Uniwersytet Zielonogórski

Wydział Informatyki, Elektrotechniki i Automatyki

Praca dyplomowa

Kierunek: Informatyka

Praktyczny prototyp sprzętowo-programowy mikroprocesorowego przestrzennego manipulatora 3D

Mikołaj Mosoń

Promotor:
dr inż. Łukasz Hładowski

Zielona Góra, luty 2024

Spis treści

Spis rysunków

Spis tabel

Rozdział 1

Wstęp

1.1. Cel i zakres pracy

Celem pracy jest opracowanie prototypu praktycznego urządzenia które, przyspieszy proces projektowania modeli 3D w programach typu CAD. Urządzenie ma za zadanie pozwolić użytkownikowi na swobodne obracanie kamerą w 6 osiach. Komercyjne rozwiązania dostępne aktualnie na rynku są drogie. Motywacją pracy jest stworzenie taniej alternatywy, która wykorzysta łatwo dostępne gotowe moduły żyroskopu i akcelerometru. Aby uzyskać ruch w osiach XYZ wykorzystane są elementy podatne w postaci parametrycznych plastikowych sprężyn.

Praca swym zakresem obejmuje:

- Opracowanie i praktyczne wykonanie układu elektronicznego dla manipulatora
- Opracowanie projektu i wykonanie modelu zewnętrznego dla manipulatora
- Projekt i implementacja oprogramowania umożliwiającego działanie urządzenia
- Wykonanie testów praktycznych opracowanego przyrządu

1.2. Analiza rynku

Rynek w tym przypadku to głównie zbiór klientów, którzy byliby zainteresowani wdrożeniem rozwiązania pozwalającego na zwiększenie komfortu oraz przyspieszenie projektowania w programach CAD. Możemy rozróżnić dwa główne typy klientów: klienci biznesowi, czyli centra inżynieryjne oraz klientów indywidualnych. Centra inżynieryjne - są to miejsca w których często projektuje się linie produkcyjne. Przykładem może być dostosowywanie paletek przewożących produkt na taśmociągu produkcyjnym. Klienci indywidualni - urządzenie mogłoby znaleźć swoje zastosowanie jako akcesorium dla freelancerów. Kolejną potencjalną grupą odbiorców indywidualnych byłyby osoby zajmujące się hobbistycznym modelowaniem 3D. Z racji zasady działania urządzeń takich jak joystick lub pad komputerowy, urządzenie mogłoby potencjalnie znaleźć swoje zastosowanie jako akcesorium dla graczy.

1.3. Potrzeba biznesowa

Obecnie rynek urządzeń wskazujących dla programów CAD zmonopolizowany jest przez jedną firmę - 3Dconnexion. Inne urządzenia które byłyby w stanie wypełnić tę lukę wymagają skomplikowanej konfiguracji aby działać. Urządzenie opisywane w niniejszej pracy ma na celu stanowić tańszą alternatywę. Dodatkową zaletą jest bezproblemowa współpraca opracowanego narzędzia z systemem Linux. Należy podkreślić, iż rozwiązania opracowane przez firmę 3DConnection nie wspierają oficlajnie tego systemu.

1.4. Przegląd rozwiązań

Aktualnie na rynku dostępne są następujące typy urządzeń wejściowych:

- `Trackball`[1] - urządzenie polegające na poruszaniu kulą zamiast poruszaniem całym urządzeniem, jak w przypadku myszy.
- `Joystick`[2], w zależności od modelu posiadają 2 lub 3 osie. Bez zewnętrznego oprogramowania nie są wspierane przez oprogramowanie CAD.
- `3DConnexion SpaceMouse`[3] - to urządzenie do obsługi oprogramowania CAD, które umożliwia precyzyjne i sterowanie kursorami w aplikacjach 3D. Jest to urządzenie typu "joystick" z 6-osiową kontrolą ruchów.
- `Steam controller`[4] - urządzenie zaprojektowane z myślą o graniu na komputerze. Jednak oprogramowanie które jest do niego dołączone pozwala na skonfigurowanie go tak aby działał z każdą aplikacją. Posiada wbudowany żyroskop oraz zestaw trackballi i joysticków.

Rozdział 2

Implementacja

2.1. Wykorzystane technologie

W tej sekcji zostaną omówione narzędzia, które zostały wykorzystane do realizacji projektu. Wśród tych narzędzi znajdują się m.in. oprogramowanie, drukarka 3D, komponenty elektroniczne, filament do drukarki oraz wiele innych. Wybór tych konkretnych narzędzi był wynikiem ich dostępności i przydatności w kontekście projektu oraz doświadczeniem przed rozpoczęciem projektu.

2.1.1. Oprogramowanie

- `Arduino IDE`[5] - narzędzie wykorzystywane w początkowej fazie projektu. Jednym z powodów wyboru `Arduino IDE` do programowania projektu było to, że jest to darmowe środowisko programistyczne. Ponadto, projekt wykorzystywał płytkę `Arduino Leonardo`, która była obsługiwana przez IDE. Ale wraz z rozwojem pojawiła się potrzeba bardziej zaawansowanych narzędzi do programowania i debugowania. W związku z tym środowisko programistyczne zostało zmienione.

- `VsCode`[6] - Wraz z rozwojem projektu podjęta została decyzja o zmianie środowiska programistycznego z `Arduino IDE` na `VsCode`. Rozważano również edytor tekstu `NVIM`, jednak pojawiły się problemy z konfiguracją serwera języków, aby był on kompatybilny z bibliotekami `Arduino`. Ostatecznie wybrano `VsCode`, ponieważ oferuje on większą funkcjonalność nie tracąc na prostocie użytkowania.

- `PlatformIO`[7] - to zaawansowane, open-source'owe środowisko programistyczne dla mikrokontrolerów, które umożliwia rozwijanie aplikacji dla różnych platform sprzętowych. Jest to uniwersalne narzędzie, które pozwala na pracę z różnymi platformami sprzętowymi, takimi jak `Arduino`, `ESP8266`, `ESP32`, `STM32`, oraz wiele innych. `PlatformIO` oferuje dostęp do tych samych bibliotek co `Arduino IDE`. Funkcje `PlatformIO` połączone wraz z frameworkiem `Google Test` pozwalają na testy jednostkowe oraz kompilację i weryfikacje części kodu na komputerze bez potrzeby przesyłania kodu na mikrokontroler.

- `FreeCAD`[8] - został wybrany do projektu z kilku powodów. Przede wszyst-

kim, jest to darmowe oprogramowanie. Dodatkowo, `FreeCAD` posiada szereg narzędzi, które pozwalają na projektowanie modeli 3D oraz testowanie ich za pomocą metody FEM (Finite Element Method). Jednym z ważnych aspektów `FreeCAD` jest to, że modele, które się tworzy, mogą być parametryczne. Oznacza to, że po stworzeniu modelu, można go modyfikować, zmieniając wartości parametrów, takich jak długość, szerokość, wysokość itp. To bardzo przydatne na etapie prototypowania, pozwala to na osiągniecie wielu wariantów tego samego modelu. Ponadto, dzięki rozszerzeniom stworzonym przez społeczność, `FreeCAD` pozwala na składanie modeli w oprogramowaniu bez konieczności drukowania w celu sprawdzenia czy części do siebie pasują. Dzięki temu można zaoszczędzić czas i koszty związane z drukowaniem i testowaniem wielu wersji prototypu.

- `Cura`[9] - to darmowy, otwartoźródłowy program do przygotowywania plików do wydruku dla drukarek 3D. Dzięki Cura użytkownicy mogą dostosować ustawienia drukowania do swoich potrzeb, takie jak rozdzielczość, prędkość drukowania, ilość wypełnienia modelu i wiele innych. Cura oferuje także symulację druku, co pozwala na wczesne wykrywanie błędów i uniknięcie nieudanych prób drukowania. Sprawdzenie wyżej wymienionych parametrów jest krytycznie ważne w celu poprawnego wydruku elementów odkształcalnych.

2.1.2. Narzędzia

- Drukarka 3D tevo tornado[10] - wraz z zestawem części między innymi dyszami o różnych średnicach. Jest to drukarka typu FDM oparta na konstrukcji CR-10. W porównaniu do innych drukarek jak na przykład Prusa i3 MK3S lub Anycubic i3 Mega, Tevo Tornado posiada dużą przestrzeń roboczą o wymiarach 300 x 300 x 400 mm. Co z kolei niwelowało problem dzielenia modeli na mniejsze części oraz pozwalało na wydruk wielu modeli jednocześnie.

- `Prusament PLA firmy Prusa Polymers`[11] - wybór filamentu był kwestią bardzo ważną. Firma Prusa udostępnia parametry tego filamentu do pobrania[12], co pozwoliło na precyzyjne określenie jego właściwości i wykorzystanie ich w symulacjach numerycznych (FEM).

2.1.3. Biblioteki

- `Google Test`[13] - framework do testowania jednostkowego w języku C++, opracowany przez Google. Umożliwiający pisanie testów jednostkowych do weryfikacji poprawności zachowania kodu. Jest wyposażony w funkcje do odkrywania, wykonywania i raportowania wyników testów. Google Test wyposażone jest również w Google Mock (gmock). Pozwala to na prostsze testowanie skomplikowanych systemów, izolując poszczególne komponenty i sprawdzając ich interakcje z innymi częściami kodu.

- `Spacenavd`[14] - jest to sterownik umożliwiający komunikację pomiędzy urządzeniami typu myszy 3D a oprogramowaniem CAD. Jest w pełni kompatybilny z oprogramowaniem 3DConnexion, i jest jego darmową alternatywą. Oznacza

to że urządzenia współdziałające z tym sterownikiem powinny działać z ponad 200 programami wymienionymi na stronie 3DConnexion.

2.1.4. Elementy odkształcalne

Tradycyjnie aby uzyskać zakres ruchu w projektowanym urządzeniu stosowane są elementy sztywne (przykład stanowią zawiasy w drzwiach). Przeciwieństwem takich mechanizmów są elementy odkształcalne. Elementami odkształcalnymi są jednostki, które są częścią większej całości i są w stanie zmieniać swoje kształty i rozmiary w reakcji na siły mechaniczne, takie jak nacisk, rozciąganie lub skręcanie. Elementy odkształcalne są powszechnie stosowane w inżynierii i są istotne w projektowaniu i analizie konstrukcji, takich jak mosty, budynki, maszyny i sprzęt. Mogą być wykonane z różnych materiałów, takich jak stali, aluminium czy tworzywa sztuczne.

2.2. Modelowanie

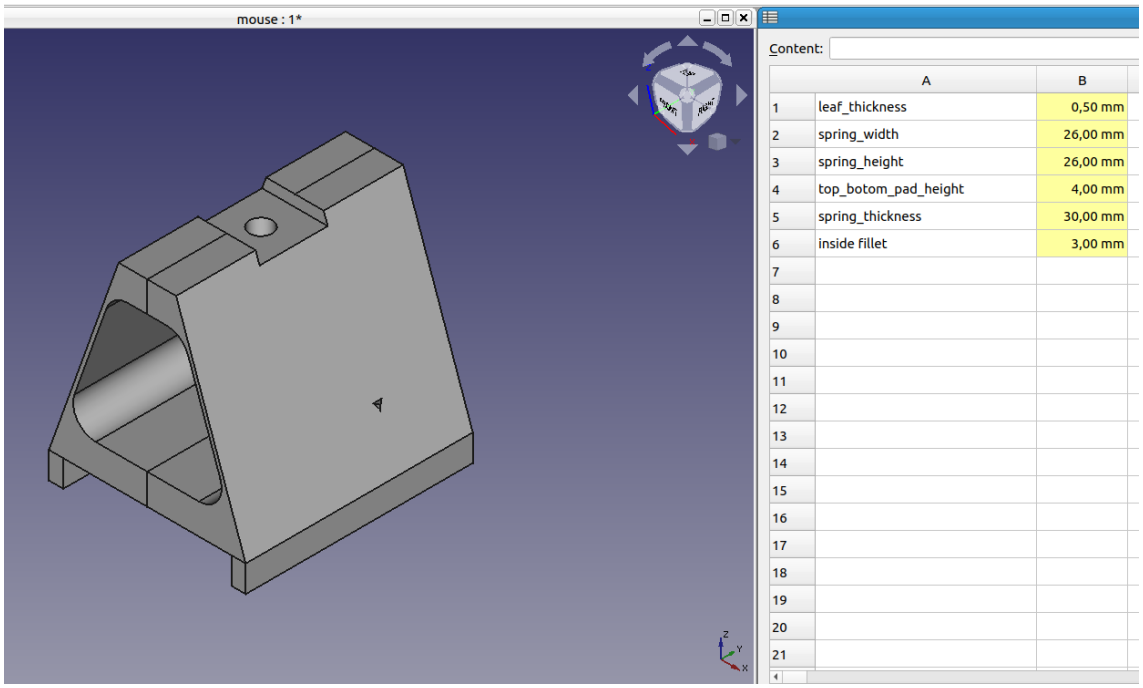
W projekcie zastosowano technikę modelowania twardo powierzchniowego (z ang. hard surface) z parametryzacją modeli. Parametryzacja modeli 3D w programach CAD polega na definiowaniu i kontrolowaniu różnych parametrów i właściwości modelu, które wpływają na jego kształt, rozmiar, proporcje, wygląd i inne cechy. Pozwala to na łatwe dostosowywanie i modyfikowanie modelu, zachowując przy tym jego zależności geometryczne i konstrukcyjne. Dzięki parametryzacji modeli 3D w programach CAD uzyskuje się większą elastyczność i kontrolę nad projektem. Zmiany w parametrach mogą być łatwo wprowadzane, a modele mogą być dostosowywane do różnych wymagań. Parametryzacja pozwala również na automatyzację generowania wariantów modeli i ułatwia iteracyjne modyfikowanie projektu, co może przyspieszyć proces prototypowania i zwiększyć efektywność pracy. Sprężyny zostały wymodelowane na podstawie książki [15].

Dodatkową zaletą parametryzowanych sprężyn jest możliwość spersonalizowania urządzenia, dzięki czemu użytkownik może wybrać czy chce aby stawiały one mniejszy lub większy opór.

2.2.1. Symulacje FEM

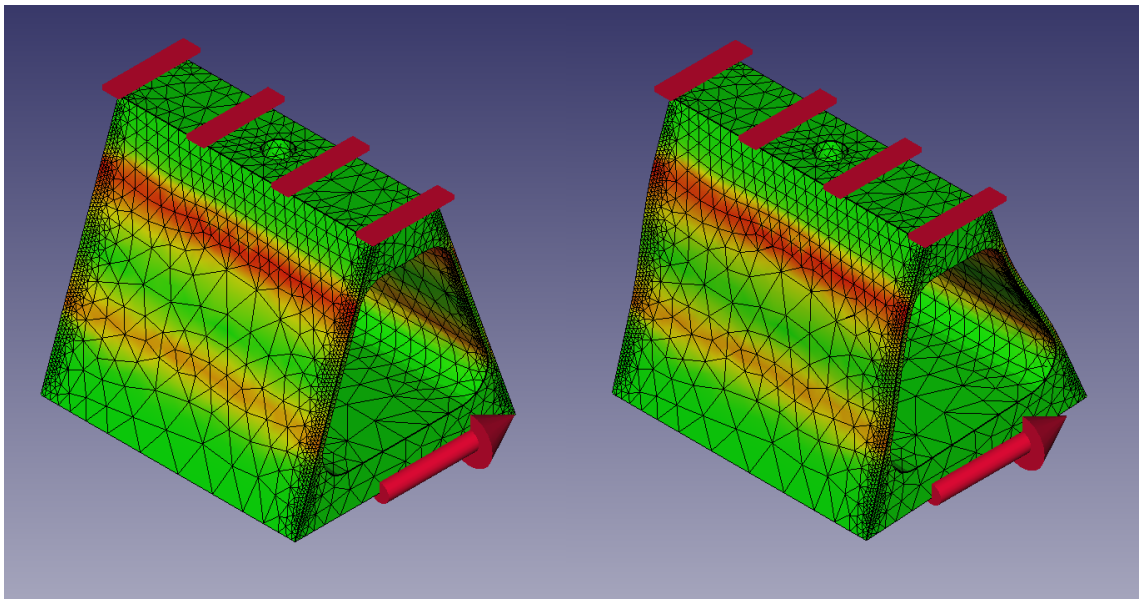
Symulacje w oparciu o technikę elementów skończonych (Finite Element Method) w programie `FreeCAD` są narzędziem do analizy i symulacji zachowania modeli 3D pod wpływem obciążeń. FEM jest popularną metodą numeryczną wykorzystywaną w inżynierii do modelowania i analizy zachowania się różnych struktur, takich jak elementy mechaniczne. Metoda ta jest wykorzystywana w projekcie do sprawdzenia poprawności zachowania się elementów odkształcalnch. Moduł FEM, który umożliwia wygenerowanie odpowiedniej siatki modelu i przypisanie jej właściwości materiałowych. Właściwości materiałowe PLA zostały przypisane na podstawie parametrów technicznych podanych przez producenta filamentu[12].

Warto zwrócić uwagę że rys. 2.2 oraz 2.1 mogą wydawać się tym samym modelem. Jednakże do wykonania symulacji FEM wykorzystany został model uproszczony. Dla modelu ze zrzutu 2.1 możnaby wytworzyć siatkę potrzebną do symulacji ale trwałoby



Rys. 2.1. Zrzut ekranu przedstawiajacy zastosowanie parametryczności w programie `FreeCAD`.

to znacznie dłużej oraz ze względu na znacznie bardziej skomplikowaną geometrie ukończyłoby się błędem podczas próby obliczeń.



Rys. 2.2. Zrzut ekranu przedstawiajacy symulacje FEM w programie `FreeCAD`.

2.2.2. Wydruk modeli

Dobór parametrów wydruku w przypadku obudowy urządzenia nie ma większego znaczenia. Wygląda to jednak inaczej w przypadku elementów odkształcalnych. Sprężyny posiadają ścianki które są bardzo cienkie przez co nie można wykorzysty-

wać ustawień takich jak brim. Brim to dodatkowa warstwa materiału na obrzeżach obiektu, która zapewnia lepszą przyczepność do stołu drukującego. Przykład zaznaczono kolorem niebieskim na rys. 2.5.

W celu utrzymania modelu w miejscu podczas wydruku najlepiej zastosować taśmę dwustronną lub klej. Rys. 2.3 przedstawia poprawnie wygenerowany g-code w programie cura. G-code jest to zestaw instrukcji które drukarka 3D musi wykonać w celu utworzenia modelu. Należy zwrócić uwagę że zewnętrzne cienkie ścianki sprężyny drukowane są jedną linią. Efekt ten można uzyskać dostosowując parametr `leaf thickness` widoczny na rys. 2.1 do szerokości linii wdruku. Najlepiej by te wartości były takie same jak średnica dyszy w drukarce 3D. Jest to najbardziej optymalne podejście do drukowania modelu ponieważ jest on wtedy najbardziej wytrzymały.

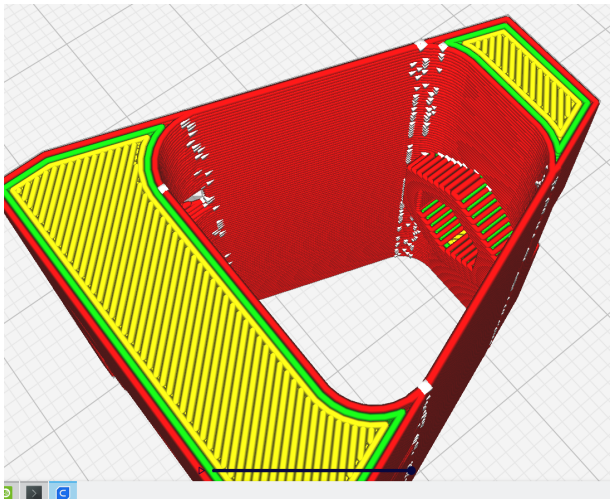
Na rys. 2.3 oraz 2.5 widać białe punkty. Każda warstwa posiada dwa takie punkty i oznaczają one miejsce startowe oraz końcowe. W figurze z poprawnie dobranymi parametrami można zauważyć że punkty rozmieszczone są w losowy sposób. Losowość ta wraz z dodatkowym zastosowaniem zaokrąglenia (ang. fillet) wewnątrz modelu sprawia że wydruk jest wytrzymalszy niż to co zostałoby uzyskane w przypadku który widać na 2.5. Rys. 2.5 niemalże wszystkie punkty startowe w tym samym miejscu modelu. Podczas wydruku powoduje to nagromadzenie PLA w tych miejscach przez co ścianka sprężyny nie zawsze prawidłowo przylega poprawnie do reszty modelu. Staje się to najsłabszym miejscem uzyskanej bryły i w praktyce to będzie miejsce które pęknie jako pierwsze.

W przypadku rys. 2.4 w miejscu ścianki sprężyny wygenerowane zostało wypełnienie które widać za pomocą koloru pomarańczowego. Tak krótkie linie mają problem z adhezją do poprzednich warstw. Prowadzi to do niestabilności, słabej jakości wydruku oraz nierównomiernej ekstruzji plastiku. A w skrajnych przypadkach powoduje zacięcie się drukarki.

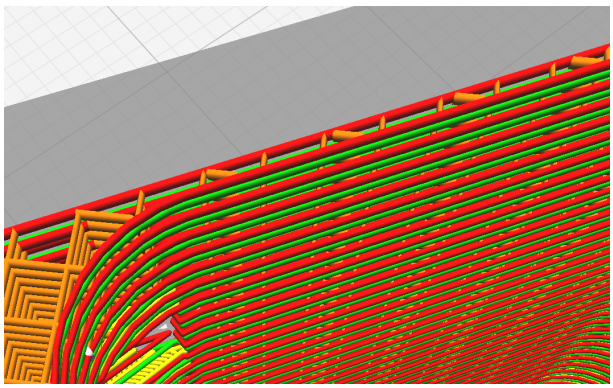
W celu zwiększenia komfortu gałka urządzenia posiada ergonomiczny kształt tak by użytkownik mógł ją wygodnie chwycić. Wygładzona powierzchnia urządzenia powoduje, że użytkownik ma trudności z pewnym chwytem. Potencjalnym rozwiązaniem jest zapożyczenie techniki wywodzącej się z obsługi tokarki zwaną radełkowaniem widoczną na zrzucie 4.2. Na gałce urządzenia został wytworzony wzór przypominający diamenty. Wytworzenie wzoru polega na ekstrudowaniu dwóch przecinających się linii i powielaniu wzoru z przesunięciem o kilka stopni. Jest to operacja bardzo wymagająca obliczeniowo dla komputera, i zajmuje więcej czasu w zależności od ilości powtórzeń. Ponieważ `FreeCAD` przy każdej modyfikacji modelu ponownie przelicza wszystkie kroki tą operacje najlepiej zastosować na samym końcu hierarchii. W przeciwnym razie najmniejsza nawet zmiana może potrwać do kilku minut.

2.2.3. Problem nazewnictwa topologicznego

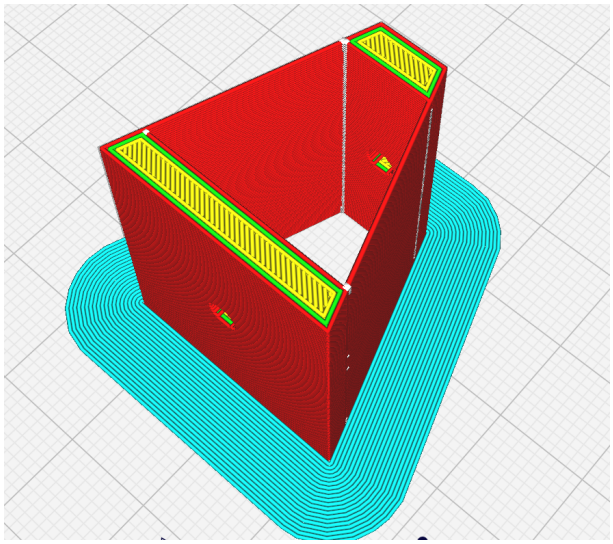
Problem z nazewnictwem topologicznym w programie `FreeCAD` polega na tym, gdy po wprowadzeniu zmian w obiekcie, takie jak wyciągnięcie, wycięcie, połączenie itp., wewnętrzna nazwa kształtu może ulec zmianie. Ta zmiana może wpłynąć na inne parametryczne właściwości zależne od tego kształtu. Skutkiem tego błędu może być na przykład przypisanie szkicu do innej ściany. Jest to zjawisko który dotyczy wszystkich modeli tworzonych za pomocą programu `FreeCAD`. Problem można zniwelować stosując odpowiednie techniki omówione w dokumentacji programu [16].



Rys. 2.3. Poprawne dobranie parametrów wydruku.



Rys. 2.4. Niepoprawne dobranie parametrów wydruku.



Rys. 2.5. Zrzut ekranu przedstawiający najsłabsze punkty modelu, oraz przykład brim.

2.3. Wzorce projektowe

Wzorce projektowe są uważane za najlepsze praktyki projektowania, które mają na celu zwiększenie jakości, wydajności i ponownego wykorzystywania kodu źródłowego.

W tej sekcji zostaną przedstawione wybrane wzorce projektowe wykorzystane w ramach realizacji projektu. Przedstawione zostaną opisy i przykłady zastosowań tych wzorców w celu zwiększenia zrozumienia ich roli. Wykorzystane w pracy implementacje wzorców projektowych bazują na książkach [17] i [18].

2.3.1. Protowątki

Protowątki są zwykle używane w systemach wbudowanych, gdzie zasoby są ograniczone, a wymagane jest jednoczesne wykonywanie wielu zadań. Dzięki prostocie i efektywności, są często stosowane w projektach do obsługi urządzeń wielu sensorów.

`Arduino` jest kontrolerem jednoprocesorowym i jednowątkowym. Oznacza to że jedynym sposobem na uzyskanie złudzenia wielozadaniowości jest zastosowanie protowątków.

Protowątki opierają się na technice opakowywania kodu w funkcje, które mogą być wykonywane sekwencyjnie w ramach jednego wątku. Działają one dzięki precyzyjnemu odmierzaniu czasu jaki upłynął pomiędzy kolejnymi wykonaniami funkcji.

W celu zwiększenia czytelności kodu jeden wątek zajmuje się tylko jednym zadaniem. W projekcie zostały zastosowane 4 protowątki:

- `protothreadMpuUpdate` - odpowiedzialny za kolejne odczytywanie wartości z czujnika żyroskopu (`MPU6050`),
- `protothreadJoystickBroadcast` - stosowany za emulację urządzenia HID,
- `protothreadRGB` - wątek zastosowany w celu wizualnej informacji zwrotnej,
- `protothreadPrintSerial` - jest to wątek który wyświetla najważniejsze parametry urządzenia,

Do każdego z protowątków za pomocą referencji przekazywane są obiekty enum klasy Mode lst. 2.1. Jest to referencja typu const ponieważ wątki jedynie odczytują stan w jakim urządzenie aktualnie się znajduje.

```
enum class Mode
{
  XYZ,
  rXrYrZ,
  BROADCASTING,
  NOT_BROADCASTING,
};
```

Listing 2.1. Implementacja stanów w których urządzenie może sie znaleźć

Następnie na wzór automatu Moore'a (tzn kolejne stany urządzenia nie są zależne od poprzednich) dany protowątek za pomocą instrukcji switch decyduje jak powinien się zachować.

```
1  static void protothreadMpuUpdate(pt &protoThread, std::array<
       float, 3> &deviceValues, volatile const Mode &broadcastMode)
       {
2    static unsigned long lastMpuUpdate = 0;
3    static std::array<float, 3> offsets;
4    PT_BEGIN(&protoThread);
5    while (1) {
6      lastMpuUpdate = millis();
7      std::array<float, 3> filteredValues;  //Mpu for one
       iteration sends max values
8      // timing between 19 - 49 ms results in buffer error
9      PT_WAIT_UNTIL(&protoThread, millis() - lastMpuUpdate >
       MPU_THREAD_TIMER);
10     if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) {
11       mpu.dmpGetQuaternion(&q, fifoBuffer);
12       mpu.dmpGetGravity(&gravity, &q);
13       mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
14       for (int i = { 0 }; i < allAxis.size(); i++) {
15         axis &currentAxis = allAxis.at(i);
16         currentAxis.pop_front();
17         currentAxis.push_back(ypr[i] * 180 / M_PI);
18         filteredValues.at(i) = median(currentAxis);
19       }
20       if (broadcastMode == Mode::NOT_BROADCASTING) {
21         offsets = { filteredValues };
22       }
23       for (int axisIndex{ 0 }; axisIndex < deviceValues.size();
       axisIndex++) {
24         deviceValues.at(axisIndex) = { filteredValues.at(
       axisIndex) - offsets.at(axisIndex) };
25       }
26     }
27   }
28   PT_END(&protoThread);
29 }
```

Listing 2.2. Protowątek odpowiedzialny za obsługę MPU6050

Funkcja na listingu 2.2 zajmuje się obsługą żyroskopu i akcelerometru. Urządzenie nieustannie zapisuje dane z żyroskopu i zapisuje je do dwukierunkowej kolejki. Gdy urządzenie nie znajduje się w trybie nadawania wartości te zapisywane są jako offsety. Ta potrzeba wynika z tego że gdy użytkownik obróci urządzenie gdy to nie znajduje się w trybie nadawania, wyniki będą przekłamane. Urządzenie musi się samo skorygować aby wysyłać poprawne sygnały. Dlatego wartości pomniejszane są o offsety i zapisywane do kontenera `deviceValues`, który wykorzystany wykorzystany jest w innych protowątkach 2.3.

```
1  static void protothreadJoystickBroadcast(pt &protoThread, const
       std::array<float, 3> &deviceValues, volatile const Mode &
       deviceMode, volatile const Mode &broadcastMode)
2  {
3    static unsigned long JoystickBroadcast = 0;
```

```
4   PT_BEGIN(&protoThread);
5   while (1)
6   {
7     JoystickBroadcast = millis();
8     PT_WAIT_UNTIL(&protoThread, millis() - JoystickBroadcast >
     JOYSTICK_THREAD_TIMER);
9     switch (broadcastMode)
10    {
11    case Mode::BROADCASTING:
12      for (int setterIndex{0}; setterIndex < 3; setterIndex++)
13      {
14        auto val = deviceValues.at(setterIndex) * 100;
15        switch (deviceMode)
16        {
17        case Mode::XYZ:
18          joystick.setXYZ(setterIndex, val);
19          break;
20        case Mode::rXrYrZ:
21          joystick.setRXRYRZ(setterIndex, val);
22          break;
23        default:
24          Serial.print("Wrong mode chceck: ");
25          Serial.print(__LINE__);
26          deviceFailed(10);
27          break;
28        }
29        joystick.send();
30      }
31      break;
32    case Mode::NOT_BROADCASTING:
33      break;
34    default:
35      Serial.print("Wrong mode chceck: ");
36      Serial.print(__LINE__);
37      deviceFailed(10);
38      break;
39    }
40  }
41  PT_END(&protoThread);
42 }
```

Listing 2.3. Przykład maszyny stanów w protowątku JoystickBroadcast

Funkcja na listingu 2.3 jest odpowiedzialna za nadawanie danych z urządzenia joystick przez określony czas zgodnie z aktualnym trybem pracy. W przypadku, gdy aktualnym trybem pracy jest `BROADCASTING`, funkcja iteruje po każdym elemencie tablicy wartości z urządzenia, mnoży ją przez 100, aby zmienić zakres wartości, a następnie przesyła wartości na zewnętrzny interfejs przez odpowiednie wywołania funkcji `joystick.setXYZ()` lub `joystick.setRXRYRZ()` i `joystick.send()`.

W przypadku, gdy aktualnym tryb pracy jest "NOT_BROADCASTING", funkcja nic nie robi. Funkcja kończy się po skończeniu pętli while, a protokół wątku

wykorzystuje makro "PT_END" do zakończenia wykonywania wątku.

2.3.2. Wzorzec adapter

Istnieje wiele bibliotek emulujących urządzenia HID. Mimo że większość z nich w wielu przypadkach wykonuje te same czynności ich interfejsy różnią się od siebie jak wskazano na listingu 2.4.

```
setXAxis(0); //Metoda z biblioteki Joystick.h
setJoystickX(JOY_LEFT, 0, false)); //Metoda z biblioteki Xinput.
    h
```

Listing 2.4. Obie metody setXAxis oraz setJoystickX odpowiadają za to samo zadanie mianowicie ustawianie wartości osi X.

Utrudnia to znalezienie biblioteki która okazałaby się najlepszym rozwiązaniem ponieważ każda nowa biblioteka wymagałaby drastycznych zmian w kodzie. Ten problem można jednak zniwelować stosując wzorzec projektowy adaptera. Na potrzeby głównej pętli został napisany interfejs do którego wykonano adaptery dla bibliotek XInput oraz Joystick.

```

/// Zawartosc JoystickEmulator.h
class JoystickEmulator
{
public:
    ...
    virtual void setXAxisRange(int MIN_X, int MAX_X) = 0;
    ...
};

/// Zawartosc JoystickLibraryAdapter.h

class JoystickLibraryAdapter : public JoystickEmulator
{
    Setter joystickXYZsetters[3];
    Setter joystickRXRYRZsetters[3];
    JoystickInterface *joystick;
public:
    ...
    void setXAxisRange(int MIN_X, int MAX_X)
    {
        joystick->setXAxisRange(MIN_X, MAX_X);
    }
    ...
};

/// Zawartosc JoystickLibraryAdapter.h

class XInputAdapter : public JoystickEmulator
{
    XInputInterface *XInput;
```

```
32
33 public:
34     ...
35     void setXAxisRange(int MIN_X, int MAX_X)
36     {
37         XInput->setRange(JOY_LEFT, MIN_X, MAX_X);
38     }
39     ...
40 };
41
42 #endif // XINPUT_ADAPTER
```

Listing 2.5. Zastosowanie wzorca adaptera na przykładzie metody setXAxisRange

Przed przystąpieniem do procesu wgrywania oprogramowania na płytce, należy wybrać adapter. W miejscu deklaracji adaptera należy od komentować odpowiednią sekcję kodu 2.6. Linijka 4 przypisuje obiekt klasy podrzędnej `JoystickLibraryAdapter` do referencji klasy nadrzędnej `JoystickEmulator`. Ponieważ `JoystickEmulator` jest interfejsem abstrakcyjnym, nie można utworzyć jego instancji. Można jednak użyć go jako referencji lub wskaźnika do obiektu konkretnej klasy implementującej interfejs, takiej jak `JoystickLibraryAdapter` lub `XInputAdapter`.

Należy w tym miejscu zwrócić uwagę, że `&&` oznacza referencję rvalue, która w tym kontekście jest używana do powiązania z tymczasowym obiektem zwracanym przez konstruktor `JoystickLibraryAdapter`. Wydłuża to czas życia obiektu tymczasowego do końca bieżącego zakresu, zapewniając, że obiekt pozostanie ważny do użycia przez referencję joysticka.

Jest to część polimorfizmu, gdzie wskaźnik lub referencja klasy bazowej może być użyta do wywołania metod zaimplementowanych przez dowolną klasę pochodną.

```
1 #include <XInputAdapter.h>
2 #include <XInput.h>
3 XInputController x;
4 JoystickEmulator &&joystick = XInputAdapter(x);
```

Listing 2.6. Wybór adaptera.

2.3.3. Wzorzec watchdog

Systemy niezależnie od swojej skali i zaawansowania, skazane są na błędy. Wynika to z faktu, że każdy system jest złożony z wielu elementów, które mogą wpływać na siebie nawzajem, a także zależeć od otoczenia i warunków, w których działają. W celu zapewnienia niezawodności systemów wbudowanych stosuje się mechanizm watchdog. Jego zadaniem jest monitorowanie pracy systemu i w razie wykrycia awarii automatycznie przerywanie jego pracy lub wykonywanie określonych działań awaryjnych, takich jak restart systemu.

```
1 #include <avr/wdt.h>
2
3 void deviceFailed(int code)
4 {
5   // (if it's going to break, usually the code will be 1)
```

```
6     // 1 = initial memory load failed
7     // 2 = DMP configuration updates failed
8     // 10 = Wrong mode detected
9     while (1)
10    {
11      Serial.print(F("Device failed (code "));
12      Serial.print(code);
13      Serial.println(F(")"));
14      delay(1000);
15      analogWrite(RED_PIN, 255);
16      delay(1000);
17      analogWrite(RED_PIN, 0);
18    }
19  }
20
21  void setup()
22  {
23      ...
24    wdt_enable(WDTO_4S);
25    Serial.println(F("Initializing I2C devices..."));
26    mpu.initialize();
27    devStatus = mpu.dmpInitialize();
28    Serial.println(devStatus);
29    if (devStatus != 0)
30    {
31      deviceFailed(devStatus);
32    }
33      ...
34  }
35
36  void loop()
37  {
38    wdt_reset();
39    ...
40  }
```

Listing 2.7. Zastosowanie mechanizmu watchdog

Funkcja `deviceFailed()` na listingu 2.7 generuje informacje zwrotną dla użytkownika gdy wystąpi błąd. Dioda LED urządzenia zacznie migać na czerwono oraz wyświetli na porcie Serial informacje z kodem błędu. Wykonując tą funkcję urządzenie wpada w nieskończoną pętle przez co `wdt_reset()` nie zostanie wywołane skutkując resetem urządzenia. Opracowane na potrzeby pracy urządzenie zostało skonfigurowane tak, aby w przypadku błędu zwracało jeden z poniższych kodów:

- 1, kod informuje o tym że nie udało się zainicjalizować `MPU6050`,
- 2, kod informuje o tym że wystąpił problem z odczytem informacji z DMP (Digital Motion Processor),
- 10 kod informuje o tym że maszyna stanów otrzymała niepoprawną wartość,

Autorka książki [17] zaleca wyłączenie funkcji watchdog podczas debuggowania z wykorzystaniem debbugera hardwareowego, w celu uniknięcia niespodziewanego zachowania.

2.4. Porównanie bibliotek HID

Wśród bibliotek HID dostępnych dla `Arduino` najodpowiedniesze i najbardziej dopracowane wydają się być `XInput` oraz `Joystick`. Realizują podobne do siebie funkcjonalności. Mają jednak znaczące różnice w tej sekcji zostaną one omówione.

2.4.1. Joystick

Biblioteka Joystick była w pierwszej kolejności brana pod uwagę przy projektowaniu urządzenia. Na jej bazie był zaprojektowany jest interfejs dla wzorca projektowego adapter. Pozwala ona na emulację szerokiej gamy sygnałów wejściowych:

- 32 przyciski,
- joystick osie - X, Y, Z (16 bitowa precyzja),
- joystick osie rotacji - X, Y, Z (16 bitowa precyzja),
- pedał steru (16 bitowa precyzja),
- pedał gazu (16 bitowa precyzja),
- przepustnica (16 bitowa precyzja),
- pedał hamulca (16 bitowa precyzja),
- kierownica analogowa (16 bitowa precyzja),

W przypadku gier pojawiają się problemy jeżeli chodzi o poprawne wykrywanie urządzenia które wykorzystuje tę bibliotekę. Po zainstalowaniu sterowników i ich odpowiedniej konfiguracji urządzenie będzie natomiast widziane przez wszystkie programy CAD które obsługują myszy 3D. Biblioteka jest oparta na licencji LGPL-3.0, pozwala ona na to aby urządzenie zostało wykorzystane komercyjnie.

2.4.2. XInput

`XInput` jest bez problemu wykrywany przez wszystkie gry wspierające XInput Microsoftu. Jeżeli chodzi o programy typu CAD, to domyślnie nie wspierają padów `XInput`. Ponieważ urządzenia `XInput` nie są kompatybilne z `Spacenavd` oznacza to że w zależności od programu trzeba szukać odpowiednich rozszerzeń. `XInput` nie może być zastosowany komercyjnie ponieważ stosuje rozwiązania Microsoftu `USB VID` i `PID`. Bez `VID/PID` sterowniki ignorują urządzenie i nie będzie ono wykorzystane jako kontroler[19]. `XInput` pozwala na emulacje:

- 10+1 przycisków,

- 2 joysticków analogowych (16 bitowa precyzja),
- 2 triggery analogowe (8 bitowa precyzja),
- 1 czterokierunkowy D-Pad.

Programowanie z wykorzystaniem XInput jest niewygodne. Sama biblioteka działa domyślnie w trybie debug, to jest wyświetla przez interfejs szeregowy Serial wartości które urządzenie powinno nadawać. Żeby nie działać w trybie debug trzeba wykorzystać `XInput` USB Core for Arduino AVR. Korzystanie z tego projektu znacznie komplikuje dalszy rozwój, ponieważ każdorazowa aktualizacja oprogramowania wymaga całkowitego resetu płytki. Warto jeszcze zwrócić uwagę że wykorzystania tej biblioteki urządzenie nie zostanie nigdy poprawnie rozpoznane przez konsole Microsoftu[19], ponieważ układ nie posiada odpowiedniego chipu. Mimo że `XInput` pozwala na 6 wejść analogowych to realnie do tego projektu można wykorzystać jedynie 4 wejścia czyli oba joysticki. Triggery natomast nie nadają sie do wykorzystania.

2.5. Testowanie bibliotek HID

Jako framework testujący brane pod uwagę biblioteki Google Test, ArduinoUnit oraz Unity[20]. Dwa ostatne frameworki pozwalają na testowanie bezpośrednio na `Arduino`. Są one znacznie mniej popularne niż Google Test. Warto również dodać że nazwa Unity pokrywa się z silnikiem do gier o takiej samej nazwie. Znacznie utrudnia to znalezienie informacji o tej bibliotece. W ostateczności zdecydowano na wykorzystanie Google Test. Jest to najczęściej wykorzystywane narzędzie do pisania testów. Jego największą zaletą jest możliwość korzystania z Google Mock.

W celu zapewnienia poprawności działania metod, stworzono testy jednostkowe. Użycie Google Mock umożliwiło testowanie funkcji bez konieczności podłączania płytki do komputera, co znacząco przyspieszyło proces pisania kodu. Dzięki temu podejściu możliwe było skoncentrowanie się na testowaniu poszczególnych funkcji bez konieczności manualnego sprawdzania na fizycznym sprzęcie.

2.5.1. Wykorzystanie GMock

Listing 2.8 przedstawia przykład zastosowania biblioteki Google Mock dla biblioteki Joystick. Utworzono klasę `JoystickMock`, która dziedziczy po interfejsie `JoystickInterface`. W klasie `JoystickMock` zdefiniowano szereg metod, które są symulowane za pomocą makr `MOCK_METHOD`. Te makra pozwalają na definiowanie metod w klasach mock w sposób deklaratywny, bez konieczności dostarczania ciała tych metod. Makro `MOCK_METHOD` przyjmuje zwracany typ, nazwę metody oraz parametry. Następnie generowany jest kod, który tworzy symulowaną implementację metody w klasie mock, co pozwala na kontrolowanie zachowania tych metod podczas testów. Analogiczna klasa mock jaką widać na listingu 2.8 została utworzona dla biblioteki XInput.

```
1 #ifndef JOYSTICK_MOCK
2 #define JOYSTICK_MOCK
3 #include <gmock/gmock.h>
```

```
4  #include <mocks/JoystickInterface.h>
5
6  class JoystickMock : public JoystickInterface
7  {
8  public:
9      MOCK_METHOD(void, begin, (bool initAutoSendState), (override
       ));
10     MOCK_METHOD(void, end, (), (override));
11     MOCK_METHOD(void, pressButton, (uint8_t button));
12     MOCK_METHOD(void, releaseButton, (uint8_t button), (override
       ));
13     MOCK_METHOD(void, sendState, (), (override));
14     MOCK_METHOD(void, setAccelerator, (int32_t value), (override
       ));
15     MOCK_METHOD(void, setBrake, (int32_t value), (override));
16     MOCK_METHOD(void, setButton, (uint8_t button, uint8_t value)
       , (override));
17     MOCK_METHOD(void, setHatSwitch, (int8_t hatSwitch, int16_t
       value), (override));
18     MOCK_METHOD(void, setRudder, (int32_t value), (override));
19     MOCK_METHOD(void, setXAxis, (int32_t value), (override));
20     MOCK_METHOD(void, setYAxis, (int32_t value), (override));
21     MOCK_METHOD(void, setZAxis, (int32_t value), (override));
22     MOCK_METHOD(void, setRxAxis, (int32_t value), (override));
23     MOCK_METHOD(void, setRyAxis, (int32_t value), (override));
24     MOCK_METHOD(void, setRzAxis, (int32_t value), (override));
25     MOCK_METHOD(void, setRxAxisRange, (int32_t MIN_X, int32_t
       MAX_X), (override));
26     MOCK_METHOD(void, setRyAxisRange, (int32_t MIN_Y, int32_t
       MAX_Y), (override));
27     MOCK_METHOD(void, setRzAxisRange, (int32_t MIN_Z, int32_t
       MAX_Z), (override));
28     MOCK_METHOD(void, setXAxisRange, (int32_t MIN_X, int32_t
       MAX_X), (override));
29     MOCK_METHOD(void, setYAxisRange, (int32_t MIN_Y, int32_t
       MAX_Y), (override));
30     MOCK_METHOD(void, setZAxisRange, (int32_t MIN_Z, int32_t
       MAX_Z), (override));
31     MOCK_METHOD(void, setSteering, (int32_t value), (override));
32     MOCK_METHOD(void, setThrottle, (int32_t value), (override));
33 };
34
35 #endif // JOYSTICK_MOCK
```

Listing 2.8. Przykład zastosowania gmock dla biblioteki Joystick.

2.5.2. Testy jednostkowe

Makro TEST jest podstawowym elementem frameworka do testowania oprogramowania w Google Test. Służy do definiowania testów jednostkowych. Makro przyjmuje nazwę grupy testów oraz nazwę samego testu. Grupowanie testów jest

wskazane w celu zachowania porządku. W projekcie przyjęto podejście, w którym stosuje się jak najbardziej opisowe nazwy dla testów.

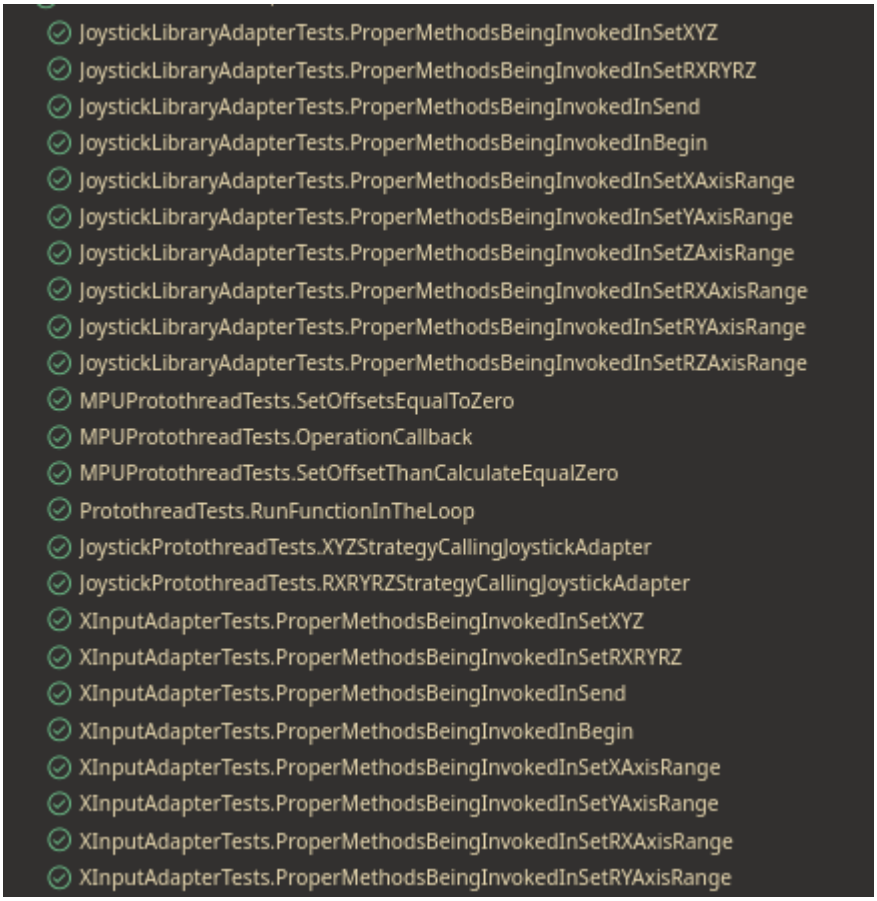
```
...
TEST(JoystickLibraryAdapterTests,
    ProperMethodsBeingInvokedInSetXYZ)
{
    JoystickMock j;
    JoystickEmulator &&joystick = JoystickLibraryAdapter(j);
    constexpr int axisCount{3};
    EXPECT_CALL(j, setXAxis(0)).Times(testing::AtLeast(1));
    EXPECT_CALL(j, setYAxis(0)).Times(testing::AtLeast(1));
    EXPECT_CALL(j, setZAxis(0)).Times(testing::AtLeast(1));
    for (size_t axis = 0; axis < axisCount; axis++)
    {
        joystick.setXYZ(axis, 0);
    }
}

TEST(JoystickLibraryAdapterTests,
    ProperMethodsBeingInvokedInSetRXRYRZ)
{
    JoystickMock j;
    JoystickEmulator &&joystick = JoystickLibraryAdapter(j);
    constexpr int axisCount{3};
    EXPECT_CALL(j, setRxAxis(0)).Times(testing::AtLeast(1));
    EXPECT_CALL(j, setRyAxis(0)).Times(testing::AtLeast(1));
    EXPECT_CALL(j, setRzAxis(0)).Times(testing::AtLeast(1));
    for (size_t axis = 0; axis < axisCount; axis++)
    {
        joystick.setRXRYRZ(axis, 0);
    }
}

TEST(JoystickLibraryAdapterTests,
    ProperMethodsBeingInvokedInSend)
{
    JoystickMock j;
    JoystickEmulator &&joystick = JoystickLibraryAdapter(j);
    EXPECT_CALL(j, sendState()).Times(testing::AtLeast(1));
    joystick.send();
}
...
```

Listing 2.9. Przykładowe testy jednostkowe.

W listingu 2.9 przedstawiono kod napisany na potrzeby testowania klasy `JoystickLibraryAdapter`. Każdy z testów przyjmuje obiekty typu mock na którym będzie sprawdzane poprawne wywoływanie metod. Makro `EXPECT_CALL` przyjmuje wspomniany mock, nazwę odpowiedniej metody oraz oczekiwaną ilość wywołań.



Rys. 2.6. Zrzut ekranu przedstawiający rezultat testów jednostkowych w `VsCode`.

Rysunek 2.6 przedstawia rezultat wszystkich testów stworzonych do projektu. Jednak nie wszystkie zaimplementowane klasy zostały ostatecznie wykorzystane. Klasy do których powstały testy jednostkowe a które nie zostały wykorzystane zostaną omówione w sekcji dotyczącej dalszego rozwoju projektu.

Rozdział 3

Kontroler urządzenia

3.1. Mikrokontroler

Jedną z najważniejszych decyzji podczas projektowania urządzenia jest wybór odpowiedniego mikrokontrolera. To, który mikrokontroler zostanie wybrany, ma wpływ na potencjalne możliwości i sposób komunikacji z komputerem.

Na rynku dostępnych jest wiele gotowych rozwiązań dla małych, wydajnych jednostek obliczeniowych, począwszy od prostych platform takich jak Arduino Leonardo, opartych na procesorach Atmel ATmega[21], aż po znacznie szybsze i wydajniejsze mikrokontrolery, takie jak Teensy LC[22] z procesorem ARM Cortex[23] na pokładzie. Tabela 3.1 przedstawia porównanie mikrokontrolerów które były brane pod uwagę.

Szukany kontroler powinien posiadać:

- możliwość emulacji urządzenia HID,
- obsługiwać magistralę I2C,
- możliwie mały rozmiar.

Parametr	**Arduino Leonardo**	**STM32 blue pill**	**ESP-WROOM-32**	**Teensy-LC**
Mikrokontroler	ATmega32u4	STM32F103C8T6	ESP32	MKL26Z64
Architektura	AVR	ARM Cortex-M3	Xtensa LX6	ARM Cortex-M0+
Zegar	16 MHz	72 MHz	240 MHz	48 MHz
Pamięć Flash	32 KB	64 KB	4 MB	256 KB
Pamięć RAM	2.5 KB	20 KB	520 KB	32 KB
Ilość pinów cyfrowych	20	37	26	27
Ilość pinów analogowych	12	10	18	13
Komunikacja	USB, UART, SPI, I2C	UART, SPI, I2C	Wi-Fi, Bluetooth, UART, SPI, I2C	USB, UART, SPI, I2C

Tab. 3.1. Porównanie Arduino Leonardo, STM32F103C8T6, ESP-WROOM-32 i Teensy-LC

Po rozważeniu różnych opcji, w tym Arduino Leonardo, STM32, ESP32 i Teensy LC, ostatecznie wybrano Arduino Leonardo. Wybór ten opiera się na tym, że Arduino Leonardo spełnia powyższe wymagania, a ponadto jest bardzo popularnym rozwiązaniem, co ułatwia znalezienie informacji na jego temat. Mimo relatywnie małych możliwości względem konkurencji Arduino Leonardo nie ma problemów z osiągnięciem założeń projektu.

Zastosowanie ESP32 wydaje się być nieuzasadnione, ponieważ projekt nie wykorzystałby w pełni potencjału tej płytki, co mogłoby prowadzić do nadmiernego skomplikowania i zużycia zasobów bez realnych korzyści dla projektu. Wbudowany moduł wifi nie zostałby nigdy wykorzystany, natomiast moduł bluetooth który można by użyć w celu ewentualnego rozwinięcia projektu jest też dostępny w formie osobnego modułu dla Arduino Leonardo.

3.1.1. Moduł inercyjny

Do przechwytywania informacji w osiach XYZ zastosowane zostały moduły nawigacji inercyjnej. Na rynku dostępne są moduły wykorzystujące magnetometr, żyroskop oraz akcelerometr. Urządzenia te działając pojedynczo są podatne na błędy.

- magnetometr, wykorzystuje pole magnetyczne ziemi pomiar może być zakłócony przez różnego rodzaju magnesy.
- żyroskop, pozwala na dokładne wykrywanie położenia urządzenia. Jego działanie jest podatne na akumulację błędów w dłuższym okresie. Po pewnym czasie prowadzi to do nieakceptowalnych rezultatów.
- akcelerometr, podaje on dokładne pomiary pod warunkiem że jedyną siłą która na niego działa jest przyspieszenie ziemskie. Obracając urządzenie pomiary zaczynają oscylować, w praktyce oznacza to że w danych pojawia się dużo szumu.

Wybrany został moduł `MPU6050` jest to moduł o 6 stopniach swobody. O wyborze modułu przeważyły biblioteki dostępne dla tego modelu. Pozwalają one na wykorzystanie wbudowanego procesora DMP (z angielskiego digital motion processor). W praktyce oznacza to że dane z akcelerometru i żyroskopu są łączone aby zniwelować swoje wady.

Warto również zwrócić uwagę na problem z odczytywaniem wartości z modułu `MPU6050`. Po dokładniejszej analizie stwierdzono, że w czasie pomiędzy odczytami wynoszącym od 19 ms do 49 ms, moduł nie zwraca żadnych danych. Początkowo przypuszczano, że problem leży w wadliwej lub uszkodzonej płytce/przewodach, jednakże nawet po zakupie nowego podzespołu problem nadal występował. Pod uwagę brana była możliwość zbyt dużego obciążenia `Arduino` ale to rówież okazało się nie być przyczyną. Jedyna informacja, która mogłaby wyjaśnić to zjawisko, została odnaleziona na forach internetowych. W tych źródłach podejrzewa się, że podczas odczytu z bufora, w wyżej wspomnianych interwałach czasowych, DMP z modułu `MPU6050` blokuje wewnętrzny bufor w celu przeprowadzenia aktualizacji. Aby dokładnie określić momenty, w których moduł przestawał odpowiadać, zastosowano metodę ciągłego odczytywania wartości z MPU i wypisywania ich na port szeregowy (Serial Port). W każdej iteracji pętli dodawano także krótkie opóźnienie (delay) o 1 ms.

3.1.2. Inne

Ta podsekcja zawiera informacje o pozostałych komponentach które nie odgrywają tak ważnej roli w działaniu urządzenia niemniej jednak warto o nich wspomnieć.

- W celu przełączania trybów pracy urządzenia zastosowanio mikroswitche.
- W urządzaniu w roli wizualnej informacji zwrotnej zastosowano diody RGB lduino SE010. Każdy kolor został przypisany do odpowiedniej osi XYZ, i zapalany jest z odpowiednią intensywnością w zależności od wychylenia.

Rozdział 4

Instrukcja obsługi

Do złożenia urządzenia potrzebny będzie śrubokręt philips o średnicy nie większej niż 3 mm. Jest to konieczne ponieważ podczas montażu należy przewlec śrubokręt przez otwory o wspomnianej średnicy. Zaleca się aby śrubokręt był namagnesowany, ułatwia to umieszczenie nakrętek w odpowiednich slotach.

Potrzebne części można podzielić na elementy które należy wydrukować:

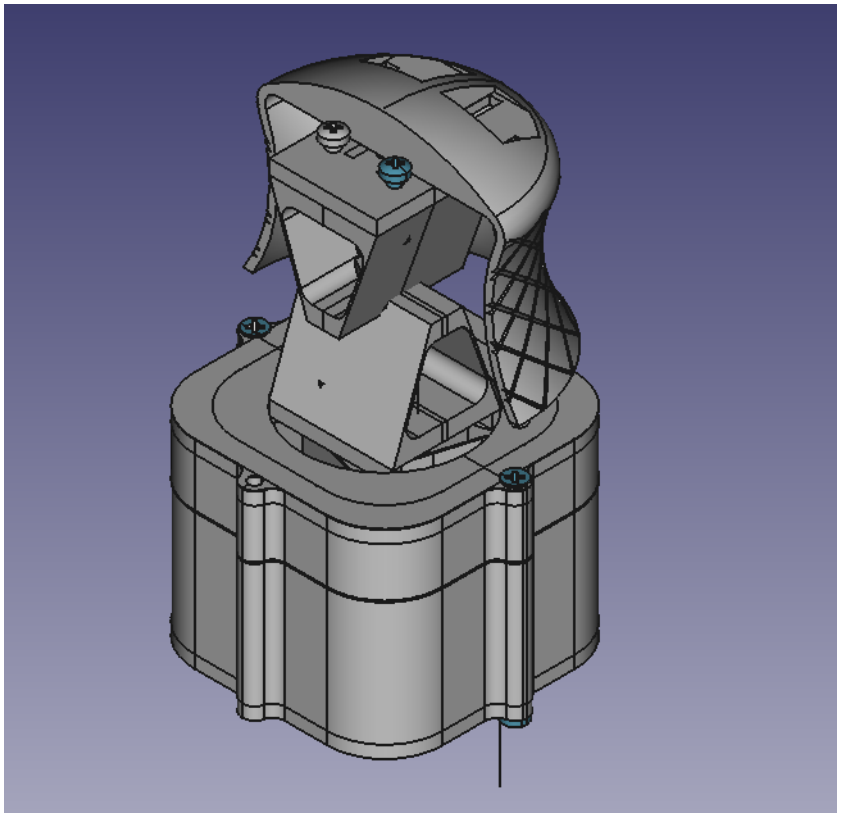
1. model sprężyny Y wersja A,
2. model sprężyny XY,
3. model gałki urządzenia,
4. model wierzchniej płytki,
5. model separatora,
6. model podstawy.

Oraz elementy które należy kupić:

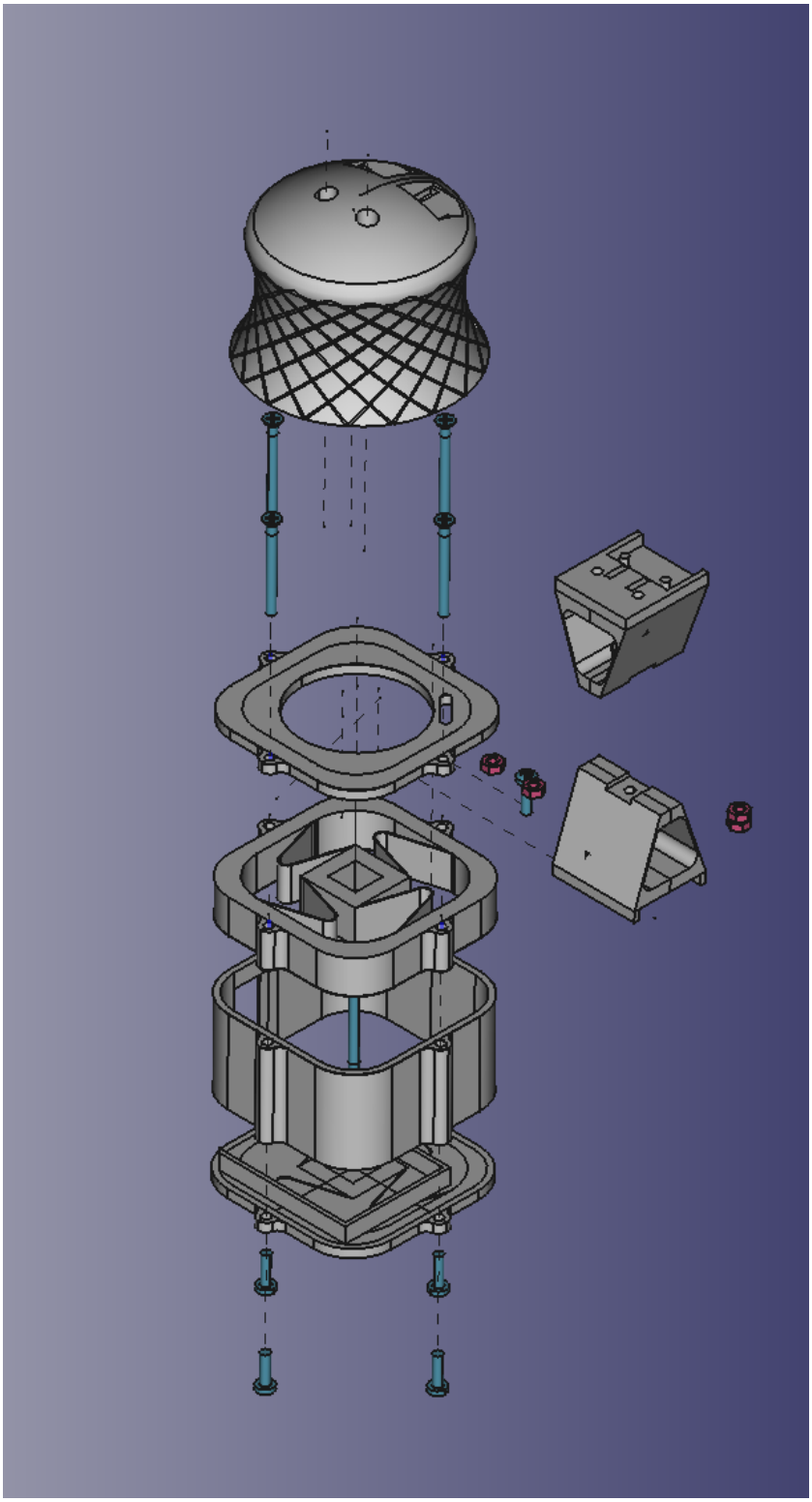
1. 7 śrub M3x13,
2. 5 śrub M3x30,
3. 4 nakrętki M3,
4. 2 przyciski microswitch,
5. sensor MPU6050,
6. Arduino Pro Micro,
7. moduł diody RGB lduino SE010,
8. przewody do połączenia układu.

4.1. Składanie fizycznej części urządzenia

Moduły TechDraw i ExplodedAssembly nie zostały zoptymalizowane do współpracy ze sobą, co sprawia, że niemożliwe jest bezpośrednie generowanie szczegółowych instrukcji montażu w formie pliku PDF. Niemniej jednak, w ramach projektu udało się wykonać animację typu exploded assembly, prezentującą dynamiczny proces składania całego modelu. Animację można odtworzyć bezpośrednio w programie `FreeCAD`.



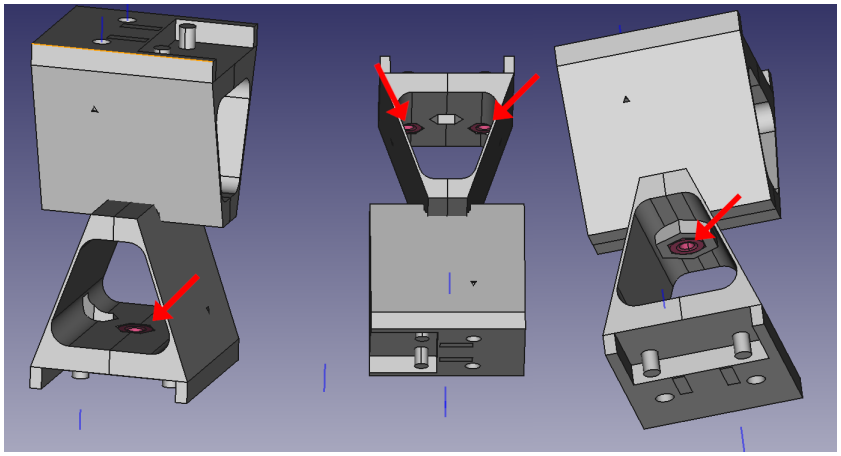
Rys. 4.1. Zrzut ekranu przedstawiający ułożenie sprężyn wewnątrz urządzenia `FreeCAD`.



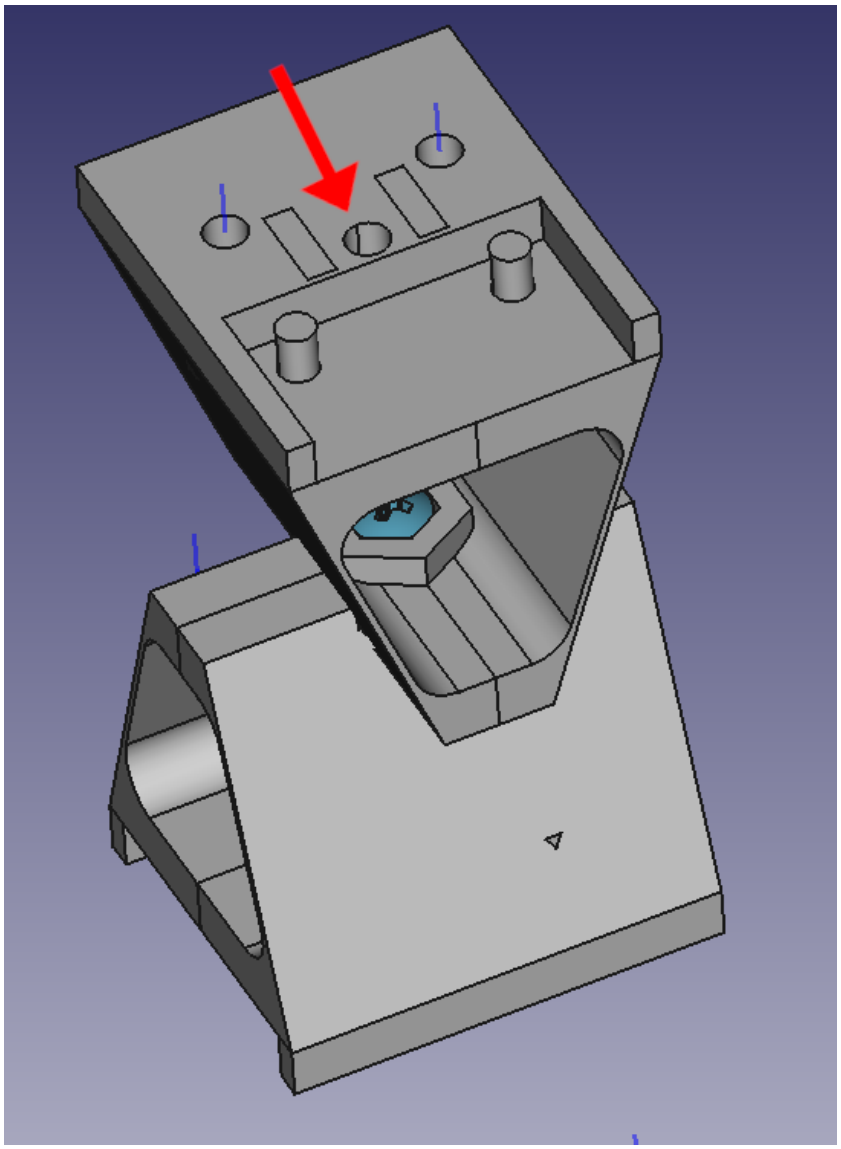
Rys. 4.2. Zrzut ekranu przedstawiający w pełni rozłożone urządzenie w programie `FreeCAD`.

1. W kroku pierwszym należy wstawić nakrętki M3 we wskazane sloty na rysunku 4.3.

2. Następnie używając otworu wskazanego na rysunku 4.4 przykręcić śrubę M3x10, tak aby połączyć ze sobą dwie sprężyny Y.

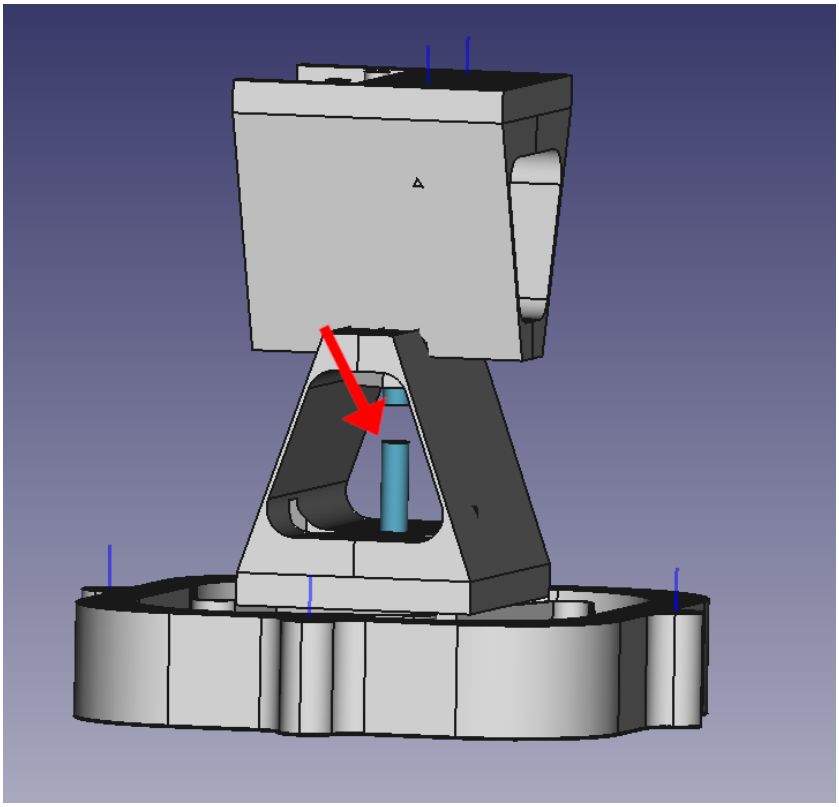


Rys. 4.3. Krok pierwszy procesu składania urządzenia.



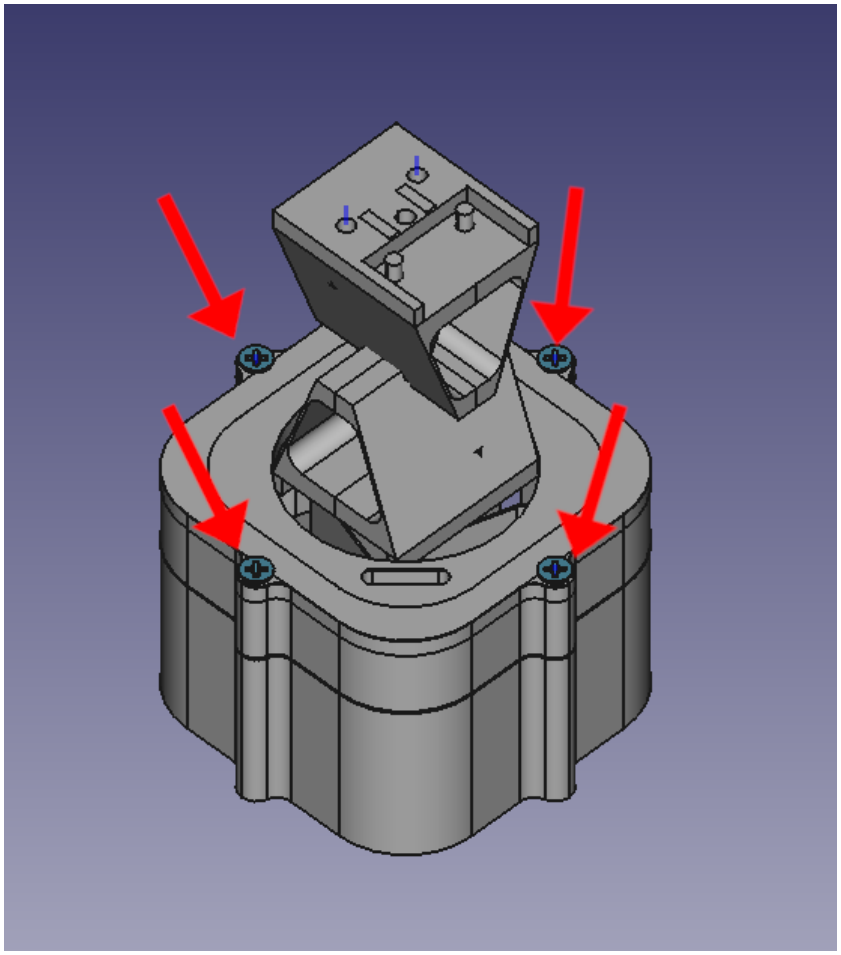
Rys. 4.4. Krok drugi procesu składania urządzenia.

3. W kroku 3 należy połączyć sprężyny Y wraz ze sprężyną XY używając śruby M3x30, tak jak wskazano na rysunku 4.5.



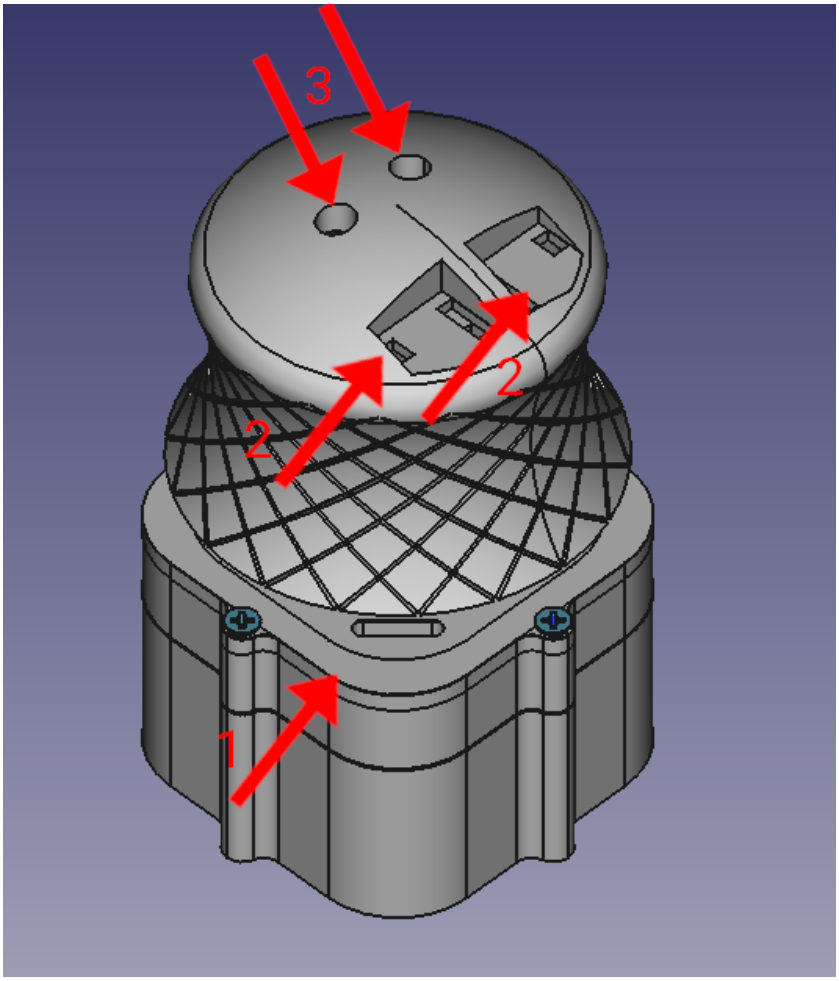
Rys. 4.5. Krok trzeci procesu składania urządzenia.

4. Następnie należy od spodu urządzenia nałożyć model dystansu obudowy, a na wierzchu nałożyć model górnej płytki. Elemenyt należy połączyć ze sobą za pomocą czterech śrub M3x30, w miejscach wskazanych na 4.6. Oraz ułożyć moduł `MPU6050` w pasującym slocie.



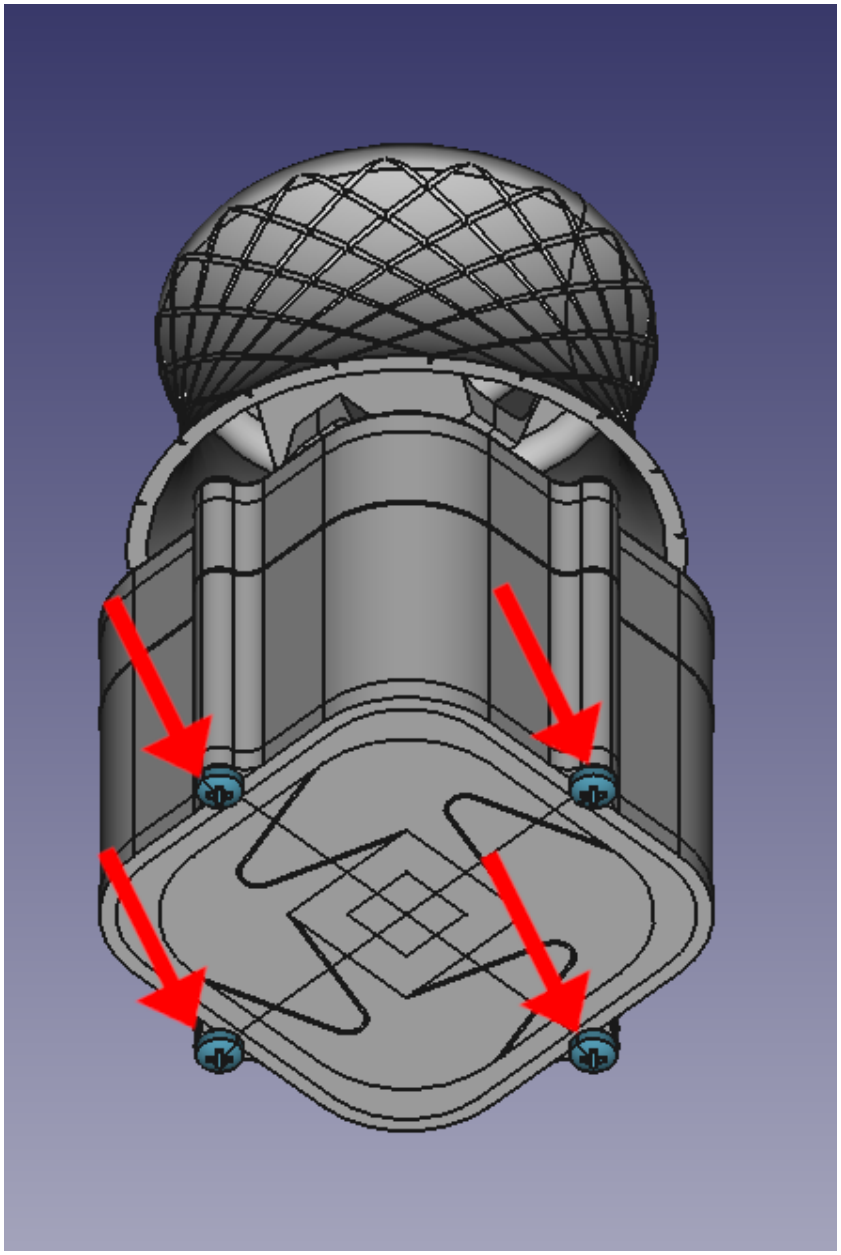
Rys. 4.6. Krok czwarty procesu składania urządzenia.

5. Przed nałożeniem modelu gałki należy przewlec przewody przez otwór wskazany przez strzałkę z numerem 1 na rysunku 4.7. Przewody należy podłączyć do `MPU6050` i do microswitchy które następnie należy zamocować w slotach wskazanych przez strzałki z numerem 2. Następnie należy użyć śrub M3x10 do zamocowania gałki urządzenia do sprężyn w miejscach wskazanych przez strzałki oznaczone numerem 3.



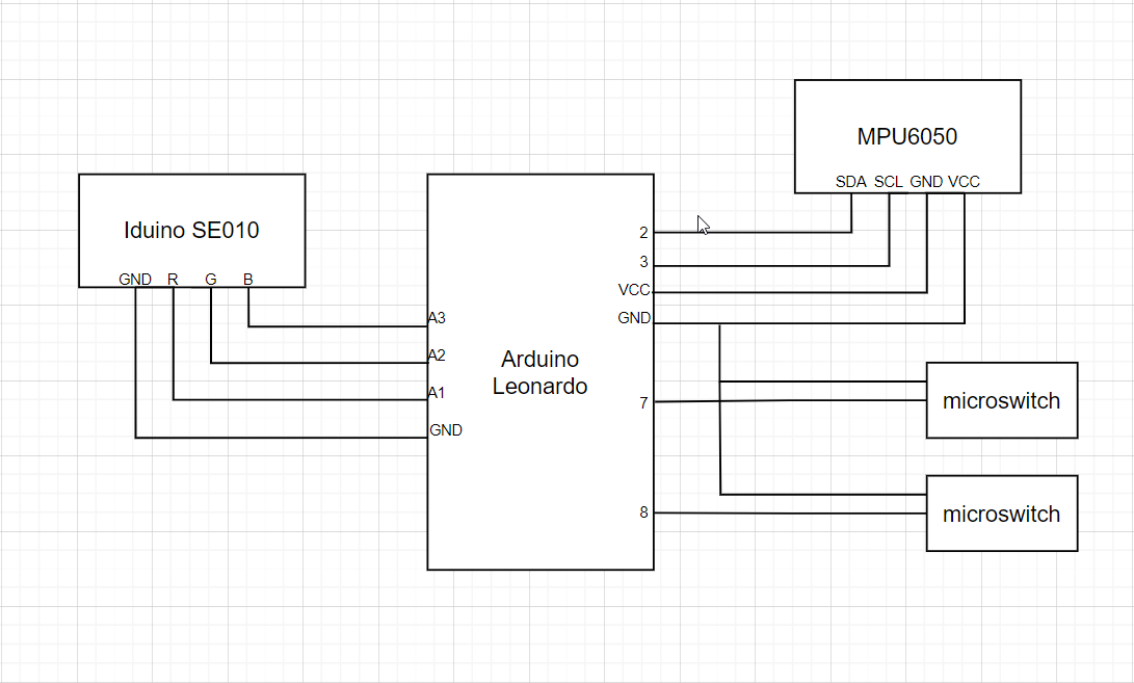
Rys. 4.7. Krok piąty procesu składania urządzenia.

6. W ostatnim kroku należy podłączyć przewody microswtchy, `MPU6050` oraz modułu diody do `Arduino` i zamocować podstawę urządzenia za pomocą śrub M3x10 w miejscach wskazanych na rysunku 4.8.



Rys. 4.8. Krok szósty procesu składania urządzenia.

Rys. 4.9 przedstawia schemat połączenia wszystkich komponentów elektronicznych. Moduł diody należy podłączyć pod piny analogowe A1, A2, A3. Układ wykorzystuje tylko piny SDA, SCL modułu `MPU6050` które należy podłączyć kolejno do pinów 2 oraz 3. Microswitche należy podłączyć do pinów 7 oraz 8.



Rys. 4.9. Schemat podłączenia urządzenia.

4.2. Wymagania software

Należy pobrać oraz zainstalować `Spacenavd` ze strony projektu. Aby sterownik działał poprawnie trzeba mu podać ID urządzenia. W systemie Linux można zastosować poniższe polecenie aby utworzyć konfiguracje minimalną. `Spacenavd` [14] jest w stanie obsłużyć wiele urządzeń, więc możliwe jest wpisanie więcej niż jednego ID.

```
1   sudo echo "device-id = $(lsusb | grep Arduino | cut -d' ' -
    f6)" > /etc/spnavrc
2
```

Listing 4.1. Minimalna konfiguracja spacenavrc.

Listing 4.1 pobiera informacje o wszystkich podłączonych urządzeniach USB wykorzystując komendę `lsusb`. Przeszukuje je za pomocą polecenia `grep` a następnie wykorzystując polecenie `cut` wyciąga identyfikator urządzenia i zapisuje go do odpowiedniego folderu tak aby być w stanie współpracować z `Spacenavd`.

Rozdział 5

Wnioski

5.1. Mikrokontrolery

Z perspektywy czasu, wybór mikrokontrolerów Arduino do projektu, mimo swojej przyjazności dla użytkownika, nie był najlepszym rozwiązaniem. Mikrokontrolery Arduino są wprawdzie bardzo łatwe w użyciu i posiadają wiele gotowych bibliotek i narzędzi, ale ich możliwości są ograniczone w porównaniu do bardziej zaawansowanych mikrokontrolerów, takich jak STM32 i ESP32. Dodatkowo, warto zauważyć, że mikrokontrolery takie jak STM32 i ESP32 cenowo niewiele się różnią od mikrokontrolerów Arduino, a jednak posiadają funkcjonalności, które są krytycznie ważne dla rozwoju projektu. Na przykład, mikrokontroler STM32 pozwala na wykorzystanie debugera sprzętowego (ST-Link 2), który pozwala na bezpośrednie debugowanie kodu na poziomie sprzętu, który jest niezbędny w celu odnalezienia błędów. Dla porównania oficjalna dokumentacja Arduino zaleca debugowanie za pomocą interfejsu szeregowego (Serial). Natomiast w przypadku mikrokontrolera ESP32 poza sprzętowym debugowaniem, możliwe jest testowanie jednostkowe bezpośrednio na płytce[7], dzięki czemu testy jednostkowe są po prostu lepsze. Podsumowując hobbistyczna natura tej platformy ułatwiła pracę we wczesnym stadium projektu, ale w późniejszych stadiach opóźniła postęp.

5.1.1. `MPU6050`

W module żyroskopu i akcelerometru obie funkcjonalności działają na zasadzie komplementarnej. Żyroskop jest w stanie dość precyzyjnie wychwytywać zmiany swojego położenia, jednakże zbiera on ze sobą błąd który kumuluje się w miare popierania kolejnych próbek z modułu. Problem ten niweluje zastosowanie odczytu z akcelerometru i naniesienia poprawek. Stąd też wynika 6 stopni swobody urządzenia. W początkowej fazie projektowania błędnie założono że każdy stopień swobody odpowiadałby odpowiednio za osie X, Y, Z, rX, rY, rZ.

5.2. Modelowanie

Z powodu braku doświadczenia, dokonano niewłaściwego wyboru struktury projektu. W miarę tworzenia kolejnych modeli, zauważono, że praca stawała się coraz

trudniejsza. W efekcie, napotkano trudności związane z wydajnością programu FreeCad, które wynikły z wcześniejszych decyzji projektowych. Na początku postanowiono trzymać wszystkie modele w jednym pliku. Jednak z biegiem czasu praktyka ta utrudniła pracę nad projektem. Okazało się, że prawidłowym podejściem byłoby tworzenie osobnego pliku dla każdego modelu. Pozwoliłoby to na lepszą organizację projektu, łatwiejsze zarządzanie i modyfikację poszczególnych modeli. Trzymając wszystko w jednym pliku, utracono przejrzystość i elastyczność, co przyczyniło się do trudności w pracy nad projektem.

Wykorzystanie techniki radełkowania oraz ergonomiczny kształt urządzenia zdecydowanie zwiększyły komfort w porównaniu do pierwszej iteracji gdzie zastosowano zwykły cylinder. Jednakże dłuższe użytkowanie urządzenia sprawia dyskomfort w palcach u dłoni. Jest to spowodowane tym że plastik PLA jest po prostu zbyt twardy. W kolejnej iteracji urządzenia należałoby pokryć czymś gałkę urządzenia lub zastosować inny materiał. Przykładem innego materiału mógłby być plastik TPU który jest znacznie bardziej elastyczny w porównaniu do PLA.

5.3. Dalszy rozwój

Potencjalne pomysły aby rozbudować projekt:

- **Klawiatura makr,** po zbudowaniu prototypu i przetestowaniu go w docelowych aplikacjach stwierdzono brak możliwości szybkiego korzystania ze skrótów klawiszowych. Rozwiązaniem byłaby klawiatura zawierająca programowalne makra. Najodpowiedniejsze byłoby rozwiązanie modularne. Pozwoliłoby to użytkownikowi na zapoznanie się z podstawową wersją urządzenia, a w przyszłości dałoby możliwość jego rozbudowy. W przypadku komercjalizacji projektu oznaczałoby to, że potencjalny klient inwestując w bazową wersję mógłby potencjalnie stać się powracającym klientem i inwestowałby dalej w ekosystem urządzenia.

- **Formowanie wtryskowe,** aktualnie modele są przygotowane pod wydruk 3D dla materiały PLA. Pozwala to tylko na produkcję urządzenia na małą skale. Zastosowanie form wtryskowych pozwoliłoby zmniejszyć czas trwania produkcji oraz zwiększyć jakość tworzonych części. Oznaczałoby to potrzebę zmodyfikowania bieżących modeli oraz przeprowadzenie nowych symulacji ponieważ formy wtryskowe wykorzystują plastik ABS.

- **Czujnik pojemnościowy,** aby korzystać z urządzenia trzeba przytrzymywać przycisk odpowiedzialny za nadawanie. Można by zwiększyć komfort użytkowania poprzez zastosowanie czujnika pojemnościowego w gałce urządzenia. Oznaczałoby to rozpoczęcie nadawania przez urządzenie natychmiast po wykryciu dotyku.

- **Połączanie trybów XYZ oraz rXrYrZ,** czujnik `MPU6050` nie pozwala na wykrywania ruchu w obu trybach jednocześnie. Potrzebny byłby inny sensor aby było to możliwe potencjalnym rozwiązaniem mogłyby się okazać magnesy oraz czujniki Halla.

- **Dedykowana płytka PCB**, stworzenie dedykowanej płytki pozwoliłoby na stworzenie linii produkcyjnej. Po projekcie płytki następnie należałoby zająć się zestawem testów jakościowych:
  - X-ray, testy z wykorzystaniem X-ray wykonywane są aby sprawdzić poprawność lutów oraz mocowania komponentów na płytce PCB. Testy za pomocą X-ray są szczególnie ważne w przypadku stosowania cyny bezołowiowej ponieważ która jest znacznie bardziej skłonna do mikropęknięć.
  - ICT, (In-Circuit-Test) jest to metoda sprawdzania poprawności działania komponentów nałożonych już na płytkę. Umożliwia wykrycie różnych rodzajów błędów, takich jak zwarcia, przerwy, błędy w rezystancji czy pojemności. W zależności od planowanego wolumenu zastosować można tester typu flying probe, lub tester z dedykowaną fiksturą.
  - FIT, (Fault Injection Test) tego typu testy polegają na celowym uszkadzaniu płytek PCB i ponownym przeprowadzeniu testów.

  Testy typu HBT oraz BIC zostały pominięte w tym zestawieniu ponieważ urządzenie nie było projektowanie z myślą o pracy w temperaturach ekstremalnych.

- **Rozwój projektu od strony software**, w sekcji omawiającej testowanie oprogramowania wspomniano o klasach które nie zostały wykorzystane. Napisano klasy które pozwalają ukryć niepotrzebny kod protowątków. Listing 5.1 znacznie upraszcza korzystanie z protowątków. Klasa `Protothread` powstała z myślą o tym aby być klasą bazową. Klasy pochodne miałby nadpisywać metodę wirtualną `operation()`, oraz ewentualnie dodawać własne zmienne które byłby potrzebne w protowątku.

```
#ifndef PROTOTHREAD
#define PROTOTHREAD

#include <pt.h>
#ifndef ARDUINO
#include <chrono>

class myTimer
{
public:
    myTimer() { start = std::chrono::steady_clock::now(); }
    unsigned long operator()()
    {
        return std::chrono::duration_cast<std::chrono::
    milliseconds>(std::chrono::steady_clock::now() - start).
    count();
        ;
    }

private:
    std::chrono::time_point<std::chrono::steady_clock> start
    ;
};
```

```
21
22 myTimer millis;
23 #endif
24
25 class Protothread
26 {
27     unsigned long threadTimer{10};
28     unsigned long timer{0};
29     pt thread;
30
31 public:
32     Protothread() {}
33     Protothread(const Protothread &p)
34     {
35         threadTimer = p.threadTimer;
36         thread = p.thread;
37     };
38     void setThreadTimer(unsigned long newTimer)
39     {
40         threadTimer = newTimer;
41     }
42     void init()
43     {
44         PT_INIT(&thread);
45     }
46     void run()
47     {
48         PT_BEGIN(&thread);
49         while (1)
50         {
51             timer = millis();
52             PT_WAIT_UNTIL(&thread, millis() - timer >
    threadTimer);
53             operation();
54         }
55         PT_END(&thread);
56     }
57     virtual void operation() = 0;
58 };
59
60 #endif // PROTOTHREAD
```

Listing 5.1. Przykładowe testy jednostkowe.

Ponieważ protowątki wykorzystują wbudowany w bibliotekę `Arduino` timer `millis()`. Została napisana klasa `myTimer` którą widać na listingu 5.1. Jest ona includowana tylko wtedy kiedy kompilator wykryje brak klasy `Arduino`. Podczas tworzenia instancji klasy `myTimer` mierzony jest czas. Dzięki przeładowaniu operatora `()` i stworzeniu obiektu o nazwie `millis` możliwe jest korzystanie z klasy jako funktora dzięki czemu klasa `Protothread` zachowuje się tak samo zarówno na Arduino jak i na komputerze.

```
#ifndef JOYSTICK_PROTOTHREAD
#define JOYSTICK_PROTOTHREAD

#include <JoystickEmulator.h>

class JoystickStrategy
{
public:
    virtual ~JoystickStrategy() = default;
    virtual void execute(const float *ypr, JoystickEmulator
   &joystick) = 0;
};

class JoystickPrototheread : public Prototheread
{
private:
    const float *ypr;
    JoystickEmulator *j;
    JoystickStrategy *strat;

public:
    void setStrategy(JoystickStrategy *newStrat)
    {
        strat = newStrat;
    }
    JoystickPrototheread(const float *yawPitchRollArray,
   JoystickEmulator &joystick)
    {
        ypr = yawPitchRollArray;
        j = &joystick;
    }
    void operation() override
    {
        strat->execute(ypr, *j);
    }
    ~JoystickPrototheread()
    {
        delete strat;
    }
};

class XYZ : public JoystickStrategy
{
    void execute(const float *ypr, JoystickEmulator &
   joystick)
    {
        for (int setterIndex = 0; setterIndex < 3;
   setterIndex++)
        {
            joystick.setXYZ(setterIndex, ypr[setterIndex]);
        }
```

```
48      }
49  };
50
51  class RXRYRZ : public JoystickStrategy
52  {
53      void execute(const float *ypr, JoystickEmulator &
    joystick)
54      {
55          for (int setterIndex = 0; setterIndex < 3;
    setterIndex++)
56          {
57              joystick.setRXRYRZ(setterIndex, ypr[setterIndex
    ]);
58          }
59      }
60  };
61
62  class DoNothing : public JoystickStrategy
63  {
64      void execute(const float *ypr, const JoystickEmulator &
    joystick) {}
65  };
66
67  #endif // !JOYSTICK_PROTOTHREAD
```

Listing 5.2. Przykładowe testy jednostkowe.

Listing 5.2 przedstawia klasę pochodną `JoystickPrototh read` od omówionej w listingu 5.1 klasy `Protothread`. Klasa pochodna ma prywatne zmienne `ypr` jest to tablica która zawiera kolejno yaw, pitch, roll. Klasa zwiera również wskaźnik do abstrakcyjnej klasy `JoystickEmulator` dzięki czemu może obsługiwać omówione w projekcie adaptery. Klasa powstawała z myślą o wzorcach projektowych strategii oraz maszyny stanów. Idea była taka aby urządzenie w zależności od stanu w którym się znajduje podmieniałoby zachowanie za pomocą metody `setStrategy()`. Na implementację samej maszyny stanów zabrakło czasu.

```
1       ...
2   TEST(PrototheadTests, RunFunctionInTheLoop)
3   {
4       struct PrototheadTester : Protothread
5       {
6           int count{0};
7           void operation() override
8           {
9               count++;
10          }
11      };
12      PrototheadTester unitUnderTest;
13      unsigned long int timer = millis();
14      while (millis() - timer < 50)
15      {
```

```
16          unitUnderTest.run();
17      }
18      EXPECT_GT(unitUnderTest.count, 1);
19  }
20
21  TEST(JoystickPrototheadTests,
        XYZStrategyCallingJoystickAdapter)
22  {
23      constexpr float EXPECTED{1};
24      JoystickMock j;
25      JoystickEmulator &&joystick = JoystickLibraryAdapter(j);
26      float ypr[3];
27      ypr[0] = EXPECTED;
28      ypr[1] = EXPECTED;
29      ypr[2] = EXPECTED;
30      JoystickPrototheadread unitUnderTest(ypr, joystick);
31      unitUnderTest.setStrategy(new XYZ);
32      EXPECT_CALL(j, setXAxis(EXPECTED)).Times(1);
33      EXPECT_CALL(j, setYAxis(EXPECTED)).Times(1);
34      EXPECT_CALL(j, setZAxis(EXPECTED)).Times(1);
35      unitUnderTest.operation();
36  }
37
38  TEST(JoystickPrototheadTests,
        RXRYRZStrategyCallingJoystickAdapter)
39  {
40      constexpr float EXPECTED{1};
41      JoystickMock j;
42      JoystickEmulator &&joystick = JoystickLibraryAdapter(j);
43      float ypr[3];
44      ypr[0] = EXPECTED;
45      ypr[1] = EXPECTED;
46      ypr[2] = EXPECTED;
47      JoystickPrototheadread unitUnderTest(ypr, joystick);
48      unitUnderTest.setStrategy(new RXRYRZ);
49      EXPECT_CALL(j, setRxAxis(EXPECTED)).Times(1);
50      EXPECT_CALL(j, setRyAxis(EXPECTED)).Times(1);
51      EXPECT_CALL(j, setRzAxis(EXPECTED)).Times(1);
52      unitUnderTest.operation();
53  }
54  ...
```

Listing 5.3. Przykładowe testy jednostkowe.

Listing 5.3 zawiera testy napisane dla klasy `Prototheread` oraz jej pochodnych. Test `RunFunctionInTheLoop` tworzy strukturę `PrototheadTester` oraz nadpisuje metodę `operation()` w której zlicza ilość wykonań pętli. Wymagane jest wykonanie więcej niż jednej pętli. Wymóg ten został postawiony ponieważ w zależności od szybkości procesora i dostępnych zasobów zmienna `count` z testu na test może mieć inne wartości ale gwarantowane jest że jeżeli wszystko działa poprawnie to wykona się więcej niż jeden raz. Z racji że `PrototheadTester` tworzone jest tylko i wyłącznie na potrzeby testów zdecydowano się na struk-

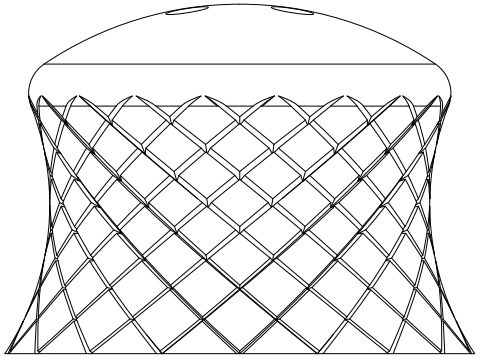
turę dzięki czemu upraszcza to dostęp do zmiennej `count` która w strukturze jest domyślnie publiczna.

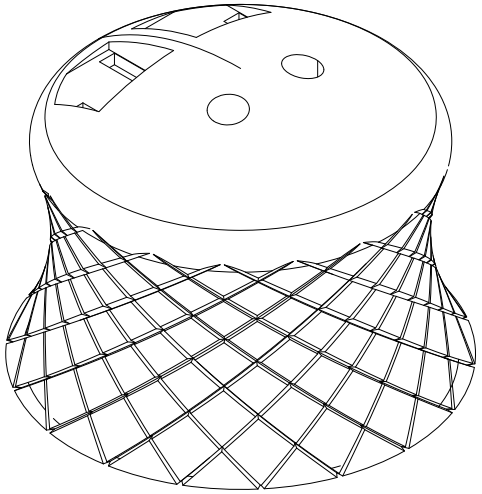
Testy `XYZStrategyCallingJoystickAdapter` oraz `RXRYRZStrategyCallingJoystickAdapter` działają w sposób analogiczny do testów omówionych w listingu 2.9. Aby omówione w tej sekcji klasy działały poprawnie należy zaimplementować wzorzec maszyny stanów.

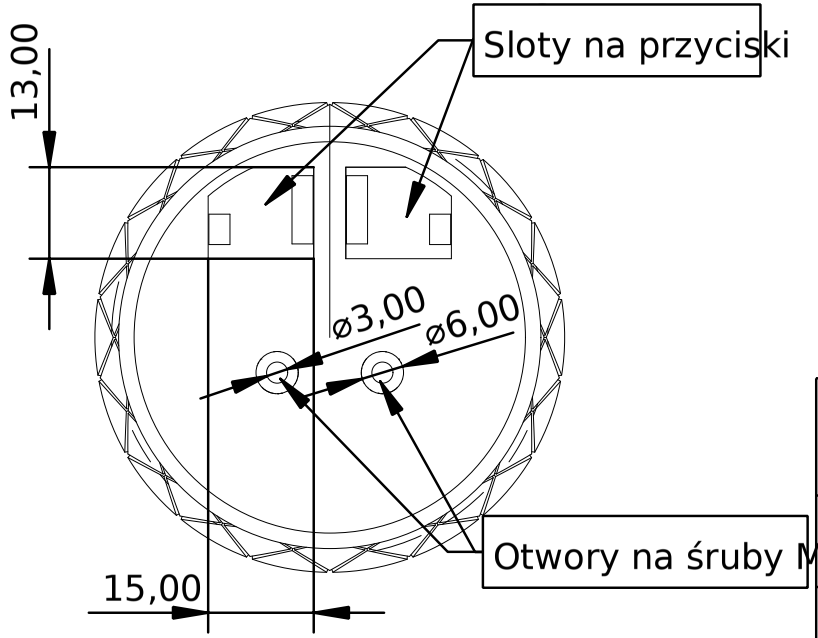
Dodatek A

Schematy urządzenia

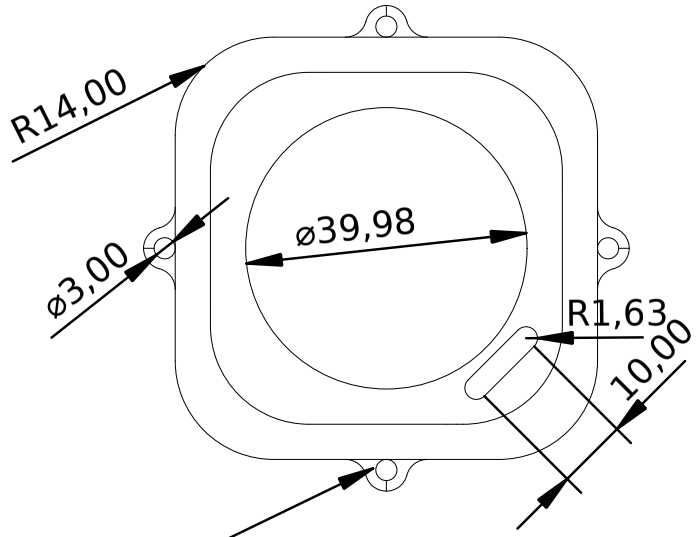
Niniejszy dodatek zawiera schematy opracowanego urządzenia.

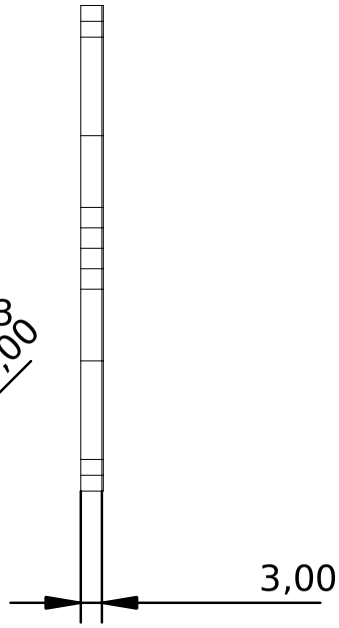


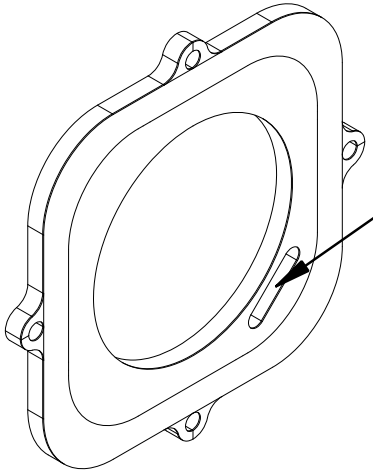




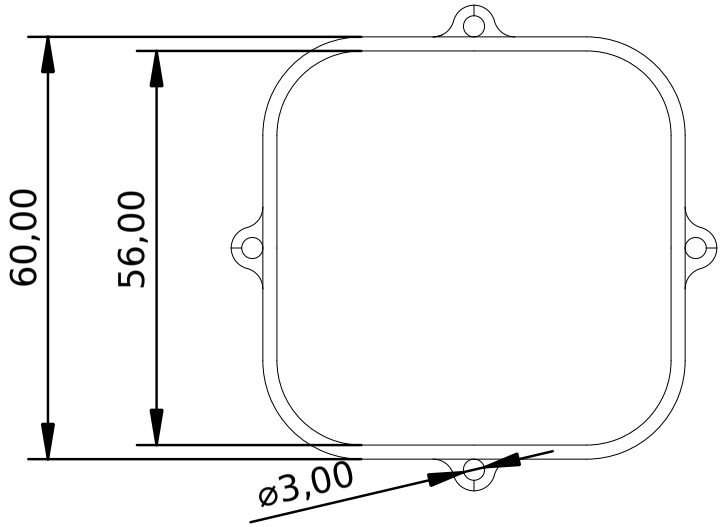
DESIGNED BY: Mikołaj Mosoń		Gałka	
DATE: 15.03.2023			
SIZE A4			
SCALE 1:1	WEIGHT (kg)	DRAWING NUMBER 3	SHEET

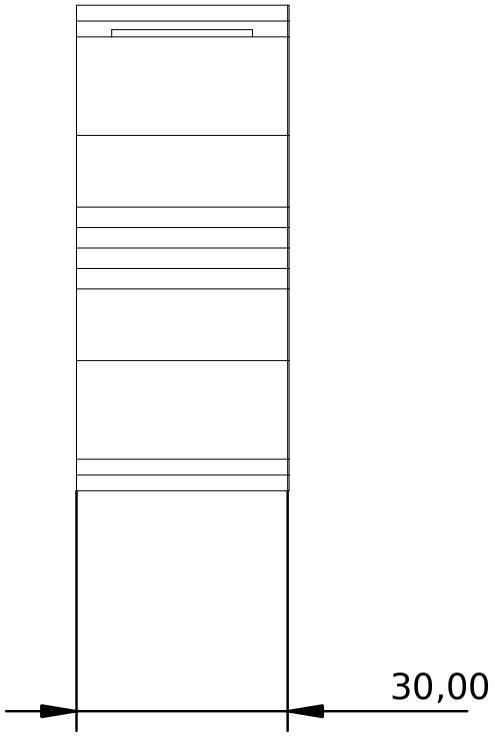


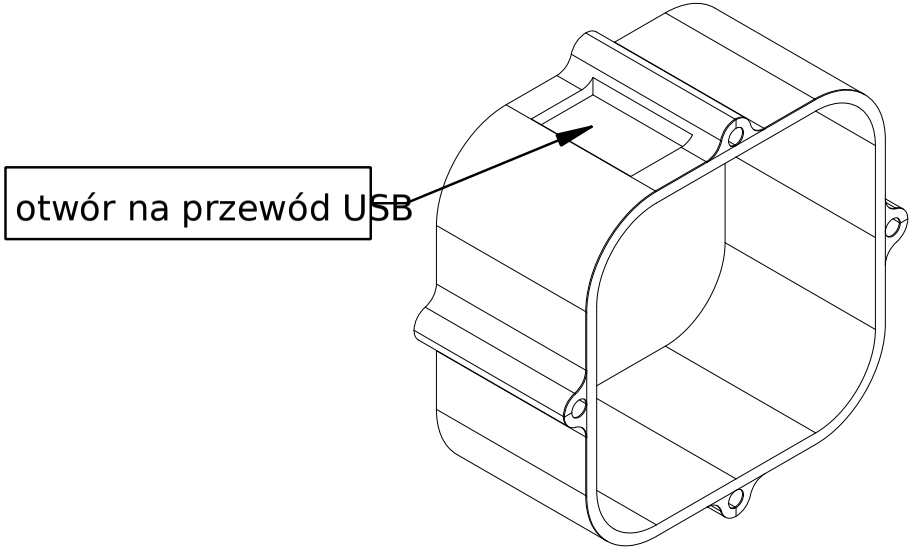




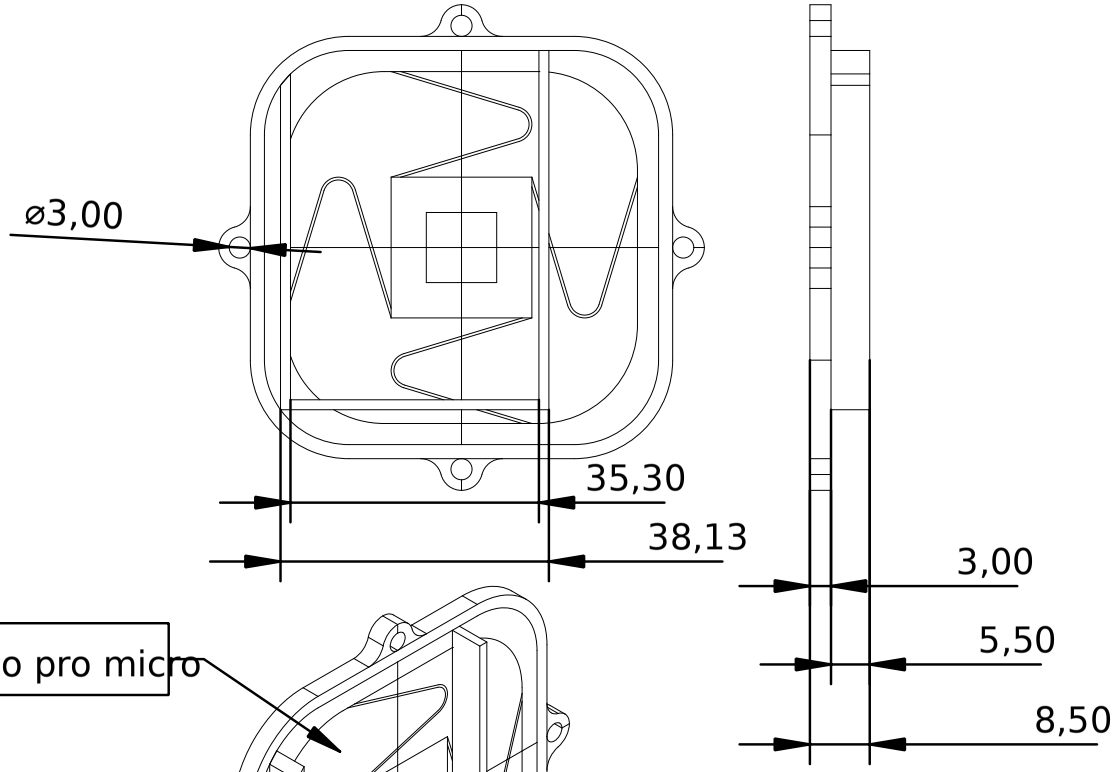
DESIGNED BY: Mikołaj Mosoń DATE: 15.03.2023		Górna płytka	
SIZE A4			
SCALE 1:1	WEIGHT (kg)	DRAWING NUMBER 4	SHEET

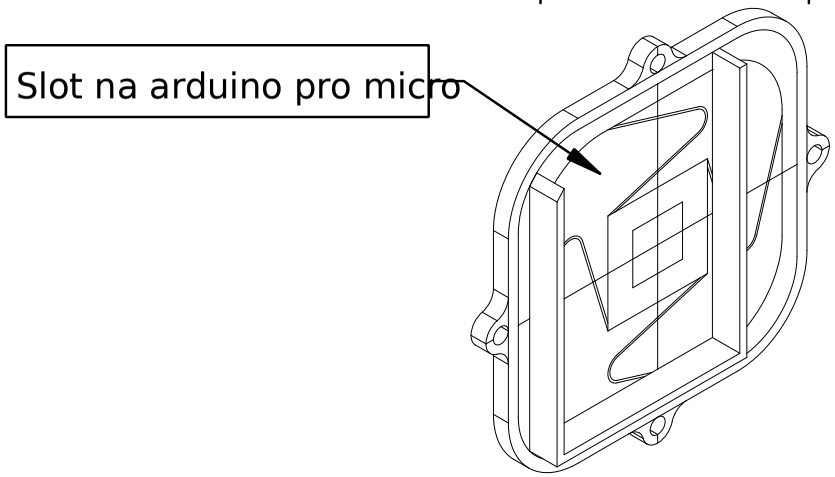




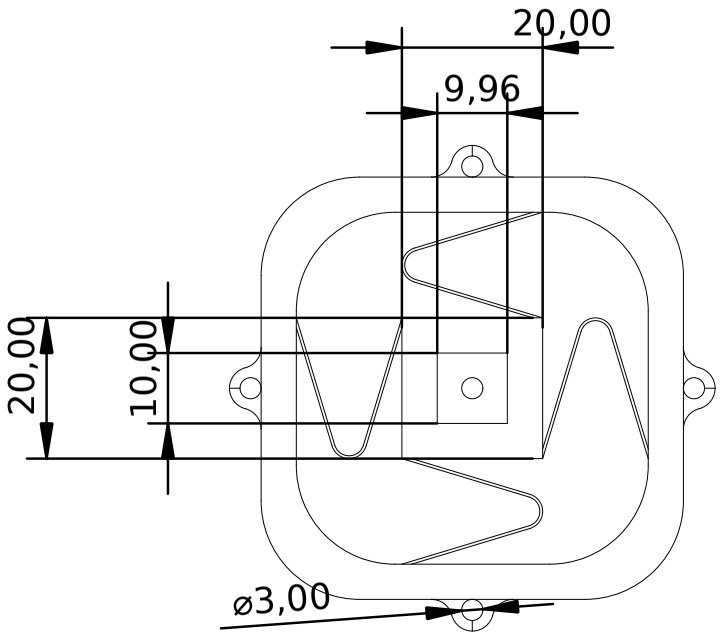


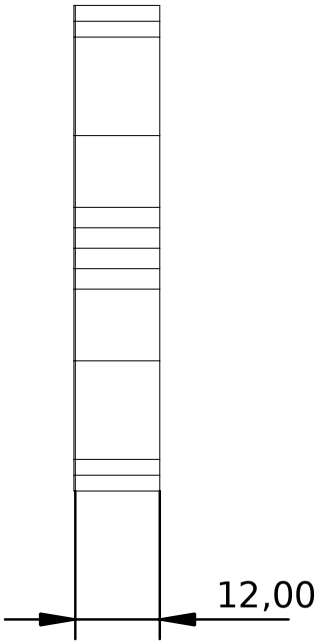
DESIGNED BY: Mikołaj Mosoń DATE: 15.03.2023		Obudowa dystans	
SIZE A4			
SCALE 1:1	WEIGHT (kg)	DRAWING NUMBER 6	SHEET

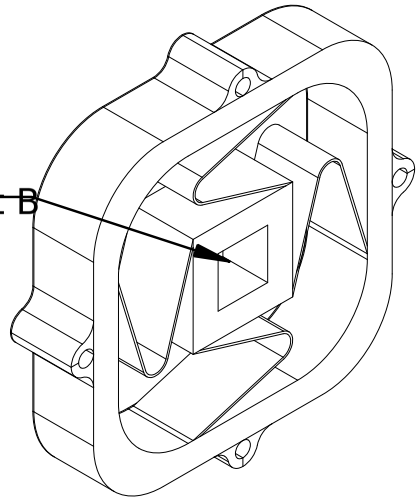




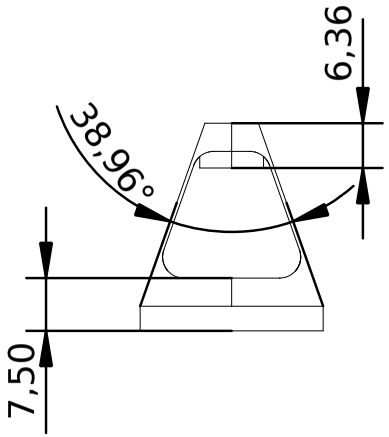
DESIGNED BY: Mikołaj Mosoń DATE: 15.03.2023		Płytka podstawy	
SIZE A4			
SCALE 1:1	WEIGHT (kg)	DRAWING NUMBER 7	SHEET

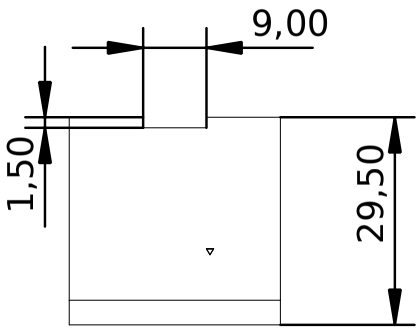


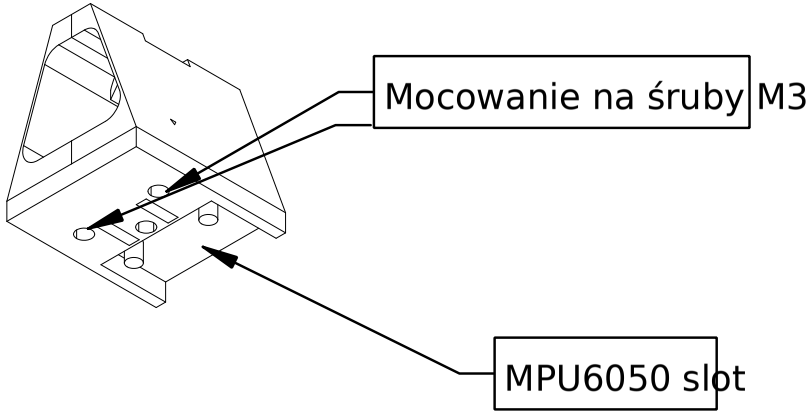


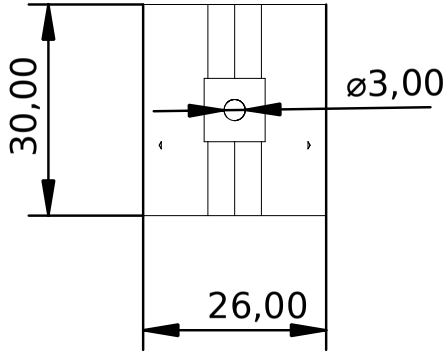


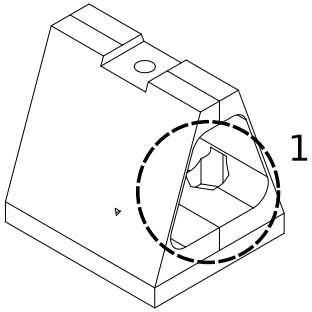
DESIGNED BY: Mikołaj Mosoń DATE: 15.03.2023		Sprężyna XY	
SIZE A4			
SCALE 1:1	WEIGHT (kg)	DRAWING NUMBER 5	SHEET

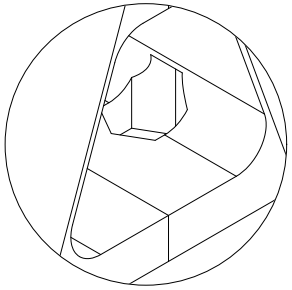












DESIGNED BY: Mikołaj Mosoń DATE: 10.03.2023		Y Spring Wariant A		G _ F _ E _
SIZE A4				D _
SCALE 1:1	WEIGHT (kg)	DRAWING NUMBER 1	SHEET	C _ B _ A _

Bibliografia

[1] Various athors. Trackball. https://en.wikipedia.org/wiki/Trackball, 2023.

[2] Various athors. Joystick. https://en.wikipedia.org/wiki/Joystick, 2023.

[3] 3DConnexion. 3DConnexion official website. https://3dconnexion.com/pl/product/spacemouse-compact/, 2023.

[4] Valve. Steam official website. https://store.steampowered.com/app/353370/Steam_Controller/, 2023.

[5] Arduino. Arduino Official Website. https://www.arduino.cc/en/software, 2023.

[6] Microsoft. Visual Studio Code Official Website. https://code.visualstudio.com/, 2023.

[7] PlatformIO. PlatformIO: An open-source ecosystem for IoT development. https://platformio.org/, 2023.

[8] FreeCAD. FreeCAD Official Website. https://www.freecadweb.org/, 2023.

[9] Create Education. Cura Software. https://www.createeducation.com/software/cura/, 2023.

[10] Tevoup. Tornado 3D Printer. https://tevoup.com/collections/tornado, 2023.

[11] Prusa3D. Prusament PLA. https://www.prusa3d.com/en/product/prusament-pla-prusa-orange-1kg/, 2023.

[12] Prusa Polymers by Joseph Prusa. *Karta Danych Technicznych Prusament PLA firmy Prusa Polymers*, Luty 2022.

[13] Google. Google Test. https://github.com/google/googletest, 2023.

[14] jtsiomb. SpaceNavD. https://spacenav.sourceforge.net/index.html, 2023.

[15] Various contributors. *Handbook of Compliant Mechanisms*. A John Wiley Sons, Ltd., Publication, John Wiley Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom, 2013.

[16] Społeczność FreeCAD. Dokumetacja freecad. `https://www.freecadweb.org/wiki/Main_Page`, 2023.

[17] Elecia White. *Making Embedded Systems: Design Patterns for Great Software*. O'Reilly and Associates, California, USA, 2011.

[18] Richard Helm John Vlissides Erich Gamma, Ralph Johnson. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, USA, 2009.

[19] Tom Mason Dave Madison, Benjamin Van Ryseghem. Arduinoxinput. `https://github.com/dmadison/ArduinoXInput`, 2023.

[20] ThrowTheSwitch. Unity. `https://github.com/ThrowTheSwitch/Unity/tree/master`, 2023.

[21] Microchip. Atmel official website. `https://start.atmel.com/`, 2023.

[22] PJRC. Teensy official website. `https://www.pjrc.com/teensy/`, 2023.

[23] Arm Holdings. Arm official website. `https://www.arm.com/products/silicon-ip-cpu`, 2023.