



Wydział Automatyki, Elektroniki i Informatyki

Projekt z Systemów Mikroprocesorowych
Ramię robota

Mateusz Siedliski i Radosław Tchórzewski
Rok akademicki 2022/2023, semestr 5, grupa 6, sekcja 2

Kierujący pracą: dr inż. Jacek Loska

Gliwice 2023

Spis treści

1	DONE? Wstęp	2
1.1	DONE? Cel i zakres projektu	2
	DONE? Cel projektu	2
	DONE? Wymagania	3
	DONE? Zakres projektu	3
2	DONE? Harmonogram	4
2.1	DONE? Harmonogram zatwierdzony	4
2.2	DONE? Harmonogram wykonany	4
3	DONE? Kosztorys	5
4	WIP Urządzenie wraz z aplikacją	6
4.1	DONE? Określenie problemu	6
4.2	DONE? Analiza rozwiązań	7
4.3	DONE? Zaproponowane rozwiązanie	8
4.4	WIP Wykonanie	9
	DONE? Wybór elementów	9
	DONE? Model 3D	10
	DONE? Konstrukcja mechaniczna	13
	DONE? Schemat elektryczny	15
	DONE? Aplikacja	17
	DONE? Protokół komunikacyjny	19
	DONE? Kod mikrokontrolera	20
4.5	TODO Problemy w trakcie tworzenia sprzętu i aplikacji . . .	21
5	TODO Podsumowanie	22
6	WIP Literatura	23
7	WIP Załączniki	24

1 **DONE? Wstęp**

Podczas studiowania na kierunku Automatyka i Robotyka można zauważać zadziwiający brak fizycznych pomocy naukowych. Ten projekt ma na celu poprawę tej sytuacji choćby w niewielkim stopniu. W tym celu zaproponowano stworzenie ramienia robota (manipulatora). Ma on na celu pomoc studentom z wizualizacją koncepcji teoretycznych w prawdziwym świecie, a nie tylko w książkach, czy na ekranie komputera.

Projekt może posłużyć także do zachęcenia potencjalnych studentów podczas na przykład dni otwartych czy wycieczek szkolnych.

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 urządzenia z systemem Android.

1.1 **DONE? Cel i zakres projektu**

DONE? 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.

Projekt może posłużyć także do zachęcenia potencjalnych studentów podczas na przykład dni otwartych czy wycieczek szkolnych.

DONE? Wymagania

- Niski koszt budowy.
- Niski koszt eksploatacji.
- Stworzony z łatwodostępnych materiałów.
- Łatwość obsługi.
- Niski koszt szkolenia.
- Niska awaryjność.
- Łatwość naprawy.
- Atrakcyjny wygląd.

DONE? Zakres projektu

- Określenie problemu i wykonanie do niego założeń.
- Analiza możliwych rozwiązań.
- Wybór elementów elektronicznych.
- Wybór elementów mechanicznych.
- Wykonanie projektu zgodnie z wcześniejszymi założeniami.
- Uruchomienie, weryfikacji i przetestowanie sprzętu i aplikacji.
- Nakreślenie ewentualnych kierunków rozwoju projektu.
- Wnioski końcowe.

2 DONE? Harmonogram

2.1 DONE? Harmonogram zatwierdzony

1. Projektowanie modelu fizycznego robota oraz jego druk w technologii 3D.
2. Montaż mechaniczny oraz elektryczny.
3. Tworzenie oprogramowania na mikrokontroler.
4. Tworzenie aplikacji sterującej.
5. Projektowanie oraz realizacja komunikacji między mikrokontrolerem, a aplikacją sterującą.

2.2 DONE? Harmonogram wykonany

Realizacja działającego prototypu zajęła zdecydowanie mniej czasu, niż początkowo zakładano. Pozwoliło to na poświęcenie większej ilości czasu na uprawnienia i udoskonalanie projektu.

1. Projektowanie modelu fizycznego robota oraz jego druk w technologii 3D. Montaż mechaniczny oraz elektryczny.
2. Tworzenie oprogramowania na mikrokontroler oraz aplikacji sterującej. Opracowanie protokołu komunikacji między mikrokontrolerem, a aplikacją sterującą.
3. Doskonalenie projektu — debugowanie, poprawki mechaniczne, usprawnienia
4. Doskonalenie projektu — debugowanie, poprawki mechaniczne, usprawnienia
5. Doskonalenie projektu — debugowanie, poprawki mechaniczne, usprawnienia

3 DONE? Kosztorys

Lp.	Typ	Producent	Ilość	Cena	Wartość
1.	Mikrokontroler Wemos D1 mini	Wemos	1	9,48 zł	9,48 zł
2.	Moduł Bluetooth HC-05	SZYTF	1	9,40 zł	9,40 zł
3.	Micro Servo 9g SG90	HWAYEH	3	3,97 zł	11,91 zł
4.	Servo Mg996r	WAVGAT	3	12 zł	36 zł
5.	Obudowa ramienia (druk 3D)	n/d	1	40 zł	40 zł
6.	Przewody	n/d	n/d	n/d	10 zł
7.	Płytki prototypowe	diymore	1	3,15 zł	3,15 zł
8.	Wkręty M3	n/d	4	0,20 zł	0,80 zł
9.	Śruby M3	n/d	8	0,20 zł	1,60 zł
10.	Nakrętki M3	n/d	18	0,20 zł	3,60 zł
11.	Drewniana podstawa	n/d	1	10,00 zł	10,00 zł
12.	Kondensator 1000 μF	Chong	1	0,50 zł	0,50 zł
Suma = 136,44 zł					
Ilość roboczogodzin = 60					

Tabela 1: Kosztorys

4 WIP Urządzenie wraz z aplikacją

Tworząc ten projekt mamy na celu stworzenie przydatnej pomocy naukowej dla osób zainteresowanych tematem automatyki i robotyki, który pozwoli na łatwiejsze przyswojenie teorii oraz zobaczenie jej praktycznego zastosowania.

4.1 DONE? Określenie problemu

Problemem większości studentów jest często brak motywacji do nauki związany z poczuciem braku sensu przerabianego materiału. Ten projekt ma na celu poprawić ten stan rzeczy. Brak wyobrażenia o realnym zastosowaniu zdobytej wiedzy utrudnia pracę. Studenci nie widzą połączenia teorii z praktyką, jak przedstawiono na Rysunku 1.

Rozwiążanie? Stworzenie fizycznego modelu.

$$A_i = \text{Rot}(z, \theta_i) \text{Trans}(0, 0, \lambda_i) \text{Trans}(l_i, 0, 0) \text{Rot}(x, \alpha_i) =$$

$$= \begin{bmatrix} C_i & -S_i C_{\alpha_i} & S_i S_{\alpha_i} & l_i C_i \\ S_i & C_i C_{\alpha_i} & -C_i S_{\alpha_i} & l_i S_i \\ 0 & S_{\alpha_i} & C_{\alpha_i} & \lambda_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[2]

⇓

?

⇓



Rysunek 1: Połączenie teorii z praktyką

4.2 DONE? Analiza rozwiązań

Przykładowe rozwiązania przedstawionego problemu:

- Wycieczki do zakładów przemysłowych
 - + Obserwacja prawdziwych zastosowań w praktyce
 - Zmiana organizacji roku akademickiego
 - Organizacja wyjazdu
- Więcej zajęć praktycznych
 - + Praca na urządzeniach pozwoli lepiej przyswoić wiedzę
 - Zmiana organizacji roku akademickiego
 - Wysoki koszt zakupu urządzeń
 - Wysoki koszt kadrowy
- Dopuszczenie studentów do pracy na prawdziwym sprzęcie już od początku studiowania
 - + Motywacja od wczesnych etapów edukacji
 - Zmiana organizacji roku akademickiego
 - Wysoki koszt zakupu urządzeń
 - Wysoki koszt kadrowy
- Stworzenie modelu prawdziwego urządzenia
 - + Niski koszt
 - + Prostota realizacji
 - Nakład pracy włożony w stworzenie modelu

Kryteria wyboru rozwiązania:

- Niskie koszta finansowe
- Niskie koszta kadrowe
- Niska ingerencja w organizację roku akademickiego
- Wysoka dostępność
- Motywacja studentów

4.3 DONE? Zaproponowane rozwiązanie

W tym punkcie należy przedstawić wybrane rozwiązanie problemu, wraz z uzasadnieniem wyboru na postawie kryteriów z poprzedniego punktu.

Wybrane rozwiązanie: „Stworzenie modelu prawdziwego urządzenia“.

Ta propozycja spełnia wszystkie wymagania projektowe:

- Niskie koszta finansowe
 - Koszt < 150 zł
- Niskie koszta kadrowe
 - Uczelnia już na chwilę obecną posiada wyszkoloną kadrę
- Niska ingerencja w organizację roku akademickiego
 - Niewielka ilość godzin potrzebna na pracę z modelem
- Wysoka dostępność
 - Niski koszt i dostępność materiałów pozwala na stworzenie wielu modeli
- Motywacja studentów
 - Dostęp do fizycznego modelu

4.4 **WIP** Wykonanie

DONE? Wybór elementów

Prace rozpoczęto od doboru elementów elektronicznych i mechanicznych oraz od wyboru procesu technologicznego wykonania manipulatora. Poniżej przedstawiono argumentację wyboru najważniejszych elementów.

Elementy:

- Mikrokontroler Wemos D1 mini
- Moduł Bluetooth HC-05
- HWAYEH Micro Servo 9g SG90
- WAVGAT Servo Mg996r

wybrano ze względu na:

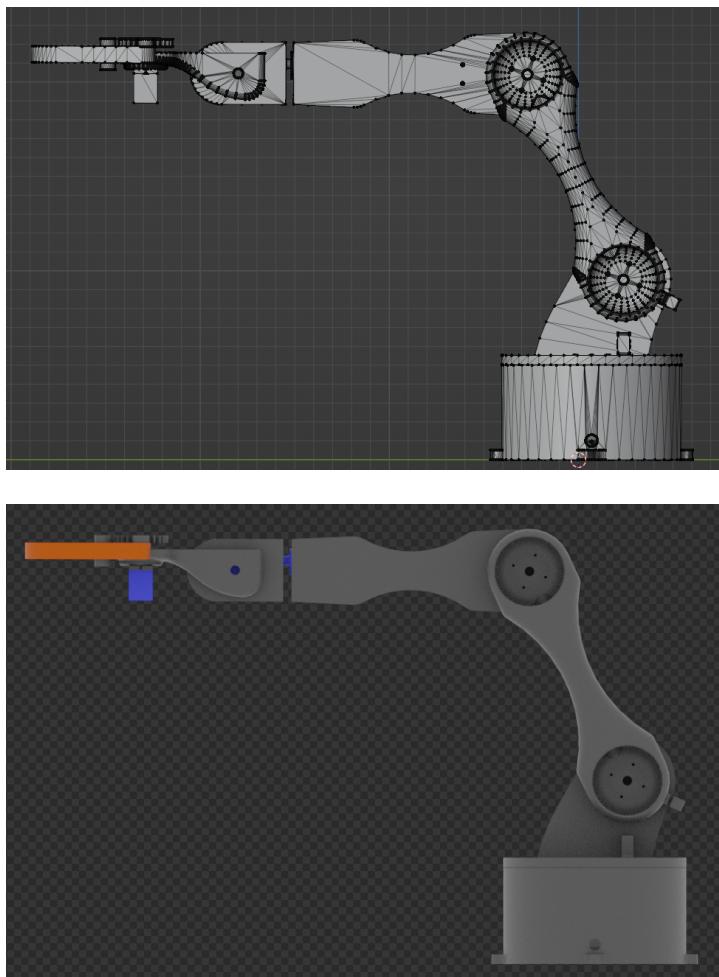
- Niski koszt
- Duża dostępność
- Zaznajomienie autorów z elementem

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.

DONE? Model 3D

Następnym krokiem było zaprojektowanie modelu 3D manipulatora przy użyciu programu Autodesk Fusion 360¹.

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 2 i 3², a gotowy model przedstawiono na Rysunku 4.



Rysunek 2: Proces tworzenia modelu 3D - Widok z boku

¹Zintegrowane oprogramowanie CAD, CAM, CAE i PCB

²Wizualizacje stworzono w programie Blender

Ramię robota



Rysunek 3: Proces tworzenia modelu 3D - Widok z góry

Ramię robota



 **AUTODESK** Viewer

 **AUTODESK**

Rysunek 4: Gotowy model w programie [Autodesk Viewer](#)
Narzędzie online do wyświetlania plików 2D i 3D

DONE? Konstrukcja mechaniczna

Model złożono przy użyciu śrub, wkrętów i nakrętek M3, co przedstawiono na Rysunkach 5 i 6.



Rysunek 5: Konstrukcja mechaniczna - chwytak

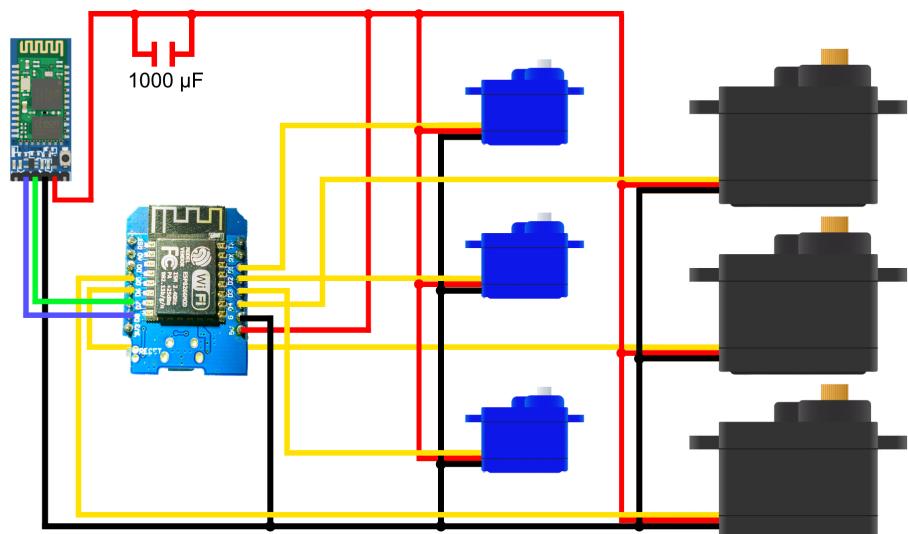
Ramię robota



Rysunek 6: Konstrukcja mechaniczna - podstawa

DONE? Schemat elektryczny

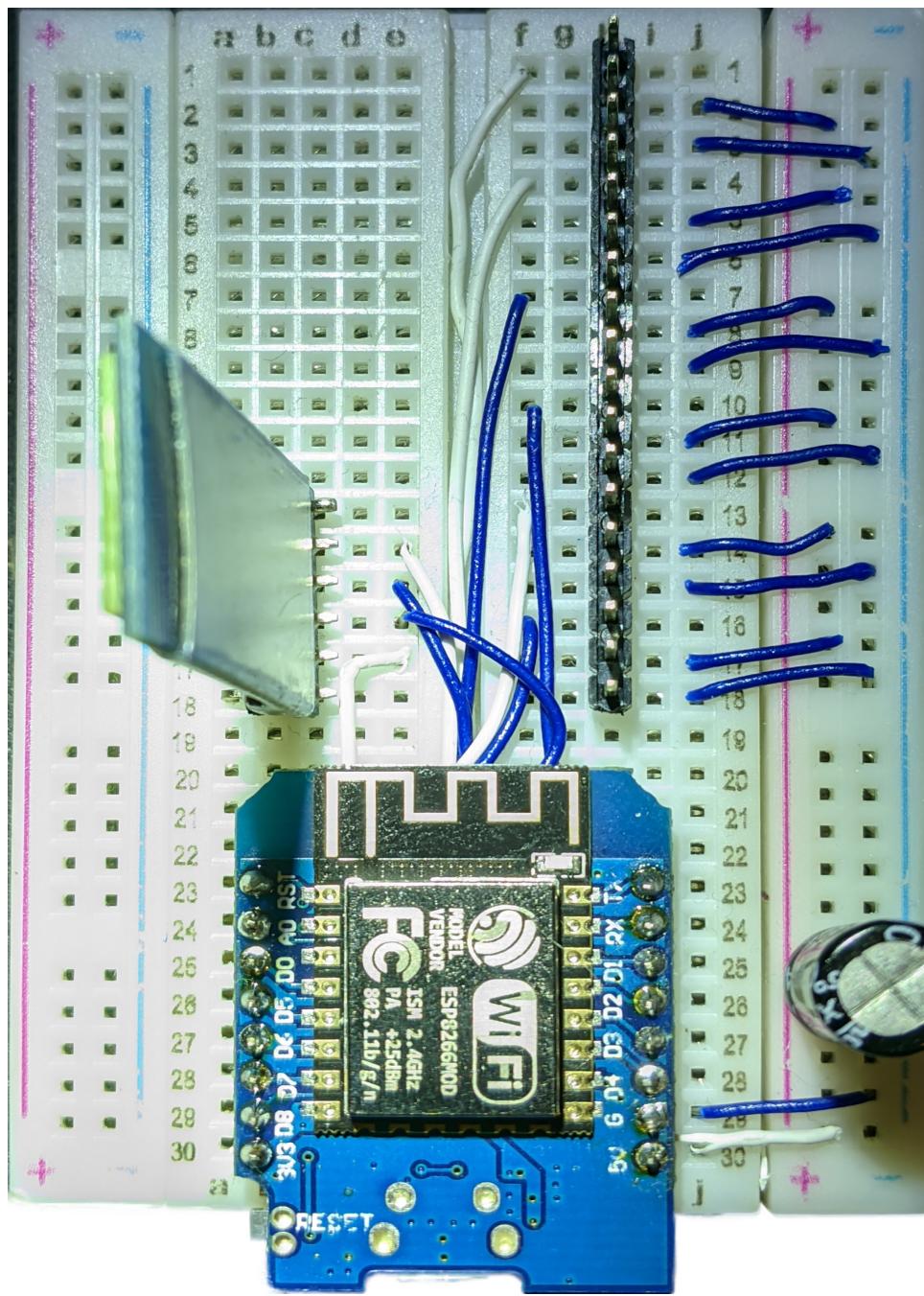
Na Rysunku 7³ przedstawiono połączenie elementów elektronicznych. Natomiast faktyczne połączenia elektryczne pokazano na Rysunku 8.



Rysunek 7: Schemat połączeń elektrycznych

³Stworzono przy użyciu circuito.io

Ramię robota



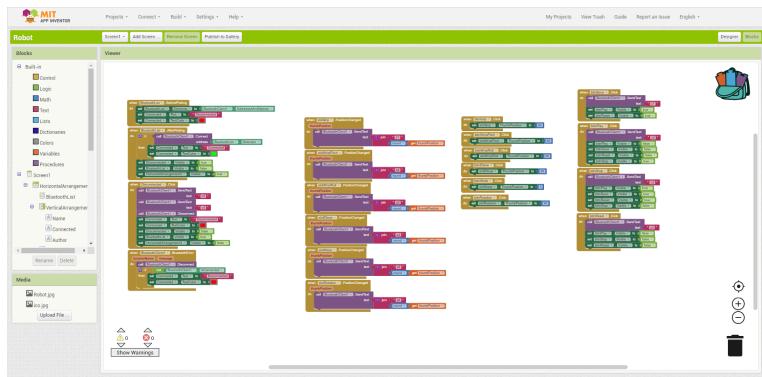
Rysunek 8: Połączenia elektryczne

DONE? Aplikacja

Aplikacja powstała w MIT App Inventor⁴ (Rysunki 9 i 10). Środowisko to oferuje prostotę w realizacji projektów. Nie wymaga żadnego doświadczenia od użytkownika. Programowanie odbywa się we własnym języku graficznym (duże podobieństwo do Scratch⁵). Kod dostarczonej aplikacji pokazano na Załącznikach 2, 3, 4 i 5. Na Rysunku 11 przedstawiono ostateczną szatę graficzną aplikacji.



Rysunek 9: MIT App Inventor - aplikacja

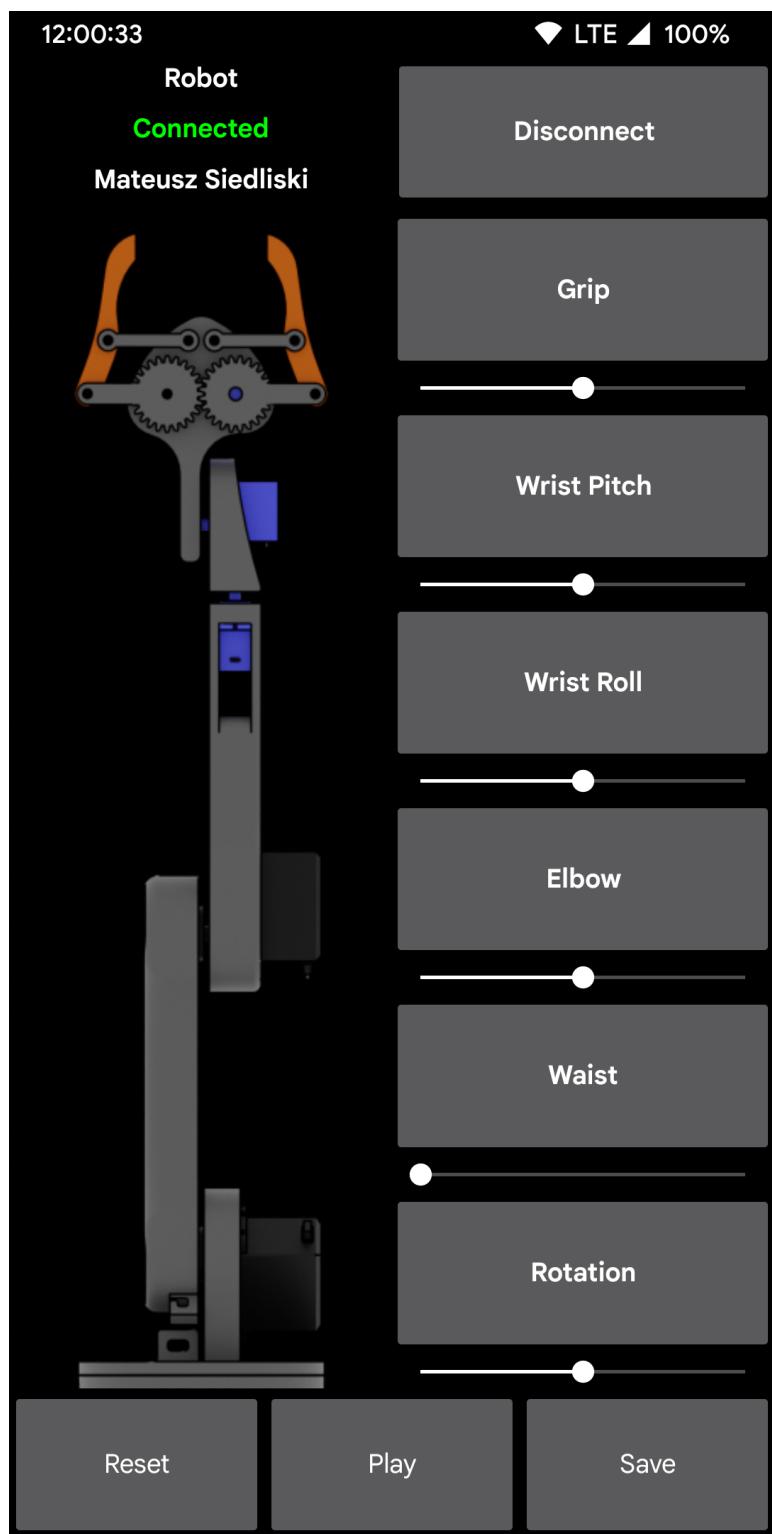


Rysunek 10: MIT App Inventor - kod

⁴Zintegrowane środowisko programistyczne do tworzenia aplikacji mobilnych

⁵Interpretowany wizualny język programowania

Ramię robota



Rysunek 11: Gotowa aplikacja

DONE? Protokół komunikacyjny

Komunikacja aplikacji mobilnej z mikrokontrolerem odbywa się poprzez protokół Bluetooth przy użyciu modułu HC-05. Każdy komunikat to odpowiednio sformatowany string. Komendy komunikacyjne przedstawiono w Tabeli 2. Przykładowe komunikaty:

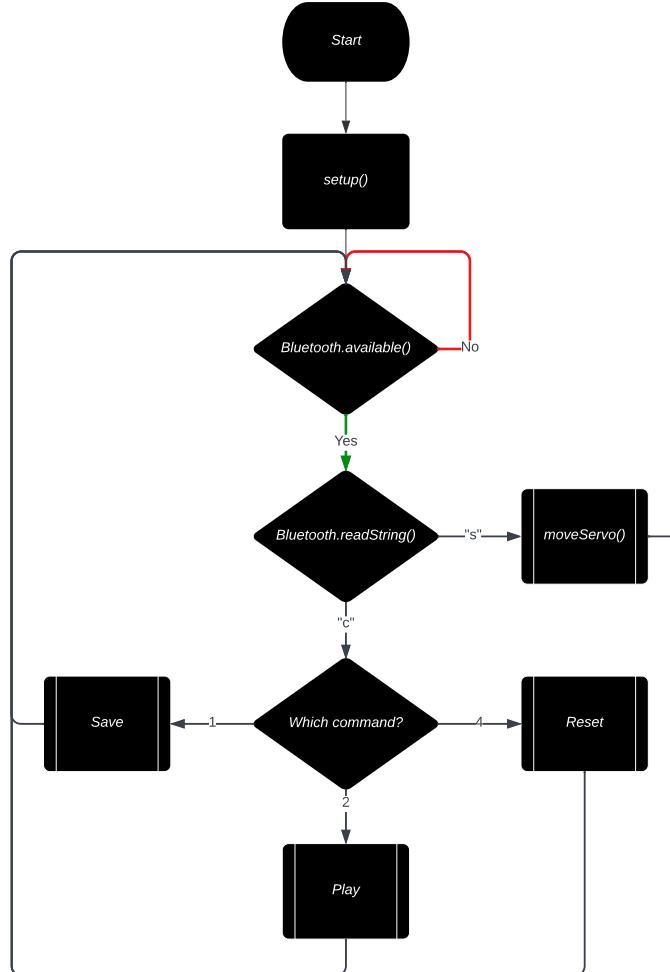
- „s1128\0“ - ustawienie serwomechanizmu 1 na pozycję 128
- „s564\0“ - ustawienie serwomechanizmu 5 na pozycję 64
- „c1\0“ - zapisanie aktualnej pozycji serwomechanizmów w pamięci do późniejszego odtworzenia
- „c4\0“ - wyczyszczenie zapisanej sekwencji ruchowej

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	x	Save - zapisanie aktualnej pozycji serwomechanizmów w pamięci do późniejszego odtworzenia
c	2	x	Play - odtwarzanie zapisanej sekwencji ruchowej
c	3	x	Stop - koniec odtwarzania zapisanej sekwencji ruchowej
c	4	x	Reset - wyczyszczenie zapisanej sekwencji ruchowej

Tabela 2: Protokół komunikacyjny

DONE? Kod mikrokontrolera

Kod sterujący działaniem mikrokontrolera powstał w Visual Studio Code⁶ przy pomocy PlatformIO⁷. Schemat blokowy kodu przedstawiono na rysunku 12⁸. Całość kodu źródłowego znajduje się w Załączniku 1.



Rysunek 12: Schemat blokowy kodu

⁶Darmowy edytor kodu źródłowego z kolorowaniem składni dla wielu języków, stworzony przez Microsoft o otwartym kodzie źródłowym

⁷ „A user-friendly and extensible integrated development environment with a set of professional development instruments, providing modern and powerful features to speed up yet simplify the creation and delivery of embedded products”[3]

⁸Schemat wykonano w programie Lucidchart

4.5 **TODO** Problemy w trakcie tworzenia sprzętu i aplikacji

W tym punkcie należy przedstawić jakie były problemy przed którymi stanęli autorzy projektu w trakcie jego realizacji i jak je rozwiązali.

5 **TODO** Podsumowanie

W ostatnim punkcie opisać co zostało wykonane, jakiej części założeń nie wykonano i dlaczego. Co można zrobić, by dany projekt poprawić i w jakim kierunku może pójść dalszy rozwój tego projektu.

6 WIP Literatura

- [1] How To Mechatronics. *DIY Arduino Robot Arm with Smartphone Control*. Data dostępu: 2022. URL: https://www.youtube.com/watch?v=_B3gWd3A_SI.
- [2] Tadeusz Szkodny. *Zbiór zadań z podstaw robotyki*. Gliwice: Wydawnictwo Politechniki Śląskiej, 2013. ISBN: 978-83-7880-162-7.
- [3] *PlatformIO*. Data dostępu: 2022. URL: <https://platformio.org>.

7 **WIP Załączniki**

Na prezentację (obronę) projektu trzeba przygotować prezentację w PowerPoint lub Prezi trwającą ok 12-15 minut Zwykle jest to ok. 20 slajdów. Te slajdy muszą zawierać tytuł projektu, autora, prowadzącego. Następnie plan prezentacji, najważniejsze osiągnięcia , podsumowanie. Slajdy muszą być czytelne, należy umieszczać rysunki i schematy, także nie powinno się na nich umieszczać zbyt dużo tekstu (tekst powinien mieć ok. 20-24pkt). W trakcie lub po prezentacji powinno się zademonstrować wykonany sprzęt i oprogramowanie. Można również zaprezentować w zamian tego film z działania urządzenia.

Przykładowa zawartość załącznika: Kody źródłowe (z komentarzami!!!) na CD spakowane. Wysłane mailem zawartość projektu do prowadzącego. Prezentacja. Wszystkie opisy układów, programów, narzędzi stosowanych w projekcie itp.

Załącznik 1: Kod mikrokontrolera

```
1 #include <Arduino.h>
2 #include <SoftwareSerial.h>
3 #include <Servo.h>
4
5 #define memory 50
6
7 const int servoNum = 6;
8
9 // Initial positions
10#define servo01235InitPos 90;
11#define servo4InitPos 0;
12
13// Creating servos
14 Servo servo[servoNum];
15
16// Defining BT
17 SoftwareSerial Bluetooth(D7, D8);
18
19// Servo positions
20 int servoPos[servoNum];
21
22// Servo previous positions
23 int servoPPos[servoNum];
24
25// Servo saved positions
26 int servoSPos[servoNum][memory];
27
28// Position in saved positions
29 int indexS = 0;
30
31// Received BT data
32 String dataIn = "";
33
34// Function prototypes
35 void moveServo(int whichServo, int PosServo);
36 void setMem();
37 bool checkPos(int pos);
38
39 void setup()
40 {
41     setMem(); // Setting initial positions
42
43     // Attaching servos
```

```
44  servo[0].attach(D6);
45  servo[1].attach(D5);
46  servo[2].attach(D4);
47  servo[3].attach(D3);
48  servo[4].attach(D2);
49  servo[5].attach(D1);
50
51 // Setting initial positions
52 for (size_t i = 0; i < servoNum; i++)
53 {
54     servo[i].write(servoSPos[i][0]);
55     servoPPos[i] = servoSPos[i][0];
56 }
57
58 // Initializing BT
59 Bluetooth.begin(38400);
60 Bluetooth.setTimeout(1);
61 delay(100);
62 }
63
64 void loop()
65 {
66     while (Bluetooth.available() <= 0) // Check BT
67     {
68         // Nothing to do
69     }
70
71     delay(1); // Wait for BT data
72     dataIn = Bluetooth.readString(); // Receive BT data
73
74     if (dataIn.startsWith("s")) // Servo position?
75     {
76         // Which servo?
77         String dataInServo = dataIn.substring(1, 2);
78         int SelectServo = dataInServo.toInt();
79
80         // Which position?
81         String dataInServoPos = dataIn.substring(2, dataIn.length());
82         int PosServo = dataInServoPos.toInt();
83
84         moveServo(SelectServo - 1, PosServo);
85     }
86     else if (dataIn.startsWith("c")) // Command?
```

```
87  {
88  // Which command?
89  String dataInFunc = dataIn.substring(1, 2);
90  int SelectFunc = dataInFunc.toInt();
91
92  switch (SelectFunc)
93  {
94
95  case 1: // Save
96
97  // Save all current servo positions
98  if (indexS < memory)
99  {
100    for (size_t i = 0; i < servoNum; i++)
101    {
102      servoSPos[i][indexS] = servoPos[i];
103    }
104    indexS++;
105  }
106
107  break;
108
109 case 2: // Play
110
111 // Repeat until Stop command
112 while (!dataIn.startsWith("c3"))
113 {
114   for (int j = 0; j < indexS; j++)
115   {
116
117     dataIn = Bluetooth.readString(); // Receive BT data
118     if (dataIn.startsWith("c3")) // Stop?
119     {
120       break;
121     }
122
123     // Move all servos simultaneously
124     while (checkPos(j)) // Check if all servos are in
125     posioton
126     {
127       for (size_t i = 0; i < servoNum; i++)
128       {
129         if (servoPos[i] > servoSPos[i][j])
130         {
```

```
130         servo[i].write(--servoPos[i]);
131     }
132     else if (servoPos[i] < servoSPos[i][j])
133     {
134         servo[i].write(++servoPos[i]);
135     }
136 }
137 }
138 }
139 }
140
141     break;
142
143 case 4: // Reset
144
145     setMem(); // Setting initial positions
146
147     break;
148 }
149 }
150 }
151
152 void moveServo(int whichServo, int PosServo)
153 {
154     servoPos[whichServo] = PosServo;
155
156     // Movement speed adjustment
157     int offset = 2;
158
159     if (whichServo <= 2)
160     {
161         offset = 30;
162     }
163
164     if (servoPos[whichServo] > servoPPos[whichServo])
165     {
166         for (int i = servoPPos[whichServo]; i <= servoPos[
167             whichServo]; i += offset)
168         {
169             servo[whichServo].write(i);
170         }
171         servo[whichServo].write(servoPos[whichServo]);
172     }
173     else if (servoPos[whichServo] < servoPPos[whichServo])
```

```
173 {  
174     for (int i = servoPPos[whichServo]; i >= servoPos[  
175         whichServo]; i -= offset)  
176     {  
177         servo[whichServo].write(i);  
178     }  
179     servo[whichServo].write(servoPos[whichServo]);  
180 }  
181 servoPPos[whichServo] = servoPos[whichServo];  
182 }  
183  
184 bool checkPos(int pos)  
185 {  
186     for (size_t i = 0; i < servoNum; i++)  
187     {  
188         if (servoPos[i] - servoSPos[i][pos] != 0)  
189         {  
190             return 1;  
191         }  
192     }  
193     return 0;  
194 }  
195  
196 void setMem()  
197 {  
198     // Setting initial positions  
199     for (size_t j = 0; j < memory; j++)  
200     {  
201         for (size_t i = 0; i < servoNum; i++)  
202         {  
203             servoSPos[i][j] = servo01235InitPos;  
204         }  
205         servoSPos[4][j] = servo4InitPos;  
206     }  
207     indexS = 0;  
208 }
```

Ramię robota

```
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 " c3 "
call BluetoothClient1 .SendText
text " c4 "
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 2: Kod aplikacji - Bluetooth

Ramię robota

```
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 3: Kod aplikacji - Przyciski

Ramię robota

```
when slidRotation .PositionChanged
  thumbPosition
do call BluetoothClient1 .SendText
  text [ join [ " s6 " ]
    round [ get thumbPosition ] ]
```

```
when slidWaist .PositionChanged
  thumbPosition
do call BluetoothClient1 .SendText
  text [ join [ " s5 " ]
    round [ get thumbPosition ] ]
```

```
when slidElbow .PositionChanged
  thumbPosition
do call BluetoothClient1 .SendText
  text [ join [ " s4 " ]
    round [ get thumbPosition ] ]
```

```
when slidWristRoll .PositionChanged
  thumbPosition
do call BluetoothClient1 .SendText
  text [ join [ " s3 " ]
    round [ get thumbPosition ] ]
```

```
when slidWristPitch .PositionChanged
  thumbPosition
do call BluetoothClient1 .SendText
  text [ join [ " s2 " ]
    round [ get thumbPosition ] ]
```

```
when slidGrip .PositionChanged
  thumbPosition
do call BluetoothClient1 .SendText
  text [ join [ " s1 " ]
    round [ get thumbPosition ] ]
```

Załącznik 4: Kod aplikacji - Slidery

Ramię robota

```
when [btnSave v].Click
do
  call [BluetoothClient1 v].SendText
    text [c1]
  set [btnPlay v].Visible to [true v]
  set [btnReset v].Visible to [true v]

when [btnPlay v].Click
do
  call [BluetoothClient1 v].SendText
    text [c2]
  set [btnPlay v].Visible to [false v]
  set [btnSave v].Visible to [false v]
  set [btnReset v].Visible to [false v]
  set [btnStop v].Visible to [true v]

when [btnStop v].Click
do
  call [BluetoothClient1 v].SendText
    text [c3]
  set [btnPlay v].Visible to [true v]
  set [btnSave v].Visible to [true v]
  set [btnReset v].Visible to [true v]
  set [btnStop v].Visible to [false v]

when [btnReset v].Click
do
  call [BluetoothClient1 v].SendText
    text [c4]
  set [btnPlay v].Visible to [false v]
  set [btnStop v].Visible to [false v]
  set [btnReset v].Visible to [false v]
```

Załącznik 5: Kod aplikacji - Komendy