodzących do ula i opuszczających go – co pozwala szacować chwilowe natężenie ruchu zbieraczek. Metryka ta zapewnia informacje odnośnie siły i struktury wiekowej kolonii, a także zapotrzebowania i dostępności pożywienia [4]. Zliczanie pszczół ma również znaczenie podczas prowadzenia badań naukowych, które ich dotyczą – między innymi prace Kolmesa i Sama [5], czy Corbeta z zespołem [6].

1.2. Stosowane metody automatycznego zliczania pszczół

Potrzeba liczenia pszczół prowadziła przez lata do rozwoju metod automatyzujących to zadanie. Przegląd kolejnych pojawiających się rozwiązań [7] otwiera ponad stuletnia praca Lundiego z 1925 roku – urządzenie elektromechaniczne, w którym owady przechodzące przez system zapadek aktywowały styki, wysyłając tym samym sygnał do licznika. Podejście to było awaryjne i wymagało częstego czyszczenia z nanoszonego pyłku, jednak pozwoliło autorowi na realizację badań o życiu pszczół, i położyło podwaliny pod dalszy rozwój podobnych technik [8]. W późniejszych latach powstawały systemy zliczania pszczół oparte na układach optycznych. Wykorzystywano fotokomórki na światło podczerwone, którego promień, niewidzialny dla pszczół, był przez nie przecinany, co generowało sygnał. Do pierwszych prac bazujących na tej technice należy dzieło Ericksona wraz z zespołem z roku 1975 [9]. Globalny rozwój technologii umożliwił dalsze rozpowszechnianie rozwiązań opartych na optyce: systemy wykorzystujące technologię LED zaprezentowali m.in. Pešović [10] i Jiang [11] wraz z zespołami, a także, w ostatnich latach, Cunha z zespołem [12]. Autorzy zwróciли uwagę na znaczącą wadę takiego podejścia – zbierający się w torze optycznym pyłek po pewnym czasie uniemożliwiał poprawne działanie systemu, wymuszając regularne

1. Wprowadzenie

czyszczenie wnętrz urządzeń. Pomimo tego ograniczenia, urządzenia zliczające pszczoły oparte na technologii LED są dzisiaj dostępne komercyjnie [13].

W ostatnich latach, w literaturze pojawia się coraz więcej systemów zliczających pszczoły opartych na zaawansowanych algorytmach przetwarzania obrazu i wizji komputerowej. Narzędzie wykrywające pszczoły w obrazie wykorzystujące konwolucyjne sieci neuronowe zostało zaprezentowane przez Tauscha wraz z zespołem [14]. Takie rozwiązania są znacznie bardziej wymagające obliczeniowo, jednak mogą pozwalać na realizację dodatkowych funkcji – system zaprezentowany przez Bilika wraz z zespołem [15] umożliwia wykrycie roztoczy *Varroa* na pszczołach, a rozwiązanie Dunkera wraz z zespołem [16] pozwala na ocenę ilości pyłku transportowanego przez każdą z pszczoł.

Podejściem, które, mimo swej prostoty, wciąż nie zostało szeroko przyjęte jest zastosowanie do wykrywania pszczoł czujników pojemnościowych. Wykorzystanie różnicy pojemności elektrycznej kondensatora wywołanej ruchem pszczoły zostało pierwszy raz zastosowane w pracy Campbell wraz z zespołem [17], która zaprojektowała czujnik składający się z pierścieniowych okładek, między którymi poruszają się owady. Zmodyfikowana wersja podobnego systemu została zaprezentowana przez Perraulta i Teachmana [18], którzy zwróciili uwagę na szereg zalet tego rozwiązania: czujnik pojemnościowy nie jest podatny na zabrudzenia i nie wymaga częstego czyszczenia; ponadto, jest całkowicie nieszkodliwy dla pszczoł. Zwrócono uwagę na podatność systemu na błąd wynikający z grupowania się pszczoł. Problem ten zaadresował Bermig wraz z zespołem [19], konstruując urządzenie *BeeCheck* posiadające siedem bramek pojemnościowych w torze ruchu pszczoł. W ostatnim czasie popularność pojemnościowych liczników pszczołów zaczęła rosnąć, pojawił się nawet internetowy artykuł instruktażowy opisujący, jak wykonać podobne urządzenie samodzielnie [20].

1.3. Cel i zakres pracy

Celem niniejszej pracy jest stworzenie urządzenia zliczającego pszczoły w ulu opartego o zestaw czujników pojemnościowych. Ma ono być montowane na wylotku (szczelinie, przez którą pszczoły wchodzą do ula i opuszczają go) tak, by cały ruch owadów odbywał się poprzez czujniki. Jego zadaniem jest wykrycie każdej pojawiającej się pszczoły, określenie chwili jej przejścia, a także kierunku jej ruchu (wejście/wyjście) – w celu umożliwienia zliczania pszczoł aktualnie znajdujących się w ulu.

Na pracę składać się będą rozwiązania dwóch głównych problemów:

1. zaprojektowanie czujników pojemnościowych i metody akwizycji ich sygnałów, a także konstrukcja urządzenia wyposażonego w ich szereg,
2. opracowanie algorytmu detekcji, umożliwiającego wykrywanie pszczoł na podstawie wyjść poszczególnych czujników pojemnościowych, a także jego implementacja na platformie sprzętowej urządzenia.

Tworzony system zostanie kompleksowo przetestowany, zarówno w warunkach laboratoryjnych, jak i w pracy w środowisku rzeczywistym, by zweryfikować skuteczność zliczania pszczoł i sprawdzić jego podatność na błędy.

2. Budowa urządzenia

2.1. Założenia projektu

Aby możliwe było wykorzystanie urządzenia zliczającego pszczoły do diagnostyki ula, musi ono spełniać szereg kryteriów. Po pierwsze, naturalne funkcjonowanie kolonii nie może zostać znacząco zakłócone. Należy zadbać, by ruch pszczoły został zaburzony w minimalnym stopniu, a wentylacja gniazda nie była zbytnio ograniczona. Ponadto, istotna jest odporność systemu na warunki atmosferyczne i aktywność pszczoły: m.in. na zmiany nasłonecznienia, wilgotności, zbieranie się pyłu.

Urządzenie ma dostarczać informację o chwilach wejścia/wyjścia, a także kierunku poruszania się pszczoły. Wyprodukowanie czujnika nie może mimo to wymagać trudno dostępnych lub drogich komponentów, aby możliwe było intensywne iteracyjne prototypowanie i ewentualne szerokie wykorzystanie stworzonego systemu.

2.2. Prototypowy czujnik pojemnościowy

By spełnić wszystkie wymienione w poprzedniej sekcji założenia, stworzono autorski układ wykrywający pszczoły w oparciu o czujnik pojemnościowy.

2.2.1. Zasada działania

Pojemność elektryczna kondensatora zależy od przenikalności elektrycznej ośrodka między jego okładkami. W przypadku kondensatora płaskiego, pojemność opisywana jest wzorem:

$$C = \frac{\epsilon_r \epsilon_0 S}{d}, \quad (2.1)$$

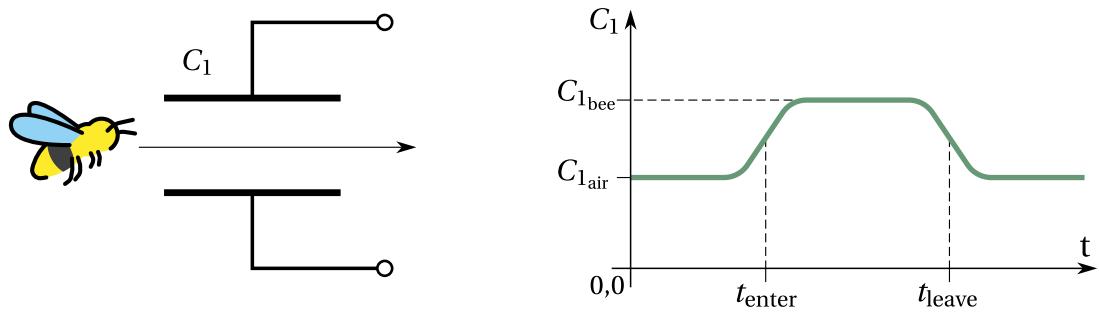
gdzie C – pojemność kondensatora, ϵ_r – względna przenikalność elektryczna ośrodka, ϵ_0 – przenikalność elektryczna próżni, S – pole powierzchni okładki, d – odległość między okładkami [21].

Przenikalność elektryczna powietrza wynosi $\epsilon_r = 1.00$ [22]. Pszczoła nie jest jednorodnym ośrodkiem i jej parametry elektryczne nie były nigdy mierzone, jednak w znacznej części składa się ona z wody – substancji o wysokim $\epsilon_r = 80.2$ (przy 20°C) [23]. W literaturze proponowana jest arbitralna wartość przenikalności elektrycznej pszczoły równa $\epsilon_r = 50$, która znajduje pośrednie potwierdzenie w eksperymentach [17].

Wprowadzenie pszczoły pomiędzy okładki kondensatora spowoduje zatem wzrost średniej przenikalności elektrycznej ośrodka, zwiększając tym samym pojemność układu. Na rysunku 2.1 przedstawiony został układ kondensatora oraz teoretyczny wykres pojemności elektrycznej w zależności od położenia przemieszczającej się pszczoły. Na podstawie kształtu przebiegu pojemności określić można momenty wejścia i opuszczenia kondensatora (odpowiednio t_{enter} i t_{leave} na rysunku).

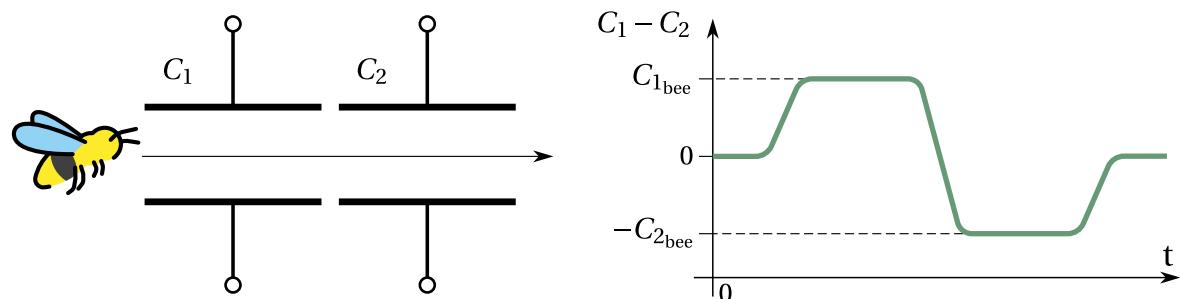
Przedstawiony układ nie umożliwia natomiast określenia kierunku poruszania się pszczoły w czujniku. Aby zapewnić tę funkcję, układ rozszerzony jest o drugi kondensator. Schemat takiego układu został przedstawiony na rysunku 2.2. Wyjście czujnika stanowi sygnał $C_1 - C_2$. Założono, że kondensatory są identyczne, a więc gdy nie znajduje się w nich

2. Budowa urządzenia

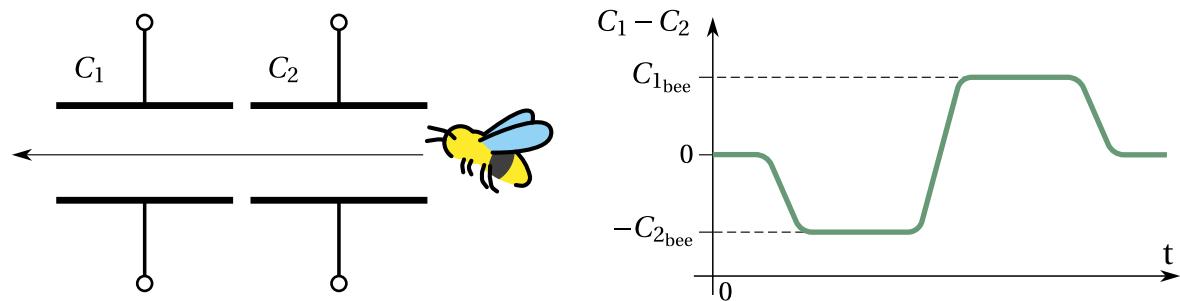


Rysunek 2.1. Schemat układu kondensatora z poruszającą się ruchem jednostajnym pszczołą oraz odpowiadający przebieg pojemności elektrycznej C_1 .

pszczoła, to $C_1 - C_2 = 0$. Ruch pszczoły między okładkami generuje na wyjściu bipolarny, środkowosymetryczny impuls, który daje się zidentyfikować na podstawie kształtu, pozwalając na wykrycie pszczół w czujniku, oraz pozwala określić kierunek ruchu pszczoły na podstawie kolejności występowania pików dodatniego i ujemnego. Na rysunku 2.3 przedstawiono sytuację, w której pszczoła porusza się w odwrotnym kierunku.



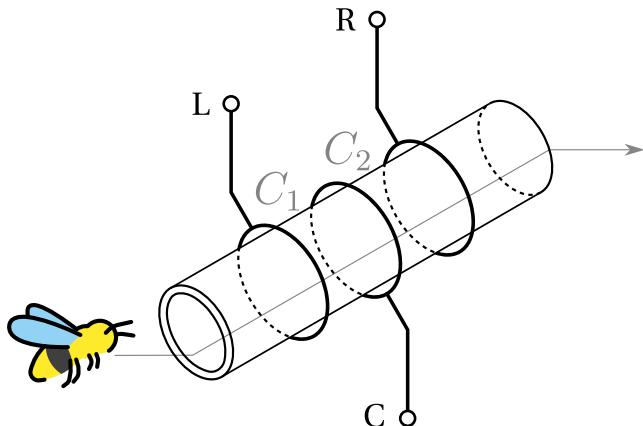
Rysunek 2.2. Schemat i przebieg $C_1 - C_2$ dla układu dwóch kondensatorów.



Rysunek 2.3. Schemat i przebieg $C_1 - C_2$ dla układu dwóch kondensatorów – pszczoła porusza się w odwrotnym kierunku.

Przedstawione ilustracje mają na celu wyłącznie prezentację podstawowej zasady działania czujnika pojemnościowego, a podczas ich tworzenia pominięte zostało wiele zjawisk fizycznych. W praktyce, kondensatory o płaskich okładkach nie sprawdzają się takim zastosowaniem – są podatne na zewnętrzne zakłócenia, a także muszą być znacznie oddalone od siebie by uniknąć wzajemnego zakłócania [17]. Mimo to, wykorzystujące je rozwiązania znalazły wykorzystanie w pracach naukowych, na przykład urządzenie *BeeCheck* [19], kompensujące wady płaskich okładek wykorzystaniem 7 kondensatorów w czujniku.

Znacznie skuteczniejszym podejściem jest wykorzystanie okładek pierścieniowych. Na rysunku 2.4 przedstawiona została ogólna idea tego rozwiązania. Na czujnik składa się rurka z izolatora, przez którą przechodzi owad, oraz trzy pierścienie z przewodnika. Konstrukcja ta realizuje taki sam układ elektryczny jak analizowano poprzednio, przy czym kondensatory C_1 i C_2 mają wspólną okładkę (środkowy pierścień – wyprowadzenie C). Kondensator C_1 znajduje się między wyprowadzeniami L i C , natomiast kondensator C_2 między R i C . Ułożenie to, pozwalając na równoczesne ładowanie obu kondensatorów, umożliwia łatwy pomiar różnicy pojemności $C_1 - C_2$.

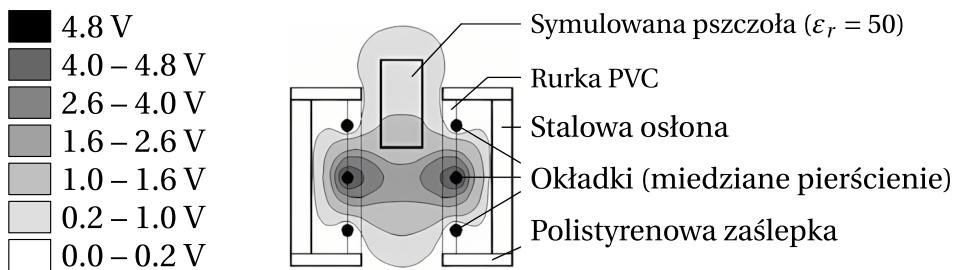


Rysunek 2.4. Konstrukcja czujnika opartego na elektrodach pierścieniowych.

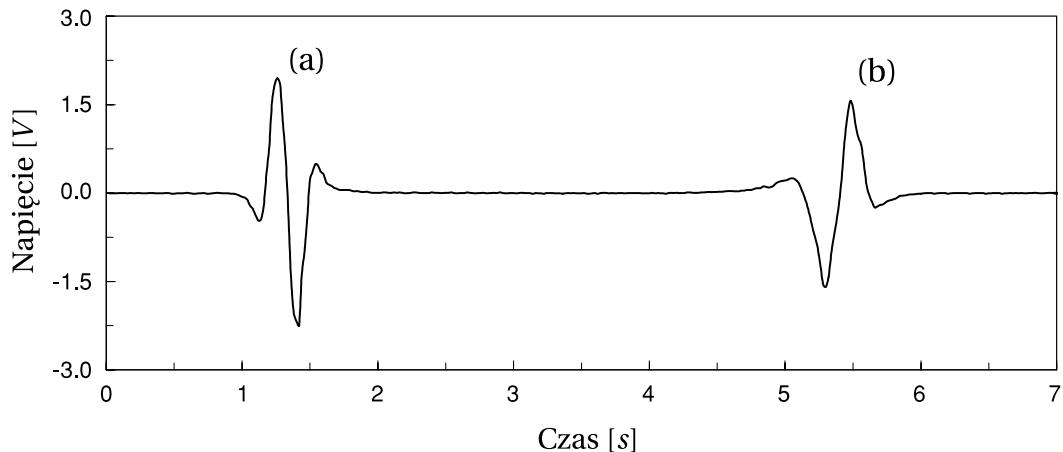
Działanie czujnika tego typu zostało zasymulowane numerycznie przez Campbell wraz z zespołem [17]. Przyjęto: średnicę wewnętrzną tunelu 15 mm, okładki kondensatora wykonane z drutu o średnicy 1.7 mm, odległość między okładkami 7 mm. Pszczołę zamodelowano jako cylinder o długości 12 mm i średnicy 6 mm. Wyniki tej symulacji przedstawione zostały na rysunkach 2.5 i 2.6. Model czujnika zakładał dodatkowe obudowanie całości stalowym ekranem i zastosowanie polistyrenowych zaślepek. Na rysunku 2.5 widać, że pszczoła (symulowana jako cylinder o $\epsilon_r = 50$) wchodząca do tunelu powoduje niesymetryczne rozłożenie pól elektrycznych w kondensatorach. Przejście pszczoły skutkuje pojawieniem się na wyjściu czujnika asymetrycznych, bipolarnych impulsów (rysunek 2.6), podobnych do tych na przebiegach z rozważań teoretycznych (rysunki 2.2, 2.3). W symulacji przetestowano dodatkowo odpowiedź układu na pszczoły różnych rozmiarów (impulsy a , b na rysunku 2.6). Zaobserwować można pewną różnicę w wyglądzie wygenerowanych przebiegów, jednak ogólny kształt nadal pozwala na rozpoznanie momentu oraz kierunku przejścia pszczoły.

2.2.2. Układ pomiarowy

Sygnal wyjściowy omawianego czujnika pojemnościowego jest trudny do zmierzenia. Pojemności kondensatorów C_1 , C_2 wynoszą zaledwie ok. 0,5 pF [17], a przy pojawieniu się pszczoły zwiększą się tylko o kilka–kilkanaście procent. Pomiar tak małych pojemności wymaga specjalistycznego sprzętu oraz niezwykłej dbałości podczas projektowania układu. Pojemności pasożytnicze ścieżek na płytce drukowanej mogą być przy słabym projekcie nawet o rząd wielkości wyższe niż pojemność badanego kondensatora. Szczegółowe założ-



Rysunek 2.5. Przekrój poprzeczny czujnika oraz linie ekwipotencjalne – symulacja przeprowadzona przez Campbell wraz z zespołem. [17]



Rysunek 2.6. Impulsy wygenerowane przez pszczołę przechodzącą przez czujnik. Wyniki symulacji. Różnica pojemności kondensatorów była przetwarzana na napięcie przez liniowy przetwornik. (a) – duża pszczoła opuszczająca ul; (b) – mała pszczoła wchodząca do ula. [17]

żeniami, którymi kierowano się przy projekcie PCB by zminimalizować ich wpływ, zostały szczegółowo opisane w sekcji 2.4.2.

W literaturze znaleźć można przykładowe układy pomiarowe, które działają na zasadzie różnicy przesunięcia fazowego. Wspólna okładka kondensatorów eksytowana jest sygnałem okresowym (generowanym z pomocą układu Intersil ICL8038 [17] lub NE555P [20]). Kondensatory czujnika połączone są z masą układu przez rezystory, tworząc razem filtr górnoprzepustowy przesuwający fazę sygnału w zależności od pojemności. Przesunięte sygnały są następnie odejmowane z pomocą precyzyjnego wzmacniacza różnicowego. Następnie następuje demodulacja i wzmacnianie sygnału. W efekcie, wyjście układu stanowi sygnał o napięciu proporcjonalnym do różnicy pojemności obu kondensatorów.

Taki sposób pomiaru został z powodzeniem zastosowany w urządzeniach zliczających pszczoły, jednak jest bardzo kosztowny. Konieczne jest użycie układów scalonych o wysokiej precyzyji – szacunkowy koszt układów scalonych potrzebnych do obsłużenia jednej pary kondensatorów to 231 zł¹. Całe urządzenie ma się składać z ośmiu równoległych tuneli, łączna cena układów scalonych wyniosłaby 1848 zł znacznie przekraczając limit kosztu całego urządzenia.

Alternatywnym podejściem jest zastosowanie dedykowanego scalonego układu przetwornika pojemnościowo-napięciowego. Przykładem takiego układu odpowiedniego do

¹ Ceny na dzień 09.10.2025 za: ICL8038, AD620, AD630, LF411.

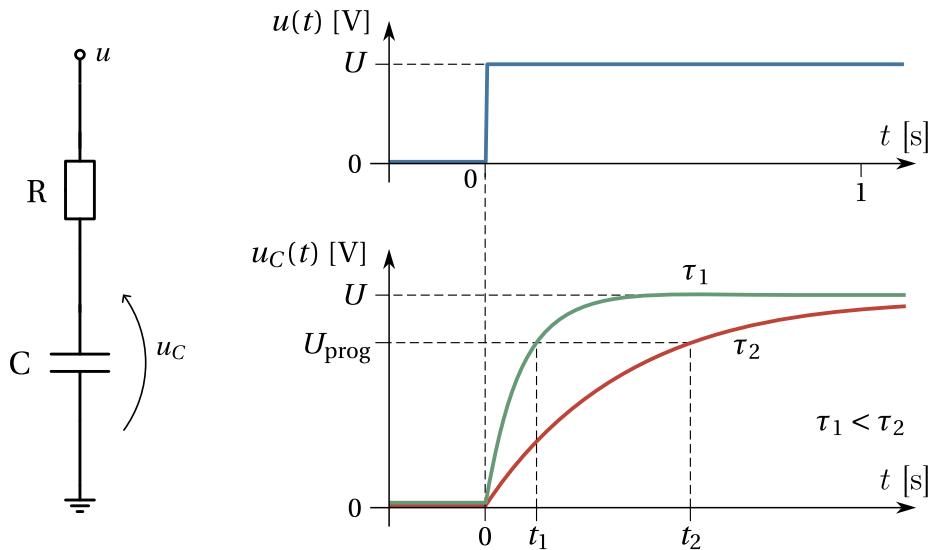
małych pojemności jest AD7746, zastosowany w pracy Perraulta [18]. Jest on przetwornikiem dwukanałowym, wystarczy zatem jeden na oba kondensatory w pojedynczym czujniku [24]. Koszt zakupu jednego układu scalonego to 67 zł (sumarycznie 536 zł na 8 tuneli), co jednak wciąż przekracza zakładany budżet.

W ramach niniejszej pracy opracowany został prosty i niezwykle tani sposób akwizycji sygnału z czujnika pojemnościowego. Jakość pomiaru jest znacznie niższa niż w opisanych wcześniej rozwiązaniach, lecz uzyskane przebiegi czasowe wciąż pozwalają skutecznie wykrywać ruch pszczół i określić jego kierunek. Schemat proponowanego układu przedstawiony jest na rysunku 2.8. Jego działanie opiera się na różnicy czasu ładowania kondensatorów o różnych pojemnościach.

Rozważmy układ pojedynczego kondensatora (rysunek 2.7). Odpowiedź $u_C(t)$ na skok napięcia u od 0 do U dana jest jako:

$$u_C(t) = U \left(1 - e^{-\frac{t}{\tau}}\right), \quad (2.2)$$

gdzie $\tau = RC$ — stała czasowa układu [25]. Przykładowe przebiegi odpowiedzi skokowej przedstawione zostały na rysunku 2.7. Większa stała czasowa wiąże się z wolniejszym narastaniem napięcia na kondensatorze. Przyjmując pewne napięcie progowe U_{prog} , możemy określić czas ładowania kondensatora jako czas od skoku u do przekroczenia U_{prog} przez u_C . Ponieważ $u_C(t)$ jest funkcją rosnącą, to dla stałych czasowych $\tau_1 < \tau_2$ odpowiadające im czasy ładowania spełniają zależność $t_1 < t_2$. Odpowiedni dobór U_{prog} ma wpływ na jakość działania systemu: przy niskim progu t_1 oraz t_2 będą niewielkie i bardzo zbliżone do siebie. Wysoki próg zwiększa podatność na szумy w pomiarze u_C . Wybór konkretnej wartości U_{prog} został szerzej opisany dalej.



Rysunek 2.7. Ładowanie kondensatora: schemat układu i przebiegi napięć.

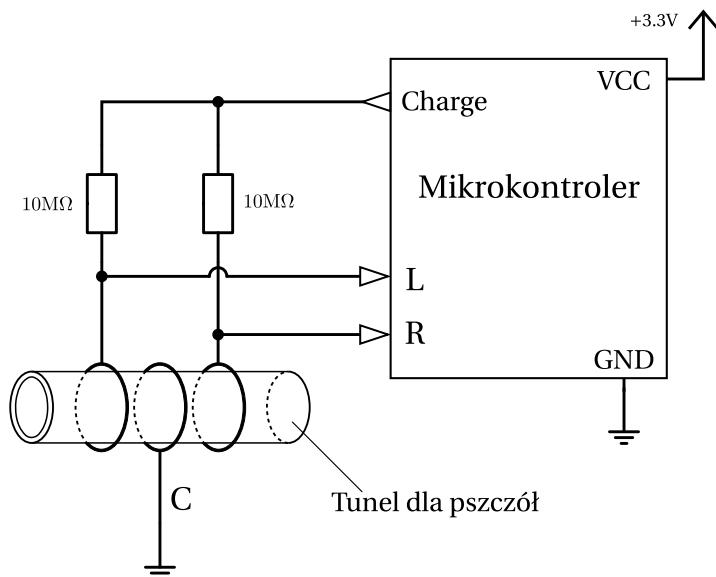
W układzie czujnika każdy z kondensatorów ładowany jest przez taką samą rezystancję — stałe czasowe podukładów zależą wyłącznie od pojemności elektrycznej samych

kondensatorów. Metoda wykrywania pszczół stworzona w ramach niniejszej pracy nie została zatem oparta bezpośrednio na różnicy pojemności kondensatorów, tylko na różnicy w czasie ich ładowania.

Wyzwaniem technicznym jest sam pomiar czasu ładowania kondensatora o tak niewielkiej pojemności. Przyjmując, jak wcześniej, orientacyjną pojemność kondensatora równą $0,5 \text{ pF}$, wyznaczyć można szacowaną stałą czasową układu (z rezystorem $1\text{M}\Omega$):

$$\tau = R \cdot C = 1\text{M}\Omega \cdot 0.5\text{pF} = 10^6\Omega \cdot 0.5 \cdot 10^{-12}\text{F} = 2 \cdot 10^{-6}\text{s} = 2\mu\text{s}. \quad (2.3)$$

Jak widać, stała czasowa układu przyjmuje wartości rzędu pojedynczych mikrosekund.



Rysunek 2.8. Schemat układu pomiarowego.

Precyzyjny pomiar takich czasów jest zasadniczo niemożliwy z wykorzystaniem sprzętu o niskim koszcie. Z tego powodu, w niniejszej pracy zastosowana została pewna sztuczka: ładowanie kondensatora mierzone jest nie w jednostkach czasu, ale w liczbie wykonania pojedynczych instrukcji mikrokontrolera monitorującego stan wyjść czujnika. Działanie to formalnie nie daje faktycznego pomiaru czasu, jednak wynik pozwala się jako taki pomiar zastosować – co zostanie udowodnione w dalszych częściach pracy. Na rysunku 2.8 przedstawione zostało połączenie czujnika z mikrokontrolerem, natomiast pseudokod 1 pokazuje działanie algorytmu akwizycji danych.

Algorytm inicjalizuje dwa liczniki iteracji odpowiadające dwóm kondensatorom: n_L oraz n_R . Następnie na pin *Charge* wystawiany jest stan wysoki, co rozpoczyna ładowanie kondensatorów. Program iteracyjnie inkrementuje liczniki dopóki piny GPIO *L* oraz *R* nie zostaną podniesione do stanu wysokiego. Stan wysoki występuje po przekroczeniu przez napięcie kondensatora progu jedynki logicznej mikrokontrolera – stanowi on w tym przypadku U_{prog} układu. Wartości progowe są dobierane tak, by minimalizować czas odpowiedzi pinu, ale również podatność na szумy. Optymalizowane przez projektantów mikrokontrolerów kryteria są podobne do tych, które musi spełniać odpowiednio dobrany próg U_{prog} , można się zatem spodziewać, że system wykorzystujący cyfrowe piny GPIO

Pseudokod 1 Algorytm akwizycji pomiaru

```

 $n_L \leftarrow 0$ 
 $n_R \leftarrow 0$ 
 $s_L \leftarrow 0$ 
 $s_R \leftarrow 0$ 
write_gpio(Charge, 1)
while  $s_L = 0$  or  $s_R = 0$  do
     $s_L \leftarrow \text{read\_gpio}(L)$ 
     $s_R \leftarrow \text{read\_gpio}(R)$ 
    if  $s_L = 0$  then
         $n_L \leftarrow n_L + 1$ 
    end if
    if  $s_R = 0$  then
         $n_R \leftarrow n_R + 1$ 
    end if
end while
write_gpio(Charge, 0)
return  $n_L - n_R$ 

```

będzie działał prawidłowo. Po naładowaniu kondensatorów i ustaleniu n_L oraz n_R pin *Charge* ponownie ustawiany jest do stanu niskiego. Przed wykonaniem kolejnego pomiaru kondensatory muszą zostać rozładowane, nie wymaga to jednak rozbudowywania układu, ponieważ zastosowany rodzaj kondensatora szybko rozładowuje się samoistnie. Należy jednak pamiętać, aby na kolejnych etapach projektu wprowadzić ograniczenie częstotliwości próbkowania, aby dać ładunkom elektrycznym czas na rozproszenie.

Zaproponowany algorytm akwizycji oferuje rozdzielczość pomiaru zależną od czasu wykonania instrukcji znajdujących się wewnętrz pętli *while*. Najdłuższą operacją jest odczyt stanu pinu (*read_gpio*) – dla zastosowanego mikrokontrolera ESP32-C3 może on trwać 0.05 – 0.17 μs (w zależności od implementacji) [26][27]. Nawet przy konieczności dwukrotnego wykonania instrukcji, rozdzielczość taka jest wystarczająca na potrzeby tworzonego urządzenia.

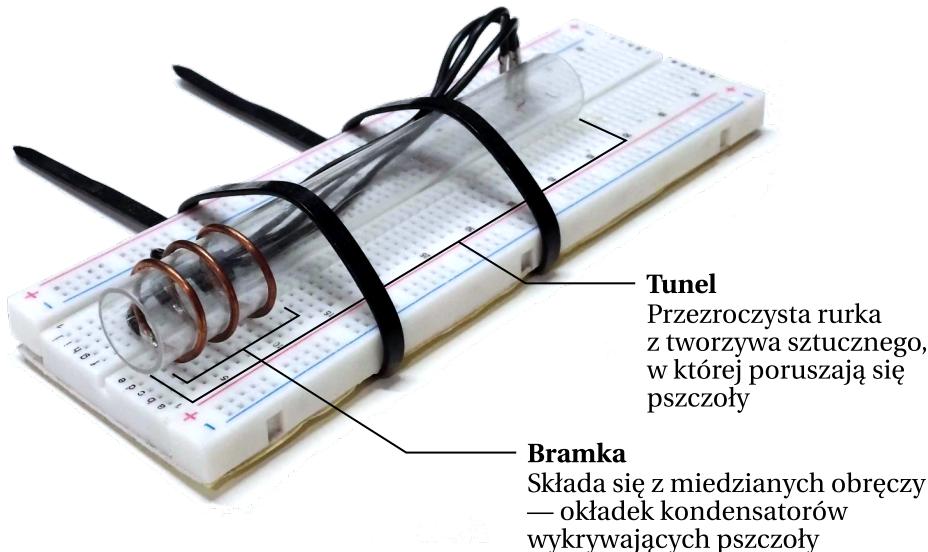
Alternatywnym podejściem, niewykorzystanym w niniejszej pracy z braku wyraźnej potrzeby, jednak mogącym zapewnić wyższą rozdzielczość byłoby wykorzystanie zewnętrznych przerwań mikrokontrolera. Modyfikacja przedstawionego algorytmu polega na usunięciu odczytu GPIO z pętli *while*, zamiast czego s_L oraz s_R ustawiane są na wartość 1 w ramach obsługi przerwań EXTI wynikających z wykrycia pojawienia się zboczy narastających na pinach *L* i *R*.

2.2.3. Realizacja sprzętowa

W celu przetestowania działania opisanego systemu zbudowana została prototypowa konstrukcja czujnika (rysunek 2.9). Tunel czujnika wykonany został z rurki z przezroczystego tworzywa sztucznego, natomiast okładki wykonano z obręczy wygiętych z drutu miedzianego i zlutowanych w miejscułączenia. Wyprowadzenia okładek kondensatora zostały wykonane z izolowanych przewodów o długości ok. 15 cm. W tabeli 2.1 zawarte

2. Budowa urządzenia

zostały parametry opracowanej konstrukcji. Obrane wymiary oparte zostały na wynikach optymalizacji podobnego czujnika w pracy Campbell wraz z zespołem, w której przeprowadzono symulację elektrostatyczną, by następnie dobrać średnice s i rozstaw okładek d , tak by maksymalizować amplitudę sygnału generowanego przez pszczołę w czujniku [17]. Wypracowane w tej pracy kryteria to:

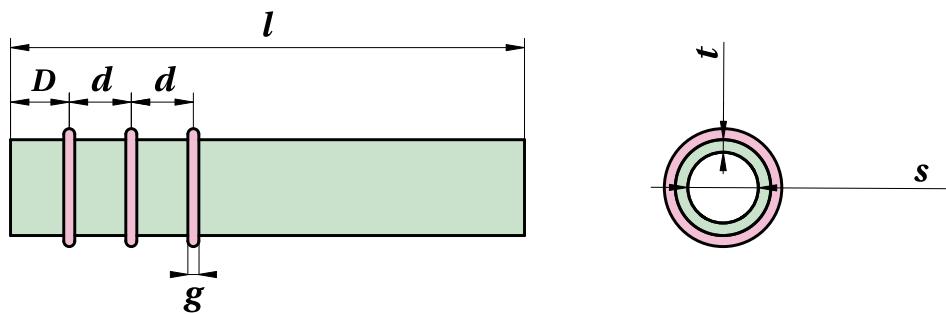


Rysunek 2.9. Prototypowy czujnik pojemnościowy.

1. $\frac{1}{2}s + t < d < s + 2t$ – rozstaw okładek powinien być większy od promienia okładki, ale nie większy od średnicy;
2. g – średnica drutu, z którego wykonane są okładki powinna być jak największa – na ile pozwolą inne ograniczenia konstrukcyjne.

W ramach niniejszej pracy założono, że wewnętrzna średnica tunelu musi wynosić około 14 mm, aby zapewnić swobodny ruch pszczoły, a przy tym uniemożliwić równoległe poruszanie się kilku osobników obok siebie. Średnica zewnętrzna zastosowanej rurki z tworzywa sztucznego to 16 mm, co daje promień okładki wynoszący 8 mm. Dla spełnienia wyżej wymienionego kryterium obrano rozstaw okładek $d = 9$ mm. Do wykonania okładek wybrano drut miedziany o średnicy 1.5 mm.

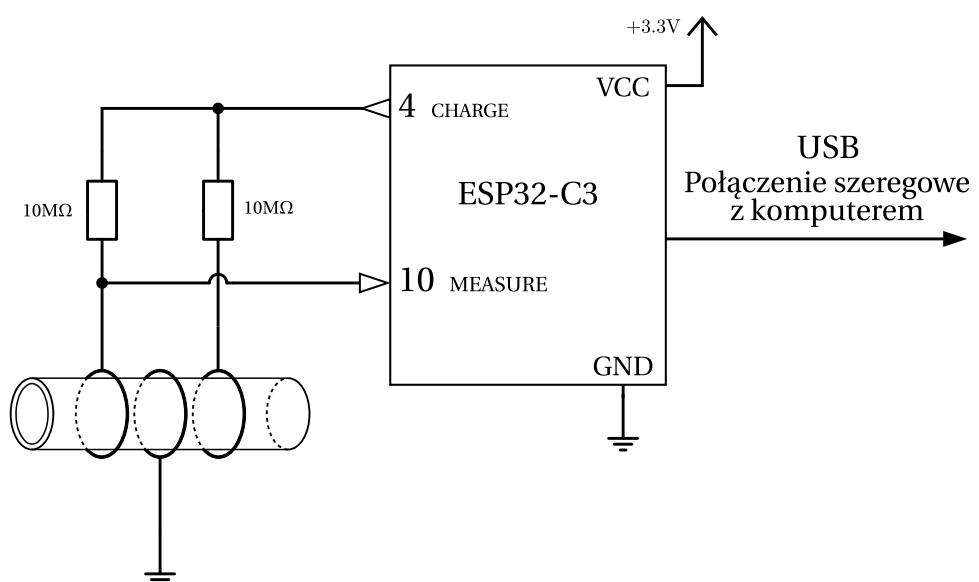
Na uniwersalnej płytce prototypowej stworzono układ elektroniczny zgodny tym przedstawionym na rysunku 2.8. Na potrzeby przetestowania prototypu pominięto jeden z kondensatorów w tunelu i do mikrokontrolera podłączono jedynie okładkę lewego kondensatora. Zastosowany został wspominany wcześniej mikrokontroler ESP32-C3 na płytce deweloperskiej ESP32-C3 Super Mini [26]. Moduł ten wyposażony jest w konwerter USB-UART umożliwiający komunikację urządzenia z komputerem z wykorzystaniem złącza USB. Na mikrokontroler zaimplementowana została uproszczona (mierząca tylko jedną z bramek) wersja algorytmu 1 – program w zamieszczonym dalej pliku `prototype/main.cpp`.



Rysunek 2.10. Wymiary prototypowego tunelu – oznaczenia dla tabeli 2.1.

Tabela 2.1. Wymiary prototypowego tunelu.

| Oznaczenie | Opis | Wartość [mm] |
|------------|-------------------------|--------------|
| <i>l</i> | Długość tunelu | 120 |
| <i>D</i> | Odległość do 1. okładki | 8 |
| <i>d</i> | Rozstaw okładek | 9 |
| <i>g</i> | Średnica drutu okładki | 1.5 |
| <i>t</i> | Grubość ścianki tunelu | 0.85 |
| <i>s</i> | Średnica wewn. tunelu | 14.3 |



Rysunek 2.11. Schemat układu elektronicznego prototypowego tunelu.

2. Budowa urządzenia

prototype/main.cpp

```
1 #include <Arduino.h>
2
3 #define MEASURE_PIN 10
4 #define CHARGE_PIN 4
5
6 uint32_t measureChargeTime();
7
8 void setup() {
9     Serial.begin(115200);
10    pinMode(MEASURE_PIN, INPUT);
11    pinMode(CHARGE_PIN, OUTPUT);
12    digitalWrite(CHARGE_PIN, LOW);
13 }
14
15 void loop() {
16     Serial.println(measureChargeTime());
17     delay(100);
18 }
19
20 uint32_t measureChargeTime() {
21     uint32_t counter = 0;
22     digitalWrite(CHARGE_PIN, HIGH);
23     while (!digitalRead(MEASURE_PIN)) {
24         ++counter;
25     }
26     digitalWrite(CHARGE_PIN, LOW);
27     while (digitalRead(MEASURE_PIN)) {
28     }
29     return counter;
30 }
```

Do programowania ESP32-C3 wykorzystano popularny framework *Arduino*, udostępniający warstwę abstrakcji sprzętowej umożliwiającą szybkie i łatwe tworzenie oprogramowania dla mikrokontrolerów [28]. Framework Arduino pozwala pisać kod programu w języku C++, a także udostępnia wiele wysokopoziomowych funkcji obsługujących układy peryferialne mikrokontrolera. W utworzonym programie, w funkcji `setup()` inicjalizowana jest komunikacja z komputerem, a następnie ustawiane są tryby pracy odpowiednich pinów GPIO. Pin CHARGE ustawiany jest na stan niski, by rozładować bramkę czujnika przygotowując system do pierwszego pomiaru. Kolejne pomiary wykonywane są cyklicznie co ok. 100 ms (funkcja `loop()`). W funkcji `measureChargeTime()` zaimplementowany jest proponowany w niniejszej pracy algorytm akwizycji sygnału. Zmienna `counter` stanowi licznik iteracji algorytmu. Po ustawieniu pinu CHARGE na stan wysoki (rozpoczęciu

ładowania bramki) rozpoczyna się pętla kolejnych odczytów pinu MEASURE. Zliczane są iteracje pętli. Pętla zostaje przerwana po odnotowaniu stanu wysokiego na pinie MEASURE. Na tym etapie w zmiennej counter jest już przechowywany wynik działania algorytmu, jednak w celu umożliwienia kolejnego pomiaru konieczne jest ponowne rozładowanie bramki czujnika. W tym celu pin CHARGE ustawiany jest do stanu niskiego, a następnie w kolejnej pętli oczekuje się rozproszenia ładunku kondensatora – wykrywanego jako stan niski na pinie MEASURE. Pętla aktywnego oczekiwania na rozładowanie kondensatora nie jest koniecznym elementem programu o ile zachowana zostanie wystarczająco dłuża przerwa między kolejnymi odczytami.

Narzędziem komplikacji i wgrywania programów zalecanym przy pracy we framework Arduino jest *Arduino IDE* – zintegrowane środowisko programistyczne łączące edytor kodu, kompilator, menadżer zewnętrznych bibliotek [28]. W niniejszej pracy wykorzystano jednak nowszą alternatywę – *PlatformIO*, zbiór narzędzi deweloperskich umożliwiający realizację tych samych zadań, a także zapewniający dodatkowe funkcjonalności [29]. Jest on dostępny nie jako osobny program, ale jako wtyczka do najpopularniejszych środowisk deweloperskich, co umożliwia użytkownikowi korzystanie z innych zaawansowanych funkcji dzisiejszych edytorów, jak np. zaawansowane dopełnianie składni w *Visual Studio Code*. Opcje komplikacji i wgrywania programu ustawiane są w jednym pliku konfiguracyjnym projektu – *platformio.ini*. Poniżej przedstawiona została konfiguracja stworzona na potrzeby realizowanego prototypu.

```
prototype/platformio.ini
1 [env:esp32-c3-devkitm-1]
2 platform = espressif32
3 board = esp32-c3-devkitm-1
4 framework = arduino
5
6 build_flags =
7     -D ARDUINO_USB_CDC_ON_BOOT=1
8     -D ARDUINO_USB_MODE=1
9
10 monitor_speed = 115200
```

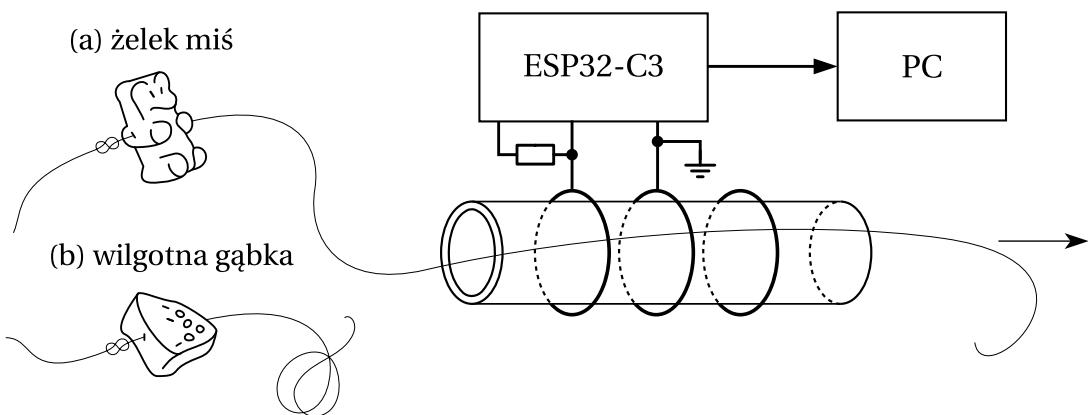
Ustawienie platformy espressif32 oraz frameworku arduino sprawia, że do komplikacji projektu wykorzystane zostaną dedykowane ESP32 narzędzia oraz dołączone będą wymagane pliki nagłówkowe. Zdefiniowano również prędkość komunikacji monitora portu szeregowego (jednego z dostępnych w PlatformIO narzędzi) na wartość zgodną z wcześniej przedstawionym programem. Po odpowiednim skonfigurowaniu projektu możliwe było wgranie utworzonego oprogramowania na mikrokontroler, a następnie przetestowanie proponowanego rozwiązania.

2.2.4. Testy

Po zakończeniu budowy prototypu przeprowadzony został test mający na celu weryfikację opracowanej metody wykrywania pszczoły w tunelu czujnika. Względy praktyczne wykluczyły wykorzystanie żywych pszczoły podczas tego eksperymentu, konieczne było zatem znalezienie ciała, które będzie w stanie je odpowiednio zasymulować. Kluczowe było, by odpowiadało pszczoły rozmiarem i kształtem, a także przenikalnością elektryczną (przez zawartość wody). Wybrane zostały dwa potencjalne modele pszczoły:

- (a) **Żelek miś** – posiada odpowiedni kształt i rozmiar, oraz zawiera wodę. Zastosowanie go do symulowania pszczoły jest polecane w literaturze [20];
- (b) **Wilgotna gąbka** – została wprowadzona jako dodatkowy eksperyment w razie, gdyby żelek posiadał nieodpowiednie właściwości elektryczne. Z gąbki kuchennej wycięta została bryła o wymiarach zbliżonych do pszczoły, która przed samym eksperymentem zanurzono w naczyniu z wodą. Właściwości elektryczne ciała powinny być zbliżone do pszczoły, której większość masy stanowi woda.

Każdy z obiektów został nawleczony na cienką żyłkę. W ramach każdego z dwóch przeprowadzonych eksperymentów żyłkę przeprowadzano przez tunel czujnika, po czym w ciągu kilkudziesięciu sekund kilkukrotnie przeciągano nią model pszczoły. Schemat doświadczenia przedstawiono na rysunku 2.12. Żelek został przeciągnięty przez bramkę czterokrotnie (dwa razy w każdym kierunku), natomiast gąbka jedenastokrotnie (po pięć razy w każdym kierunku i szósty raz tylko w jednym). Pomiary generowane przez czujnik były wysyłane do komputera i zapisywane do pliku.



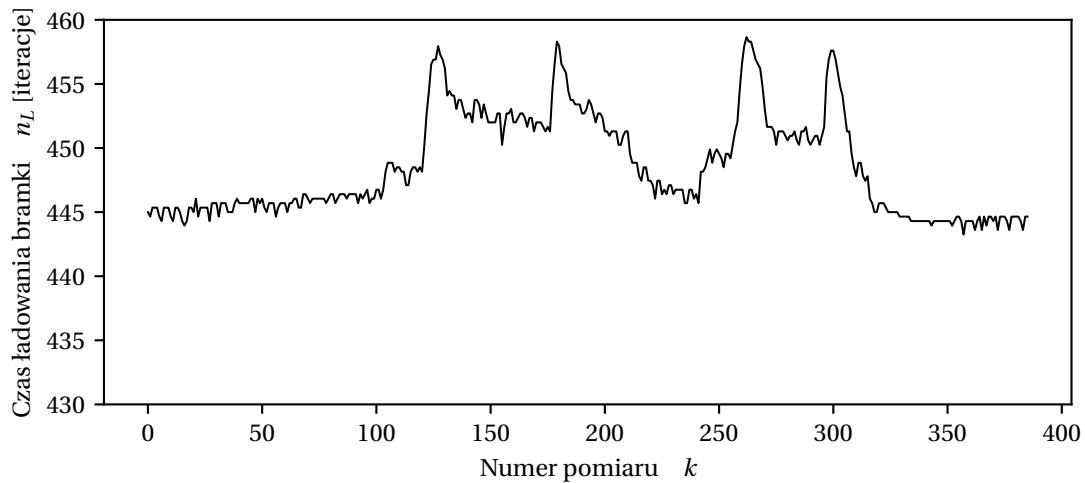
Rysunek 2.12. Schemat przeprowadzonego eksperymentu.

Uzyskane wyniki przedstawione zostały na rysunku 2.13. Przebieg zebrany podczas eksperymentu z wilgotną gąbką (rysunek 2.13b) zawiera wyraźne skoki sygnału wyjściowego w liczbie odpowiadającej przejściom ciała przez bramkę czujnika. Obserwacja przebiegu sygnału na żywo w trakcie prowadzenia doświadczenia pozwoliła dodatkowo potwierdzić, że zmiany widoczne na wyjściu czujnika są spowodowane wprowadzaniem do bramki wilgotnej gąbki. Same wartości skoków są niewielkie w porównaniu do wielkości sygnału – zmierzono ok. 435 iteracji przy pustej bramce, a podczas wykrycia skok o ok. 20 iteracji. Mimo to, zmiany te są wyraźnie widoczne na wykresie przebiegu.

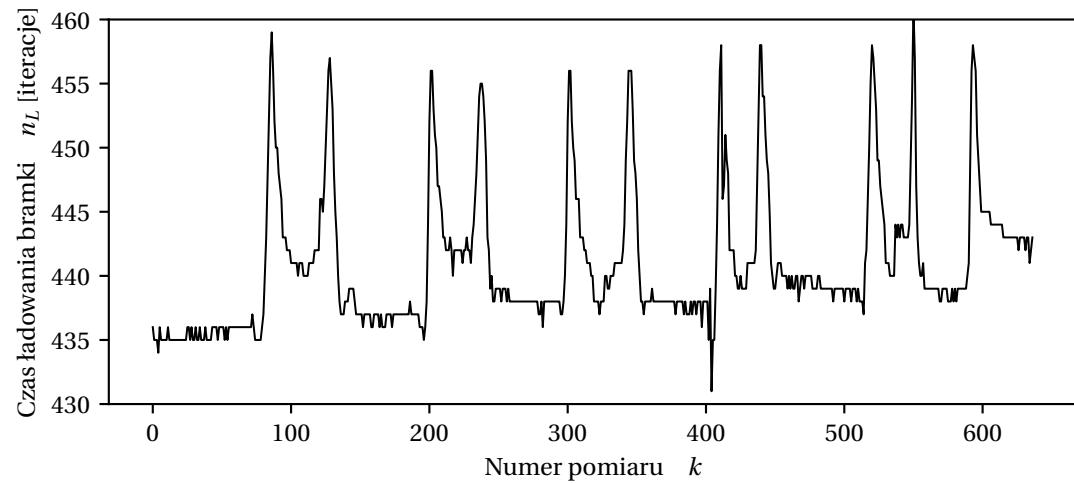
W przypadku doświadczenia z żelkiem misiem (rysunek 2.13a) czujnik również wygenerował skoki sygnału wyjściowego odpowiadające przejściom ciała przez bramkę. Na tym przebiegu można jednak zaobserwować, że amplituda powstających szpylek jest znacznie mniejsza niż w przypadku mokrej gąbki (skok o niecałe 10 iteracji). Najprawdopodobniej wynika to z niższej zawartości wody w żelku.

Rysunek 2.13. Przebiegi wyjściowe czujnika uzyskane podczas przeprowadzonych testów.

(a) Sygnał wyjściowy czujnika podczas eksperymentu z żelkiem misiem



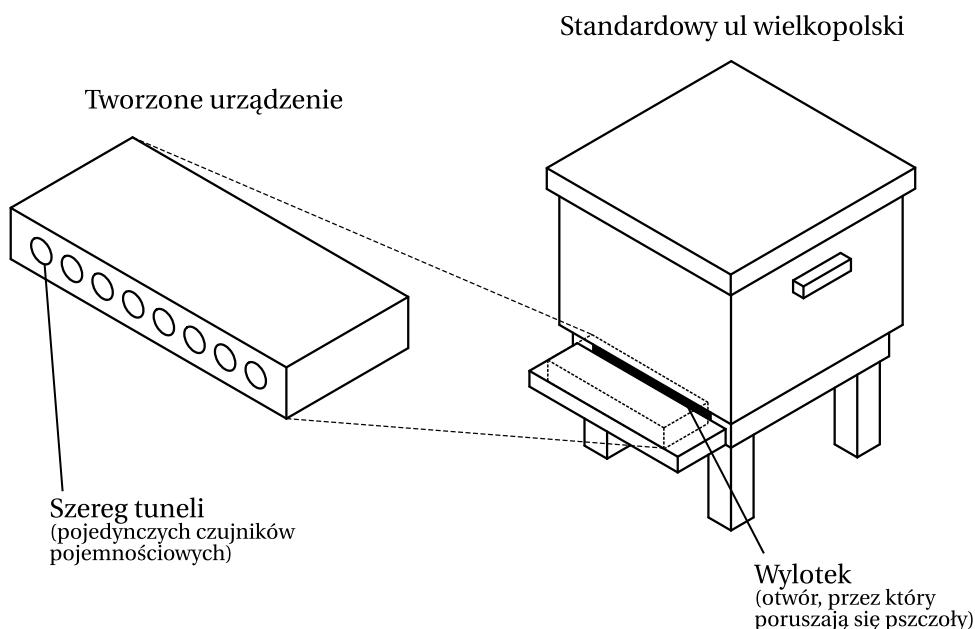
(b) Sygnał wyjściowy czujnika podczas eksperymentu z wilgotną gąbką



Na podstawie przeprowadzonych eksperymentów można stwierdzić, że stworzony według przedstawionej metody czujnik może skutecznie wykrywać ciała pojawiające się w tunelu. Możliwe będzie wykorzystanie go do wykrywania pszczół wchodzących i wychodzących z ula. Obydwia ciała modelujące pszczoły w eksperymentach, pomimo wyraźnych różnic, dały zadowalające wyniki. Ponieważ nie można na tym etapie stwierdzić, które z nich stanowi lepszą symulację prawdziwego owada, w dalszych rozważaniach będą wykorzystywane oba z nich.

2.3. Konstrukcja urządzenia

Pozytywny wynik testu prototypowego czujnika pojemnościowego pozwala kontynuować pracę. W kolejnym kroku zaprojektowany została konstrukcja pełnego systemu. Budowane urządzenie z założenia musi przechwycić cały ruch pszczół wchodzących i wychodzących z ula. Zadanie to jest dość proste w realizacji, ponieważ standardowo ule posiadają tylko jeden otwór, przez który mogą przedostawać się pсы: wąską szczelinę zwaną wylotkiem [30]. Jej wymiary typowo wynoszą 15 cm szerokości i 1.8 cm wysokości. Pełne zasłonięcie wylotka wymaga zastosowania szeregu tuneli z bramkami pojemnościowymi. Ustalono, że przy średnicy tunelu równej 15 mm i zachowaniu odstępów konieczne będzie 8 tuneli. Na rysunku 2.14 przedstawiona została wizualizacja urządzenia oraz miejsca jego montażu na ulu. Długość samych tuneli jest również ograniczona – powinny być krótkie by jak najmniej utrudniać wentylację ula oraz nie wpływać nadmiernie na ruch pszczół.

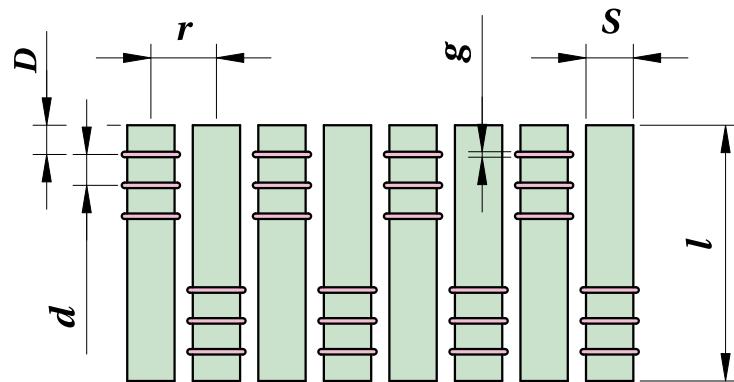


Rysunek 2.14. Wizualizacja konstrukcji i sposobu mocowania urządzenia do ula.

W tabeli 2.2 przedstawione zostały opracowane wymiary konstrukcji, obrane na podstawie parametrów prototypu z następującymi modyfikacjami:

1. Tunele wykonane zostaną z rurki o średnicy mniejszej niż w prototypie – prototyp bazował na urządzeniu stworzonym do zliczania owadów z rodzaju *Bombus* [17]. Konkretny gatunek nie został podany, ale najprawdopodobniej chodzi o któryś z gatunków trzmieli występujących także w Polsce. Owady te są większe od pсы miodnej (*Apis mellifera*), której populacje monitorowane mają być projektowanym urządzeniem. W literaturze traktującej o tym gatunku pojawiają się otwory o rozmiarze od 6 mm [11] do 10 mm [10]. W niniejszej pracy zastosowane zostały tunele z rurki o średnicy wewnętrznej 9 mm, a zewnętrznej 13 mm. Zwężenie tunelu względem prototypu teoretycznie powinno również zapewnić większą amplitudę odpowiedzi, ponieważ zmniejsza średnica okładek, powodując wzrost pojemności elektrycznej bramek.

2. Długość tunelu została skrócona do 70 mm, by pszczoły mogły swobodniej się przemieszczać.
3. Tunele zostały ułożone końcami naprzemiennie, w celu zminimalizowania możliwości wzajemnego zakłócania bramek na sąsiadujących tunelach. Dzięki takiemu ustawieniu wymagany dystans pomiędzy sąsiadującymi tunelami jest niewielki (rozstaw tuneli $r = 18 \text{ mm}$).



Rysunek 2.15. Rozmieszczenie tuneli w czujniku – oznaczenia dla tabeli 2.2

Tabela 2.2. Rozmieszczenie tuneli w czujniku

| Oznaczenie | Opis | Wartość [mm] |
|------------|-------------------------|--------------|
| r | Rozstaw tuneli | 18 |
| D | Odległość do 1. okładki | 8 |
| d | Rozstaw okładek | 8.45 |
| g | Średnica drutu okładki | 1.5 |
| s | Średnica zewn. tunelu | 13 |
| l | Długość tunelu | 70 |

2.4. Układ elektroniczny

W niniejszej sekcji opisana została realizacja układu elektronicznego pozwalającego na obsługę wymaganej liczby czujników pojemnościowych oraz komunikacji ze światem zewnętrznym.

2.4.1. Dobór komponentów

Przedstawiony w poprzedniej części pracy układ akwizycji sygnału nie wymaga wielu elementów (rysunek 2.8). Serce urządzenia stanowi mikrokontroler, i to jego dobór ma kluczowe znaczenie dla działania systemu. W niniejszej pracy rozważono 3 popularne mikrokontrolery: ATmega32U4 (Arduino Micro Pro), STM32F401 (STM32 Blackpill), oraz

2. Budowa urządzenia

ESP32-C3 (ESP32 Super Mini). Kryteria ich oceny oraz parametry poszczególnych propozycji przedstawione zostały w tabeli 2.3.

Tabela 2.3. Porównanie mikrokontrolerów

| Mikrokontroler (Płytki rozwojowa) | ATmega32U4 [31] (Arduino Micro Pro) | STM32F401 [32] (STM32 Blackpill) | ESP32-C3 [26] (ESP32 Super Mini) |
|--------------------------------------|--|-------------------------------------|-------------------------------------|
| Taktowanie | 16 MHz | Do 84 MHz | Do 160 MHz |
| Pamięć Flash | 32 KB | 256 KB | 4 MB |
| Pamięć SRAM | 2.5 KB | 64 KB | 400 KB |
| Napięcie pracy | 5 V | 3.3 V | 3.3 V |
| Liczba GPIO | 18 | 34 | 13 |
| Łączność bezprzewodowa | — | — | Wi-Fi, Bluetooth 5 |

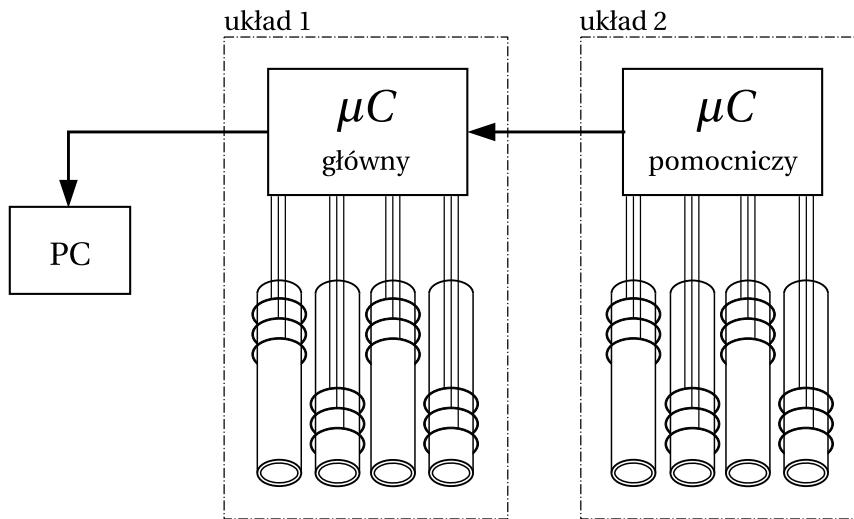
Najważniejszymi kryteriami oceny są:

- **Taktowanie** — od szybkości pracy mikrokontrolera bezpośrednio zależy rozdzielczość, z jaką możliwy jest pomiar sygnału bramki. W tej kategorii najlepszym mikrokontrolerem jest ESP32-C3.
- **Liczba pinów GPIO** — do obsługi ośmiu bramek potrzebne są 24 wyprowadzenia mikrokontrolera. Płytki rozwojowe Arduino Micro Pro oraz ESP32 Super Mini posiadają znacznie mniej pinów, zatem jedynym mikrokontrolerem pozwalającym w pełni zrealizować założenie liczby tuneli jest STM32F401.
- **Łączność bezprzewodowa** — wymagana jest do wydajnej komunikacji ze światem zewnętrznym w praktycznym zastosowaniu urządzenia w pasieku. Jedynym mikrokontrolerem wyposażonym w tę funkcjonalność jest ESP32-C3.

Na podstawie przeprowadzonego porównania wybrano do projektu mikrokontroler ESP32-C3 Super Mini. Znacząco przewyższa on pozostałe propozycje pod wszystkimi względami poza liczbą wyprowadzeń. Niewielka liczba pinów GPIO wynika bezpośrednio z bardzo małych wymiarów płytki rozwojowej, które również stanowią zaletę tego urządzenia w niniejszym projekcie, jako że jest on ograniczony wymiarami.

Niewystarczająca liczba wyprowadzeń mikrokontrolera nie uniemożliwia realizacji założeń projektu. W celu obsługi pełnej liczby bramek stworzono rozwiązanie wykorzystujące dwa ESP32-C3 współpracujące ze sobą. Schemat tego rozwiązania zostało przedstawiony na rysunku 2.16. W jego skład wchodzą dwa układy, z których każdy obsługuje połowę tuneli dla pszczół – wymagane jest po 12 pinów każdego z mikrokontrolerów. Pozostałe pojedyncze piny na płytach rozwojowych wykorzystane są do zapewnienia komunikacji pomiędzy układami. Wyróżnione zostały:

1. Mikrokontroler główny, którego zadaniem jest akwizycja sygnału z 4 bramek, integracja z danymi odebranymi od mikrokontrolera pomocniczego, a także komunikacja ze światem zewnętrznym (np. PC);



Rysunek 2.16. Schemat systemu z dwoma mikrokontrolerami do obsługi ośmiu bramek.

2. Mikrokontroler побoczny, którego zadaniem jest akwizycja sygnału z 4 bramek oraz przesłanie danych do mikrokontrolera głównego.

Układy obu mikrokontrolerów mogą być identyczne, wymagane jest wyłącznie odpowiednie dostosowanie oprogramowania obu z nich – zapewnia to systemowi modularność i uproszcza dalsze prototypowanie. Takie rozwiązanie posiada dodatkową zaletę: zastosowanie dwóch mikrokontrolerów pozwala na umieszczenie ich bliżej czujników pojemnościowych, dzięki czemu wyprowadzenia okładek kondensatorów będą mogły być krótsze, co zmniejszy podatność systemu na zakłócenia i pojemności pasożytnicze.

Dzięki prostocie opracowanego czujnika pojemnościowego, oprócz doboru mikrokontrolera pozostaje tylko wybrać odpowiednie rezystory. Aby sygnał różnicowy z bramek czujnika był zbalansowany, należy zadbać, by tory ładowania oby kondensatorów miały równą rezystancję. W niniejszej pracy zastosowane zostały rezystory o rezystancji $10\text{ M}\Omega$ z rozrzutem produkcyjnym 1%.

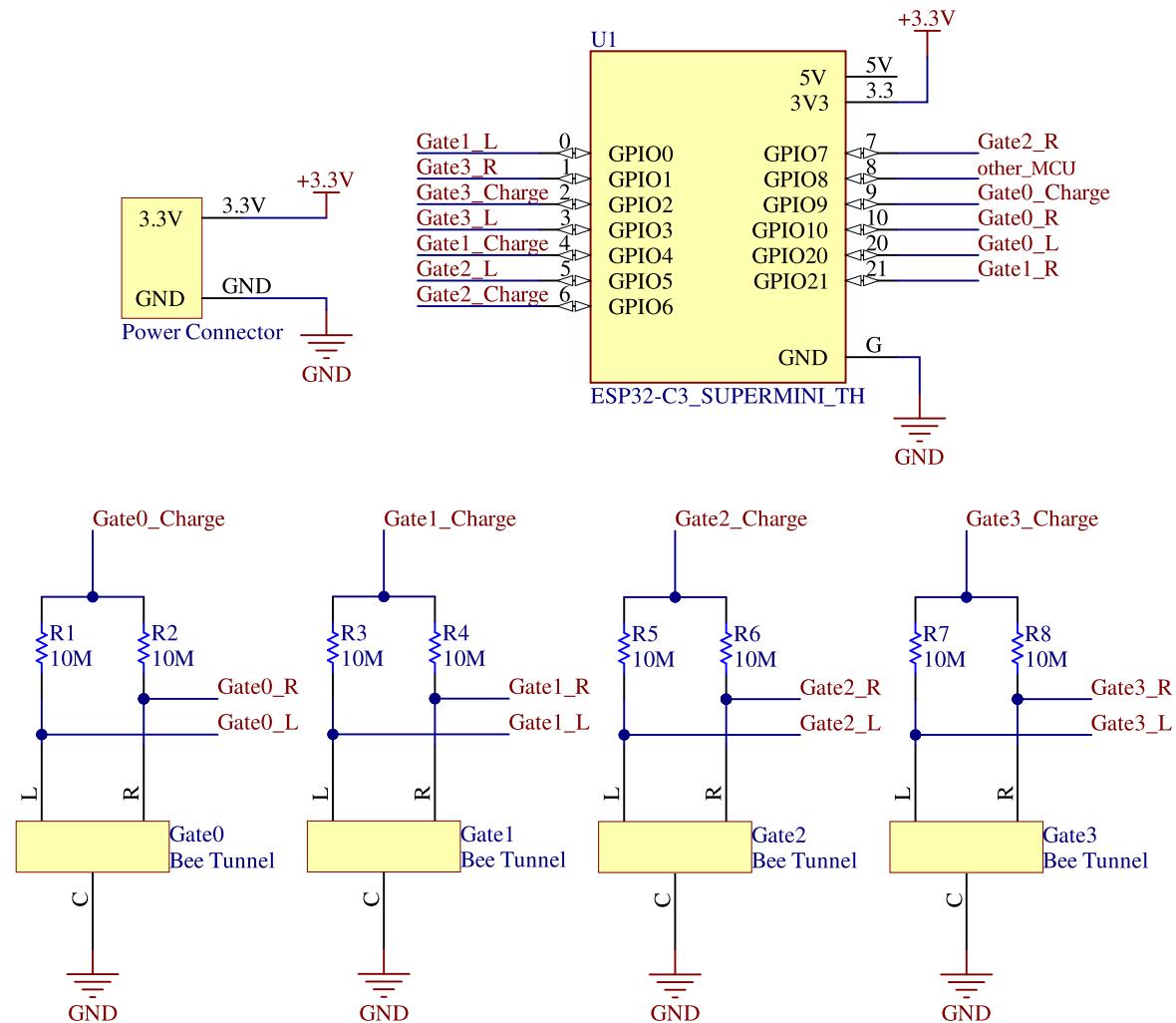
2.4.2. Realizacja płytki PCB

Układ elektroniczny zrealizowany został w formie płytki PCB, co zapewnia powtarzalność wytwarzania modułów urządzenia, ułatwia montaż układu, a także przy odpowiednim projektowaniu zmniejsza podatność systemu na pojemności pasożytnicze.

W pierwszym kroku realizacji PCB zaprojektowany został schemat układu elektronicznego zgodny z omówionym dotychczas projektem. W jego skład weszły cztery podukłady bramek wykrywających pszczoły, połączone z odpowiednimi wyprowadzeniami mikrokontrolera ESP32-C3. Odpowiednie przypisanie pinów okładkom bramek okazało się mieć kluczowe znaczenie. Prezentowany na rysunku schemat stanowi drugą wersję – w pierwszym podejściu w wyniku nieprawidłowego doboru pinów prawidłowo działały wyłącznie dwie bramki z czterech. Dogłębna analiza schematu płytki[33] ujawniła źródło problemów: niektóre piny GPIO mikrokontrolera mają w module ESP32-C3 SuperMini podłączone rezystory podciągające (o wartościach tysiąckrotnie niższych od rezystancji w torze ładowania – $10\text{ K}\Omega$). Elektrody L lub R podłączone do tych pinów były utrzymywane stale

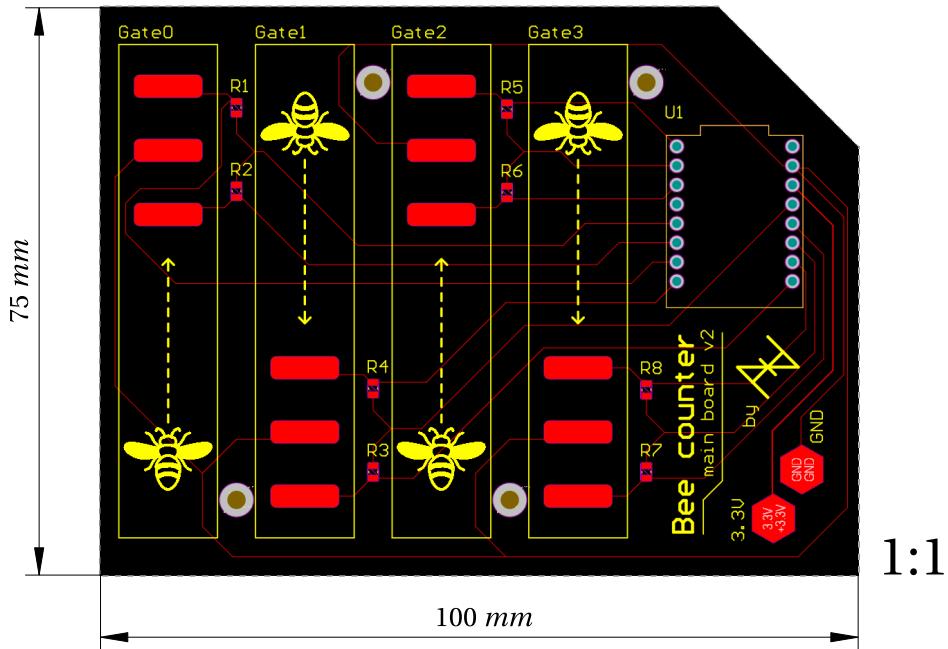
2. Budowa urządzenia

w stanie wysokim, a przez to mierzony czas ładowania wynosił zawsze zero. Konieczne było stworzenie zaktualizowanej wersji układu, w której piny z rezystorami podciagającymi wykorzystywane były wyłącznie do ładowania kondensatorów, nigdy do pomiaru. Piny z podciągnięciami to: GPIO2, GPIO8, GPIO9 [33]. Ostateczny sposób połączenia wyrowadzeń przedstawiony został w schemacie na rysunku 2.17. Pin GPIO8 wybrany został do realizacji połączenia pomiędzy mikrokontrolerem głównym i pomocniczym.

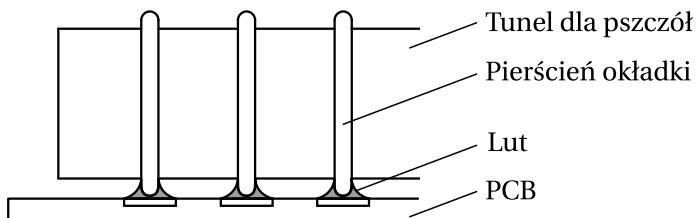


Rysunek 2.17. Schemat układu elektronicznego.

Zaprojektowany układ został przełożony na projekt płytki PCB, przedstawiony na rysunku 2.18. Podstawowym elementem PCB są footprinty tuneli dla pszczół. Tunele montuje się w wyznaczonym miejscu poprzez przylutowanie pierścieni do odsłoniętych miedzianych padów znajdujących się na płytce (rysunek 2.19). Rozmieszczenie footprintów tuneli dobrano tak, by zachować rozmieszczenie zdefiniowane w tabeli 2.2. W układach pomiarowych zastosowano rezystory w rozmiarze 0805, aby zapewnić względną łatwość lutowania przy zachowaniu małych wymiarów urządzenia. Po prawej stronie płytki znajduje się footprint modułu ESP32-C3 SuperMini, pozwalający wlutować bezpośrednio mikrokontroler, a także, alternatywnie, szynę goldpin pozwalającą na wygodne wpinanie i wypinanie modułu w razie potrzeby. Podczas projektu PCB najwięcej uwagi przyłożono planowaniu



Rysunek 2.18. Projekt płytki PCB.



Rysunek 2.19. Montaż tunelu dla pszczół do płytki PCB.

rozmieszczenia ścieżek. Mają one znaczenie, ponieważ ich parametry wpływają znacząco na pojemności pasożytnicze zakłócające działanie czujnika. Dla minimalizacji ich negatywnego wpływu zastosowano następujące reguły:

1. Unikano równoległych ścieżek, a w przypadku ich wystąpienia zapewniono jak największą odległość między nimi;
2. Nie dodano płaszczyzny masy na odwrocie płytki – jej zastosowanie jest w ogólności zalecane, jednak w przypadku niniejszego projektu mogłyby wystąpić stosunkowo duże pojemności między nią a ścieżkami;
3. Minimalizowano liczbę przełotek (*via*) na płytce – możliwe okazała się realizacja układu bez zastosowania ani jednej;
4. Zastosowano najmniejszą możliwą szerokość ścieżki, co oprócz zmniejszenia pojemności pasożytniczej wpłynęło również na zwiększenie dystansu między ścieżkami względem ich szerokości, poprawiając kryterium 1. oraz zmniejszając sprzężenie krzyżowe w układzie [34]. Została dobrana szerokość ścieżki równa 0.1 mm – minimum oferowane przez producenta płyt drukowanych JLCPCB [35].

Ostateczny projekt PCB przedstawiony został na rysunku 2.18. Oprócz wymienionych elementów zawiera on również styki zasilania oraz otwory montażowe. Wykonanie pły-

tek zostało zamówione u producenta JLCPCB². Do skonstruowania całego urządzenia konieczne było złożenie dwóch egzemplarzy – w pierwszej kolejności wlutowano rezystry ładowania bramek, a następnie przymocowane zostały tunele. W celu zapewnienia precyzji ich rozmieszczenia, w technologii druku 3D wykonany został specjalny wzornik utrzymujący płytki i tunele w odpowiedniej pozycji na czas lutowania. Na samym końcu przymocowane zostały moduły mikrokontrolerów. Stworzone układy (rysunek 2.20 były gotowe do programowania.



Rysunek 2.20. Gotowe płytki PCB.

2.5. Oprogramowanie mikrokontrolerów

Wyróżnione zostały cztery podstawowe zadania, które realizować musi program napisany na mikrokontrolery:

1. Akwizycja pomiaru z bramek — realizacja algorytmu umożliwiającego odczyt sygnału z kondensatorów w poszczególnych czujnikach, zgodnie z pseudokodem 1 opisany w sekcji 2.2.2;
2. Komunikacja między mikrokontrolerami — przesył danych z układu pomocniczego do układu głównego;
3. Komunikacja ze światem zewnętrznym — docelowo komunikacja bezprzewodowa, jednak na tym etapie pracy dowolna metoda komunikacji, która umożliwi potwierdzenie prawidłowego działania systemu;

² <https://jlpcb.com/>

4. Przetwarzanie pomiaru (liczenie pszczół) — konwersja surowego sygnału z bramek na informację o wchodzących i wychodzących pszczółach. Funkcjonalność ta nie jest realizowana na tym etapie pracy – stanowi ona obiekt rodzaju 3.

W kolejnych sekcjach zawarte zostały fragmenty kodu projektu. Wszystkie omawiane pliki znaleźć można w publicznym repozytorium git, dostępnym pod adresem <https://github.com/tototmek/bee-counter>.

2.5.1. Algorytm akwizycji

Przedstawione na następnych fragmentach kodu zawierają implementację algorytmu akwizycji sygnału bramki, zgodną z pseudokodem 1 w sekcji 2.2.2. W ramach dobrej praktyki programistycznej w pliku `include/gate.h` zdefiniowano również abstrakcyjne reprezentacje struktur danych występujących w systemie. Struktura `GateConfig` zawiera wszystkie dane reprezentujące pojedynczą bramkę – przypisania pinów GPIO, a także binarną flagę umożliwiającą odwrócić kierunek bramki. Zmiana kierunku zliczania części bramek na pewno będzie konieczna, ponieważ konstrukcja urządzenia zakłada bramki zorientowane naprzemiennie w obu kierunkach (rysunek 2.20). Struktura `GateReading` reprezentuje pojedynczy odczyt z czujnika (surowe sygnały obu kondensatorów w bramce), opisany dodatkowo numerem bramki, z której pochodzą dane. Czasy ładowania kondensatorów zapisywane są jako 32-bitowe dodatnie liczby całkowite, co daje wystarczająco duży zakres reprezentacji wyników na potrzeby tworzonego urządzenia. Największe znaczenie ma klasa `Gate`, która reprezentuje pojedynczą bramkę czujnika: opisywana jest przez konfigurację bramki i posiada metody `void initialize()` oraz `gate_reading_t measure()`. Służą one, odpowiednio, do inicjalizacji stanu początkowego bramki, oraz do akwizycji odczytu.

```
include/gate.h
1 #ifndef GATE_H
2 #define GATE_H
3
4 #include <cstdint>
5
6 namespace bee_counter {
7
8 constexpr uint8_t kNumGates = 4;
9
10 typedef struct GateConfig gate_config_t;
11 typedef struct GateReading gate_reading_t;
12
13 struct GateConfig {
14     uint8_t chargePin;
15     uint8_t measurePinL;
16     uint8_t measurePinR;
17     bool invertDirection;
18 };
19
20 struct GateReading {
```

2. Budowa urządzenia

```
21     uint8_t gateId;
22     int32_t timeRawL;
23     int32_t timeRawR;
24 };
25
26 class Gate {
27 private:
28     static constexpr uint32_t kChargeTimeout_ = 0xffff;
29     const gate_config_t config_;
30
31 public:
32     Gate(gate_config_t config);
33     void initialize();
34     gate_reading_t measure();
35 };
36
37 } // namespace bee_counter
38
39 #endif // GATE_H
```

Implementacja metod klasy `Gate` zawarta jest w poniższym pliku `src/gate.cpp`. Inicjalizacja bramki w funkcji `void Gate::initialize()` opiera się głównie na odpowiedniej konfiguracji pinów GPIO mikrokontrolera i zapewnieniu rozładowania kondensatorów bramki. Konfigurowany jest parametr `drive_capability` wszystkich pinów połączonych z bramką. Ustawiana jest jego maksymalna wartość `GPIO_DRIVE_CAP_3`, co zapewnia najkrótszy możliwy czas narastania sygnału na pinie wyjściowym mikrokontrolera po ustawieniu go w programie na stan wysoki [36]. Ma to znaczenie, ponieważ czas narastania napięcia na kondensatorze, który mierzymy jest bardzo krótki i należy unikać możliwych zakłóceń pomiaru, takich jak m.in. efekt slew-rate na wyjściu mikrokontrolera.

src/gate.cpp

```
1 #include "gate.h"
2
3 #include <Arduino.h>
4
5 namespace bee_counter {
6
7     Gate::Gate(gate_config_t config) : config_(config) {}
8
9     void Gate::initialize() {
10         pinMode(config_.chargePin, OUTPUT);
11         pinMode(config_.measurePinL, INPUT);
12         pinMode(config_.measurePinR, INPUT);
13         digitalWrite(config_.chargePin, LOW);
14         gpio_set_drive_capability(
15             (gpio_num_t)config_.measurePinL, GPIO_DRIVE_CAP_3);
16         gpio_set_drive_capability(
```

```

17     (gpio_num_t)config_.measurePinR, GPIO_DRIVE_CAP_3);
18     gpio_set_drive_capability(
19         (gpio_num_t)config_.chargePin, GPIO_DRIVE_CAP_3);
20 }
21
22 gate_reading_t Gate::measure() {
23     int32_t totalCount = 0;
24     int32_t counterL = 0;
25     int32_t counterR = 0;
26     bool lTriggered, rTriggered;
27     noInterrupts();
28     digitalWrite(config_.chargePin, HIGH);
29     do {
30         lTriggered = digitalRead(config_.measurePinL);
31         rTriggered = digitalRead(config_.measurePinR);
32         if (!lTriggered) {
33             ++counterL;
34         };
35         if (!rTriggered) {
36             ++counterR;
37         }
38         ++totalCount;
39         if (totalCount >= kChargeTimeout_) {
40             break;
41         }
42     } while (!lTriggered || !rTriggered);
43     digitalWrite(config_.chargePin, LOW);
44     interrupts();
45     gate_reading_t output = {0};
46     if (config_.invertDirection) {
47         output.timeRawL = counterR;
48         output.timeRawR = counterL;
49     } else {
50         output.timeRawL = counterL;
51         output.timeRawR = counterR;
52     }
53     return output;
54 }
55
56 } // namespace bee_counter

```

Funkcja `gate_reading_t Gate::measure()` realizuje omawiany wcześniej algorytm akwizycji z dwoma nieznaczonymi rozszerzeniami. Po pierwsze, pętla zliczania iteracji ładowania kondensatorów poprzedzona została wywołaniem funkcji `noInterrupts()`. Funkcja ta wyłącza obsługę przerwań mikrokontrolera, których pojawiienie się w trakcie akwizycji zakłócałoby sygnał poprzez zajmowanie czasu procesora. Wyłączenie przerwań umożliwia niezakłócone zliczanie iteracji algorytmu. Po zakończeniu akwizycji sygnału przerwania są ponownie włączane – ich obsługa jest konieczna do prawidłowej pracy.

2. Budowa urządzenia

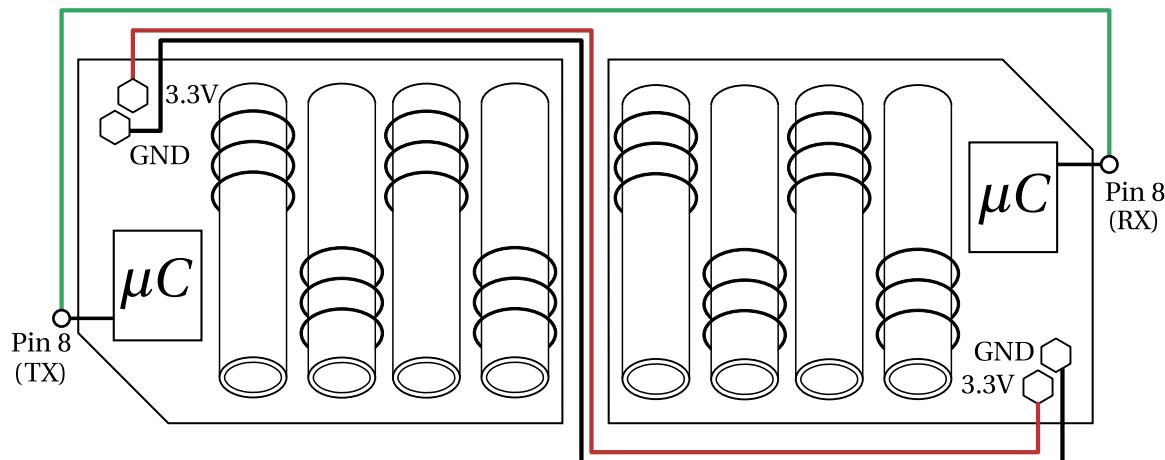
systemu, w którym zrównoległe zostanie wiele zadań. Uruchomienie przerwań realizuje wywołanie funkcji `interrupts()`.

Kolejnym rozszerzeniem względem najprostszej wersji algorytmu jest wprowadzony parametr `kChargeTimeout`, czyli suma zliczonych iteracji, przy której pętla jest przerywana pomimo braku wykrycia naładowania kondensatora. Podczas testów urządzenia zaobserwowano, że zdarzały się sytuacje (np. przy dotknięciu okładek), w których pojemność bramki i czas jej ładowania rosły na tyle, że licznik iteracji przepełniał się, a sama akwizycja trwała zbyt długo. Sytuacje takie oczywiście stanowią anomalię, pomimo to dla zapewnienia stabilności systemu wprowadzono mechanizm, który przerywa pomiar jeśli trwa on zbyt długo. Domyślnie skonfigurowane maksimum to `0xffff`, czyli zarazem maksymalna wartość licznika iteracji, po której nastąpiłoby przepełnienie. Implementacja tego zachowania wymagała dodanie dodatkowych poleceń do głównej pętli algorytmu, której czas wykonania ma krytyczne znaczenie, natomiast nie będzie ona miała zauważalnego negatywnego wpływu – czas wykonania porównania dwóch zmiennych typu `uint16_t` wynosi pojedyncze cykle zegara procesora.

Na koniec, wartości liczników iteracji wpisywane są do struktury wynikowej jako czasy ładowania poszczególnych kondensatorów w bramce. Struktura ta stanowi wyjście funkcji.

2.5.2. Komunikacja między mikrokontrolerami

Ze względu na kompromisy podjęte podczas doboru sprzętu, system składa się z dwóch analogicznych układów sterowanych przez osobne mikrokontrolery. Konieczne jest zapewnienie komunikacji pomiędzy nimi, w celu zebrania pełnych danych o stanie bramek. Problem ten stanowi wyzwanie, ponieważ każdy z mikrokontrolerów posiada tylko jeden wolny pin, możliwy do wykorzystania w tym celu. W ramach niniejszej pracy zaprojektowano protokół przesyłu danych z mikrokontrolera pomocniczego do mikrokontrolera głównego wykorzystujący tylko jeden przewód, oparty na protokole UART. Na rysunku 2.21 przedstawione zostało połączenie elektryczne pomiędzy układami.



Rysunek 2.21. Połączenia pomiędzy płytami PCB umożliwiające współpracę podukładów.

Jaką podstawę komunikacji wybrano protokół UART (ang. *Universal Asynchronous Receiver-Transmitter*). Jego przewagą nad innymi podobnymi protokołami (np. SPI, I²C)

jest asynchroniczność – czyli nie wymaga on używania sygnału zegarowego. Komunikujące się urządzenia znają zakładaną prędkość komunikacji (*baudrate*) i taktują kolejne bity we własnym zakresie [37]. Standardowo, UART wykorzystuje dwie linie danych, RX oraz TX, dla dwóch kierunków komunikacji. W przypadku projektowanego systemu konieczne (i możliwe) jest przesyłanie danych tylko w jednym kierunku – od układu pomocniczego do głównego. W ten sposób, wymagana jest tylko jedna linia asynchronicznej wymiany informacji: RX dla układu pomocniczego i TX dla układu głównego (rysunek 2.21).

Wykorzystywany mikrokontroler ESP32-C3 posiada wbudowany kontroler UART, pozwalający na prostą implementację podstawowej komunikacji przez programistę [26]. Układ ten realizuje kolejkowanie wysyłanych i odbieranych ramek, sprawdzanie bitów startu i bitów parzystości, umożliwiając wymianę pojedynczych bajtów informacji w sposób bezbłędny i łatwy w obsłudze. Pojedyncze bajty nie wystarczają jednak do zapisu pełnej informacji zwrotnej z czujników pojemnościowych: pełna informacja (struktura *GateReading*) składa się z 9 bajtów. Z tej przyczyny zdefiniowano 11-bajtową ramkę UART reprezentującą pojedynczy pomiar bramki, której struktura przedstawiona została na rysunku 2.22. Jej dwa pierwsze bajty stanowią ustalona sekwencja startowa, która umożliwia-

| Sekwencja startowa | Numer bramki | Pomiar okładki L | | | | Pomiar okładki R | | | | |
|--------------------|--------------|------------------|------|------|------|------------------|------|------|------|------|
| 0xfe | 0xfd | 0x01 | 0x00 | 0x00 | 0x08 | 0x59 | 0x00 | 0x00 | 0x1b | 0x39 |

Rysunek 2.22. Ramka UART reprezentująca pojedynczy pomiar bramki.

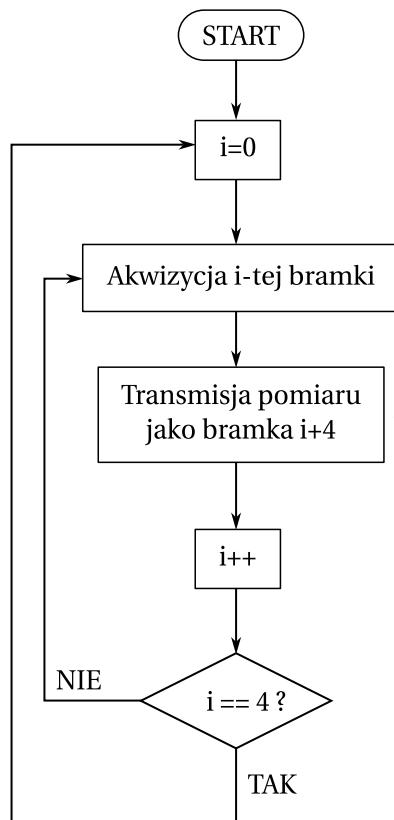
wia odbiorcy potoku danych zlokalizowanie początków poszczególnych ramek. Dziewięć ostatnich bajtów stanowią dane odczytu, zapisywane w sposób zgodny z reprezentacją struktury *GateReading* w pamięci mikrokontrolera, co pozwala uniknąć konwersji podczas nadawania i odbioru. Ze względu na konieczność implementacji komunikacji po wyłącznie jednej linii, zdecydowano się na dość nietypowy model wymiany informacji, przedstawiony na rysunku 2.23. Nie zastosowano najpopularniejszego w podobnych sytuacjach modelu zapytanie-odpowiedź, w którym układ główny wysyła do pomocniczego zapytanie o dane, a tamten odsyła przechowywane w pamięci wyniki. Takie rozwiązanie w prosty sposób zapewnia synchronizację przesyłanych ramek, natomiast wymaga dwukierunkowej komunikacji. Implementacja połączenia half-duplex pozwoliłaby na zastosowanie tego modelu, jednak w niniejszej pracy zastosowano alternatywne rozwiązanie – prostsze, dokładające jednak pewien narzut w postaci dodatkowych sekwencji startowych przesyłanych przez mikrokontroler pomocniczy. Ponieważ ilość przesyłanych danych nie jest szczególnie duża, aspekt ten nie ma negatywnego wpływu na działanie systemu.

Mikrokontroler w układzie pomocniczym pracuje w nieskończonej pętli, w której dokonuje akwizycji kolejnych bramek, a następnie przesyła ramki z wynikami. Dane przesyłane są sekwencyjnie, a wykrycie początków poszczególnych ramek jest możliwe dzięki występowaniu w nich sekwencji startowej 0xfefd. Mikrokontroler główny pracuje w analogicznej pętli, na przemian dokonując akwizycji jednej z bramek, oraz odbierając jedną ramkę od

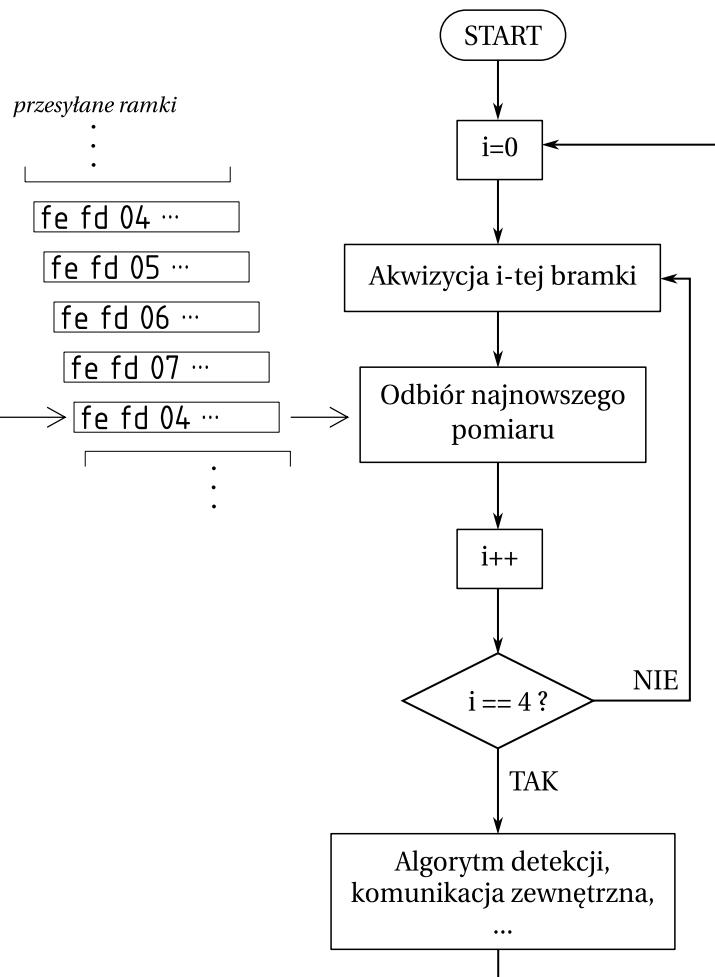
układu pomocniczego. Dane zapisywane są wyłącznie przez mikrokontroler główny, który przekazuje je później do algorytmu detekcji pszczół.

Ważne dla skutecznej wymiany informacji jest wykrywanie sekwencji startu ramki. Zostało to zrealizowane poprzez implementację automatu stanowego przedstawionego na rysunku 2.24. Przejścia między jego stanami odbywają się w wyniku odebrania przez kontroler UART kolejnego bajtu. Po skutecznym odbiorze sekwencji startowej następujące dalej bajty zapisywane są do tablicy wynikowej, i inkrementowany jest ich licznik. Zapełnienie tablicy wynikowej jest warunkiem zakończenia pracy automatu stanowego – odebrana została pełna ramka. Implementacja omawianego automatu zawarta jest funkcji `Receiver::receiveReading` z pliku `src/connection.cpp`, którego odpowiedni fragment zawarto w przedstawionym dalej fragmencie kodu.

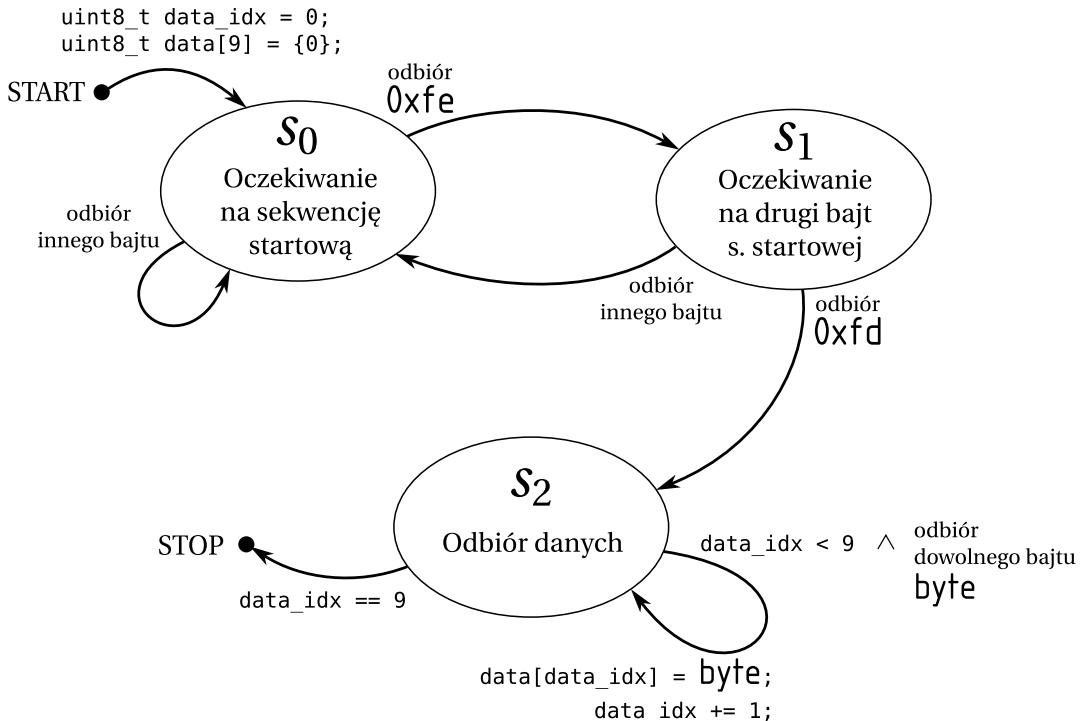
(a) mikrokontroler pomocniczy – algorytm



(b) mikrokontroler główny – algorytm



Rysunek 2.23. Algorytmy pracy mikrokontrolera głównego i pomocniczego.



Rysunek 2.24. Odbiór ramek uart – schemat automatu stanowego.

```

src/connection.cpp > Receiver::receiveReading
1  bool Receiver::receiveReading(gate_reading_t* reading, uint32_t timeoutMs) {
2      uint8_t bytes[sizeof(gate_reading_t)];
3      uint32_t timeoutTime = millis() + timeoutMs;
4      uint8_t state = 0;
5      int data_idx = 0;
6      int b~ = 0;
7      while (millis() < timeoutTime) {
8          if (!Serial1.available()) continue;
9          b~ = Serial1.read();
10         if (b~ == -1) continue; // No bytes received, continue spinning
11         switch (state) {
12             case 0:
13                 if (b~ == 0xfe) state = 1;
14                 break;
15             case 1:
16                 if (b~ == 0xfd) state = 2;
17                 else state = 0;
18                 break;
19             case 2:
20                 bytes[data_idx++] = b~;
21                 if (data_idx == sizeof(gate_reading_t)) {
22                     memcpy((uint8_t*)reading, bytes, sizeof(gate_reading_t));
23                     return true; // All data received
24                 }
25                 break;
26         }
27     }
28     return false;
29 }
```

Dodatkowo, funkcja pozwalająca na transmisję ramki została przedstawiona na poniższym fragmencie kodu.

```
src/connection.cpp > Transmitter::transmitReading
1 void Transmitter::transmitReading(const gate_reading_t* reading) {
2     Serial1.write(0xfe);
3     Serial1.write(0xfd);
4     Serial1.write((const uint8_t*)reading, sizeof(gate_reading_t));
5 }
```

Działanie utworzonego sposobu komunikacji przetestowano z wykorzystaniem przykładowych, arbitralnych danych przesyłanych między mikrokontrolerami. System działał prawidłowo i umożliwia przetwarzanie sygnałów ze wszystkich ośmiu tuneli na mikrokontrolerze głównym.

2.5.3. Komunikacja ze światem zewnętrznym

Docelowo, tworzone urządzenie ma komunikować się bezprzewodowo z serwerem aggregującym dane, jednak na bieżącym etapie pracy zostanie wykorzystana jedynie prostsza w implementacji komunikacja przewodowa. Wystarczy ona do zbadania jakości działania urządzenia, a także do eksperymentów pozwalających rozwijać algorytmy przetwarzające dane. Do komunikacji z komputerem wykorzystane zostało złącze USB typu C na urządzeniu oraz kontroler protokołu USB wprowadzony w ESP32-C3. Osiem próbek pobranych ze wszystkich tuneli urządzenia przesyłane jest łączem szeregowym po uprzednim zakodowaniu do postaci napisu z dziesiętną reprezentacją tych liczb, poprzedzanych przecinkami, wszystko w kodowaniu znaków ASCII. Kolejne rekordy oddzielane są od siebie znakiem nowej linii. Taka reprezentacja danych oczywiście nie jest optymalna, jednak wydajność transmisji nie ma wpływu na działanie urządzenia, wybrano więc to rozwiązanie jako najbardziej intuicyjne w późniejszej interpretacji danych. Znaki odbierane z urządzenia mogą być zapisywane bezpośrednio do pliku w formacie CSV z zapisem przebiegów.

Do zakodowania danych w odpowiedni napis wykorzystana została biblioteka `fmt` dla języka C++ [38], a w szczególności oferowana w jej ramach funkcja `fmt::format`. Wysłanie danych w protokole USB zrealizowane zostało za pomocą funkcji `Serial.print`, stanowiącej element frameworka Arduino. Pełen kod pętli programu został przedstawiony na fragmencie kodu poniżej (plik `src/main.cpp`, funkcja `primaryMcuLoop`).

Pomiędzy akwizycją danych, a ich przesyłem do komputera użytkownika, wykonywany jest dodatkowy krok obliczenia sygnałów różnicowych oraz odpowiedniego przyporządkowania numeracji bramek. Przemapowanie kolejności zapewnia, że kolejność sygnałów w zwracanych przez urządzenie danych odpowiada fizycznej kolejności tuneli w obudowie urządzenia.

```

src/main.cpp > primaryMcuLoop
1 void primaryMcuLoop() { // Kod wykonywany w~pętli na mikrokontrolerze głównym
2     // Pomiar bramek 0-3 i~odbiór bramek 4-7
3     for (int i~= 0; i~< bee_counter::kNumGates; ++i) {
4         gate_reading[i] = gate[i].measure();
5         receiver.receiveReading(&single_reading, 5);
6         gate_reading[single_reading.gateId] = single_reading;
7     }
8     // Obliczenie sygnałów różnicowych i~dostosowanie numeracji tuneli
9     delta[0] = gate_reading[7].timeRawR - gate_reading[7].timeRawL;
10    delta[1] = gate_reading[6].timeRawR - gate_reading[6].timeRawL;
11    delta[2] = gate_reading[5].timeRawR - gate_reading[5].timeRawL;
12    delta[3] = gate_reading[4].timeRawR - gate_reading[4].timeRawL;
13    delta[4] = gate_reading[0].timeRawL - gate_reading[0].timeRawR;
14    delta[7] = gate_reading[3].timeRawL - gate_reading[3].timeRawR;
15    delta[6] = gate_reading[2].timeRawL - gate_reading[2].timeRawR;
16    delta[5] = gate_reading[1].timeRawL - gate_reading[1].timeRawR;
17    // Przesył danych do PC
18    std::string message =
19        fmt::format("{}{},{}{},{}{},{}{},{}{}\n", millis(),
20                    delta[0], delta[1], delta[2], delta[3],
21                    delta[4], delta[5], delta[6], delta[7]);
22    Serial.print(message.c_str());
23}

```

```

src/main.cpp > secondaryMcuLoop
1 void secondaryMcuLoop() { // Kod wykonywany w~pętli na kontrolerze pomocniczym
2     // Pomiar bramek 0-3 i~przesłanie do układu głównego jako bramki 4-7
3     for (int i~= 0; i~< bee_counter::kNumGates; ++i) {
4         single_reading = gate[i].measure();
5         single_reading.gateId = i+bee_counter::kNumGates;
6         transmitter.transmitReading(&single_reading);
7     }
8 }

```

Powyżej przytoczona została również treść funkcji secondaryMcuLoop, która wykonywana jest w pętli na mikrokontrolerze pomocniczym. Realizuje ona wyłącznie pobranie pomiarów i ich transmisję do mikrokontrolera głównego. Wybór, który z dwóch mikrokontrolerów urządzenia wykonuje zadania pętli głównej odbywa się poprzez ustawienie pierwszego bajtu pamięci EEPROM. Wartość 0x00 oznacza mikrokontroler główny, natomiast 0x01 mikrokontroler pomocniczy. Dzięki takiemu podejściu istnieje możliwość wgrywania tego samego programu na oba mikrokontrolery, a zaprogramowany tryb pozostaje zapamiętany również po aktualizacji programu. Wystarczy jedynie sprawdzić w programie wartość pierwszego pola pamięci i wykonywać odpowiednią funkcję z przedstawionych powyżej. Z implementacją tego mechanizmu można się zapoznać w kodzie źródłowym projektu – w plikach include/connection.h, src/connection.cpp oraz src/main.cpp.

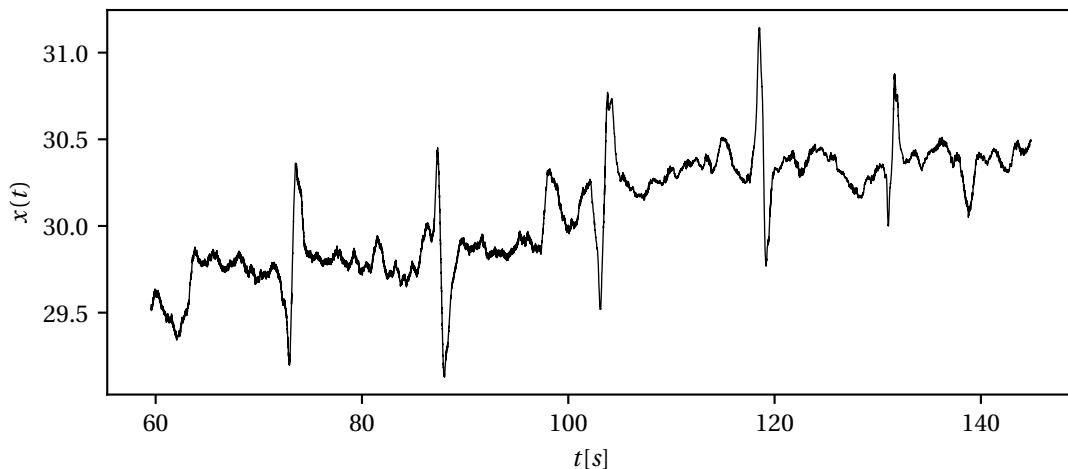
2.6. Testy

Przeprowadzony został szereg testów mających na celu weryfikację sprawnego działania całego urządzenia. W pierwszej kolejności zweryfikowano, czy sygnał różnicowy generowany przez bramkę czujnika zawiera impulsy wywołane przejściem pszczoły przez tunel zbliżone do teoretycznych idealnych przebiegów z rysunku 2.6. Zasada eksperymentu była zbliżona do działań opisanych w sekcji 2.2.4, z następującymi modyfikacjami:

1. W celu zasymulowania przejścia owada przez czujnik wykorzystana została prawdziwa pszczoła – martwy osobnik zebrany z dna ula³. Po jego pobraniu zadano o niezwłoczne przeprowadzenie eksperymentu w celu uniknięcia wysuszenia organizmu. Wykorzystanie prawdziwej pszczoły zapewnia bliższy żywemu osobnikowi model właściwości elektrycznych;
2. Zbierane były dane różnicowe z bramki jednego z czujników.

Test został wykonany na tunelu o numerze 0. Pszczołę przeprowadzono przez bramkę kilkukrotnie w obu kierunkach. Wynikowy przebieg sygnału wyjściowego został przedstawiony na rysunku 2.25. Dane zostały przefiltrowane filtrem średniej kroczącej na oknie 30 próbek. Na prezentowanym zakresie przebiegu wyraźnie widoczne jest pięć podwójnych

Rysunek 2.25. Przebieg sygnału wyjściowego bramki podczas przeprowadzonego testu.



asymetrycznych impulsów odpowiadających kolejnym przejściom obiektu przez czujnik. Ich polaryzacja zmienia się naprzemiennie – odpowiada to kierunkowi ruchu pszczoły. Należy zwrócić uwagę na cechy sygnału, którymi odbiega on od przebiegów idealnych. Po pierwsze, widoczne są drobne wahania poziomu sygnału o czasie trwania zbliżonym do analizowanych impulsów. Dodatkowo, widoczny jest wyraźny trend w sygnale – poziom zerowy sygnału różnicowego mocno przesuwa się w czasie. Konieczne okaże się opracowanie metody przetwarzania sygnału, która pozwoli zminimalizować wpływ tych efektów na działanie algorytmów detekcji pszczoły. Pomimo tych wad, pojawiające się impulsy są

³ Pojawianie się martwych osobników na dennicy ula jest naturalnym elementem życia kolonii. Pszczoły umierają wewnętrz gniazda i osypują się na jego dno, skąd zazwyczaj od razu wynoszone są na zewnątrz przez robotnice – jednak zimą lub w okresie złej pogody pozostają tam dłużej.

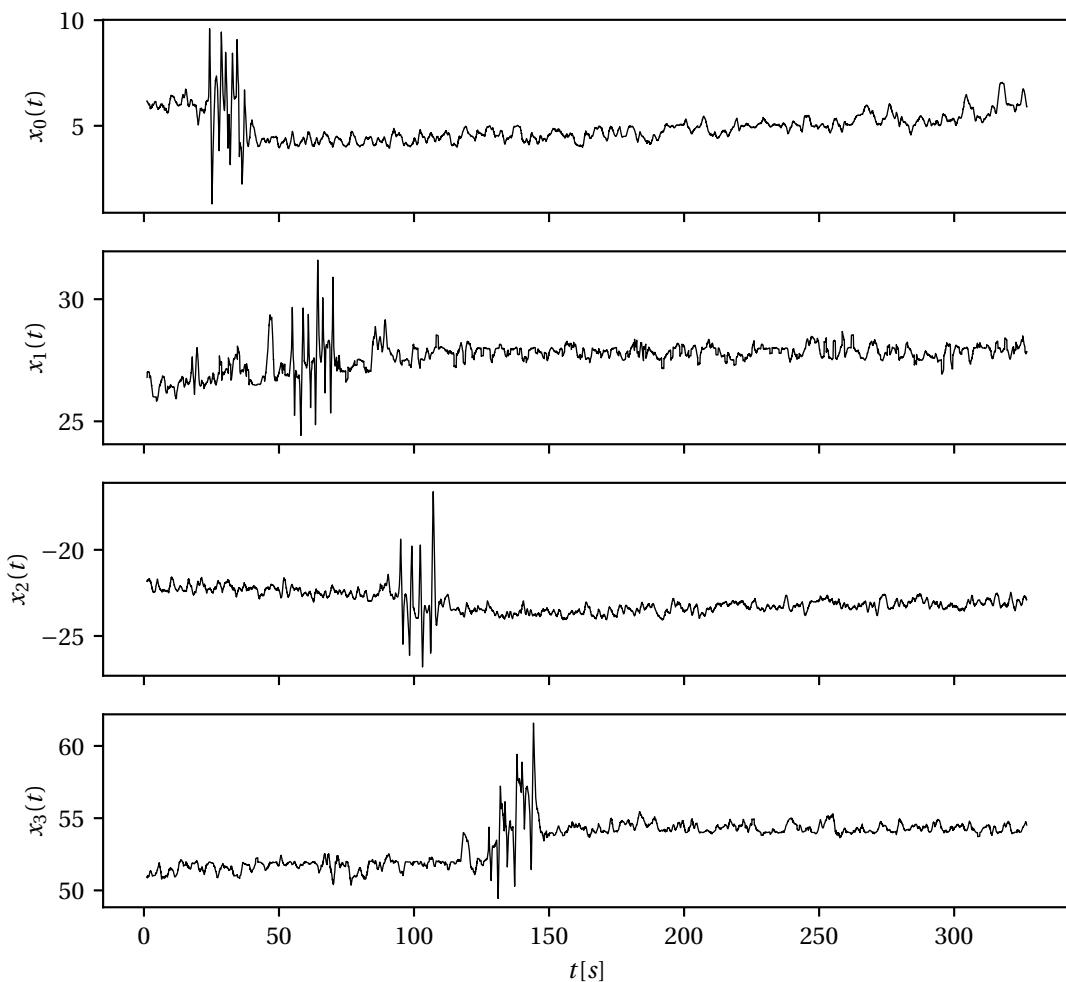
bardzo wyraźne, ich wykrycie będzie zatem możliwe z pomocą dość prostych algorytmów mogących działać na mikrokontrolerze o ograniczonych zasobach.

Przeprowadzono kolejny eksperyment weryfikujący działanie całości urządzenia. Badanie analogiczne jak poprzednio przeprowadzono dla wszystkich ośmiu tuneli, przeprowadzając martwą pszczołę czterokrotnie kolejno przez każdy z nich. Sygnał wyjściowy bramek zbierano przez cały czas trwania eksperymentu. Przeprowadzone badanie miało udowodnić, że stworzony system czujników nadaje się do zastosowania jeżeli:

- (a) Na przebiegu sygnału każdego z czujników pojawią się impulsy w momentach odpowiadających pojawianiu się pszczoły w odpowiedniej bramce,
- (b) Impulsy widoczne na każdym z sygnałów będą miały kształt wystarczająco podobny, by w ramach dalszego rozwoju umożliwić wykrywanie ich jednym uniwersalnym algorytmem.

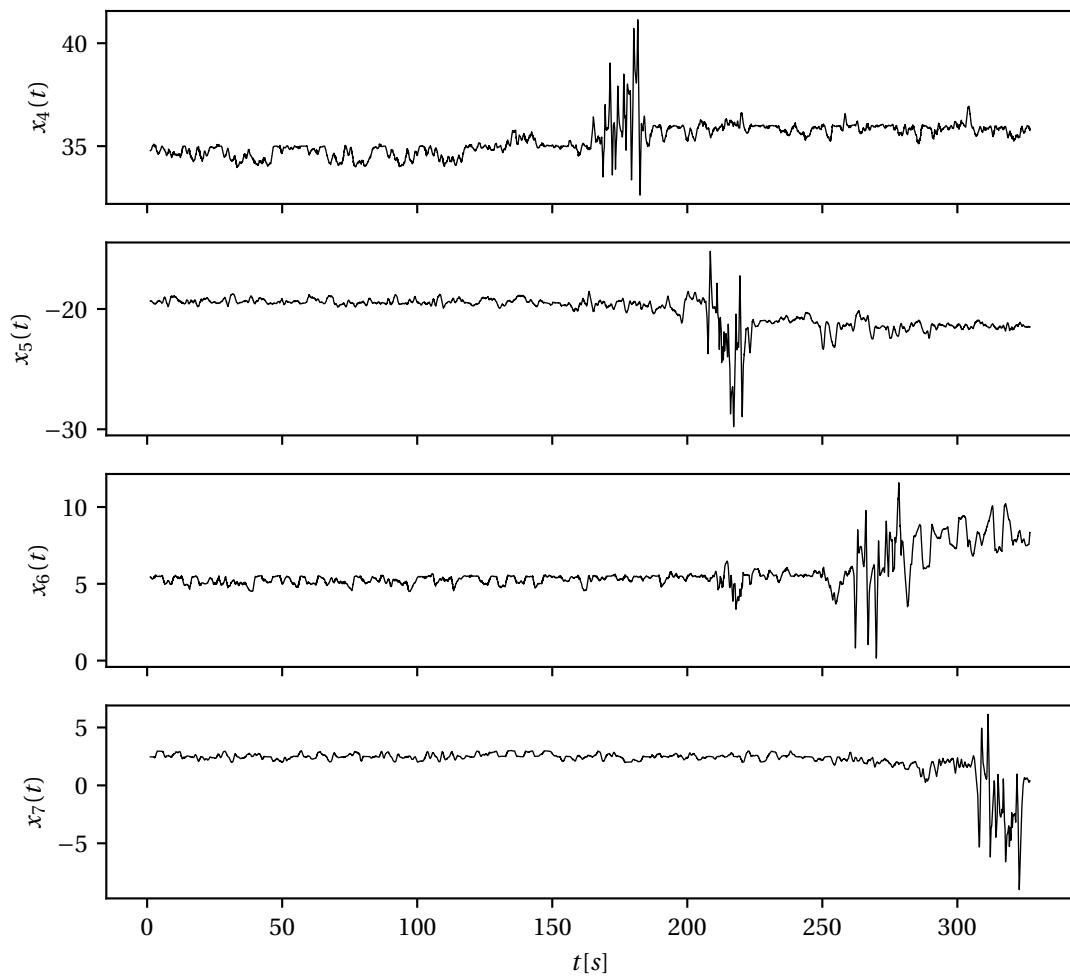
Dane zebrane podczas niniejszego eksperymentu zostały przedstawione na rysunkach 2.26 oraz 2.27. Na ich podstawie można jednoznacznie stwierdzić, że każda z bramek

Rysunek 2.26. Przebiegi sygnałów wyjściowych bramek x_0-x_3 podczas testu działania urządzenia.



reaguje prawidłowo na pojawienie się pszczoły w tunelu. Na sygnałach wyjściowych kolejno pojawiały się serie szpilek jasno odpowiadających przejściom obiektu przez tunel.

Rysunek 2.27. Przebiegi sygnałów wyjściowych bramek x_4-x_7 podczas testu działania urządzenia.



Impulsy zebrane przez urządzenie nie były idealne: wyraźnie widoczne są wspomniane wcześniej efekty spontanicznych wychyleń sygnału oraz pojawianie się trendu – ponownie jednakże stwierdzono, że jakość sygnału jest wystarczająca do wykorzystania w dalszym przetwarzaniu sygnału.

Przeprowadzone eksperymenty pozwoliły stwierdzić, że skonstruowane urządzenie działa prawidłowo. Możliwe było rozpoczęcia dalszych prac, aby umożliwić przetestowanie urządzenia na żywych pszczołach w pasiece.

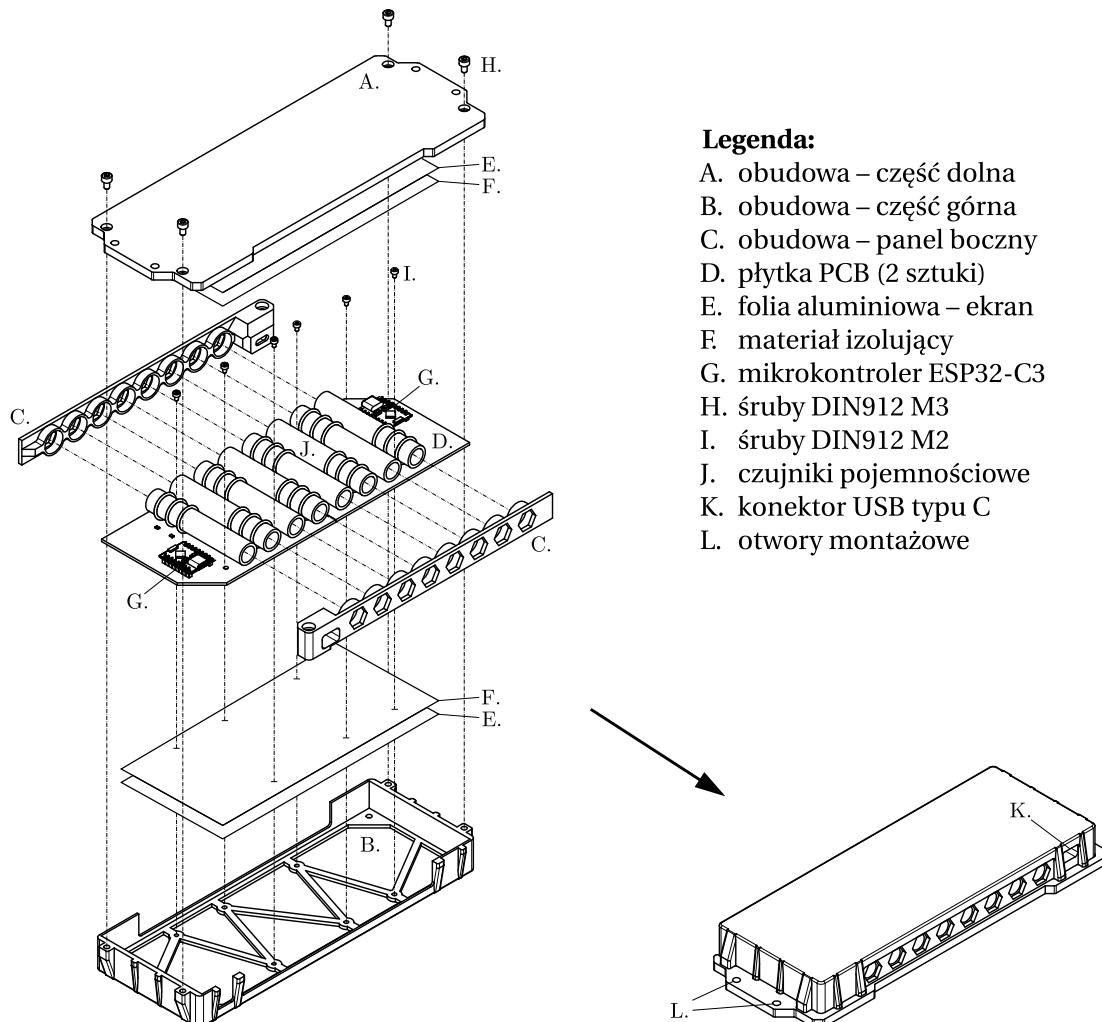
2.7. Montaż urządzenia

Aby możliwe było wykorzystanie stworzonego systemu w warunkach rzeczywistych konieczne jest umieszczenie urządzenia w obudowie, która odizoluje je od zmiennych warunków otoczenia, oraz ograniczy ruch pszczół do wnętrza tuneli czujników. Przyjęto następujące dodatkowe założenia dla jej projektu:

1. Technologia wykonania: druk 3D, materiał PLA;
2. Jak największa kompaktowość, w celu ułatwienia montażu na ulu dowolnego rodzaju;

3. Zapewnienie izolacji mechanicznej (uniemożliwienie dostania się owadów do środka, a także zanieczyszczenia wnętrza pyłkiem), oraz elektrostatycznej.

Do stworzenia projektu obudowy wykorzystano oprogramowanie *Onshape*⁴ służące do parametrycznego modelowania 3D. Zaprojektowane części, pozostałe elementy, a także sposób ich złożenia zostały przedstawione na rysunku 2.28. Drukowana obudowa składa



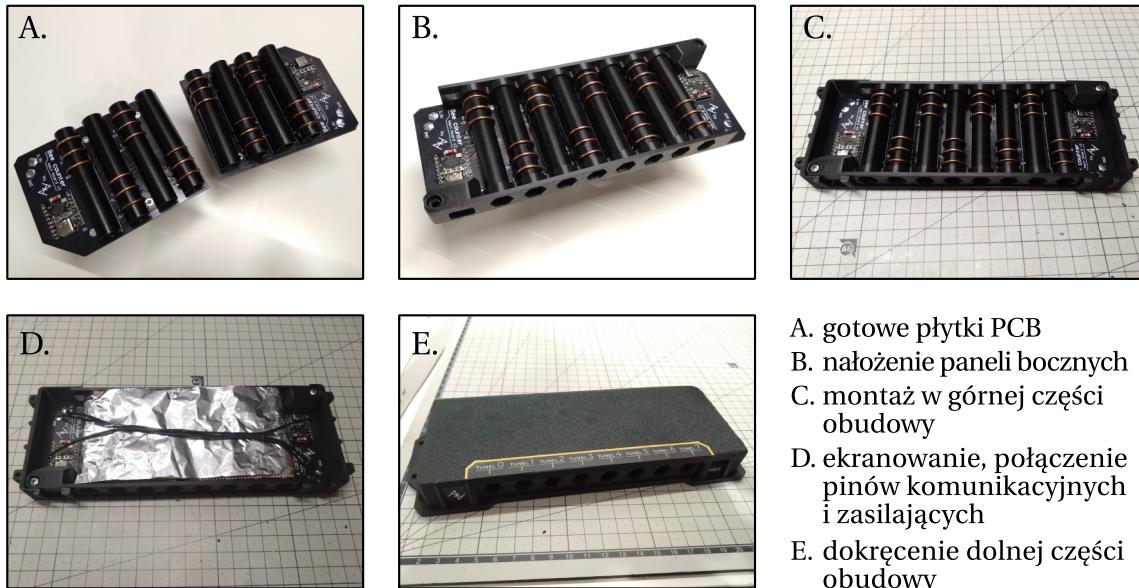
Rysunek 2.28. Schemat złożenia obudowy urządzenia

się z czterech fragmentów, oznaczonych A., B. i dwukrotnie C. Panele boczne (C.) posiadają kilkumilimetrowe kołnierze nachodzące na końcach tuneli czujnika, zapewniając szczelne połączenie. Wchodzą one w dopasowane wpusty w dolnej i górnej części obudowy (A., B.), również zapobiegając występowaniu szczelin. Płytki PCB czujnika przykręcone są przez otwory montażowe sześcioma śrubami M2, po uprzednim ekranowaniu warstwą materiałów: izolującego (arkusz papieru do pieczenia), oraz ekranującego (folia aluminiowa). Piny komunikacyjne i zasilające obu podkładów łączone są ze sobą przewodami, a w ostatnim kroku złożenia obie połowy obudowy skręcane są ze sobą śrubami M3. Dolna część obudowy wyposażona jest dodatkowo w otwory montażowe, z których użytkownik urządzenia może skorzystać, aby przymocować całość do półki przy wylotku ula. Na ry-

⁴ <https://cad.onshape.com>

2. Budowa urządzenia

sunku 2.29 przedstawiona została dokumentacja fotograficzna kolejnych kroków montażu płytka PCB w obudowie.



Rysunek 2.29. Zdjęcia z kolejnych etapów montażu urządzenia w obudowie

Budowa urządzenia została niniejszym zakończona – możliwe było rozpoczęcie pracy nad tworzeniem algorytmów detekcji pszczół.

3. Algorytm detekcji

Opisany w poprzedniej części pracy system czujników generuje sygnały wyjściowe, na przebiegach czasowych których widoczne są impulsy wywołane ruchem pszczół. W celu umożliwienia wykorzystania stworzonego urządzenia do automatycznego zliczania pszczół w ulu, konieczne jest zaprojektowanie algorytmu wykrywającego te impulsy. Przyjęte zostały następujące założenia:

1. Algorytm ma być uruchomiony na mikrokontrolerze wchodząącym w skład urządzenia, co pozwoli uniknąć konieczności rozszerzenia systemu o dodatkowy węzeł, którego zadaniem byłoby jedynie wykonywanie obliczeń. Wymagać to będzie niskiej złożoności pamięciowej algorytmu oraz niewielkiego narzutu obliczeniowego.
2. Algorytm ma być wykonywany *on line* – w każdej iteracji działania urządzenia, lub na buforze z maks. kilku sekund. W ten sposób zagwarantowane będą bieżące dane, oraz brak przerw w akwizycji sygnału z czujników pojemnościowych;
3. Ten sam algorytm zostanie uruchomiony niezależnie dla wszystkich tuneli urządzenia, z opcją dostosowania jego parametrów konfiguracyjnych do zmiennych właściwości poszczególnych bramek.

3.1. Zbiór danych przykładowych

Pierwszym krokiem, zanim rozpoczęto pracę nad projektem algorytmu, było zebranie dużej ilości przykładowych danych z pracy urządzenia w warunkach rzeczywistych. Są one niezwykle ważne, posłużą bowiem do oceny proponowanych w kolejnych sekcjach algorytmów i ich konfiguracji – posłużą do przeprowadzenia testów *offline*.

3.1.1. Testy urządzenia w pasiece

W celu uzyskania reprezentatywnych danych z pracy urządzenia w środowisku rzeczywistym przeprowadzone zostały eksperymenty w dwóch niezależnych pasiekach, w różnych tygodniach, przy innych warunkach atmosferycznych i aktywności pszczół. Zbierano sygnały $x_i(t)$, dla $i = 0 \dots 7$, czyli wyjściowe przebiegi czasowe wszystkich bramek urządzenia. Testy prowadzono według następującej metody:

1. Wybierano ul, który zostanie wykorzystany do eksperimentu. W obu przypadkach kierowano się największą aktywnością pszczół w całej pasiece, by w zebranych przebiegach wystąpiło jak najwięcej przejść owadów przez czujnik;
2. Na wylotku wybranego ula umieszczano urządzenie. Zastosowano dwie alternatywne jego pozycje: wewnątrz ula – zamocowane do dennicy, a także na zewnątrz ula na półce przed wylotkiem. Pszczelarze w prowadzonych rozmowach sugerowali, że przy drugiej z wymienionych metod pszczoly mogą być niechętne do wchodzenia do gniazda, jednak w eksperymencie nie zaobserwowano tego zjawiska. Wszelkie szczeleliny pozostające między urządzeniem a ścianami ula zatkano, aby uniemożliwić pszczołom ominięcie tuneli czujnika.
3. W obu eksperimentach odczekano około 30 minut, aby pszczoły poruszone zamie-

3. Algorytm detekcji

szaniem przy montażu urządzenia uspokoili się, a następnie aby przyzwyczaiły się do nowego obiektu na wylotku.

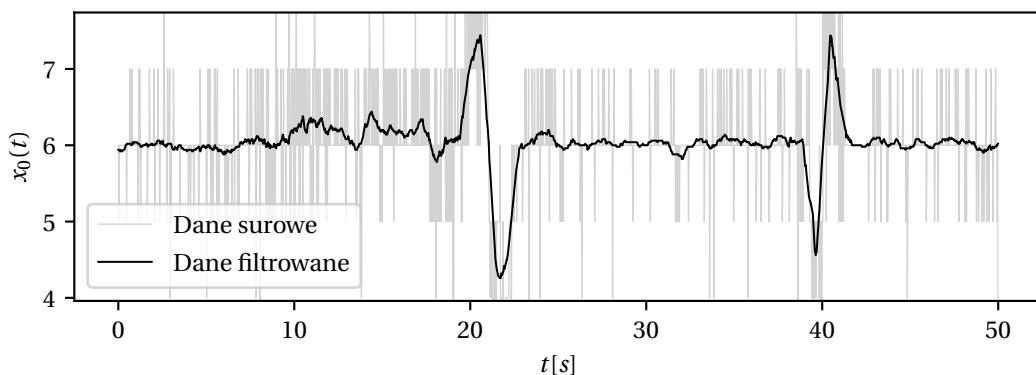
4. Na statywie przed ulem umieszczono kamerę wideo, nagrywającą obraz urządzenia. Kadr dobrany został tak, by wyraźnie widoczne były wloty wszystkich tuneli urządzenia. Nagranie to będzie później wykorzystane do anotacji danych – ręcznego opisu chwil, w których pszczoły wchodziły i wychodziły.
5. Urządzenie łączono z portem USB laptopa, aby zapewnić przesył kolejnych próbek sygnałów wyjściowych każdego z czujników pojemnościowych. Uruchamiano stworzony wcześniej skrypt Python (plik `scripts/collect_data.py` w repozytorium projektu), zapisujący otrzymywane dane do pliku CSV.
6. W chwili uruchomienia w komputerze akwizycji sygnału, wykonywano przed kamerą klaśniecie, co pozwoli na późniejszą synchronizację nagrania z zebranymi danymi.

Na rysunku 3.2 przedstawiona jest fotografia prezentująca jeden z przeprowadzonych eksperymentów, natomiast na rysunku 3.3 przedstawiono alternatywne miejsce montażu urządzenia (od środka ula). W tabeli 3.1 podsumowano informacje o eksperymentach.

Tabela 3.1. Zbiór danych przykładowych – podsumowanie eksperymentów

| | Eksperyment 1 | Eksperyment 2 | Łącznie |
|----------------------|-----------------|-----------------|----------------|
| Data | 20 lipca 2025 | 26 lipca 2025 | |
| Lokalizacja | gmina Miastkowo | gmina Choroszcz | |
| Czas trwania | ok. 45 min | ok. 30 min | ok. 1 h 15 min |
| Liczba próbek | 274214 | 182435 | 456649 |

Przeprowadzone eksperymenty stanowiły również pierwszą weryfikację działania urządzenia w warunkach rzeczywistych – przy współpracy z żywymi pszczołami. Fragment zebranych danych zwizualizowano na wykresie (rysunek 3.1), w celu sprawdzenia czy zawiera prawidłowo wyglądające impulsy wywoływane ruchem pszczół. Stwierdzono, że system działa prawidłowo: pomimo występującego szumu widoczne są wzory bardzo podobne do wzorcowych przebiegów; warto porównać z rysunkiem 2.6.



Rysunek 3.1. Przebieg sygnału $x_0(t)$ (wyjścia pierwszego czujnika pojemnościowego) w pierwszych kilkudziesięciu sekundach trwania eksperymentu. Przebieg filtrowany został wygenerowany po-przez nałożenie na dane oryginalne filtra średniej kroczącej o długości okna równej 50.



Rysunek 3.2. Zdjęcie jednego z eksperymentów. Widoczne jest urządzenie zamontowane na wyłotku ula, oraz kamera nagrywająca jego wylot. Znajdujący się na obrazie fioletowy przewód łączy czujnik z komputerem zapisującym dane.



Rysunek 3.3. Zdjęcie prezentujące montaż urządzenia podczas drugiego eksperymentu. Jest ono przybite do dennicy – pozostałą część ula nastawia się na widoczny na obrazie element.

3.1.2. Ręczna anotacja danych

Umożliwienie oceny algorytmów z wykorzystaniem zbudowanego zbioru danych wymaga przeprowadzenia jego anotacji. Polega ona na niezależnym, ręcznym oznaczeniu chwil przechodzenia pszczół przez tunele czujnika. Podczas testowania algorytmów weryfikowana ma być zgodność generowanych przez nie wyników z tymi właśnie oznaczeniami. Do anotacji wykorzystane zostały wspomniane wcześniej nagrania wideo z eksperymentów. W celu przyspieszenia procesu ręcznego przeglądu materiału, a także zredukowania prawdopodobieństwa popełniania błędów, stworzono dedykowany program: wyświetlał on użytkownikowi nagranie, dając możliwość kontroli tempa odtwarzania, a także zautomatyzowanego tworzenia anotacji danych poprzez pojedyncze uderzenia klawiszy (plik scripts/manual_annotation.py). Wciśnięcie wybranego przycisku zapisywało dla danej chwili nagrania wystąpienie wejścia/wyjścia pszczóły przez odpowiedni tunel. Na rysunku 3.4 przedstawiony został zrzut ekranu z programu – prezentuje on także przykładowy kadr z nagrania eksperymentu. Ręczne etykietowanie całego materiału nagraniowego

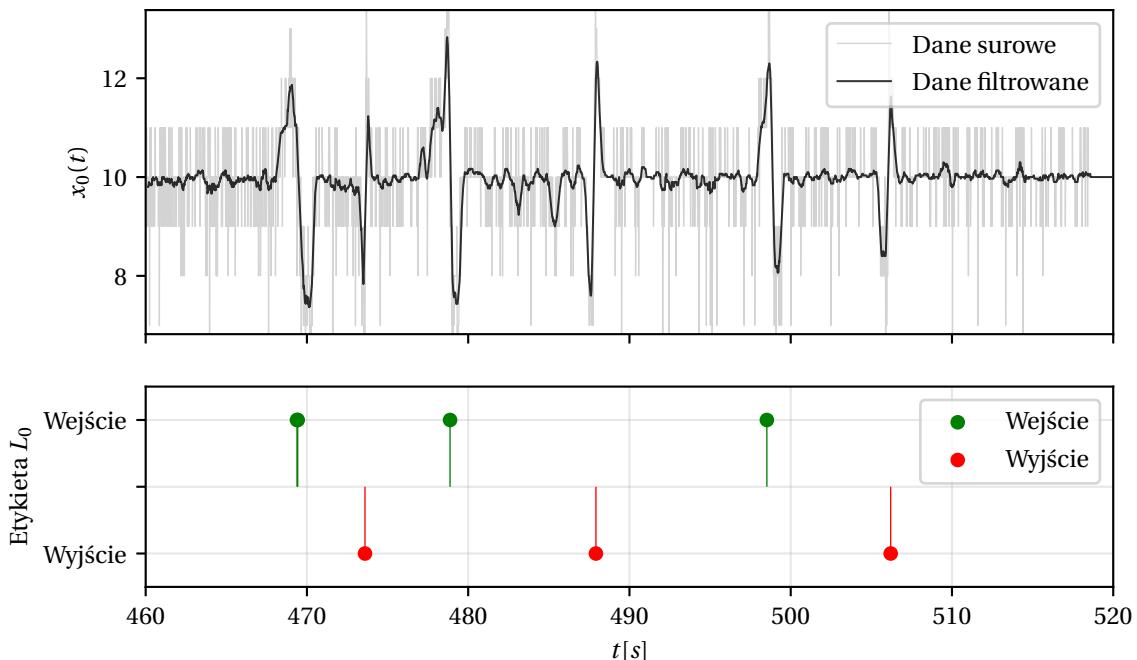


Rysunek 3.4. Zrzut ekranu z programu do ręcznej anotacji danych z eksperymentu.

okazało się wymagać większego nakładu pracy niż początkowo oczekiwano – ze względu na ograniczenia percepcji operatora programu konieczne było przetwarzanie całej długości nagrania z osobna dla każdego z tuneli, co ośmiokrotnie wydłużyło oczekiwany czas pracy.

Wynikiem niniejszego etapu pracy było powstanie dla każdego z tuneli urządzenia dwóch zbiorów anotacji: L_i^{we} – zawierającego chwile t , w których pszczoły wchodziły do ula przez tunel i ; oraz L_i^{wy} – zawierającego chwile t , w których pszczoły wychodziły z ula przez tunel i . Etykiety w takiej formie porównano z oryginalnymi sygnałami wyjściowymi $x_i(t)$, aby empirycznie potwierdzić, że impulsy generowane przez pszczoły pokrywają się z anotacjami. Na rysunku 3.5 przedstawiony jest fragment danych, na którym wyraźnie widać, że anotacje są prawidłowe, oraz że nie wystąpiły problemy z synchronizacją danych z czujnika i nagrania. Każdemu z wyraźnie widocznych impulsów odpowiada element

zbioru etykiet; ponadto, rodzaj anotacji (wejście/wyjście) jest zgodny z polaryzacją impulsów.

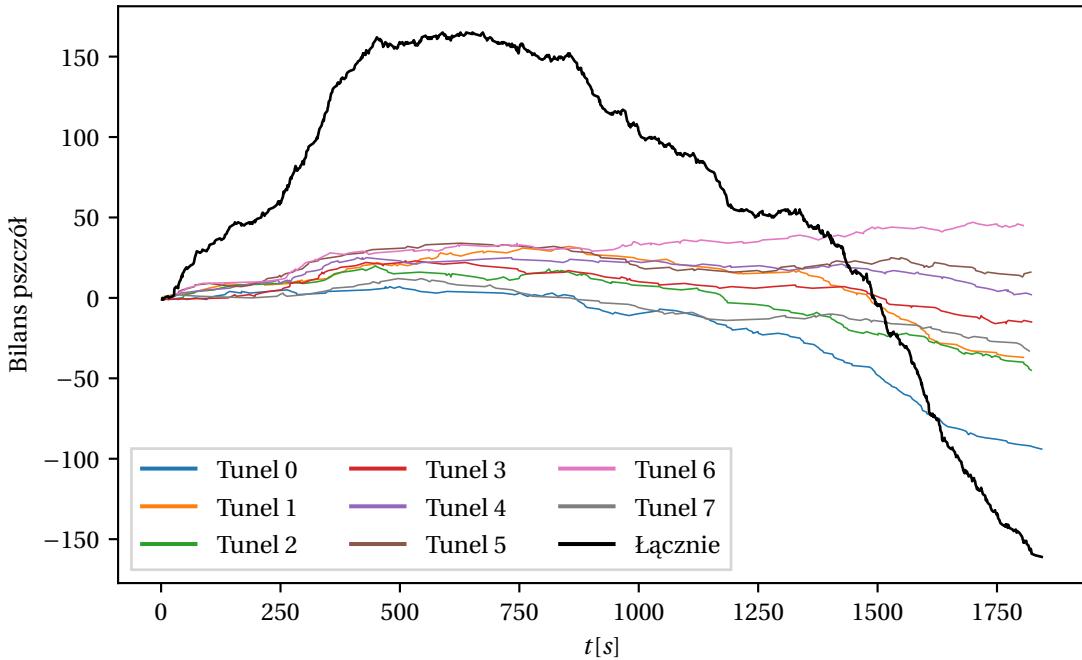


Rysunek 3.5. Fragment przebiegu sygnału x_0 zestawiony z ręcznie dodaną anotacją.

Na rysunku 3.6 przedstawione zostały przebiegi czasowe bilansu pszczół w jednym z przeprowadzonych eksperymentów. Wyznaczono je osobno dla poszczególnych tuneli, oraz łącznie dla całego urządzenia, poprzez sumowanie łącznej liczby osobników, które do danej chwili t weszły do ula, minus te, które go opuściły daną ścieżką. Można zaobserwować, że na początku eksperymentu owady prawie wyłącznie wchodziły do ula. Jest to rezultat nasilonego powrotu osobników, które opuściły gniazdo podczas montażu czujnika w grupowym instynkcie obronnym. Po kilkunastu minutach tendencja ta została zmieniona, a pszczoły zaczęły intensywnie opuszczać ul – prawdopodobnie wracając do pracy po jej zaburzeniu. Warto również zauważyć, że owady miały swój ulubiony tunel, którym wchodziły do ula (tunel 0), oraz ulubiony tunel wyjściowy (tunel 6).

Należy zwrócić uwagę na możliwe problemy wynikające z przedstawionej metody prowadzenia anotacji danych. W przeprowadzonym procesie, oznaczano na podstawie nagrania video chwile, w których pszczoła pojawiała się na wlocie tunelu. W ogólności, jest to dość dobry przybliżenie chwili jej przejścia przez czujnik pojemnościowy, bez względu na kilka centymetrów dzielące koniec tunelu od położenia czujnika – pszczoły zazwyczaj poruszają się ze znaczną szybkością. Niestety, mogą wystąpić sytuacje szczególne czterech rodzajów:

1. Pszczoła spoza ula wlatuje do tunelu, zatrzymuje się przed dotarciem do czujnika, czeka dłuższy czas, i dopiero przelatuje przez bramkę czujnika. W tym przypadku generowana jest etykieta wejścia znacznie wyprzedzająca moment faktycznego pojawienia się impulsu na sygnale bramki;



Rysunek 3.6. Bilans pszczół w poszczególnych tunelach podczas jednego z eksperymentów.

2. Pszczoła spoza ula wlatuje do tunelu, a następnie zwraca przed dotarciem do czujnika i opuszcza tunel. W tym przypadku generowane są dwie etykiety: wejście i wyjście, pomimo braku faktycznego wykrycia pszczoły przez bramkę;
3. Sytuacja symetryczna dla 1: pszczoła z wewnętrz ula przechodzi przez bramkę czujnika, i oczekuje dłuższy czas przed opuszczeniem tunelu. W tym przypadku generowana jest etykieta wyjścia znacznie opóźniona względem faktycznego wykrycia;
4. Sytuacja symetryczna dla 2: pszczoła z wewnętrz ula przechodzi przez bramkę czujnika, a następnie zwraca przed dotarciem do końca tunelu, i przechodząc ponownie przez czujnik wraca do ula. W tym przypadku nie zostanie wygenerowana żadna etykieta, a na sygnale wyjściowym bramki pojawiają się dwa impulsy: wyjście i wejście.

Przeprowadzone zostało empiryczne porównanie, na podstawie którego stwierdzono, że błędy rodzajów 1, 3 oraz 4 nie występują często (nie natrafiono na ich ślady), natomiast czasem zdarzają się błędy rodzaju 2 (pszczoły zwracające przed dotarciem do bramki – dwie nadmiarowe etykiety). Nie została wyznaczona konkretna częstotliwość pojawiania się błędów, jednak ustalono, że zachodzą one na tyle rzadko, że nie będą stanowić problemu przy zastosowaniu opracowanej anotacji do oceny działania rozwijanych algorytmów. Gdyby planowane było stosowanie metod opartych o machine learning, nawet bardzo rzadkie występowanie błędnych etykiet mogłoby znacząco pogorszyć działanie algorytmów, natomiast w przypadku niniejszej pracy, spowoduje ono jedynie lekkie pogorszenie oceny każdego z proponowanych rozwiązań. Kara ta będzie jednak równa dla każdego z testowanych algorytmów, ponieważ każdy z nich otrzyma tyle samo przykładów testowych z błędnie oznaczonymi rozwiązaniami.

Na podstawie zebranych w niniejszych badaniach przebiegów można jasno stwierdzić, że zaprojektowane czujniki pojemnościowe sprawdzają się we współpracy z żywymi pszczo-

łami poruszającymi się samodzielnie przez tunele urządzenia. Zebrano zbiór danych, na którym możliwe będzie testowanie metod detekcji impulsów wywoływanych przez owady pojawiające się w bramkach czujników – możliwe było zatem przejście do projektowania algorytmów przetwarzania sygnałów, opisanych w kolejnych sekcjach.

3.2. Wstępne przetwarzanie danych

Niezależnie od stosowanej później metody detekcji, surowe dane z czujnika muszą być wstępnie przygotowane. Każdy z sygnałów przejdzie dwa kroki przetwarzania: filtr uśredniający oraz usuwanie trendu. W kolejnych sekcjach opisano te kroki, rozważając je dla sygnału pojedynczego tunelu urządzenia – dla uproszczenia zapisu pominięto we wzorach indeks czujnika i – ten sam wzór stosuje się dla każdego z sygnałów $x_i(k)$, gdzie $i = 0 \dots 7$. Ponadto, ściśle podkreślona zostanie ich dyskretna natura, stąd indeksowanie numerem próbki k , a nie chwilą czasu ciągłego t .

3.2.1. Filtr uśredniający

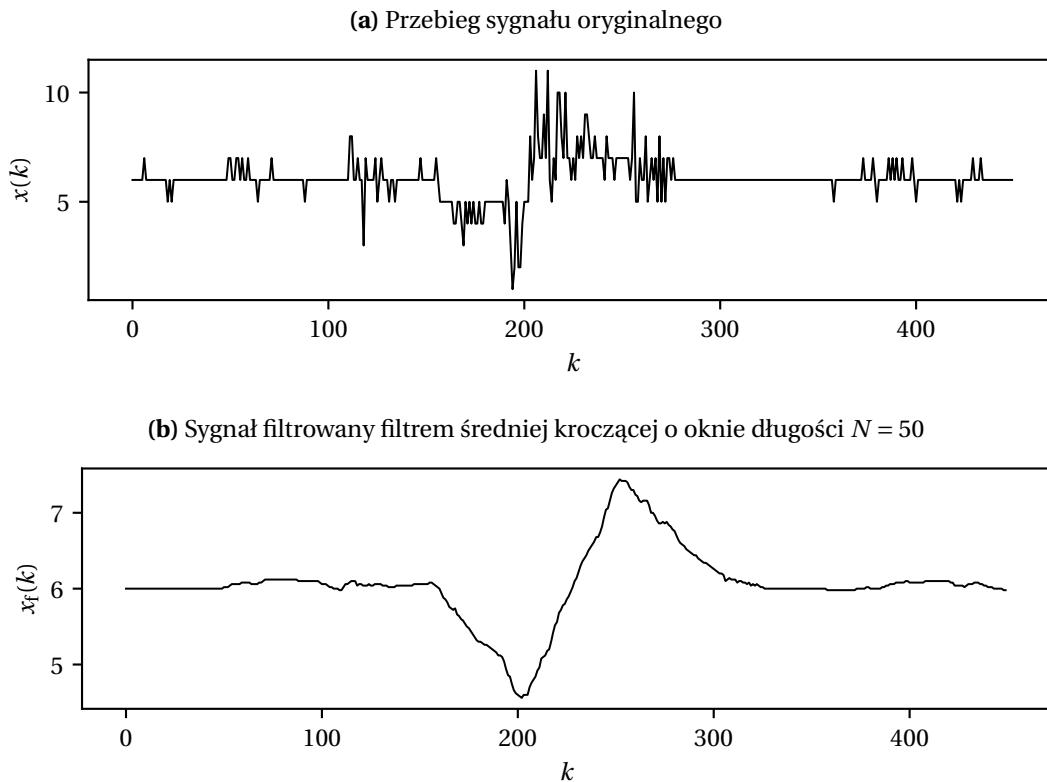
Sygnały wyjściowe czujników pojemnościowych obarczone są znacznym szumem, widocznym wyraźnie m.in. na rysunku 3.7a. Oprócz tego, przyjmuję one jedynie wartości całkowite, przy czym amplituda impulsów nie przekracza zwykle 10 – a co za tym idzie – ich rozdzielcość pozostawia wiele do życzenia. Oba te problemy rozwiązywane są poprzez nałożenie na sygnały filtra średniej kroczej. Zdecydowano się na ten rodzaj filtra, ponieważ jest optymalny na potrzeby niniejszego zastosowania, a przy tym jest jednym z najprostszych i najłatwiejszych w implementacji filtrów cyfrowych [39]. Filtr ten zaimplementowano zgodnie z przedstawionym poniżej wzorem:

$$x_f(k) = \frac{1}{N} \sum_{i=0}^{N-1} x(k-i), \quad (3.1)$$

gdzie $x_f(k)$ oznacza przefiltrowaną wersję sygnału $x(k)$, a N to wybrana długość okna – parametr filtra. Na rysunku 3.7 przedstawione zostało zestawienie oryginalnego, nieprzefiltrowanego sygnału, z wynikiem działania filtra średniej kroczej o przykładowej długości okna $N = 50$. Widoczne są pożądane efekty pracy filtra:

- po pierwsze, zlikwidował on prawie zupełnie sumy obecne w oryginalnych danych;
- ponadto, dzięki uśrednieniu próbek, zbiór wartości sygnału znacznie się powiększył, poprawiona została znaczco również jego rozdzielcość.

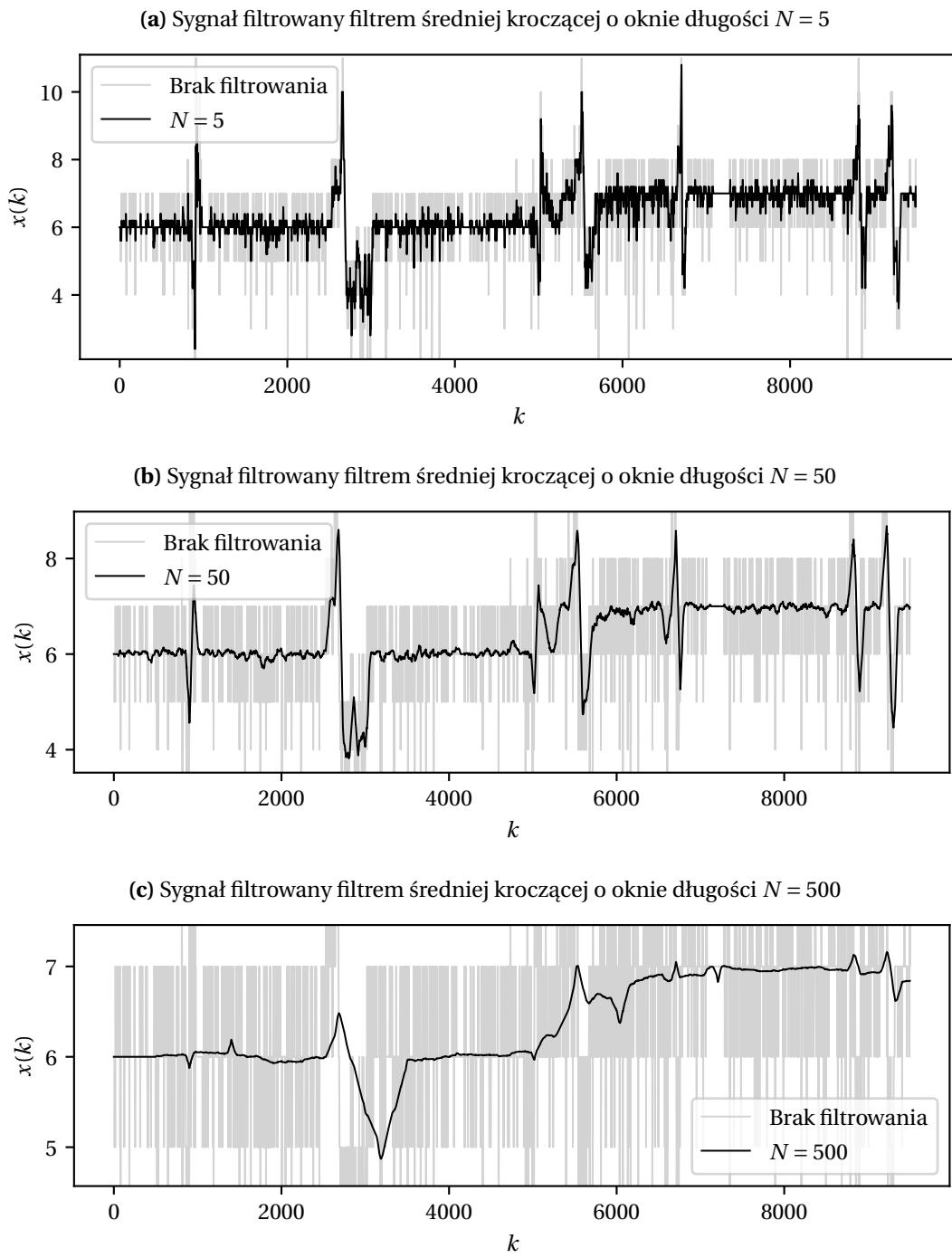
W wyniku obu tych zjawisk, przebieg wyjścia czujnika stał się łagodny, a impuls wygenerowany przez ruch pszczoły w tunelu jest bardzo wyraźnie widoczny. Można stwierdzić, że zaprojektowana metoda filtrowania danych sprawdza się w przypadku niniejszego zastosowania. Warto zwrócić uwagę, że filtrowanie sygnału wpłynęło na znaczące zmniejszenie amplitudy prezentowanego impulsu – stanowi to pewien koszt filtrowania. Dopóki jednak redukcja amplitudy nie zrównuje jej z pozostającym po filtrowaniu szumem, wzory generowane przez pszczoły pozostaną odróżnialne i będzie możliwe ich wykrycie. Inną wadą stosowania filtrów może być wprowadzanie przez nie opóźnienia sygnału, jednak



Rysunek 3.7. Porównanie sygnału oryginalnego z przefiltrowanym filtrem średniej kroczącej.

w przypadku projektowanego systemu nie stanowi to zagrożenia – nie zawiera on sprzężeń zwrotnych, które mogłyby się zdestabilizować, a poza tym opóźnienie w zliczeniu pszczół, nawet wynoszące wiele sekund, zupełnie nie ma znaczenia dla prawidłowości ogólnego szacunku populacji w skali dnia.

Istotne jest odpowiednie dobranie długości okna N , aby zastosowany filtr skutecznie tłumił szумy sygnału, ale przy tym nie redukował zbytnio amplitudy odpowiedzi na ruch pszczół. Na rysunku 3.8 przedstawione zostało porównanie tego samego fragmentu przebiegu poddanego filtracji z różnymi wartościami N : 5, 50 oraz 500. Na ostatnim z prezentowanych wykresów (rysunek 3.8c), widoczne są efekty zbyt długiego okna: szum został skutecznie usunięty, jednak praktycznie zupełnie pozbyto się również impulsów generowanych przez pszczoły. Filtr o $N = 500$ jest znacznie mocny. Z kolei na pierwszym z wykresów (rysunek 3.8a), impulsy pozostają wyraźnie widoczne, ale redukcja szumu nie jest wystarczająca. Dobrą wartością długości okna filtra okazuje się być prezentowane wcześniej $N = 50$ (wykres na rysunku 3.8b): szumy oryginalnego sygnału są w znacznej mierze usunięte, natomiast odpowiedź sygnału na obecność pszczół w tunelu pozostaje ostra i wyraźnie widoczna. Konkretnie wartości N zostaną wybrane na etapie optymalizacji parametrów poszczególnych algorytmów detekcji, jednak już na tym etapie można stwierdzić, że wartości optymalnej należy szukać w pobliżu $N = 50$. Wszystkie przedstawione w kolejnych częściach pracy przebiegi wyjścia czujnika, o ile nie stwierdzono inaczej, zostały uprzednio przefiltrowane średnią kroczącą o oknie tej właśnie długości.



Rysunek 3.8. Porównanie działania filtrów o długościach okna: (a) $N = 5$, (b) $N = 50$, (c) $N = 500$.

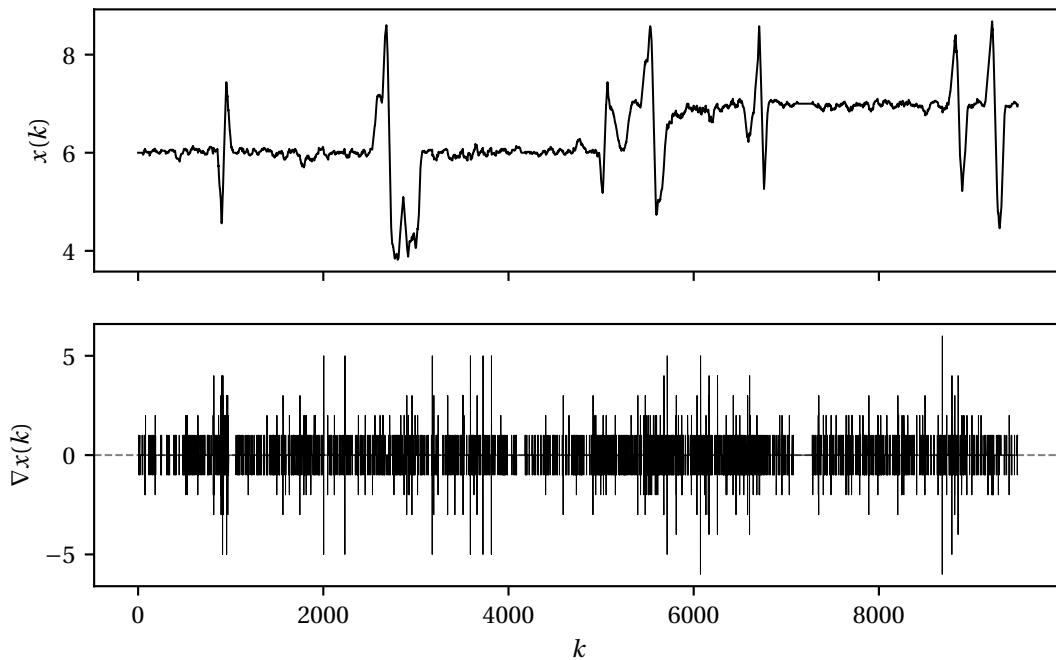
3.2.2. Usuwanie trendu

Przefiltrowany sygnał wciąż nie jest w pełni gotowy do dalszego zastosowania, ze względu na występujący w nim trend. Przesunięcie poziomu spoczynkowego sygnału względem zera wynika z subtelnego różnic pomiędzy kondensatorami w bramce czujnika, może się również w zauważalnym stopniu zmieniać w czasie przez zmienne warunki zewnętrzne – jeden taki skok zawarty jest w analizowanym przykładowym przebiegu (okolice próbki $k = 6000$). Konieczne jest opracowanie metody usuwania trendu, która zapewni, że poziom spoczynkowy sygnału niezależnie od zewnętrznych zakłóceń znajduje się w wartości zero. Należy zadbać, by stworzone rozwiązań nie pogarszało zbytnio jakości impulsów generowanych przez pszczoły.

Najprostszą podejściem, które potencjalnie może przynieść oczekiwane efekty, jest detrending przez różniczkowanie (ang. *detrending by differencing*) [40]. Metoda ta polega na wykorzystaniu, zamiast sygnału $x(k)$ obarczonego trendem, odpowiadającego mu sygnału różnicowemu $\nabla x(k)$, wyznaczonego zgodnie ze wzorem:

$$\nabla x(k) = x(k) - x(k-1). \quad (3.2)$$

Wzór ten zaaplikowano dla przykładowych danych, analizowanych już w poprzedniej sekcji. Na rysunku 3.9 przedstawione zostały wygenerowane przebiegi. Wyraźnie widać, że



Rysunek 3.9. Usuwanie trendu z wykorzystaniem sygnału różnicowego. Jako $x(k)$ oznaczony jest sygnał oryginalny obarczony trendem, natomiast jako $\nabla x(k)$ – wyznaczony na jego podstawie sygnał różnicowy.

szumy pozostające w sygnale zostały w efekcie różniczkowania wzmacnione w stopniu, który uniemożliwia odróżnienie momentów pojawiania się pszczoły. Metoda ta zupełnie się zatem nie nadaje do zastosowania w niniejszej pracy.

Z tego względu, konieczne staje się opracowanie alternatywnej metody usuwania trendu. W dalszej kolejności przetestowano technikę opartą na odejmowaniu od sygnału jego mediany wyznaczonej na pewnym oknie przeszłych próbek. Sygnał pozbawiony trendu, oznaczony $y(k)$, dany jest w niej wzorem:

$$y(k) = x(k) - x_m(k), \quad (3.3)$$

gdzie:

$$x_m(k) = \text{median} \left(\begin{bmatrix} x(k) \\ x(k-1) \\ \vdots \\ x(k-M) \end{bmatrix} \right), \quad (3.4)$$

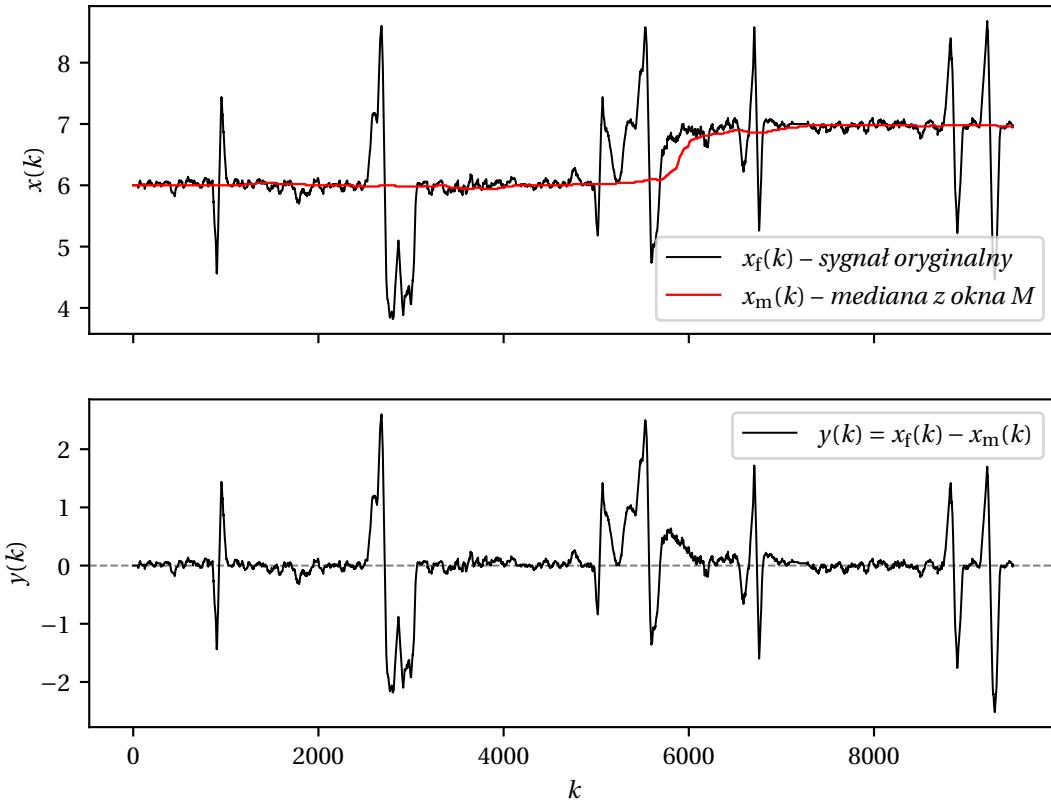
przy czym M to parametr oznaczający długość okna mediany. Wyniki testu tej metody, dla $M = 1200$, zostały przedstawione na rysunku 3.10. Można zaobserwować, że sygnał mediany $x_m(k)$ utrzymuje się na poziomie spoczynku sygnału $x(k)$, natomiast jego odpowiedź na zmianę tego poziomu jest szybka (okolice chwili $k = 6000$). Sygnał $x_m(k)$ nie reaguje ponadto na impulsy wywołane obecnością pszczół – dzięki temu, na wykresie sygnału $y(k)$, pozostały one perfekcyjnie zachowane po usunięciu trendu. Metoda ta działa bardzo dobrze. Nie wzmacnia ona szumów sygnału i pozostawia interesujące impulsy nienaruszone. Jej pewną wadę stanowi jedynie jej złożoność algorytmiczna – wyliczanie mediany w każdym kroku programu wymaga sortowania buforu M ostatnich próbek (o możliwej optymalizacji tego procesu napisano w sekcji 4.1). Stosownym wydało się, z tego względu, przetestowanie pewnej modyfikacji przedstawionej metody detrendingu – wykorzystanie zamiast mediany średniej arytmetycznej. Metryka ta posiada podobne właściwości, jest jednak mniej obciążająca obliczeniowo. Powtórzono eksperyment, zastępując w równaniu 3.3 składnik $x_m(k)$ sygnałem $x_a(k)$, danym jako:

$$x_a(k) = \frac{1}{M} \sum_{i=0}^{M-1} x(k-i). \quad (3.5)$$

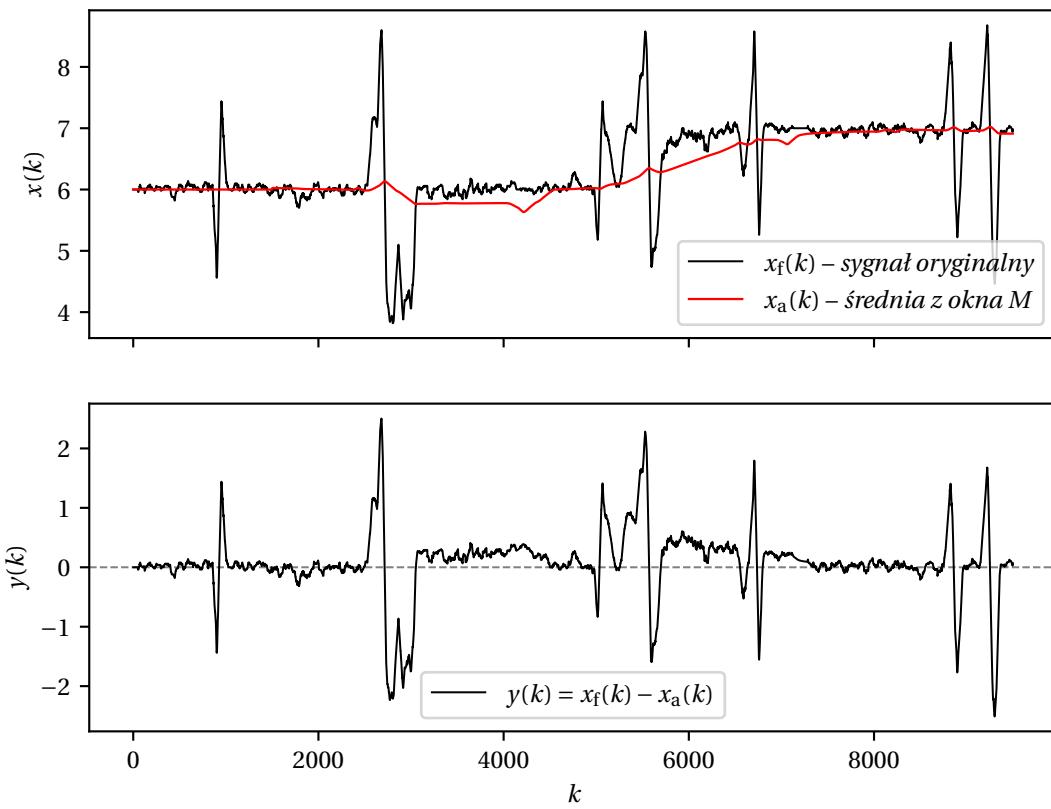
Otrzymane przebiegi zostały przedstawione na rysunku 3.11. Jak widać, zmodyfikowana metoda zachowała tę cechę, że skutecznie usuwa przesunięcie sygnału względem zera (chwile k od 0 do ok. 2000 oraz po $k \approx 7000$). Charakteryzuje ją jednak znaczco wolniejsza odpowiedź na zmianę poziomu przesunięcia sygnału (okolice chwili $k = 6000$) – w której wyniku fragment sygnału, który winien leżeć w okolicach wartości zero, znacznie od niej odbiega. Ponadto, sygnał $x_a(k)$ w widocznym stopniu reaguje na impulsy pszczół, co prowadzi do modyfikacji ich kształtu oraz zmniejszenia amplitudy. W okolicach chwili $k = 4300$ występuje ponadto odpowiedź sygnału $x_a(k)$ na impuls, który pojawił się znacznie wcześniej (ok. $k = 2500$). Deformuje ona fragment przebiegu $y(k)$, który powinien być zupełnie płaski i leżeć w zerze.

Zmodyfikowana wersja metody za mocno odbiega jakością od jej podstawowej wersji, nie zostanie zatem zastosowana w dalszych etapach pracy.

3. Algorytm detekcji



Rysunek 3.10. Metoda usuwania trendu z wykorzystaniem mediany sygnału – przebiegi testowe.



Rysunek 3.11. Wariant usuwania trendu wykorzystujący średnią sygnału – przebiegi testowe.

3.3. Opis proponowanych algorytmów

Po opracowaniu metod wstępniego przetwarzania danych możliwe było przejście do kolejnego, kluczowego etapu pracy – projektu algorytmu detekcji pszczół. Jego wejściem jest sygnał $y(k)$, czyli pomiar czujnika pojemnościowego, poddany uprzednio filtrowaniu i usunięciu trendu. Za zadanie ma on wyznaczenie chwil k , w których w bramce czujnika pojawiły się pszczoły, a także klasyfikację kierunku ruchu pszczół (wejście/wyjście) – w celu ustalenia łącznego bilansu pszczół, które weszły do ula. Ogólna struktura algorytmu została przedstawiona na rysunku 3.12. Wewnętrzny licznik oznaczono jako *bee*, natomiast wyjściem systemu jest jego wartość na daną chwilę k , czyli $bee(k)$. Zaprojektowano kilka

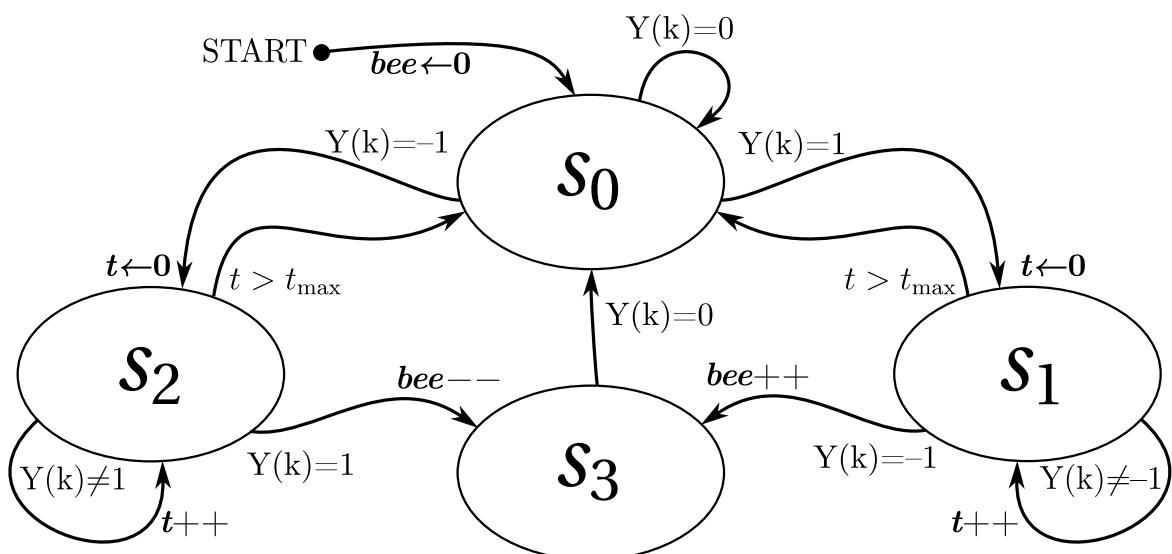


Rysunek 3.12. Ogólna struktura algorytmu detekcji.

różnych algorytmów detekcji, które potencjalnie mogą zostać wykorzystane do analizy generowanych sygnałów. Zostały one opisane w kolejnych sekcjach pracy.

3.3.1. Automat stanowy

Główną ideą stojącą za pierwszą proponowaną metodą detekcji pszczelnych impulsów jest prosta obserwacja, że fragmenty które należy wykryć zawsze składają się z dwóch ostrzych szpilek sygnału, następujących tuż po sobie. Algorytm opiera się na progowaniu $y(k)$, w celu wykrycia skoków jego wartości, a następnie wykorzystaniu automatu stanowego – FSM (rysunek 3.13), który wykryje zadane sekwencje w skwantowanym sygnale.



Rysunek 3.13. Struktura automatu stanowego. Warunki przejścia zapisano przy początkach strzałek, natomiast akcje przejścia przy końcach strzałek. Dodatkowo, akcje oznaczono pogrubieniem.

3. Algorytm detekcji

Sygnal progowany $Y(k)$ wyznaczany jest na podstawie $y(k)$ zgodnie ze wzorem:

$$Y(k) = \begin{cases} 1 & \text{jeśli } y(k) > \gamma(k) \\ 0 & \text{jeśli } -\gamma \leq y(k) \leq \gamma(k) \\ -1 & \text{jeśli } y(k) < -\gamma(k), \end{cases} \quad (3.6)$$

gdzie γ oznacza wartość progowania. Rozważane są dwa warianty niniejszej metody:

1. Progowanie statyczne, w którym $\gamma(k) = \gamma_0$ dla każdej chwili k ;
2. Progowanie adaptacyjne, w którym $\gamma(k)$ jest dynamicznie dostosowywane do właściwości sygnału wejściowego. Zastosowany został następujący wzór na wartość dynamicznego progu:

$$\gamma(k) = \alpha \cdot \mu \cdot \hat{Q}_y(q, K, k) + (1 - \alpha) \cdot \gamma_0, \quad (3.7)$$

gdzie α , μ oraz q to parametry strojenia, γ_0 to wartość progu statycznego, natomiast $\hat{Q}_y(q, K, k)$ oznacza q -ty kwantyl ze zbioru K ostatnich próbek y dla chwili k . Zapisano wzór na próg dolny – dla progu górnego przyjąć kwantyl $(1 - q)$.

Struktura zaprojektowanego automatu jest symetryczna. Jego prawa część (stany s_0, s_1, s_3) odpowiada za wykrywanie pszczół wchodzących do ula. W stanie s_0 , algorytm oczekuje pojawienia się jedynki na wejściu (wystąpienia dodatniej szpilki na sygnale czujnika) – w wyniku której przechodzi do stanu s_1 . Jeśli niedługo po przejściu na wejściu pojawi się -1 , system wykrywa wchodzącą pszczołę: inkrementuje licznik *bee* i przechodzi do stanu s_3 . Następnie, oczekuje na pojawienie się ponownie zera na wejściu, by powrócić do oczekiwania w stanie s_0 . Dodatkowo, podczas wejścia w stan s_1 , ustawiany jest na zero licznik *t*. Zlicza on, ile iteracji trwa oczekiwanie w stanie s_1 – jeżeli przekroczeniu zostanie wartość t_{\max} (parametr algorytmu), następuje powrót do stanu s_0 . Dzięki temu, algorytm nie zablokuje się przy wystąpieniu pojedynczej szpilki, wynikającej na przykład z szumu danych wejściowych. Lewa część automatu (stany s_0, s_2, s_3) działa analogicznie, z tą różnicą, że wykrywa odwrotną sekwencję sygnału wejściowego, a licznik *bee* jest dekrementowany.

Algorytm zgodny z powyższym opisem, w obu wariantach, uruchomiono na fragmencie danych eksperymentalnych. Na rysunkach 3.14 oraz 3.15 przedstawiono uzyskane w ten sposób przebiegi. Zastosowano konfiguracje parametrów przedstawione w tabeli 3.2.

Tabela 3.2. Parametry testowanych automatów stanowych.

| Parametr | Symbol | FSM, prog. statyczne | FSM, prog. adaptacyjne |
|--------------------------------|------------|----------------------|------------------------|
| Długość okna filtra | N | 30 | 30 |
| Długość okna mediany | M | 1200 | 1200 |
| Maks. oczekiwanie na impuls | t_{\max} | 200 | 200 |
| Wartość progu statycznego | γ_0 | 1.0 | 1.0 |
| Waga progu adaptacyjnego | α | — | 0.4 |
| Mnożnik progu adaptacyjnego | μ | — | 7 |
| Kwantyl progu adaptacyjnego | q | — | 0.2 |
| Dł. okna kwantylu prog. adapt. | K | — | 1500 |

Warto uważnie przyjrzeć się wykresom przedstawionym na rysunku 3.14. Na górnym panelu zaprezentowano momenty przejść pszczół przez czujnik, załadowane ze zbioru etykiet wypracowanego wcześniej (oznaczone *Anotacja*). Na zielono oznaczono pszczoły wchodzące do ula, natomiast na czerwono wychodzące. Osie czasowe wszystkich przebiegów na rysunku wyskalowane są jednakowo, można zatem stwierdzić, że wszystkie etykiety są prawidłowe – w jasny sposób odpowiadają impulsom widocznym na sygnale $y(k)$ (panel środkowy). Na tle przebiegu wejścia algorytmu przedstawione zostały linie, reprezentujące poziom progu γ_0 . Oznaczono próg górny i dolny. Na ostatnim panelu rysunku zawarto wykres skwantowanego sygnału wejściowego $Y(k)$, a na jego tle wygenerowane przez algorytm momenty inkrementacji lub dekrementacji licznika *bee* (oznaczone *Detekcja*).

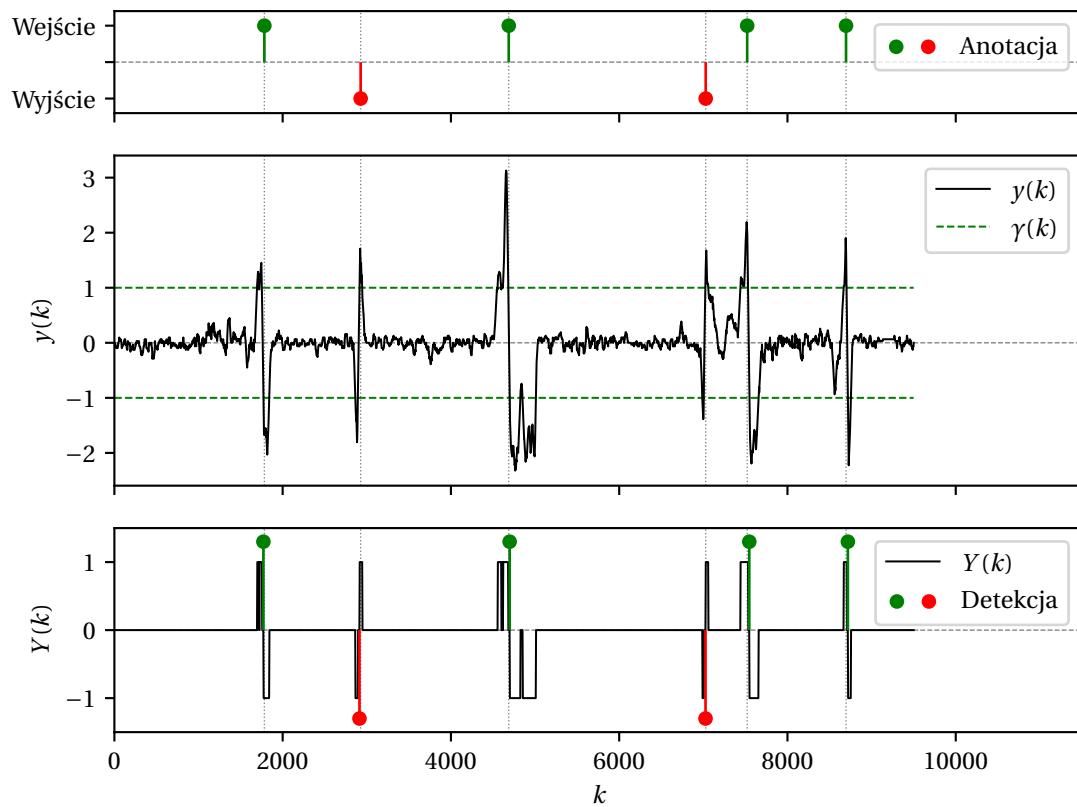
Widać, że progowanie statyczne pozwoliło skutecznie wykryć skoki $y(k)$, generując prawidłowy sygnał $Y(k)$. Automat stanowy również działa na niniejszym przykładzie poprawnie – każdy z impulsów w $y(k)$ został wykryty, a ich typy zostały sklasyfikowane poprawnie. Zastosowana struktura automatu poradziła sobie z nieidealnym sygnałem $Y(k)$ (pierwszy i trzeci impuls nie składają się z pojedynczych szpilek).

Na rysunku 3.15 przedstawione zostały analogiczne przebiegi uzyskane z wykorzystaniem adaptacyjnego progowania sygnału $y(k)$. Należy zwrócić uwagę na kształt linii oznaczających poziomy progu γ (środkowy panel). Zmieniają się one w czasie, dostosowując się do szumów i skoków $y(k)$. Można wyciągnąć analogiczne wnioski, jak dla poprzedniej metody: progowanie i automat stanowy pozwoliły z pełną skutecznością zrealizować dane im zadanie.

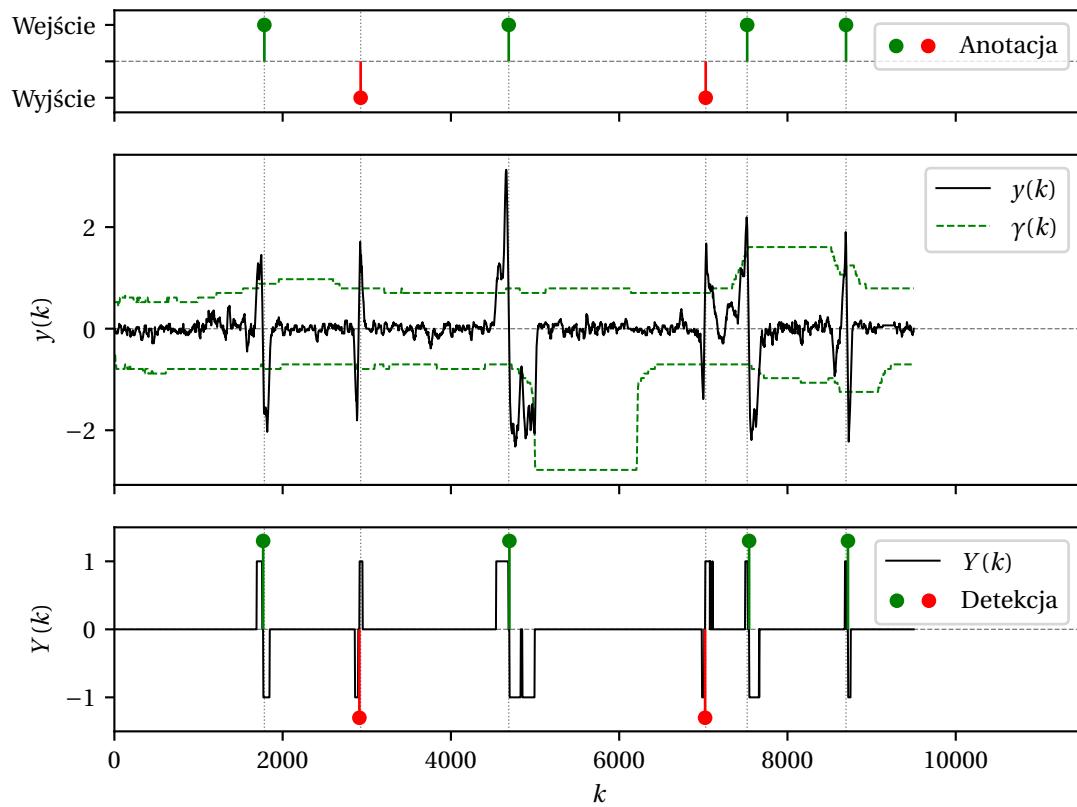
Na podstawie wykonanych doświadczeń można wybrać metodę ze stałym $\gamma(k)$ jako lepszą, ze względu na mniejszą złożoność obliczeniową wyznaczania wartości progowych. Wykonane zostały jednak dodatkowe eksperymenty, na przykładzie innego fragmentu danych eksperymentalnych, z wadliwym fragmentem przebiegu. Przebieg ten, przedstawiony m.in. na rysunku 3.16, zawiera szum dorównujący amplitudą typowym impulsom pszczół. Ten chaotyczny wzór został, najprawdopodobniej, spowodowany przez pszczołę, która zatrzymała się w środku bramki czujnika. Zdarzenie to, choć rzadkie (znaleziono tylko jeden taki fragment w całości zebranych danych), może potencjalnie mieć znaczący negatywny wpływ na estymację bilansu pszczół. Oba warianty algorytmu uruchomiono na omawianym fragmencie przebiegu. Wyniki dla progowania statycznego jednoznacznie potwierdzają powyższą tezę: gdy szum przekroczył poziom progowania, automat zaczął zliczać dziesiątki pszczół. W przypadku progowania adaptacyjnego natomiast, poziomy γ podążały za rosnącą amplitudą szumu, znaczco redukując liczbę przypadkowych zliczeń (dwa błędy *false positive* kontra 25). Ze względu na wzrost γ , nie zostały zliczone dwa przejścia, które pojawiły się podczas okresu podwyższonego szumu; pomimo to, algorytm adaptacyjny popełnił przeważającco mniej błędów.

Szczegółowe porównanie obu metod, wraz z opisem optymalizacji parametrów, zostały zawarte w sekcji 3.4. Na ten moment utrzymać można hipotezę, że algorytm adaptacyjny będzie charakteryzował się lepszymi wynikami, kosztem większej złożoności obliczeniowej.

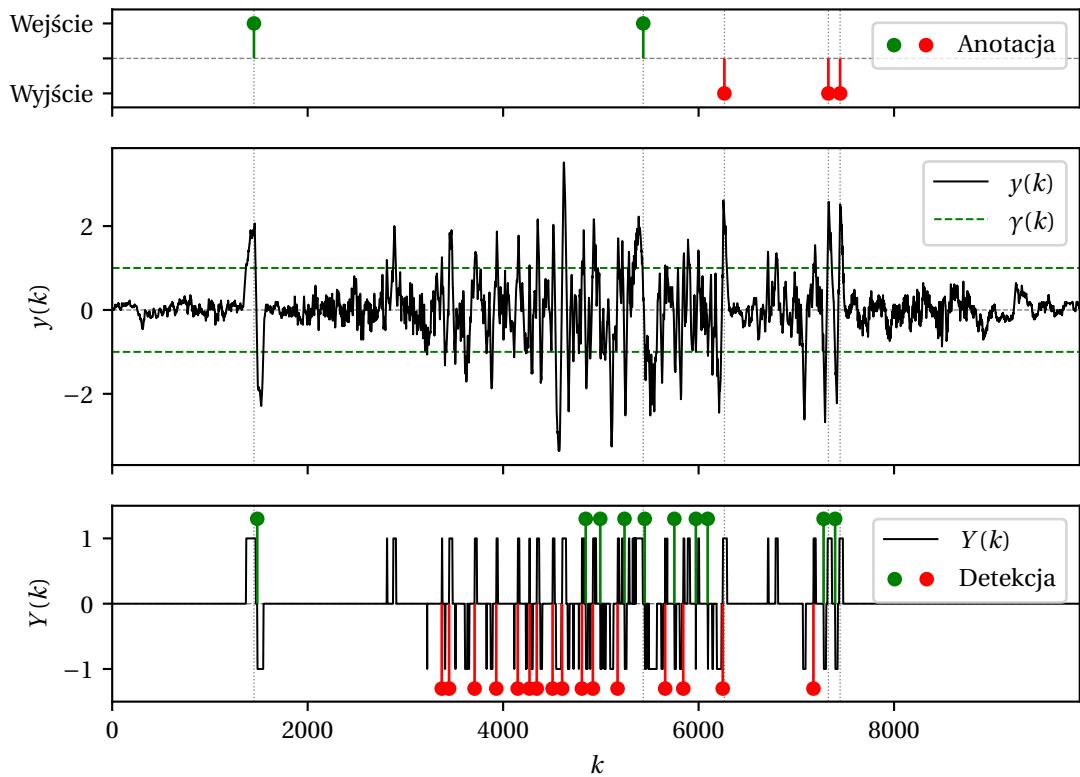
3. Algorytm detekcji



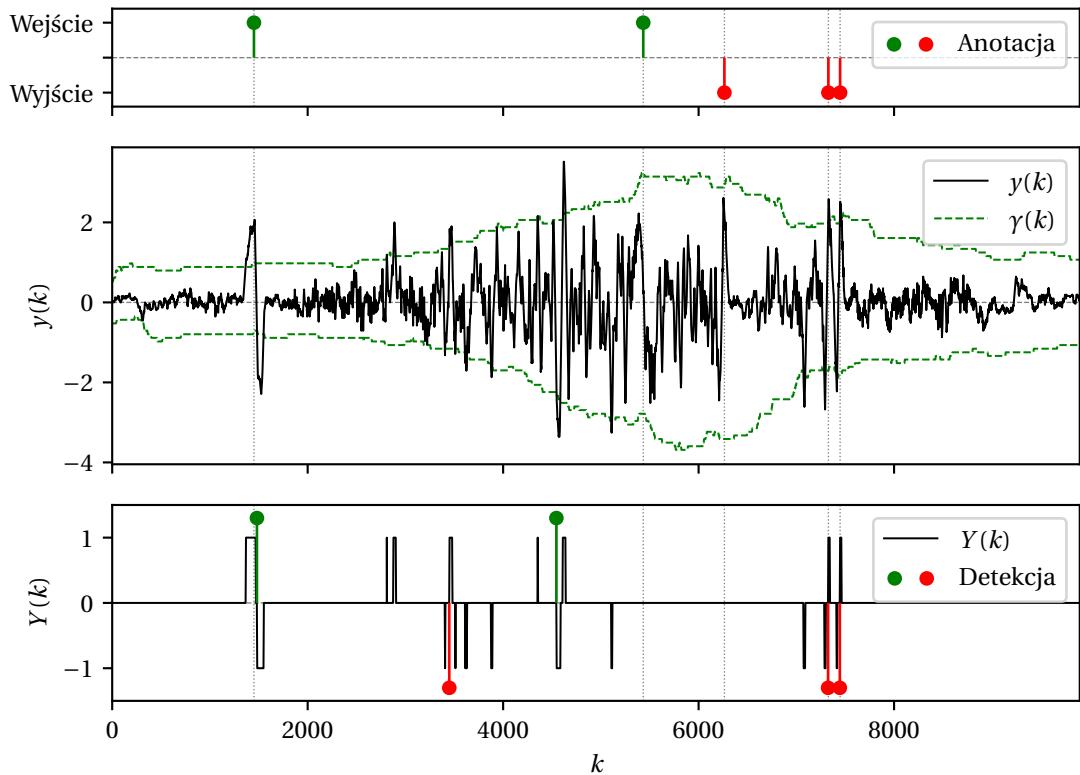
Rysunek 3.14. Fragment przebiegów z testu automatu stanowego z progowaniem statycznym.



Rysunek 3.15. Fragment przebiegów z testu automatu stanowego z progowaniem adaptacyjnym.



Rysunek 3.16. Fragment przebiegów z testu automatu stanowego z progowaniem statycznym, zawierający słabej jakości sygnał wejściowy algorytmu.

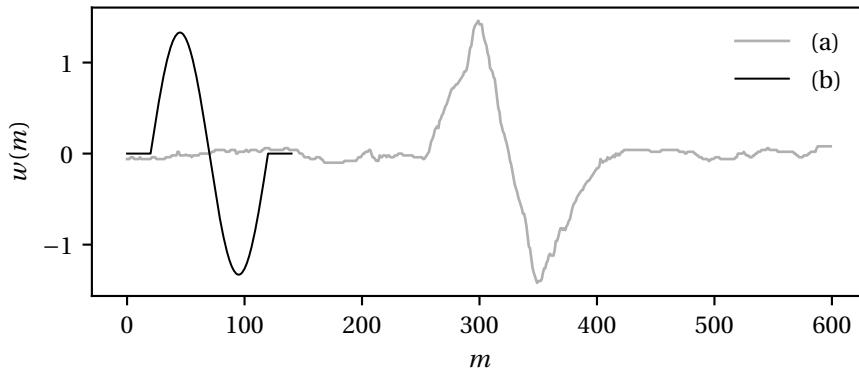


Rysunek 3.17. Fragment przebiegów z testu automatu stanowego z progowaniem adaptacyjnym, zawierający słabej jakości sygnał wejściowy algorytmu.

3.3.2. Korelacja wzajemna z wzorcem

Kolejna z proponowanych metod opiera się na wykorzystaniu korelacji krzyżowej sygnału $y(k)$ z opracowanym wzorcem idealnego impulsu pszczoły $w(m)$. Korelacja krzyżowa stanowi miarę podobieństwa sygnałów, w funkcji ich przesunięcia względem siebie – nadaje się zatem idealnie do wykrycia chwil, w których w sygnale wejściowym występuje zadany wzorzec.

Na rysunku 3.18 zostały przedstawione dwa przykładowe wzorce zastosowane w ramach przeprowadzonych z niniejszym algorytmem eksperymentów. Jeden z nich, oznaczony jako (a), to wycięty fragment zebranych danych, zawierający szczególnie niezakłócony impuls wywołyany ruchem pszczoły. Drugi, oznaczony (b), jest wzorcem syntetycznym, będącym fragmentem sinusoidy przeskalowanym tak, by odpowiadać długością i amplitudą poprzedniemu.



Rysunek 3.18. Przykładowe wzorce impulsu generowanego przez pszczołę; a – wzorzec wybrany z danych eksperymentalnych, b – wzorzec syntetyczny (fragment sinusoidy).

Sygnal korelacji krzyżowej $y(k)$ z wzorcem $w(m)$ o długości M dany jest jako:

$$r_{wy}(k) = \sum_{m=0}^{M-1} w(m) y(k+m). \quad (3.8)$$

Zgodnie z właściwościami korelacji krzyżowej, na przebiegu $r_{wy}(k)$ występować będzie dodatni skok wartości na fragmentach, w których przebieg $y(k)$ jest podobny do wzorca $w(m)$. Ponadto, pojawić będzie się ujemny skok $r_{wy}(k)$, kiedy na przebiegu $y(k)$ wystąpi wzorzec $w(m)$ o odwróconej polaryzacji. Zgodnie z powyższym, aby wyznaczyć momenty, w których system powinien naliczać pszczoły, wystarczy wykrywać gwałtowne skoki wartości sygnału korelacji krzyżowej $r_{wy}(k)$. Zadanie to będzie realizowane poprzez progowanie tego sygnału, zgodnie z poniższym wzorem:

$$R(k) = \begin{cases} 1 & \text{jeśli } r_{wy}(k) > \gamma(k) \\ 0 & \text{jeśli } -\gamma(k) \leq r_{wy}(k) \leq \gamma(k) \\ -1 & \text{jeśli } r_{wy}(k) < -\gamma(k), \end{cases} \quad (3.9)$$

przy czym $\gamma(k)$ (poziom progowania), podobnie jak w przypadku automatu stanowego, może przyjąć stałą wartość, lub być dynamicznie dostosowywany – do właściwości sygnału $r_{wy}(k)$. Na podstawie wartości progowanej, dla każdej chwili k , wyznaczany jest inkrement licznika pszczół *bee*, oznaczony jako $B(k)$. By każda pszczoła została zliczona dokładnie raz, $B(k)$ wynosi 1 lub -1 tylko w chwili pojawienia się zbocza narastającego $R(k)$ do wartości 1, lub zbocza opadającego $R(k)$ do wartości -1, zgodnie z poniższym wzorem.

$$B(k) = \begin{cases} 1 & \text{jeśli } R(k) = 1 \wedge R(k-1) \neq 1 \\ -1 & \text{jeśli } R(k) = -1 \wedge R(k-1) \neq -1 \\ 0 & \text{w pozostałych przypadkach.} \end{cases} \quad (3.10)$$

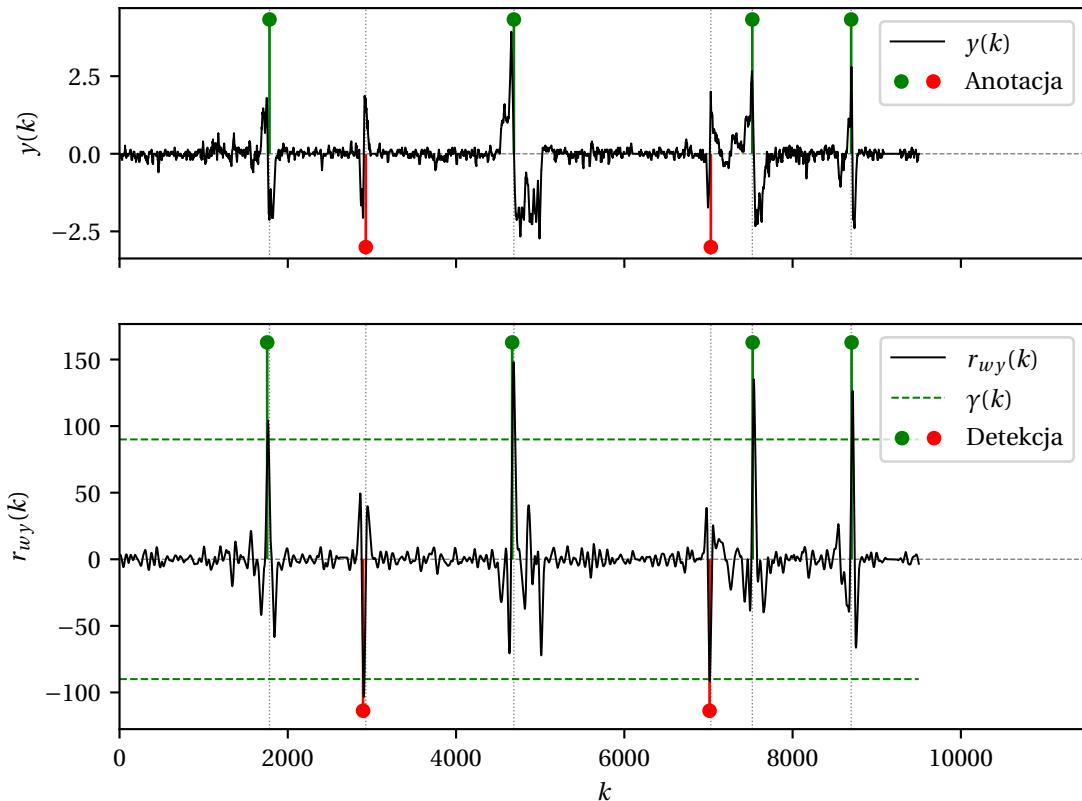
Opisany algorytm został uruchomiony na przykładowym fragmencie $y(k)$ w dwóch konfiguracjach: z progowaniem statycznym oraz adaptacyjnym. Obrane wartości parametrów zostały zawarte w tabeli 3.3. Przebiegi uzyskane w ramach obu eksperymentów przedstawione zostały na rysunkach 3.19 oraz 3.20.

Tabela 3.3. Parametry testowanych algorytmów korelacji krzyżowej.

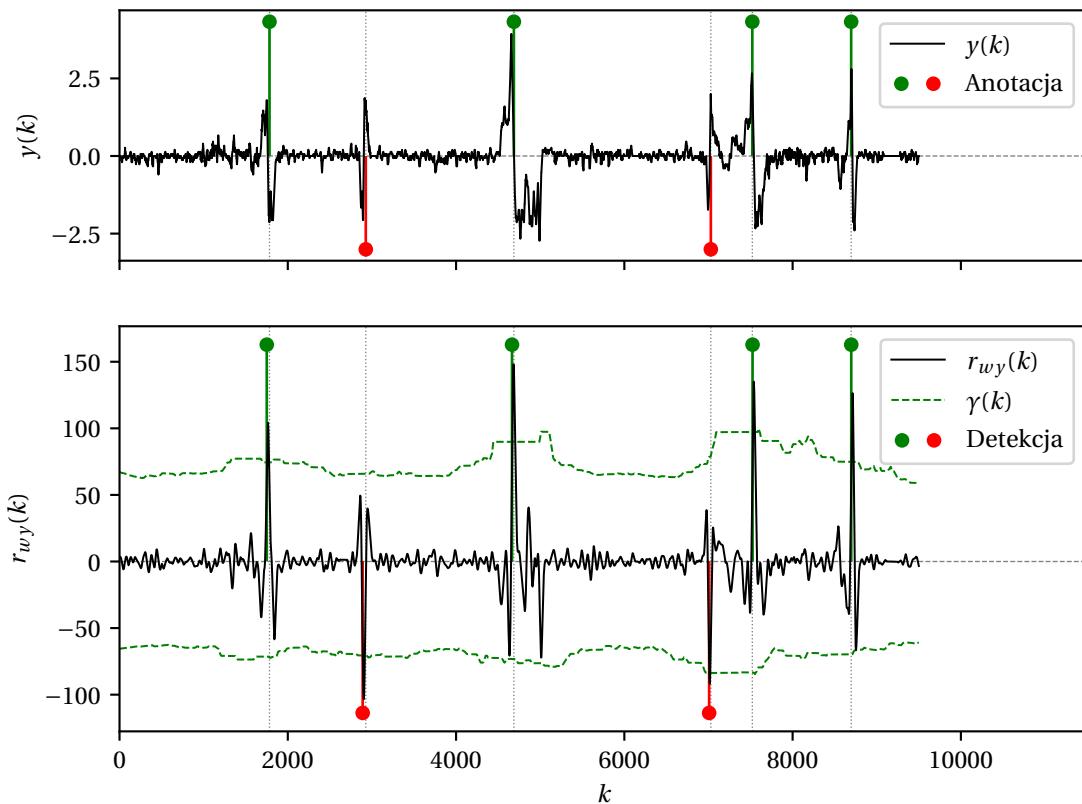
| Parametr | Symbol | Korelacja, prog. stat. | Korelacja, prog. adapt. |
|--------------------------------|------------|--------------------------|--------------------------|
| Długość okna filtra | N | 30 | 30 |
| Długość okna mediany | M | 1200 | 1200 |
| Wartość progu statycznego | γ_0 | 90 | 90 |
| Waga progu adaptacyjnego | α | — | 0.4 |
| Mnożnik progu adaptacyjnego | μ | — | 9 |
| Kwantyl progu adaptacyjnego | q | — | 0.2 |
| Dł. okna kwantylu prog. adapt. | K | — | 1200 |
| Zastosowany wzorzec impulsu | $w(m)$ | syntetyczny (rys. 3.18b) | syntetyczny (rys. 3.18b) |

Zgodnie z oczekiwaniem, impulsom na przebiegu $y(k)$ odpowiadają szpilki występujące na $r_{wy}(k)$. Oba zastosowane sposoby progowania pozwoliły na osiągnięcie bezbłędnej detekcji i klasyfikacji pszczół – chwile k oznaczone przez algorytm (ozn. Detekcja) odpowiadają tym ze zbioru etykiet (ozn. Anotacja). Podobnie jak w przypadku automatu stanowego, przeprowadzono dodatkowy eksperyment na fragmencie zakłóconych danych wejściowych, by porównać odporność obu wariantów algorytmu. Uzyskane wyniki zawarte zostały na rysunku 3.21. Zgodnie z oczekiwaniem, algorytm z progowaniem adaptacyjnym zadziałał w tym przypadku znacznie lepiej – wygenerował tylko jedną błędą detekcję (kontra 9 dla algorytmu z progowaniem statycznym). Oprócz tego, był on w stanie wykryć jeden z impulsów występujących razem z zakłóceniem, z czym nie poradził sobie automat stanowy. Na podstawie obu przeprowadzonych algorytmów stwierdzono, że algorytm korelacji krzyżowej ze wzorcem działa poprawnie. Stawiana jest również hipoteza, że jest on lepszy niż algorytm z automatem stanowym.

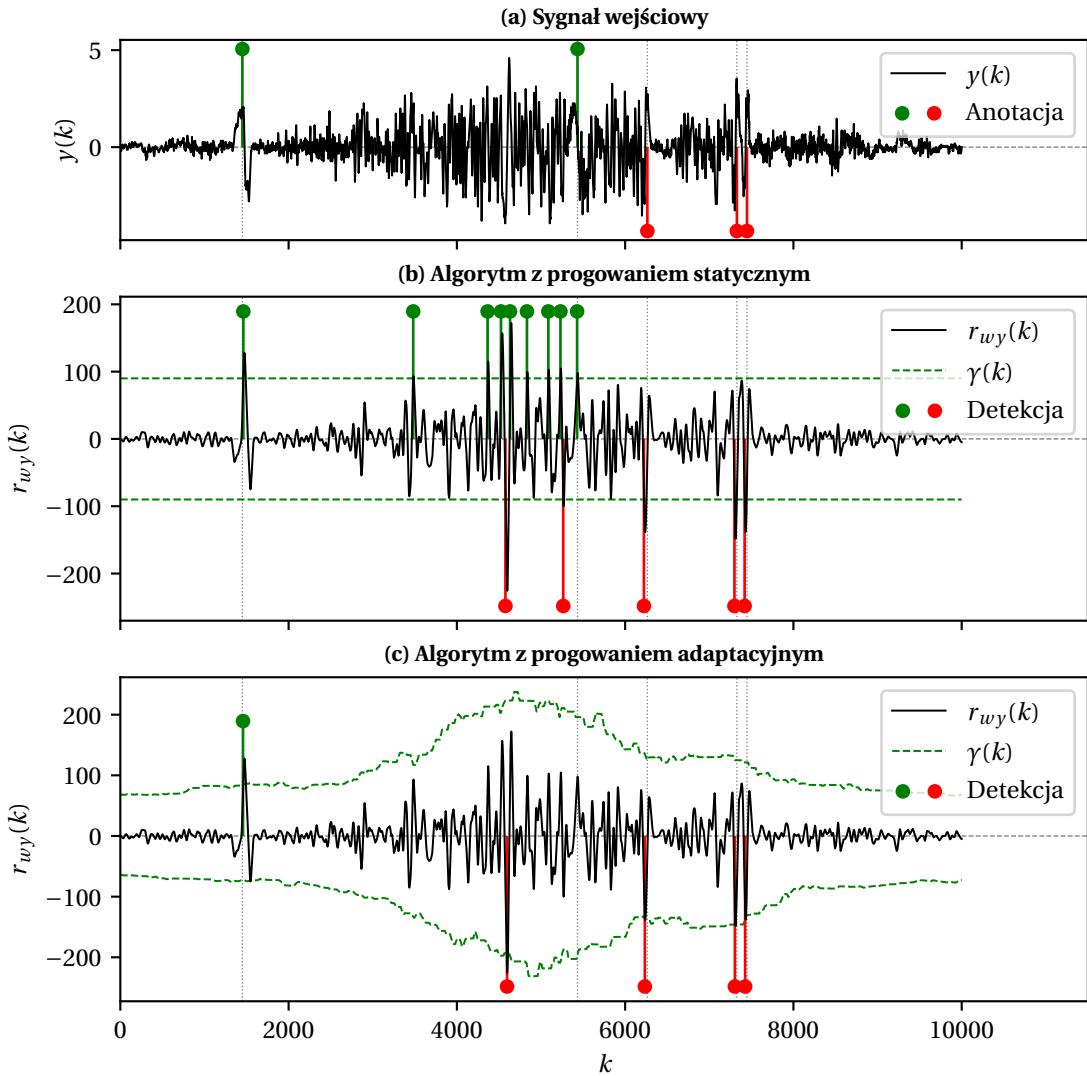
3. Algorytm detekcji



Rysunek 3.19. Fragment przebiegów z testu korelacji krzyżowej z progowaniem statycznym.



Rysunek 3.20. Fragment przebiegów z testu korelacji krzyżowej z progowaniem adaptacyjnym.



Rysunek 3.21. Porównanie działania wariantów algorytmu korelacji krzyżowej na fragmencie danych wejściowych niskiej jakości.

Dodatkowym rozszerzeniem omawianego algorytmu jest zastosowanie wielu różnych wzorców $w(m)$. Rozwiążanie takie może pozwolić na skuteczniejsze rozpoznawanie impulsów o różnych kształtach, amplitudach, lub czasie trwania. Sygnały korelacji $y(k)$ ze wszystkimi wzorcami mogą być uśredniane, w celu wyznaczenia łączonego $r_{wy}(k)$. Alternatywnie, zastosować można medianę, lub najwyższą wartość spośród poszczególnych sygnałów korelacji. Kilka wariantów takiej modyfikacji algorytmu zostało przetestowane w ramach niniejszej pracy, a uzyskane wyniki zostaną przedstawione w kolejnej sekcji.

3.4. Wyniki testów

W celu wyboru najlepszego z algorytmów, a także jego zestawu parametrów, należało przeprowadzić szereg eksperymentów testujących ich skuteczność. Każdy wariant algorytmu uruchamiano na całości zebranych danych rzeczywistych ze wszystkich tuneli urządzenia. Jakość jego działania oceniano według metryki E zdefiniowanej jako:

$$E = \frac{1}{2} \left(F_1^{\text{wejście}} + F_1^{\text{wyjście}} \right), \quad (3.11)$$

gdzie norma F_1^t typu zdarzenia t (wejście lub wyjście z ula) dana jest jako:

$$F_1^t = 2 \cdot \frac{\text{precyzja}^t \cdot \text{czułość}^t}{\text{precyzja}^t + \text{czułość}^t}, \quad (3.12)$$

przy czym precyzja (ang. *precision*), oraz czułość (ang. *recall*), wyznaczone są zgodnie z następującymi wzorami:

$$\text{precyzja}^t = \frac{\text{TP}^t}{\text{TP}^t + \text{FP}^t} \quad (3.13)$$

$$\text{czułość}^t = \frac{\text{TP}^t}{\text{TP}^t + \text{FN}^t}. \quad (3.14)$$

TP^t , FP^t oraz FN^t oznaczają, kolejno, całkowitą liczbę prawdziwie dodatnich (ang. *true positive*), fałszywie dodatnich (ang. *false positive*), oraz fałszywie ujemnych (ang. *false negative*) wyników wygenerowanych przez algorytm dla rodzaju zdarzenia t . Metryki te obliczone są zgodnie z następującą metodą: Niech A^t oznacza zbiór chwil oznaczonych przez algorytm jako czasy występowania zdarzenia t , natomiast L^t – zbiór etykiet t , czyli ręcznie oznaczonych chwil, dla których faktycznie wystąpiło t . Dla każdego elementu a ze zbioru A^t , jeżeli L^t zawiera odpowiadający mu element l , nie różniący się od a o więcej niż 3 sekundy, inkrementuje się TP^t oraz usuwa l ze zbioru L^t ; w przeciwnym wypadku, inkrementuje się FP^t . Po przetworzeniu całego zbioru A^t , wartość FN^t równa jest liczbie elementów pozostałych w L^t .

W ramach poszukiwania najlepszego algorytmu przetestowano dziesiątki różnych konfiguracji, w ramach równie wielu eksperymentów. Szczegółowy proces optymalizacji parametrów każdej z metod nie został udokumentowany w ramach niniejszej pracy; zamiast tego zdecydowano się zaprezentować kilka wybranych konfiguracji, które uznano za najważniejsze. Zostały one przedstawione poniżej:

1. Automat stanowy, progowanie statyczne:

- parametry przetwarzania wstępnego: $N = 31$, $M = 1269$;
- parametry automatu stanowego: $t_{\max} = 180$;
- parametry progowania: $s_0 = 1.22$;

2. Automat stanowy, progowanie adaptacyjne:

- parametry przetwarzania wstępnego: $N = 31$, $M = 1269$;
- parametry automatu stanowego: $t_{\max} = 180$;

- parametry progowania: $s_0 = 0.87, \alpha = 0.4, \mu = 7, q = 0.2, K = 1500$;
3. Korelacja krzyżowa, wzorzec rzeczywisty, progowanie statyczne:
- parametry przetwarzania wstępnego: $N = 36, M = 1252$;
 - parametry korelacji krzyżowej: 1 wzorzec wybrany z danych eksperymentalnych (rys. 3.18a);
 - parametry progowania: $s_0 = 100$;
4. Korelacja krzyżowa, 3 wzorce rzeczywiste, progowanie statyczne:
- parametry przetwarzania wstępnego: $N = 36, M = 1252$;
 - parametry korelacji krzyżowej: 3 wzorce wybrane z danych eksperymentalnych, o zróżnicowanym kształcie. Sygnały korelacji uśrednione;
 - parametry progowania: $s_0 = 85$;
5. Korelacja krzyżowa, 3 wzorce rzeczywiste, progowanie adaptacyjne:
- parametry przetwarzania wstępnego: $N = 36, M = 1252$;
 - parametry korelacji krzyżowej: 3 wzorce tak samo jak w punkcie poprzednim;
 - parametry progowania: $s_0 = 0.87, \alpha = 0.4, \mu = 8.6, q = 0.2, K = 1200$;
6. Korelacja krzyżowa, wzorzec syntetyczny, progowanie statyczne:
- parametry przetwarzania wstępnego: $N = 15, M = 850$;
 - parametry korelacji krzyżowej: 1 wzorzec syntetyczny: jeden okres sinusoidy, rozciągnięty na 81 próbek, i przeskalowany o współczynnik 1.33, a następnie otoczony z obu stron 20 próbkami równymi 0 (rys. 3.18b);
 - parametry progowania: $s_0 = 88$;
7. Korelacja krzyżowa, 3 wzorce syntetyczne, progowanie statyczne:
- parametry przetwarzania wstępnego: $N = 15, M = 850$;
 - parametry korelacji krzyżowej: 3 wzorzec syntetyczny – jeden jak powyżej, pozostałe przeskalowane w czasie i amplitudzie;
 - parametry progowania: $s_0 = 80$;
8. Korelacja krzyżowa, wzorzec syntetyczny, progowanie adaptacyjne:
- parametry przetwarzania wstępnego: $N = 15, M = 850$;
 - parametry korelacji krzyżowej: 1 wzorzec syntetyczny – jak w punkcie 6;
 - parametry progowania: $s_0 = 94, \alpha = 0.4, \mu = 9, q = 0.2, K = 1200$;

Wyniki w metryce E uzyskane przez każdy z opisanych algorytmów zostały zaprezentowane w tabeli 3.4. Automat stanowy z progowaniem adaptacyjnym uzyskał najwyższy wynik, wynoszący $E = 0.9044$ – zgodnie z oczekiwaniemi przewyższając skutecznością wariant z progowaniem statycznym – ten jednak, okazał się również bardzo dobry. Jedynym innym algorymem z wyższym wynikiem od niego jest korelacja krzyżowa z wzorcem syntetycznym i progowaniem adaptacyjnym (drugie miejsce, wynik $E = 0.8740$). Korelacja krzyżowa wykorzystująca kilka wzorców impulsu, w przypadku zastosowania wzorców rzeczywistych, uzyskała wynik znacznie lepszy niż w przypadku pojedynczego wzorca ($E = 0.8209$ kontra $E = 0.7656$). Wbrew oczekiwaniom, efekt ten nie jest jednak widoczny w przypadku wzorca syntetycznego, gdzie wariant z pojedynczym wzorcem uzyskał lepszy

Tabela 3.4. Wyniki porównania algorytmów detekcji.

| Lp. | Algorytm | Wynik E |
|-----|---|-----------|
| 1 | Automat stanowy, progowanie statyczne | 0.8719 |
| 2 | Automat stanowy, progowanie adaptacyjne | 0.9044 |
| 3 | Korelacja krzyżowa, wzorzec rzeczywisty, prog. statyczne | 0.7656 |
| 4 | Korelacja krzyżowa, 3 wzorce rzeczywiste, prog. statyczne | 0.8209 |
| 5 | Korelacja krzyżowa, 3 wzorce rzeczywiste, prog. adaptacyjne | 0.8370 |
| 6 | Korelacja krzyżowa, wzorzec syntetyczny, p. statyczne | 0.8336 |
| 7 | Korelacja krzyżowa, 3 wzorce syntetyczne, p. statyczne | 0.8014 |
| 8 | Korelacja krzyżowa, wzorzec syntetyczny, p. adaptacyjne | 0.8740 |

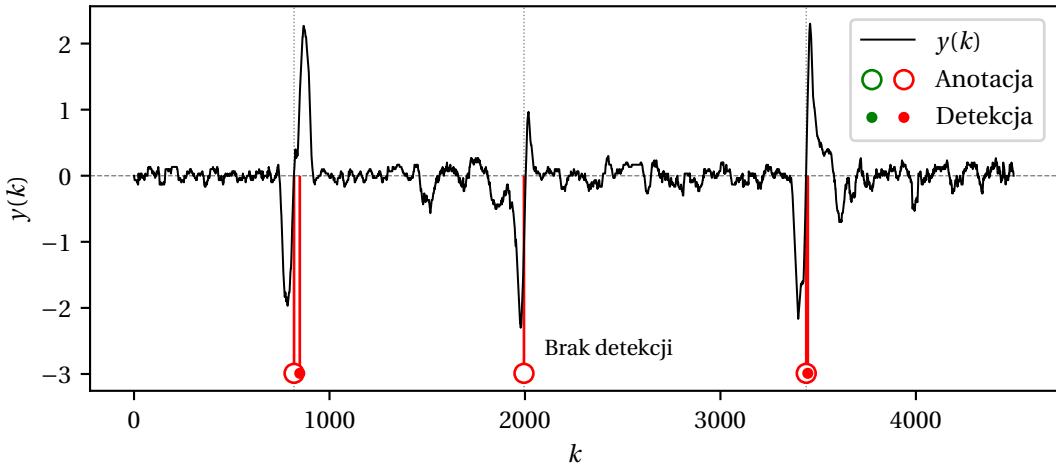
wynik. Progowanie adaptacyjne poprawiło działanie korelacji krzyżowej z wzorcami rzeczywistymi, jednak w zaskakującą małym stopniu. W przypadku wzorca syntetycznego, poprawa ta jest bardzo znacząca.

Na podstawie zebranych obserwacji można wnioskować co następuje:

1. Co najważniejsze: algorytm detekcji pszczół oparty na automacie stanowym jest lepszy od tego z korelacją krzyżową – odrzucając tym samym odwrotną hipotezę z sekcji 3.3.2;
2. Zastosowanie odpowiednio skonfigurowanego progowania adaptacyjnego zawsze zapewnia lepszą skuteczność algorytmu detekcji;
3. Wykorzystanie większej ilości wzorców w metodzie korelacji krzyżowej nie musi gwarantować poprawy wyników.

Uzyskany najlepszy wynik, równy $E = 0.9044$, jest niestety poniżej optymistycznych oczekiwów przyjętych przed wykonaniem eksperymentów. Taki poziom dokładności uniemożliwia skuteczne szacowanie łącznego bilansu pszczół, który przemieścił się tunelami urządzenia; niemożliwa jest zatem ocena całkowitej populacji ula. Błędy generowane przez system akumulowałyby się w bardzo szybkim tempie, błyskawicznie prowadząc do błędnych wyników. Nie mniej jednak, precyza stworzonego rozwiązania jest wystarczająca do szacowania chwilowego natężenia ruchu owadów na wejściu ula, co również stanowi bardzo przydatną informację dla pszczelarzy.

Przeanalizowano znaczną część przebiegów wygenerowanych podczas testu najlepszego z algorytmów w celu wyszukania źródeł tak znacznych błędów. Za część z nich odpowiadają błędne etykiety danych, stanowią one jednak mniejszość. Więcej przeanalizowanych błędów stanowią faktyczne błędy algorytmu detekcji, występujące głównie w przypadku próby wykrycia impulsów o nietypowych kształtach – jak na rysunku 3.22. Mimo to, okazuje się, że większość pojawiających się błędów nie wynika bezpośrednio z niedostatecznej jakości algorytmu: ich źródłem, w głównej mierze, są nietypowe zachowania pszczół poruszających się wewnętrz bramki czujnika. Przebiegi odpowiadające takim sytuacjom przedstawione zostały na rysunku 3.23. Pierwszy z wykresów (3.23a) zawiera błąd spowodowany najprawdopodobniej przez pszczołę zwracającą wewnętrz bramki



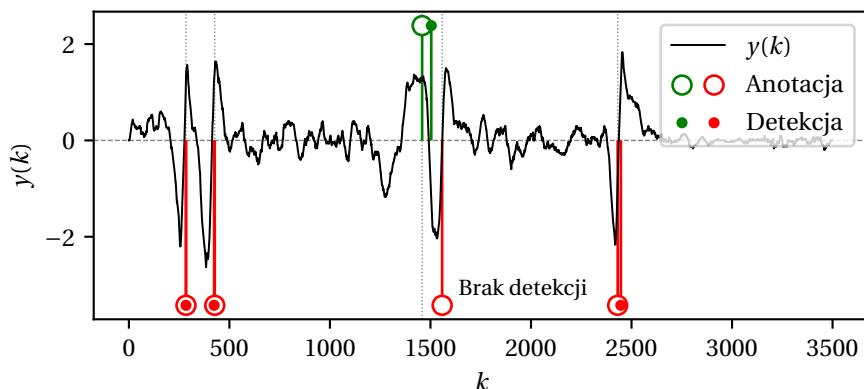
Rysunek 3.22. Przykładowy błąd popełniony przez automat stanowy. Impuls nie został wykryty ze względu na nietypowy, asymetryczny kształt.

czujnika. W wyniku tego zachowania wygenerowany został impuls o zupełnie innym kształcie, na podstawie którego niemożliwe jest wnioskowanie o sposobie ruchu owada w tunelu. Podobny przypadek występuje na wykresie 3.23b, gdzie dwie pszczoły mijające się w bramce czujnika wygenerowały jeden wspólny zdeformowany impuls. Na ostatnim rysunku (3.23c) prezentowany jest błąd detekcji wywołany grupowaniem pszczół: dwa osobniki poruszające się tuż obok siebie przeszły przez czujnik tuż po sobie, generując sygnał, na podstawie którego również bardzo trudno wnioskować o ich faktycznym ruchu. Na podstawie przeprowadzonej analizy stwierdzono, że uzyskany poziom błędu nie świadczy o niedostatecznie dobrym algorytmie detekcji, ale raczej prezentuje wadę opracowanego rozwiązania wykorzystującego pojedynczą parę czujników pojemnościowych. Najlepszym sposobem rozwiązania tego problemu wydaje się na ten moment zastosowanie większej liczby bramek w każdym z tuneli urządzenia, tak jak zrobiono w projekcie BeeCheck [19] – co jednak znacząco powiększa złożoność całości systemu. Alternatywnie, poprawę mogłoby przynieść zastosowanie innych, bardziej zaawansowanych metod detekcji impulsów, przystosowanych również do radzenia sobie ze znymi sytuacjami granicznymi – te jednak wymusiłyby wykorzystanie platformy sprzętowej o większych możliwościach.

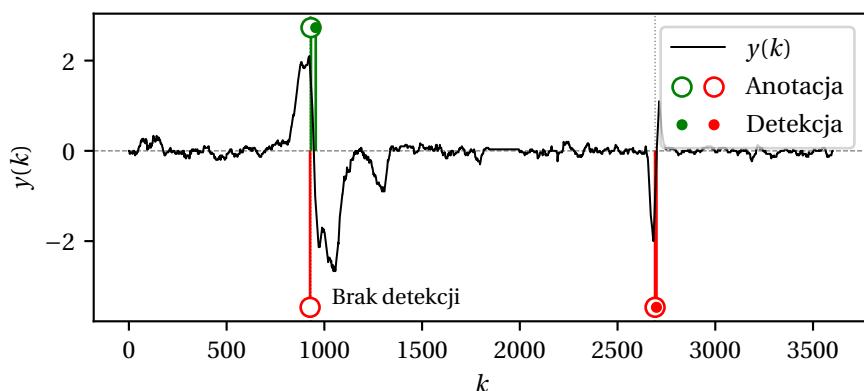
3.5. Wybór algorytmu

Na podstawie przedstawionych dotychczas wyników łatwo zdecydować, że w niniejszym projekcie należy wykorzystać metodę detekcji pszczół opierającą się na automacie stanowym z progowaniem adaptacyjnym. Dała ona lepsze wyniki w przeprowadzonych eksperymentach ($E = 0.9044$), a przy tym jest prostsza w implementacji na mikrokontrolerze niż korelacja krzyżowa: jest bowiem mniej wymagająca obliczeniowo i pamięciowo. W kolejnym rozdziale opisana została zrealizowana implementacja sprzętowa niniejszego algorytmu, a także jego test *on line*, ostatecznie weryfikujący działanie całego systemu.

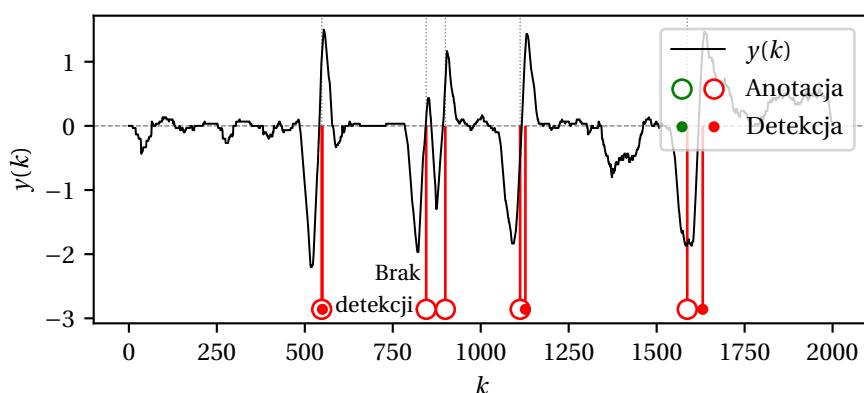
(a) Błąd spowodowany przez pszczołę zawracającą wewnątrz bramki.



(b) Błąd spowodowany przez pszczoły mijające się nawzajem w bramce.



(c) Błąd spowodowany przez pszczoły poruszające się przez tunel w grupie.



Rysunek 3.23. Przykładowe błędy popełnione przez algorytm detekcji (automat stanowy).

4. Test integracyjny

Na tym etapie realizacji pracy możliwe było przetestowanie pełnej funkcjonalności stworzonego urządzenia w finalnym teście integracyjnym. Ostatnim, co pozostało do wykonania przed jego przeprowadzeniem jest implementacja wybranego algorytmu detekcji pszczół na platformie sprzętowej ESP32-C3.

4.1. Implementacja algorytmu na sprzętie

Ograniczenia narzucane przez wykorzystywany mikrokontroler wymagają przeanalizowania etapów algorytmu pod kątem złożoności obliczeniowej i rozważenia optymalizacji implementacji niektórych z nich. Przytoczone poniżej wartości w notacji *big O* zostały wyznaczone dla wykonania danej procedury w pojedynczej pętli urządzenia – czyli dla jednej przychodzącej próbki sygnału wejściowego. Ponadto, analizowana jest podstawowa implementacja oparta bezpośrednio na przedstawionych wzorach matematycznych, bez żadnych wstępnych optymalizacji.

Automat stanowy ma złożoność obliczeniową $O(1)$. Jest niezwykle prosty i nadaje się świetnie do pracy w ograniczonych zasobach. Problematicznym krokiem algorytmu jest natomiast progowanie adaptacyjne. Opiera się ono na wyznaczaniu pewnego kwantylu z buforu ostatnich próbek – wymaga zatem posortowania próbek z całego wymaganego okna. Z tego względu, jego złożoność obliczeniowa to $O(n \log n)$. Analogicznie w przypadku kroku usuwania trendu: korzysta on z mediany próbek w buforze o innej długości – konieczne jest kolejne sortowanie ($O(n \log n)$). Dodatkowo, filtr średniej kroczącej, w podstawowej implementacji, posiada złożoność obliczeniową $O(n)$, ponieważ w każdym kroku uśrednia próbki ze swojego buforu.

Występowanie w implementacji etapów o złożoności $O(n \log n)$ samo w sobie nie stanowi problemu, dopóki algorytm wykonuje się wystarczająco szybko. Aby sprawdzić, czy konieczna jest jego optymalizacja, przeprowadzony został test czasu wykonania pojedynczej operacji usunięcia trendu z próbki. Założono okno $K = 1500$, zgodnie z najlepszą uzyskaną wcześniej konfiguracją. Algorytm uruchomiony został wiele razy, a zmierzony czas uśredniono. Wyniki dla komputera stacjonarnego oraz ESP32-C3 przedstawione zostały w tabeli 4.1. Wyznaczony czas wykonania na mikrokontrolerze (3.79 milisekundy) jest znacząco za wysoki: pomnożony przez 8 tuneli uniemożliwiłby, sam w sobie, pracę urządzenia z zakładaną częstotliwością próbkowania 100 Hz. Wynika z tego, że konieczne jest zoptymalizowanie implementacji metody usuwania trendu.

Tabela 4.1. Porównanie czasów wykonania implementacji usuwania trendu.

| Wersja implementacji | Komputer stacjonarny | ESP32-C3 |
|----------------------|----------------------|---------------|
| bez optymalizacji | 51.63 μs | 3783 μs |
| zoptymalizowana | 6.075 μs | 19.65 μs |

Proponowana optymalizacja opiera się na spostrzeżeniu, że w buforze zawsze dodawana i usuwana jest dokładnie jedna próbka – pełne sortowanie całości jest zatem nadmiarowe.

4. Test integracyjny

Do reprezentacji buforu próbek wykorzystane zostanie zbalansowane binarne drzewo poszukiwań (BST). Jest to struktura danych, która przechowuje posortowane elementy, umożliwiając ich dodawanie i usuwanie ze złożonością $O(\log n)$ [41]. Dostęp do środkowego elementu (median) odbywa się w czasie $O(n)$. W każdym wykonaniu pętli programu wykonane zostanie zatem odnalezienie ($O(\log n)$) i usunięcie z drzewa pojedynczego elementu (najstarszej próbki), dodanie najnowszej próbki, oraz pobranie mediany. Sumarycznie, rozwiążanie takie pozwoli na usuwanie trendu z próbek z całkowitą złożonością obliczeniową $O(n)$, co powinno dać znaczną poprawę czasu wykonania względem wersji bez optymalizacji. W języku C++ zbalansowane binarne drzewo poszukiwań jest dostępne jako element biblioteki standardowej, pod nazwą `std::multiset` [42]. Opracowana implementacja została przedstawiona na przedstawionym poniżej fragmencie kodu:

```
1     include/detection.h > RollingMedianDetrender
2
3 class RollingMedianDetrender {
4     public:
5         RollingMedianDetrender() {
6             // Na początku, wypełnia BST próbками z~buforu
7             for (size_t i = 0; i < kDetrendWindow; ++i) {
8                 float value = x.at(i);
9                 bst.insert(value);
10            }
11        }
12
13        float update(float newValue) {
14            float oldestValue = x.at(0);           // Aktualizacja buforu
15            x.push_back(newValue);
16
17            bst.erase(bst.find(oldestValue));    // Aktualizacja BST
18            bst.insert(newValue);
19
20            float median = *next(bst.rbegin(), kDetrendWindow/2); // Pobranie mediany
21            return newValue - median;
22        }
23
24     private:
25         CircularBuffer<kDetrendWindow> x;
26         std::multiset<float> bst;
27     };
28 }
```

Warto zwrócić uwagę, że oprócz samego drzewa binarnego, przechowywany jest również oryginalny bufor. Jest on konieczny do zachowania informacji o kolejności próbek – należy wiedzieć, jaką jest najstarsza próbka w buforze, by usunąć ją z drzewa. Powstaje dodatkowy narzut pamięciowy, ale ESP32-C3 posiada wystarczająco dużo adresów RAM, by nie stanowiło to problemu. Bufor reprezentowany jest przez strukturę `CircularBuffer`, zaimplementowaną na potrzeby niniejszej pracy. Jej celem jest przechowywanie buforu ostatnich próbek w postaci, która nie wymaga przesuwania całej zawartości jego pamięci

o jedno pole w momencie pojawiania się nowej próbki; zamiast tego przesuwany jest wskaźnik na najnowszy element. Implementację klasy CircularBuffer przedstawiono poniżej.

```

1   include/detection.h > CircularBuffer
2
3   template<size_t N>
4   class CircularBuffer {
5       public:
6           CircularBuffer() : m_head(0) {
7               for (size_t i = 0; i < N; ++i) {
8                   m_buffer[i] = 0.0f;
9               }
10      }
11
12      void push_back(float value) {
13          m_buffer[m_head] = value;
14          m_head = (m_head + 1) % N;
15      }
16
17      float at(size_t index) const {
18          size_t physicalIndex = (m_head + index) % N;
19          return m_buffer[physicalIndex];
20      }
21
22      size_t size() const { return N; }
23
24      private:
25          float m_buffer[N];
26          size_t m_head;
27      };

```

Wyniki z tabeli 4.1 pozwalają jednoznacznie potwierdzić, że optymalizacja przyspieszyła działanie algorytmu. Obliczenia zajęły w tym przypadku zaledwie $19.65 \mu s$ – wystarczająco mało, by system działał z prawidłowym czasem próbkowania.

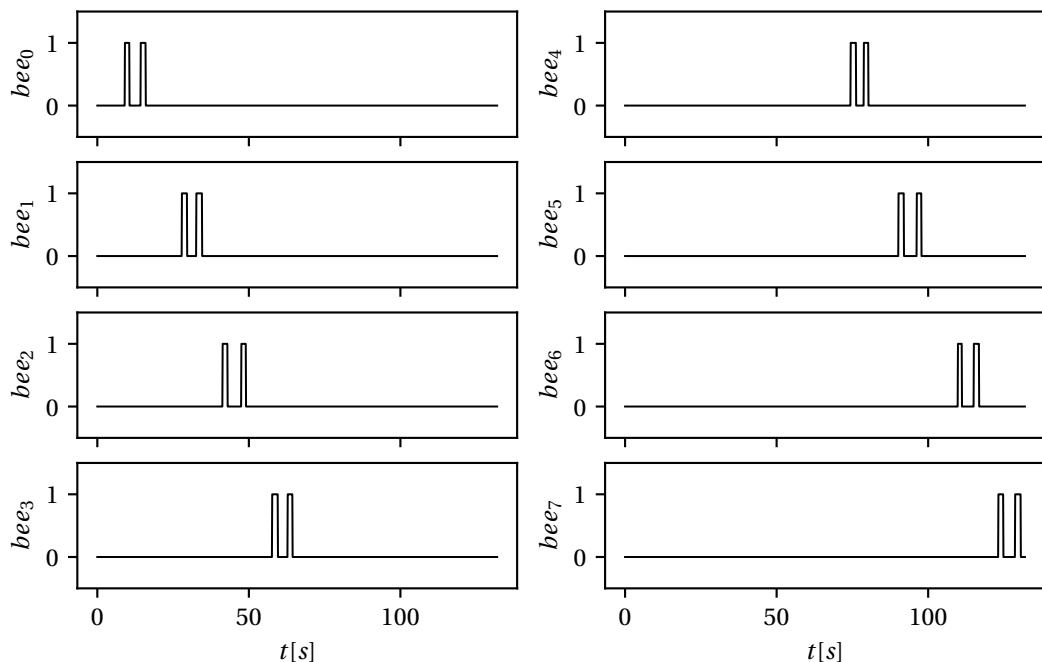
Kolejnym etapem algorytmu o złożoności $O(n \log n)$, wymagającym tym samym optymalizacji, jest obliczanie wartości dynamicznego progu $\gamma(k)$. Wąskim gardłem procesu jest wyznaczanie wartości $\hat{Q}_y(q, K, k)$ (q -tego kwantylu z buforu dłużności K). Zastosowano analogiczną optymalizację jak w przypadku mediany – jedyną różnicą jest pozycja wartości pobieranej z BST. W tym przypadku, zamiast na pozycji środkowej, interesująca próbka leży na pozycji $q \cdot K$.

Dalsza optymalizacja implementacji algorytmu detekcji nie jest konieczna, jednak dla elegancji rozwiązania zmodyfikowano kod filtra średniej kroczącej w sposób analogiczny do przedstawionych uprzednio usprawnień. Zamiast w każdej pętli programu uśredniać całe okno na nowo, całkowita wartość średniej jest jedynie aktualizowana o różnice między usuwaną, najstarszą próbką, a dodawaną – najnowszą. W ten sposób z pracy w czasie $O(n)$ usprawniono ten krok algorytmu do $O(1)$.

Łączny czas wykonania całego algorytmu detekcji na mikrokontrolerze ESP32-C3 wynosi 245 mikrosekund. Dla wszystkich ośmiu tuneli urządzenia daje to sumaryczny czas nie przekraczający 2 milisekund, wykorzystywane jest zatem jedynie niecałe 20% dostępnego czasu przy próbkowaniu 100 Hz. Dalsza optymalizacja nie jest konieczna – choć byłaby możliwa poprzez zastosowanie do reprezentacji buforu kilku drzew binarnych, balansowanych tak, by na początku jednego z nich znajdowała się zawsze interesująca próbka, np. mediana.

4.2. Test liczania pszczół

Stworzony program zawierający opracowany algorytm detekcji pszczół wgrano na skonstruowane urządzenie. W celu weryfikacji działania systemu powtórnie przeprowadzono eksperyment z kawałkiem wilgotnej gąbki na żyłce symulującym pszczołę. Przez każdy z tuneli urządzenia przeciągnięto go czterokrotnie: po dwa razy w każdym kierunku. Urządzenie połączone przez USB z komputerem stacjonarnym stale raportowało stany liczników pszczół bee_i , odpowiadające poszczególnym tunelem $i = 0 \dots 7$. Uzyskane przebiegi czasowe zostały przedstawione na rysunku 4.1. Na każdym z przedstawionych wykresów

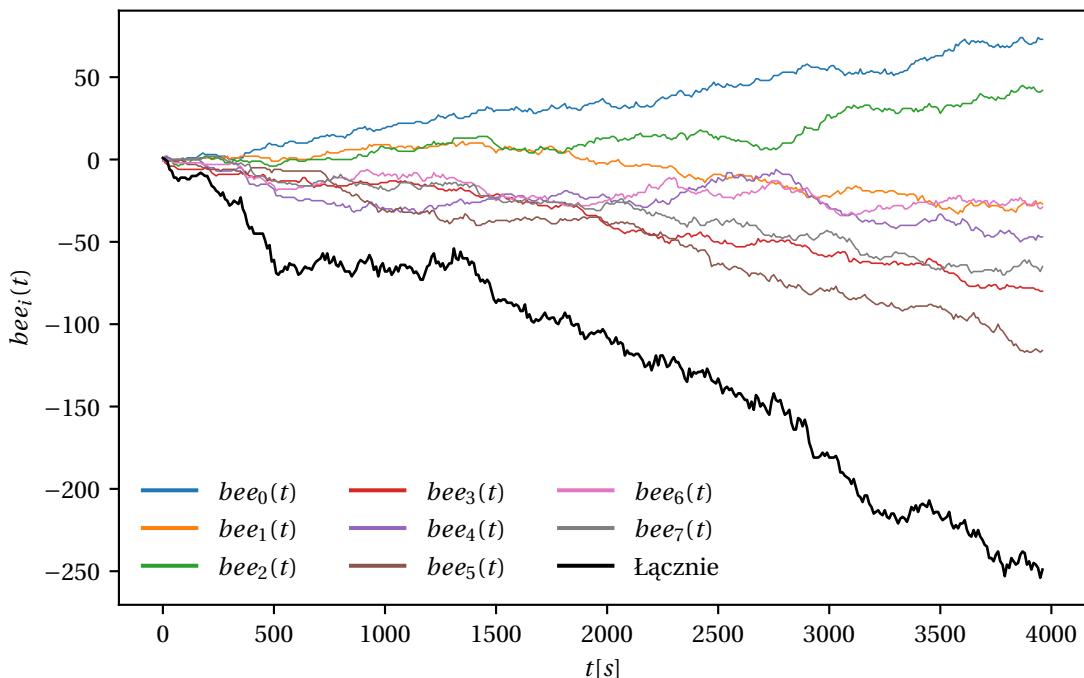


Rysunek 4.1. Przebiegi liczników bee_i , $i = 1 \dots 7$, zebrane podczas testu integracyjnego.

widac po dwa fragmenty, na których licznik przyjmuje wartość 1. Chwile skoku sygnału bee_i odpowiadają momentom przeciągania przez czujnik obiektu, i występują w prawidłowej kolejności. Można jednoznacznie wnioskować, że algorytm detekcji zadziałał prawidłowo w teście integracyjnym: wygenerował on prawidłowe momenty i kierunki wykrywanych zdarzeń. Na tym etapie pracy, stworzony system jest w pełni sprawny – można przejść do uruchamiania go w warunkach rzeczywistych.

5. Rezultaty pracy w warunkach rzeczywistych

Stworzony system zliczania pszczół ponownie uruchomiono w jednej z odwiedzonych wcześniej pasiek. Urządzenie zostało zamontowane na wylotku ula, a następnie podłączono je przewodem USB z laptopem. Odczekano około godziny, by ruch pszczół uspokoił się po zakłóceniu. Następnie uruchomiono zapis danych: urządzenie co sekundę przesyłało do komputera aktualne stany wszystkich swoich liczników, które były zapisywane do pliku CSV. System pracował przez około 1 godzinę 20 minut, po czym pobrano z niego zapisane dane. Uzyskane przebiegi czasowe przedstawione zostały na rysunku 5.1.



Rysunek 5.1. Przebiegi czasowe liczników pszczół zebrane podczas pracy testowanego urządzenia w warunkach rzeczywistych.

Na wykresach pojedynczych liczników widoczna jest mocna preferencja pszczół przy wyborze tunelu: niektórymi prawie wyłącznie wchodzą do ula, a innymi głównie wychodzą – jest to zgodne z tym, co było widoczne na ręcznie anotowanych danych z rysunku z innej pasieki (rysunek 3.6). Przez większość czasu trwania eksperymentu łączna liczba pszczół w ulu małała. Wynika to najpewniej z opuszczania ula przez pszczoły zbieraczki – odlatują one w dużych falach, by następnie w podobnym czasie powrócić do gniazda. Uzyskane rezultaty przypominają dane, które zebrano wcześniej ręcznie: zarówno pod kątem kształtu przebiegów i zależności między nimi, jak i ogólnej intensywności ruchu pszczół. Można wnioskować, że stworzony system zadziałał prawidłowo poza laboratorium, w warunkach rzeczywistych. Uruchomienie urządzenia na dłuższym horyzoncie czasowym może zapewnić pszczelarzom głębszy wgląd w zdrowie i stan kolonii, zapewniając dokładną estymację natężenia ruchu pszczół na wylotku ula. Możliwości oceny chwilowego rozmiaru populacji pozostają jednak ograniczone ze względu na występujący w systemie błąd zliczania.

6. Podsumowanie

Cel niniejszej pracy stanowiło stworzenie systemu czujników umożliwiającego zliczanie pszczół w ulu. Na podstawie przeglądu powiązanej literatury zdecydowano się wykorzystać czujniki pojemnościowe, których stosowanie zyskuje w ostatnich latach popularność. Przyjęto, że konstruowane urządzenie ma przyjąć formę szeregu tuneli z bramkami pojemnościowymi, montowanego na wejściu do ula. Realizacja zadania wymagała rozwiązań dwóch głównych problemów:

1. zaprojektowania czujników pojemnościowych i metody akwizycji ich sygnałów,
2. opracowania algorytmu umożliwiającego wykrywanie pszczół na podstawie wyjść czujników pojemnościowych.

W pierwszym etapie pracy zbudowano prototyp czujnika, bazując na konstrukcjach dostępnych w literaturze. Opracowano dla niego uproszczoną metodę akwizycji sygnału, nie wymagającą kosztownych komponentów ani kalibracji. W ramach eksperymentu z modelem pszczoły potwierdzono prawidłowe działanie prototypowej konstrukcji i metody akwizycji. Kolejnym krokiem w pracy była realizacja sprzętowa urządzenia wykorzystujące czujniki stworzone na bazie prototypu, dostosowane do gatunku pszczół *Apis mellifera*. Dobrano mikrokontroler, a następnie zaprojektowano płytę PCB z układem ośmiu czujników. Po iteracji projektu, układ działał w pełni poprawnie, co potwierdzono kolejnym badaniem laboratoryjnym na modelu pszczoły. Uzyskiwane przebiegi wyjść czujników były bardzo podobne do literaturowych, uzyskanych przez symulację elektrostatyczną.

Przed przejściem do etapu tworzenia algorytmu detekcji pszczół, konieczne było zebranie dużej ilości danych przykładowych, które miały umożliwić ocenę jakości działania kolejnych rozwiązań. W tym celu, przeprowadzone zostały pierwsze testy pracy urządzenia w warunkach rzeczywistych: odwiedzono dwie różne pasieki, w których zebrano dane generowane przez urządzenie działające z żywymi pszczołami, wraz z referencyjnym nagraniem wideo. Kilka godzin uzyskanych danych zostało poddane ręcznej anotacji, w ramach której oznaczono chwile przechodzenia pszczół przez tunele czujników.

Z obszernym zbiorzem oetykietowanych danych, można było przejść do projektu algorytmu detekcji. W ramach niniejszej pracy zaproponowane zostały dwa rozwiązania, które wykazywały duży potencjał, zachowując przy tym niską złożoność obliczeniową: wykrywanie sekwencji progowanego sygnału automatem stanowym, oraz korelacja krzyżowa z wzorcem wykrywanego impulsu. Liczne konfiguracje obu algorytmów zostały uruchomione dla zbioru danych przykładowych, a ich wyjścia porównano z ręcznie oznaczonymi etykietami. Najwyższą zgodność osiągnął algorytm oparty na automacie stanowym. Wybraną metodę detekcji zaimplementowano dla platformy sprzętowej urządzenia, po uprzedniej koniecznej optymalizacji najważniejszych fragmentów. Skuteczność opracowanego oprogramowania dla nowych danych potwierdzono w ostatnim eksperymencie laboratoryjnym. Stworzony system ponownie uruchomiono na prawdziwym ulu – tym razem z jego pełną funkcjonalnością automatycznego zliczania pszczół. Zebrane w ten sposób dane zostały zawarte i omówione w niniejszej pracy.

Przeprowadzone badania pozwoliły na wyciągnięcie następujących wniosków:

1. Czujniki pojemnościowe prezentowanego typu działają prawidłowo, a w połączeniu z opracowaną metodą akwizycji i przetwarzaniem generują dobryj jakości sygnał wyjściowy. Impulsy powstające przy ruchu pszczół w czujniku są wyraźnie widoczne i bez problemów mogą zostać wykryte automatycznie. Zastosowanie w roli sygnału wyjściowego różnicy między pojemnościami dwóch kondensatorów zapewnia rozwiązańu odporność na zmienne warunki środowiska, niestacjonarność systemu, a także niweluje konieczność kalibracji. Niezwykła prostota proponowanego układu akwizycji znaczaco obniża koszt wykonania urządzenia;
2. Montaż urządzenia na wylotku ula wydaje się mieć znacznie mniejszy negatywny wpływ na zachowanie pszczół, niż się spodziewano. Ruch owadów przez tunele rozpoczęta się praktycznie od razu; pszczoły nie próbowaly także omijać urządzenia i przedostawać się do ula bokiem. Ograniczenie ruchu powietrza też nie wydało się być problemem – nawet przy dłuższej pracy systemu nie pojawiało się zbyt wiele osobników wentylujących wejście ula;
3. Opracowany algorytm detekcji działa poprawnie, kiedy pszczoły w tunelu poruszają się typowo. Zdarzają się błędy wynikające z niestandardowych zachowań pszczół, takich jak: grupowanie w tunelu, zatrzymywanie w czujniku. Zachowania te okazują się występować częściej niż zakładano, przez co dokładność systemu jest niższa, niż oczekiwano. W aktualnej wersji, urządzenie nie pozwala na skuteczne szacowanie aktualnego stanu populacji ula. Zapewnia natomiast możliwość oceny chwilowego natężenia ruchu pszczół na jego wejściu, co również stanowi ważną metrykę świadczącą o zdrowiu i sile kolonii.

Udowodniono, że opracowana technika automatycznego zliczania pszczół działa poprawnie. Znaczaco obniżono koszt budowy urządzenia względem innych prac, co może pozwolić na stosowanie systemu na większą skalę również w mniejszych pasiekach. Kontynuacja pracy nad stworzonym rozwiązaniem da szanse ograniczyć ilość występujących błędów. Możliwe kierunki rozwoju to:

1. Zastosowanie większej liczby bramek pojemnościowych w każdym tunelu urządzenia – dodatkowe dane mogą pozwolić lepiej rozróżnić atypowe zachowania pszczół;
2. Wykorzystanie bardziej zaawansowanych metod detekcji pszczelich impulsów, które byłyby zdolne do interpretacji przebiegów z anomaliami;
3. Opracowanie metody szacowania całkowitej populacji ula opartej na fuzji danych z modelem – rozszerzenie analizy sygnałów o model probabilistyczny uzależniający natężenie ruchu pszczół od ich liczby w gnieździe. Połączenie danych z dwóch źródeł: prostego zliczania wejść i wyjść, wraz z częstotliwością ruchu na wlotku mogłoby zapewnić estymację stanu populacji znacznie mniej podatną na akumulację występujących w systemie błędnych zliczeń.

Podziękowania

Dziękuję dr. Jakubowi Gąbce za podzielenie się ekspercką wiedzą dotyczącą pszczół, oraz p. Andrzejowi Bielackiemu i p. Wojciechowi Sokołowi za udostępnienie swoich uli do eksperymentów.

Bibliografia

- [1] H. Hadjur, D. Ammar i L. Lefèvre, „Toward an intelligent and efficient beehive: A survey of precision beekeeping systems and services”, *Computers and Electronics in Agriculture*, t. 192, s. 106 604, 2022. DOI: 10.1016/j.compag.2021.106604
- [2] P. P. Danieli, N. F. Addeo, F. Lazzari, F. Manganello i F. Bovera, „Precision Beekeeping Systems: State of the Art, Pros and Cons, and Their Application as Tools for Advancing the Beekeeping Sector”, *Animals*, t. 14, nr 1, s. 70, 2024. DOI: 10.3390/ani14010070
- [3] W. G. Meikle i N. Holst, „Application of continuous monitoring of honeybee colonies”, *Apidologie*, t. 46, nr 1, s. 10–22, czerwiec 2014. DOI: 10.1007/s13592-014-0298-x
- [4] A. R. McLellan, „Honeybee Colony Weight as an Index of Honey Production and Nectar Flow: A Critical Evaluation”, *The Journal of Applied Ecology*, t. 14, nr 2, s. 401, sierpień 1977. DOI: 10.2307/2402553
- [5] S. A. Kolmes i Y. Sam, „Foraging Rates and Hive Contents During the Establishment of Honeybee Colonies (*Apis MelliferaL.*)”, *Journal of Apicultural Research*, t. 29, nr 3, s. 126–131, styczeń 1990. DOI: 10.1080/00218839.1990.11101208
- [6] S. A. CORBET i in., „Temperature and the pollinating activity of social bees”, *Ecological Entomology*, t. 18, nr 1, s. 17–30, luty 1993. DOI: 10.1111/j.1365-2311.1993.tb01075.x
- [7] R. Odemer, „Approaches, challenges and recent advances in automated bee counting devices: A review”, *Annals of Applied Biology*, t. 180, nr 1, s. 73–89, wrzesień 2021. DOI: 10.1111/aab.12727
- [8] A. E. Lundie, „The flight activities of the honeybee”, w: Washington, D.C, U.S. Dept. of Agriculture, 1925.
- [9] E. H. Erickson, H. H. Miller i D. J. Sikkema, „A Method of Separating and Monitoring Honeybee Flight Activity at the Hive Entrance”, *Journal of Apicultural Research*, t. 14, nr 3–4, s. 119–125, styczeń 1975. DOI: 10.1080/00218839.1975.11099814
- [10] U. Pešović, S. Randić i Z. Stamenkovic, „Design and Implementation of Hardware Platform for Monitoring Honeybee Activity”, czerwiec 2017.
- [11] J.-A. Jiang i in., „A WSN-based automatic monitoring system for the foraging behavior of honey bees and environmental factors of beehives”, *Computers and Electronics in Agriculture*, t. 123, s. 304–318, kwiecień 2016. DOI: 10.1016/j.compag.2016.03.003
- [12] A. Souza Cunha, J. Rose, J. Prior, H. Aumann, N. Emanetoglu i F. Drummond, „A novel non-invasive radar to monitor honey bee colony health”, *Computers and Electronics in Agriculture*, t. 170, s. 105 241, marzec 2020. DOI: 10.1016/j.compag.2020.105241
- [13] beehivemonitoring.com. „Bee counter”. URL: <https://beehivemonitoring.com/product/bee-counter/>
- [14] F. Tausch, K. Schmidt i M. Diehl, „Current achievements and future developments of a novel AI based visual monitoring of beehives in ecotoxicology and for the monitoring of landscape structures”, luty 2020. DOI: 10.1101/2020.02.04.933580

6. Bibliografia

- [15] S. Bilik i in., „Visual Diagnosis of the Varroa Destructor Parasitic Mite in Honeybees Using Object Detector Techniques”, *Sensors*, t. 21, nr 8, s. 2764, kwiecień 2021. DOI: 10.3390/s21082764
- [16] S. Dunker i in., „Pollen analysis using multispectral imaging flow cytometry and deep learning”, *New Phytologist*, t. 229, nr 1, s. 593–606, wrzesień 2021. DOI: 10.1111/nph.16882
- [17] J. M. Campbell, D. C. Dahn i D. A. J. Ryan, „Capacitance-based sensor for monitoring bees passing through a tunnel”, *Measurement Science and Technology*, t. 16, nr 12, s. 2503–2510, listopad 2005. DOI: 10.1088/0957-0233/16/12/015
- [18] P. Perrault i M. Teachman, „Bee Counters: Measuring a nest's occupation by its capacitance [Resources_{HandsOn}]”, *IEEE Spectrum*, t. 53, nr 2, s. 20–21, luty 2016. DOI: 10.1109/mspec.2016.7419791
- [19] S. Bermig, R. Odemer, A. J. Gombert, M. Frommberger, R. Rosenquist i J. Pistorius, „Experimental validation of an electronic counting device to determine flight activity of honey bees (*Apis mellifera L.*)”, en, *Journal of Cultivated Plants*, Bd. 72 Nr. 5 (2020): Themenheft Bienenschutz, 2020. DOI: 10.5073/JFK.2020.05.03
- [20] Mathematastic. „Bee++”, dostęp 26 kwietnia 2025. URL: <https://mathematastic.com/bee-plus-plus/>
- [21] M. Kozielski, K. Wosińska i P. Duda. „e-Fizyka, Tom III, Rozdział 1. Elektrostatyka”, dostęp 26 kwietnia 2025. URL: <https://ilf.fizyka.pw.edu.pl/podrecznik/3/1/7>
- [22] L. G. Hector i H. L. Schultz, „The Dielectric Constant of Air at Radiofrequencies”, *Physics*, t. 7, nr 4, s. 133–136, kwiecień 1936, ISSN: 2163-5102. DOI: 10.1063/1.1745374
- [23] D. G. Archer i P. Wang, „The Dielectric Constant of Water and Debye-Hückel Limiting Law Slopes”, *Journal of Physical and Chemical Reference Data*, t. 19, nr 2, s. 371–411, marzec 1990, ISSN: 1529-7845. DOI: 10.1063/1.555853
- [24] Analog Devices, *AD7745/AD7746: 24-Bit Capacitance-to-Digital Converter with Temperature Sensor*, Datasheet, Rev. 0, Analog Devices, Norwood, MA, USA, kwiecień 2005. URL: <https://www.analog.com/>
- [25] O. Kształcenia na Odległość Politechniki Warszawskiej. „eSezam, Podręcznik. 2. Stany nieustalone w obwodach RL i RC”, dostęp 26 kwietnia 2025. URL: <https://esezam.okno.edu.pl/mod/book/tool/print/index.php?id=26&chapterid=371>
- [26] *ESP32-C3 Series Datasheet*, wersja 2.2, Ultra-Low-Power SoC with RISC-V Single-Core CPU, Espressif Systems, 2025.
- [27] „Microcontroller I/O ADC Benchmarks”. Arduino.CC Forum, dostęp 26 kwietnia 2025. URL: <https://forum.arduino.cc/t/microcontroller-i-o-adc-benchmarks/315304>
- [28] M. Banzi, *Getting Started with Arduino: The Open Source Electronics Prototyping Platform*. Sebastopol, CA: O'Reilly Media, 2011.
- [29] PlatformIO. „PlatformIO: Your Gateway to Embedded Software Development Excellence”, PlatformIO, dostęp 6 listopada 2025. URL: <https://platformio.org/>

- [30] P. Skubida, „Systemy i typy uli: Ul wielkopolski, ul Ostrowskiej - ule o małej ramce”, *Pszczelarstwo*, t. 58, nr 09, s. 28–30, 2007.
- [31] SparkFun Electronics, *Pro Micro - 5V/16MHz (DEV-12640) Datasheet*, Pinout and Technical Specifications for the ATmega32U4 Board, ProMicro16MHzv1, SparkFun Electronics, 2014. URL: <https://www.sparkfun.com/products/12640>
- [32] *STM32F401xB/STM32F401xC Arm® Cortex®-M4 32-bit MCU+FPU, 105 DMIPS, 256KB Flash / 64KB RAM, 11 TIMs, 1 ADC, 11 comm. interfaces Datasheet*, Datasheet production data, DS9716 Revision 11, STMicroelectronics, kwiecień 2019.
- [33] Maker go, *ESP32-C3 SuperMini Schematic*, Circuit Diagram/Schematic for IoT Development Board, Accessed: November 13, 2025. URL: <https://make.net.za/wp-content/datasheets/ESP32-C3%20SuperMini%20Schematic.pdf>
- [34] OurPCB. „Kapacytancje pasożytnicze: Niechciane i nieuniknione ładunki pomiędzy przewodnikami”, OurPCB, dostęp 13 listopada 2025. URL: <https://ourpcb.pl/kapacytancje-pasozytnicze.html>
- [35] JLCPCB. „PCB Manufacturing & Assembly Capabilities”, JLCPCB, dostęp 13 listopada 2025. URL: <https://jlcpcb.com/capabilities/pcb-capabilities>
- [36] Espressif Systems, *GPIO & RTC GPIO — ESP-IDF Programming Guide v4.3 Documentation*, Espressif Systems, 2021.
- [37] E. Peña i M. G. Legaspi, „UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter”, *Analog Dialogue*, t. 54, nr 4, s. 1–5, grudzień 2020, Vol 54, No 4-December 2020.
- [38] V. Zverovich i other contributors, *fmt: A Modern Formatting Library*, <https://github.com/fmtlib/fmt>, Accessed: 2025-12-4, 2024.
- [39] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1997, ISBN: 0-9660176-3-3. URL: <https://www.dspguide.com/ch15.htm>
- [40] G. E. P. Box, G. M. Jenkins, G. C. Reinsel i G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th. Springer, 2015, s. 58, ISBN: 978-1-118-67502-1.
- [41] T. H. Cormen, C. E. Leiserson, R. L. Rivest i C. Stein, *Wprowadzenie do algorytmów*, VII (druk). Warszawa: Wydawnictwo Naukowe PWN, 2019, ISBN: 9788301205362.
- [42] cplusplus.com. „`std::multiset` - C++ Reference”. Dostęp online, dostęp 13 stycznia 2026. URL: <https://cplusplus.com/reference/set/multiset/>

Wykaz symboli i skrótów

LED – Light-Emitting Diode
PCB – Printed Circuit Board
GPIO – General-Purpose Input/Output
EXTI – External Interrupt
USB – Universal Serial Bus
UART – Universal Asynchronous Receiver/Transmitter
SPI – Serial Peripheral Interface
I²C – Inter-Integrated Circuit
SRAM – Static Random-Access Memory
PC – Personal Computer
CSV – Comma-Separated Values
EEPROM – Electrically Erasable Programmable Read-Only Memory
FSM – Finite-State Machine
BST – Binary Search Tree
RAM – Random-Access Memory

Spis rysunków

| | | |
|------|---|----|
| 2.1 | Schemat układu kondensatora z poruszającą się ruchem jednostajnym pszczołą oraz odpowiadający przebieg pojemności elektrycznej C_1 | 10 |
| 2.2 | Schemat i przebieg $C_1 - C_2$ dla układu dwóch kondensatorów. | 10 |
| 2.3 | Schemat i przebieg $C_1 - C_2$ dla układu dwóch kondensatorów – pszczoła porusza się w odwrotnym kierunku. | 10 |
| 2.4 | Konstrukcja czujnika opartego na elektrodach pierścieniowych. | 11 |
| 2.5 | Przekrój poprzeczny czujnika oraz linie ekwipotencjalne – symulacja przeprowadzona przez Campbell wraz z zespołem. [17] | 12 |
| 2.6 | Impulsy wygenerowane przez pszczołę przechodzącą przez czujnik. Wyniki symulacji. Różnica pojemności kondensatorów była przetwarzana na napięcie przez liniowy przetwornik. (a) – duża pszczoła opuszczająca ul; (b) – mała pszczoła wchodząca do ula. [17] | 12 |
| 2.7 | Ładowanie kondensatora: schemat układu i przebiegi napięć. | 13 |
| 2.8 | Schemat układu pomiarowego. | 14 |
| 2.9 | Prototypowy czujnik pojemnościowy. | 16 |
| 2.10 | Wymiary prototypowego tunelu – oznaczenia dla tabeli 2.1. | 17 |
| 2.11 | Schemat układu elektronicznego prototypowego tunelu. | 17 |
| 2.12 | Schemat przeprowadzonego eksperymentu. | 20 |
| 2.13 | Przebiegi wyjściowe czujnika uzyskane podczas przeprowadzonych testów. . . | 21 |
| 2.14 | Wizualizacja konstrukcji i sposobu mocowania urządzenia do ula. | 22 |
| 2.15 | Rozmieszczenie tuneli w czujniku – oznaczenia dla tabeli 2.2 | 23 |

| | |
|---|----|
| 2.16 Schemat systemu z dwoma mikrokontrolerami do obsługi ośmiu bramek. | 25 |
| 2.17 Schemat układu elektronicznego. | 26 |
| 2.18 Projekt płytki PCB. | 27 |
| 2.19 Montaż tunelu dla pszczół do płytki PCB. | 27 |
| 2.20 Gotowe płytki PCB. | 28 |
| 2.21 Połączenia pomiędzy płytami PCB umożliwiające współpracę podukładów. . | 32 |
| 2.22 Ramka UART reprezentująca pojedynczy pomiar bramki. | 33 |
| 2.23 Algorytmy pracy mikrokontrolera głównego i pomocniczego. | 34 |
| 2.24 Odbiór ramek uart – schemat automatu stanowego. | 35 |
| 2.25 Przebieg sygnału wyjściowego bramki podczas przeprowadzonego testu. . . | 38 |
| 2.26 Przebiegi sygnałów wyjściowych bramek x_0-x_3 podczas testu działania urządzenia. | 39 |
| 2.27 Przebiegi sygnałów wyjściowych bramek x_4-x_7 podczas testu działania urządzenia. | 40 |
| 2.28 Schemat złożenia obudowy urządzenia | 41 |
| 2.29 Zdjęcia z kolejnych etapów montażu urządzenia w obudowie | 42 |
| 3.1 Przebieg sygnału $x_0(t)$ (wyjścia pierwszego czujnika pojemnościowego) w pierwszych kilkudziesięciu sekundach trwania eksperymentu. Przebieg filtrowany został wygenerowany poprzez nałożenie na dane oryginalne filtra średniej kroczącej o długości okna równej 50. | 44 |
| 3.2 Zdjęcie jednego z eksperymentów. Widoczne jest urządzenie zamontowane na wylocie ula, oraz kamera nagrywająca jego wylot. Znajdujący się na obrazie fioletowy przewód łączy czujnik z komputerem zapisującym dane. | 45 |
| 3.3 Zdjęcie prezentujące montaż urządzenia podczas drugiego eksperymentu. Jest ono przybite do dennicy – pozostała część ula nastawia się na widoczny na obrazie element. | 45 |
| 3.4 Zrzut ekranu z programu do ręcznej anotacji danych z eksperymentu. | 46 |
| 3.5 Fragment przebiegu sygnału x_0 zestawiony z ręcznie dodaną anotacją. | 47 |
| 3.6 Bilans pszczół w poszczególnych tunelach podczas jednego z eksperymentów. | 48 |
| 3.7 Porównanie sygnału oryginalnego z przefiltrowanym filtrem średniej kroczącej. | 50 |
| 3.8 Porównanie działania filtrów o długościach okna: (a) $N = 5$, (b) $N = 50$, (c) $N = 500$ | 51 |
| 3.9 Usuwanie trendu z wykorzystaniem sygnału różnicowego. Jako $x(k)$ oznaczony jest sygnał oryginalny obarczony trendem, natomiast jako $\nabla x(k)$ – wyznaczony na jego podstawie sygnał różnicowy. | 52 |
| 3.10 Metoda usuwania trendu z wykorzystaniem mediany sygnału – przebiegi testowe. | 54 |
| 3.11 Wariant usuwania trendu wykorzystujący średnią sygnału – przebiegi testowe. | 54 |
| 3.12 Ogólna struktura algorytmu detekcji. | 55 |
| 3.13 Struktura automatu stanowego. Warunki przejścia zapisano przy początkach strzałek, natomiast akcje przejścia przy końcach strzałek. Dodatkowo, akcje oznaczono pogrubieniem. | 55 |
| 3.14 Fragment przebiegów z testu automatu stanowego z progowaniem statycznym. | 58 |

| | |
|--|----|
| 3.15 Fragment przebiegów z testu automatu stanowego z progowaniem adaptacyjnym. | 58 |
| 3.16 Fragment przebiegów z testu automatu stanowego z progowaniem statycznym, zawierający słabej jakości sygnał wejściowy algorytmu. | 59 |
| 3.17 Fragment przebiegów z testu automatu stanowego z progowaniem adaptacyjnym, zawierający słabej jakości sygnał wejściowy algorytmu. | 59 |
| 3.18 Przykładowe wzorce impulsu generowanego przez pszczołę; a – wzorzec wybrany z danych eksperymentalnych, b – wzorzec syntetyczny (fragment sinusoidy). | 60 |
| 3.19 Fragment przebiegów z testu korelacji krzyżowej z progowaniem statycznym. | 62 |
| 3.20 Fragment przebiegów z testu korelacji krzyżowej z progowaniem adaptacyjnym. | 62 |
| 3.21 Porównanie działania wariantów algorytmu korelacji krzyżowej na fragmencie danych wejściowych niskiej jakości. | 63 |
| 3.22 Przykładowy błąd popełniony przez automat stanowy. Impuls nie został wykryty ze względu na nietypowy, asymetryczny kształt. | 67 |
| 3.23 Przykładowe błędy popełnione przez algorytm detekcji (automat stanowy). | 68 |
| 4.1 Przebiegi liczników bee_i , $i = 1 \dots 7$, zebrane podczas testu integracyjnego. | 72 |
| 5.1 Przebiegi czasowe liczników pszczół zebrane podczas pracy testowanego urządzenia w warunkach rzeczywistych. | 73 |

Spis tabel

| | |
|--|----|
| 2.1 Wymiary prototypowego tunelu. | 17 |
| 2.2 Rozmieszczenie tuneli w czujniku | 23 |
| 2.3 Porównanie mikrokontrolerów | 24 |
| 3.1 Zbiór danych przykładowych – podsumowanie eksperymentów | 44 |
| 3.2 Parametry testowanych automatów stanowych. | 56 |
| 3.3 Parametry testowanych algorytmów korelacji krzyżowej. | 61 |
| 3.4 Wyniki porównania algorytmów detekcji. | 66 |
| 4.1 Porównanie czasów wykonania implementacji usuwania trendu. | 69 |