
POLSKO-JAPOŃSKA AKADEMIA
TECHNIK KOMPUTEROWYCH

**WYDZIAŁ INFORMATYKI**

**KATEDRA METOD PROGRAMOWANIA**

**PROGRAMOWANIE APLIKACJI BIZNESOWYCH**

**Kamil Kacprzak**
s14004

# Zastosowanie technologii rzeczywistości rozszerzonej w biznesie, ze szczególnym uwzględnieniem rękawicy-kontrolera

**Praca inżynierska**
**dr Krzysztof Barteczko**

**Warszawa, Czerwiec, 2020**

# Spis treści


**1 Wstęp** **1**

**2 Zagadnienia** **2**
2.1 Wpływ kultury na rozwój technologii . . . . . 2
2.1.1 Kinematografia . . . . . 2
2.1.2 Świadome sny . . . . . 2
2.2 Rodzaje rzeczywistości . . . . . 2
2.2.1 XR . . . . . 2
2.2.2 VR . . . . . 2
2.2.3 AR . . . . . 2
2.2.4 MR . . . . . 2
2.3 Potencjał technologii . . . . . 2

**3 Technologia prezentowania kreowanej rzeczywistości** **3**
3.1 Okulary VR . . . . . 3
3.2 Smart-fon jako urządzenie do prezentacji . . . . . 3
3.3 Kontrolery i IoT . . . . . 3
3.3.1 Problemy kontrolerów . . . . . 3
3.3.2 Działanie czujników . . . . . 3
3.4 Dłoń jako kontroler . . . . . 3

**4 Rękawice jako kontrolery** **4**
4.1 Rozwój rękawicy jako kontrolera . . . . . 4
4.2 Podział rękawic-kontrolerów . . . . . 4
4.2.1 Klasyczne . . . . . 4
4.2.2 Nasadowe . . . . . 4
4.2.3 Egzoszkielety . . . . . 4

**5 Zastosowania w biznesie** **5**
5.1 Lotnictwo i symulatory . . . . . 5
5.2 Medycyna . . . . . 5

5.3 Architektura . . . . . 5
5.4 Rozrywka . . . . . 5

**6 Projekt rękawicy** **6**
6.1 Przegląd podzespołów użytych w projekcie . . . . . 6
6.1.1 Mikrokontroler . . . . . 7
6.1.2 Czujnik wygięcia . . . . . 9
6.1.3 Rezystor . . . . . 11
6.2 Budowa rękawicy . . . . . 13
6.3 Oprogramowanie mikrokontrolera . . . . . 17
6.4 Dalszy rozwój . . . . . 26
6.4.1 Problemy mikroprocesora . . . . . 26
6.4.2 Problemy budowy i czujników wygięcia . . . . . 27
6.4.3 Animacja modelu . . . . . 28
6.4.4 Błąd rotacji . . . . . 29
6.4.5 Problem obliczania przesunięcia . . . . . 30
6.5 Podsumowanie projektu . . . . . 30

**7 Podsumowanie** **32**

# Spis rysunków

6.1 Opis wejść oraz wyjść Arduino Nano 33 BLE . . . . . 8
6.2 Sensor wygięcia własnego wykonania . . . . . 11
6.3 Oznaczenia rezystorów . . . . . 12
6.4 Układ dzielnika napięcia . . . . . 13
6.5 Poglądowy układ kontrolera przedstawiający sposób podłączenia dzielnika napięcia . . . . . 14
6.6 Efekt końcowy rękawicy-kontrolera . . . . . 15

**Streszczenie**

**Abstract**

# Rozdział 1

# Wstęp

cel i zarys, wytłumaczenie tytułu

# Rozdział 2

# Zagadnienia

czym jest rzecywistość i skąd wiemy co jest rzeczywiste

## 2.1 Wpływ kultury na rozwój technologii

### 2.1.1 Kinematografia

### 2.1.2 Świadome sny

## 2.2 Rodzaje rzeczywistości

wytłymaczyć skróty i używanie angielskich nazwa jako standard

### 2.2.1 XR

### 2.2.2 VR

### 2.2.3 AR

### 2.2.4 MR

## 2.3 Potencjał technologii

# Rozdział 3

# Technologia prezentowania kreowanej rzeczywistości

## 3.1 Okulary VR

## 3.2 Smart-fon jako urządzenie do prezentacji

## 3.3 Kontrolery i IoT

### 3.3.1 Problemy kontrolerów

### 3.3.2 Działanie czujników

IMU

## 3.4 Dłoń jako kontroler

historia, pierwsze firmy, popularne rozwiazania

# Rozdział 4

# Rękawice jako kontrolery

czym są i po co
jak i dlaczego

## 4.1 Rozwój rękawicy jako kontrolera

Pierwsze firmy, upadek, wzrost i spadek zainteresowania

## 4.2 Podział rękawic-kontrolerów

### 4.2.1 Klasyczne

### 4.2.2 Nasadowe

### 4.2.3 Egzoszkielety

# Rozdział 5

# Zastosowania w biznesie

## 5.1 Lotnictwo i symulatory

## 5.2 Medycyna

## 5.3 Architektura

## 5.4 Rozrywka

# Rozdział 6

# Projekt rękawicy

Istotą poniższego rozdziału jest pokazanie użytych w projekcie podzespołów i technologii oraz lepsze zrozumienie powodów dla których to właśnie te produkty zostały wybrane. Po zapoznaniu się z motywem, zostanie szczegółowo opisana specyfikacja tych produktów a także sposób ich poskładania w spójną całość, co sprawiło pewne problemy względem oryginalnego szkicu - próby oraz efekty rozwiązywania tych problemów również zostaną opisane w tym rozdziale. Po nakreśleniu podstawowych założeń projektu zostanie zaprezentowana finałowa wersja, a także szczegółowo zostanie omówiony kod rękawicy-kontrolera, który jest obsługiwany przez mikrokontroler i stanowi najważniejszą część tego projektu.

proof read

Opis: BLE, Rotacja, przesunięcie i animacja, przy opisywanych sensorach

## 6.1 Przegląd podzespołów użytych w projekcie

Jak dowiedzieliśmy się z rozdziału **??** dotyczącego podstawowych komponentów rękawic, kluczowym dla powodzenia projektu jest ustalenie następujących pozycji:

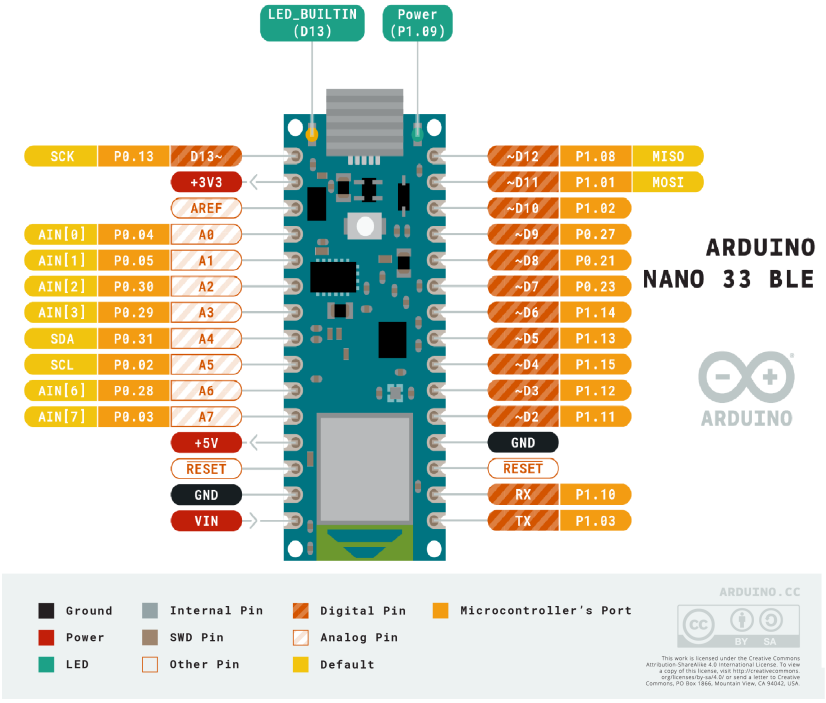
- orientacji dłoni względem punktu początkowego
- położenia względem punktu początkowego
- moment i stopień zgięcia palców

W tym celu należy zebrać informacje z czujników, a następnie wszystkie te informacje należy przesłać do pożądanego urządzenia. Elementem które pozwala to osiągnąć w tym projekcie jest mikrokontroler Arduino nano 33 BLE, który odpowiada za dostarczenie informacji z żyroskopu, akcelerometru a także czujników wygięcia. Zasady działania pierwszych dwóch zostały opisane z podrozdziale **??**. Natomiast w poniższych podrozdziałach zostanie opisane rozwiązanie zastosowane do odczytu położenia palców, zasada działania przy wykorzystaniu rezystorów oraz sposób połączenia wszystkich wspomnianych elementów w jeden finałowy kontroler.

### 6.1.1 Mikrokontroler

Jak przed chwilą wspomniano, w projekcie wykorzystywana jest płytka od Arduino, która nosi nazwę Nano 33 BLE. Jest to małych rozmiarów płytka o wymiarach 45 x 18 mm, pozwalająca na wysoką wydajność przy jednoczesnym małym poborze prądu, co zapewnia użyty mikrokontroler nRF52480 o taktowaniu 64 MHz. Do dyspozycji mamy również pamięć RAM o pojemności 256 kB oraz pamięć Flash o pojemności 1 MB. IMU które zostało zamontowane na płytce to LSM9DS1, które obsługuję akcelerometr, żyroskop oraz magnetometr w trzech osiach. Więcej infomracji na temat IMU jest przedstawione w części 6.3. Warto na wstępie zauważyć że Nano 33 BLE pracuje domyślnie wyłącznie z napięciem 3,3 V, w związku z czym nie należy podłączać bezpośrednio zasilania o większym napięciu. W celu podłączenia zasilania 5 V należy zlutować zworkę znajdującą się pomiędzy pinami RDT oraz A7 - temat ten nie zostaje jednak poruszony w tej pracy, ponieważ na potrzeby projektu używane jest zasilanie poprzez złącze micro USB które to również jest obsługiwane. Płytka ta posiada wiele użytecznych sensorów, jednak na potrzeby tej pracy została wybrana z powodu wbudowanej inercyjnej jednostki pomiarowej, dzięki czemu można było uprościć konstrukcję

oraz zmniejszyć ilość połączeń na rękawicy, wbudowanego modułu Bluetooth - a w tym przypadku modułu Bluetooth Low Energy obsługiwanego w standardzie 5.0, a to wszystko w przystępnej cenie co również było jednym z kryteriów przy tworzeniu tego projektu. Płytkę w momencie tworzenia tej pracy można kupić za 119 zł. Wartym uwagi jest fakt możliwości zakupu płytki bez wyprowadzonych złącz, co w przypadku opisywanego projektu pozwoli na zmniejszenie wymiarów oraz większą swobodę montażu [4].




Rysunek 6.1: Opis wejść oraz wyjść Arduino Nano 33 BLE
Źródło: `https://content.arduino.cc/assets/Pinout-NANOble_latest.pdf`

Z najważniejszych elementów układu płytki należy wiedzieć że posiada ona dwie diody po dwóch stronach portu micro USB - zielona indykuje podłączone zasilanie, natomiast pomarańczowa zaczyna mrugać gdy jest przesyłany kod do mikrokontrolera. Oprócz tego do dyspozycji są dwa piny wyjściowe zasilające o

napięciu 3,3 V oraz 5 V, jeden pin zasilający wejściowy, którego ograniczenia zostały wspomniane w poprzednim paragrafie, a także dwa piny uziemiające, po jednym z każdej strony płytki. Posiada ona piny zarówno analogowe jak i cyfrowe, jednak na potrzeby tego projektu zostały użyte jedynie piny analogowe, których do dyspozycji jest aż osiem umiejscowionych po jednej stronie, przy czym warto zwrócić uwagę że piny A4 oraz A5 używane są jako magistrala I2C w związku z czym zalecane jest nie stosowanie tych wejść analogowych. Szczegółowy opis wejść/wyjść płytki przedstawiony jest na grafice 6.1. W tym projekcie wykorzystano zasilanie poprzez złącze micro USB, wyjście o napięciu 3,3 V w celu uzyskania odczytów z czujników wygięcia na pinach analogowych A0, A1, A3, A6 oraz A7, o których zostanie więcej powiedziane w punkcie 6.2, a także uziemienie znajdujące się po tej samej stronie płytki.

### 6.1.2 Czujnik wygięcia

Kluczowym dla działania kontrolera jest możliwość określenia pozycji palców względem dłoni. Ma to wiele zastosowań zarówno wizualnych jak i praktycznych. Ważne jest aby odwzorować świat rzeczywisty tak dokładnie jak to możliwe - im lepsze odwzorowanie tym bardziej zmysły użytkownika zostaną oszukane, podwyższając komfort użytkowania technologii wirtualnej rzeczywistości. Za stroną praktyczną natomiast przemawia możliwość śledzenia palców w celu dokładnego ich użycia w stworzonym środowisku np. do ściskania i podnoszenia obiektów czy też korzystania z klawiatury wirtualnej. Jak wspomniano w rozdziale **??** dotyczącym komponentów komercyjnych rękawic - wśród znanych producentów na rynku, decyduję się na użycie wielu inercyjnych jednostek pomiarowych, na podstawie których są w stanie dokładnie określić położenia każdej części palca, bądź też nie aż tak popularne rozwiązanie, które wykorzystujące specjalistyczne sensory służące do pomiaru stopnia wygięcia czujnika względem

pozycji prostej. Czujnik ten po podłączeniu do prądu zwiększa swój opór wraz ze zwiększonym stopniem odchylenia. Oba te rozwiązania pomimo wysokiej dokładności pomiarów nie są rozwiązaniami tanimi. W związku z tym na potrzeby stworzenia taniego kontrolera, należało znaleźć rozwiązanie bardziej przystępna a jednocześnie pozwalające na osiągnięcie tego samego celu.

Aby osiągnąć postawione założenia zostały skonstruowane czujniki wygięcia w domowych warunkach. Rozwiązanie to jest często używane do osiągnięcia pomiaru stopnia wygięcia bez konieczności wydawania ponad 100 zł na jeden czujnik [5]. Jest ono stosunkowo proste w założeniach i wymaga jedynie dwóch kluczowych elementów. Tkaniny przewodzącej o specjalnych właściwościach oraz dwóch przewodników po obu stronach materiału. Z jednej strony zostanie podłączone napięcie z drugiej natomiast uziemienie. Ważnym jest aby połączenia te się ze sobą nie stykały w żadnym punkcie a jedynie zostały nałożone na siebie, z tkaniną ściśniętą pomiędzy nimi - dzięki temu mamy pewność że odczyty które otrzymamy będą prawidłowe. Pozostaje odpowiedzieć na pytanie jakiego rodzaju materiał należy wykorzystać. Na rynku znajdziemy wiele rodzajów materiałów które zmieniają swój opór w zależności od spełnienia takich kryteriów jak nacisk, temperatura, rozciągnięcie czy też właśnie zgięcie materiału. Pomimo próby uzyskania materiału który zmienia swój opór w zależności od rozciągnięcia, co pozwoliłoby na skonstruowanie części na palce rękawiczki z tego materiału, zapewniając dokładniejszy i bardziej estetyczny efekt końcowy, w momencie projektowania rękawicy był on jedynie możliwy do sprowadzenia ze stanów, co nie było najtańszym rozwiązaniem. W związku z tym zdecydowano się na użycie foli Velostat która jest czuła na nacisk or zginanie [2]. W roli przewodnika wybrano nić przewodzącą, która zapewniła potrzebną elastyczność oraz możliwość przymocowania poszczególnych elementów przy jednoczesnym zapewnieniu funkcjonalności. Elementy te zostały sklejone na kawałku taśmy samoprzylepnej oraz do-

datkowo sklejone przy brzegu aby nić nie wyśliznęła się w trakcie korzystania z czujnika. Efekt końcowy jest widoczny na zdjęciu 6.2. W celu otrzymania pomiarów wszystkich palców zostało wykonanych pięć takich sensorów, o szerokości 15 mm; dwa o długości 8 cm, dwa o długości 10 cm a także jeden 11 cm, w celu jak najlepszego dopasowania względem miejsca na palce na zakupionej rękawicy do której sensory zostaną przymocowane, co można zobaczyć na zdjęciu 6.6.

Rysunek 6.2: Sensor wygięcia własnego wykonania

### 6.1.3 Rezystor

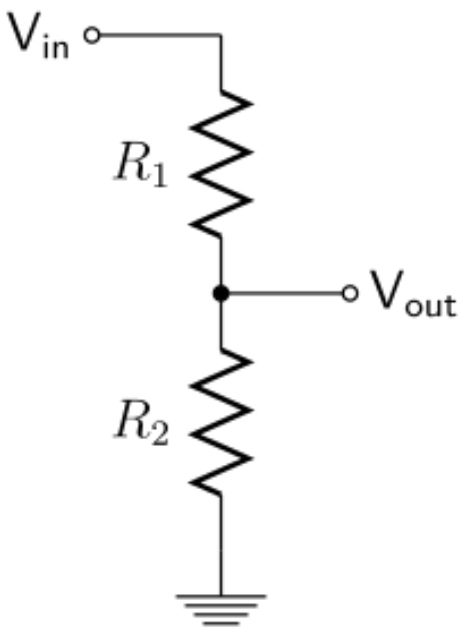
Rezystor, potocznie zwany opornikiem, jest to prosty element elektroniczny, posiadający jedynie wyjścia z dwóch stron elementu łączącego. Element ten tworzy opór, powodując ograniczenie przepływającego przez niego prądu gdy jest włączony do obwodu szeregowo. Opór ten jest mierzony w omach. Istotną informacją jest fakt że nadmiar prądu jest zamieniany przez opornik na energie cieplną, a także brak zdefiniowanego kierunku - co oznacza że działa on niezależnie od sposoby zanotowania go w układzie.

Pomimo swojej prostoty budowy i zastosowania, dla danego projektu ważne

jest aby wybrać odpowiednie rezystory. Podstawową wartością na którą należy zwrócić uwagę jest rezystancja. Rezystancję podaje się w omach i można spotkać na rynku zakres od miliomów do megaomów. Spośród dostępnych rodzajów rezystorów w projekcie zostały użyte rezystory THT ( z ang. Through-Hole Technology) - czyli tak zwane rezystory do montażu przewlekanego. W tym rodzaju oporników rezystancja jest ilustrowana poprzez kolorowe paski umieszczone wokół oporu, co pozwala odczytać ich wartość według ilustracji 6.3. Alternatywą do tego sposobu jest podłączenie rezystora pod miernik elektryczny ustawiony w tryb pomiaru oporu [8]. Element ten jest kluczowy w celu ograniczenia przepływu prądu w obwodzie rękawicy, co pozwala na monitorowanie oporu wytwarzanego poprzez czujnik wygięcia.

| Kolor | Wartość | | Mnożnik | Tolerancja ± % | Współczynnik temp. ± ppm/K |
|---|---|---|---|---|---|
| | 1 pasek | 2 pasek | 3 pasek | 4 pasek | Ostatni pasek |
| czarny | | 0 | x 1 Ω | | 250 |
| brązowy | 1 | 1 | x 10 Ω | 1 | 100 |
| czerwony | 2 | 2 | x 100 Ω | 2 | 50 |
| pomarańczowy | 3 | 3 | x 1 kΩ | | 15 |
| żółty | 4 | 4 | x 10 kΩ | | 25 |
| zielony | 5 | 5 | x 100 kΩ | 0,5 | 20 |
| niebieski | 6 | 6 | x 1 MΩ | 0,25 | 10 |
| fioletowy | 7 | 7 | x 10 MΩ | 0,1 | 5 |
| szary | 8 | 8 | x 100 MΩ | 0,05 | 1 |
| biały | 9 | 9 | x 1 GΩ | | |
| złoty | | | 0,1 Ω | 5 | |
| srebrny | | | 0,01 Ω | 10 | |
| brak | | | | 20 | |

Rysunek 6.3: Oznaczenia rezystorów
Źródło: `https://sites.google.com/site/informatykaunijna/home/poszczegolne-czesci/rezystor`




Rysunek 6.4: Układ dzielnika napięcia
Źródło: https://www.wikiwand.com/en/Voltage_divider

Sposób działania układu, nazywanego dzielnikiem napięcia jest pokazany na rysunku 6.4, oraz wyraża się wzorem

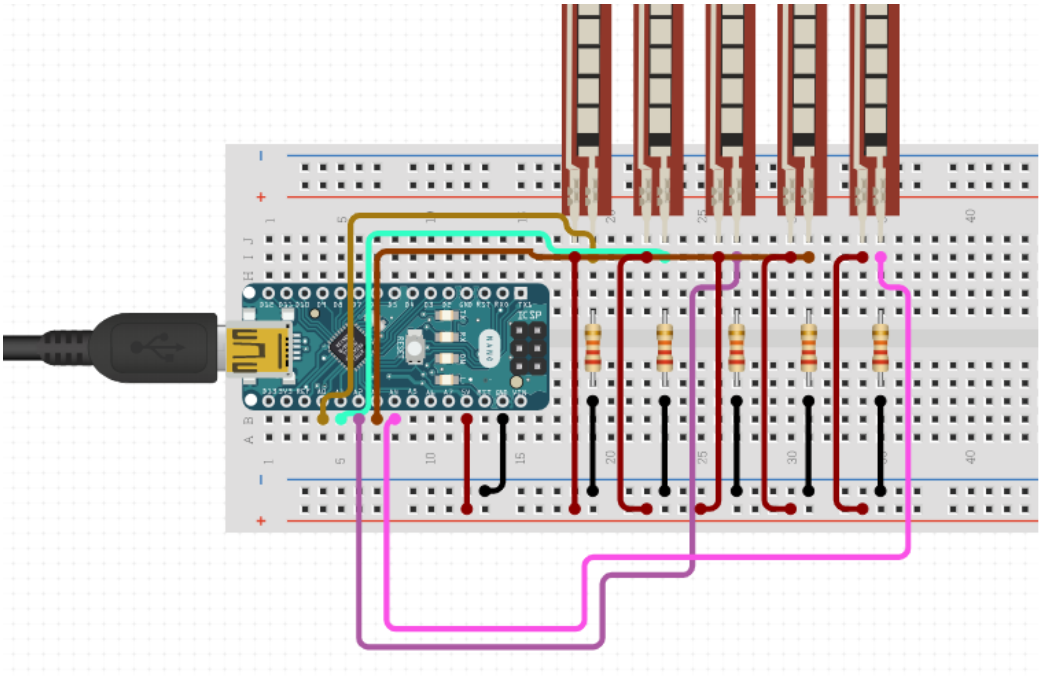
$$\nu_{out} = \nu_{in} \left[ \frac{R_2}{R_1 + R_2} \right]$$

Wzór ten podaje napięcie wyjściowe $\nu_{out}$, które równa się napięciu wejściowemu $\nu_{in}$ przeskalowanemu przez stosunek rezystorów. W opisywanym przypadku jest to stosunek zastosowanego rezystora $4.7k\Omega$ wyrażonego we wzorze poprzez $R_2$, do sumy tego rezystora wraz z oporem wytwarzanym poprzez czujnik wygięcia $R_1$ - który jak opisano w podrozdziale 6.1.2 jest zmienny. Oznacza to że im bardziej czujnik wygięcia jest zgięty, wytwarza on większy opór a co za tym idzie napięcie wyjściowe spada. Miara ta obrazuje jak bardzo palec jest zgięty i jest możliwa do uzyskania właśnie dzięki zastosowaniu układu dzielnika napięcia [1].

## 6.2 Budowa rękawicy

W poniższej sekcji zaprezentowano sposób w jaki zostały złączone wszystkie elementy rękawicy wspomniane w sekcji 6.1, aby była gotowa na oprogramowanie mikrokontrolera tworząc finałowy produkt. W tym celu została wybrana rękawica budowlana o grubych niciach ze ściągaczem wokół nadgarstka w celu zapewnienia komfortu, jak i precyzji położenia. Wybór ten również jest uzasadniony faktem początkowego planu dotyczącego wszycia materiału bezpośrednio w ręka-

wice jak i elastyczności które zapewnia grubsza rękawica. Tak jak wspomniano w podsekcji 6.1.2, do połączenia elementów została wykorzystana nić przewodząca. Dzięki grubym włóknom odstępu pomiędzy nićmi przewodzącymi prąd mogły być mniejsze, bez obawy przed spięciami w trakcie poruszania ręką. Dla tego projektu kontroler jest budowany dla lewej dłoni. Układ przewodów kontrolera jest zobrazowany na rysunku 6.5. Jest to jedynie obraz poglądowy, przedstawiony na płytce prototypowej a szczegółowy opis połączeń kontrolera zostanie opisany poniżej.



Rysunek 6.5: Poglądowy układ kontrolera przedstawiający sposób podłączenia dzielnika napięcia
Źródło: `https://www.circuito.io/app?components=514,8606,8606,8606,8606,11022`

Mikroprocesor został umieszczony na wysokości centralnej części dłoni, przy kciuku, skierowany złączem USB na zewnątrz, pozwalając na łatwe podłączenie przewodu zasilającego jak i również ustawienie wejść analogowych w stronę

Rysunek 6.6: Efekt końcowy rękawicy-kontrolera

palców lewej dłoni. Z tej strony płytki Arduino będziemy też korzystać - zarówno przy wyprowadzaniu napięcia jak i uziemienia. Montaż samej płytki nie stanowi żadnego problemu ze względy na cztery otwory w rogach pozwalające na mocowanie, do którego została użyta zwykła nić do szycia. Aby stworzyć obwód wychodzący od pinu GND (z ang. Ground) czyli właśnie uziemienia, należy najpierw poprowadzić go przez rezystory. W tym celu wyjścia oporników zostały wygięte w pętle oraz zalutowane aby w łatwy sposób można było je przyszyć do rękawicy. Aby dodatkowo ułatwić sobie to zadanie - dodatkowo zostały one przyklejone bezpośrednio do materiału na bardzo małą ilość kleju. Umiej-

scowienie rezystorów jest po zewnętrznej wierzchniej stronie dłoni, zaczynając się na wysokości kontrolera, zmierzając szeregowo w stronę nadgarstka. W ten oto sposób nić przewodząca została poprowadzona na zewnętrzną część dłoni a następnie poprzez wszystkie pętle rezystorów, kończąc na ostatnim. Z drugiej strony rezystora dochodzi natomiast nić pochodząca z czujnika oporu. W tym celu podczas tworzenia czujników oporu pozostawiono 25 cm długości nici, co pozwoliło na przymocowanie sensorów, a następnie połączenie obwodu. Sensory zostały zaopatrzone w krótkie, 1-2 cm długości kawałki taśmy elastycznej która została przyklejona na czubku każdego z sensorów pozwalając na elastyczny ruch sensora bez zerwania nici. Taśma ta została przyszyta w czubkach palców co dało punkt zaczepu dla sensorów. W tym momencie pozwoliło to na wygodne używanie nici wychodzących z sensorów w celu zamknięcia obwodu. Z powodu dużego zagęszczenia nici, które nie mogą się ze sobą stykać podczas ruchu ręki, ważnym dla projektu było naprzemienne używanie przestrzeni na rękawicy od spodu jak i od góry. Dzięki temu nici uziemienia oraz napięcia mogą się krzyżować nie zaburzając przy tym odczytów z sensorów. W celu jak najlepszego zagospodarowania przestrzenią wokół sensorów, uziemienie kciuka oraz małego palca zostały umiejscowione na nicie wychodzącej z prawej strony sensora, natomiast napięcie po lewej stronie. Dla pozostałych palców ustawienie to jest odwrotne. Mając to na uwadze została poprowadzona nić od kciuka poprzez wyjście 3.3 V na płytce a następnie wzdłuż knykci rękawicy, pozwalając nicią odpowiedzialnym za napięcie w odpowiednich sensorach na przyczepienie się w celu poboru napięcia tworząc tym samym dodatnią stronę układu. Nici wychodzące z sensorów które natomiast nie zostały do tej pory użyte, zostały przeszyte poprzez wyjścia analogowe a następnie podłączone kolejno do odpowiadających im rezystorów. Wyjścia odpowiadające każdemu z palców zostały opisane w tabeli poniżej.

| Palec | Wyjście analogowe |
|---|---|
| Kciuk | A3 |
| Wskazujący | A0 |
| Środkowy | A1 |
| Serdeczny | A6 |
| Mały | A7 |

Sposób w jaki zostały dobrane wyjścia jest podyktowany zaleceniami o nie korzystaniu z wyjść analogowych A4 oraz A5 oraz pozostawieniu przestrzeni wokół wyjścia A3 przeznaczonego na kciuka, ponieważ jako jedyne połączenie musiało najpierw zmierzać w kierunku palców a następnie w dół w celu połączenia z rezystorem. Efekt końcowy pracy przedstawia zdjęcie 6.6.

## 6.3 Oprogramowanie mikrokontrolera

Do tej pory została opisana teoria omawianego kontrolera, elementy które zostały użyte w projekcie a także sposób ich połączenia. W niniejszej sekcji przedstawiony jest kod programu który został napisany w środowisku programistycznym oraz języku o tej samej nazwie - Arduino. Język ten poza drobnymi zmianami opiera się na języku C/C++, a jego pełny kod źródłowy jest dostępny na platformie Github [6]. Aby rozpocząć pracę z wybranym produktem od Arduino należy wykonać pewne czynności przygotowywacze, takie jak instalacja sterowników płytki dla danego systemu czy sposób obsługi w samym środowisku Arduino. Czynności te zostały opisane na stronie producenta, w związku z czym nie zostaną one szczegółowo opisane [15]. Po spełnieniu wszystkich wymagań, został przygotowany program który został napisany oraz wgrany do mikrokontrolera, który można podzielić na trzy części:

- Deklaracje
- Ustawienie mikrokotrnolera

- Główna pętla programu

których cel oraz opis został przedstawiony poniżej.

W pierwszej kolejności zostanie przedstawiona część deklaracji, w której to dołączono wymagane biblioteki dla poprawnej obsługi wszystkich sensorów. Pierwszą biblioteką która została dołączona do programu jest *Arduino_LSM9DS1.h*. Biblioteka ta jest odpowiedzialna za obsługę IMU która przekazuje dane poprzez I2C do mikroprocesora. Biblioteka ta zajmuję się obsługą połączenia jak i kalibracja całego modułu. Inicjalizacja wartości poprzez biblitoekę wygląda w następujący sposób [13]

| Sensor | Zakres | Częstotliwość |
|---|---|---|
| Akcelerometr | $[-4, +4]g - / + 0.122mg$ | $104Hz$ |
| Żyroskop | $[-2000, +2000]dps + / - 70mdps$ | $104Hz$ |
| Magnetometr | $[-400, +400]uT + / - 0.014uT$ | $20Hz$ |

Tak jak wspomniano w sekcji 6.1.1 do połączenia bezprzewodowego Arduino Nano 33 BLE wykorzystuje moduł Bluetooth Low Energy, w związku z czym w części deklaracji dołączamy przeznaczoną do tego bibliotekę o nazwie *ArduinoBLE.h*. Biblioteka ta pozwala na dostęp oraz sterowanie modułem BLE ( z ang. Bluetooth Low Energy), i na potrzeby projektu zostaną opisane używane elementy z poniższych klasy których znajomość jest wymagana w celu zrozumienia napisanego programu. Klasy te to

- BLEService
- BLECharacteristic
- BLEDescriptor
- BLE
- BLEDevice

Szczegóły na temat zasad działania BLE są podane w sekcji **??**, w tej sekcji jedynie zostanie opisana struktura tych klas. *BLEService* umożliwia nawiązanie połączenia z innym urządzeniem obsługującym bluetooth i jako parametr przyjmuje UUID - jest to jedyny paramater jaki należy podać i w ten sposób zostaje stworzony nowy serwis BLE dostępny pod tym identyfikatorem. Klasa *BLECharacteristic* tworzy nową cechę którą należy przypisać do danego serwisu. Cecha ta może zostać zadeklarowana jako cecha wybranego z pośród dostępnych typów w Arduino bądź jako uniwersalna poprzez podstawową klasę *BLECharacteristic*. Klasa ta przyjmuje trzy wymagane parametry: UUID, właściwości cechy oraz wartość. Wartość może zostać zadeklarowana jako podany ciąg znaków, bądź też poprzez określenie rozmiaru danych i wartość początkową. Wartość początkowa nie musi być podana w trakcie deklaracji. Ważnym elementem tej klasy jest określenie jej właściwości. W zależności od przeznaczenia mamy do wybory następujące opcje, które mogą być ze sobą łączone: BLEBroadcast, BLERead, BLEWriteWithoutResponse, BLEWrite, BLENotify, BLEIndicate. W celu wysyłania danych podczas gdy dane te się zmienią cechy w prezentowanym programie przyjęły dwie wartości BLE: read ( z ang. czytać) oraz notify (z ang. powiadomić). Klasa *BLEDescriptor* jest niejako klasą pomocniczną. Wartości tej klasy przypisuję się do danej cechy w celu lepszej obsługi serwisy oraz łatwiejszego rozpoznawania w przypadku pracy ze skomplikowanymi serwisami. Standardowo należy podać UUID danego deskryptora jako parametr, a także jego wartość jako ciąg znaków, bądź wartość w postaci tablicy bajtów oraz maksymalnego rozmiaru danych. Są to podstawowe klasy znajdujące się w części deklaracji których użycie zostały przedstawione na listingu 6.1 prezentującym wycinek części deklaracji [14]. Pozostałe dwie klasy zostaną opisane w części głównej pętli programu gdzie zostanie również pokazane ich zastosowanie.

```
#include <Arduino_LSM9DS1.h>
#include <ArduinoBLE.h>
BLEService imuService("1101");

BLECharacteristic imuAccChar("2101", BLERead | BLENotify
   ,12);
BLECharacteristic imuGyroChar("2102", BLERead | BLENotify,
   12);
BLECharacteristic fingersChar("2103", BLERead | BLENotify,
   20);

BLEDescriptor     imuAccDescriptor("3101",(byte*)"Acc
   Descriptor",15);
BLEDescriptor     imuGyroDescriptor("3102",(byte*)"Gyr
   Descriptor",15);
BLEDescriptor     fingersDescriptor("3103",(byte*)"Fin
   Descriptor",15);
```

Listing 6.1: Część deklaracji programu mikrokontrolera

Następną częścią programu jest ustawienie mikrokontrolera oraz sprawdzenia poprawności działania sensorów. Gdy proces ten zakończy się powodzeniem zostanie uruchomiona główna część programu - funkcja *loop()*, która wykonuję się nieprzerwania pozwalając kontrolować pracę naszego kontrolera. Za część związaną z inicjalizacją jest odpowiedzialna funkcja *setup()*. Funkcja ta jest wywoływana tylko raz za każdym razem gdy płytka jest podłączona do prądu bądź zostanie zresetowana [3]. W funkcji tej wywoływana jest statyczna metoda begin(z ang. rozpocząć) na klasie *IMU* która należy do biblioteki *Arduino_LSM9DS1.h*, i to właśnie ta funkcja pozwala na incjalizacje wspominanych sensorów wchodzących w skład jednostki pomiarowej - program zapętli się w tym miejscu jeśli funkcja ta zwróci błąd, poprzez rozpoczęcie pętli z warunkiem zawsze prawdziwym, co przedstawia listing 6.2 [13].

```
  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }
```

```
16
17  if (!BLE.begin()) {
18    Serial.println("Failed to start Bluetooth!");
19    while (1);
20  }
```

Listing 6.2: Inicjalizacja IMU oraz BLE

Jeżeli sensory działają poprawnie, sprawdzana jest możliwość połączenia poprzez moduł bezprzewodowy. W tym celu wykorzystywana jest statyczna klasa *BLE*, która odpowiada za włączenie modułu Bluetooth oraz jego ustawienia. Tak jak w przypadku klasy *IMU*, program zapętli się w tym miejscu zwracając błąd jeżeli połączenie nie będzie możliwe. W przypadku powodzenia program rozpoczyna łączenie zadeklarowanych elementów, takich jak serwis, cechy oraz deskryptory tych cech. Klasa *BLE* posiada funkcje pozwalającą ustawić nazwę dla naszego połączenia, jej adres a przede wszystkim gdy wszystkie elementy są już gotowe wywołuję metodę *advertise()*, która pozwala na połączenie się z naszym serwisem. Na listingu 6.3 pokazane jest szczegółowa konfiguracja klasy BLE oraz serwisu a także odpowiednie użycie funkcji dla jednej z cech, które należy powtórzyć dla każdej dodawanej cechy z odpowiednimi wartościami.

```
21  BLE.setLocalName("VrGlove");
22  BLE.setAdvertisedService(imuService);
23
24  imuService.addCharacteristic(imuGyroChar);
25  imuGyroChar.addDescriptor(imuGyroDescriptor);
26  imuGyroChar.writeValue((byte)0x01);
27
28  BLE.addService(imuService);
29  BLE.advertise();
```

Listing 6.3: Obsługa serwisu przy użycia klasy BLE

W ten oto sposób kończy się funkcja *setup()* i zostaje wywołana kolejna funkcja która od tej pory będzie odpowiedzialna za pracę kontrolera - *loop()*. W funkcji tej zostaje zadeklarowana ostatnia z wymienionych klas do obsługi Bluetooth. Klasa

ta nazywa się *BLEDevice* i jest bezpośrednio powiązana z urządzeniem które jest aktualnie podłączone. Dopóki nie ma podłączonego do kontrolera urządzenia, żadne czynności nie zostają podjęte. W momencie uzyskania danych z modułu łączności o nawiązanym połączeniu zostaje uruchomiona pętla, funkcjonująca tak długo jak połączenie to nie zostanie przerwane. Fragment kodu znajduję się na listingu 6.4 [14]. Niestety w trakcie pracy z urządzeniem odkryto błąd związany z rozłączeniem się centrali. Problem pojawia się gdy urządzenie zostaję rozłączone z mikrokontrolerem - w tym momencie klasa *BLE* nie zawsze zgłasza informację o rozłączeniu, myśląc że urządzenia cały czas są podłączone. Problem został zauważony, jednak nie jest rozwiązany na stan z maja 2020 roku [12].

```
30  BLEDevice central = BLE.central();
31  if (central) {
32    Serial.print("Connected to central: ");
33    Serial.println(central.address());
34  }
35   while( central.connected()){
36        [...]
37    }
```

Listing 6.4: Oczekiwanie na połączenie z urządzeniem przez mikrokontroler

Gdy wszystkie dotychczasowo opisane elementy programu nie zgłoszą problemów, następuje ostatnia faza którą jest zbieranie i przesyłanie danych. W pierwszej kolejności sprawdzany jest warunek czy jednostka pomiarowa ma dostęp do nowych danych. W przypadku tej aplikacji dane które zostały poddane analizie pochodzą z żyroskopu oraz z akcelerometru i są wyrażone jako zmienne x,y oraz z typu *float*. Z powodu naturalnych drgań dłoni dane te ciągle się zmieniają w mikro skali która nie ma wpływu na efekty, niemniej jednak możemy założyć że gdy mikroprocesor nie jest przymocowany do stałego obiektu, dane te będą dostarczane z częstotliwością pracy sensorów. Za odczyt danych z żyroskopu i akcelerometru odpowiadają odpowiednio funkcje *readGyroscope(x,y,z)* oraz *re-*

*adAcceleration(x,y,z)* z klasy *IMU* które przypisuję odpowiednie wartości do podanych parametrów. Ważnym elementem podczas odczytu tych danych jest tak zwany szum który powstaje w trakcie przygotowania sensorów. W trakcie pierwszych odczytów szum ten został zmierzony oraz usunięty z pomiarów. Akcelerometr położony na płaskim obiekcie prawidłowo podawał odczyty, czyli zwracał wektor $[0, 0, g]$ gdzie $g$ oznacza grawitacje ziemską. Żyroskop natomiast wskazywał błąd rzędu $[2.80, 0.18, 0.18]$ w związku z czym od każdego odczytu właśnie taką wartość należy odjąć w celu uzyskania prawidłowych danych - czyli wektora zbliżonego do $[0, 0, 0]$ w pozycji w której został skalibrowany. Jak wspomniano w sekcji **??** dotyczącej żyroskopu, wartości zwracane są podane w $\frac{rad}{s}$, również znane jako dps (z ang. Degrees per second) czyli w kątach na sekundę. W celu określenie kątów w jakim urządzenie się znajduję w danym momencie musimy te dane przemnożyć przez częstotliwość z jaką są one pobierane - czyli przemnożyć przez czas co zwróci nam miarę kątów. Osiągamy to poprzez zapisanie czasu w którym dane zostały pobrane a także poprzez zapisanie tej wartości przed następnym pobraniem danych jako czas ostatniego poboru. Różnica pomiędzy tymi wartościami daje czas jaki upłynął aby uzyskać nowe wartości. Zaczynając w pozycji kalibracyjnej, z idealnie usuniętym szumem, żyroskop zwróci wartość zero, a wraz ze zmianą wartości żyroskopu, suma tych zmian wskaże na orientację kontrolera względem pozycji wyjściowej. Więcej informacji na ten temat znajduję się w rozdziale **??** [7]. Ostatnią cechą którą chcemy przekazać są dane pobierane z sensorów wygięcia w celu określenia pozycji palców. Tak jak wspomniano w sekcji 6.1 dane te pobieramy przez mierzenie napięcia jakie znajduje się na poszczególny analogowych pinach wejściowych. Odczyty te uzyskujemy poprzez wywołanie metody *analogRead(pin)* [3]. Pobieranie danych oraz ich wstępna obróbka przedstawia listing 6.5.

```
38  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable
        ()) {
39          previousTime = currentTime;
40          currentTime = millis();
41          elapsedTime = (currentTime - previousTime) / 1000;
42
43          IMU.readAcceleration(acc[0],acc[1], acc[2]);
44          IMU.readGyroscope(gyro[0],gyro[1], gyro[2]);
45
46          gyro[0] -= 2.80;
47          gyro[1] -= 0.18;
48          gyro[2] -= 0.18;
49
50          fingers[0] = analogRead(A3);
51          [...]
52  }
```

Listing 6.5: Wczytywanie danych z sensorów.

Na tym etapie mamy już dostęp do danych z akcelerometru w $\frac{m}{s^2}$ oraz dane z żyroskopu wyrażone w stopniach. Informacje te mogłyby zostać przesłane w takiej formie, jednak z racji znajomości projektu, dane można dodatkowo skorygować poprzez zastosowanie dodatkowych filtrów. Podstawowym zabiegiem który został zastosowany jest eliminacja danych nieznaczących, czyli szumu który powstaje poprzez naturalne drgania ciała. Aby to osiągnąć, w programie przechowywane są poprzednie wartości z poszczególnych sensorów. Jak dane znaczące dla akcelerometru przyjęto różnice $\pm 0.1$, a dla danych z sensorów wygięcia $\pm 15$. Dla żyroskopu natomiast przyjęto wartości różniące się o przynajmniej $\pm 0.2$, a także dodatkowo sprawdzane jest czy dane z żyroskopu nie są w pozycji przy kalibracyjnej. Oznacza to że jeżeli wartości zwracane z żyroskopu wynoszą $0 \pm 0.1$, zostaną one zamienione na wartość równą $0$. Filtrowanie wartości pokazane jest na listingu 6.6. W ten oto sposób otrzymano wartości z sensorów które były gotowe do wysłania na inne urządzenie. Wartości te jednak są wyrażone w postaci tablic typu *float*, natomiast jako wartości cechy bluetooth przyjmują tablice by-

tów. W związku z tym przed przypisaniem danych są przekazywane adresy tablic do nowych zmiennych, a następnie zmienne te zapisane w poszczególnej cesze przy użyciu metody *setValue(value,valueSize)*. Rozmiar jest ten określony jako 12 ponieważ mamy do czynienia z tablicą 3 zmiennych typu *float* z których każda zajmuje 4 bajty pamięci. Zapisywanie danych jest pokazana na listingu 6.7 dla jednej cechy - proces ten należy powtórzyć dla wszystkich danych. Program następnie zapisuje obecne dane jako dane obecnej pętli, tak aby następne odczyty mogły zostać porównane z obecną iteracją, tym samym kończąc wykonywaną pętle. Warto zauważyć że program pomimo swojego głównego działa w funkcji *loop()*, jest wykonywany w ramach jednej iteracji jak tylko dojdzie do połączenia kontrolera z odbiornikiem.

```
53 for (int i = 0;i<3;i++){
54     if(!(gyro[i] < oldGyroData[i]-0.2 || gyro[i] >
          oldGyroData[i]+0.2)){
55          if( gyro[i] < -0.1 ||  gyro[i] > 0.1){
56            gyro[i] = oldGyroData[i];
57          }else{
58           gyro[i] = 0.0;
59          }
60     }
61
62     if(!(acc[i] < oldAccData[i]-0.02 || acc[i] > oldAccData
          [i]+0.02)){
63        acc[i] = oldAccData[i];
64        }
65 }
66
67 for (int i = 0;i<5;i++){
68    if(!(fingers[i] < oldFingersData[i]-15.0 || fingers[i] >
          oldFingersData[i]+15.0)){
69                fingers[i] = oldFingersData[i];
70             }
71          }
72  }
```

Listing 6.6: Wstępne filtrowanie danych z sensorów.

```
73          byte *accChar = (byte*)&acc;
74          imuAccChar.setValue(accChar,12);
```

Listing 6.7: Zapisywanie danych do cech w serwisie.

## 6.4 Dalszy rozwój

Przypis z pracy dla wszystkich podrzodziałów

Wykorzystania projketu pokazano w drugiej pracy

W rozdziałach **??** i **??** pokazano budowę kontrolera oraz aplikacji która wykorzystuje zbudowany kontroler w celu obsługi podstawowych funkcji rękawicy-kontrolera. Projekt ten powstał z myślą ograniczonego budżetu, prostoty wykonania oraz możliwości replikacji. Założenia te spowodowały że zdecydowano się na pewne rozwiązania które w końcowej wersji projektu pokazały swoje wady. W tym rozdziale zostanie poruszony temat błędów popełnionych w pierwszej wersji tego projektu oraz przykładowe sposoby na ich rozwiązanie w przyszłości.

### 6.4.1 Problemy mikroprocesora

Przede wszystkim szukano małego mikrokontrolera tak aby nie był on przeszkodą podczas użytkowania kontrolera. O ile założenie to było dobrym pomysłem, okazało się że umiejscowienie przy brzegu sprawiło że ruch palców, w szczególności kciuka, może zmienić położenie jednostki IMU na rękawicy. Oznacza to że nawet jeśli nasza ręka znajduję się w stałej pozycji, samo poruszanie palcami wprowadza błąd w odczycie. Niestety wybór tego produktu od Arduino również przysporzył wiele kłopotów z racji błędnego rozłączania się adaptera Bluetooth. Z dotychczasowego użytkowania można stwierdzić iż kontroler poprawnie łączy się i rozłącza dwa razy, zaś w większości testowanych przypadków dochodzi do błędu połączenia przy trzeciej próbie. Aby usunąć ten błąd należy odłączyć płytkę od zasilania i podłączyć ponownie, resetując tym samym moduł. Samo oprogramowanie rękawicy skupia się na odczytach z dwóch sensorów. Praktyka pokazała że dane te nie

są wystarczająco dokładne, i jeżeli to możliwe powinny być pobierane również dane magnetometru w celu dodatkowego korygowania odczytów z żyroskopu.

### 6.4.2 Problemy budowy i czujników wygięcia

Problem z użytkowaniem rękawicy pojawił się dość szybko od jej zbudowania. Mianowicie wybrana do projektu rękawica była zbyt gruba, powodując dyskomfort w użytkowaniu w szczególności przez dłuższy okres czasu. Początkowe kryterium elastyczności, przesądziło o wybraniu tej rękawicy, jednak cienka rękawica również spełniła by wymagania końcowego produktu. Rozwiązanie zastosowane w celu odczytu wygięcia palców z założenia wyglądało na idealnie pasujące do wymagań projektu. Pomimo swojej prostoty wykonania oraz braku pomiaru takich cech jak odwodzenie palców czy też wygięcie poszczególnych stawów, spełnia ono swoją podstawową funkcję. Problemem tego rozwiązania jest natomiast brak elastyczności sensorów. W momencie zgięcia palców droga od knykci do paznokci się wydłuża sprawiając że sensor jest poddawany sile nacisku od strony palca która jest tym spowodowana. Sensory te pomimo braku elastyczności są zbudowane z materiału wytrzymałego na rozciąganie dzięki czemu nie pękają podczas zgięciu palców, jednak w celu zapewnienia lepszego mocowania i większej ochrony, z jednej strony została przymocowana elastyczna guma która trzyma sensor przy czubkach palców, z drugiej natomiast sensor został wszyty w rękawicę. Problem który się pojawił w trakcie użytkowania pochodził ze sposobu wszycia sensora. Została do tego użyta nić przewodząca która z powodu rozciągliwości rękawicy nie mogła zostać wszyta na sztywno, w związku z czym stawiała ona mniejszy opór podczas zginania palców i niejako została wyciągnięta przez sensor, powodując tym brak dokładności odczytów. Nić ta oprócz niskiej elastyczności okazała się być nietrwała. W trakcie korzystania z rękawicy doszło do kilku pęknięć, które zostały ponownie związane, jednak została przerwana w ten

sposób ciągłość obwodu. Przez dodanie dodatkowych wiązań odczyty z sensorów się pogorszyły, sprawiając że wygięcie palca wskazującego ma większy wpływ na odczyty z kciuka, niż zgięcie kciuka samo w sobie. Podobna sytuacja przytrafiła się z sensorem małego palca oraz serdecznego. Mała powierzchnia na dłoni wokół której należało poprawić wiele połączeń, sprawiła że część nici była blisko siebie, powodując momentami odczuwalne mrowienie na dłoni. Problem ten został rozwiązany poprzez zastosowanie izolacji od wewnętrznej strony rękawicy, jednak nie gwarantuje to przeciwdziałaniu zwarć w obwodzie. Konkludując, nić przewodząca nie jest najlepszym rozwiązaniem w celu połączenia elementów dla tego projektu i powinno zostać zastąpiona trwalszym połączeniem. Gdyby jednak została ona użyta, element przewodzący powinien znajdować się w środku oplotu, bądź powinny zostać zastosowane inne sposoby izolacji, a sama wytrzymałość nici powinna być znacznie większa. Mocowanie sensorów wygięcia powinno być bardziej trwałe oraz statyczne, nie pozwalając na przemieszczenie sensora na palcu. Alternatywą dla tego rozwiązania jest wykorzystanie czujników pomiaru wygięcia opartych o światło nadawane z jednej strony plastikowej tuby oraz miernika natężenia światła z drugiej. W ten sposób wiadomo że im mniejszy pomiar otrzymywanego światła, tym większe wygięcie tuby, której załamanie blokuje bezproblemowy dopływ światła. Rozwiązanie to również zapewnia pomiary niezniekształcone poprzez zachowanie innych sensorów a także wygląda na bardziej dokładne [11].

### 6.4.3 Animacja modelu

Ostatnim elementem aplikacji dla projektu jest zapewnienie animacji dłoni. W tym celu został wykorzystany *Google Sceneform*, dzięki któremu zaimportowano modele, ustawiono scenę, przypisano model a także obsługiwano przemieszczenie i orientację. Ostatnim brakującym elementem jest animacja modelu. Według

dokumentacji starszej wersji projektu osiągnąć to można poprzez klasę *Skeleton-Node*, pozwalającą na dostęp do kości modelu, bez wykorzystania zewnętrznego programu graficznego. Jednak z niejasnych przyczyn klasa ta została usunięta w ostatniej wersji SDK, powodując brak możliwości wprowadzania zmian w modelu który został zaimportowany przy użyciu wtyczki. Problem ten rozwiązano poprzez wykorzystanie programu *Blender*, dzięki któremu można było wyeksportować modele w wyznaczonej pozycji. Aby osiągnąć jednak animację modelu w czasie rzeczywistym, na podstawie dostępnych danych z sensorów wygięcia - cała klasa renderująca fragment **??** musi zostać napisana od nowa z wykorzystaniem innej technologii, ponieważ na oficjalnej stronie dystrybucji *Sceneform*, jest napisane iż projekt został zarchiwizowany, w związku z czym taka opcja nie zostanie dodana [9].

### 6.4.4 Błąd rotacji

W przypadku rotacji jest wiele sposobów na polepszenie rezultatów. W prezentowanym projekcie wybrano podstawową metodę która wykorzystuje jedynie żyroskop oraz akcelerometr i przy ich użyciu wykorzystuje filtr komplementarny. Tak jak wcześniej wspomniano, aby dokładnie skorygować żyroskop na wszystkich trzech osiach, należy wykorzystać również magnetometr. Oprócz tego istnieje wiele filtrów takich jak Kalmana czy Madgwick'a które skutecznie usuwają szum, a także algorytmy wykorzystujące nowe pomiary w połączeniu z tymi zebranymi przed nimi. Możliwości łączenia technik udoskonalania odczytu rotacji z jednostek IMU sprawia, że nie ma jednego najlepszego rozwiązania, a ich wybór jest uzależniony on rodzaju projektu nad którym się pracuje [10].

### 6.4.5 Problem obliczania przesunięcia

Obliczenie położenia kontrolera w przestrzeni, niewątpliwie należy do najtrudniejszego problemu w tym projekcie. Sedno problemu tkwi w niedokładności danych. Z powodu wykorzystania metody podwójnego całkowania, błąd uzyskiwany w cm przy pojedynczej całce, rośnie do m przy całkowaniu podwójnym. Ekran aplikacji jest mierzony w m, a ruch dłoni z kontrolerem ma ograniczony zasięg długości ramienia. Pomimo tego w niedużym czasie błąd rośnie do poziomu w którym model znika z ekranu użytkownika. Niestety nie istnieje łatwy sposób na skorygowanie błędów powstałych w wyniku tego algorytmu. Firmy zajmujące tym się tym problemem dodatkowo umieszczają czujniki pozwalające określić odbiornik urządzenia i ustalić pozycję względem niego, kamery zewnętrzne obserwujące ruch w przestrzeni a także dodatkowe czujniki optyczne. W przypadku rękawicy-kontrolera który może poruszać się we wszystkich kierunkach dodatkowy problem stanowi rotacja, przez którą błąd staję się coraz większy. W przypadku prostej aplikacji nie wykorzystującej zaawansowanych jednostek pomiarowych oraz algorytmów filtrujących, często efekt jaki można osiągnąć tą techniką nie sprawdza się w zastosowaniu, dlatego też dla tej aplikacji domyślnie funkcja ta jest wyłączona [10].

## 6.5 Podsumowanie projektu

W tym rozdziale podano informacje na temat projektu rękawicy kontrolera a także elementów które są wymagane w celu jego funkcjonowania. Zostały podane specyfikacje podzespołów, sposoby pozyskiwania danych, metoda komunikacji z innymi urządzeniami a także rozwiązano problemy związane z kosztem projektu. Krok po kroku przedstawiono złożenie elementów w celu uzyskania końcowej wersji produktu i ostatecznie omówiono zasadę działania programu napisanego

w Arduino wraz z bibliotekami które zostały użyte w ramach projektu, natomiast zasady ich praktycznego działania zostały pokazane na listingach. W ten oto sposób została zakończona główna część projektu, pozwalająca na wykorzystanie rękawicy kontrolera jako część większych przedsięwzięć. Aby jeszcze lepiej zobrazować sposób obsługi kontrolera, oraz metody wykorzystania danych, w rozdziale **??** zostanie pokazana przykładowa aplikacja wykorzystująca dane w celu analizy oraz prezentacji.

# Rozdział 7

# Podsumowanie

# Bibliografia


[1] K. Academy. Voltage divider. https://www.khanacademy.org/science/electrical-engineering/ee-circuit-analysis-topic/ee-resistor-circuits/a/ee-voltage-divider, 7 2020.

[2] Adafruit. Pressure-sensitive conductive sheet. https://www.adafruit.com/product/1361, 7 2020.

[3] Arduino. Language reference. https://www.arduino.cc/reference/en/, 7 2020.

[4] Botland.com. Arduino nano 33 ble - abx00030. https://botland.com.pl/pl/arduino-seria-nano-oryginalne-plytki/14758-arduino-nano-33-ble-abx00030-7630049201491.html, 7 2020.

[5] Botland.com. Czujnik ugięcia 112 x 6,3 mm - sparkfun sen-08606. https://botland.com.pl/pl/czujniki-nacisku/2406-czujnik-ugiecia-112-x-63-mm-sparkfun-sen-08606.html, 7 2020.

[6] P. Bugalski. (arduino, co w środku... – część 3 – źródło wbudowanych funkcji). https://forbot.pl/blog/arduino-co-w-srodku-3-zrodlo-wbudowanych-funkcji-id17291, 7 2020.

[7] Dejan. Arduino and mpu6050 accelerometer and gyroscope tutorial. https://howtomechatronics.com/tutorials/arduino/arduino-and-mpu6050-accelerometer-and-gyroscope-tutorial/, 7 2020.

[8] Forbot. Rezystor - co warto wiedzieć. https://forbot.pl/blog/leksykon/rezystor, 7 2020.

[9] Google. Sceneform overview. https://developers.google.com/sceneform/develop, 3 2020.

[10] R. Labs. Um7 content library. https://redshiftlabs.com.au/support-services/um7-content-library/, 7 2020.

[11] A. D. Murray. Oprical flex sensors. https://www.instructables.com/id/Optical-Flex-Sensor/, 7 2017.

[12] Russell108. Ble nano 33 does not report or disconnect from central. https://github.com/arduino-libraries/ArduinoBLE/issues/33, 5 2020.

[13] SM. Arduino lsm9ds3 library. https://www.arduino.cc/en/Reference/ArduinoLSM9DS1, 7 2020.

[14] SM. Arduinoble library. https://www.arduino.cc/en/Reference/ArduinoBLE, 7 2020.

[15] SM. Getting started with the arduino nano 33 ble. https://www.arduino.cc/en/Guide/NANO33BLE, 7 2020.

# TODOs:


cel i zarys, wytłumaczenie tytułu . . . . . 1

czym jest rzecywistość i skąd wiemy co jest rzeczywiste . . . . . 2

wytłymaczyć skróty i używanie angielskich nazwa jako standard . . . . 2

IMU . . . . . 3

historia, pierwsze firmy, popularne rozwiazania . . . . . 3

czym są i po co jak i dlaczego . . . . . 4

Pierwsze firmy, upadek, wzrost i spadek zainteresowania . . . . . 4

proof read . . . . . 6

Opis: BLE, Rotacja, przesunięcie i animacja, przy opisywanych sensorach 6

Przypis z pracy dla wszystkich podrzodziałów . . . . . 26

Wykorzystania projketu pokazano w drugiej pracy . . . . . 26