



**Politechnika  
Śląska**

**Wydział Automatyki, Elektroniki  
i Informatyki**

Podstawy Robotyki

Mateusz Siedliski  
Radosław Tchórzewski  
Paweł Mendzik  
Jakub Kula  
Oliver Davis

Gliwice 2023



---

## Spis treści

<b>1 Wstęp</b>	<b>3</b>
1.1 Cel i zakres projektu . . . . .	3
Cel projektu . . . . .	3
<b>2 Urządzenie wraz z aplikacją</b>	<b>3</b>
2.1 Wykonanie . . . . .	3
Wybór elementów . . . . .	3
Model 3D . . . . .	4
Konstrukcja mechaniczna . . . . .	7
Aplikacja . . . . .	9
Protokół komunikacyjny . . . . .	12
<b>3 Podsumowanie</b>	<b>13</b>
3.1 Zrealizowane założenia . . . . .	13
3.2 Poprawki . . . . .	13
3.3 Dalszy rozwój . . . . .	13
<b>4 Załączniki</b>	<b>14</b>



## Spis rysunków

1	Proces tworzenia modelu 3D — Widok z boku . . . . .	4
2	Proces tworzenia modelu 3D — Widok z góry . . . . .	5
3	Gotowy model w programie <u>Autodesk Viewer</u> . . . . .	6
4	Konstrukcja mechaniczna — chwytak . . . . .	7
5	Konstrukcja mechaniczna — podstawa . . . . .	8
6	MIT App Inventor — aplikacja . . . . .	9
7	MIT App Inventor — kod . . . . .	9
8	Gotowa aplikacja . . . . .	10
9	Logo aplikacji . . . . .	11
10	Model stworzonego manipulatora . . . . .	13
3	Kod aplikacji — Bluetooth . . . . .	21
4	Kod aplikacji — Przyciski . . . . .	22
5	Kod aplikacji — Slidery . . . . .	23
6	Kod aplikacji — Komendy . . . . .	24

## Spis tabel

1	Protokół komunikacyjny . . . . .	12
---	----------------------------------	----

## Bibliografia

- [1] How To Mechatronics. *DIY Arduino Robot Arm with Smartphone Control*. Data dostępu: 2022.  
URL: [https://www.youtube.com/watch?v=\\_B3gWd3A\\_SI](https://www.youtube.com/watch?v=_B3gWd3A_SI).



## 1 Wstęp

Główną inspiracją projektu był film zamieszczony na platformie YouTube z kanału „How To Mechatronics“ pod tytułem „DIY Arduino Robot Arm with Smartphone Control“ [1].

Nasz projekt korzysta z tych samych technologii, aczkolwiek wszystkie elementy (model 3D, oprogramowanie mikrokontrolera, kod aplikacji, schemat połączeń itd.) zostały przygotowane przez nas.

W ramach projektu stworzono dydaktyczny model 5 osiowego manipulatora z chwytkiem, zrealizowanego w technologii druku 3D, sterowany aplikacją na urządzeniu mobilne z systemem Android.

### 1.1 Cel i zakres projektu

#### Cel projektu

Celem projektu była realizacja fizycznego modelu manipulatora przeznaczonego do celów dydaktycznych. Może być on pomocny dla studentów w celu wizualizacji koncepcji teoretycznych w prawdziwym świecie, a nie tylko w książkach, czy na ekranie komputera.

## 2 Urządzenie wraz z aplikacją

### 2.1 Wykonanie

#### Wybór elementów

Prace rozpoczęto od doboru elementów elektronicznych i mechanicznych oraz wyboru procesu technologicznego wykonania manipulatora:

- M5Stack Core2 — ESP32-D0WD-V3
- PCA9685PW — 16 kanałowy 12-Bit sterownik I<sup>2</sup>C PWM serwo
- Micro Servo 9g SG90
- Servo Mg996r
- Obudowa ramienia (druk 3D)
- Przewody
- Wkręty M3
- Śruby M3
- Nakrętki M3
- Drewniana podstawa

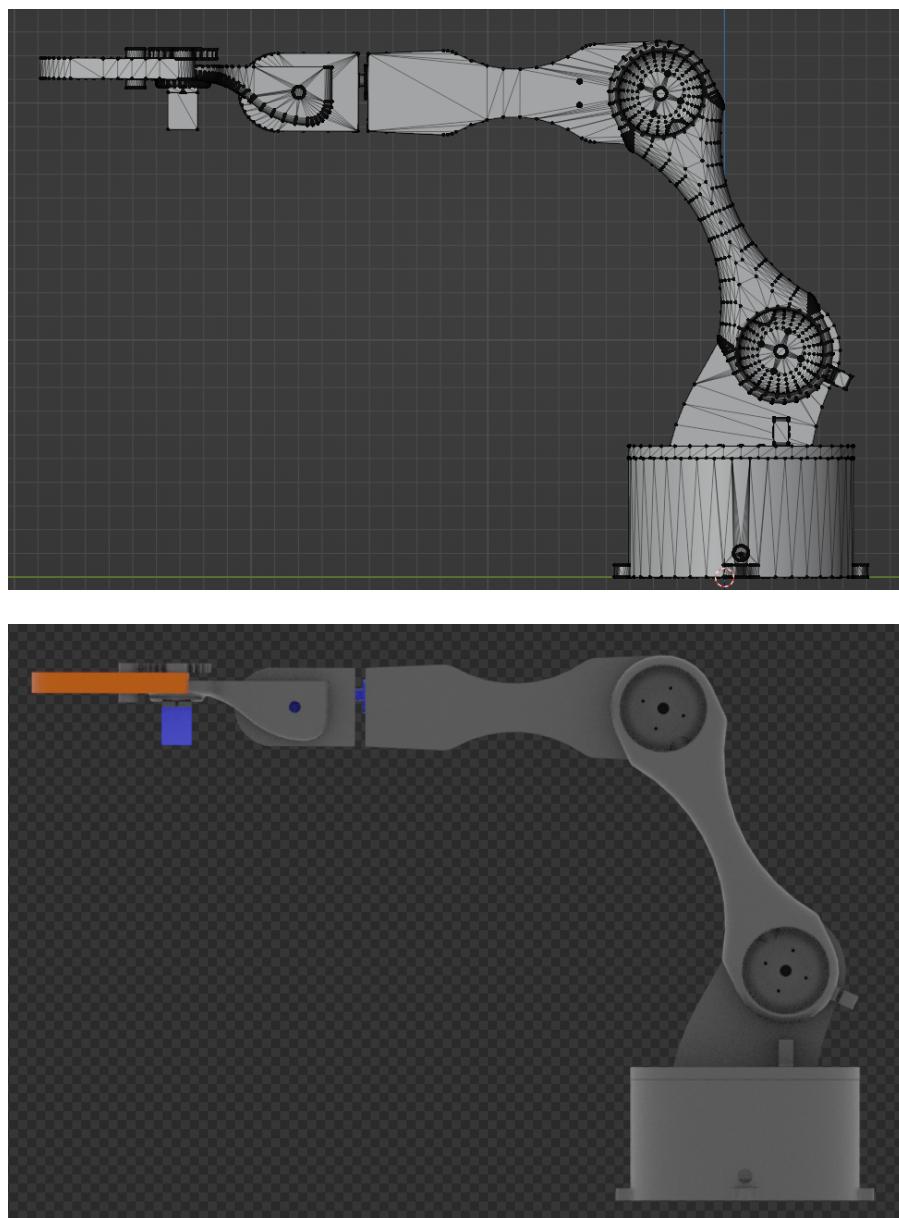
Jako proces technologiczny wykorzystany do stworzenia korpusu urządzenia wybrano druk 3D ze względu na niski koszt i powszechną dostępność. Technologia ta również umożliwia optymalizację procesu twórczego poprzez wielokrotne iteracje.



## Model 3D

Następnym krokiem było zaprojektowanie modelu 3D manipulatora przy użyciu programu Autodesk Fusion 360<sup>1</sup>.

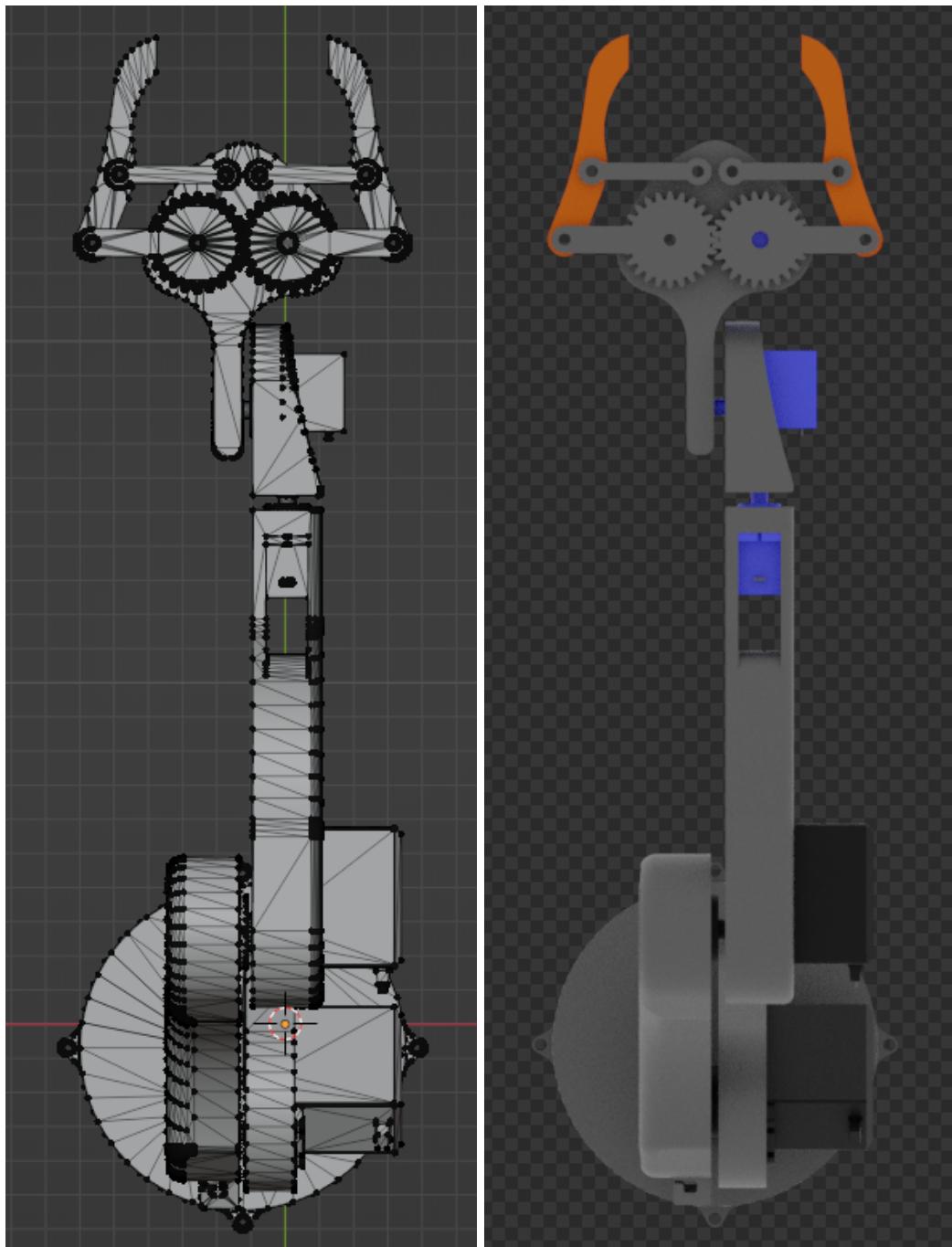
Program Autodesk Fusion 360 to bardzo przystępna alternatywa dla środowiska Autodesk Inventor. Przy braku poprzedniego doświadczenia autorzy byli w stanie całkowicie od zera zaprojektować i zrealizować w pełni działający model manipulatora. Proces tworzenia korpusu pokazano na Rysunkach 1 i 2<sup>2</sup>, a gotowy model przedstawiono na Rysunku 3.



Rysunek 1: Proces tworzenia modelu 3D — Widok z boku

<sup>1</sup>Zintegrowane oprogramowanie CAD, CAM, CAE i PCB

<sup>2</sup>Wizualizacje stworzono w programie Blender



Rysunek 2: Proces tworzenia modelu 3D — Widok z góry



**AUTODESK** Viewer

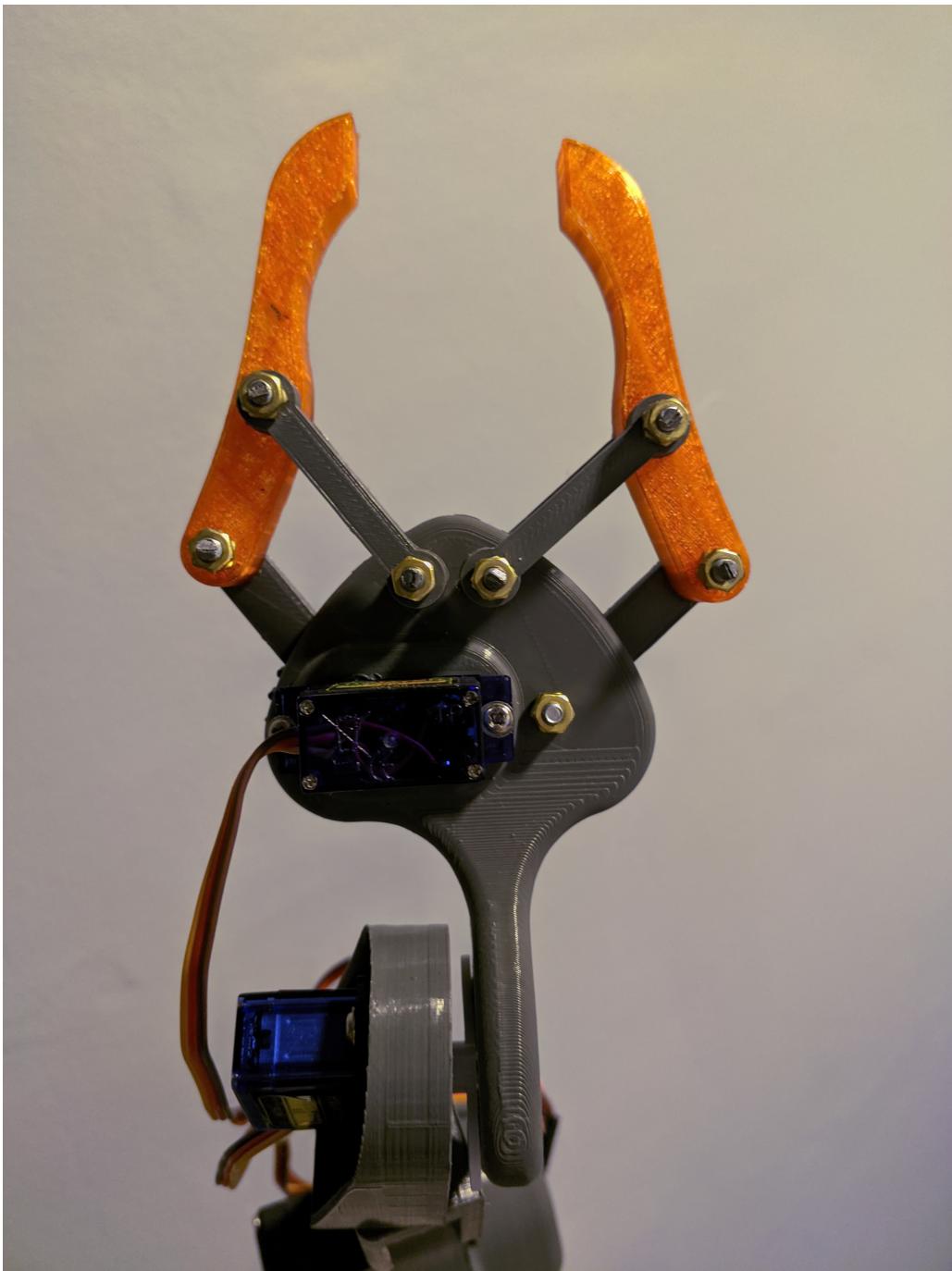
**AUTODESK**

Rysunek 3: Gotowy model w programie Autodesk Viewer  
Narzędzie online do wyświetlania plików 2D i 3D



### Konstrukcja mechaniczna

Model złożono przy użyciu śrub, wkrętów i nakrętek M3, co przedstawiono na Rysunkach 4 i 5.



Rysunek 4: Konstrukcja mechaniczna — chwytak

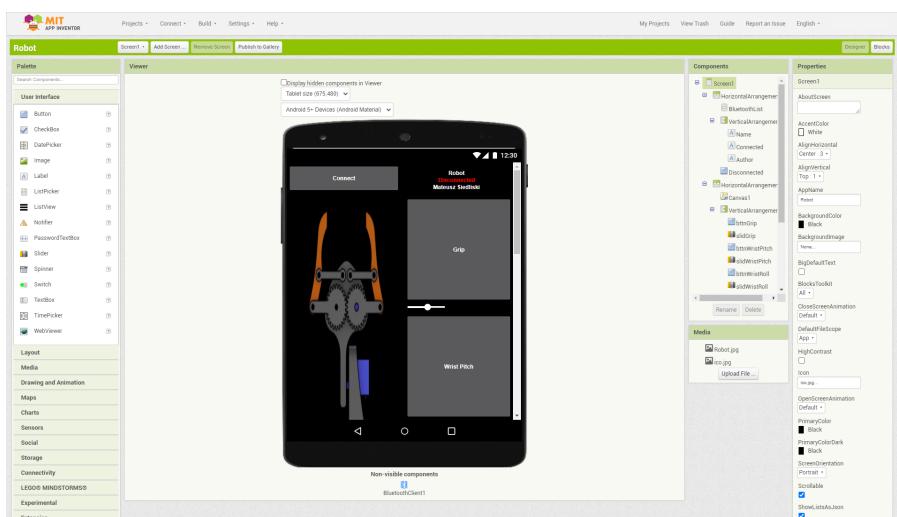


Rysunek 5: Konstrukcja mechaniczna — podstawa

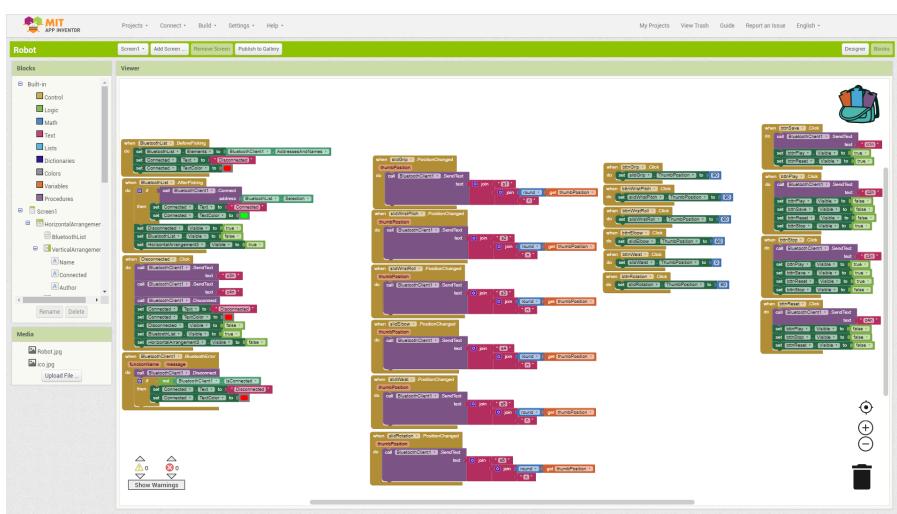


## Aplikacja

Aplikacja powstała w MIT App Inventor<sup>3</sup> (Rysunki 6 i 7). Środowisko to oferuje prostotę w realizacji projektów. Nie wymaga żadnego doświadczenia od użytkownika. Programowanie odbywa się we własnym języku graficznym (duże podobieństwo do Scratch<sup>4</sup>). Kod dostarczonej aplikacji pokazano na Załącznikach 3, 4, 5 i 6. Na Rysunku 8 przedstawiono ostateczną szatę graficzną aplikacji, a na Rysunku 9 zademonstrowano logo aplikacji.



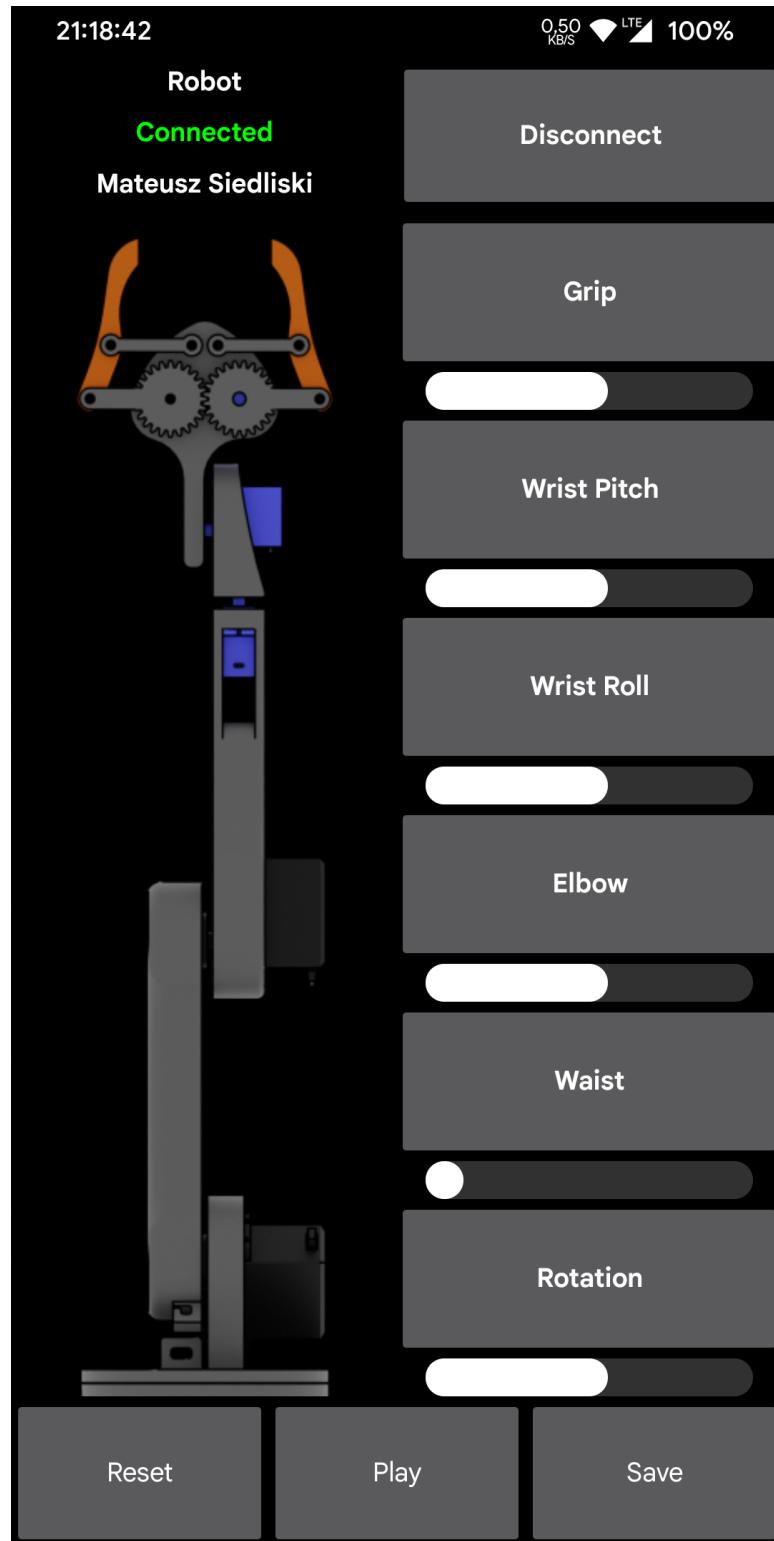
Rysunek 6: MIT App Inventor — aplikacja



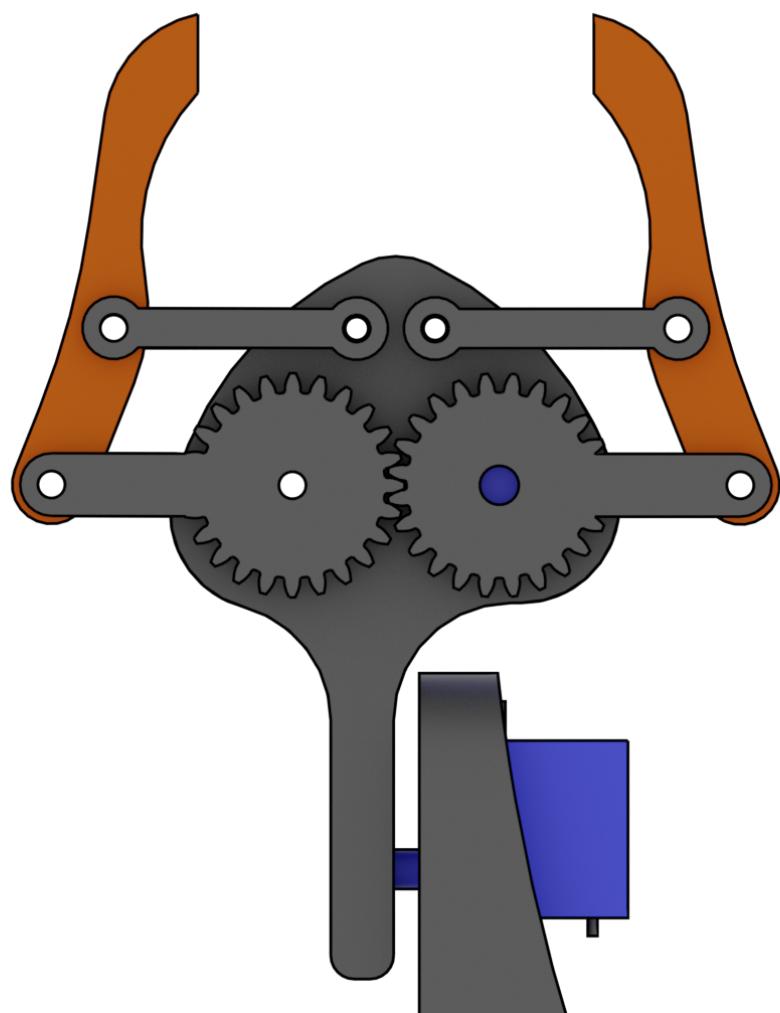
Rysunek 7: MIT App Inventor — kod

<sup>3</sup>Zintegrowane środowisko programistyczne do tworzenia aplikacji mobilnych

<sup>4</sup>Interpretowany wizualny język programowania



Rysunek 8: Gotowa aplikacja



Rysunek 9: Logo aplikacji



## Protokół komunikacyjny

Komunikacja aplikacji mobilnej z mikrokontrolerem odbywa się poprzez protokół Bluetooth. Każdy komunikat to odpowiednio sformatowany string. Komendy komunikacyjne przedstawiono w Tabeli 1. Przykładowe komunikaty:

- „s1128n“ - ustawienie serwomechanizmu 1 na pozycję 128
- „s564n“ - ustawienie serwomechanizmu 5 na pozycję 64
- „c1n“ - zapisanie aktualnej pozycji serwomechanizmów w pamięci do późniejszego odtworzenia
- „c4n“ - wyczyszczenie zapisanej sekwencji ruchowej

Komenda	Numer	Wartość	Funkcja
s	1	x	Ustawienie serwomechanizmu na pozycję x
s	2	x	Ustawienie serwomechanizmu na pozycję x
s	3	x	Ustawienie serwomechanizmu na pozycję x
s	4	x	Ustawienie serwomechanizmu na pozycję x
s	5	x	Ustawienie serwomechanizmu na pozycję x
s	6	x	Ustawienie serwomechanizmu na pozycję x
c	1	-	Save — zapisanie aktualnej pozycji serwomechanizmów w pamięci do późniejszego odtworzenia
c	2	-	Play — odtwarzanie zapisanej sekwencji ruchowej
c	3	-	Stop — koniec odtwarzania zapisanej sekwencji ruchowej
c	4	-	Reset — wyczyszczenie zapisanej sekwencji ruchowej

Tabela 1: Protokół komunikacyjny



### 3 Podsumowanie

#### 3.1 Zrealizowane założenia

Wszystkie założenia projektowe zostały zrealizowane. Model stworzonego manipulatora przedstawiono na Rysunku 10.



Rysunek 10: Model stworzonego manipulatora

#### 3.2 Poprawki

Najważniejszą poprawką, którą można zastosować, jest zmiana modelu 3D. W kolejnej wersji należy uwzględnić tolerancje mechaniczne druku oraz nabycie doświadczenia.

#### 3.3 Dalszy rozwój

Propozycje dalszego rozwoju:

- Zastosowanie wyżej wymienionych poprawek.
- Implementacja enkoderów cyfrowych na każdej z osi w celu uzyskania sprzężenia zwrotnego.
- Rozwój funkcjonalności aplikacji mobilnej.
- Aplikacja mobilna na inne systemy operacyjne.
- Zastosowanie kinematyki odwrotnej.
- Alternatywne formy sterowania.



## 4 Załączniki

Załącznik 1: Kod mikrokontrolera

```
1 #include <Arduino.h>
2 #include <M5Core2.h>
3 #include <Adafruit_PWMServoDriver.h>
4 #include <BluetoothSerial.h>
5
6 // Task stack size
7 #define STACK 2048
8
9 // Initial positions
10 #define servo01235InitPos 90;
11 #define servo4InitPos 0;
12
13 // Max memory size
14 #define memory 50
15
16 // Timing for degtoms
17 const uint16_t minUs = 1000;
18 const uint16_t maxUs = 2000;
19
20 // Number of servos
21 const int servoNum = 6;
22
23 // PWM driver
24 Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
25
26 // Bluetooth
27 BluetoothSerial Bluetooth;
28
29 // Servo screen positions
30 const int servoPosScreen[] = {20, 60, 100, 140, 180, 220};
31
32 // Servo max positions
33 const int servoPosMax[] = {180, 180, 180, 180, 130, 180};
34
35 // Servo positions
36 int servoPos[servoNum];
37
38 // Servo previous positions
39 int servoPPos[servoNum];
40
41 // Servo saved positions
42 int servoSPos[servoNum][memory];
43
44 // Position in saved positions
45 int indexS = 0;
46
47 enum Mode
48 {
49     move = 0,
50     play = 1,
51     stop = 2
52 };
53
54 struct MoveCommand
```



```
55 {
56     int whichServo;
57     int posServo;
58     Mode mode;
59 };
60
61 // Queue
62 static QueueHandle_t xQueueData;
63 static QueueHandle_t xQueueMove;
64 static QueueHandle_t xQueueDraw;
65
66 // Prototypes
67 // Tasks
68 void taskBT(void *param);
69 void taskParser(void *param);
70 void taskMoveServo(void *param);
71 void taskUpdateScreen(void *param);
72
73 // Functions
74 void setMem();
75 void saveMem();
76 bool checkPos(int pos);
77 int mapper(int value);
78 void updateText();
79
80 // Tasks
81
82 void setup()
83 {
84     disableCoreOWDT();
85
86     M5.begin(true, true, true, true);
87     M5.Lcd.drawJpgFile(SD, "/logo.jpg");
88
89     Serial.begin(115200);
90
91     // Initializing PWM driver
92     pwm.begin();
93     pwm.setPWMFreq(60);
94
95     // Initializing Bluetooth
96     Bluetooth.begin("M5Core2");
97
98     // Setting initial state
99     setMem();
100    for (size_t i = 0; i < servoNum; i++)
101    {
102        servoPos[i] = servoSPos[i][0];
103        pwm.writeMicroseconds(i, mapper(servoPos[i]));
104    }
105
106    while (!Bluetooth.available()) // Check BT
107    {
108        // Nothing to do
109    }
110
111    // Creating queues
112    xQueueData = xQueueCreate(10, sizeof(String));
```



```
113 xQueueMove = xQueueCreate(10, sizeof(MoveCommand));
114 xQueueDraw = xQueueCreate(10, sizeof(int));
115
116 // Initializing the screen
117 M5.Lcd.clear(BLACK);
118 for (size_t i = 0; i < servoNum; i++)
119 {
120     if (xQueueSend(xQueueDraw, (void *)&i, 10) != pdTRUE)
121     {
122         Serial.println("ERROR: Could not put item on draw queue.");
123     }
124 }
125 updateText();
126
127 // Creating tasks
128 xTaskCreate(taskBT, "BT", STACK, NULL, 1, NULL);
129 xTaskCreate(taskParser, "Parser", STACK, NULL, 2, NULL);
130 xTaskCreate(taskMoveServo, "MoveServo", STACK, NULL, 3, NULL);
131 xTaskCreate(taskUpdateScreen, "updateScreen", STACK, NULL, 4, NULL);
132
133 vTaskDelete(NULL);
134 }
135
136 void loop() {}
137
138 void taskBT(void *param)
139 {
140     String data;
141
142     while (1)
143     {
144         while (!Bluetooth.available()) // Check BT
145         {
146             vTaskDelay(pdMS_TO_TICKS(100)); // Nothing to do
147         }
148
149         data = Bluetooth.readStringUntil('\n'); // Receive BT data
150         Serial.println(data);
151         if (xQueueSend(xQueueData, (void *)&data, 10) != pdTRUE)
152         {
153             Serial.println("ERROR: Could not put item on data queue.");
154         }
155     }
156 }
157
158 void taskParser(void *param)
159 {
160     String data;
161     MoveCommand command;
162
163     while (1)
164     {
165         if (xQueueReceive(xQueueData, (void *)&data, 0) == pdTRUE)
166         {
167             if (data.startsWith("s")) // Servo position?
168             {
169                 command.whichServo = data.substring(1, 2).toInt() - 1;
170                 command.posServo = data.substring(2, data.length()).toInt();
171             }
172         }
173     }
174 }
```



```
171         command.mode = move;
172
173         xQueueSend(xQueueMove, (void *)&command, 10);
174         xQueueSend(xQueueDraw, (void *)&command.whichServo, 10);
175     }
176     else if (data.startsWith("c")) // Command?
177     {
178         // Which command?
179         int SelectFunc = data.substring(1, 2).toInt();
180
181         switch (SelectFunc)
182         {
183             case 1: // Save
184                 // Save all current servo positions
185                 saveMem();
186                 break;
187
188             case 2: // Play
189                 command.mode = play; // Repeat until Stop command
190                 xQueueSend(xQueueMove, (void *)&command, 10);
191                 break;
192
193             case 3: // Stop
194                 command.mode = stop; // Stop command
195                 xQueueSend(xQueueMove, (void *)&command, 10);
196                 break;
197
198             case 4: // Reset
199                 setMem(); // Setting initial positions
200                 break;
201         }
202         vTaskDelay(pdMS_TO_TICKS(1));
203     }
204 }
205 }
206 }
207
208 void taskMoveServo(void *param)
209 {
210     MoveCommand command;
211     while (1)
212     {
213         if (xQueueReceive(xQueueMove, (void *)&command, 0) == pdTRUE)
214         {
215             switch (command.mode)
216             {
217                 case move:
218                 {
219                     int whichServo = command.whichServo;
220                     int posServo = command.posServo;
221
222                     servoPos[whichServo] = posServo;
223
224                     // Movement speed adjustment
225                     int offset = 2;
226
227                     if (whichServo <= 2)
228                     {
```



```
229         offset = 30;
230     }
231
232     if (servoPos[whichServo] > servoPPos[whichServo])
233     {
234         for (int i = servoPPos[whichServo]; i <= servoPos[whichServo]; i
235             += offset)
236         {
237             pwm.writeMicroseconds(whichServo, mapper(i));
238         }
239         pwm.writeMicroseconds(whichServo, mapper(servoPos[whichServo]));
240     }
241     else if (servoPos[whichServo] < servoPPos[whichServo])
242     {
243         for (int i = servoPPos[whichServo]; i >= servoPos[whichServo]; i
244             -= offset)
245         {
246             pwm.writeMicroseconds(whichServo, mapper(i));
247         }
248         pwm.writeMicroseconds(whichServo, mapper(servoPos[whichServo]));
249     }
250
251     servoPPos[whichServo] = servoPos[whichServo];
252     break;
253 }
254
255 case play: // Repeat until Stop command
256 {
257     while (command.mode != stop)
258     {
259         for (int j = 0; j < indexS; j++)
260         {
261             xQueueReceive(xQueueMove, (void *)&command, 10);
262             if (command.mode == stop) // Stop?
263             {
264                 break;
265             }
266
267             // Move all servos simultaneously
268             while (checkPos(j)) // Check if all servos are in position
269             {
270                 for (size_t i = 0; i < servoNum; i++)
271                 {
272                     if (servoPos[i] > servoSPos[i][j])
273                     {
274                         pwm.writeMicroseconds(i, mapper(--servoPos[i]));
275                     }
276                     else if (servoPos[i] < servoSPos[i][j])
277                     {
278                         pwm.writeMicroseconds(i, mapper(++servoPos[i]));
279                     }
280                 }
281             }
282         }
283     }
284 }
```



```
285
286     default:
287         break;
288     }
289     vTaskDelay(pdMS_TO_TICKS(1));
290 }
291 }
292 }
293
294 void taskUpdateScreen(void *param)
295 {
296     int servo;
297     while (1)
298     {
299         if (xQueueReceive(xQueueDraw, (void *)&servo, 0) == pdTRUE)
300         {
301             int pos = (servoPos[servo] / float(servoPosMax[servo])) * 320;
302             M5.Lcd.fillRect(0, servoPosScreen[servo], pos, 20, RED);
303             M5.Lcd.fillRect(pos, servoPosScreen[servo], 320, 20, BLACK);
304         }
305         vTaskDelay(pdMS_TO_TICKS(1));
306     }
307 }
308
309 // Functions
310
311 bool checkPos(int pos)
312 {
313     for (size_t i = 0; i < servoNum; i++)
314     {
315         if (servoPos[i] - servoSPos[i][pos] != 0)
316         {
317             return 1;
318         }
319     }
320     return 0;
321 }
322
323 void setMem()
324 {
325     // Setting initial positions
326     for (size_t j = 0; j < memory; j++)
327     {
328         for (size_t i = 0; i < servoNum; i++)
329         {
330             servoSPos[i][j] = servo01235InitPos;
331         }
332         servoSPos[4][j] = servo4InitPos;
333     }
334     indexS = 0;
335 }
336
337 void saveMem()
338 {
339     if (indexS < memory)
340     {
341         for (size_t i = 0; i < servoNum; i++)
342         {
```

```
343         servoSPos[i][indexS] = servoPos[i];
344     }
345     indexS++;
346 }
347 }
348
349 int mapper(int value)
350 {
351     return map(value, 0, 180, minUs, maxUs);
352 }
353
354 void updateText()
355 {
356     M5.Lcd.drawString("Grip", 0, 2, 2);
357     M5.Lcd.drawString("Wrist Pitch", 0, 42, 2);
358     M5.Lcd.drawString("Wrist Roll", 0, 82, 2);
359     M5.Lcd.drawString("Elbow", 0, 122, 2);
360     M5.Lcd.drawString("Waist", 0, 162, 2);
361     M5.Lcd.drawString("Rotation", 0, 202, 2);
362 }
```



```
when BluetoothList .BeforePicking
do set BluetoothList .Elements to BluetoothClient1 . AddressesAndNames
set Connected .Text to "Disconnected"
set Connected .TextColor to red

when BluetoothList .AfterPicking
do if call BluetoothClient1 .Connect
address BluetoothList .Selection
then set Connected .Text to "Connected"
set Connected .TextColor to green
set Disconnected .Visible to true
set BluetoothList .Visible to false
set HorizontalArrangement3 .Visible to true

when Disconnected .Click
do call BluetoothClient1 .SendText
text "c3n"
call BluetoothClient1 .SendText
text "c4n"
call BluetoothClient1 .Disconnect
set Connected .Text to "Disconnected"
set Connected .TextColor to red
set Disconnected .Visible to false
set BluetoothList .Visible to true
set HorizontalArrangement3 .Visible to false

when BluetoothClient1 .BluetoothError
functionName message
do call BluetoothClient1 .Disconnect
if not BluetoothClient1 .IsConnected
then set Connected .Text to "Disconnected"
set Connected .TextColor to red
```

Załącznik 3: Kod aplikacji — Bluetooth



```
when btnRotation .Click
do set slidRotation .ThumbPosition to 90

when btnWaist .Click
do set slidWaist .ThumbPosition to 0

when btnElbow .Click
do set slidElbow .ThumbPosition to 90

when btnWristRoll .Click
do set slidWristRoll .ThumbPosition to 90

when btnWristPitch .Click
do set slidWristPitch .ThumbPosition to 90

when btnGrip .Click
do set slidGrip .ThumbPosition to 90
```

Załącznik 4: Kod aplikacji — Przyciski



```
when [slidRotation v].PositionChanged
  [thumbPosition v]
  do
    call [BluetoothClient1 v].SendText
      text [join [s6] [join [round [get [thumbPosition v]]] [n]]]
when [slidWaist v].PositionChanged
  [thumbPosition v]
  do
    call [BluetoothClient1 v].SendText
      text [join [s5] [join [round [get [thumbPosition v]]] [n]]]
when [slidElbow v].PositionChanged
  [thumbPosition v]
  do
    call [BluetoothClient1 v].SendText
      text [join [s4] [join [round [get [thumbPosition v]]] [n]]]
when [slidWristRoll v].PositionChanged
  [thumbPosition v]
  do
    call [BluetoothClient1 v].SendText
      text [join [s3] [join [round [get [thumbPosition v]]] [n]]]
when [slidWristPitch v].PositionChanged
  [thumbPosition v]
  do
    call [BluetoothClient1 v].SendText
      text [join [s2] [join [round [get [thumbPosition v]]] [n]]]
when [slidGrip v].PositionChanged
  [thumbPosition v]
  do
    call [BluetoothClient1 v].SendText
      text [join [s1] [join [round [get [thumbPosition v]]] [n]]]
```

Załącznik 5: Kod aplikacji — Slidery



```
when btnSave .Click
do call BluetoothClient1 .SendText
    text "c1n"
    set btnPlay .Visible to true
    set btnReset .Visible to true

when btnPlay .Click
do call BluetoothClient1 .SendText
    text "c2n"
    set btnPlay .Visible to false
    set btnSave .Visible to false
    set btnReset .Visible to false
    set btnStop .Visible to true

when btnStop .Click
do call BluetoothClient1 .SendText
    text "c3n"
    set btnPlay .Visible to true
    set btnSave .Visible to true
    set btnReset .Visible to true
    set btnStop .Visible to false

when btnReset .Click
do call BluetoothClient1 .SendText
    text "c4n"
    set btnPlay .Visible to false
    set btnStop .Visible to false
    set btnReset .Visible to false
```