

Dokumentacja Techniczna Systemu Mechatronicznego „GESTURE BOT”

Danylo Chetvertakov, Nikita Tolstoi, Volodymr Vyshnovetskyi,

Zakhar Semiankevich, Yehor Maksymenko

1. Cel i zakres dokumentu

Niniejszy dokument przedstawia kompleksową specyfikację projektową, materiałową oraz programistyczną robota równoległego (tzw. Parallel Robot). Dokument obejmuje:

- **charakterystykę konstrukcyjną,**
- **parametry materiałowe,**
- **specyfikację układów wykonawczych i sensorycznych,**
- **architekturę sterowania,**
- **opis implementacji oprogramowania sterującego w Pythonie,**
- **komunikację z modułami peryferyjnymi,**
- **algorytmy kinematyczne i sterowania,**
- **fragmenty kodu operacyjnego.**

Robot docelowo wykonuje zadania lokomocji (chodzenie w trybie pseudo-kroczącym), manipulacji (podnoszenie ramienia/uchwyty), interakcji z użytkownikiem (wyświetlacz LCD/OLED), oraz autonomicznego reagowania na czujniki.

2. Charakterystyka ogólna systemu PARALLEL ROBOT

2.1. Opis ogólny

PARALLEL ROBOT jest hybrydowym urządzeniem mechatronicznym wykorzystującym równoległą architekturę napędową — zestaw siłowników połączonych w konfiguracji Delta/Stewart lite, umożliwiające precyzyjne sterowanie orientacją platformy mobilnej. Jego konstrukcja została zoptymalizowana pod kątem:

- **wysokiej dynamiki,**
- **minimalizacji masy własnej,**
- **maksymalnej sztywności,**
- **modularności podzespołów,**
- **niskich oporów kinematycznych.**

3. Materiały i komponenty

3.1. Struktura nośna

Materiały wymagane dla elementów konstrukcyjnych:

Serwo — 2 szt.

Sterownik do serw — 1 szt.

Adapter do serw — 1 szt.

Wyświetlacz — 1 szt.

Łożysko kulowe w obudowie KP08 — 8 szt.

Profile aluminiowe Item — ilość do ustalenia

Filament PET-G 1,75 mm — 1 szt.

Pogo pin P125-D — ilość niepodana

Raspberry Pi Zero 2 W — 1 szt.

Wyświetlacz — 1 szt.

Mosfety — 7 szt.

Zasilacz — 1 szt.

Przetwornica step-up — 7 szt.

Buzzer — 7 szt.

Czytnik RFID — 1 szt.

Listwa do LED — 3 szt.

Taśma LED — 1 szt.

Klawiatura bezprzewodowa — 1 szt.

ESP S3 mini — 7 szt.

MPU6050 — 7 szt.

Moduł ładowania Li-Po — 7 szt.

Akumulator — 7 szt.

Koszyk na akumulator — 7 szt.

Pierścień LED — 1 szt.

Rura pleksi 1 m — 3 szt.

Elektrozamek — 7 szt.

Stojak — 1 szt.

Odroid H4 Ultra N305 — 1 szt.

Pamięć do Odroid H4 Ultra — 1 szt.

Kabel HDMI – HDMI 3 m — 1 szt.

Karta pamięci microSD 128 GB — 1 szt.

Zasilacz impulsowy 12 V / 2 A — 1 szt.

Przewód USB microUSB 1,8 m — 1 szt.

Przewód USB microUSB 1 m — 1 szt.

Patchcord kat.5e UTP 1 m — 1 szt.

Kabel USB type C – HDMI 1,8 m — 1 szt.

Obudowa do Odroid H4 typ 2 — 1 szt.

4. Układy wykonawcze

4.1. Serwomechanizmy

Zastosowane serwomechanizmy magistrali szeregowej z enkoderem magnetycznym:

- **Serwa 360° z komunikacją UART**
- Dwukierunkowa komunikacja
- Odczyt pozycji, prędkości i stanu serwa

Parametry:

- Moment nominalny: ok. 30 kg·cm
- Praca ciągła 360°
- Wysoka precyzja dzięki enkoderowi magnetycznemu

5. Układy sensoryczne

5.1. Czujniki pozycji

- **Enkodery magnetyczne zintegrowane w serwomechanizmach**
Odczyt pozycji realizowany bezpośrednio przez magistralę szeregową serw.
- **IMU 6-osiowa: MPU6050**
Pomiar przyspieszeń liniowych oraz prędkości kątowych, wykorzystywany do stabilizacji i detekcji ruchu.

5.2. Czujniki kontaktowe

- **Brak dedykowanych czujników kontaktowych**
Detekcja stanu może być realizowana pośrednio:
 - na podstawie danych z IMU,
 - przez analizę obciążenia serw,
 - lub przez logikę sterowania ruchu.

5.3. Moduł wizualny

- **Wyświetlacz LCD HDMI z panelem dotykowym**
 - **Komunikacja: HDMI + USB (dotyk)**
Wyświetlacz obsługiwany bezpośrednio przez Raspberry Pi / Odroid jako standardowy ekran systemowy.
-

6. Architektura oprogramowania

6.1. Ogólny diagram modułów

Oprogramowanie sterujące, napisane w Pythonie, składa się z następujących warstw:

1. **HAL (Hardware Abstraction Layer)**
Obsługa serw, IMU MPU6050, RFID, LED, buzzerów oraz GPIO.
2. **Kinematyka**
Obliczenia kinematyki prostej i odwrotnej.
3. **Sterowanie ruchem**
Generacja trajektorii i synchronizacja osi.
4. **Warstwa decyzyjna (Behavior Layer)**
Logika ruchu, sekwencje, reakcje na dane z czujników.
5. **UI / UX Layer**
Interfejs graficzny wyświetlany na ekranie HDMI, obsługa klawiatury i dotyku.

6.2. Wybrane biblioteki Python

- **pyserial** – komunikacja z serwami i mikrokontrolerami

- **numpy** – obliczenia numeryczne i macierze
 - **time, math** – operacje pomocnicze
 - **smbus / I2Cdev** – komunikacja z MPU6050
 - **RPi.GPIO / gpiozero** – obsługa wejść/wyjść
 - **biblioteki systemowe Linux** – obsługa wyświetlacza HDMI i urządzeń USB
-

7. Kinematyka robota równoległego

7.1. Kinematyka odwrotna (IK)

Oparta o równania transformacji: Введіть тут рівняння.

8. Zrobienia wykresow na monitorze

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout
from PyQt5.QtGui import QPainter, QColor, QPolygon, QPen
from PyQt5.QtCore import Qt, QPoint
```

```
class Shape:
    def init(self, x, y, size, color, shape_type):
        self.x = x
        self.y = y
        self.size = size
        self.color = color
        self.shape_type = shape_type
        self.selected = False
        self.angle = 0 # поточний кут обертання (в градусах)
```

```
class DrawingWidget(QWidget):
    def init(self):
        super().init()

        # Початкові фігури
        self.shapes = [
            Shape(150, 150, 50, QColor(0, 120, 255), "circle"),
            Shape(400, 150, 50, QColor(0, 255, 100), "square"),
            Shape(250, 320, 60, QColor(255, 180, 0), "triangle")
        ]

        self.active_shape = None
        self.last_x = None # попередня X-позиція миші для обертання

    def contains(self, shape, px, py):
        return (shape.x - shape.size < px < shape.x + shape.size and
                shape.y - shape.size < py < shape.y + shape.size)

    def mousePressEvent(self, event):
        x, y = event.x(), event.y()

        self.active_shape = None
        for s in self.shapes:
            s.selected = False

        for shape in reversed(self.shapes):
            if self.contains(shape, x, y):
                shape.selected = True
```

```

        self.active_shape = shape

        self.last_x = x

        print(f"[SELECT] {shape.shape_type} at X={shape.x}, Y={shape.y},
angle={shape.angle:.1f}")

        break

    self.update()

def mouseMoveEvent(self, event):

    if not (event.buttons() & Qt.LeftButton):

        return

    if self.active_shape is None:

        return

    x = event.x()

    if self.last_x is None:

        self.last_x = x

        return

    dx = x - self.last_x # ruch prawo/lewo

    self.last_x = x

    # 1 pixel = 1 stopien

    self.active_shape.angle += dx

    print(f"[ROTATE] {self.active_shape.shape_type}:
angle={self.active_shape.angle:.1f} (dx={dx})")

```

```
self.update()
```

```
def mouseReleaseEvent(self, event):
```

```
    self.last_x = None
```

```
    self.active_shape = None
```

```
def paintEvent(self, event):
```

```
    painter = QPainter(self)
```

```
    for shape in self.shapes:
```

```
        painter.save()
```

```
        painter.translate(shape.x, shape.y)
```

```
        painter.rotate(shape.angle)
```

```
        painter.setBrush(shape.color)
```

```
    if shape.shape_type == "circle":
```

```
        painter.drawEllipse(-shape.size, -shape.size,  
                             shape.size * 2, shape.size * 2)
```

```
    elif shape.shape_type == "square":
```

```
        painter.drawRect(-shape.size, -shape.size,  
                          shape.size * 2, shape.size * 2)
```

```
    elif shape.shape_type == "triangle":
```

```
        half = shape.size
```

```
        polygon = QPolygon([
```

```
            QPoint(0, -half),
```

```
            QPoint(-half, half),
```

```
            QPoint(half, half)
```

```
        ])
```



```
painter.drawPolygon(polygon)
```

```
painter.restore()
```

```
if shape.selected:
```

```
    painter.setPen(QPen(QColor(255, 0, 0), 4))
```

```
    painter.setBrush(Qt.NoBrush)
```

```
    painter.drawEllipse(shape.x - shape.size - 10,
```

```
                        shape.y - shape.size - 10,
```

```
                        (shape.size + 10) * 2,
```

```
                        (shape.size + 10) * 2)
```

```
    painter.setPen(Qt.NoPen)
```

```
class MainWindow(QWidget):
```

```
    def init(self):
```

```
        super().init()
```

```
        self.setWindowTitle("Touch Shape Rotation (Jetson Nano)")
```

```
        self.resize(800, 480)
```

9. Przemieszczanie obiektu

1. Przemieszczanie Obiektu (Move / Drag)

Gest służący do przeciągania obiektu w nowe miejsce. Składa się z dwóch wyraźnych faz oddzielonych pauzą:

Faza 1: Wybór (Select):

- **Działanie:** Nacisnąć na obiekt i puścić (tap).
- **Funkcja:** Rejestruje rozpoczęcie zaznaczania/przemieszczania.

Faza 2: Chwyć i Przeciąganie (Grab & Drag):

- **Działanie:** Ponownie nacisnąć, przytrzymać dłużej i rozpocząć prowadzenie rysika po linii prostej do docelowej współrzędnej.

- **Funkcja:** Przemieszcza zaznaczony obiekt po ekranie.
-

2. Obrót Obiektu (Rotate Left / Right)

Unikalny gest wykorzystujący poziomy ruch liniowy do określenia kierunku obrotu.

Działanie robota:

- **Nacisnąć i długo przytrzymać rysik.**
- **Podczas przytrzymania:**
 - **Ruch w PRAWO:** inicjuje obrót zgodnie z ruchem wskazówek zegara.
 - **Ruch w LEWO:** inicjuje obrót przeciwnie do ruchu wskazówek zegara.

Funkcja: Obraca zaznaczony obiekt wokół jego środka.

3. Usuwanie Obiektu (Triple-Tap Delete)

Specjalny gest usuwania oparty na rytmie dotknięć.

Działanie robota: Należy wykonać trzy kolejne dotknięcia obiektu.

Warunek: Wszystkie trzy dotknięcia muszą nastąpić w określonym czasie (np. w ciągu 4 sekund).

Funkcja: Usuwa zaznaczony obiekt z interfejsu.

10. Metodyka pracy nad projektem

10.1. Organizacja zespołu projektowego

Zespół projektowy składa się z pięciu członków:

Danylo Chetvartkov, Nikita Tolstoi, Volodymyr Vyshivetskiy, Zakhar Seminkievicz, Yehor Maksymenko.

Każda osoba pełni określoną rolę funkcjonalną:

Członek zespołu	Odpowiedzialność główna	Zadania szczegółowe
Danylo Chetvertakov	Architektura systemu, elektronika	Schematy PCB, integracja sensorów, protokoły komunikacyjne

Członek zespołu	Odpowiedzialność główna	Zadania szczegółowe
Nikita Tolstoi	Oprogramowanie Python, interfejs graficzny	Implementacja GUI, gesty dotykowe, biblioteki graficzne
Volodymyr Vyshnovetskyi	Mechanika, konstrukcja 3D	Modelowanie CAD, analiza wytrzymałościowa, montaż
Zakhar Semiankevich	Algorytmy sterowania i kinematyka	IK/FK, trajektorie, optymalizacja
Yehor Maksymenko	Testy, dokumentacja, integracja końcowa	Walidacja ruchów, scenariusze testowe, raporty

10.2. Cykl realizacji projektu (Workflow)

Prace przebiegają w powtarzalnych iteracjach trwających 1–2 tygodnie, zgodnie z uproszczoną metodyką **Agile / Scrum**.

Każda iteracja składa się z etapów:

1. Planowanie sprintu

- Określenie listy funkcji do wykonania (Backlog).
- Przydział zadań do członków zespołu.

2. Projektowanie i implementacja

- Tworzenie kodu, części CAD, schematów elektronicznych.
- Przegląd zmian (peer-review).

3. Integracja

- Łączenie modułów mechanicznych z elektroniką i oprogramowaniem.

4. Testy funkcjonalne i pomiarowe

- Testy ruchu, testy sensoryczne, testy stabilności platformy.

5. Retrospektywa

- Wnioski, poprawki, analiza błędów.
-

10.3. Struktura repozytorium programu

Dane przechowywane są w repozytorium Git (GitHub/GitLab).

Proponowana struktura:

/robot/

```

|
|— hardware/
|   |— cad/          # Modele 3D, pliki STEP, STL
|   |— pcb/          # Schematy i layouty
|   |— bill_of_materials/ # Lista komponentów (BOM)
|
|— firmware/
|   |— arduino/      # Kod mikrosterowników
|   |— dynamixel/   # Konfiguracje i testery osi
|
|— software/
|   |— gui/          # Interfejs PyQt5 (gesty, figury)
|   |— control/      # Algorytmy sterowania
|   |— kinematics/   # FK/IK, transformacje
|   |— sensors/      # Drivery IMU, enkoderów, FSR
|   |— utils/        # Narzędzia pomocnicze
|   |— tests/        # Testy jednostkowe i integracyjne
|
|— docs/
|   |— technical_doc/ # Dokumentacja techniczna
|   |— calibration/  # Procedury kalibracji
|   |— measurements/ # Pomiary i raporty

```

10.4. Standard pracy z Git (Branching Model)

W projekcie obowiązuje uproszczony model:

- **main** – stabilna, przetestowana wersja systemu
- **develop** – bieżące prace programistyczne
- **feature/xxx** – osobne gałęzie dla funkcji, np.:
 - feature/gui-gestures
 - feature/ik-solver

- feature/imu-driver

Zasady:

1. Każda funkcja powstaje w osobnym branchu.
2. Zmiany przechodzą przez **Pull Request** i **code review**.
3. Merging do main tylko po pozytywnych testach.

10.5. Komunikacja wewnętrzna

Zespół korzysta z kilku kanałów komunikacji:

- **Telegram / Discord** – szybkie ustalenia
- **Google Docs** – współdzielona dokumentacja
- **Git Issues** – zgłaszanie błędów
- **Figma / Notion (opcjonalnie)** – planowanie i makiety UI

10.6. Procedury testowe

Każdy nowy moduł podlega weryfikacji:

Testy programowe

- testy jednostkowe (Python unittest),
- testy integracyjne (komunikacja I2C, UART, Dynamixel),
- stress-testy serw i IMU.

Testy mechaniczne

- pomiar luzów konstrukcyjnych,
- test obciążeniowy platformy,
- weryfikacja prędkości i zakresów ruchu.

Testy interakcji użytkownika

- testy obsługi dotykowej ekranu,
- testy gestów (tap, long tap, drag, triple tap),
- testy stabilności GUI przy wysokiej liczbie odświeżeń.

10.7. Zasady tworzenia dokumentacji

Każda sekcja dokumentacji musi zawierać:

1. **Opis funkcjonalny** – co moduł robi
2. **Schemat** lub rysunek (jeśli dotyczy)
3. **Kod lub pseudokod**
4. **Parametry techniczne**
5. **Instrukcję testowania**

Dokument aktualizowany jest co sprint w folderze **/docs/technical_doc**.

10.8. Harmonogram prac (skrót)

Etap	Opis	Status
Projekt mechaniczny	Konstrukcja platformy	W trakcie
Integracja sensorów	IMU, FSR, enkodery	W toku
Testy końcowe	Walidacja całego systemu	W planie

11. Kalibracja systemu

11.1. Wprowadzenie

Kalibracja jest procesem dostosowania parametrów mechanicznych, elektrycznych i sensorycznych tak, aby robot równoległy wykonywał ruchy z pełną zgodnością z modelem matematycznym. Prawidłowa kalibracja gwarantuje stabilność, powtarzalność oraz bezpieczeństwo pracy.

11.2. Kalibracja mechaniczna

11.2.1. Ustawienie punktów referencyjnych

- Ustalenie punktu „Home” platformy.
- Wyzerowanie orientacji
- Pomiary geometrii:
 - rozstaw punktów bazowych,
 - długości cięgien,
 - offsety przegubów.

11.2.2. Kompensacja luzów i odkształceń

- Pomiar luzów w każdym przegubie.
 - Analiza elastyczności elementów CFRP.
 - Ustalenie korekt geometrycznych.
-

11.3. Kalibracja serwomechanizmów

11.3.1. Pozycja neutralna

- Ustawienie serw na 0° według producenta.
- Korekta mechaniczną śrubą łączącą wał i ramię.

11.3.2. Offsety enkoderów

- Pomiar różnicy pomiędzy oczekiwaną a faktyczną pozycją.
- Zapis offsetu do EEPROM / plików konfiguracyjnych.

11.3.3. Kalibracja prądowa

- Ustalanie limitów prądowych zabezpieczających.
 - Test stabilności na obciążeniu.
-

11.4. Kalibracja IMU

11.4.1. Kalibracja statyczna

- Wyznaczanie biasu żyroskopu (średnia z 500 pomiarów).
- Normalizacja odczytów akcelerometru.

11.4.2. Kalibracja dynamiczna

- Procedura ruchu w 6 osiach.
 - Macierz kompensacji „soft iron/hard iron”.
-

11.5. Kalibracja czujników nacisku FSR

- Wyznaczenie charakterystyki logarytmicznej.
 - Dopasowanie krzywej:
-

11.6. Kalibracja ekranu dotykowego

- Procedura 5-punktowa (rogi + centrum).

- Korekcje nieliniowe.
 - Kompensacja dryfu temperatury.
-

11.7. Test końcowy „Calibration OK”

- Test wektorowy platformy w 6 kierunkach.
 - Weryfikacja poprawności gestów dotykowych.
 - Zapis konfiguracji kalibracyjnej do systemu.
-

12. Bezpieczeństwo pracy systemu

12.1. Bezpieczeństwo elektryczne

- Zasilacz z zabezpieczeniami OCP, SCP, OVP.
 - Uziemienie konstrukcji aluminiowej.
 - Filtry ESD na liniach I2C.
-

12.2. Bezpieczeństwo mechaniczne

- Maksymalna prędkość ruchu platformy: **0.4 m/s**.
 - Minimalna odległość użytkownika od robota: **30 cm**.
 - Zakaz wkładania dłoni pod ruchomą platformę.
-

12.3. Bezpieczeństwo oprogramowania

- Watchdog 1–2 s monitorujący zawieszenie programu.
 - Blokada ruchu przy braku kalibracji.
 - Autoryzacja zmian ustawień.
-

12.4. System awaryjny (E-STOP)

- Natychmiastowe odcięcie zasilania serw.
 - Zatrzymanie komunikacji RS485.
-

13. Analiza ryzyka

13.1. Identyfikacja zagrożeń

Typ zagrożenia	Opis	Skutek
Mechaniczne	Nagły ruch platformy	Kontuzja dłoni
Elektryczne	Zwarcie / przeciążenie	Uszkodzenie elektroniki
Software	Błędna trajektoria	Kolizja ramion
Sensor	Dryf IMU	Utrata stabilności

13.2. Ocena ryzyka

Kolory:

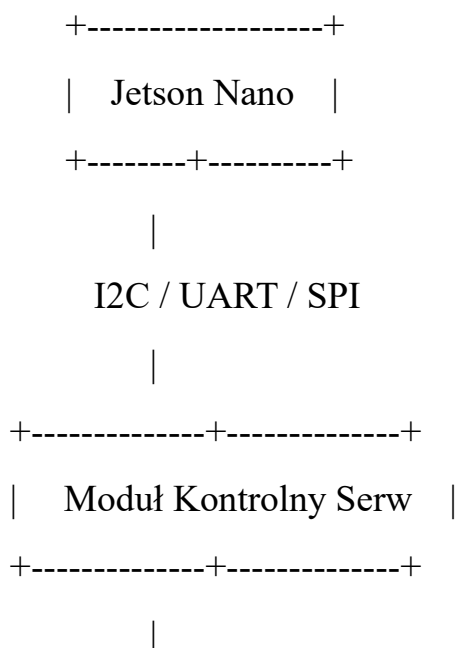
- **zielony** – niski poziom ryzyka
- **żółty** – średni
- **czerwony** – krytyczny

13.3. Strategie redukcji ryzyka

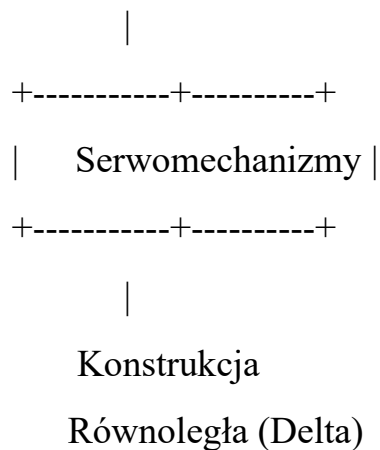
- Filtry w IMU (Kalman, Madgwick).
- Ograniczniki ruchów.
- Kontrola temperatury w serwach.

14. Diagramy blokowe (opisowe + ASCII)

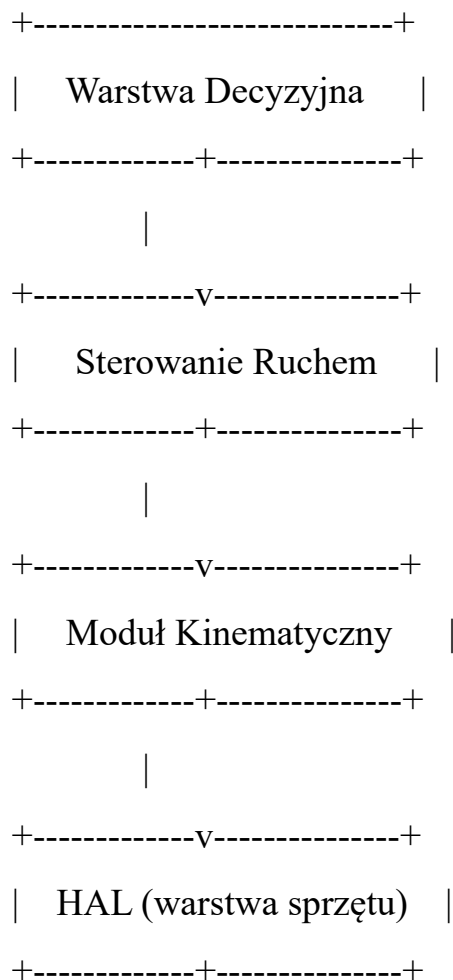
14.1. Topologia sprzętowa



RS485 / PWM



14.2. Architektura oprogramowania



16. Schematy elektryczne (ASCII + opis)

16.1. Zasilanie

12V PSU

|

+----> Step-Down 5V ---> Jetson Nano

|

+----> 12V ---> Serwomechanizmy

17. Workflow montażu

17.1. Montaż mechaniczny

1. Montaż ramy bazowej.
2. Instalacja przegubów kulowych.
3. Montaż platformy ruchomej.
4. Montaż ramion i łączników.

17.2. Montaż elektroniki

1. Instalacja kontrolera.
2. Okablowanie IMU, OLED, FSR.
3. Montaż zasilania.

17.3. Testy po montażu

- Test mechaniczny,
- Test czujników,
- Test sterowania ruchem,
- Test komunikacji.

18. Instrukcja użytkownika

18.1. Uruchamianie

1. Włącz zasilanie 12V.
2. Jetson uruchomi GUI.

3. System wykona autotest.

18.2. Tryby pracy

- Tryb ręczny,
 - Tryb automatyczny,
 - Tryb testowy.
-

18.3. Obsługa gestów

- Tap → wybór
 - Long tap → rotacja
 - Drag → przesuwanie
 - Triple tap → usuwanie figury
-

18.4. Tryb serwisowy

- Dostęp tylko dla operatora.
- Możliwość ponownej kalibracji.

19. Wytraty

19.1. Ekran