PRACE KÓŁ NAUKOWYCH Politechniki Rzeszowskiej w roku akademickim 2020/2021

Tom 1

Rzeszów 2021

Wydano za zgodą Rektora

Przewodniczący Rady Redakcyjnej
Bartosz TRYBUS

Rada Redakcyjna
Prac Kół Naukowych 2020/2021
Bartosz PAWŁOWICZ
Bartosz TRYBUS
Tomasz ŻABIŃSKI

W procesie wydawniczym
pominięto etap opracowania językowego.

Skład i łamanie
Joanna MIKUŁA

inteligentny budynek, arduino, prędkość obrotowa,
systemy CAD, algorytmy, system operacyjny, metodyki zwinne,
zdalne sterowanie, silniki BLDC, bezzałogowy statek powietrzny,
systemy wbudowane

p-ISBN 978-83-7934-503-8
e-ISBN 978-83-7934-513-7

Oficyna Wydawnicza Politechniki Rzeszowskiej
al. Powstańców Warszawy 12, 35-959 Rzeszów
https://oficyna.prz.edu.pl

Ark. wyd. 7,52. Ark. druk. 9,75.
Wydrukowano we wrześniu 2021 r.
Drukarnia Oficyny Wydawniczej Politechniki Rzeszowskiej, al. Powstańców Warszawy 12, 35-959 Rzeszów
Zam. nr 45/21

SPIS TREŚCI

STUDENCKIE KOŁO NAUKOWE AUTOMATYKÓW I ROBOTYKÓW „ROBO”

STUDENCKIE KOŁO NAUKOWE INFORMATYKÓW „KOD”

STUDENCKIE KOŁO NAUKOWE ELEKTRONIKI I TECHNOLOGII INFORMACYJNYCH

KOŁO

NAUKOWE

AUTOMATYKÓW

I ROBOTYKÓW

„ROBO”

Michał CIESZYŃSKI, Marcin GORAL
Konrad GUZY, Krzysztof RAK

dr inż. Tomasz ŻABIŃSKI
opiekun naukowy

STEROWANIE PRZYKŁADOWYMI PODZESPOŁAMI INTELIGENTNEGO BUDYNKU

Artykuł opisuje zasadę sterowania elementami elektronicznymi, przy pomocy mikrokontrolera Arduino MEGA. Wspomniane powyżej sterowanie realizuje się na podstawie danych odczytanych z czujników, które można wykorzystać w inteligentnym budynku. W celu uzyskania efektywności ekonomicznej używania opisywanego systemu oczekiwany jest dobór odpowiednich czujników, ze względu na ich koszt oraz niezbędne jest sporządzenie programu sterującego i przeprowadzenie testu jego działania. W artykule przedstawione są możliwości wykorzystania poszczególnych elementów elektronicznych, które mogą wchodzić w skład inteligentnego budynku.
Słowa kluczowe: inteligentny budynek, RFID, Arduino.

WPROWADZENIE

Inteligentne budynki to systemy zarządzania budynkiem, które umożliwiają sterowanie pracą większości urządzeń elektrycznych. Mają na celu poprawę bezpieczeństwa i komfortu życia użytkownika. Kolejnym celem jest optymalizacja zużycia energii elektrycznej i grzewczej poprzez sprawne zarządzanie zasobami. Większość systemów zarządzania jest ze sobą zintegrowana i powiązana, co pozwala użytkownikowi na zdalne sterowanie elementami budynku. Istotnym elementem systemu jest możliwość adaptacji do potrzeb użytkownika oraz warunków panujących wewnątrz i na zewnątrz budynku. Aktualnie na rynku istnieje wiele różnych rozwiązań systemów inteligentnego budynku. Rozwiązania te są zazwyczaj kosztowne i umożliwiają jedynie wykorzystanie urządzeń jednego producenta.

Głównym celem projektu było stworzenie systemu sterowania inteligentnym budynkiem, który nie będzie kosztowny oraz pozwoli na dalszą jego rozbudowę. W założeniu system ten można wykorzystać w budynku jednorodzinnym, będzie

posiadał możliwość sterowania urządzeniami wyjścia oraz zawierać będzie system alarmowy sygnalizujący niepożądane zdarzenia losowe.

1. DOBÓR ELEMENTÓW ELEKTRONICZNYCH INTELIGENTNEGO BUDYNKU

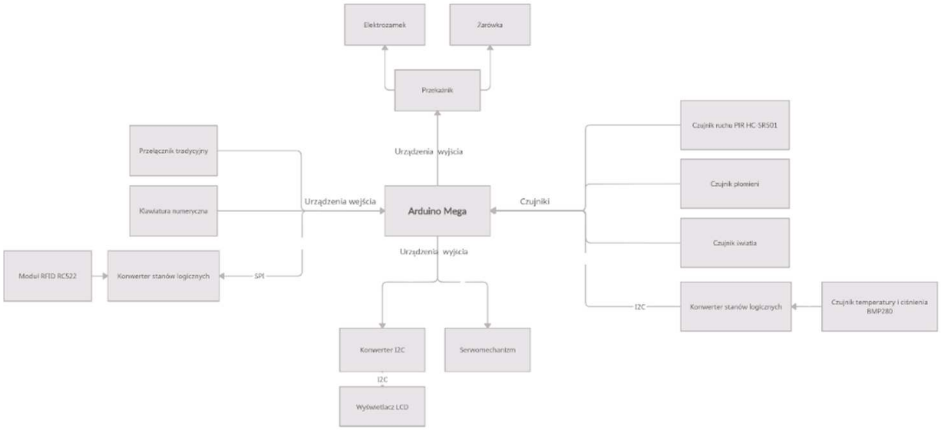
Centralą całego systemu jest Arduino MEGA, który wyposażony jest w 54-cyfrowe porty wejść/wyjść, w tym 15 wyjść PWM oraz 16 analogowych wejść, umożliwiających precyzyjny odczyt wartości z czujników. Mikrokontroler ten pozwala na wykorzystanie kilku sposobów komunikacji takich jak: UART, SPI i I^2C, które zapewniają połączenie między modułami a platformą programistyczną.

Za wyświetlanie komunikatów systemu odpowiedzialny jest wyświetlacz LCD HD44780, do którego dołączony został dedykowany konwerter I^2C. Konwerter ten umożliwia zmniejszenie wykorzystania liczby wyprowadzeń na centrali sterującej. W założeniu systemu, wyświetlacz alfanumeryczny oprócz aktualnego stanu ma ukazywać aktualną temperaturę oraz ciśnienie odczytane przez czujnik BMP280.

Za bezpieczeństwo systemu odpowiedzialny jest czujnik płomieni, który po wykryciu fali podczerwonej, przekraczającej zadany próg, ustawia stan wysoki na wyjściu cyfrowym. Czujnik ruchu PIR HC-SR501 wykorzystany został zarówno jako element do wykrywania zagrożenia, jak i poprawę komfortu użytkowania inteligentnego budynku. Identyfikację użytkownika umożliwia czytnik RFID RC522, którego komunikacja odbywa się za pośrednictwem interfejsu SPI, oraz klawiatura numeryczna. Do centrali został również podłączony moduł przekaźnika, który steruje żarówką oraz elektrozamkiem drzwiowym.

Zaprojektowany system ma również dostosowywać wysokość opuszczenia rolet względem siły nasłonecznienia. Za symulację rolet odpowiada micro serwo SG90 sterowane po przez PWM. Siłę nasłonecznienia mierzy czujnik światła wyposażony w komparator napięcia LM393 generujący na wyjściu sygnał binarny.

Moduł RFID RC522 oraz BMP280 pracuje na napięciu 3,3 V_{DC}, z kolei Arduino MEGA pracuje na logice 5 V_{DC}. W celu uniknięcia uszkodzenia modułów oraz poprawnego odczytywania informacji należy zastosować konwerter stanów logicznych.



Rys. 1. Schemat ideowy wykorzystanych elementów w projekcie
Źródło: opracowanie własne.

2. ŚRODOWISKO PROGRAMISTYCZNE I UŻYTE BIBLIOTEKI

Program sterujący poszczególnymi elementami elektronicznymi został napisany w środowisku programistycznym Arduino IDE, które jest dedykowane przez producenta. Język programowania oparty jest głównie na C/C++.

Znając już poszczególne elementy elektroniczne, które będą wykorzystane w pracy, należy dołączyć do programu biblioteki odpowiadające za poprawną komunikację z urządzeniami. Poniżej znajduje się fragment programu odpowiadający za dołączenie bibliotek.

```
#include <LiquidCrystal_I2C.h> //wyswietlacz LCD z uzyciem magi-
strali I2C
#include <Wire.h> //komunikacja I2C
#include <SPI.h> //komunikacja SPI
#include <Adafruit_BMP280.h> //czujnik temperatury i cisnienia
#include <MFRC522.h> //RFID
#include <Keypad.h> //obsluga klawiatury
#include <Servo.h> //sterowanie serwomechanizmem
```

Listing 1. Kod dołączający biblioteki

3. DZIAŁANIE PROGRAMU STERUJĄCEGO

Program sterujący składa się z trzech stanów: „Czuwania”, „Odbezpieczony” oraz „Alarm”. Podczas uruchamiania mikroprocesora (inicjalizacja programu), algorytm przechodzi w stan „Czuwania”. Podczas tego stanu program oczekuje na odczyt identyfikatora przez moduł RFID lub wpisanie czterocyfrowego hasła przy pomocy klawiatury numerycznej w określonym czasie. Wpisując z klawiatury cyfry na wyświetlaczu LCD pojawiają się znaki „*”. Program dopuszcza trzykrotne wpisanie błędnego hasła lub przyłożenie nieznanego identyfikatora. Jeżeli to nastąpi, algorytm przechodzi w stan alarmu, który jest opisany w kolejnym podrozdziale. Poniżej zamieszczony został fragment programu odpowiedzialny za sprawdzanie poprawności hasła wpisanego z klawiatury.

```
key = klawiatura.getKey(); //odczyt znaku z klawiatury
      if ( timerStart & aktualnyCzas - zapamietanyCzas > czasKodu
& !poprawny) //pomiar czasu
      {
        //koniec czasu na wpisanie hasla
        Serial.print("KONIEC CZASU ");
        wpisaneHasloString = ""; //reset wpisanego hasla
        proba++; //zwiększenie ilości prob wpisywania hasla
        timerStart = 0;
        break;
      }
      if (proba >= 3) //sprawdzenie ilości prob
      {
        //przejście w stan alarmu
        proba = 0;
        stan = 0;
        stanBMP(stan, access); //komunikat na LCD o alarmie
        break;
      }
      if (key) //sprawdzenie czy zostala wybrana wartosc z klawia-
tury
      {
        flaga = 1;
        wpisaneHasloString = wpisaneHasloString + String(key);
        j++;
        wpisywanieHasla(j);
        if (j == 1)
        { //uruchomienie timera
          Serial.print("Start timera");
          timerStart = 1;
          zapamietanyCzas = aktualnyCzas;
        }
        else if (j == 4)
        { //wpisanie czterech znakow
          j = 0;
```

```
          timerStart = 0;
          //sprawdzanie zgodnosci hasla z uzytkownikami
          if (wpisaneHasloString == hasloChild)
          {
            Serial.print("Child");
            otworzZamek = 1;
            digitalWrite(ZAMEK, LOW); //otwarcie zamka
            zapamietanyCzasZamka = aktualnyCzas;
            access = 2; //przyznanie dostepu
            stan = 1; //zmiana stanu na „odbezpieczony”
            proba = 0;
            stanBMP(stan, access);
          }
          else if (wpisaneHasloString == hasloMaster)
          {
            Serial.print("Master");
            otworzZamek = 1;
            digitalWrite(ZAMEK, LOW);
            zapamietanyCzasZamka = aktualnyCzas;
            access = 1;
            stan = 1;
            proba = 0;
            stanBMP(stan, access);
          }
          else //niepoprawne haslo
          {
            lcd.clear();
            lcd.print("Bledne haslo");
            proba++;
          }
          //reset wpisanego hasla
          flaga = 0;
          wpisaneHasloString = "";
        }
        key = "";//czyszczenie znaku z klawiatury
      }
```

Listing 2. Kod sprawdzający wpisane hasło

Tekst na wyświetlaczu alfanumerycznym odświeżany jest co pewien okres czasu lub po przejściu na kolejny stan. W pierwszej linii pokazywane są komunikaty na temat systemu, z kolei w drugiej linijce pokazywana jest aktualna temperatura i ciśnienie odczytane z czujnika BMP280. Program, przechodząc do stanu odbezpieczonego ukazuje na LCD osobę „zalogowaną” do systemu.

```
  aktualnyCzas = millis(); //zapis obecnego czasu licznika
  if (aktualnyCzas - zapamietanyCzasCzujnika >= czasCzujnika &
!flaga)
  {
    stanBMP(stan, access); wywolanie funkcji
    zapamietanyCzasCzujnika = aktualnyCzas;// zapis poprzedniego
czasu
  }
```

Listing 3. Fragment kodu odpowiadający za okresowe wywołanie funkcji *stanBMP()*

```
void stanBMP(int stan, int access)
{
  lcd.clear(); //”oczyszczanie” wyświetlacza LCD
//odczyt wartosci z czujnika BMP280
  temp1 = bmp.readTemperature();
  cisnienie1 = bmp.readPressure() / 100;
  switch (stan)
  {//ukazywanie stanu systemu oraz dostepu na wyświetlaczu
    case 0:
      lcd.print("ALARM");
      break;
    case 1:
      lcd.print("ODBEZP ");
      if (access == 1)
      {
        lcd.print("Master");
      }
      else if (access == 2)
      {
        lcd.print("Child ");
      }
      break;
    case 2:
      lcd.print("UZBROJ");
      break;
  }
//ukazywanie wartosci temperatury i ciśnienia
  lcd.setCursor(0, 1);
  lcd.print(temp1);
  lcd.print("*C");
  lcd.print(cisnienie1);
  lcd.print("kPa");
}
```

Listing 4. Funkcja *stanBMP()*, odpowiadająca za wypisywanie informacji na wyświetlaczu LCD

Będąc w stanie odbezpieczonym, mamy możliwość sterowania poszczególnymi komponentami (listing 5). Żarówkę możemy uruchomić na dwa sposoby, pierwszy to przy pomocy tradycyjnego włącznika. Drugi sposób jest automatyczny i uruchamia on żarówkę na ustalony czas, gdy czujnik światła wykryje ciemność oraz czujnik PIR wykryje ruch. Kolejnym elementem, który jest sterowany przez program, to roleta. Jej położenie zależne jest od wartości z czujnika zmierzchu, gdy fotorezystor wykryje natężenie świetlne, roleta jest otwarta, w przeciwnym wypadku roleta jest zamknięta. Poniżej zaprezentowano fragment programu odpowiedzialny za sterowanie poszczególnymi komponentami i funkcję odpowiedzialną za sterowanie roletami.

```
      if  ((digitalRead(PIR)  &  digitalRead(ZMIERZCH))  |  digi-
talRead(WLACZNIK_SWIATLA))
      {//wlaczenie swiatla
        digitalWrite(SWIATLO, LOW);
      }
      else digitalWrite(SWIATLO, HIGH); //zgaszenie swiatla

      if (digitalRead(ZMIERZCH) == HIGH & !roletaZamknieta)
      {//opuszczenie rolet
        Serial.print("zamykanie rolety /n");
        roleta = 2;
      }
      if (digitalRead(ZMIERZCH) == LOW & roletaZamknieta)
      { //otwarcie rolet
        Serial.print("otwieranie rolety /n");
        roleta = 1;
      }
```

Listing 5. Sterowanie światłem i roletą w stanie odbezpieczonym

```
void sterowanieRoleta(int stan)
{
  while (roleta != 0)
  {
    switch (stan)
    {
      case 1: //wykrycie swiatla
        if (pos < 180)
        {//podnoszenie rolety
          roleta1.write(pos);
          pos++;
        }
        else
        {//roleta otwarta
          roleta1.write(180);
          roleta = 0;
          roletaZamknieta = 0;
```

```
        }

        break;

      case 2: //wykrycie zmierzchu
        if (pos > 0)
        {//opuszczanie rolety
          roleta1.write(pos);
          pos--;
        }
        else //roleta zamknieta
        {
          roleta1.write(0);
          roleta = 0;
          roletaZamknieta = 1;
        }
    }
  }
}
```

Listing 6. Funkcja sterująca roletą w zależności od natężenia światła

W stanie „odbezpieczonym” istnieje możliwość uzbrojenia systemu. Uzbrojenie systemu polega na przejściu programu w stan „czuwania”, podczas którego algorytm monitoruje budynek przy pomocy czujników. Aby system przeszedł w ten stan, należy wcisnąć przycisk „*” z klawiatury numerycznej. Wtedy następuje otwarcie elektrozamka, a następnie po upłynięciu wyznaczonego w programie czasu program załącza czujniki aktywujące alarm i zamyka elektrozamek. W przypadku wykrycia zagrożenia takiego jak wykrycie przez czujnik PIR ruchu lub wykrycie ognia przez czujnik płomieni, system przechodzi w stan „Alarmu”.

```
      if (klawiatura.getKey() == '*')
      {
        lcd.clear();
        lcd.print("Uzbrajanie");
        digitalWrite(ZAMEK, LOW); //zamek otwarty
        delay(5000); //opoznienie 5 sekund
        digitalWrite(ZAMEK, HIGH); //zamek zamkniety
        stan = 2; //stan uzbrojenia
         roleta = 2; //zamykanie rolet

      }
```

Listing 7. Fragment programu odpowiedzialny za uzbrojenie systemu

4. STAN „ALARM”

Stan alarmowy może zostać uruchomiony na trzy sposoby. Pierwszy to trzykrotne wpisanie błędnego hasła lub przyłożenie błędnego identyfikatora. Kolejny sposób to wykrycie ruchu przez czujnik PIR, podczas gdy system jest w stanie czuwania. Ostatnim sposobem jest przekroczenie wartości promieniowania podczerwonego poza zadany próg. Niezależnie od obecnego stanu, wymusza to w programie przejście w stan alarmu.

```
if (digitalRead(CZUJ_PLOMIENI) == LOW)
  { //wykrycie zrodla ognia
    stan = 0; //zmiana stanu na „alarm”
    digitalWrite(ZAMEK, LOW); //otwarcie zamka
    stanBMP(stan, access);
  }
```

Listing 8. Warunek odpowiadający za wykrycie ognia

Przechodząc w stan alarmowy, program pokazuje na wyświetlaczu alfanumerycznym komunikat o alarmie oraz rozpoczyna migotanie żarówką LED. Aby opuścić ten stan, należy wpisać hasło użytkownika lub przyłożyć transponder, który jest przypisany do użytkownika. Po wykonaniu jednej z powyższych czynności, program przechodzi w stan odbezpieczony.

```
case 0: //stan alarmu
  if (!roletaZamknieta)
  {
    Serial.println("zamykanie rolety");
    roleta = 2;
  }
  //migotanie zarowki//
  if (digitalRead(SWIATLO) == LOW & aktualnyCzas - zapamietanyCza-
sAlarmu >= czasAlarmu)
  {
    digitalWrite(SWIATLO, HIGH);
    zapamietanyCzasAlarmu = millis();
  }
  else if (digitalRead(SWIATLO) == HIGH & aktualnyCzas - zapamie-
tanyCzasAlarmu >= czasAlarmu)
  {
    digitalWrite(SWIATLO, LOW);
    zapamietanyCzasAlarmu = millis();
  }
  //
```

```
  odczytRFID(); //odczyt ID z identyfikatora

  ... //fragment sprawdzajacy poprawnośc ID identyfikatora

  key = klawiatura.getKey(); //przypisanie wartosci z klawiatury do
  if (key) //wykrycie wcisnietego klawisza
  {
    flaga = 1;
    wpisaneHasloString = wpisaneHasloString + String(key);
    Serial.print(key);
    j++;
    wpisywanieHasla(j);

... //sprawdzanie poprawności hasla z klawiatury
}

  break;
```

Listing 9. Fragment programu zawierający stan alarmu

5. ODCZYT IDENTYFIKATORA RFID

Moduł RC522 po wykryciu identyfikatora przesyła informacje do centrali Arduino MEGA. Komunikacja ta odbywa się przy pomocy magistrali SPI. Magistrala ta pozwala na komunikację w danej chwili wyłącznie z jednym urządzeniem, gdzie za wybór urządzenia odpowiedzialna jest linia SS (ang. *Slave Select*).

Funkcja służąca do odczytu UID identyfikatora pozwala na pracę z wieloma modułami RFID. Podczas gdy czytnik/programator wykryje transponder, to algorytm zapisuje UID identyfikatora jako wartość typu String. Następnie ta wartość jest porównywana z zapisanymi w programie UID. W przypadku braku zgodności, następuje zwiększenie ilości błędnych logowań, z kolei gdy występuje zgodność, to udzielany jest dostęp do sterowania inteligentnym budynkiem i zerowana jest wartość błędnych logowań. Poniżej przedstawiono funkcję odpowiedzialną za odczyt UID transpondera oraz fragment programu odpowiedzialny za sprawdzenie poprawności identyfikatora.

```
void odczytRFID()
{
//petla sprawdzajaca poszczegolne czytniki RFID
  for (uint8_t reader = 0; reader < NR_OF_READERS; reader++)
  {
    if         (mfrc522[reader].PICC_IsNewCardPresent()         &&
mfrc522[reader].PICC_ReadCardSerial())
    {//identyfikator przylozony do modulu
      Serial.print(F("Reader "));
      Serial.print(reader); //pokazanie nr. identyfikatora
```

```
      Serial.print(F("UID karty: "));

//konwersja ID identyfikatora na zmienna typu String
//oraz usuniecie niepotrzebnych znakow
      uidString   =   dump_byte_array(mfrc522[reader].uid.uidByte,
mfrc522[reader].uid.size);
      Serial.println(uidString);
      Serial.println();
      //rozlaczenie z identyfikatorem
      mfrc522[reader].PICC_HaltA();
      mfrc522[reader].PCD_StopCrypto1();
    }
  }
}
```

Listing 10. Funkcja odczytująca dane z identyfikatorów RFID

```
odczytRFID(); //wywolanie funkcji odczytującej ID identyfikatora
RFID
//sprawdzenie poprawności UID identyfikatora
if (uidString == codeMaster)
{   //uzytkownik master
       Serial.print("Master");
       otworzZamek = 1; //otwarcie zamka na wcześniej ustalony okres
czasu
       digitalWrite(ZAMEK, LOW);
       zapamietanyCzasZamka = aktualnyCzas;
       access = 1; //nadanie praw dostepu
       stan = 1; //zmiana stanu na odbezpieczony
       proba = 0; //wyzerowanie ilości prob logowan
       stanBMP(stan, access); //wyświetlanie stanu systemu
       uidString = ""; //resetowanie UID
       break;
}
else if (uidString == codeChild)
{   //uzytkownik child
       Serial.print("Kid");
       otworzZamek = 1; //otwarcie zamka na wcześniej ustalony okres
czasu
       digitalWrite(ZAMEK, LOW);
       zapamietanyCzasZamka = aktualnyCzas;
       access = 2; //nadanie praw dostępu child
       stan = 1;
       proba = 0; //wyzerowanie ilości prob logowan
       stanBMP(stan, access); //wyświetlanie stanu systemu
       uidString = "";//resetowanie UID
       break;
}
else if (uidString != "")
```

```
{
    //nieznane UID
       proba++; //zwiekszenie ilosci blednych prob logowan
       uidString = "";
}
```

Listing 11. Fragment programu sprawdzający zgodność UID identyfikatorów RFID

6. PODSUMOWANIE

Nowoczesne technologie pozwalają zmienić nasze życie na lepsze. W projekcie zaprezentowano kilka sposobów rozwiązań, które można wykorzystać w inteligentnym budynku. Co ważne, rozwiązania te opierają się na ogólnodostępnych elementach elektronicznych charakteryzujących się niską ceną. Jedynym odstępstwem od takiego podejścia byłby przypadek konieczności zastosowania w systemie rolet ochronnych. W takim przypadku ze względów technicznych system musiałby być doposażony o bardziej kosztowny i mocniejszy serwomechanizm.

Ogólnie, można stwierdzić, że niewielkim nakładem środków poprawiona zostanie ergonomia oraz bezpieczeństwo naszego mieszkania. Również system ten można rozbudowywać o kolejne elementy elektroniczne, które będą miały wpływ na poprawę funkcjonalności budynku.

LITERATURA

1. Jepson B., Margolis M., Weldin N.R., *Arduino. Przepisy na rozpoczęcie, rozszerzanie i udoskonalanie projektów*. Wydanie III, Wydawnictwa Helion, 2021.
2. Prata S., *Język C++. Szkoła programowania*, Wydawnictwa Helion, 2013.

ŹRÓDŁA INTERNETOWE

3. Czujnik ciśnienia i temperatury BMP280, *Data sheet 1.23 BMP280 Digital Pressure Sensor BOSCH*, https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf Listopad 2020 (dostęp: 24.05.2021).
4. Dokumentacja modułu RFID MF RC522 13,56MHz SPI, *MFRC522 Standard performance MIFARE and NTAG frontend,* https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf (dostęp: 25.05.2021).

Michał CIESZYŃSKI, Marcin GORAL
Konrad GUZY, Krzysztof RAK

dr inż. Tomasz ŻABIŃSKI
opiekun naukowy

MAKIETA AUTOMATYCZNEGO WIELOPOZIOMOWEGO MAGAZYNU

W niniejszym artykule opisano projekt wielopoziomowego magazynu wykorzystującego technikę RFID do identyfikacji poszczególnych obszarów magazynowania. Każdy przedmiot w momencie umieszczania go w magazynie otrzymuje numer identyfikatora RFID i zostaje umieszczony w najbliższym możliwym miejscu. W przypadku wywołania numeru identyfikatora z przypisanym miejscem, następuje rozpoczęcie procedury zwracania przedmiotu. Makieta została wykonana przy użyciu techniki druku 3D, wycinania laserowego oraz ogólnodostępnych części konstrukcyjnych i elektronicznych.
Słowa kluczowe: magazyn, arduino, RFID, automatyka.

WPROWADZENIE

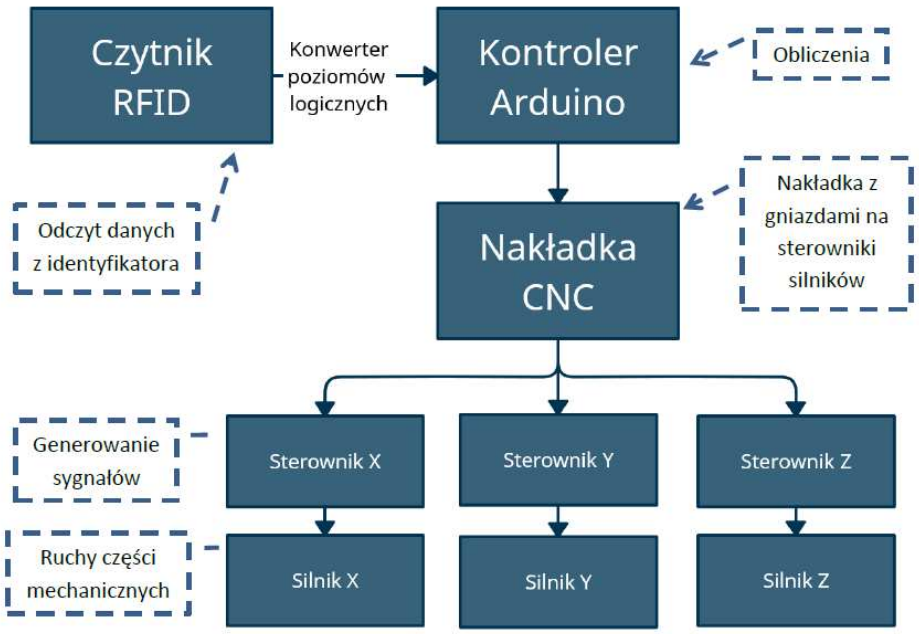
Obecnie obserwuje się znaczący wzrost sprzedaży produktów przez Internet. Handel internetowy pozwala na zrezygnowanie z lokalnych placówek sklepu, a co za tym idzie, z dodatkowych kosztów takich jak wynajem powierzchni, pracowników czy utrzymanie obiektu. Niesie za sobą jednak konieczność posiadania dobrze funkcjonującego magazynu. Sprawne radzenie sobie z wysyłaniem zamówień to często klucz do sukcesu. Nic dziwnego, iż największe sklepy internetowe np. „Amazon” inwestują duże kwoty w technologie autonomicznych robotów przewożących ładunki czy inteligentnych, samoobsługowych magazynów.

1. OGÓLNA KONCEPCJA

Makieta ma prezentować działanie wielopiętrowego magazynu z automatycznym przenośnikiem, używającego techniki RFID do przemieszczania zasobów. Posiada on trzy piętra, na każdym sześć miejsc ułożonych po sześcianie. Na samym środku systemu znajduje się system transportowy. Dzięki takiemu umiejscowieniu każde z miejsc jest w równej odległości od przenośnika. Składa się on

z windy poruszającej się w osi pionowej, przenośnika odbierającego i odkładającego przedmioty oraz obrotnicy obracającej przenośnik na jedno z sześciu miejsc na piętrze.

Po przyłożeniu identyfikatora do czytnika, system sprawdza, czy jego numer znajduje się w bazie danych, czyli czy jest mu przypisane konkretne miejsce. Jeżeli tak, oznacza to chęć pozyskania przedmiotu z przestrzeni magazynowej przez użytkownika, więc przenośnik pobiera go i transportuje na miejsce odbioru. Jeżeli nie, oznacza to chęć odłożenia przedmiotu do obszaru składowania. System wybiera najbliższe wolne miejsce, po czym przypisuje mu numer identyfikatora i rozpoczyna procedurę transportu.



Rys. 1. Schemat działania makiety
Źródło: opracowanie własne.

2. KONSTRUKCJA

2.1. Główny szkielet konstrukcji

Główny szkielet konstrukcji wyznaczają profile aluminiowe „T-slot" 2020 o długościach 600 i 300 milimetrów. Dzięki ich uniwersalnej konstrukcji można zamocować do nich dowolny przedmiot na dowolnej długości za pomocą specjalnej nakrętki „T-nut" i zwykłej śrubki. Profile te zostały połączone ze sobą wydrukowanymi łącznikami z materiału PET-G. Posiadają one 53 milimetrów wysokości, długości i szerokości. Z racji niedokładności technologii druku 3D, otwory, w które wsuwane są profile, zostały dodatkowo powiększone o 0,4 milimetra. Ewentualne luzy niwelowane są śrubami montażowymi. Każdy łącznik posiada piętnaście takich otworów.

Z racji wielkości konstrukcji, podstawa parkingu, pokrywa elektroniki, dach i każde z pięter zostały wykonane ze sklejki drewnianej o grubości trzech milimetrów przy użyciu plotera laserowego. Dzięki wykonaniu tych elementów w jednym kawałku sztywność całej makiety znacznie się zwiększyła, przez co nie było potrzeby montowania dodatkowych profili aluminiowych i ich łączników na górze makiety. Dodatkowo znacznie zmniejszyło to czas projektowania i montowania, oraz istotnie zniwelowało koszty wykonania całości.

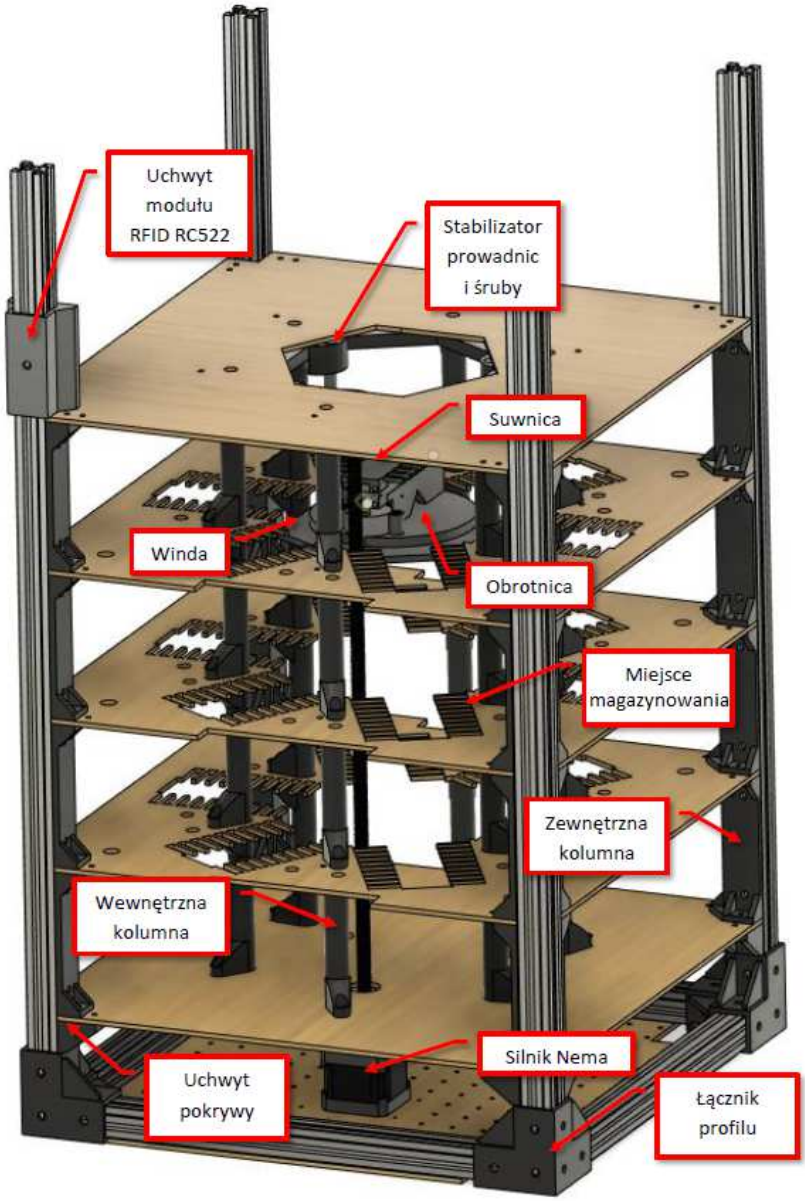
Miejsca magazynowania posiadają szereg wystających belek, które to mijają się z elementem przenośnika podczas odkładania/zabierania przedmiotu. Ich wymiary są od siebie ściśle zależne. Błędne dobranie długości, szerokości czy odstępów między nimi mogłoby skutkować uszkodzeniem przenośnika lub innych elementów.

Każde z pięter połączone jest z sąsiadującym przy użyciu kolumn wewnętrznych, jak i zewnętrznych. Wewnętrzne zostały zaprojektowane tak, aby możliwe było wykonanie tego samego modelu wielokrotnie i zatrzaśnięcie ich końców ze sobą. Dodatkowo aby wzmocnić połączenie, każda z nich posiada okrągły otwór na łeb śruby i sześciokątny otwór na nakrętkę, który znacząco ułatwił proces montażu. Kolumny zewnętrzne znajdują się na krawędziach każdego piętra. Dzięki zaprojektowanym otworom mają możliwość przytwierdzenia ich do ramy z profili aluminiowych.

Druk 3D został również użyty do wykonania kilku dodatkowych elementów konstrukcyjnych takich jak:

- stabilizator prowadnic i śruby – ma za zadanie utrzymywanie ich idealnie w pionie, co przekłada się na płynny ruch windy w osi pionowej,
- uchwyt czytnika RFID – mocowanie modułu bezpośrednio do dowolnego profilu aluminiowego,
- uchwyt prowadnic – solidny blok z dwoma otworami na prowadnice i śruby przytwierdzające go do podstawy parkingu,
- uchwyt na silnik windy – przytwierdza silnik do podstawy parkingu,

- uchwyt pokrywy elektroniki – w niewielkim zakresie pozwala regulować wysokość położenia pokrywy.



Rys. 2. Makieta magazynu
Źródło: opracowanie własne.

2.2. Przenośnik

Ruch i jego blokowanie w osi pionowej odbywa się za pomocą śruby trapezowej o średnicy 8 milimetrów, długości 400 milimetrów oraz skoku równym 8 milimetrów. Oznacza to, że przy pełnym obrocie nakrętka na śrubie przemieści się o 8 milimetrów. Przekazanie obrotów z silnika krokowego realizowane jest poprzez elastyczne sprzęgło przykręcone na jego wale. Aby zapewnić stabilność, całość ślizga się przy pomocy łożysk liniowych po dwóch prowadnicach o długości 400 milimetrów i średnicy 8 milimetrów. Oprócz łożysk i nakrętki z brązu, winda posiada również wycięcie na silnik krokowy z przekładnią, który obraca suwnicą.

Suwnica jest fragmentem systemu, przenoszącym przedmiot poza obrys windy. Składa się z ośmiu części: silnika, dwóch elementów transportujących sprzężonych ze sobą, zębatki, czterech wałków o długości 80 milimetrów oraz podstawy. Za pomocą przekładni liniowej wkomponowanej w strukturę jednego z elementów transportujących, ruch z wału silnika zostaje zamieniony z obrotowego na posuwisty. Przez wspomnianą wcześniej niedokładność technologii druku 3D, niektóre detale w celu idealnego spasowania należało poddać obróbce, szlifowaniu czy rozwierceniu.

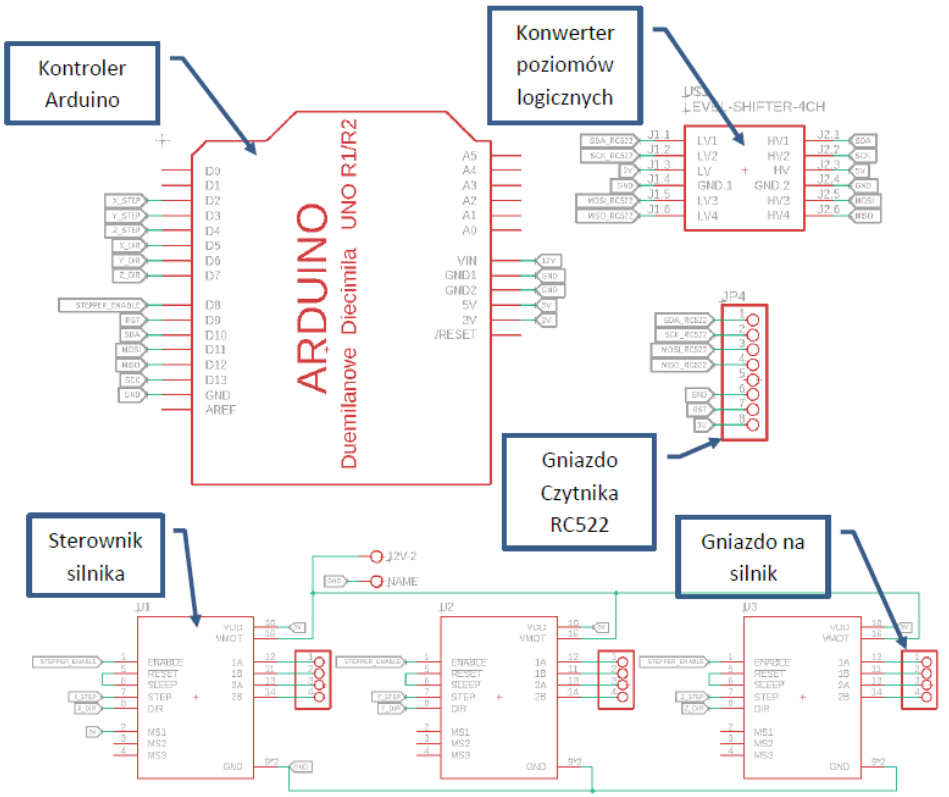
2.3. Elektronika

Elementem sterującym pracą modelu jest platforma Arduino UNO. Jest to pojedyncza płytka drukowana z mikrokontrolerem ATmega328P firmy Atmel. Posiada 31 wyprowadzeń typu goldpin, gniazdo USB typu B, gniazdo zasilania, a także elementy niezbędne do prawidłowej pracy, takie jak stabilizator napięcia, rezonator kwarcowy i szereg rezystorów z kondensatorami. Jest ona bardzo popularna wśród hobbystów, elektroników i programistów mniej i bardziej zaawansowanych. Jej zadaniem jest przetwarzanie informacji na podstawie zebranych danych z modułu RFID, interpretowanie ich i sterowanie silnikami w zależności od otrzymanych wyników. Do odczytu identyfikatorów RFID został użyty moduł RC522. Pracuje on na częstotliwości 13,5 MHz i logice 3,3 V. Posiada wbudowaną antenę w postaci ścieżki na płycie PCB. Z uwagi na to, że Arduino UNO działa na logice 5 V, konieczne było zastosowanie wielokanałowego konwertera stanów logicznych.

Do sterowania silnikami krokowymi zostały użyte moduły sterowników A4988. Są to układy scalone firmy Allegro MicroSystems, umieszczone na płytce PCB razem z niezbędnymi rezystorami, kondensatorami, podpisanymi wyprowadzeniami i potencjometrem służącym do ustawiania maksymalnego prądu płynącego przez cewki silnika. Układ może być zasilany napięciami od 8 do 35 V i od 3 do 5,5 V w części logicznej. Do sterowania silnikiem służą wyprowadzenia DIR – kierunek obrotów i STEP – ruch o określoną wartość kroku silnika (do wyboru: pełny krok, 1/2, 1/4, 1/8 i 1/16 kroku). Aby wygodnie podłączyć sterowniki, wykorzystana została specjalna nakładka do Arduino z gniazdami na cztery moduły

sterowników, gniazdami na silniki, złączem zasilnia oraz wyprowadzeniami do sterowania i podłączenia dodatkowych komponentów.

Do poruszania przenośnikiem zostały wykorzystane trzy modele silników krokowych, z racji na łatwość ich precyzyjnego pozycjonowania. Pierwszy z nich to silnik bipolarny NEMA17 - KS42STH34. Napędza on śrubę trapezową odpowiadającą za ruch w pionie. Znajduje się na samym dole makiety przymocowany do całości wydrukowanym uchwytem. Pozostałe dwa to silniki o oznaczeniu 28BYJ-48. W odróżnieniu od poprzedniego modelu, te silniki są unipolarne. Oznacza to konieczność lekkiej modyfikacji jego wewnętrznej budowy. Należy przerwać połączenie pomiędzy cewkami oraz zamienić ułożenie przewodów na wtyczce. Dzięki małym wymiarom i wbudowanej przekładni 64:1 stanowi on bardzo popularny i tani model napędu w tego typu konstrukcjach.



Rys. 3. Schemat elektroniczny
Źródło: opracowanie własne.

3. PROGRAM STERUJĄCY

3.1. Biblioteki i deklaracje

Cały program został napisany w środowisku Arduino IDE, które jest stworzone specjalnie pod kontrolery z rodziny Arduino. Język programowania opiera się na C/C++, jednak posiada względem niego uproszczoną i zmienioną składnię. Oprogramowanie zajmuje niewiele miejsca i jest proste w obsłudze, dzięki czemu zyskało szerokie grono odbiorców.

Aby usprawnić działanie programu i ułatwić proces programowania, w projekcie skorzystano z kilku bibliotek do obsługi modułu RFID (MRFC522.h), sterownika silnika czy komunikacji SPI (SPI.h).

Do obsługi i sterowania silnikami krokowymi służy biblioteka SpeedyStepper.h. Przed wywołaniem ruchu należy najpierw określić maksymalne przyspieszenie i prędkości oraz ile kroków musi wykonać sterownik, aby wał silnika wykonał pełen obrót, lub przy zastosowaniu przekładni, ile kroków należy wykonać, aby uzyskać przesunięcie o jeden milimetr. Twórca zadbał również o automatyczne przyspieszanie przy ruszaniu i zwalnianie przy zbliżaniu się do celu. Oprócz tego biblioteka umożliwia stosowanie dwóch rodzajów ruchu: absolutnego i relatywnego. Różnica polega na określeniu pozycji początkowej, w odniesieniu do której następuje przemieszczenie. Aby odnieść się do aktualnego położenia silnika, należy użyć ruchów relatywnych, natomiast, aby odnieść się do pozycji, w której był silnik podczas wywołania pierwszej komendy, należy używać ruchów absolutnych. Po załączeniu bibliotek należało zdefiniować wyprowadzenia i zadeklarować używane zmienne.

```
const int MOTOR_X_STEP_PIN = 2;   //silnik X na wyprowadzeniu 2
const int MOTOR_Y_STEP_PIN = 3;   //silnik Y na wyprowadzeniu 3
const int MOTOR_Z_STEP_PIN = 4;   //silnik Z na wyprowadzeniu 4
const int MOTOR_X_DIR_PIN = 5; //kierunek silnika X na wyprowadzeniu 5
const int MOTOR_Y_DIR_PIN = 6; // kierunek silnika Y na wyprowadzeniu 6
const int MOTOR_Z_DIR_PIN = 7; // kierunek silnika Z na wyprowadzeniu 7
const int STEPPERS_ENABLE_PIN = 8;  //aktywacja sterowników na wyprowa-
dzeniu 8
const int pinRST = 9;         //reset
const int pinSDA = 10;        //linia danych
int miejsce = 0;        //dane o miejscu na piętrze
float obr = 0;                //dane o wartości obrotu
int pietro = 0;               //dane o piętrze
int flaga = 0;                  //zmienna pomocnicza
uint32_t nr_id_taga;            //numer identyfikatora RFID
unsigned long zajetosc[4][6];    //numery identyfikatorów według przypi-
sanego miejsca
MFRC522 mfrc522(pinSDA, pinRST);    //obsługa modułu RFID

//globalne obiekty dla silników
```

```
SpeedyStepper stepperX;
SpeedyStepper stepperY;
SpeedyStepper stepperZ;
```

Listing 1. Deklaracje zmiennych i wyprowadzeń

3.2. Funkcje

Pierwsza funkcja podczas wywołania uruchamia proces odkładania przedmiotu przez przenośnik. Najpierw następuje aktywacja sterownika, po czym wysyłane są komendy mówiące, na jaką dokładnie długość mają poruszyć się poszczególne elementy suwnicy i windy.

```
void postaw()
{
  //wysyłanie komunikatu przez port szeregowy
  Serial.print("Odkładanie przedmiotu");

  //aktywacja sterowników silników
  digitalWrite(STEPPERS_ENABLE_PIN, LOW);

  //komendy odpowiedzialne za ruch silników
  stepperX.moveRelativeInMillimeters(10.0);
  stepperY.moveRelativeInMillimeters(-64.0);
  stepperX.moveRelativeInMillimeters(-20.0);
  stepperY.moveRelativeInMillimeters(64.0);
  stepperX.moveRelativeInMillimeters(10.0);
  //dezaktywacja sterowników silników
  digitalWrite(STEPPERS_ENABLE_PIN, HIGH);
}
```

Listing 2. Kod funkcji odpowiedzialnej za odkładanie przedmiotu

Następna funkcja jest bardzo podobna do poprzedniej – służy do odbierania przedmiotu z magazynu.

```
void zabierz()
{
  Serial.print("Odbieranie przedmiotu");
  digitalWrite(STEPPERS_ENABLE_PIN, LOW);
  stepperX.moveRelativeInMillimeters(-10.0);
  stepperY.moveRelativeInMillimeters(-64.0);
```

```
    stepperX.moveRelativeInMillimeters(20.0);
    stepperY.moveRelativeInMillimeters(64.0);
    stepperX.moveRelativeInMillimeters(-10.0);
    digitalWrite(STEPPERS_ENABLE_PIN, HIGH);
  }
```

Listing 3. Kod funkcji odpowiedzialnej za zabieranie przedmiotu

Kolejna funkcja ustawia obrotnicę skierowaną na konkretne miejsce na piętrze. Dodatkowo aby uniknąć zaplątania przewodów, za pomocą prostych działań funkcja sprawdza, w którą stronę należy się obrócić, aby szybciej osiągnąć cel.

```
  int obrot(int miejsce)
  {
    Serial.print("Obracanie platformy na miejsce ");
    Serial.println(miejsce);
    digitalWrite(STEPPERS_ENABLE_PIN, LOW);
    if (miejsce >= 2) //jeżeli miejsce wypadło na drugiej połówce
    {
      miejsce = miejsce - 6;  //zmiana znaku
      obr = 0.1667 * miejsce; // 1-cały obrót, 6 miejsc - 1/6=0.1667
    }
    else
    {
      obr = 0.1667 * miejsce;
    }
    //Komenda odpowiedzialna za obrót platformy według wyliczonego pa-
rametru
    stepperZ.moveToPositionInRevolutions(obr);
    digitalWrite(STEPPERS_ENABLE_PIN, HIGH);
  }
```

Listing 4. Kod funkcji odpowiedzialnej za obrót przenośnika

Ostatnia funkcja odpowiedzialna jest za ruch windy na określony poziom. Aby otrzymać, o ile milimetrów należy się przemieścić, numer piętra mnożony jest przez wcześniej zdefiniowaną, stałą odległość pomiędzy piętrami.

```
int poziom(int pietro)
{
  Serial.print(" Ruch na piętro ");
  Serial.println(pietro);
  digitalWrite(STEPPERS_ENABLE_PIN, LOW);
  pietro = pietro * (-90);   //odległość pomiędzy piętrami to 90 milimetrów
  stepperX.moveToPositionInMillimeters(pietro);
  digitalWrite(STEPPERS_ENABLE_PIN, HIGH);
}
```

Listing 5. Kod funkcji odpowiedzialnej za ruch windy

Do ustawienia wszystkich parametrów pracy układu, takich jak przypisane wyprowadzeń mikrokontrolera czy zainicjowanie początkowych wartości zmiennych, została użyta funkcja *setup()*. Określane są w niej również parametry definiujące pracę silników, takie jak maksymalne prędkości czy przyspieszenia.

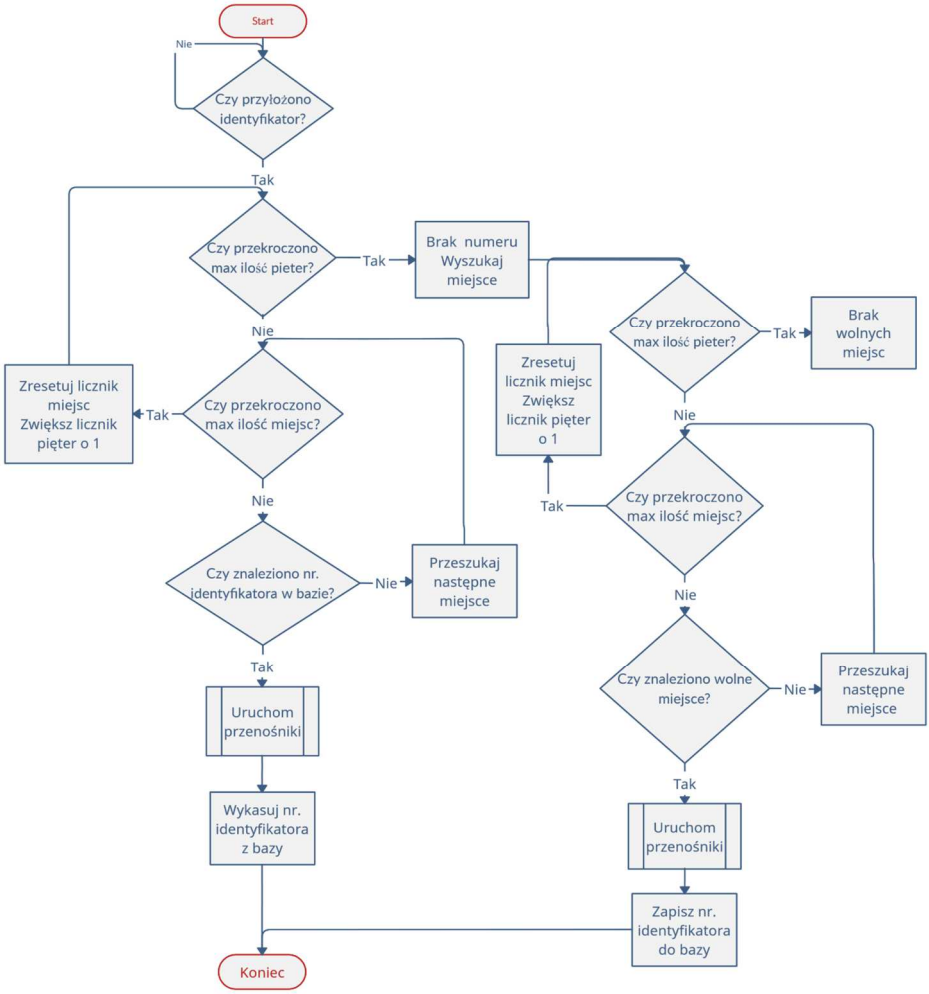
```
   void setup() {
     //inicjalizacja SPI dla czytnika RFID
     SPI.begin();
     Serial.begin(9600);

     mfrc522.PCD_Init();     //inicjalizacja identyfikatora RFID

     pinMode(STEPPERS_ENABLE_PIN, OUTPUT);  //wyprowadzenie odpowiedzialne
za aktywowanie   sterowników

     //przypisanie obiektom silników wyprowadzeń sterujących
     stepperX.connectToPins(MOTOR_X_STEP_PIN, MOTOR_X_DIR_PIN);
     stepperY.connectToPins(MOTOR_Y_STEP_PIN, MOTOR_Y_DIR_PIN);
     stepperZ.connectToPins(MOTOR_Z_STEP_PIN, MOTOR_Z_DIR_PIN);

     digitalWrite(STEPPERS_ENABLE_PIN, HIGH);

     //ilość kroków na 1 mm
     stepperX.setStepsPerMillimeter(50 * 1);
     stepperY.setStepsPerMillimeter(50 * 1);
```

```
    // ilość kroków na 1 mm na pełny obrót
    stepperZ.setStepsPerRevolution(2048);

    //maksymalne prędkości
    stepperX.setSpeedInMillimetersPerSecond(30.0);
    stepperY.setSpeedInMillimetersPerSecond(10.0);
    stepperZ.setSpeedInRevolutionsPerSecond(0.5);

    //maksymalne przyspieszenia
    stepperX.setAccelerationInMillimetersPerSecondPerSecond(50.0);
    stepperY.setAccelerationInMillimetersPerSecondPerSecond(100.0);
    stepperZ.setAccelerationInRevolutionsPerSecondPerSecond(1);
    nr_id_taga = 0;
  }
```

Listing 6. Kod funkcji *setup()*

3.3. Algorytm

Algorytm rozpoczyna swoje działanie od ciągłego sprawdzania, czy do czytnika RFID został przyłożony identyfikator. Po jego wykryciu i odczytaniu przypisanego numeru, następuje przeszukiwanie bazy danych w celu sprawdzenia, czy taki numer jest już zapisany. Po kolei sprawdzane są wszystkie miejsca na kolejnych piętrach. Jeśli zostanie odnaleziony, rozpoczyna się procedura pobierania i zwracania przedmiotu z magazynu. Jeżeli numer nie znajduje się w bazie, algorytm rozpoczyna wyszukiwanie najbliższego wolnego miejsca, po czym wywoływane są funkcje transportujące nowy przedmiot na określone stanowisko magazynu.



Rysunek 4. Schemat blokowy algorytmu
Źródło: opracowanie własne.

3.4. Pętla główna programu

Pętlę główną programu rozpoczyna instrukcja warunkowa, która rozpoczyna pracę algorytmu odczytaniem numeru identyfikatora RFID. Za pomocą sumy logicznej i przesunięć bitowych zostaje on zapisany do jednej zmiennej. Następnie za pomocą dwóch instrukcji *for* rozpoczyna się przeszukiwanie bazy danych, podczas którego przez port szeregowy wysyłane są komunikaty informujące o procesie. Istotne jest, aby przeszukiwanie poziomów rozpocząć od wartości „1”, ponie-

waż wartość „0” oznacza miejsce pozostawienia/odbioru ładunku. Po odnalezieniu numeru i przypisanej mu lokalizacji, następuje wywołanie funkcji z parametrami określającymi jego położenie, odpowiedzialnych za transport przedmiotu na miejsce odbioru, po czym numer jest kasowany. W przypadku gdy baza danych nie rozpoznała numeru identyfikatora, program wyszukuje najbliższe wolne miejsce magazynowania, po czym uruchamia procedurę przenoszenia ładunku, po czym powraca na miejsce oczekiwania.

```
   void loop()
   {
     //oczekiwanie na identyfikator
     if   (mfrc522.PICC_IsNewCardPresent()   &&   mfrc522.PICC_ReadCardSe-
rial())
     {
       Serial.println("Wykryto identyfikator");

       //scalanie numeru identyfikatora RFID do jednej zmiennej
       nr_id_taga     =     (mfrc522.uid.uidByte[0]     <<     24)     |
(mfrc522.uid.uidByte[1]  <<  16)  |  (mfrc522.uid.uidByte[2]  <<  8)  |
(mfrc522.uid.uidByte[3]);

       for (pietro = 1; pietro < 4; pietro++) //przeszukiwanie pięter
       {
         Serial.print("ZMIANA PIETRA NA ");
         Serial.println(pietro);
         for (miejsce = 0; miejsce < 6; miejsce++)  //przeszukiwanie miejsc
         {
           Serial.print("ZMIANA MIEJSCA NA ");
           Serial.println(miejsce);

           //sprawdzanie czy odczytany numer jest taki jak ten zapisany w
tablicy
           if (nr_id_taga == zajetosc[pietro][miejsce])
           {
             Serial.print("Odnaleziono przedmiot");
             //wywołanie funkcji transportu
             poziom(pietro);
             obrot(miejsce);
             zabierz();
             Serial.print(" zabieranie ");
             Serial.println(zajetosc[pietro][miejsce]);
```

```
              Serial.print(" z piętra ");
              Serial.println(pietro);
              Serial.print(" i miejsca ");
              Serial.println(miejsce);
              zajetosc[pietro][miejsce] = 0;  //usuwanie nr identyfikatora
z bazy danych

              //powrót na miejsce oczekiwania
              miejsce = 0;
              pietro = 0;
              obrot(miejsce);
              poziom(pietro);
              flaga = 1;
              break;
            }
          }
          if (flaga) break;
        }
        if (flaga)
        {
          flaga = 0;
        }
        else
        {
          Serial.print("Nie odnaleziono przedmiotu ");
          Serial.print("Wyszukiwanie wolnego miejsca");
          for (pietro = 1; pietro < 4; pietro++) //przeszukiwanie pięter
          {
            Serial.println("Wyszukiwanie wolnego pietra");
            for (miejsce = 0; miejsce < 6; miejsce++) //przeszukiwanie
miejsc
            {
              Serial.print("Wyszukiwanie wolnego miejsca na pietrze ");
              Serial.println(pietro);
              if (zajetosc[pietro][miejsce] == 0) //odnalezienie pustego
miejsca
              {
                //przypisanie identyfikatora do pustego miejsca
                zajetosc[pietro][miejsce] = nr_id_taga;
                poziom(pietro);
                obrot(miejsce);
                postaw();
```

```
              Serial.println(zajetosc[pietro][miejsce]);
              Serial.print(" znajduje się na pietrze");
              Serial.println(pietro);
              Serial.print(" i miejscu ");
              Serial.println(miejsce);
              miejsce = 0;
              pietro = 0;
              obrot(miejsce);
              poziom(pietro);
              flaga = 1;

              break;
            }
          }
          if (flaga)
          {
            flaga = 0;
            break;
          }
        }
      }
      nr_id_taga = 0;
    }
  }
```

Listing 7. Główna pętla programu

4. PODSUMOWANIE

Zbudowany model makiety został uruchomiony i przetestowany. Algorytm działał poprawnie, operacje zabierania i odbierania przedmiotów były wykonywane prawidłowo.

Podczas projektowania części pod druk 3D należy zwrócić szczególną uwagę na dobranie odpowiednich tolerancji. Kluczem do sukcesu działania części mechanicznych tworzonych tą metodą jest dobranie odpowiednich luzów pomiędzy ruchomymi elementami konstrukcji. Należy wziąć też pod uwagę zużywanie się tych elementów podczas testów w nieodpowiednich warunkach.

Przedstawione rozwiązanie może znaleźć zastosowanie w miejscach, gdzie zbudowanie magazynu o dużej powierzchni nie byłoby możliwe ze względu na

ograniczenia przestrzenne czy finansowe lub w systemach, gdzie potrzeba małej instalacji do transportu materiałów/przedmiotów bez pomocy człowieka.

LITERATURA

1. Gotfryd M., Pawłowicz B., Pitera G., *Aktywne, pasywne i półpasywne systemy RFID*, Zeszyty Naukowe Politechniki Rzeszowskiej Elektrotechnika, 2015(2/2015), 243–258.

ŹRÓDŁA INTERNETOWE

2. https://github.com/miguelbalboa/rfid (dostęp: 08.04.2021).
3. https://github.com/Stan-Reifel/SpeedyStepper (dostęp: 20.05.2021).
4. https://pl.wikipedia.org/wiki/RFID (dostęp: 18.05.2021).
5. https://www.arduino.cc/en/software (dostęp: 10.04.2021).

Krzysztof RAK, Michał CIESZYŃSKI
Konrad GUZY, Marcin GORAL

dr inż. Tomasz ŻABIŃSKI
opiekun naukowy

SPOSOBY POMIARÓW PRĘDKOŚCI OBROTOWEJ

Niniejszy artykuł opisuje sposoby i cel pomiarów prędkości obrotowej oraz porusza temat metod pomiaru prędkości obrotowej i urządzeń umożliwiających ich przeprowadzenie. Omówione zostaną zasady działania urządzeń pomiarowych oraz warunki, które są niekorzystne dla poszczególnych metod pomiarowych z wykorzystaniem rozpatrywanych technologii. W artykule przedstawione zostaną wady oraz zalety poszczególnych rozwiązań, jak i warunki ich zastosowań przy wykorzystaniu zjawisk fizycznych takich jak zjawisko indukcji elektromagnetycznej. Artykuł zawiera również przykłady zastosowania jednego z czujników służących do pomiaru prędkości obrotowej występujących w samochodach osobowych. Rozpatrzone zostanie jego działanie oraz wpływ na monitorowanie poprawności pracy pojazdu na bazie prędkości obrotowej.
Słowa kluczowe: prędkość obrotowa, metody pomiarowe, urządzenia pomiarowe, czujniki, VSS.

WPROWADZENIE

Jedną z podstawowych wielkości fizycznych jest prędkość obrotowa opisująca ruch obiektów wirujących, która definiuje liczbę obrotów wykonanych w jednostce czasu. Prędkość obrotowa przeważnie wyrażana jest w obrotach na minutę (obroty/minutę) (ang. *rotation per minute*) lub hercach (Hz). Oznacza się ją za pomocą liter *n* lub *f*. Prosta zależność łącząca obie jednostki daje możliwość bardzo szybko przeliczać wartości prędkości. Prędkość obrotowa ściśle powiązana jest z prędkością kątową (radian/sekundę) określającą zmianę kąta obrotu w czasie:

$$\omega = \frac{\Delta\theta}{\Delta t}, \frac{rad}{s}.$$

Między prędkością kątową ω, a częstotliwością obrotów f (wyrażaną w Hz) zachodzi zależność. Jest ona wiążąca również dla prędkość wyrażanej w obrotach/minutę w następujący sposób:

$$\omega = 2 \cdot \pi \cdot f = \frac{2 \cdot \pi \cdot n}{60}, \frac{rad}{s}$$

$$n = \frac{\omega \cdot 60}{2 \cdot \pi}, \frac{obr}{min}.$$

Zazwyczaj konieczne jest wyznaczenie prędkości liniowej (obwodowej) elementu obrotowego o zadanej średnicy (gdzie *d* to średnica obrotowego elementu podana w metrach). W takim wypadku można posłużyć się poniższymi zależnościami:

$$v = \frac{\pi \cdot d \cdot n}{60}, \frac{m}{s}$$

$$n = \frac{60 \cdot v}{\pi \cdot d}, \frac{obr}{min}$$

Realizacja pomiaru może być realizowana dzięki układom stacjonarnym (np. instalowanym na stanowisku) bądź przenośnym urządzeniom. Rozwiązania te niezależnie od sposobu realizacji pomiaru zawierają elementy takie jak:

- przetwornik pomiarowy (konwersja sygnału wejściowego),
- układ akwizycji i przetwarzania (przetworzenie i przygotowanie danych do dalszej obróbki),
- układ wyjściowy (zaprezentowanie danych).

Badany parametr drogi kątowej elementu przetwarzany jest przez przetwornik pomiarowy na sygnał cyfrowy bądź analogowy. Kolejnym krokiem jest poddanie sygnału akwizycji (kontroli, scharakteryzowaniu, monitorowaniu) i przetworzeniu, skutkiem czego może być określenie prędkości obrotowej. Wielkością wyznaczaną jest średnia prędkość obrotowa, chwilowa prędkość obrotowa albo droga kątowa. Układ wyjściowy ma za zadanie zobrazowanie wyników, np. na wyświetlaczu, utworzenie sygnału (analogowego bądź cyfrowego) niosącego informację o prędkości obrotowej i przekazanie go układom sterowania i akwizycji, takim jak na przykład PLC, SCADA, DCS itp. Układy wyjściowe mogą również pełnić funkcję kontrolera wyznaczonych wartości granicznych i sterować układami poprzez generowanie sygnałów cyfrowych zatrzymujących realizowany proces. Powyższe składowe procesu pomiarowego są w pełni integrowane przy użyciu odpowiednich rozwiązań technicznych. Tachometry, monitory prędkości czy też liczniki impulsów są jednymi z przykładów takich rozwiązań. Na rysunku 1. zaprezentowane są przykłady omawianych mierników.

Rys. 1. Przykłady przenośnego i stacjonarnego miernika prędkości obrotowej
Źródło: opracowanie własne.

1. METODY POMIARU PRĘDKOŚCI OBROTOWEJ

Pomiary prędkości obrotowej klasyfikuje się, uwzględniając metodę:

- metoda stykowa – stosowane poprzez sprzężenie przetwornika bezpośrednio z badanym obiektem. W tych przyrządach wykorzystywane jest przemieszczenie liniowe (konwertowane z prędkości obrotowej). Proporcjonalnie do mierzonej wielkości urządzenie generuje napięcie. Największą zaletą takiego rozwiązania jest ciągłość pomiaru. Występują również wady takie jak mały zakres pomiarowy czy niezdolność do poprawnego pomiaru obiektów o małej mocy (z powodu dużego poboru mocy przez urządzenie, co wyklucza możliwość wykorzystania takiego miernika),
- metody bezstykowe – bazują na zjawiskach optycznych, magnetycznych bądź przetwornikach porównawczych. Częściowo wyzbyte są wad metod stykowych, jednak znaczniki przymocowane do wirnika, będące ingerencją w konstrukcję elementu obracającego się, mogą być powodem błędów pomiarowych.

Lepsze właściwości metrologiczne wykazują urządzenia niepobierające mocy obiektu poddanego badaniu. Pozwala to osiągnąć większą dokładność mierzonej wartości oraz odznaczają się szerokością pasma (aż do 300 tys. obr./min).

2. PRĄDNICE TACHOMETRYCZNE

Prądnica tachometryczna jest klasyfikowana jako przetwornik bezpośrednio przetwarzający ruch wirnika na sygnał elektryczny proporcjonalny do prędkości obrotów. Pomiar prędkości obrotowej przy użyciu prądnicy tachometrycznej wymaga mechanicznego sprzężenia badanego obiektu z osią prądnicy. Analogowy

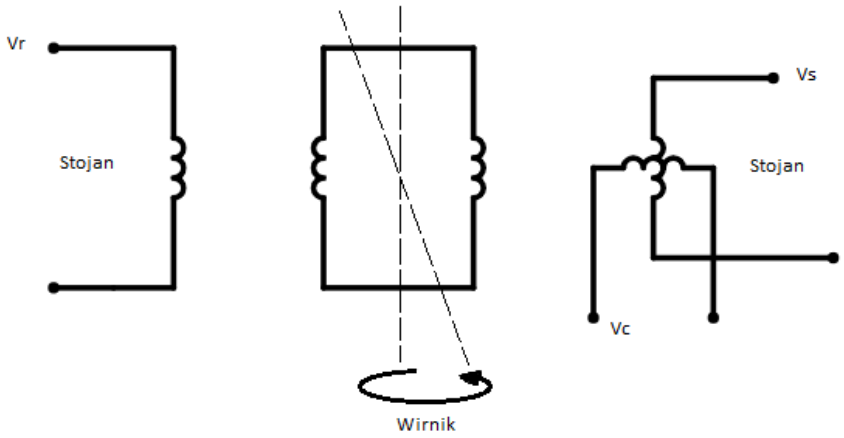
sygnał generowany przez prądnicę cechuje się amplitudą wprost proporcjonalną do prędkości obrotowej. Jej wartością jest iloczyn strumienia magnetycznego, prędkości obrotowej i wartości napięcia określonej stałą prądnicy. Sygnał wyjściowy prądnicy zależy także od temperatury i obciążenia generowanego przez uzwojenie wyjściowe. Aby zminimalizować skutki działania tych czynników, wykorzystuje się układ o dużej impedancji na wejściu. Charakterystyczne dla prądnic tachometrycznych prądu stałego jest uzyskanie dużego zakresu pasma pomiarowego oraz możliwość obrotu w przeciwnych kierunkach. Prądnice prądu przemiennego poprzez zastosowanie prostownika nie dopuszczają do rozróżnienia kierunków obrotów.

Tabela 1. Wady i zalety prądnic tachometrycznych wykorzystanych do pomiaru prędkości obrotowej

Zalety	Wady
• minimalny czas pomiaru chwilowej prędkości obrotowej • znikome błędy pomiarowe • szybka reakcja w przypadku zmiany prędkości obrotowej • łatwy odczyt sygnału • małe wymiary • mały ciężar	• zużywanie się podzespołów w trakcie pracy urządzenia • dokładne sprzężenie wirnika z urządzeniem pomiarowym ciężkie do uzyskania • wymaga konserwacji i regularnych przeglądów

3. RESOLWERY

Resolwery klasyfikowane są bliźniaczo do prądnic tachometrycznych. Nazywane potocznie transformatorem położenia kątowego, są jednymi z najczęściej stosowanych czujników pomiaru kąta i przemieszczenia. Resolwer jest małym dwufazowym urządzeniem z uzwojeniem na wirniku. Uzwojenie wirnika zasilane jest prądem przemiennym wysokiej częstotliwości, co skutkuje indukcją pola magnetycznego na uzwojeniach stojana (zorientowanych względem siebie pod kątem 90°). Jedną z głównych zalet tych przetworników jest dokładność informacji o położeniu wału, co pozwala precyzyjnie podawać prędkości obrotowe urządzenia. Cechuje go również niezawodność, odporność na zakłócenia i wymagające warunki pracy. Dzięki temu resolwery często wykorzystuje się w serwonapędach, maszynach militariów, lotnictwie i obrabiarkach numerycznych.



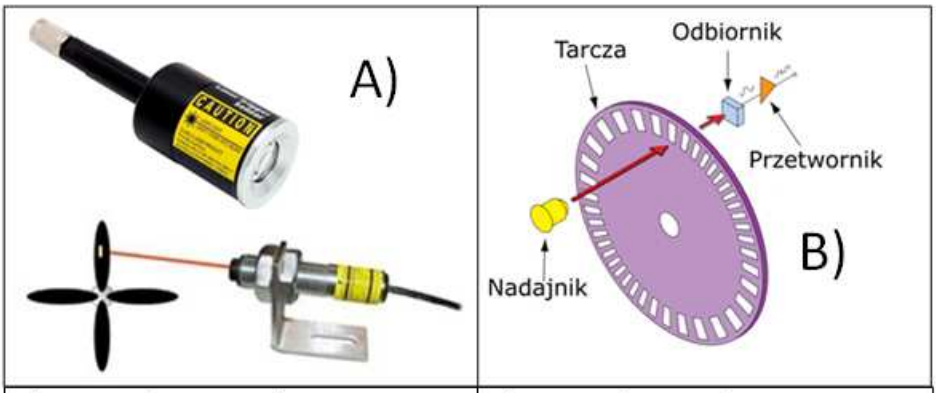
Rys. 2. Przykładowy schemat układu uzwojeń resolvera
Źródło: opracowanie własne.

4. PRZETWORNIKI OPTYCZNE

Wiele układów pomiaru prędkości obrotowej w swoim działaniu bazuje na optycznych przetwornikach sygnału. Wykorzystują one zjawisko wysyłania wiązki światła. Nadajnik emituje wiązkę, która trafia do odbiornika. Czujniki optyczne można podzielić na dwie grupy:

- optyczne odbiciowe, gdzie nadajnik emituje wiązkę promieni świetlnych, które na swojej drodze spotykają przeszkodę, tam zostają odbite przez wirnik; część odbitych promieni trafia bezpośrednio do odbiornika, gdzie po wzmocnieniu służą do wytworzenia sygnału przełączającego stan czujki;
- układ nadajnik–odbiornik (tzw. czujnik barierowy), wykorzystuje wiązkę światła z emitera wysyłaną bezpośrednio do odbiornika; Za generowanie impulsów na wyjściu czujnika odpowiedzialne są obiekty, pojawiające się chwilowo, na drodze światła przecinające odbierany sygnał.

Najczęściej jako nadajnik stosuje się diody LED lub diody laserowe. Odbiornikami sygnału są wzmacniane układy fotoczułe, fotodiody i fototranzystory. Czujniki optyczne świetnie sprawdzają się w przypadku większych odległości. Posiadają krótki czas reakcji oraz niewielkie rozmiary, co pozwala na zastosowanie ich w ciasnych przestrzeniach . Są odporne na zakłócenia elektromagnetyczne. Ich największa wada ogranicza użyteczność takich rozwiązań w przemyśle z powodu podatności na zanieczyszczenia powstałe w warunkach pracy.



Rys. 3. Wizualizacja zasady działania czujnika refleksyjnego (laserowego) i czujnika nadajnik–odbiornik używanych między innymi w en koderach optycznych
Źródło: opracowanie własne.

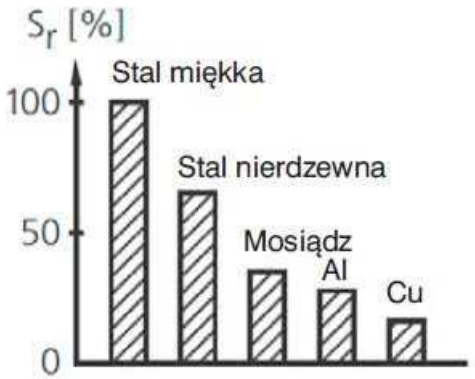
5. PRZETWORNIKI OBROTOWO-IMPULSOWE

Przetworniki obrotowo-impulsowe nazywane są również inkrementalnymi. Ich zadaniem jest pomiar przemieszczeń kątowych, dzięki czemu służą do pomiaru prędkości obrotowej i przebytej drogi kątowej. Liczba impulsów wytwarzanych przez przetwornik obrotowo-impulsowy odpowiada proporcjonalnie kątowi obrotu. Dwa wyjścia sygnałów (A i B), przesunięte względem siebie, pozwalają określić kierunek obrotu. Zazwyczaj jest stosowany dodatkowy sygnał wyjściowy, który umożliwia określenie położenia przetwornika podczas jednego obrotu. Istnieją przetworniki impulsowo-obrotowe o zakresie od 100 do 10 000 impulsów/obrót. Największe znaczenie przy doborze przetwornika ma maksymalna częstotliwość impulsów. Kluczowa jest technologia transmisji sygnału. Nie możemy wykorzystać takiego przetwornika, jeżeli częstotliwość będzie przekraczała 250 kHz.

6. PRZETWORNIKI INDUKCYJNE

W przemyśle technicznym większość wirników jest wykonana z metali. Zasada działania czujnika indukcyjnego bazuje na zmianie pola elektromagnetycznego spowodowanego wirującym metalowym elementem podróżującym przed czołem czujnika. Powstają wtedy prądy wirowe, które inicjują zmianę sygnału wejściowego. Występują wersje z wyjściem napięciowym i prądowym. Jakość pomiaru uzależniona jest od rodzaju materiału wykorzystanego przy wytworzeniu

czujnika. Bazowym metalem dla czujki jest stal. Jeżeli pojawi się inny rodzaj materiału, konieczne jest wprowadzenie korekty. Wzrost strefy działania czujnika powoduje spadek górnej wartości częstotliwości przełączania. Trudności te rekompensuje cena, jak i dostępność takich czujników. Stosowane są najczęściej w motoryzacji przy pomiarach prędkości obrotowej.



Rys. 4. Wykres współczynników korekcji w zależności od rodzaju wykrywanego metalu
Źródło: https://automatykab2b.pl/prezentacje/41183-podstawowe-parametry-czujnikow--indukcyjnych-ifm-electronic.

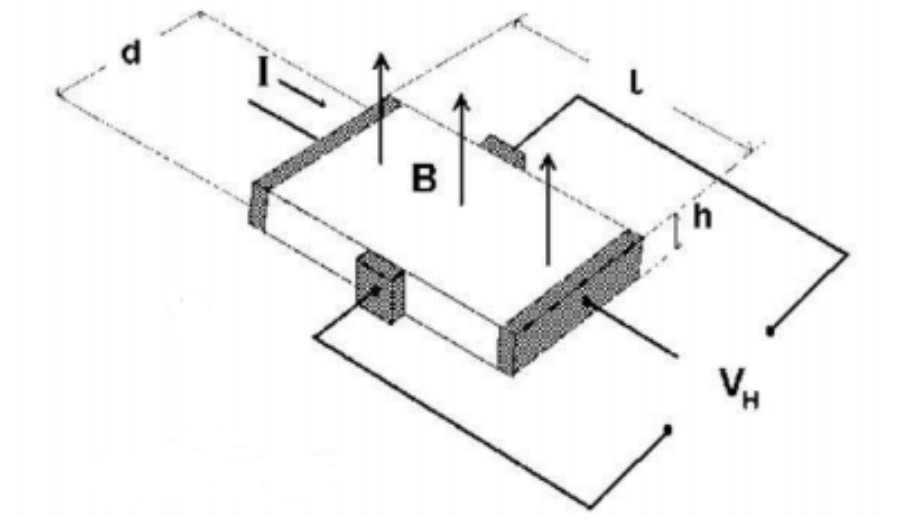
7. CZUJNIKI POJEMNOŚCIOWE

Czujniki pojemnościowe działają bliźniaczo do czujników magnetycznych. Główną różnicą jest brak konieczności zastosowania metalowych obiektów (wirników) poruszających się przed czujnikiem. Wykorzystany kondensator zmienia swoją pojemność w momencie, gdy zbliży się do niego jakikolwiek materiał. Zmiana pojemności wywołana jest różnicą stałej dielektryczną powietrza od stałej dielektrycznej materiału przed czołem czujnika. Wzrost pojemności powoduje przyrost amplitudy drgań, co wykrywane jest przez odpowiedni układ przetwarzający. Najpierw trafia do detektora, gdzie jest rozpoznawany. W dalszej części procesu przechodzi poprzez przerzutnik na wzmacniacz. Ostateczny sygnał na wyjściu pozwala nam określać zmiany położenia badanego obiektu.

8. CZUJNIKI HALLOTRONOWE

Czujniki hallotronowe działają, wykorzystując efekt Halla. Poprzez przepływ prądu przez półprzewodnik (np. arsenek indu, german), znajdujący się w polu magnetycznym, wytwarza się napięcie elektryczne (zwane inaczej napię-

ciem Halla). Indukcja pola magnetycznego oraz natężenia płynącego prądu jest proporcjonalna do wartości tego napięcia. Element Halla jest przymocowany na metalowej płycie. Magnes wyposażony jest w obwód magnetyczny wykonany z ferromagnetyku. Pole magnetyczne i napięcie czujnika wytwarzają pomiarową różnicę potencjału. Zmieniając reaktancję szczeliny powietrznej między czujnikiem Halla a magnesem, powoduje zamknięcie linii pola w otoczeniu magnetowodu, co wprowadza sygnał w stan zera. Jeżeli wirujący obiekt posiadałby piłokształtną geometrię, uzyskalibyśmy sygnał skokowy.



Rys. 5. Schemat obrazujący działanie efektu Halla
Źródło: https://pracowniefizyczne.up.krakow.pl/wp-content/uploads/sites/3/2018/03/CS4_Hall.pdf.

9. POMIARY STROBOSKOPOWE

Jedne z bezkontaktowych metod pomiarowych stosowana przy nieruchomych obiektach z wirnikami to pomiary stroboskopowe, które polegają na zsynchronizowaniu częstotliwości wyładowań szybkiej lampy błyskowej z prędkością obrotową badanego obiektu. Przyrząd umożliwiający taki pomiar jest zazwyczaj urządzeniem przenośnym, którego najważniejszym elementem jest wcześniej wspomniana lampa stroboskopowa. W nowoczesnych rozwiązaniach opiera się ona na technologii diod LED (ang. *Light-Emitting Diode*). Synchronizacja częstotliwości błysków uzyskiwana jest poprzez przestrajanie impulsów wytwarzanych generatorem. Stroboskop oświetla wirujący przedmiot.

Celem jest uzyskanie efektu stroboskopowego, czyli wrażenia, że poruszające się ciało pozornie zwalnia, a nawet zatrzymuje swój ruch. Ponadto uzyskanie efektu „pozornego zatrzymania" umożliwia diagnozę ewentualnych usterek czy zjawisk pasożytniczych (np. niepożądanych oscylacji). W zależności od wykonania pomiaru oraz klasy dokładności urządzenia błąd pomiarowy wynosi około 3%. Niewątpliwą zaletą tej metody jest możliwość pomiaru nawet najmniejszych obiektów poruszających się bardzo szybko, kiedy umieszczone są w trudno dostępnych obszarach. Minusem tej techniki pomiaru jest ograniczenie możliwości pomiarów jedynie do sytuacji, gdy wartość mierzonej prędkości obrotowej jest stała.

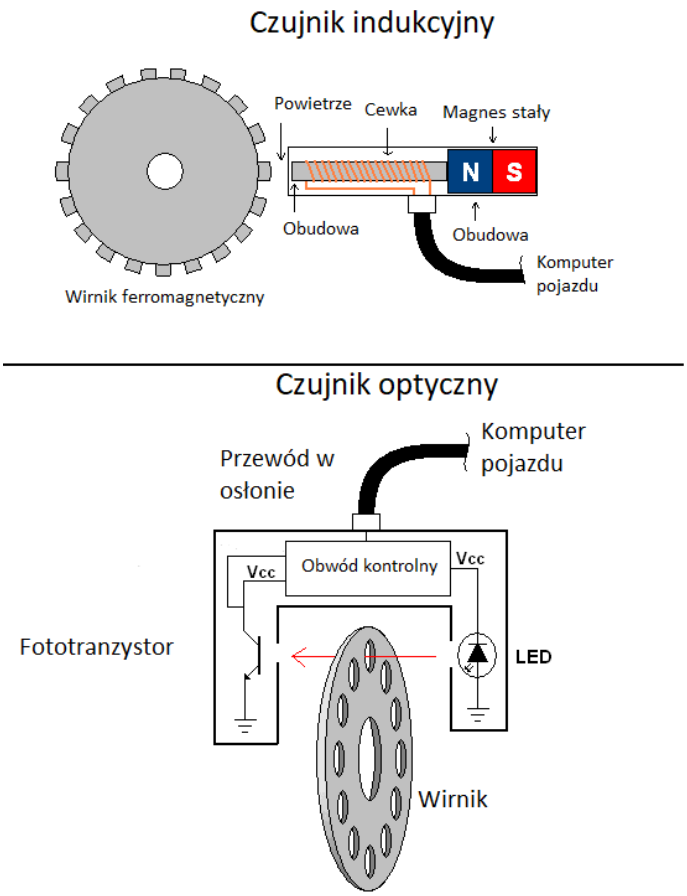
10. ZASTOSOWANIE CZUJNIKA PRĘDKOŚCI OBROTOWEJ W POJAZDACH

VSS (ang. *Vehicle Speed Sensor*) to czujnik prędkości obrotowej pobierający informacje o prędkości obrotowej skrzyni biegów przekładanej na prędkość poruszania się pojazdu, wyświetlonej na desce rozdzielczej przed kierującym. Potocznie nazywany czujnik prędkości przytwierdzony jest do obudowy przekładni. Zazwyczaj jest to czujnik indukcyjny bądź optyczny.

Najpopularniejsze czujniki indukcyjne składają się z magnesu umieszczonego na czopie otoczonym cewką. Jest umieszczony w stałej odległości od wirnika ferromagnetycznego. Gdy ząb wirnika zbliża się do czoła czujnika, zmienia się strumień magnetyczny w cewce. Zmiana strumienia powoduje impuls napięciowy. Moduł sterujący pojazdu na podstawie impulsów oblicza prędkość pojazdu.

Czujniki optyczne podobnie generują impulsy z częstotliwością równą obrotom wirnika, lecz wykorzystują do tego odbijaną bądź przepuszczaną, wiązkę światła (w zależności czy typ czujnika jest barierowy, czy odbiciowy). Na obracającej się tarczy mogą znajdować się powierzchnie dobijające światło (wyłapywane za pomocą elementów optoelektronicznych) lub szczeliny przepuszczające je do odbiornika (zazwyczaj fototranzystora). Problemem takiego rozwiązania jest podawanie przez wyświetlacz nieprawidłowej prędkości w przypadku ciągłego uślizgu kół napędzających pojazd.

Wprowadzane są nowe technologie opracowywane i wdrażane na rynek przez firmy takie jak na przykład Beru, Bosch czy Delphi. Umożliwiają one dokładniejsze i bardziej niezawodne określanie prędkości pojazdu bez konieczności obliczania i uśredniania odczytów czujnika prędkości koła. Powstało rozwiązanie mierzące prędkość pojazdu względem ziemi przy użyciu radaru. Radiolokacja pozwala na pomiar, wykorzystując fale milimetrowe. Ten czujnik zapewnia możliwość ciągłego pomiaru nawet w wypadku uślizgu wszystkich opon pojazdu.



Rys. 6. Schemat układu czujnika VSS indukcyjnego
oraz układu czujnika VSS optycznego nadajnik–odbiornik
Źródło: opracowanie własne.

11. PODSUMOWANIE

Przedstawione techniki i metody pomiarowe nie wychodzą poza obszar dobrze znanych rozwiązań powszechnie stosowanych w przemyśle czy innych gałęziach gospodarki. Opracowywane są nowe urządzenia spełniające co raz to bardziej wyśrubowane wymogi. Polem do rozwoju pomiarów prędkości obrotowych jest badanie szybkich kamer cyfrowych i algorytmów przetwarzania obrazów. Również skutecznością pomiaru prędkości obrotowej wykazuje się analiza sygnału drgań wirników. Bada ona podstawowe składowe synchronicznie ściśle po-

wiązane z obrotami wirników. Niekiedy w silnikach i serwonapędach, przetworniki pomiaru prędkości obrotowej są wbudowywane przed opuszczeniem fabryki. Należy pamiętać, że pomiar można wprowadzić w życie za pomocą układu telemetrycznego, który reaguje na rozciąganie, wykorzystując zjawisko siły odśrodkowej. Dlatego, na wstępie przed zastosowaniem jednego z rozwiązań, warto zgłębić swoją wiedzę, sięgając po dokumentację techniczną i przeanalizować ruch badanego obiektu od strony mechaniczno-fizycznej. Pomiar prędkości obrotowej to ważny element pomocniczy przy ocenie stanu pracy maszyny czy jego aktualnej kondycji technicznej. Pomaga on precyzyjnie sterować urządzeniem.

Warto zwrócić uwagę na zastosowania z dziedziny bezpieczeństwa pracy bazujące na pomiarze prędkości obrotowej. Wiele z nich zapobiega niebezpiecznym sytuacjom. Jeżeli prędkość obrotowa jest za duża, można stosować układy automatycznie wyłączające maszynę. Jeden czujnik zezwala nam na wiele różnych opcji, co gwarantuje mu użyteczność w procesach produkcyjnych i nie tylko.

LITERATURA

1. Brock S., Zawirski K., *Cyfrowy pomiar prędkości obrotowej w napędzie elektrycznym*, Pomiary Automatyka Robotyka 2005.
2. Paprzycki I., *Metody pomiarów prędkości obrotowej wykorzystywanych w dynamicznych badaniach podwozi lotniczych*, Logistyka 2015.
3. Pióro B., Pióro M., *Podstawy Elektroniki. Część I*, Wydawnictwo Szkolne i Pedagogiczne Spółka Akcyjna Warszawa 1994.
4. Szczeniowski S., *Fizyka doświadczalna. Część 3*, Elektryczność i magnetyzm, PWN, Warszawa 1980.

ŹRÓDŁA INTERNETOWE

5. https://cecas.clemson.edu/cvel/auto/sensors/vehicle-speed.html (dostęp: 10.05.2021).
6. https://automatykab2b.pl/prezentacje/41183-podstawowe-parametry-czujnikow-indukcyjnych-ifm-electronic (dostęp: 11.05.2021).

Konrad GUZY, Krzysztof RAK
Michał CIESZYŃSKI, Marcin GORAL

dr inż. Tomasz ŻABIŃSKI
opiekun naukowy

PORÓWNANIE MODELOWANIA BRYŁOWEGO I POWIERZCHNIOWEGO W SYSTEMACH CAD

W niniejszym artykule opisano różnice w procesie modelowania bryłowego oraz powierzchniowego w systemach CAD na podstawie detalu dzbanka. Do przedstawienia rozbieżności wykorzystano program Autodesk Inventor Professional 2019 w wersji studenckiej. Dodatkowo podkreślono wady i zalety poszczególnych technik modelowania. Opisano podstawowe operacje charakterystyczne dla obydwu metod. Proces porównania przeprowadzono w poszczególnych krokach. Począwszy od wykazania różnic w tworzeniu szkicu modelu, poprzez operacje wyciągnięć, obrotów, wyciągnięć złożonych oraz przeciągnięć po ścieżce, po wykonanie gotowego modelu wraz z analizą ciągłości powierzchni. Końcowo w artykule wyciągnięto wnioski na podstawie zaobserwowanych różnic, wygody modelowania oraz czasu potrzebnego na wykonanie części.
Słowa kluczowe: modelowanie bryłowe, modelowanie powierzchniowe, systemy CAD.

WPROWADZENIE

Systemy CAD (ang. *Computer Aided Design*) stanowią w dzisiejszych czasach jedną z głównych gałęzi, na których opiera się światowy przemysł. Jest to metoda projektowania wspomaganego komputerowo, w której możliwe jest kreślenie geometrii 2D również służącej lepszej wizualizacji geometrii 3D. Technologia ta znacznie przyspieszyła proces projektowania pojedynczych części oraz całych konstrukcji maszynowych, wraz z ich pełną dokumentacją techniczną. Udogodnienia związane z łatwością edycji projektu zaoszczędziły inżynierom z całego świata setki godzin projektowych poświęcanych kartce papieru. Początkowo sprowadzało się to tylko do modelowania wizualnego oraz tworzenia dokumentacji, jednak z biegiem czasu rozwiązanie to zaczęło przybierać nowe formy.

Nowymi możliwościami programów zaczęły się stawać symulacje dynamiczne, różnego rodzaju generatory modeli, a także integracja z programami służącymi do analiz wytrzymałościowych i termicznych. Podczas procesu planowania części stawiamy pewne wymagania. Zaprojektowany model musi stawić czoło tym wymogom zależnie od planowanych późniejszych działań. Projektując model, który w kolejnym procesie zostanie poddany symulacji obróbki w programie CAM, musimy mieć świadomość możliwości i ograniczeń maszyny frezującej zamodelowany detal. Podobny tok rozumowania musimy przyjąć podczas wykonywania części metodami przyrostowymi, takimi jak: spiekanie proszków, drukowanie metodą FDM czy utwardzaniu płynnej żywicy za pomocą wiązki lasera (SLA). Kolejną odnogą standardu modelowania są różnorodne symulacje komputerowe. Podczas tworzenia modelu musimy zwrócić szczególną uwagę na stan jego powierzchni. Jest to jeden z najistotniejszych punktów w późniejszej symulacji, ponieważ pokrywamy model siatką elementów skończonych, gdzie ważny jest kształt tych elementów oraz ich połączenie. Zły stan powierzchni modelu może doprowadzić do błędów w programie, co w końcowym rozrachunku może dawać nieprawidłowy wynik lub jego całkowity brak, spowodowany błędami obliczeń np. przy przerwaniu siatki elementów skończonych.

Wraz z powstawaniem coraz większej liczby dostępnych opcji tworzenia modeli, możemy obecnie podzielić modelowanie na kilka gałęzi. W tym artykule skupimy się głównie na porównaniu modelowania bryłowego wraz z modelowaniem powierzchniowym. Modele powierzchniowe zbudowane są z powierzchni o nieskończenie małej grubości oraz z krawędzi ograniczających je. Natomiast modele bryłowe zbudowane są również z krawędzi i powierzchni, jednak posiadają pewną grubość, zajmując pewny obszar pola roboczego. Zwrócimy uwagę na wygodę modelowania poszczególnych elementów detalu, a także na ilość potrzebnego czasu na zamodelowanie tego samego detalu. Cały proces projektowania zostanie przeprowadzony w środowisku programu Autodesk Inventor Professional 2019 w wersji studenckiej.

1. PODSTAWY MODELOWANIA W SYSTEMACH CAD

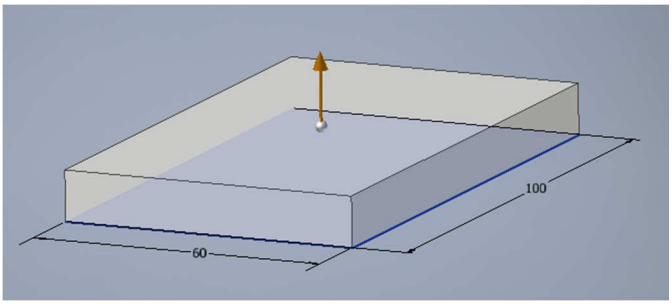
Projektowanie modeli i konstrukcji w systemach komputerowych opiera się głównie na zasadzie tworzenia modeli geometrycznych wykorzystujących szeroką wiedzę z zakresu konstrukcji maszyn, jak i matematyki.

Projektowanie zaczyna się od stworzenia szkicu dwuwymiarowego, a następnie poddanie go operacjom zależnym od sposobu modelowania i chęci osiągnięcia finalnego efektu. Operacje te różnią się zależnie od wykorzystywanego programu do modelowania, jednak wszystkie są bardzo podobne w ich wykorzystywaniu. Jest to pewne ujednolicenie, dzięki czemu użytkownik jest w stanie szybko uczyć się kolejnych programów wykorzystywanych w przemyśle.

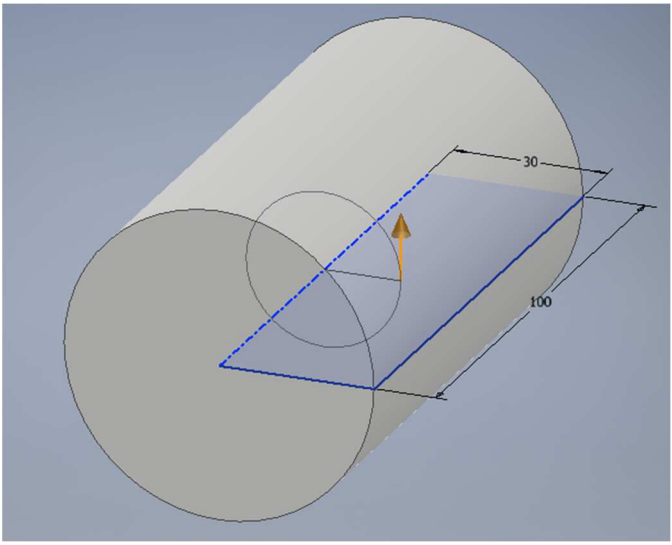
Po stworzeniu nowego pliku części wybieramy płaszczyznę, na której chcemy wykonać pierwszy szkic. Następnie wybieramy elementy, z których będzie się składała bazowa geometria. Mamy do dyspozycji elementy takie jak: linia, krzywa interpolacyjna, okrąg, elipsa, łuk, a także wiele wariantów brył geometrycznych. Wszystkie naniesione wcześniej elementy zostają powiązane za pomocą dostępnych rodzajów wiązań, każdy program cechują inne rodzaje, np.: zgodności, współliniowości, koncentryczności, stałe, równoległe, prostopadłości, poziome, pionowe, styczne, symetryczne i równe. Powstały szkic przekształcamy odpowiednimi operacjami tworzącymi bryły lub powierzchnie. Elementy te powtarzamy z zastosowaniem różnych parametrów i umiejscowień szkiców, aż do stworzenia finalnej wersji modelu.

2. MODELOWANIE BRYŁOWE W SYSTEMACH CAD

W modelowaniu bryłowym główną rolę odgrywają elementy płaskie służące do definiowania poszczególnych operacji takich jak wyciągnięcia i obroty. Głównie opiera się na zamkniętych profilach tworzących obwiednie bryły. Możemy wykorzystywać operacje Boolowskie obejmujące działania takie jak suma, różnica, iloczyn. Bryły podstawowe dzielimy na obrotowe oraz wielościenne. Wspomniana wcześniej operacja wyciągnięcia służy do tworzenia brył wielościennych, natomiast operacja obrót tworzy figury obrotowe posiadające oś symetrii.



Rys. 1. Bryła uzyskana w wyniku operacji wyciągnięcia profilu
Źródło: opracowanie własne.



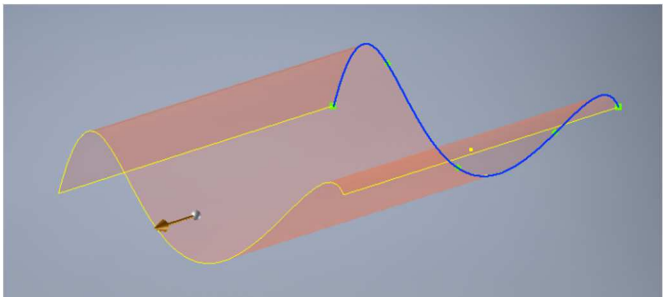
Rys. 2. Bryła uzyskana w wyniku operacji obrotu profilu
Źródło: opracowanie własne.

Występują również bardziej zaawansowane operacje takie jak: przeciągnięcie po ścieżce, wyciągnięcie złożone czy wypukłość. Po utworzeniu głównej bryły możemy użyć edycji cech modelu. Są to głównie zaokrąglenia oraz fazowania związane głównie z operacjami technologicznymi. Zaokrąglenia i fazowania możemy użyć na dowolnej krawędzi z zachowaniem odpowiednich wartości promieni oraz wymiarów fazy. Warto wziąć pod uwagę, że zaokrąglenia często wprowadzają różne niedoskonałości w końcowym modelu, który chcemy poddać analizie wytrzymałościowej. Wyróżniamy również więcej edycji cech, np.: pochylenie powierzchni, skorupa, otwór czy pogrubienie i odsunięcie powierzchni. Pozwalają one na tworzenie znacznie bardziej skomplikowanych geometrycznie elementów bryłowych.

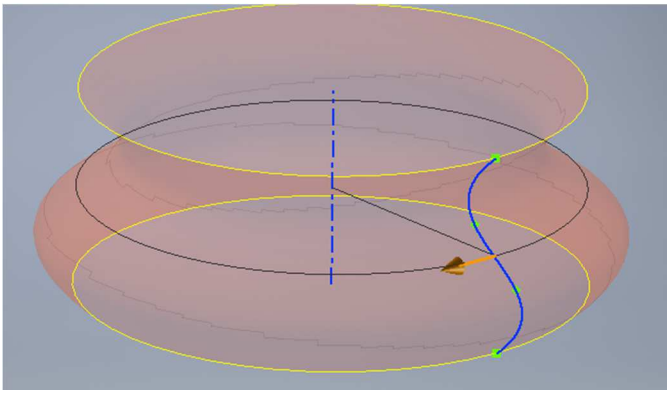
3. MODELOWANIE POWIERZCHNIOWE W SYSTEMACH CAD

W modelowaniu powierzchniowym główną rolę odgrywają elementy profilowe dwuwymiarowe, jednak zasadniczą różnicą, która wpływa na model, są punkty kontrolne definiujące krzywe zarówno płaskie, jak i przestrzenne. Na tych krzywych są rozpisane powierzchnie, które możemy podzielić na dwa warianty. Do pierwszej grupy powierzchni możemy zaliczyć te opisane za pomocą wzorów matematycznych, na przykład obrót krzywych stożkowych. Do drugiej grupy zaliczają się powierzchnie swobodne, składające się z nieskończonej liczby punk-

tów, opisywanych jedynie przez ich współrzędne. Taki model postępowania cechuje się dużą elastycznością. Do tego typu modelowania możemy używać profilów otwartych co znacznie ułatwia proces modelowania.



Rys. 3. Powierzchnia uzyskana w wyniku operacji wyciągnięcia profilu
Źródło: opracowanie własne.



Rys. 4. Powierzchnia uzyskana w wyniku operacji obrotu profilu
Źródło: opracowanie własne.

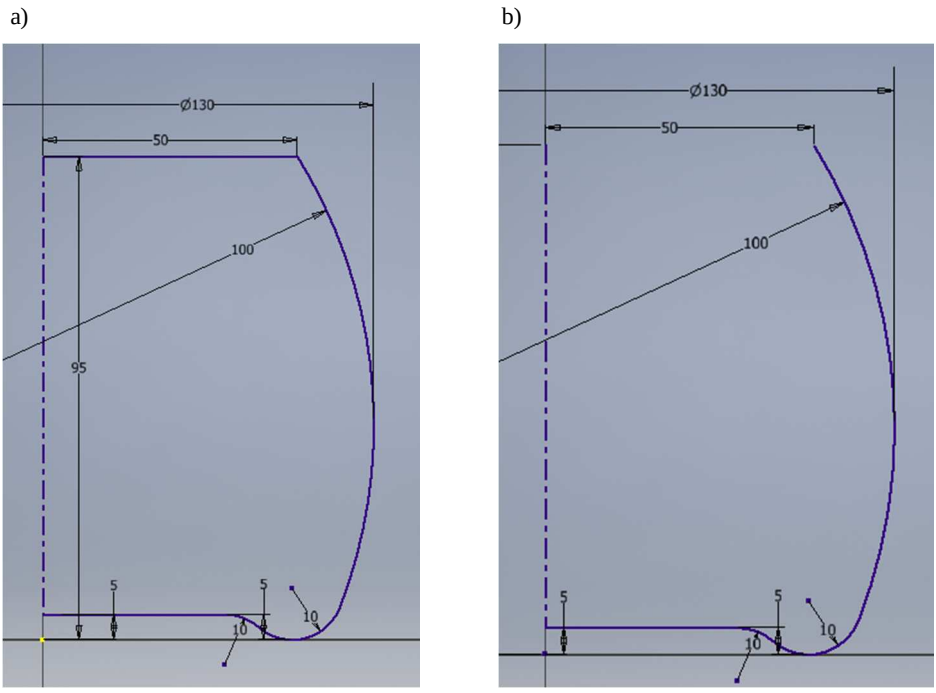
Za pomocą modelowania powierzchniowego możemy wykonywać bardzo złożone kształty wyróżniające się spójnością powierzchni, które są niemożliwe do wykonania tradycyjnymi metodami modelowania bryłowego. Często jednak w takich modelach występują elementy mocowań, co finalnie prowadzi do połączenia tych metod. Głównymi przedmiotami tworzonymi za pomocą modelowania po-

wierzchniowego są obudowy sprzętów codziennego użytku, modele karoserii samochodów czy detale wykonywane za pomocą wtryskarek.

4. PRZYKŁAD PORÓWNANIA MODELOWANIA BRYŁOWEGO I POWIERZCHNIOWEGO

Porównania modelowania dokonamy na podstawie detalu dzbanka możliwego do wykonania obiema metodami. Po każdym kroku przyjrzymy się poszczególnym opcjom i kolejności wykonywania modelu.

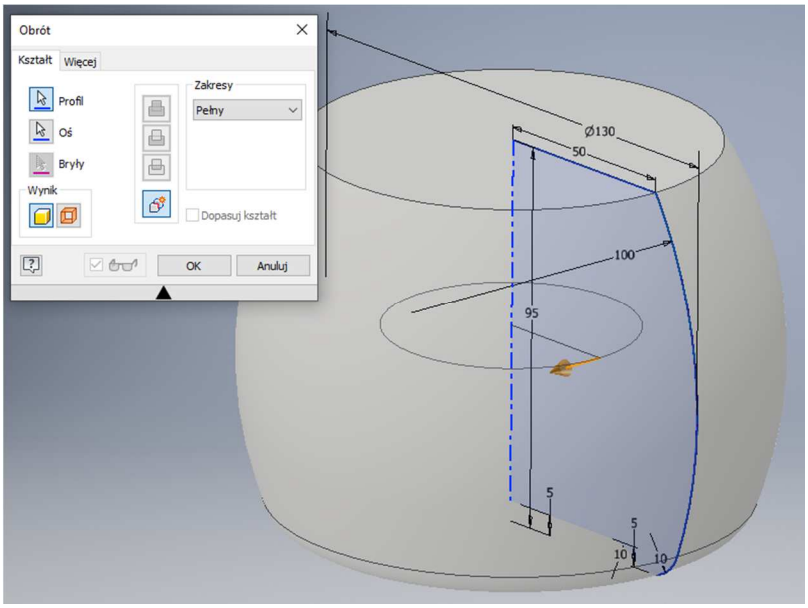
Zaczynamy od stworzenia pliku, a następnie wykonujemy pierwszy szkic. Możemy zauważyć, że geometria wykonana metodą modelowania powierzchniowego nie posiada zamkniętej obwiedni. Jest to spowodowane tym, że w tym przypadku wyciągamy nieskończenie cienki obiekt, przeciwnie do modelowania bryłowego.



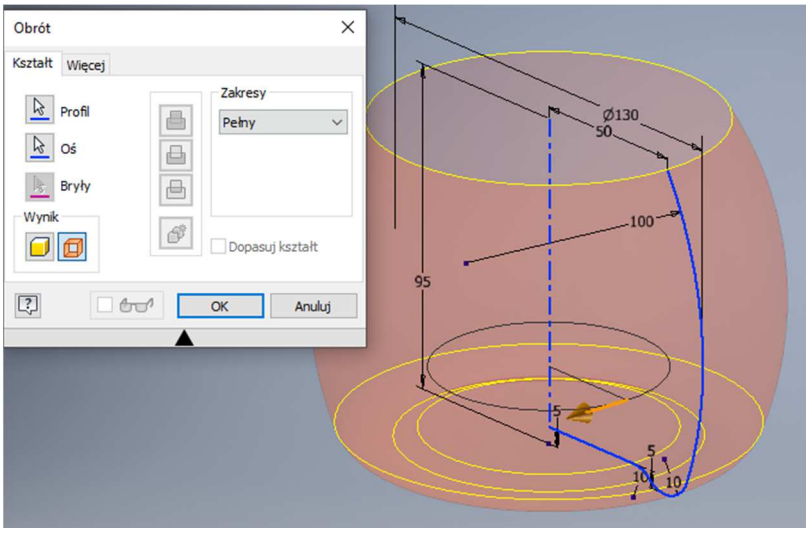
Rys. 5. Szkic modelu: a) bryłowego, b) powierzchniowego
Źródło: opracowanie własne.

Dalej, korzystając z tej samej opcji obrotu wykonanych szkiców, dostajemy poniższe wyniki. Dostrzegamy widoczną różnicę związaną z wypełnieniem modelu, zauważaną również w kolejnych krokach.

a)



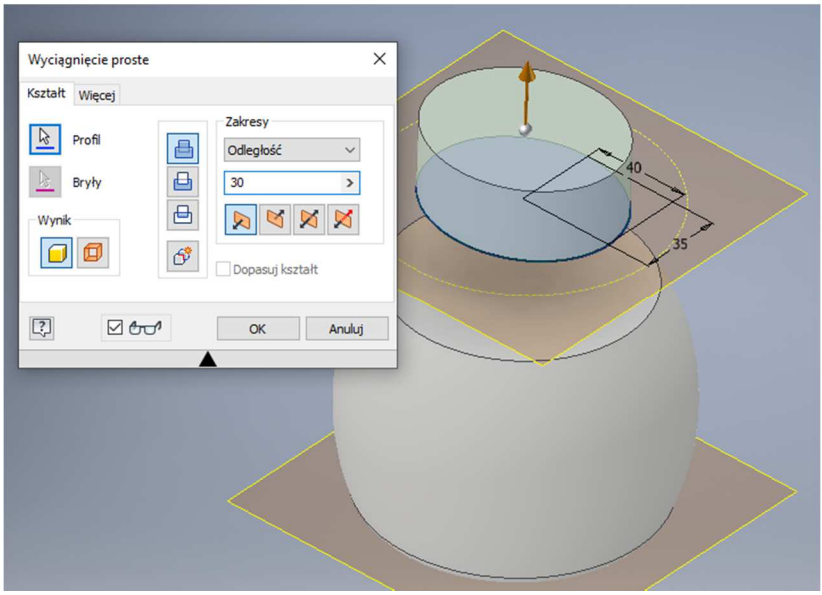
b)



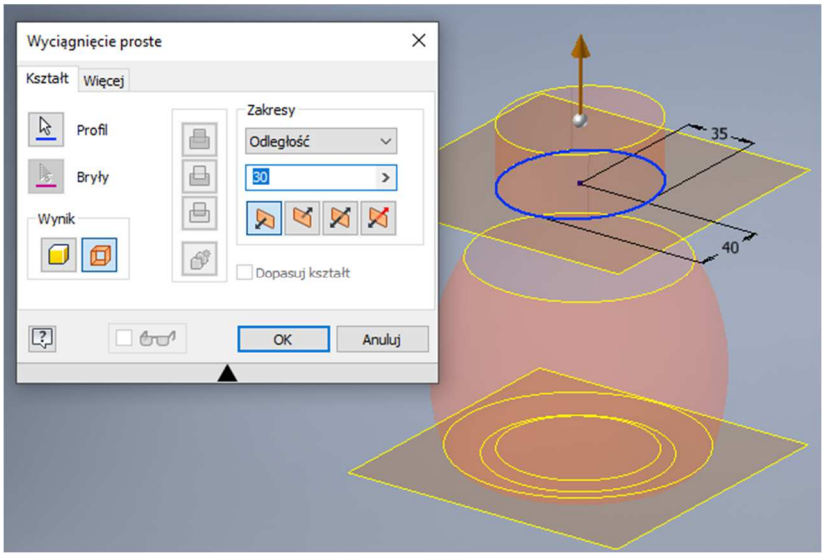
Rys. 6. Przykład obrotu szkicu: a) bryłowego, b) powierzchniowego
Źródło: opracowanie własne.

Kolejno operacja wyciągnięcia szyjki dzbanka przebiega tak samo dla obydwu przypadków z wcześniej podkreśloną różnicą.

a)



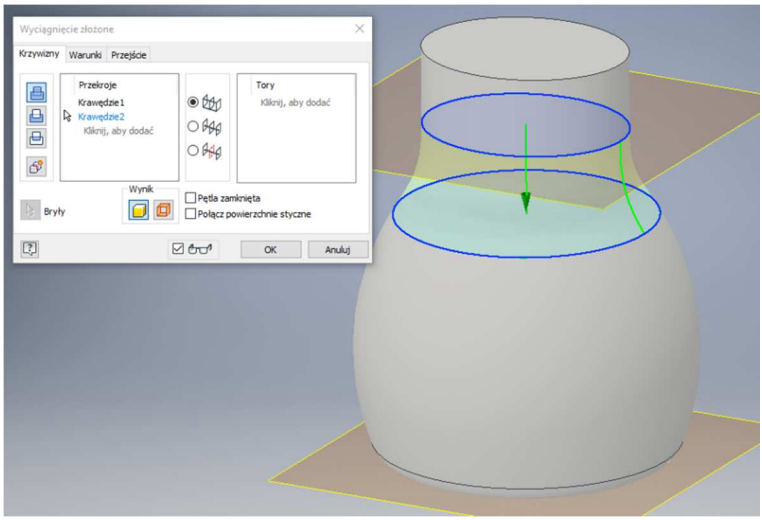
b)



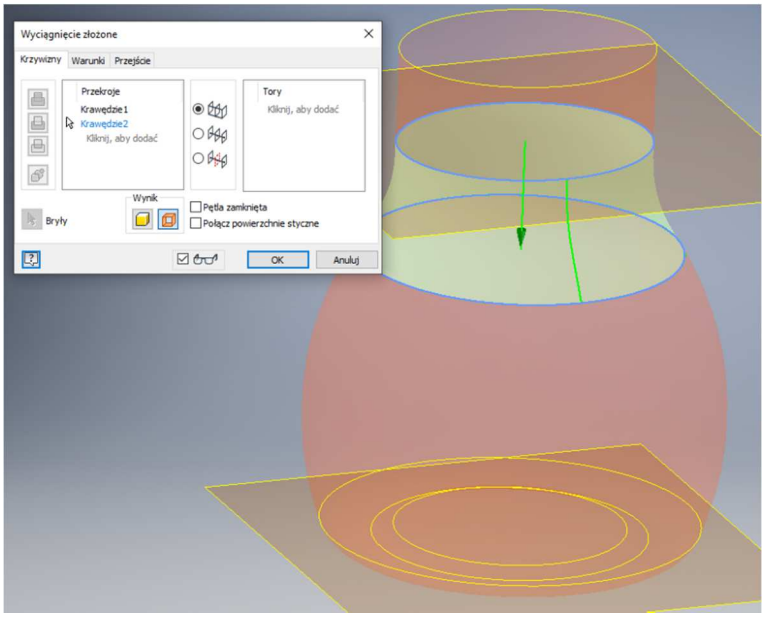
Rys. 7. Przykład wyciągnięcia szkicu: a) bryłowego, b) powierzchniowego
Źródło: opracowanie własne.

Obydwie części modelu zostają połączone operacją wyciągnięcia złożonego, wykorzystującą powstałe w poprzednich krokach zamknięte profile. Warto zauważyć, że do poprawnego połączenia modelu zastosujemy dla dwóch przypadków warunek ciągłości (G2).

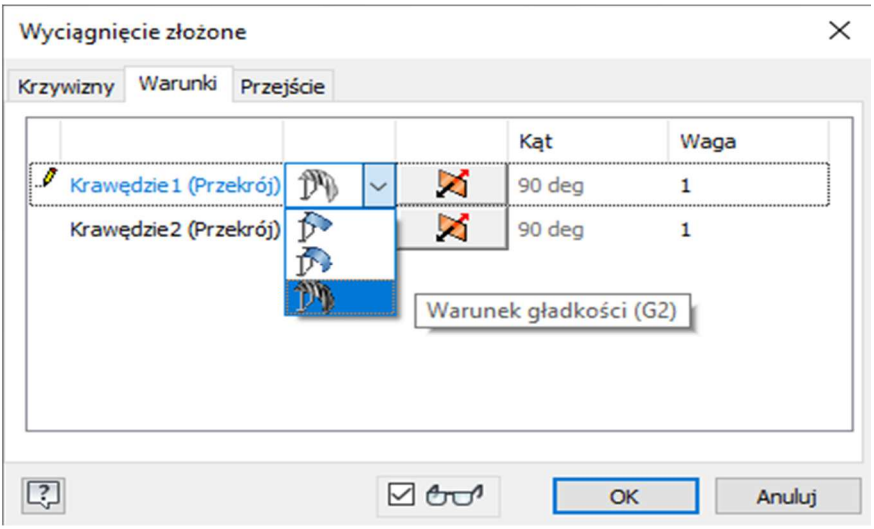
a)



b)



Rys. 8. Przykład wyciągnięcia złożonego w postaci: a) bryły, b) powierzchni
Źródło: opracowanie własne.



Rys. 9. Widoczna opcja warunku ciągłości wykorzystana dla obydwu modeli
Źródło: opracowanie własne.

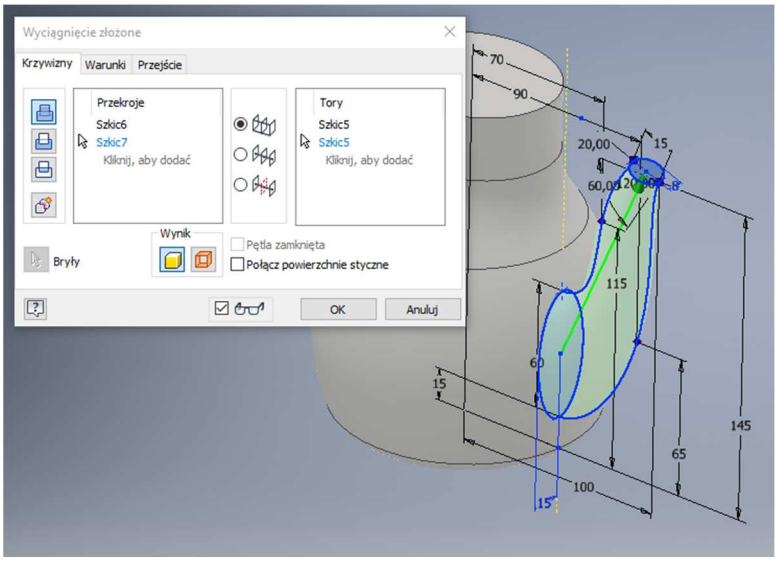
Następnie tworzymy szyjkę lejka za pomocą wyciągnięcia złożonego. Wskazujemy dwa szkice pomiędzy którymi powstaje wyciągnięcie, a następnie dodajemy szkice prowadzące tę operację. Po tym kroku różnice między sposobami modelowania mocno się uwidaczniają.

W kolejności rozdzielamy opis tworzenia modelu ze względu na duże rozbieżności w uporządkowaniu działań.

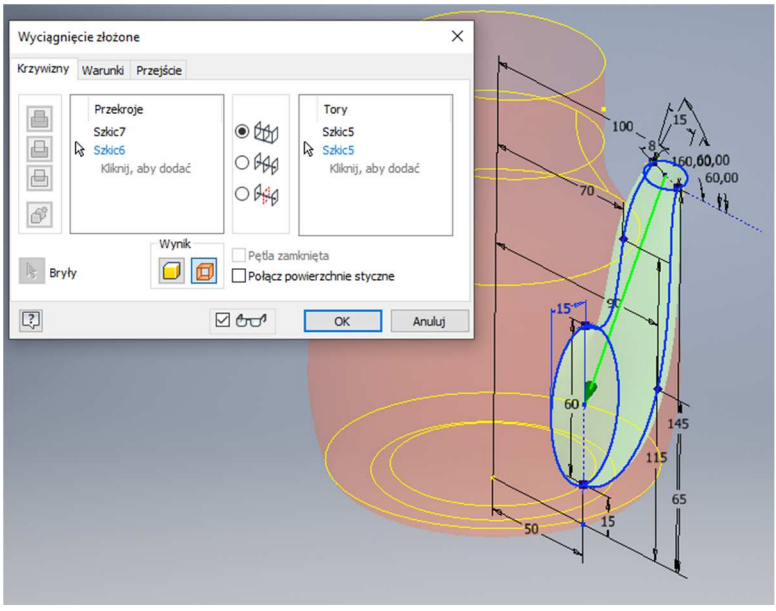
Zaczynając od modelowania bryłowego, wszystkie wyciągnięte wcześniej bryły są domyślnie łączone ze sobą. Co w następnym kroku zmusza nas do zaokrąglenia łączenia korpusu dzbanka z jego lejkiem. Pozwoli to w późniejszym kroku tworzenia skorupy na powstanie wewnętrznego zaokrąglenia.

Za pomocą polecenia “Skorupa” wybieramy wnętrze modelu, pozostawiając bryłę cienkościenną. Polecenie to jest przydatne, gdy element ma jednakową grubość. Jednak, kiedy bryła posiada różniące się wartości grubości ścianek, wtedy wykonanie modelu staje się problematyczne. W naszym przypadku przyjmujemy stałą grubość ścianki wynoszącą 3 milimetry.

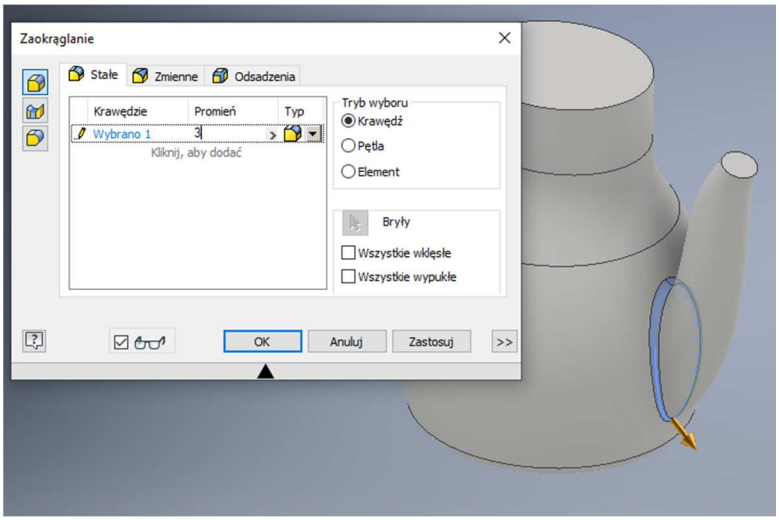
a)



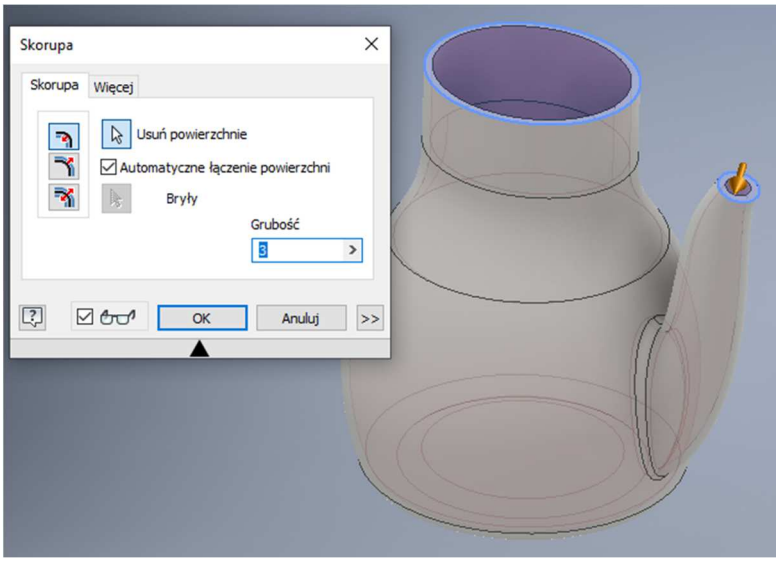
b)



Rys. 10. Tworzenie lejka operacją wyciągnięcia złożonego: a) bryłowego, b) powierzchniowego
Źródło: opracowanie własne.

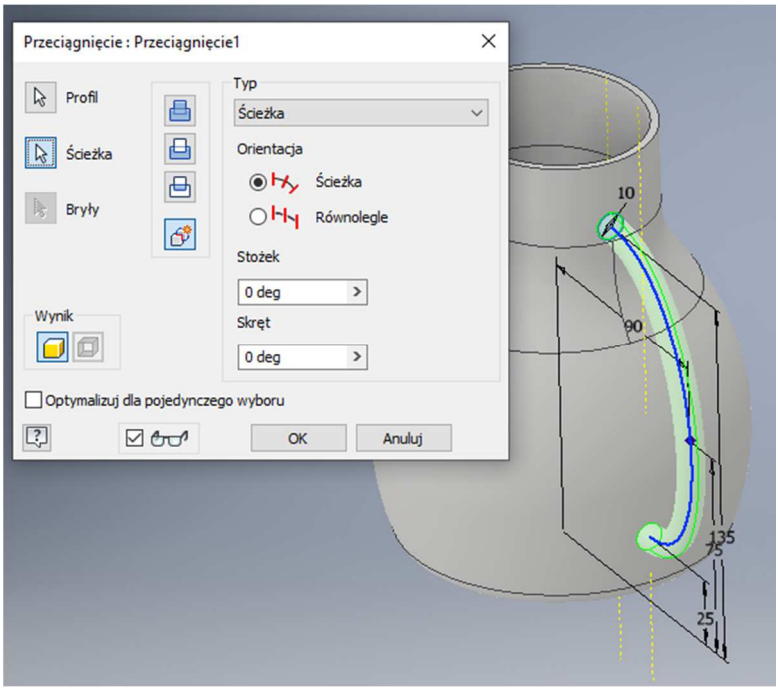


Rys. 11. Zaokrąglenie łączenia korpusu z lejkiem
Źródło: opracowanie własne.



Rys. 12. Tworzenie skorupy z powstałego modelu
Źródło: opracowanie własne

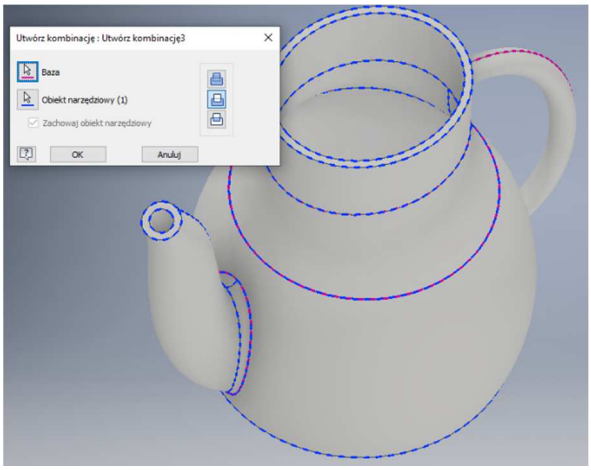
Tworzymy rączkę dzbanka jako nową bryłę, która zostanie odpowiednio do-cięta operacjami Boolowskimi.



Rys. 13. Tworzenie rączki dzbanka za pomocą przeciągnięcia po ścieżce
Źródło: opracowanie własne.

Tworzymy kombinacje brył, które docinają utworzoną rączkę do powierzchni korpusu dzbanka. Krok ten był konieczny ze względu na to, że rączka była widoczna wewnątrz korpusu.

a)

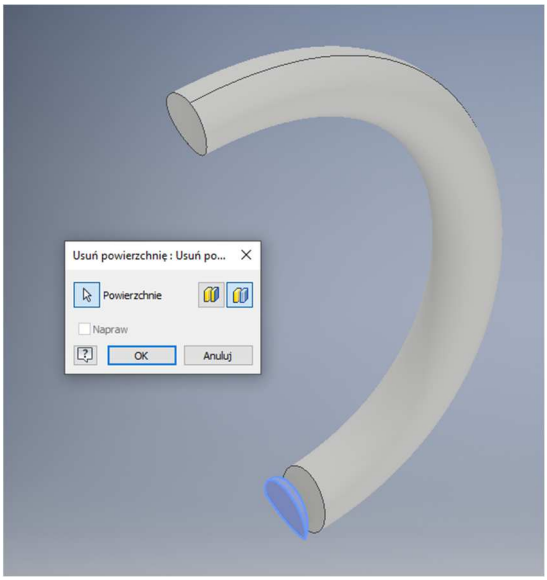


b)

Rys. 14. a) Kombinacja brył docinająca rączkę, b) docięta bryła rączki
Źródło: opracowanie własne.

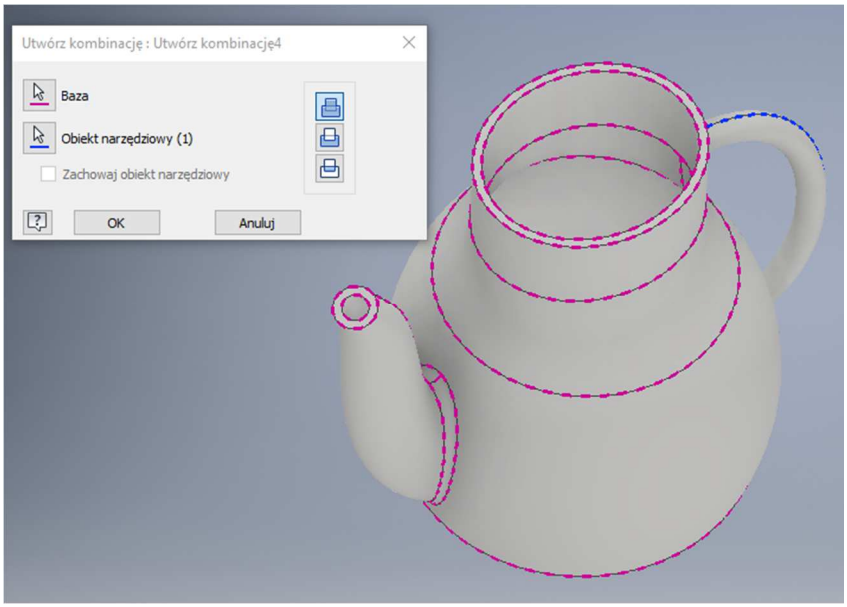
Usuwamy powstały zbiór powierzchni widocznej wewnątrz dzbanka. Finalnie dostajemy gotową bryłę uchwytu, którą łączymy z uprzednio powstałą bazą za pomocą opcji kombinacji modeli bryłowych.

a)



b)

Rys. 15. a) Zaznaczenie powierzchni do usunięcia,
b) rączka po operacji usunięcia powierzchni
Źródło: opracowanie własne.

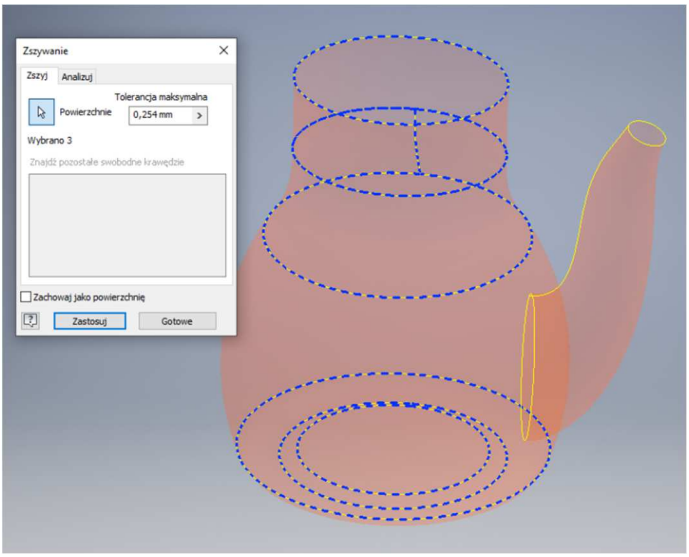


Rys. 16. Operacja połączenia powstałych modeli korpusu i rączki
Źródło: opracowanie własne.

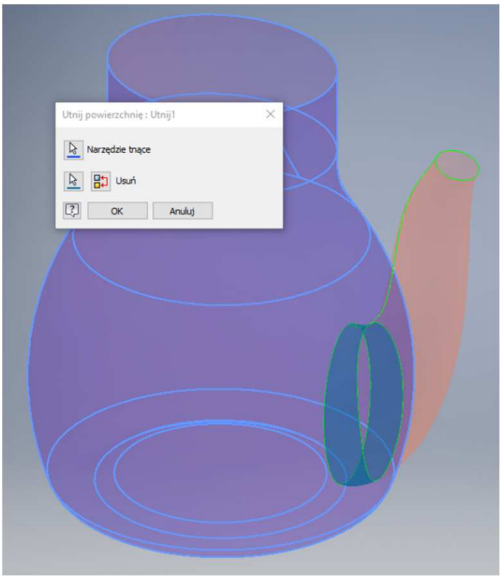
Następnie przechodzimy do modelu powierzchniowego, gdzie wykonujemy operację łączenia i docinania powierzchni. Jest to element konieczny, ponieważ płaszczyzny domyślnie nie łączą się. Opcja ta jest korzystna w przypadku tworzenia modeli o zmiennej grubości ścianki bocznej. Wraz ze stopniem skomplikowania modelu proces ten jest coraz bardziej użyteczny.

Wyróżniamy kilka podstawowych operacji podczas pracy z powierzchniami. Są to np.: zszyj powierzchnię, zamknij obwiednię, utnij powierzchnię, wydłuż powierzchnię itp. Operacje te pozwalają na szybką edycję stworzonych płaszczyzn. Narzędzie „zszyj powierzchnię" łączy ze sobą wszystkie zaznaczone elementy powierzchniowe, natomiast opcja „zamknij obwiednię" wstawia osobną płaszczyznę w środku zaznaczonego obszaru, dopasowując się do jego konturów.

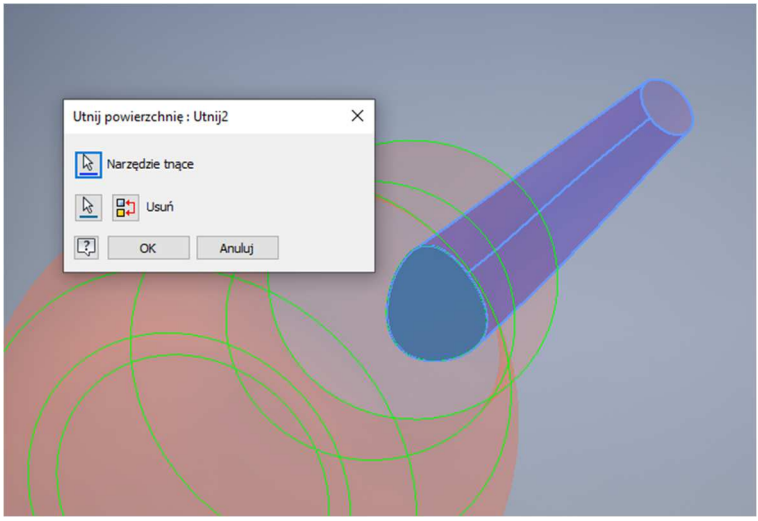
Po wykonaniu zszyć i odciąć tworzymy powierzchnię rączki za pomocą przeciągnięcia po ścieżce.



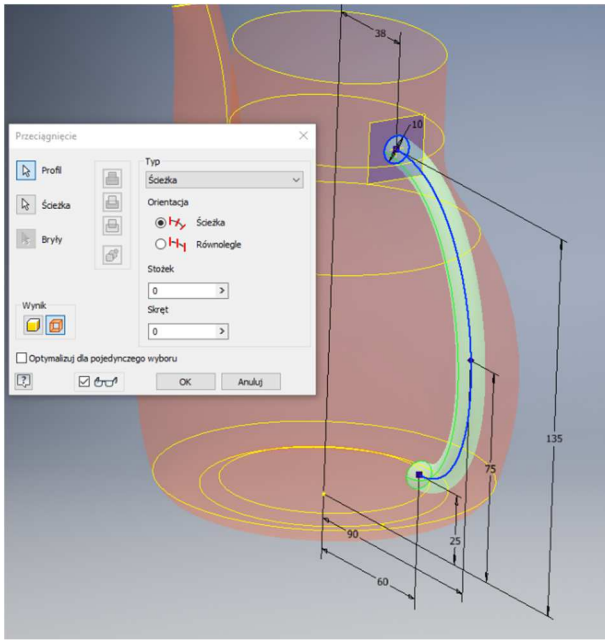
Rys. 17. Przykład operacji zszycia powierzchni korpusu dzbanka
Źródło: opracowanie własne.



Rys. 18. Docięcie zbędnej części powierzchni lejka
Źródło: opracowanie własne.

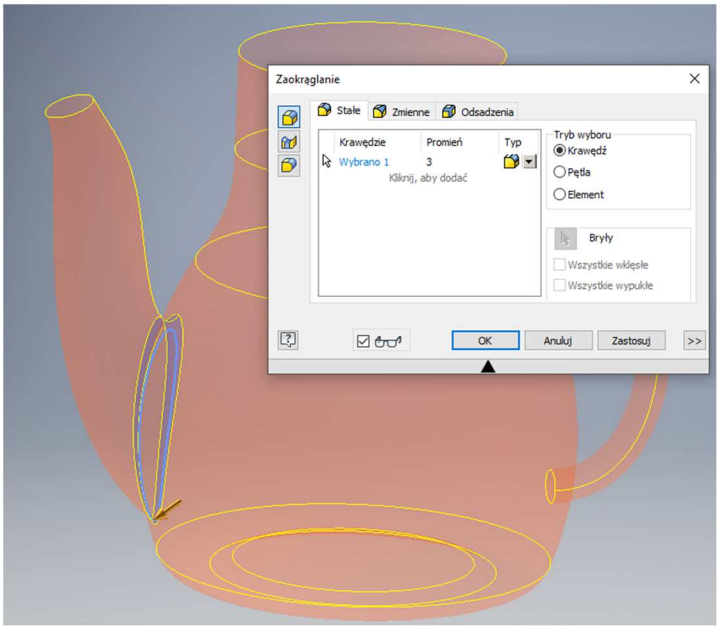


Rys. 19. Wykonanie otworu w korpusie dzbanka
Źródło: opracowanie własne.



Rys. 20. Tworzenie powierzchni rączki za pomocą operacji przeciągnięcia po ścieżce
Źródło: opracowanie własne.

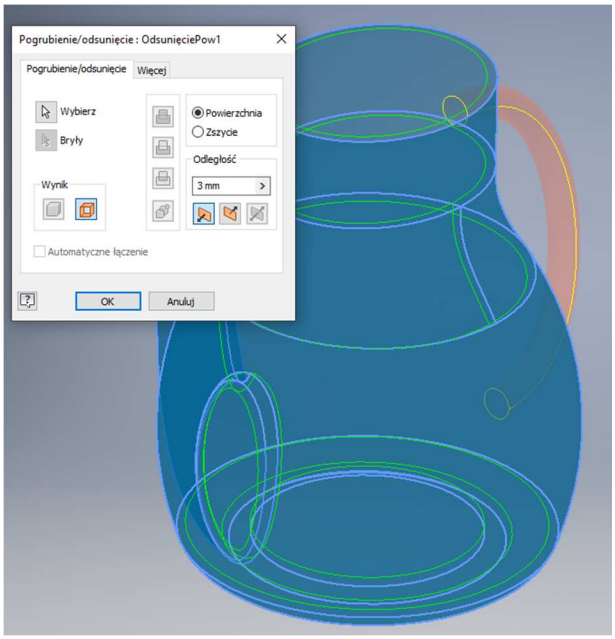
Wykonujemy zaokrąglenie w miejscu łączenia lejka z korpusem dzbanka, co spowoduje powstanie wewnętrznego zaokrąglenia, tak samo jak to miało miejsce w modelu bryłowym.



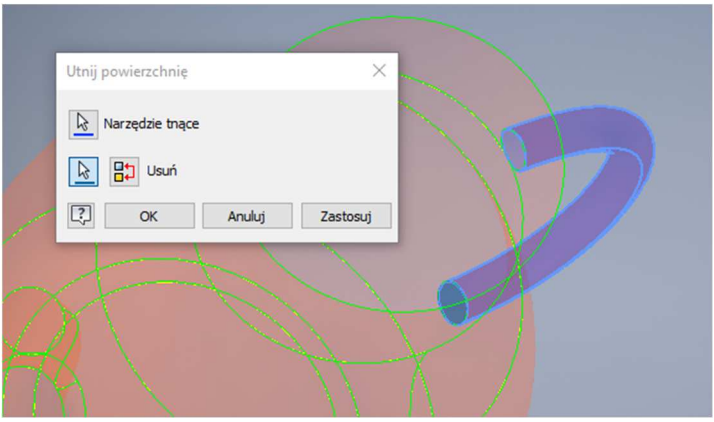
Rys. 21. Operacja zaokrąglenia łączenia lejka z korpusem
Źródło: opracowanie własne.

Kolejno odsuwamy utworzone powierzchnie z wyłączeniem rączki. Powstaje nam dodatkowa powierzchnia, która będzie granicą wewnętrzną modelu.

Docinamy uchwyt dzbanka do jego zewnętrznego korpusu. W porównaniu do operacji docinania elementów bryłowych, w tym przypadku usunięty zostanie cały fragment modelu za wskazaną płaszczyzną, a nie część modelu przecinająca się z inną bryłą.

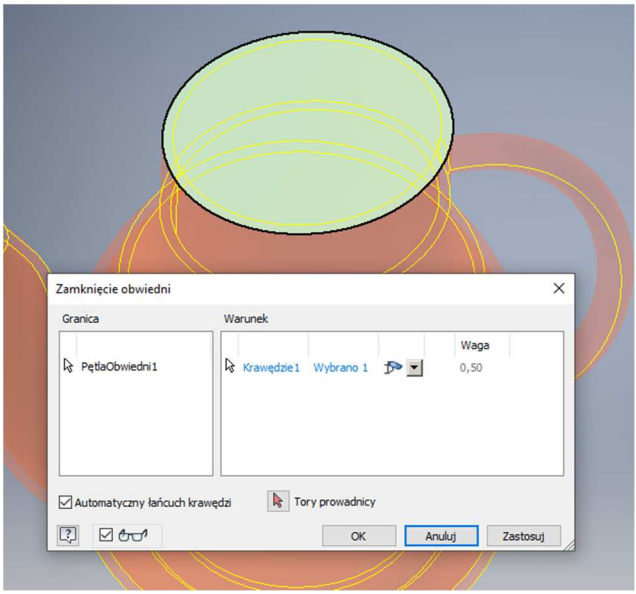


Rys. 22. Operacja odsunięcia zaznaczonych powierzchni
Źródło: opracowanie własne.



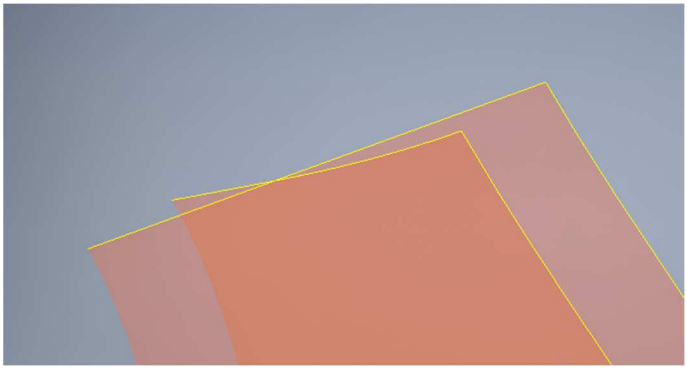
Rys. 23. Stworzenie otworów w korpusie
Źródło: opracowanie własne.

Korzystamy z polecenia “zamknij obwiednię”, tworząc górną granicę otworu powstałych dwóch powierzchni korpusu.

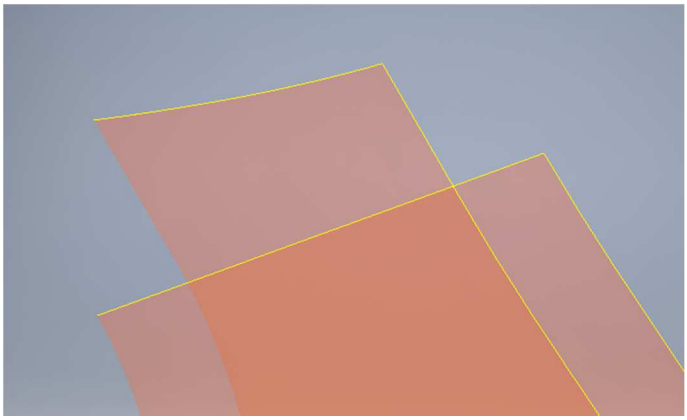


Rys. 24. Operacja zamknięcia górnej części korpusu dzbanka
Źródło: opracowanie własne.

Powstała podczas odsunięcia powierzchnia wewnętrzna części lejka wprowadza błąd w modelu. Dzięki temu nie możemy w całości okryć wewnętrznej objętości ograniczonej przez stworzony obszar. Spowodowane jest to przez model odsuwania powierzchni, której kierunek jest prostopadły do normalnej wprowadzonej w danym punkcie prostej. Rozwiązaniem tego problemu jest przedłużenie wewnętrznej powierzchni, co pozwoli na zamknięcie jej obwiednią, a następnie docięcie niepotrzebnych części modelu.

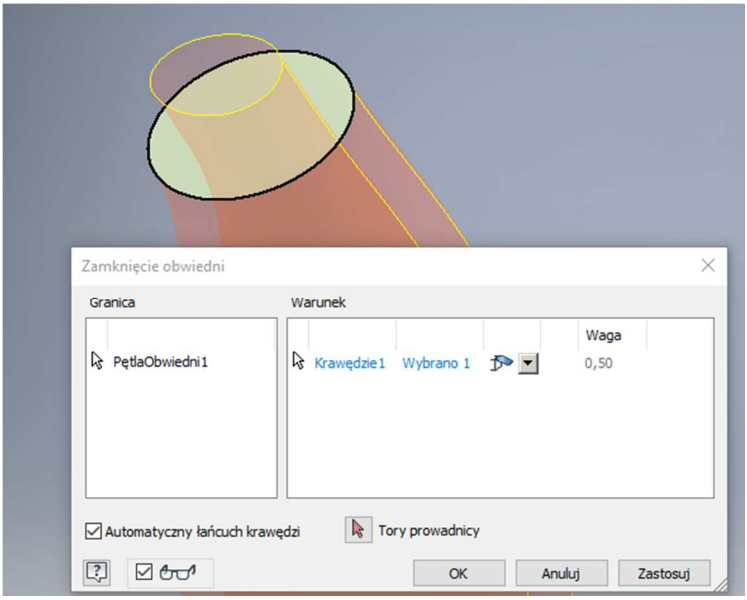


Rys. 25. Widok odsuniętej powierzchni w części końcowej lejka
Źródło: opracowanie własne.

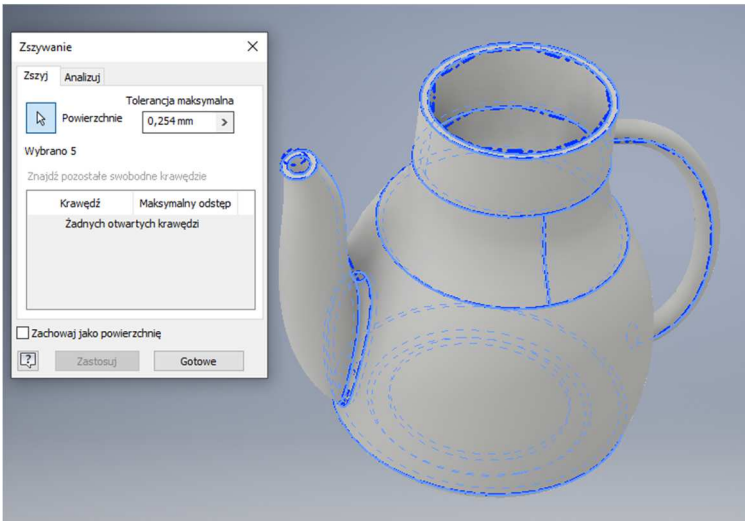


Rys. 26. Widok wydłużonej powierzchni w części końcowej lejka
Źródło: opracowanie własne.

Korzystając ponownie z narzędzia “zszyj”, łączymy wszystkie powstałe powierzchnie. Program, rozpoznając zamknięte obszary, tworzy automatycznie model bryłowy, który jest edytowalny wszystkimi opcjami dostępnymi dla tego typu modelu.

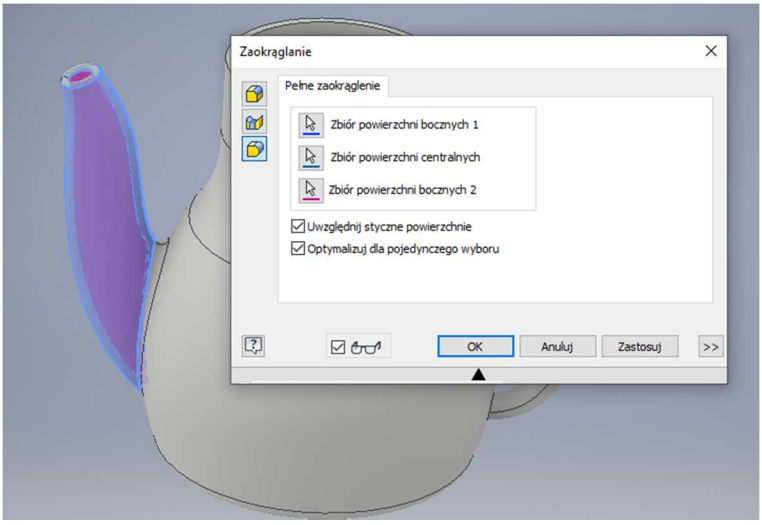


Rys. 27. Zamknięcie obwiedni końcówki lejka
Źródło: opracowanie własne.

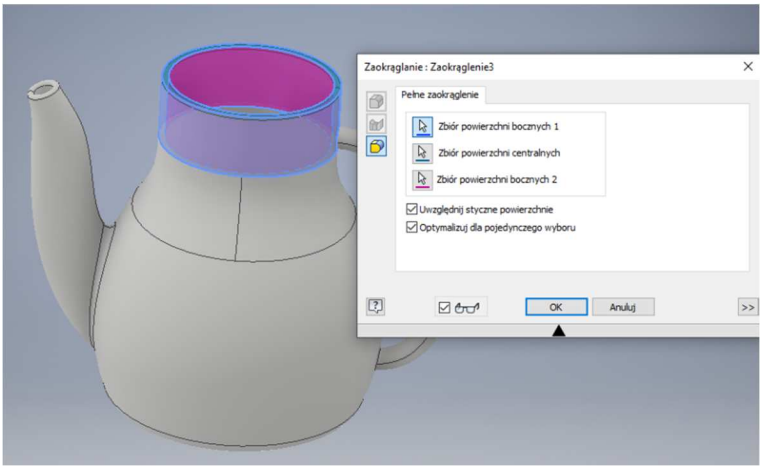


Rys. 28. Operacja zszycia wszystkich powierzchni modelu dzbanka
Źródło: opracowanie własne.

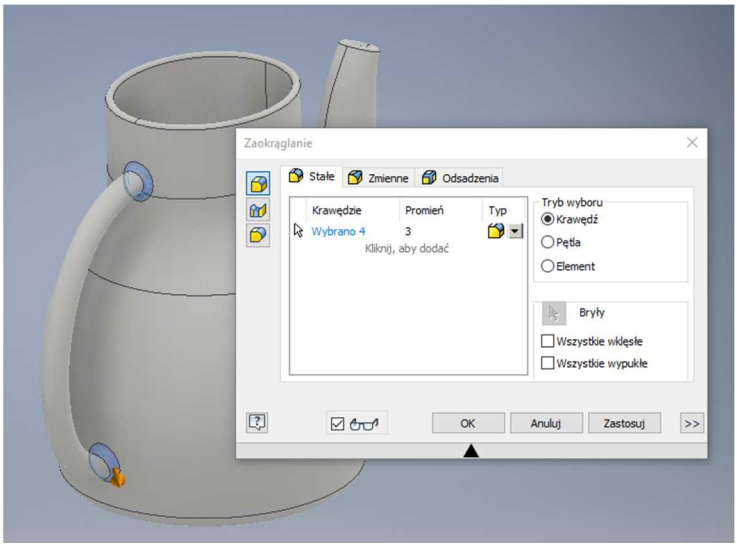
Wykańczając stworzony model, wykorzystujemy opcje zaokrągleń. Na tym etapie różnica spowodowana sposobem modelowania zanika i otrzymujemy tak samo wyglądające bryły. Samo wykończenie przebiega w identyczny sposób, prowadząc do finalnego efektu.



Rys. 29. Operacja pełnego zaokrąglenia końcówki lejka w obydwu modelach
Źródło: opracowanie własne.



Rys. 30. Operacja pełnego zaokrąglenia górnego wlewu dzbanka w obydwu modelach
Źródło: opracowanie własne.



Rys. 31. Zaokrąglenie połączenia rączki z korpusem w obydwu modelach
Źródło: opracowanie własne.

Rys. 32. Finalny wygląd zamodelowanych dzbanków
Źródło: opracowanie własne.

Możemy teraz wykonany dzbanek poddać innym procesom analizy powierzchni bądź symulacjom wytrzymałościowym.

a)  b) 

Rys. 33. Przykład analizy zebry dzbanka wykonanego metodą modelowania:
a) bryłowego, b) powierzchniowego
Źródło: opracowanie własne.

Na podstawie powyższej analizy zebry stwierdzamy, że obydwie bryły spełniają takie same warunki ciągłości powierzchni.

5. PODSUMOWANIE

Modelowanie współcześnie stało się wszechstronną dziedziną techniki, pozwalającą tworzyć wszystkie dostępne formy i kształty. Dodatkowo istnieje niezliczona ilość kształtów możliwych do wykonania różnorodnymi technikami. Analizując wykonane modele, możemy jasno stwierdzić, że każda technika ma wady, jak i zalety. Rolą konstruktora jest dobranie odpowiednich operacji, dzięki czemu jest w stanie stworzyć określony obiekt. Inżynier musi być świadomy ograniczeń dostępnych narzędzi. Ważną rolą w tworzeniu takich modeli jest czas.

W modelowaniu bryłowym główną rolę odgrywają dwuwymiarowe profile służące do definiowania poszczególnych operacji. Pozwala na osiągnięcie dowolnych kształtów, łatwych do wykonania w późniejszym procesie obróbki ubytkowej lub przyrostowej. Wraz ze wzrostem skomplikowania części o powierzchnie swobodne proces projektowania staje się uciążliwy dla użytkownika. Modelowa-

nie bryłowe stosowane jest głównie tam, gdzie najważniejszą rzeczą jest spełnienie założeń konstrukcyjnych dotyczących projektowania funkcjonalnego wyrobu, zaś estetyka odgrywa mniejszą rolę.

Zwracając uwagę na proces modelowania powierzchniowego mamy większą swobodę przy tworzeniu skomplikowanych modeli. Występuje głównie tam, gdzie spotykamy złożone kształty lub estetyka wykonania jest bardzo istotna.

Zazwyczaj przy projektowaniu części musimy zwracać uwagę na wiele aspektów, korzystnych zarówno dla modelowania bryłowego, jak i powierzchniowego. Rozwiązaniem tego problemu jest połącznie obydwu metod modelowania. W pierwszej kolejności tworzymy model konstrukcyjny, trzymając się wszystkich założeń. Następnie edytujemy go, zmieniając jego wygląd zewnętrzy za pomocą powierzchni. Takie połącznie modelowania bryłowego i powierzchniowego nazywamy modelowaniem hybrydowym.

LITERATURA

1. Tarnowski W., *Podstawy projektowania technicznego*. Wydawnictwo Naukowo-Techniczne, Warszawa 1997.
2. Winkler T., *Komputerowy zapis konstrukcji*. Wydawnictwo Naukowo-Techniczne, Warszawa 1997.

ŹRÓDŁA INTERNETOWE

3. https://techtutor.pl/autocad-3d-2-model-brylowy-i-powierzchniowy/ (dostęp: 29.05.2021).
4. https://zw3d.com.pl/latest-news/218-modelowanie-brylowo-powierzchniowe/ (dostęp: 12.04.2021).

KOŁO

NAUKOWE

INFORMATYKÓW

„KOD”

Mateusz FESZ

dr inż. Bartosz TRYBUS
opiekun naukowy

HISTORYCZNE ZASTOSOWANIA KRYPTOGRAFII W OBLICZU WSPÓŁCZESNYCH KOMPUTERÓW NA PRZYKŁADZIE SZYFRÓW PODSTAWIENIOWYCH ORAZ PRZESTAWNYCH

Artykuł przedstawia stosowane w przeszłości szyfry przestawne, będące jedną z najpopularniejszych w historii metod szyfrowania przekazywanych informacji oraz opisuje ich znaczenie w erze współczesnych komputerów o wysokiej mocy obliczeniowej. W artykule znalazła się analiza wielu kluczowych dla historii algorytmów szyfrowania, takich jak „szyfr Cezara" czy kojarzona przede wszystkim z II wojną światową maszyna Enigma. Omówione zostały zalety oraz wady tych oraz zbliżonych do nich algorytmów, połączone z dokładną analizą ich działania. Ponadto wykonane zostały implementacje elementów tychże szyfrów we współczesnym języku programowania Rust, o składni podobnej do C/C++, lecz stawiającym duży nacisk na bezpieczeństwo pamięci. Przeprowadzona analiza pokazuje ogólną prostotę tych algorytmów, połączoną jednak z ich dużą podatnością na próby złamania przy wykorzystaniu nowoczesnych technologii.
Słowa kluczowe: kryptografia, enigma, szyfry, algorytmy.

WPROWADZENIE

Kryptografia jest jedną z kluczowych dziedzin algorytmiki, istniejącą już od czasów starożytnych [1]. Co ciekawe, pierwsze szyfry nie służyły ukrywaniu treści, a raczej nadawaniu jej innej formy czy mistycyzmu. W artykule takie szyfry nie będą szerzej poruszane, ze względu na zupełnie inny cel wykorzystania od tego, który współcześnie przypisuje się szyfrom. Na przestrzeni lat pojawiały się różne metody ukrywania informacji przed osobami, które nie powinny jej otrzymać, co miało kluczowe znaczenie m.in. przy przekazywaniu informacji podczas wojen. Skuteczność takich szyfrów często decydowała, czy strony konfliktu będą w stanie przechwytywać rozkazy bądź meldunki wrogiej strony, co z kolei mogło zadecydować o wynikach bitew bądź negocjacji.

W czasach starożytnych najczęściej stosowane były dwie metody szyfrowania: przestawna bądź podstawieniowa. Pierwsza z nich polega na fizycznej zmianie położenia znaków w szyfrowanej wiadomości, co może zostać zrealizowane za pomocą algorytmu bądź fizycznej modyfikacji nośnika. Pozwala to na przekazanie informacji w sposób bezpieczniejszy niż jako tekst jawny (ang. *clear text*), ale zazwyczaj wymaga jakiegoś narzędzia pozwalającego na konstruowanie i dekonstrukcję szyfrogramu. Większą popularność zdobyła metoda podstawieniowa, polegająca na zamianie danego znaku tekstu jawnego w inny znak, niekoniecznie występujący w innym miejscu wiadomości. W przeciwieństwie do metod przestawnych, w większości, chociaż nie we wszystkich prostych szyfrach podstawieniowych, odszyfrowanie wiadomości nie wymagało zastosowania narzędzia, a jedynie znajomości klucza oraz odpowiedniego algorytmu kodującego bądź dekodującego.

Artykuł zawiera analizy kilku najpopularniejszych algorytmów szyfrowania stosowanych na przestrzeni lat, od starożytności aż do czasów współczesnych, kiedy to zostały one wyparte między innymi przez szyfrowanie asymetryczne, takie jak RSA, oparte na kilku kluczach: prywatnym i publicznym. W artykule poruszone są kwestie techniczne tych algorytmów, omówione zostały ich wady, zalety oraz szczegóły implementacyjne wraz z możliwymi metodami odszyfrowywania bez znajomości klucza.

1. KRÓTKO O SZYFRACH PRZESTAWNYCH NA PRZYKŁADZIE „SCYTALE”

Szyfry przestawne znalazły zastosowanie w ukrywaniu informacji jeszcze przed szyframi podstawieniowymi i mimo, że nie są bezpośrednim przedmiotem badań dla niniejszego artykułu, warte są przynajmniej krótkiego wspomnienia. Jak zostało wspomniane wcześniej, opierają się one o fizyczną zmianę położenia znaków w szyfrogramie (ang. *cipher text*) względem tekstu jawnego. Oznacza to, że zarówno ilość znaków, jak i ich bezpośrednia reprezentacja nie ulega zmianie, a proces deszyfracji polega na odwróceniu przestawienia, zwykle w dokładnie ten sam sposób, w jaki wykonany został szyfrogram.

Jednym z pierwszych szyfrów przestawnych był „Scytale”, stosowany przez starożytnych Greków, w szczególności Spartan [1]. Metoda szyfrowania była zaskakująco prosta, polegała na nawinięciu skórzanego pasa na pręt o zadanej średnicy i przekroju wielokąta. Następnie tekst był zapisywany wzdłuż pręta na jego kolejnych płaskich długościach, po czym odwijano pas z przyrządu i przekazywano posłańcowi. Na takiej skórzanej taśmie widniała cała zapisana wiadomość, lecz ze względu na zmianę rozkładu była ona praktycznie nieczytelna dla osoby nie znającej szyfru. Ponadto ze względu na metodę zapisu, do odczytania wiadomości konieczne było zastosowanie pręta o dokładnie tej samej średnicy i dokładne nawinięcie na niego szyfrogramu. W innej sytuacji jej zdekodowanie było względnie trudne – nawet jeśli osoba łamiąca szyfr znała zasadę jego działania, to

nadal musiała określić, które rzędy znaków należy odczytywać po sobie, co w zależności od przyjętej metody i dostępnych środków mogło być trywialne, bądź zająć absurdalnie długi czas.

Prosty schemat działania „Scytale" zaprezentowano w tab. 1.

Tabela 1. Przykładowa wiadomość „Ala ma psa a Jaś dwa", zapisana do tabeli zgodnie z metodą „Scytale"

A	L	A		M
A		P	S	A
	A		J	A
Ś		D	W	A

W tabeli 1. widzimy sposób szyfrowania „Scytale", w sytuacji gdy stosowalibyśmy 4-kątny pręt znajdujący się wirtualnie między drugim i trzecim wierszem i zapisywali dokładnie jeden znak na każdym zawoju skórzanego pasa. Czytając wiadomość w kolejności z góry na dół, a później od lewej do prawej otrzymujemy nasz szyfrogram, tzn. taką kolejność znaków, jaką będziemy widzieli po odwinięciu pasa z pręta. Algorytm ten możemy także zaimplementować w dowolnym języku programowania [2]. Przykładowy kod w języku Rust funkcji dokonującej szyfrowania zaprezentowano na listingu 1.

```
fn scytale(message: &str, cylinder_sides: usize) -> String{
    let msg = message.as_bytes();
    let row_len = if message.len()%cylinder_sides == 0{
        message.len()/cylinder_sides
    }else{
        message.len()/cyllinder_sides+1
    };
    let mut cipher = vec![32; row_len*cylinder_sides];
    for i in 0..row_len{
        let mut index = i;
        for j in 0..cylinder_sides{
            cipher[i*cylinder_sides + j] = msg[index];
            index += row_len;
        }
    }


    match String::from_utf8(cipher){
        Ok(s) => s,
```

```
        Err(_) => String::from("Error! Probably not-ASCII char-
acters present!")
    }

}
```

Listing 1. Funkcja przyjmująca jako argumenty wiadomość oraz ilość boków cylindra, zwracająca zaszyfrowaną wiadomość

Co ważne, algorytm deszyfrujący różni się od algorytmu szyfrującego, jeśli reprezentacja tablicowa wiadomości nie jest macierzą kwadratową, co oznacza, że w celu deszyfrowania wiadomości musimy dokonać transponowania macierzy, albo wykorzystać funkcję zaprezentowaną na listingu 2.

```
fn scytale_decrypt(message: &str, cylinder_sides: usize) ->
String{
    let msg = message.as_bytes();
    let row_len = if message.len()%cylinder_sides == 0{
        message.len()/cylinder_sides
    }else{
        message.len()/cylinder_sides+1
    };
    let mut cipher = vec![32; row_len*cylinder_sides];
    for i in 0..cylinder_sides{
        let mut index = i;
        for j in 0..row_len{
            cipher[i*row_len + j] = msg[index];
            index += cylinder_sides;
        }
    }


    match String::from_utf8(cipher){
        Ok(s) => s,
        Err(_) => String::from("Error! Probably not-ASCII char-
acters present!")
    }

}
```

Listing 2. Funkcja przyjmująca jako argumenty zaszyfrowaną wiadomość oraz ilość boków cylindra, zwracająca wiadomość w tekście jawnym

Patrząc na powyższy algorytm z perspektywy czasu, możemy stwierdzić, że mógł być on wyjątkowo trudny do złamania w czasach jego stosowania, ale opierał się w dużym stopniu na nieznajomości zasady szyfrowania przez osoby niepożądane. Osoba nieznająca klucza ani metody szyfrowania, mogła odczytać wia-

domość zasadniczo wyłącznie poprzez próby wpisania znaków do macierzy i znalezienia wśród nich wzoru. Jednakże, znając już zasadę działania kodu i wiedząc, że w praktyce wiadomość możemy odczytać poprzez odpowiednie dobranie ilości wierszy i kolumn, a następnie ich odczytanie w kolejności odwrotnej do oryginalnej wiadomości, złamanie szyfru staje się o wiele prostsze. W dzisiejszych czasach stosowanie tego typu szyfrów mija się z celem, gdyż ich złamanie przy użyciu komputerów jest wręcz trywialne: znając algorytm, wystarczy jedynie sprawdzić wszystkie możliwe wartości klucza z zakresu $\langle 2;\ n-1\rangle$, gdzie n oznacza ilość znaków szyfrogramu.

2. PROSTE SZYFRY PODSTAWIENIOWE. SZYFR CEZARA

Szyfry podstawieniowe, w odróżnieniu od szyfrów przestawnych nie zmieniają pozycji znaków w wiadomości, a zastępują część bądź wszystkie znaki innymi znakami. Takie podejście sprawia, że szyfrogram nie wymaga specyficznej formy zapisu, jak na przykład opisany wcześniej „Scytale", a jedynie znajomości przez odbiorcę klucza i zasady szyfrowania. Jednym z pierwszych powszechnie stosowanych szyfrów podstawieniowych był szyfr Cezara [1]. Jego zasada działania była bardzo prosta, każdy znak tekstu jawnego posiadał przypisaną wartość odpowiadającą jego pozycji w alfabecie, a proces szyfrowania polegał na zamianie każdego ze znaków, na taki o wartości większej o wartość klucza. Przykładowa wiadomość zakodowana szyfrem Cezara z kluczem równym 3 zaprezentowana została w tab. 2.

Tabela 2. Przykładowa wiadomość zaszyfrowana szyfrem Cezara

A	L	A		M	A		P	S	A
D	O	D		P	D		S	V	D

Zaprezentowana powyżej metoda może zostać łatwo zaimplementowana w dowolnym języku programowania, a przykładowy kod Rust pokazano na listingu 3. Szyfrowanie tekstu *message* odbywa się tutaj z wartością przekazaną parametrem *key*.

```
fn caesar(message: &str, key: u8) -> String{
       let msg = message.as_bytes();
       let mut cipher = vec![32;message.len()];

       for  (index,character)  in  msg.iter().enumerate().fil-
ter(|(_, &c)| c.is_ascii_alphabetic()){
           cipher[index] = if character + key > 'Z' as u8{
               character + key - 26
           }else{
               character + key
           }
       }
       match String::from_utf8(cipher){
           Ok(s) => s,
           Err(_) => String::from("Error! Probably non-ASCII in-
put!")
       }
}
```

Listing 3. Przykładowa funkcja realizująca szyfrowanie metodą szyfru Cezara

Powyższy kod obsługuje również deszyfrowanie wiadomości, w tym celu jako wartość klucza deszyfrującego należy podać liczbę w postaci:

$$26 - klucz\ szyfrujący.$$

Jak łatwo się domyślić, szyfr Cezara jest kiepską techniką kryptograficzną z uwagi na szereg wad. Przede wszystkim istnieje wyłącznie 26 możliwych ustawień klucza, co sprawia że przeprowadzenie nawet najbardziej trywialnych metod ataku, jak „brute-force", nie zajmuje wiele czasu i może być z powodzeniem stosowane do odczytania ukrytej treści. Ponadto, jako że każdy znak przy każdym wystąpieniu jest zastępowany dokładnie tym samym innym znakiem, oznacza, że w celu poznania klucza wystarczy dokonać ataku na niewielki fragment wiadomości, a dopiero po poznaniu klucza użyć go do zdekodowania całej wiadomości. Ze względu na te wady, jak i wynikającą z nich prostotę w łamaniu szyfru przez współczesne komputery, nie oferuje on dziś praktycznie żadnego zabezpieczenia, choć jego elementy wciąż widać w bardziej skomplikowanych szyfrach, jak szyfr Vigenère'a opisany dalej, oraz nadal ma zastosowanie m.in. w grach znakowych czy w celu ukrycia jakiejś treści przed niezainteresowanym nią odbiorcą.

3. SZYFR VIGENÈRE'A

Jest to kolejny szyfr podstawieniowy, którego nazwa pochodzi od nazwiska XIX wiecznego kryptografa Blaise de Vigenère'a, chociaż faktycznie został opisany już w XVI wieku przez Giovana Batista Belaso [1]. Szyfr ten niweluje jedną z największych wad szyfru Cezara, w którym każdemu znakowi przypisywane jest inne przesunięcie, uzależnione od pozycji w alfabecie odpowiedniego znaku w słowie pełniącym rolę klucza. Proces szyfrowania polega na zapisaniu w dwóch wierszach klucza i wiadomości. Jeżeli nie pokrywają się długością, należy powtórzyć klucz odpowiednią ilość razy. Przykładowy zapis klucza i wiadomości w tekście jawnym („*ALA MA PSA*") pokazano w tab. 3.

Tabela 3. Przykład szyfrowania algorytmem Vigenère'a. Wyraźnie widać tutaj problem cykliczności klucza – pierwsza i ostania litera 'A' zostały zaszyfrowane jako 'K'

K	O	T	K	O	T	K	O	T	K
A	L	A		M	A		P	S	A
K	Z	T		A	T		D	L	K

Każdej literze klucza przypisana jest wartość zgodna z jej pozycją w alfabecie i oznaczająca, o ile pozycji należy zwiększyć wartość odpowiadającego jej znaku wiadomości w celu uzyskania szyfrogramu. Przykład funkcji szyfrującej zgodnie z algorytmem Vigenère'a zaprezentowano na listingu 4. Parametrami są tekst jawny i klucz, a wartością zwracaną szyfrogram.

```
    fn vigenere(message: &str, key: &str) -> String{
        let msg = message.as_bytes();
        let key = key.as_bytes().iter().map(|k| k-65).col-
lect::<Vec<u8>>();
        let mut cipher = vec![32;message.len()];

        for (index, character) in msg.iter().enumerate().fil-
ter(|(_, &c)| c.is_ascii_alphabetic()){
            let tmp_key = key[index%key.len()];
```

```
            cipher[index] = if character + tmp_key > 'Z' as u8{
                character + tmp_key - 26
            }else{
                character + tmp_key
            }
        }

        match String::from_utf8(cipher){
            Ok(s) => s,
            Err(_) => String::from("Error! Probably non-ASCII in-
put!")
        }
}
```

Listing 4. Przykład funkcji szyfrującej metodą Vigenère'a

Analizując powyższą funkcję, widzimy, że ma ona wiele elementów wspólnych z implementacją szyfru Cezara. Pojawia się jednak zasadnicza różnica w postaci zmiany klucza szyfrowania znaku z każdą iteracją pętli, co pozwala na znaczne skomplikowanie szyfru. Zastosowanie klucza znakowego ma jednak drobną wadę: w celu odszyfrowania wiadomości konieczna jest delikatna modyfikacja funkcji szyfrującej, pokazana na listingu 5.

```
fn vigenere_decrypt(message: &str, key: &str) -> String{
    let msg = message.as_bytes();
    let    key    =    key.as_bytes().iter().map(|k|    k-65).col-
lect::<Vec<u8>>();
    let mut cipher = vec![32;message.len()];

    for (index, character) in msg.iter().enumerate().filter(|(_,
&c)| c.is_ascii_alphabetic()){
        let tmp_key = key[index%key.len()];
        cipher[index] = if character - tmp_key < 'A' as u8{
            character - tmp_key + 26
        }else{
            character - tmp_key
```

```
        }
    }

    match String::from_utf8(cipher){
        Ok(s) => s,
        Err(_) => String::from("Error! Propably non-ASCII input!")
    }
}
```

Listing 5. Przykład funkcji deszyfrującej metodą Vigenère'a

Zastosowana metoda bardzo utrudnia atak „brute-force", ponieważ nawet znając algorytm szyfrujący, nadal musielibyśmy znaleźć nie tylko wartość, ale i długość klucza, co byłoby procesem znacznie dłuższym niż w przypadku zwykłego przesunięcia jak w szyfrze Cezara. Niestety, mimo zastosowania klucza o zmiennej długości, nadal pozostaje jego wada w postaci cykliczności, co oznacza, że okresowo dana litera szyfrogramu będzie miała to samo przesunięcie względem swojej nominalnej pozycji. Przy wystarczająco długich wiadomościach pozwala to na dokonanie analizy częstotliwości, jednej z historycznie najbardziej dokładnych metod łamania szyfrów.

Dla zwykłego szyfru podstawieniowego, znając częstotliwość występowania znaków w języku, w którym przekazywane są informacje, możemy z nią zestawić częstotliwości występowania znaków w szyfrogramie. Dzięki temu możliwe jest przypisanie dowolnemu znakowi szyfrogramu prawdopodobieństwa bycia każdą z liter alfabetu, a następnie odbudowanie tekstu jawnego poprzez próby podstawienia znaków o największym prawdopodobieństwie i odrzucenie tych zaburzających sens komunikatu. Metoda ta może być również po lekkiej modyfikacji użyta do ataku na zaprezentowany powyżej szyfr Vigenère'a. W tym celu konieczne jest założenie dowolnej długości klucza szyfrującego, a następnie sprawdzenie, czy częstotliwość znaków występujących w szyfrogramie okresowo, z krokiem równym właśnie tej założonej długości, pokrywa się z występowaniem jakichś znaków w danym języku. Następnie jeżeli wzajemna częstotliwość znaków jest podobna, możemy założyć, że przyjęliśmy poprawną długość klucza równą n, podzielić naszą wiadomość na n segmentów złożonych z kolejnych okresów dla każdej z nieznanych wartości klucza, po czym dla każdego z tych segmentów dokonać już zwykłej analizy częstotliwości. Proces ten nie gwarantuje powodzenia przy pierwszym dopasowaniu, dlatego, jeśli mimo zgodności częstotliwości dla poszczególnych okresów nie jesteśmy w stanie ułożyć na ich podstawie sensownego komunikatu, należy rozważyć wykonanie prób dla innej długości klucza. Metoda Vigenère'a jest przez to znacznie skuteczniejsza w ochronie in-

formacji od szyfru Cezara czy metody „Scytale", lecz nadal może być łatwo złamana przez współczesne komputery i nie oferuje akceptowalnego zabezpieczenia przekazywanej wiadomości.

4. ENIGMA

Jedną z największych wad szyfrów podstawieniowych jest ich cykliczność, a co za tym idzie, podatność na ataki analizy częstotliwości. Pojawia się jednak pytanie, czy nie można tego problemu wyeliminować, a przynajmniej ograniczyć do poziomu, w którym nie będzie możliwe jego praktyczne wykorzystanie? Okazuje się, że przy odpowiednim skomplikowaniu algorytmu szyfrującego, możliwe jest uzyskanie tak długich cykli podstawieniowych, że jakakolwiek zaszyfrowana przy ich pomocy wiadomość nie będzie realnie możliwa do złamania poprzez wykorzystanie typowych metod analizy kryptograficznej. Jednym z najskuteczniejszych, a zarazem najbardziej znanych mechanizmów szyfrowania podstawieniowego była maszyna Enigma, której podstawowy model skonstruował Arthur Scherbius w 1918 roku [3]. Jej zasada działania opierała się na pracy trzech obrotowych bębnów oraz pojedynczego bębna stacjonarnego, zwanego odwracającym, powodujących przekształcenie znaku w inny poprzez przekierowanie prądu płynącego od wciśniętego klawisza do jednej z żarówek na wyświetlaczu. Każdy z obrotowych bębnów dokonywał zamiany znaku wejściowego na inny znak wyjściowy i przekazywał sygnał do kolejnego bębna. Ponadto każdy wprowadzony znak powodował obrót skrajnego bębna o jedną pozycję, jego pełny obrót powodował obrót bębna środkowego o jedną pozycję, a pełny obrót bębna środkowego powodował pełny obrót bębna wewnętrznego. Operacje zamiany znaku dokonywane przez każdy z bębnów możemy zapisać jako permutację w postaci cyklowej, co pokazuje listing 6.

```
(AELTPHQXRU) (BKNW) (CMOY) (DFG) (IV) (JZ) (S)
```

Listing 6. Przykładowy zapis cyklowy permutacji dla pierwszego bębna stosowanego w maszynach Enigma

W procesie szyfrowania litery były kodowane zgodnie z cyklami kolejnych bębnów, następnie sygnał trafiał do bębna odwracającego, zwanego też reflektorem, który cechował się tym, że składał się wyłącznie z cykli długości 2 (A szyfrowane w Z oznaczało, że Z będzie szyfrowane w A). Następnie prąd ponownie przepływał przez bębny obrotowe, przy czym należy zaznaczyć, że przy tym drugim przejściu szyfrowanie następowało zgodnie z permutacjami odwrotnymi do przypisanych bębnom. Takie rozwiązanie miało specyficzną cechę: sprawiało, że możliwe było zarówno szyfrowanie, jak i deszyfrowanie z użyciem tego samego urządzenia i tego samego klucza, lecz jednocześnie prowadziło do jednej z krytycznych wad Enigmy – żadna z liter nie mogła być zaszyfrowana w samą siebie.

Fragment kodu emulatora Enigmy, realizujący opisane podstawienie pokazano na listingu 7.

```
fn process_letter(
    drums: &mut Vec<(Vec<char>, char, i32)>,
    alphabet: &Vec<char>,
    subs_alphabet: &Vec<char>,
    letter: char,
) -> char {
    if !alphabet.contains(&letter) {
        return letter;
    }
    let mut tmp = letter;
    move_drums(drums, &alphabet);

    let mut idx: i32 = subs_alphabet.iter().position(|&x| x ==
tmp).unwrap() as i32 - drums[2].2;
    if idx < 0 {
        idx = alphabet.len() as i32 + idx
    }
    tmp = drums[2].0[idx as usize];

    idx = alphabet.iter().position(|&x| x == tmp).unwrap() as i32
- drums[1].2;
    if idx < 0 {
        idx = alphabet.len() as i32 + idx
    }
    tmp = drums[1].0[idx as usize];

    idx = alphabet.iter().position(|&x| x == tmp).unwrap() as i32
- drums[0].2;
    if idx < 0 {
        idx = alphabet.len() as i32 + idx
    }
    tmp = drums[0].0[idx as usize];
```

```
    idx = alphabet.iter().position(|&x| x == tmp).unwrap() as
i32;
    tmp = drums[3].0[idx as usize];

    idx = drums[0].0.iter().position(|&x| x == tmp).unwrap() as
i32 + drums[0].2;
    if idx >= alphabet.len() as i32 {
        idx = idx - alphabet.len() as i32
    }
    tmp = alphabet[idx as usize];

    idx = drums[1].0.iter().position(|&x| x == tmp).unwrap() as
i32 + drums[1].2;
    if idx >= alphabet.len() as i32 {
        idx = idx - alphabet.len() as i32
    }
    tmp = alphabet[idx as usize];

    idx = drums[2].0.iter().position(|&x| x == tmp).unwrap() as
i32 + drums[2].2;
    if idx >= alphabet.len() as i32 {
        idx = idx - alphabet.len() as i32
    }
    tmp = subs_alphabet[idx as usize];
    return tmp;
}
```

Listing 7. Funkcja realizująca podmianę pojedynczego znaku podczas procesu szyfrującego [4]. Pełny kod dostępny jest pod adresem: https://github.com/MatiF100/rust_Enigma

Powyższy kod wykorzystuje przygotowane wcześniej tablice alfabetu oraz bębnów szyfrujących w celu zasymulowania działania oryginalnego urządzenia. Odwzorowanie to jest bliskie ideałowi, lecz wciąż może zostać zoptymalizowane.

Zastosowanie obrotowych bębnów, a w późniejszych wersjach maszyny również łącznicy wtyczkowej sprawiło, że Enigma przez długi czas pozostawała szyfrem niemożliwym do złamania. Dopiero trójka polskich matematyków: Marian Rejewski, Jerzy Różycki oraz Henryk Zygalki zdołała rozpracować metodę

jej działania i opracować skuteczne metody łamania pierwszych wersji urządzenia, a ich praca pozwoliła aliantom na rozpracowanie późniejszych, znacznie bardziej złożonych modyfikacji szyfru [3].

Obecnie jednak nawet tak złożony szyfr nie może być skutecznie wykorzystywany do ochrony informacji przed odczytaniem przez osoby nieautoryzowane. Jednym z problemów, który Enigma współdzieli ze wszystkimi poprzednio opisanymi szyframi, jest oparcie się w dużym, chociaż nie w krytycznym stopniu, na nieznajomości działania mechanizmu szyfrującego przez atakującego. Pierwszy komputer zdolny do siłowego złamania szyfru w rozsądnym czasie, nazwany Colossus, powstał już w 1943 roku i był oparty na lampach elektronowych.

5. PODSUMOWANIE

Wszystkie wymienione w artykule algorytmy szyfrujące są bardzo ciekawe zarówno pod względem ich historii, jak i zasady działania. Ponadto wiedza na ich temat wydaje się kluczowa w celu lepszego zrozumienia współcześnie stosowanych algorytmów symetrycznych i asymetrycznych takich jak AES czy RSA. Analiza wad oraz metod łamania tych algorytmów pozwala nam jasno określić, czego należy unikać, kierując się wyborem algorytmu, który zamierzamy zaimplementować w naszym programie. Wartym wspomnienia jest też fakt, że w większości współcześnie stosowanych szyfrów algorytm jest w pełni transparentny i w żaden sposób nie podejmuje się prób jego ukrycia, a jedynym elementem, który w założeniu musi pozostać tajny, jest klucz deszyfrujący.

LITERATURA

1. Karbowski M., *Podstawy Kryptografii*. Wydanie III, Helion 2014.
2. Stephens. R., *Essential Algorithms: A Practical Approach to Computer Algorithms Using Python and C#*, John Wiley & Sons, 2019.

ŹRÓDŁA INTERNETOWE

3. Informacje na temat budowy oraz historii maszyny Enigma: http://edu-inf.waw.pl/inf/hist/006_col/ (dostęp: 1.07.2021).
4. Kod źródłowy emulatora Enigmy: https://github.com/MatiF100/rust_Enigma (dostęp: 1.07.2021).

Jakub PRZYSTASZ

dr inż. Bartosz TRYBUS
opiekun naukowy

IMPLEMENTACJA STEROWNIKA KONTROLERA INTERFEJSU ETHERNET W SYSTEMIE MicrOS

Artykuł opisuje sposób implementacji sterownika kontrolera interfejsu Ethernet Realtek 8139 w systemie operacyjnym MicrOS, który jest projektem rozwijanym przez Studenckie Koło Naukowe Informatyków „KOD". System jest przeznaczony na platformę x86, pracujący w trybie chronionym procesora. W ramach systemu zaimplementowane zostały podstawowe mechanizmy pozwalające na korzystanie z systemu na fizycznym sprzęcie, z możliwością uruchamiania własnych aplikacji. Niniejszy artykuł zawiera opis implementacji sterownika interfejsu Ethernet Realtek 8139, który obsługuje standard FastEthernet. W artykule zawarto fragmenty kodu źródłowego sterownika.
Słowa kluczowe: system operacyjny, karta sieciowa, interfejs Ethernet.

WPROWADZENIE

Pierwsze interfejsy Ethernet bazujące na rodzinie układów Realtek 8139 [4,5] pojawiły się na rynku w 1999 roku, szybko zyskując ogromną popularność ze względu na stosunkowo niską cenę w odniesieniu do możliwości układu: obsługa 10Base-T oraz 100Base-T, auto-negocjowanie szybkości połączenia, praca w trybie Half/Full Duplex, wsparcie dla Wake-On-Lan, ACPI.

Układ komunikuje się z procesorem za pośrednictwem interfejsu PCI [7], może posiadać do 128 KB[1] pamięci nieulotnej EEPROM, posiada dwa bufory FIFO: dla pakietów przychodzących oraz wychodzących, o rozmiarze 2 KB każdy, zegar taktujący układ pracuje z częstotliwością 25 MHz. Celem niniejszego artykułu jest przybliżenie metody implementacji sterownika kontrolera interfejsu sieciowego w systemie MicrOS. Omawiany układ określany będzie dalej jako RTL8139.

[1] Jednostki w tym artykule bazują na standardzie IEC 80000-13:2008.

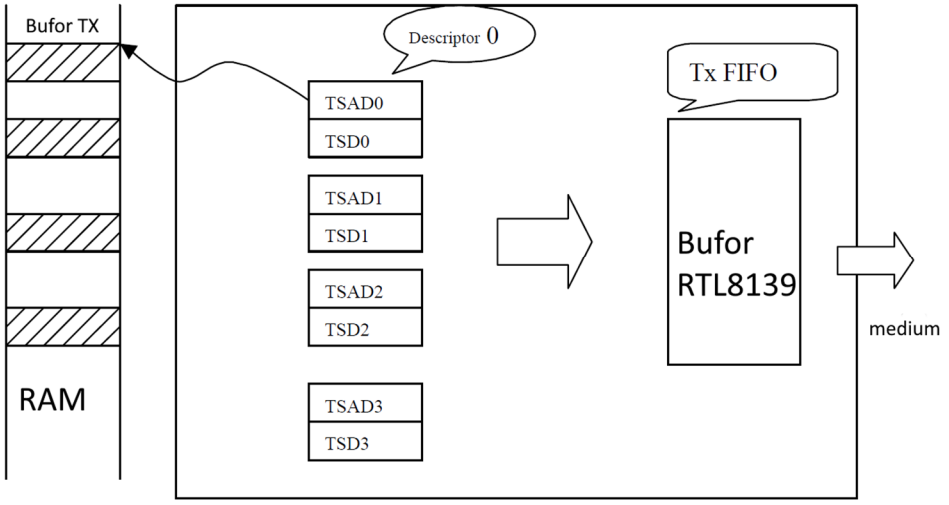
1. WYSYŁANIE ORAZ ODBIERANIE PAKIETÓW

Wysyłanie pakietów odbywa się za pośrednictwem czterech deskryptorów, z czego każdy składa się ze struktur *Transmit Status of Desciptor* oraz *Transmit Start Address of Descriptor* (TSAD).

Transmit Status of Desciptor (TSD) przechowuje następujące informacje o pakiecie:

- rozmiar przesyłanych danych,
- flaga OWN – informująca o przesłaniu danych do kolejki FIFO,
- flaga TUN – podnoszona w przypadku wystąpienia zjawiska niedopełnienia bufora FIFO,
- flaga TABT – podnoszona w momencie przerwania przesyłania pakietu,
- flaga TOK – podnoszona w momencie poprawnej transmisji pakietu,
- flaga CRS – podnoszona w przypadku utraty medium transmisyjnego.

Transmit Start Address of Descriptor (TSAD) to rejestr, gdzie przechowywany jest adres wysyłanego pakietu.



Rys. 1. Przepływ danych z pamięci komputera do medium transmisyjnego
Źródło: https://www.cs.usfca.edu/~cruse/cs326f04/RTL8139_ProgrammersGuide.pdf (dostęp: 19.03.2021).

Rysunek 1. prezentuje przepływ wysyłanych danych z pamięci komputera do medium transmisyjnego [9]. Kolejność, w jakiej wykorzystywane są poszczególne deskryptory, określa algorytm karuzelowy – pierwszy pakiet zostanie prze-

słany z deskryptora TSAD0, następnie z TSAD1 itd. Po przejściu przez wszystkie deskryptory kontroler powraca do pierwszego.

Obsługa danych przychodzących może zostać zrealizowana na dwa sposoby:

- Bufor wejściowy (RX) traktowany jest jako bufor cykliczny, tzn. w przypadku, gdy pakiet zapisywany jest na końcu bufora, to część pakietu, która może spowodować przepełnienie bufora, przenoszona jest na jego początek.
- Bufor wejściowy (RX) traktowany jest jako blok ciągłej pamięci – pakiet zapisywany na końcu bufora w przypadku przepełnienia zapisywany jest w formie ciągłej. Z tego powodu rozmiar bufora musi zostać zwiększony o dodatkowe miejsce na pakiet. Sugerowany dodatkowy rozmiar bufora to 1,5 KB. Po przepełnieniu założonego rozmiaru bufora kolejne dane trafiają na początek bufora.

2. KOMUNIKACJA Z INTERFEJSEM

Wymiana informacji pomiędzy centralną jednostką obliczeniową, a interfejsem sieciowym odbywa się poprzez magistralę PCI [7]. W momencie inicjalizacji sterownika sprawdzane są wszystkie dostępne urządzenia podłączone do magistrali PCI, jeżeli identyfikator urządzenia jest równy 0x8139, natomiast identyfikator producenta jest równy 0x10EC. Oznacza to, że w komputerze poprawnie zainstalowano interfejs sieciowy RTL8139.

Wymienione wyżej informacje można odczytać za pomocą portu wejścia/wyjścia pod adresem 0xCF8, z którego możliwe jest pobranie konfiguracji dla wszystkich urządzeń podłączonych magistralą PCI. Tabela 1. zawiera strukturę danych reprezentującą konfigurację dla pojedynczego urządzenia PCI.

Konfiguracja urządzenia odbywa się poprzez wpisywanie wartości do rejestrów. Dostęp do rejestrów możliwy jest poprzez porty wejścia/wyjścia lub odwołując się do adresów pamięci fizycznej, na które mapowane są rejestry urządzenia. Informacja o tym, w jaki sposób otrzymać dostęp do urządzenia, znajduje się w rejestrze *Base address #0*. Pierwszy bit tego rejestru wskazuje na metodę dostępu:

- Bit o wartości 0 - rejestry urządzenia są dostępne pod adresem pamięci fizycznej, który to zapisany jest na ostatnich 28 bitach rejestru *Base address #0*.
- Bit o wartości 1 - dostęp do urządzenia odbywa się poprzez porty I/O, adres portu zapisany na bitach od 3 do 32 rejestru *Base address #0*.

Tabela 1. Struktura zawierająca informacje o urządzeniu podpiętym do magistrali PCI [7]

Numer rejestru	Offset	Bity 31-24	Bity 23-16	Bity 15-8	Bity 7-0
00	00	Device ID		Vendor ID	
01	04	Status		Command	
02	08	Class code	Subclass	Prog IF	Revision ID
03	0C	BIST	Header type	Latency Timer	Cache Line Size
04	10	Base address #0 (BAR0)			
05	14	Base address #1 (BAR1)			
06	18	Base address #2 (BAR2)			
07	1C	Base address #3 (BAR3)			
08	20	Base address #4 (BAR4)			
09	24	Base address #5 (BAR5)			
0A	28	Cardbus CIS Pointer			
0B	2C	Subsystem ID		Subsystem Vendor ID	
0C	30	Expansion ROM base address			
0D	34	Reserved			Capabilities Pointer
0E	38	Reserved			
0F	3C	Max latency	Min Grant	Interrupt PIN	Interrupt Line

3. KONFIGURACJA

Pierwszą czynnością przy konfigurowaniu kontrolera jest włączenie bezpośredniego dostępu do pamięci operacyjnej – DMA. Odbywa się to poprzez ustawienie trzeciego bitu (bit ten nazywany jest *Bus mastering bit*) w rejestrze poleceń urządzenia PCI. Przypisanie temu bitowi wartości 1 odpowiada za włączenie obsługi DMA.

Aby mieć pewność, że urządzenie pracuje na domyślnych ustawieniach, przed przystąpieniem do konfigurowania kontrolera należy wykonać procedurę *software reset* – kontroler nie będzie wysyłał ani odbierał pakietów, zawartość kolejek FIFO zostanie usunięta, wskaźnik buforu TX (pakiety wychodzące) powróci do pozycji domyślnej – TSAD0. Wprowadzenie kontrolera do trybu *software reset* inicjowane jest poprzez ustawienie piątego bitu w rejestrze poleceń *Command Register*, który dostępny jest pod adresem 0x37. Wskazany adres rejestru jest adresem względnym. Aby skorzystać z deskryptora, należy najpierw odwołać się do adresu urządzenia poprzez port I/O lub poprzez mapowaną pamięć. Zakończenie procedury restartu sygnalizowane jest opuszczeniem flagi reset. Tabela 2. prezentuje zawartość rejestru poleceń.

Za pomocą wspomnianego rejestru poleceń należy włączyć lub wyłączyć odbieranie lub wysyłanie pakietów. Odpowiadają za to odpowiednio bity RE oraz TE. Jeśli wartość bitu wynosi 1, dana funkcjonalność jest włączona.

Tabela 2. Zawartość Command Register [9]

Numer bitu	Symbol
8-6	Niedostępne
5	RST – Reset
4	RE – Receiver Enable
3	TE – Transmitter Enable
2	Niedostępne
1	BUFE – Buffer empty

Sposób, w jaki kontroler ma obsługiwać pakiety przychodzące, można określić za pomocą *Receive Configuration Register*, którego najważniejsze bity zawarto w tab. 3.

Tabela 3. Fragment deskryptora 32-bitowego rejestru Receive Configuration [9]

Numer bitu	Symbol	Opis
8	WRAP	Jeśli 0 traktuj bufor RX jako cykliczny. W przeciwnym wypadku bufor jest ciągły
6	AER	Jeśli 1 akceptuj pakiety z błędną sumą kontrolną
5	AR	Jeśli 1 akceptuj pakiety mniejsze niż 64B.
4	AB	Jeśli 1 akceptuj pakiety rozgłoszeniowe
3	AM	Jeśli 1 akceptuj pakiety multicast
2	APM	Jeśli 1 akceptuj pakiety zaadresowane do interfejsu
1	AAP	Jeśli 1 akceptuj pakiety niezależnie od adresata

Odbieranie pakietów przychodzących może odbywać się na dwa sposoby podane niżej.

- Sterownik odpytuje cyklicznie kontroler o to, czy nadszedł jakiś pakiet – tzw. *pooling.*
- Kontroler zgłasza nadejście pakietu za pomocą przerwania, sterownik za pomocą funkcji obsługującej przerwanie przetwarza pakiet.

Konfiguracja przerwań możliwa jest poprzez ustawienie maski przerwania w rejestrze o nazwie *Interrupt Mask* (dostępnym pod adresem 0x3C). Ustawienie maski na wartość 1 mówi, że dane przerwanie może zostać wywołane przez kontroler. W tabeli 4. zamieszczono zestawienie dostępnych masek przerwań.

Tabela 4. Tabela maskowania przerwań [9]

Numer bitu	Symbol	Opis
16	SERR	Błąd systemu
15	TimeOut	Przekroczony limit czasu
7	FOVW	Przepełnienie kolejki FIFO RX
6	PUN	Przerwano transmisję pakietu/zmiana medium
5	RXOVW	Przepełnienie bufora RX
4	TER	Błąd transmisji
3	TOK	Poprawna transmisja
2	RER	Błąd odbierania pakietu
1	ROK	Poprawnie odebrano pakiet

W trakcie obsługi przerwania rejestr *Interrupt Status Register* (ISR) zawiera informację o tym, co spowodowało przerwanie. Rejestr dostępny jest pod adresem 0x3E. Pola w rejestrze są oznaczone analogicznie do tych, które opisano w tab. 4.

Architektura x86 jasno definiuje, które z urządzeń peryferyjnych może korzystać z określonych wektorów przerwań [3]. W przypadku interfejsów sieciowych możliwe jest skorzystanie z wektora o numerze 10 lub 11. Jak wskazano w tabeli 1., informacja o numerze przerwania, z którego korzysta przyłączone urządzenie, znajduje się w polu *Interrupt line*. W celu umożliwienia kontrolerowi RTL8139 zgłaszanie przerwań do CPU, należy odsłonić odpowiedni wektor przerwania w kontrolerze przerwań – dla architektury x86 jest to układ Intel 8259A. Pojedynczy układ 8259A potrafi obsłużyć do ośmiu urządzeń, dlatego najczęściej stosowana jest kaskada dwóch takich kontrolerów, z czego tylko pierwszy (*master*) ma bezpośrednie połączenie z główną jednostką obliczeniową. Taka konfiguracja powoduje dodatkowe problemy przy próbie zgłaszania przerwań do CPU, które wywoływane są z drugiego układu (*slave*). W sytuacji, gdy urządzenie podłączone do drugiego kontrolera zgłasza przerwanie, to sygnał w pierwszej kolejności trafia do głównego kontrolera przerwań, a dopiero stamtąd może zostać wysłany sygnał przerwania do procesora. Dlatego przy włączaniu wektora przerwania dla podrzędnego kontrolera (przerwania o numerach od 8 do 15) należy pamiętać, aby włączyć również przerwanie w głównym kontrolerze. W architekturze x86 *slave* zgłasza przerwania do głównego układu pod numerem 2.

4. IMPLEMENTACJA STEROWNIKA W SYSTEMIE MicrOS

Niniejszy rozdział przybliża sposób funkcjonowania sterownika interfejsu sieciowego. Pominięto aspekty związane z enkapsulacją pakietów.

```
typedef struct net_device
{
    kvector *rx_queue;
    kvector *tx_queue;
    char *device_name;
    uint8_t mac_address[MAC_ADDRESS_LENGTH];
    uint8_t ipv4_address[IPv4_ADDRESS_LENGTH];
    void (*send_packet)(net_packet_t *packet);
    void (*receive_packet)(net_packet_t *packet);
    uint32_t (*sent_count)(void);
    uint32_t (*received_count)(void);
} net_device_t;
```

Listing 1. Struktura kontrolera interfejsu sieciowego (NIC)

Wewnątrz sterownika zdefiniowane zostały struktury wspomagające organizację kodu. Jedną z takich struktur jest struktura definiująca kontroler interfejsu sieciowego przedstawiona na listingu 1.

Struktura agreguje wskaźniki na funkcje odpowiedzialne za wysyłanie oraz odbieranie pakietów przez NIC (*Network Interface Card*). Dodatkowo zawiera informacje o samym kontrolerze takie jak adres fizyczny MAC, adres IP, nazwa kontrolera. Mimo że w samym kontrolerze znajdują się bufory wejścia/wyjścia, to dla każdego z urządzeń przewidziane są dwa dodatkowe bufory na pakiety przychodzące/wychodzące – ich obecność pozwala zminimalizować prawdopodobieństwo wystąpienia zjawiska przepełnienia buforów karty sieciowej.

Do funkcji sterownika wysyłającej/odbierającej pakiet przekazywany jest wskaźnik na strukturę *net_packet* z listingu 2.

```
typedef struct net_packet{
    void *packet_data;
    uint32_t packet_length;
    uint8_t device_mac[MAC_ADDRESS_LENGTH];
} net_packet_t;
```

Listing 2. Struktura pakietu danych wysyłanych z/do sterownika

W strukturze znajduje się wskaźnik na przesyłane informacje, ilość przesyłanych informacji oraz adres fizyczny urządzenia, które te dane odebrało lub ma wysłać.

W trakcie uruchamiania systemu sterownik interfejsu sieciowego odpytywany jest w momencie uruchamiania modułu odpowiedzialnego za obsługę sieci. Wywoływana jest wtedy funkcja *rtl8139_init*, której fragment zawarto w listingu 3.

```
    net_dev->device_name = heap_kernel_alloc(strlen(DE-
VICE_NAME) + 1, 0);
    strcpy(net_dev->device_name, DEVICE_NAME);
    memcpy(net_dev->mac_address, rtl8139_device.mac_addr,
sizeof(uint8_t) * 6);
    net_dev->send_packet = &rtl8139_send_packet;
    net_dev->sent_count = &rtl8139_get_sent_count;
    net_dev->received_count = &rtl8139_get_received_count;
    receive_packet = net_dev->receive_packet;
```

Listing 3. Fragment funkcji inicjalizującej kontroler RTL8139

Fragment funkcji zaprezentowany w listingu 3. pokazuje sposób, w jaki struktura przekazana przez wskaźnik jako argument uzupełniana jest wskaźnikami na funkcje sterownika. Zmienna *receive_packet* jest wskaźnikiem wewnątrz sterownika NIC, który wskazuje na funkcję modułu sieciowego odpowiedzialnego za przetwarzanie przychodzących pakietów. Wysyłanie pakietów z modułu sieciowego możliwe jest za pośrednictwem wskaźnika *net_dev->send_packet*, który wskazuje na wewnętrzna funkcję sterownika *rtl8139_send_packet* wysyłającą dane przekazane przez wskaźnik.

Implementacja sterownika dla RTL8139 opiera się na przerwaniach, dlatego aby sterownik wiedział, że musi obsłużyć pakiet odebrany przez interfejs, konieczna jest implementacja funkcji obsługującej przerwanie jak na listingu 4.

```
bool rtl8139_irq_handler()
{
    uint16_t status = io_in_word(rtl8139_device.io_base +
INTRSTATUS);
    if (status & TOK)
        sent_count++;
    if (status & ROK)
    {
        received_count++;
        char str[] = "Packets count:     ";
        itoa(received_count,str+15,10);
        kprintf(str);
        rtl8139_receive_packet();
    }
    io_out_word(rtl8139_device.io_base + INTRSTATUS, 0x0);
    return true;
}
```

Listing 4. Funkcja obsługująca przerwania

W kontrolerze odsłonięte zostały maski przerwań TOK oraz ROK, dlatego jedynie te przerwania są obsługiwane przez funkcję. Makro *INTRSTATUS* ma wartość 0x3E, więc odczytywana jest wartość rejestru ISR. Następnie odczytana wartość porównywana jest w celu określenia czynnika, który wywołał przerwanie. Jeżeli karta zgłasza poprawne wysłanie pakietu – TOK, następuje inkrementacja zmiennej przechowującej ilość wysłanych pakietów. Jeśli przerwanie zostało wywołane z powodu odebrania pakietu – ROK, wywołana zostaje funkcja obsługująca nadchodzący pakiet *rtl8139_receive_packet*. Konieczne jest również przywrócenie domyślnej wartości rejestru ISR. Realizowane jest to wpisaniem wartości 0.

Wymieniona wyżej funkcja rtl8139_receive_packet wygląda jak na listingu 5.

```
void rtl8139_receive_packet()
{
    uint16_t *packet = (uint16_t *)(rtl8139_device.rx_buffer +
current_packet_ptr);
    uint32_t packet_length = *(packet + 1);
    void *packet_data = heap_kernel_alloc(packet_length, 0);
    memcpy(packet_data, packet + 2, packet_length);
    net_packet_t *out = heap_kernel_alloc(sizeof(net_packet_t),
0);
    out->packet_data = packet_data;
    out->packet_length = packet_length;
    rtl8139_get_mac_addr(out->device_mac);
    current_packet_ptr = (current_packet_ptr + packet_length +
8);
    if (current_packet_ptr > RX_BUFFER_SIZE)
        current_packet_ptr = current_packet_ptr_base;
    io_out_word(rtl8139_device.io_base + CAPR,
current_packet_ptr);
    (*receive_packet)(out);
```

Listing 5. Funkcja obsługująca nadchodzący pakiet

Rozmiar odebranego pakietu znajduje się za nagłówkiem, dlatego wskaźnik na odebrany pakiet został zwiększony o jeden: *packet_length = *(packet + 1)*. Zawartość pakietu jest kopiowana do struktury *net_packet_t* w celu późniejszego przetworzenia w module sieciowym. Utworzona struktura trafia na stertę, dlatego możliwe jest przekazanie wskaźnika do niej funkcji w module sieciowym. Przekazanie struktury do modułu sieciowego zrealizowane jest poprzez wskaźnik *receive_packet*. Aby sterownik wiedział, gdzie umieścić kolejną porcję odebranych danych, konieczne jest wskazanie nowej pozycji wskaźnika na bufor RX. Do makrodefinicji CAPR przypisana jest wartość 0x38. Jest to adres rejestru kontrolera

RTL8139, z którego kontroler odczytuje wskaźnik na bufor RX. W przypadku przepełnienia bufora RX kolejny pakiet trafi na początek bufora.

```
void rtl8139_send_packet(net_packet_t *packet)
{
    void *data = heap_kernel_alloc(packet->packet_length, 0);
    memcpy(data, packet->packet_data, packet->packet_length);
    void    *phys_addr    =    (void    *)((uint32_t)data    -
DMA_ADDRESS_OFFSET);
    io_out_long(rtl8139_device.io_base                          +
TSAD_array[rtl8139_device.tx_cur], (uint32_t)phys_addr);
    uint32_t status = 0;
    status |= packet->packet_length & 0x1FFF;
    status |= 0 << 13;
    io_out_long(rtl8139_device.io_base                          +
TSD_array[rtl8139_device.tx_cur], status);
    rtl8139_device.tx_cur++;
    if (rtl8139_device.tx_cur > 3)
        rtl8139_device.tx_cur = 0;
    heap_kernel_dealloc(data);
```

Listing 6. Funkcja sterownika odpowiedzialna za wysłanie pakietu

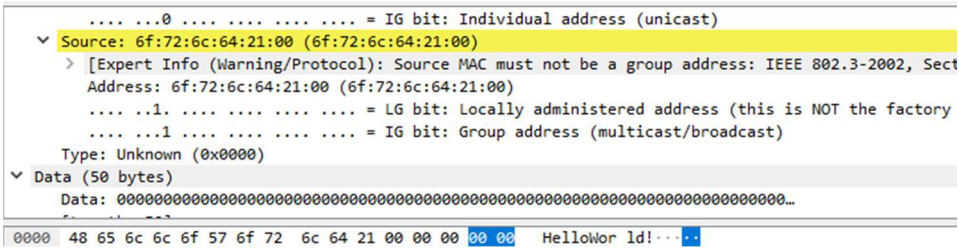
Listing 6. zawiera implementację funkcji sterownika RTL8139 odpowiedzialnej za wysłanie pakietu. Jako że przekazany wskaźnik na dane odnosi się do adresu w wirtualnej przestrzeni adresowej, a kontroler jest w stanie odczytać dane jedynie za pomocą wskaźnika na adres fizyczny pamięci operacyjnej, konieczna jest konwersja adresu wirtualnego na adres fizyczny. Aby mieć pewność, że przekazane dane znajdą się w pamięci operacyjnej, zawartość całego pakietu kopiowana jest na stos, a następnie wskaźnik na strukturę przeliczany jest na adres fizyczny. Makro *DMA_ADDRESS_OFFSET* definiuje przesunięcie, o jakie należy przeliczać adresy wirtualne na fizyczne. W przypadku systemu MicrOS pamięć fizyczna mapowana jest do przestrzeni wirtualnej z przesunięciem równym 0xC0000000. W zaprezentowanej funkcji realizowana jest zmiana bufora TSA za pomocą zmiennej *tx_cur*.

5. PODSUMOWANIE

Niniejszy artykuł przybliża problemy, z jakimi przychodzi zmierzyć się podczas implementacji sterownika dla kontrolera sieciowego. Zaprezentowane fragmenty kodu nie wykorzystują w pełni możliwości omawianego układu. Dzięki zastosowaniu warstwy abstrakcji, oddzielającej kod sterownika od modułu sieciowego odpowiedzialnego za enkapsulację pakietów, możliwe jest korzystanie

z różnych kontrolerów Ethernet bez ingerowania w kod przetwarzający pakiety. Na rysunku 2. zaprezentowano opis pakietu w programie Wireshark, wysłanego za pomocą opracowanego sterownika w środowisku emulatora Qemu.

Zrealizowany sterownik uzupełnił możliwości systemu o komunikację sieciową, dzięki czemu zakres zastosowań MicrOS znacząco się zwiększył.



Rys. 2. Program Wireshark: pakiet wysłany z systemu MicrOS za pomocą opracowanego sterownika

LITERATURA

1. Davidson J., *An Introduction to TCP/IP*, Springer, New York 1988.
2. Robertazzi T., *Basics of Computer Networking*, Springer, 2012.
3. Dandamudi S.P., *Fundamentals of Computer Organization and Design*, Springer, 2002.

ŹRÓDŁA INTERNETOWE

4. http://realtek.info/pdf/rtl8139cp.pdf (dostęp: 19.03.2021).
5. https://wiki.osdev.org/RTL8139 (dostęp: 19.03.2021).
6. https://wiki.osdev.org/Network_Stack (dostęp: 19.03.2021).
7. https://wiki.osdev.org/PCI (dostęp: 19.03.2021).
8. http://www.brokenthorn.com/Resources/OSDevPic.html (dostęp 19.03.2021).
9. https://www.cs.usfca.edu/~cruse/cs326f04/RTL8139_ProgrammersGuide.pdf (dostęp 19.03.2021).

Oskar TYNIEC

dr inż. Bartosz TRYBUS
opiekun naukowy

WPŁYW WDROŻENIA METODYK ZWINNYCH NA DZIAŁALNOŚĆ STUDENCKIEGO KOŁA NAUKOWEGO INFORMATYKÓW „KOD"

Artykuł opisuje, w jaki sposób elementy metodyk zwinnych takie jak tablica Kanban, role opisane w Scrum Guide oraz spotkania Scrumowe zostały wdrożone w działaniach Studenckiego Koła Naukowego Informatyków „KOD" na Politechnice Rzeszowskiej. W artykule opisane zostały również dwie bardzo popularnie stosowane wwspółcześnie metodyki zwinne, jakimi są Kanban oraz Scrum. Artykuł ukazuje pozytywne efekty wdrożenia takowych narzędzi. Jako główny wpływ metodyk zwinnych pokazano zwiększoną płynność w działaniach Koła Naukowego oraz poprawę organizacji czasu jego członków. Ostatnim aspektem okazał się ogólny rozwój pozwalający na zwiększenie znaczenia organizacji w otoczeniu społeczno-gospodarczym, co poskutkowało nawiązaniem współpracy z wieloma podmiotami oraz stworzeniu nowych ciekawych inicjatyw. Badania nad tematem artykułu były prowadzone od listopada 2019 roku, aż do maja 2021 roku, zaś Autor pełnił jednocześnie rolę Scrum Master w Studenckim Kole Naukowym Informatyków „KOD".
Słowa kluczowe: metodyki zwinne, scrum, agile, waterfall.

WPROWADZENIE

Na świecie istnieją dwa podejścia do zarządzania pracą: metodyki twarde oparte na modelu kaskadowym „Waterfall" oraz metodyki zwinne określane jako „Agile". Oba z tych podejść mają określone zasady ukazujące kierunek, w jaki sposób mają być prowadzone projekty. Metoda Waterfall skupia się na stopniowym przechodzeniu projektu przez kolejne etapy, takie jak opisywanie wymagań, projektowanie, wdrożenie, weryfikacja oraz utrzymanie. Etapy te zwykle wykonywane są kolejno po sobie, a gdy na jednym z nich zweryfikowana praca okaże się błędną, cały postęp cofany jest do miejsca, w którym popełniono błąd. „Waterfall", ze względu na bardzo mocne nastawienie na pracę z dokumentacją oraz wszelkiego rodzaju planowanie całego projektu z góry, wykorzystywany jest zwykle do projektów rządowych lub finansowanych ze środków państwowych, gdzie potrzebna jest akceptacja wszelkich prac przed przystąpieniem do nich.

Metodyki zwinne nazwane „Agile" skupiają się zaś na iteracyjnym podejściu, w którym praca wykonywana jest w mniejszych etapach, a działania planowane są zaledwie na parę tygodni bądź miesięcy do przodu. Przewaga metodyk Agile nad metodykami twardymi może być więc postrzegana w tym, że wytwarzany produkt ma szansę częstszej, tj. po każdej iteracji weryfikacji przez klienta czy docelowego użytkownika. Zapobiega to sytuacji, w której po wykonaniu całej pracy nad produktem okazuje się, że nie spełnia on wymagań klienta lub jest już przestarzały. W metodykach zwinnych klient może co iterację zobaczyć przyrost produktu jako działającej funkcjonalności. Po zaopiniowaniu przez klienta wymagania są aktualizowane, a praca która nie uzyskała akceptacji, cofana jest o zaledwie jedną iterację, a nie jak to czasami bywa w metodykach twardych – o całość procesu tworzenia.

Na świecie istnieje wiele metodyk zwinnych wykorzystywanych w wielu dziedzinach. Szczególnie znanymi praktykami zwinnymi do zarządzania pracą w projektach są Crystal, XP, Kanban oraz najbardziej popularny – Scrum [1]. Każdy z nich ma swoją specyfikę, lecz większość zawiera bardzo podobne lub wręcz takie same praktyki, choć wykorzystywane w innym celu. Dwie praktyki, Scrum i Kanban zostaną przedstawione w dalszej części artykułu. Warto zauważyć, iż ramy określone w tych metodykach nie zabraniają korzystania z narzędzi i procesów wykorzystywanych w pozostałych metodykach zwinnych. Co więcej, łączenie aspektów kilku metodyk zwinnych w pracy nad produktem jest bardzo często stosowaną praktyką.

Niniejsza praca obrazuje, które aspekty metodyk zwinnych zostały wdrożone do organizacji, jaką jest Studenckie Koło Naukowe Informatyków „KOD" oraz to, jak wpłynęło to na realizacje projektów prowadzonych przez Koło. W celu weryfikacji wpływu wdrożenia wykorzystano opisy słowne liderów projektów, wykresy obrazujące ilość włożonej pracy w repozytoriach Github organizacji [2] oraz ogólne opinie członków Koła Naukowego.

1. MANIFEST AGILE

Każda z metodyk zwinnych wymienionych wcześniej jest oparta lub ma wiele wspólnego z zasadami wynikającymi z „Agile Manifesto" [3]. Dokument ten został stworzony w 2001 roku, w ośrodku wypoczynkowym Snowbird w USA przez grono siedemnastu specjalistów pracujących w wielu firmach z mniej lub bardziej skomplikowanymi projektami. Manifest Agile jest zbiorem wartości, podzielonych na dwie kategorie: bardziej cenionych aspektów w pracy z projektami oraz tych mniej ważnych, lecz również wartościowych [4].

Na pierwszym miejscu postawione zostały cztery wartości:

- ludzie i interakcje,
- działające oprogramowanie,

- współpraca z klientem,
- reagowanie na zmiany.

Do mniej istotnych wartości zaliczono:

- procesy i narzędzia,
- szczegółową dokumentację,
- negocjację umów,
- realizacja założonego planu.

Zasady te pokazują, że w metodykach zwinnych stawia się na pierwszym miejscu to, co przynosi największe korzyści produktowi, a co za tym idzie również klientowi. Dokumentacja w wielu przypadkach nie jest dla klienta czymś ważnym. Klient większą uwagę przykłada do tego, jak wygląda produkt, czy jest funkcjonalny i czy spełnia wszystkie jego wymagania, a nie to czy produkt opisany jest bardzo szczegółową dokumentacją oraz jakimi narzędziami był wytwarzany.

2. SCRUM

Scrum jest to metodyka, której początek dał artykuł „The New New Product Development Game" napisany przez H. Takeuchi'ego oraz I. Nanoka w 1986 roku. Temat ten kilka lat później został przedstawiony także przez Jeffa Sutherlanda oraz Kena Schwabera w 1995 roku w artykule zatytułowanym „the scrum framework". Każda z metodyk zwinnych ma swoje narzędzia do organizacji pracy. W Scrumie głównym narzędziem jest „Rejestr Produktu", będący listą wszystkich aktualnie znanych funkcjonalności i wymagań wypracowanych we współpracy zespołu Scrum z interesariuszami [5]. Równie ważnym aspektem są spotkania zespołu. W Scrum możemy wyróżnić cztery rodzaje spotkań, są nimi:

- sprint planning,
- daily sprint,
- sprint review,
- sprint retrospective.

Każde z tych spotkań ma na celu realizację głównych filarów Scrum, jakimi są Inspekcja, Adaptacja oraz Transparentność [6]. Wykorzystywane są one do poznawania przez zespół wszystkich aspektów projektu, w tym przeszkód (Inspekcja). Gdy zespół odkryje pewne przeszkody, w ramach Adaptacji stara się przedyskutować plan ich usunięcia oraz dostosowania się do aktualnej sytuacji w projekcie. Ostatnim punktem jest Transparentność, gdzie najważniejszym aspektem jest, aby wszystkie elementy produktu, nad którymi pracujemy, zostały przejrzyście opisane. Zapobiega to generowaniu niezrozumienia oraz szybszemu wykrywaniu problemów występujących w projekcie.

3. KANBAN

Jest to metodyka opracowana w latach pięćdziesiątych, wywodząca się z Japonii. Twórcą Kanbana jest Taiichi Ohno, ówczesny prezes firmy Toyota. Stworzył on bardzo prosty, a zarazem wysoce skuteczny system planowania pracy, stosowany w fabrykach lidera w branży motoryzacyjnej. Polegał on na stworzeniu tablicy, na której zawieszone zostały kartki opisujące daną czynność oraz dodatkowe notatki określające potrzebne materiały i inne ważne dla procesu uwagi. Pracownicy wykorzystujący taką tablicę w trakcie przystąpienia do pracy przenosili odpowiednią notatkę z miejsca „To do" na pole określone jako „In progress". Po ukończeniu danej pracy notatka przesuwana była do kolumny „Done". Głównym założeniem Kanbana było to, iż ilość notatek przeniesionych do kolumny „In Progress" nie mogła przekraczać ilości dostępnych stanowisk do wykonywania danej czynności. Zapobiegało to zbędnym przestojom, a także ograniczało niepotrzebne składowanie materiałów do produkcji, które nie zostałyby natychmiast wykorzystywane.

4. PRZEGLĄD EFEKTYWNOŚCI KOŁA NAUKOWEGO „KOD" PRZED WDROŻENIEM METODYK ZWINNYCH

Badania prowadzone od października 2019 do listopada 2020 roku pokazały, że w omawianym Kole Naukowym efektywność zespołów projektowych znajdowała się na dość niskim poziomie. Pieczę nad działalnością Koła Naukowego sprawował Prezes Koła wraz z trzyosobowym Zarządem. Głównymi aktywnościami członków Koła był czynny udziału w konkursach typu Game Jam oraz Hackathon, a także wyjazdy na niektóre eventy. Dodatkowo prowadzone były projekty mające na celu tworzenie gier komputerowych, autorskiego systemu operacyjnego, a także wielu aplikacji mobilnych oraz stron WWW. Łączna ilość prowadzonych projektów wynosiła dwanaście. Liczba członków Koła określana była na około 140 osób oraz około 15 członków honorowych. Liczba ukończonych projektów w czasie przeprowadzanego badania nie była obiecująca. Około 66% projektów posiadało status „projektów zawieszonych", zaś prace nad pozostałą częścią projektów były prowadzone dosyć mozolnie. W trakcie transformacji organizacji do metodyk zwinnych na spotkaniu Zarządu zdecydowano się po konsultacjach z pozostałymi członkami Koła Naukowego usunąć część projektów, które nie wykazywały żadnych perspektyw rozwojowych. Praca w projektach polegała głównie na wykonywaniu zadań bez jakiegokolwiek planu. Nie było wyraźnych podziałów na grupy projektowe pracujące nad daną aplikacją. Powstający tym sposobem kod oprogramowania pozostawiał wiele do życzenia, gdyż trudniej było znaleźć twórcę niedziałającej funkcjonalności.

5. PLAN WDROŻENIA METODYK ZWINNYCH ORAZ RESTRUKTURYZACJI KOŁA

W okresie między listopadem a grudniem 2020 roku Zarząd Koła Naukowego wraz z częścią członków Koła opracował plan wdrożenia metodyk zwinnych oraz ogólnej restrukturyzacji Koła. Pierwszym poruszonym aspektem była hierarchia organizacyjna Koła. Poprzednio skupiała się ona głównie na czterech osobach zarządzających. Teraz dodane zostały role liderów projektu dla osób odpowiedzialnych za zarządzanie poszczególnymi projektami. Z uwagi na dodatkowe zaangażowanie wynikające z takiej roli liderom narzucono, iż mogą pełnić tę rolę tylko w jednym projekcie. Działanie to rozszerzyło grono ludzi odpowiedzialnych za projekty do 16 osób (4 zarządzających oraz 12 projekt liderów). Liderzy projektów pełnią obecnie rolę opisaną w Scrum Guide jako Project Owner. Członkowie Zarządu sprawują rolę zwaną Scrum Master, zajmując się szkoleniem członków Koła w tematyce metodyk zwinnych oraz pomagając usuwać przeszkody powstające w trakcie pracy nad projektami.

Kolejną zmianą w działaniach Koła było stworzenie tablic Kanban dla każdego z projektów oraz osobnej tablicy dla członków Zarządu. Tablice te są podzielone na kolumny według uznania każdego z zespołów projektowych.

Następnym punktem działań w kierunku wprowadzenia metodyk zwinnych było wprowadzenie cyklicznych spotkań projektowych ustalonych z poszczególnymi zespołami. Spotkania te odbywają się co tydzień dla każdego z projektów. Podczas spotkań, ze względu na ograniczony czas członków Koła wynikający z innych obowiązków, cztery spotkania Scrum okrojono do jednego, co najmniej godzinnego spotkania skupiającego się na wszystkich aspektach spotkań Scrum jednocześnie. Pierwszą część spotkań stanowią podsumowania pracy wykonanej od poprzedniego spotkania w okresie czasu nazwanym Sprintem. Analizowane są wszystkie aspekty wykonanej pracy, następnie zespół omawia, co potrzebne jest do wykonania pracy w następnym Sprincie. Informacje o brakach są przekazywane do członków Zarządu, którzy podejmują wszystkie możliwe działania potrzebne, aby powstałą przeszkodę usunąć. Kolejne spotkanie kończy się planowaniem pracy na przyszłe Sprinty oraz chwilą integracji zespołu w celu umocnienia relacji i atmosfery panującej w Kole.

Podczas prac związanych z wdrożeniem metodyk zwinnych do działań Koła jego członkowie zauważyli problem stosunkowo długo trwających zebrań obejmujących wszystkich członków Koła. Zebrania te odbywają się co dwa tygodnie w celu podsumowania całokształtu działań Koła, ogólnej integracji oraz omówienia spraw ważnych dla całej organizacji. Po dyskusji podczas jednego z takich spotkań uznano, że należy wprowadzić agendę spotkań Koła. Działanie to zostało powierzone członkom Zarządu.

6. EFEKTY WDROŻENIA METODYK ZWINNYCH

Efekt prac Zarządu w kwestii wdrożenia metodyk zwinnych jest widoczny gołym okiem, począwszy od frekwencji na spotkaniach Koła, ilości nowo rekrutowanych członków, aż po dużo lepsze zaangażowanie w pracę i płynniejszy rozwój tworzonych aplikacji, gier komputerowych oraz pozostałych projektów.

Agenda spotkań pozwoliła na skrócenie czasu trwania spotkań. Przed zmianą oficjalna część spotkania trwała zwykle od dwóch do trzech godzin. Zastosowanie metodyk zwinnych, dzięki którym wykryto problem i postarano się go rozwiązać, skróciło ten czas o ponad 80%. Jednocześnie zachęciło to studentów do częstszego oraz bardziej aktywnego uczestnictwa w spotkaniach.

Kolejnym aspektem, na który wpłynęło wdrożenie zwinnych metodyk, jest zaangażowanie w projekty oraz płynność ich rozwoju. Najlepszym przykładem będą projekty mające na celu stworzenie gier komputerowych, jak projekty Roguelike oraz Tower Defence. Pierwszy z nich rozpoczął swoje istnienie z dniem wdrożenia metodyk zwinnych. Poskutkowało to ukończeniem pierwszej grywalnej wersji w przeciągu czterech miesięcy, co, porównując do postępów prac w poprzednich projektach sprzed wdrożenia metodyk zwinnych, byłoby niemożliwe. Projekt Tower Defence uruchomiony został w październiku 2020 roku. Na początku swojego istnienia nie używano w nim żadnej z metodyk zwinnych, a praca wykonywana była chaotycznie. W momencie wdrażania nowych metodyk, Scrum Master zauważył, iż do tej pory nikt z zespołu projektowego nie znał pełnego zarysu projektu, nad którym pracuje. Wdrożenie doprowadziło do częstszych spotkań zespołu oraz do stworzenia zarysu projektu zaakceptowanego przez całość zespołu projektowego. Projekt ten z racji dużej złożoności nie osiągnął jeszcze grywalnej wersji, lecz stworzenie jej to nieodległa przyszłość. Efekty wdrożenia metodyk zwinnych pokazane na podstawie tych dwóch projektów obrazują pracę w pozostałych szesnastu płynnie rozwijanych aktualnie projektów.

Wartością dodaną dla członków Koła są również częstsze szkolenia oraz webinary prowadzone na potrzeby rozwoju projektów. Bez metodyk zwinnych nie byłoby to możliwe, gdyż członkowie nie wyrażali takich potrzeb. Teraz, gdy metodyki te są wprowadzone, członkowie Koła coraz częściej zgłaszają tematykę spotkań, która jest im potrzebna do rozwoju oraz pomocna w pracach nad projektami. Co więcej, w cykl corocznych szkoleń Koła wprowadzone zostało szkolenie z metodyk zwinnych. Pozwala to nowym członkom Koła poznać system, w jakim pracuje się w Studenckim Kole Naukowym Informatyków „KOD", a osobom już znającym te zasady pozwala na przenoszenie metodyk zwinnych do codziennego życia w ramach poszerzania wiedzy na przeprowadzanych warsztatach.

Ostatnim bardzo ważnym aspektem jest poprawa funkcjonowania Zarządu Koła. Wraz z wdrożeniem tablicy Kanban oraz cyklicznych spotkań Zarządu udoskonalone zostały metody podziału obowiązków między członków Zarządu. Poskutkowało to nawiązaniem przez Koło Naukowe współpracy z wieloma firmami

oraz kluczowymi osobami z branży, co otworzyło wiele dróg do kolejnych poziomów rozwoju Koła. Jednym z przykładów pokazujących rozwój organizacji jest przyjęcie do Koła specjalisty do spraw marketingu, który odpowiedzialny jest za promocję Koła oraz prowadzenie strony Koła Naukowego na portalu Facebook. Kolejnymi przykładami mogą być rozwój cyklu szkoleń, zwiększone zainteresowanie otoczenia społeczno-gospodarczego współpracą z Kołem Naukowym, a także lepsza organizacja pracy, co stworzyło miejsce na dodatkowe aktywności Koła Naukowego.

7. PODSUMOWANIE

Wdrożenie nowego sposobu działania opartego na metodykach zwinnych wprowadziło początkowo pewien chaos związany z transformacją. Jednak wraz z upływem czasu, w momencie gdy wszystkie z wdrażanych praktyk zostały odpowiednio opanowane i przyjęte przez większość członków Koła, zauważony został duży rozwój Studenckiego Koła Naukowego Informatyków „KOD". Metodyki zwinne pozwoliły oszczędzić dużo czasu przeznaczanego poprzednio na wzajemne doinformowywanie się członków Koła lub pracę, która po pewnym czasie okazywała się bezużyteczna. Aktualnie w opisywanej organizacji widoczny jest wzrost w wielu aspektach pracy i działalności. Metodyki zwinne przeniesione zostały również przez wielu członków Koła do ich osobistych aktywności, co umożliwiło im lepszą organizację swojej pracy, a co za tym idzie wygospodarowanie wolnego czasu do wykorzystania na rozwój osobisty oraz rozrywkę. Warto pamiętać, że stosowanie metodyk zwinnych nie zawsze daje zamierzone efekty, gdyż są one tylko wytycznymi. To, jaki wpływ będą miały na daną organizację, zależy od sposobu ich wdrożenia oraz zaangażowania zespołu. Przykład Studenckiego Koła Naukowego Informatyków „KOD" można uznać za bardzo efektywny, w którym proces przeprowadzony został sprawnie oraz dał istotne korzyści całej wspólnocie Koła oraz jego otoczeniu.

LITERATURA

1. Schwaber K., Sutherland J., *The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game*, Listopad 2020.
2. https://github.com/skni-kod (dostęp: 28.05.2021).
3. https://agilemanifesto.org (dostęp: 28.05.2021).
4. https://agilemanifesto.org/principles.html (dostęp: 28.05.2021).
5. Sutherland J., Sutherland J.J., *SCRUM. Czyli jak robić dwa razy więcej dwa razy szybciej*, Wydawnictwo Naukowe PWN, Warszawa 2015.
6. Kaczor K., *SCRUM I NIE TYLKO. Teoria i Praktyka w metodach Agile*, Wydawnictwo Naukowe PWN, Warszawa 2016.

KOŁO

NAUKOWE

ELEKTRONIKI

I TECHNOLOGII

INFORMACYJNYCH

Rafał NAZARKO, Piotr STOREK, Karol SIWIEC

dr inż. Bartosz PAWŁOWICZ
opiekun naukowy

STEROWANIE ZESTAWEM LEGO MINDSTORMS ZA POMOCĄ KOMPUTERA RASPBERRY PI 4 PRZY UŻYCIU JĘZYKA PYTHON

Artykuł opisuje zasadę działania komunikacji mikrokomputera Raspberry Pi 4 z jednostką sterującą zestawu LEGO Mindstorms NXT lub EV3 przy użyciu bardzo popularnego języka programowania Python.
Słowa kluczowe: układ elektroniczny, programowanie, zdalne sterowanie.

WPROWADZENIE

Głównym celem przyświecającym stworzeniu oprogramowania była potrzeba zwiększenia elastyczności sterowania w robotach tworzonych przez członków koła naukowego. Podstawowe narzędzie do oprogramowania, dostarczone przez firmę LEGO, ma określony zestaw poleceń, który często okazuje się niewystarczający przy bardziej wymagających projektach.

Użycie zewnętrznego kontrolera pozwala zwiększyć moc obliczeniową, którą dysponuje tworzona maszyna (a co za tym idzie, również prędkość działania), wykorzystać złożone algorytmy lub rozszerzyć komunikację o dodatkowe porty USB, połączenie internetowe lub urządzenia wejścia/wyjścia.

1. KOMUNIKACJA MIKROKOMPUTERA RASPBERRY PI 4 Z JEDNOSTKĄ STERUJĄCĄ LEGO MINDSTORMS

Jednostka sterująca LEGO Mindstorms zapewnia trzy rodzaje komunikacji: Bluetooth, Wi-Fi i USB. Wszystkie z nich pozwalają tworzyć aplikacje działające na dowolnym komputerze i komunikować się z jednostką sterującą LEGO Mindstorms. Aby zapewnić maksymalną prędkość transmisji danych, powinno się stosować połączenie za pomocą kabla USB.

Jednostka sterująca LEGO Mindstorms, podobnie jak inne współczesne komputery, grupuje osiem bitów w jeden bajt i adresuje swoją pamięć po jednym bajcie. Sąsiadujące ze sobą bajty graficznie oddzielane są dwukropkami „:” lub pionowymi kreskami „|”. Z racji tego, że zapis binarny jest bardzo długi i zazwyczaj nieczytelny, przyjęło się stosować notację szesnastkową. Zapis w tej postaci jest zwarty, a konwersja do i z systemu binarnego prosta, ponieważ jedna cyfra szesnastkowa oznacza pół bajtu.

Aby to dobrze zrozumieć, poniżej został przedstawiony przykład komendy, która pinguje jednostkę sterującą (zmusza ją do odpowiedzi):

```
0x|06:00|2A:00|00|00:00|01|
```

Listing 1. Konstrukcja komendy ping

Jak widać, komenda składa się ośmiu bajtów i znaku określającego liczbę szesnastkową *0x*. Dwa pierwsze bajty są częścią protokołów komunikacyjnych, którymi mogą być – jak wspomniano wcześniej – Bluetooth, USB lub Wi-Fi. Kolejne dwa bajty są licznikiem komunikatów, które pozwolą dopasować polecenie bezpośrednie oraz odpowiedź na nie. Piąty bajt określa, czy nadawca tego polecenia oczekuje na odpowiedź, czy też nie (|00| oznacza oczekiwanie na odpowiedź, zaś |80| jej nie potrzebuje). Bajty szósty i siódmy stanowią nagłówek komendy, który jest kombinacją dwóch liczb, które określają rozmiary polecenia bezpośredniego. Od bajtu ósmego rozpoczynają się operacje, czyli dokładne informacje o działaniu, jakich oczekuje od jednostki sterującej mikrokomputer.

2. PISANIE I WYSYŁANIE BEZPOŚREDNICH POLECEŃ

2.1. Wysyłanie komend

Aby otworzyć komunikację z jednostką sterującą z poziomu kodu Python na mikrokomputerze Raspberry Pi 4, należy zaimportować bibliotekę *ev3_dc*, która dostarcza pakiet narzędzi ułatwiających rozwiązanie podstawowych zagadnień. Konfigurację należy rozpocząć od zdefiniowania obiektu hosta, którym jest jednostka sterująca. W tym celu można użyć poniższego fragmentu kodu:

```
#!/usr/bin/env python3

import ev3_dc as ev3

my_ev3 = ev3.EV3(
    protocol=ev3.USB,
    host='00:16:53:5F:19:24'
)
my_ev3.verbosity = 1
my_ev3.sync_mode = ev3.SYNC
```

Listing 2. Konfiguracja obiektu hosta

Argumenty podane pod nazwą *protocol* oraz *host* definiują odpowiednio, który z trzech protokołów (wspomnianych wcześniej) będzie używany do transmisji danych oraz adres Mac jednostki sterującej (te informację można znaleźć w ustawieniach jednostki sterującej). Ustawienie *my_ev3.verbosity* na wartość 1 pozwala na podgląd komunikacji pomiędzy mikrokomputerem a jednostką sterującą. Gdy powyższa operacja zostanie wykonana, można przejść bezpośrednio do wysyłania komend. Konstrukcja pojedynczej operacji powinna mieć postać jak poniżej:

```
cmd = b ''. join ((
    ev3.opSound, # Moduł
    ev3.TONE, # Czynność
    ev3.LCX (440), # Częstotliwość
    ev3.LCX (1000), # Czas trwania
))
my_ev3.send_direct_cmd(cmd)
```

Listing 3. Konstrukcja komendy odtwarzającej dźwięk

W powyższym przykładzie cała komenda jest przypisana do zmiennej *cmd*. Zostaje ona wywołana przy pomocy funkcji *send_direct_cmd(cmd)* znajdującej się w instancji obiektu jednostki sterującej. Pierwszą linią zawierającą się w funkcji *join* kreującą daną komendę, powinna być informacja o module, do którego użytkownik chce się zwrócić. W tym przypadku jest to moduł dźwiękowy, dlatego użyto wartości *ev3.opSound.* Kolejna linia definiuje, w jaki sposób ma być wykorzystany wybrany moduł. Użyte w powyższym przykładzie *ev3.TONE* zainicjuje granie dźwięku o częstotliwości 440 Hz przez 1000 ms. Odpowiadają za to linie trzecia oraz czwarta. Moduł dźwiękowy w jednostkach sterujących LEGO Mindstorms nie jest rozbudowany, posiada bowiem jeszcze tylko jeden przypadek użycia, w którym można odegrać muzykę zapisaną na karcie SD. Istnieją jednak takie moduły, które oferują szeroki wachlarz możliwości użycia.

Przed rozpoczęciem korzystania z przygotowanej biblioteki, warto poznać metody statyczne, jakie zostały w niej wykorzystane. W głównej mierze konwertują one liczby lub znaki zrozumiałe dla ludzi na język czytelny dla jednostki sterującej LEGO Mindsotrms. Składają się na nie:

- *LCX(int)* – konwertuje liczbę całkowitą na polecenie bezpośrednie z bajtem identyfikacyjnym. W zależności od wartości przekazanej w argumencie tej funkcji będzie ciągiem bajtów o długości jednego (LC0), dwóch (LC1), trzech (LC2) lub pięciu bajtów (LC4).
- *LCS(string)* – dodaje do podanego w argumencie funkcji łańcucha znaków bajt identyfikacyjny oraz terminator początkowy oraz końcowy ciągu bajtów.
- *LVX(int)* – konwertuje adres pamięci lokalnej na format zgodny z poleceniem bezpośrednim z bajtem identyfikacyjnym. W zależności od wartości

przekazanej w argumencie tej funkcji będzie ciągiem bajtów o długości jednego (LV0), dwóch (LV1), trzech (LV2) lub pięciu bajtów (LV4).

- *GVX(int)* – konwertuje adres pamięci globalnej na format zgodny z poleceniem bezpośrednim z bajtem identyfikacyjnym. W zależności od wartości przekazanej w argumencie tej funkcji będzie ciągiem bajtów o długości jednego (GV0), dwóch (GV1), trzech (GV2) lub pięciu bajtów (GV4).

2.2. Zarządzanie czujnikami

Podstawowym źródłem informacji dla każdej autonomicznej maszyny jest odczyt z jej czujników. Nie jest inaczej w przypadku tych tworzonych przy użyciu zestawu LEGO Mindstorms. Choć w tym przypadku jest to dużo prostsze niż w przypadku wysoko zaawansowanych technicznie rozwiązań, trzeba zachować pewne standardy obsługi tych czujników. Z tej potrzeby powstał szablon, który odpowiada za spójność przesyłanych komend. Jej szkielet można zobaczyć na poniższym listingu:

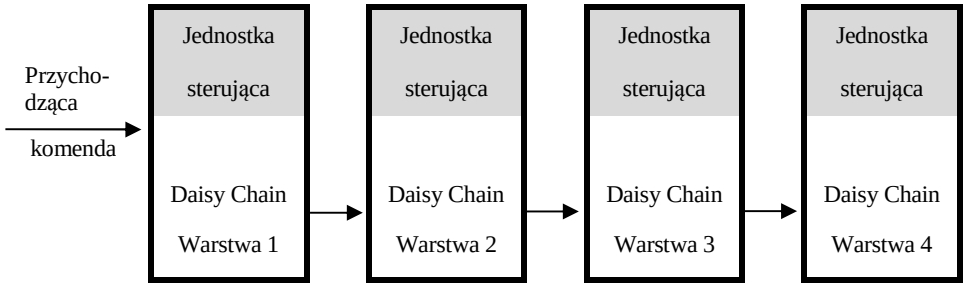
```
def ConstructCommand(self, daisyChainLayer, port, sensorType,
sensorMode, values = 1, values1 = 0):
    return b''.join((
        ev3.opInput_Device,       # Moduł
        ev3.READY_SI,             # Czynność
        ev3.LCX(daisyChainLayer), # Warstwa Daisy Chain
        ev3.LCX(port),            # Numer portu
        ev3.LCX(sensorType),      # Typ czujnika
        ev3.LCX(sensorMode),      # Tryb czujnika
        ev3.LCX(values),          # Liczba zwracanych wartości
        ev3.GVX(values1),         # Dodatkowe wartości
))
```

Listing 4. Konstrukcja szablonu komend

Używanie tego szablonu jest banalnie proste. Wystarczy podać odpowiednie wartości jako argumenty tej funkcji, a komenda zostanie zwrócona jako ciąg bajtów, który można bezpośrednio wysłać do jednostki sterującej LEGO Mindstorms. Poszczególne parametry i ich znaczenie zostały opisane poniżej:

- Moduł – parametr niepodlegający zmianie, ustawiony na *ev3.opInput_Device*, który odpowiada za komunikację z urządzeniami peryferyjnymi takimi jak czujniki lub serwomotory.
- Czynność – statyczna wartość, która odpowiada za typ zwracanych danych. Może być ustawiona tak jak w szablonie na *READY_SI*, która odczytuje wartości z czujnika jako liczbę zmiennoprzecinkową lub *READY_PCT*, wtedy zwracane wartości są w postaci procentowej (0-100).
- Warstwa Daisy Chain (ang. *Daisy Chain Layer*) – połączenie Daisy Chain oznacza szereg spiętych ze sobą szeregowo jednostek sterujących LEGO Mindstorms, w sposób, który umożliwia im swobodną komunikację.

Układ działa w architekturze Master-Slave. Każda jednostka jest sterowana za pomocą jej poprzednika. Pozycja w tym łańcuchu oznacza warstwę Daisy Chain. Można połączyć do czterech urządzeń w ramach jednego łańcucha. Aby rozkazać jakąś czynność konkretnej jednostce, wystarczy w kreatorze komend podać numer warstwy Daisy Chain jako argument *daisyChainLayer*.



Rys. 1. Połączenie Daisy Chain
Źródło: opracowanie własne.

- Numer portu – jednostka sterująca posiada osiem fizycznych portów – cztery przeznaczone są dla silników, a cztery dla czujników. Porty dla silników oznaczone są czterema pierwszymi literami alfabetu łacińskiego A, B, C oraz D, zaś dla każdego portu dla czujników przyporządkowana jest liczba z zakresu 1-4. Aby odczytać wartość z wybranego czujnika, należy podać numer portu, do którego jest podpięty, jako argument *port*.
- Typ czujnika – czujniki w zestawach LEGO Mindstorms mają kilka różnych typów. Mogą to być czujniki dźwięku, natężenia światła, koloru, ultradźwięków, temperatury, dotyku, fal podczerwonych oraz żyroskop. Każdemu z powyższych typów, odpowiada wartość numeryczna, którą można znaleźć w tab. 1. Aby określić typ czujnika, należy odczytaną z tabeli wartość wprowadzić jako argument *sensorType* funkcji *ConstructCommand*.
- Tryb czujnika – pojedynczy czujnik zazwyczaj oferuje kilka funkcjonalności. Przykładowo, czujnik koloru może pracować w trybie pomiaru światła odbitego, natężenia światła, koloru RGB, światła zielonego, światła czerwonego lub światła niebieskiego. Każdemu z powyższych typów odpowiada wartość numeryczna, którą można znaleźć w tab. 1. Aby określić tryb, należy odczytaną z tabeli wartość wprowadzić jako argument *sensorMode* funkcji *ConstructCommand*.
- Liczba zwracanych wartości – aby określić, jak wiele pomiarów powinna przeprowadzić wybrana jednostka sterująca na danym czujniku, należy

podać wartość numeryczną jako argument *values* w kreatorze komend. Domyślnie ta liczba jest ustawiona na 1.

- Dodatkowe wartości – niektóre czujniki mogą oczekiwać podania dodatkowych danych. Argument stanowi jedynie otwartą furtkę dla deweloperów, nie jest on jednak wymagany i domyślnie jest ustawiony na 0.

Tabela 1. Typy oraz tryby sensorów wraz z opisami funkcji.

Typ	Tryb	Opis
1	*0*	*NXT-Touch*
	1	*NXT-Bump*
2	*0*	*NXT-Light-Reflected*
	1	*NXT-Light-Ambient*
3	*0*	*NXT-Sound-DB*
	1	*NXT-Sound-DBA*
4	*0*	*NXT-Color-Reflected*
	1	*NXT-Color-Ambient*
	2	*NXT-Color-Color*
	3	*NXT-Color-Green*
	4	*NXT-Color-Blue*
	5	*NXT-Color-Raw*
5	*0*	*NXT-Ultrasonic-Cm*
	1	*NXT-Ultrasonic-Inch*
6	*0*	*NXT-Temperature-C*
	1	*NXT-Temperature-F*
7	*0*	*EV3-Large-Motor-Degree*
	1	*EV3-Large-Motor-Rotation*
	2	*EV3-Large-Motor-Power*
8	*0*	*EV3-Medium-Motor-Degree*
	1	*EV3-Medium-Motor-Rotation*
	2	*EV3-Medium-Motor-Power*
9		*Free*
…	…	…
13		*Free*
14	*0*	*Output for 3th party devices*
15		*Free*
16	*0*	*EV3-Touch*
	1	*EV3-Bump*
17		*Free*
…	…	…
20		*Free*
21	*0*	*Test purpose*
22		*Free*
…	…	…
27		*Free*
28	*0*	*3th party input, 1 mode, Scale 0 - 4095*
	1	*3th party input, 2 mode, Scale 0 - 5000*

	2	*3th party input, 3 mode, Scale 0 - 10000*
	3	*3th party input, 4 mode, Scale 0 - 20000*
29	*0*	*EV3-Color-Reflected*
	1	*EV3-Color-Ambient*
	2	*EV3-Color-Color*
	3	*EV3-Color-Reflected-Raw*
	4	*EV3-Color-RGB-Raw*
	5	*EV3-Color-Calibration*
30	*0*	*EV3-Ultrasonic-Cm*
	1	*EV3-Ultrasonic-Inch*
	2	*EV3-Ultrasonic-Listen*
	3	*EV3-Ultrasonic-SI-Cm*
	4	*EV3-Ultrasonic-SI-Inch*
	5	*EV3-Ultrasonic-DC-Cm*
	6	*EV3-Ultrasonic-DC-Inch*
31		*Free*
32	*0*	*EV3-Gyro-Angle*
	1	*EV3-Gyro-Rate*
	2	*EV3-Gyro-Fast*
	3	*EV3-Gyro-Rate & Angle*
	4	*EV3-Gyro-Calibration*
33	*0*	*EV3-IR-Proximity*
	1	*EV3-IR-Seeker*
	2	*EV3-IR-Remote*
	3	*EV3-IR-Remote-Advanced*
	4	*Not utilized*
	5	*EV3-IR-Calibration*
34		*Free*
…	…	…
98		*Free*
99	*0*	*Energy-Meter-Voltage-In*
	1	*Energy-Meter-Amps-In*
	2	*Energy-Meter-Voltage-Out*
	3	*Energy-Meter-Amps-Out*
	4	*Energy-Meter-Joule*
	5	*Energy-Meter-Watts-In*
	6	*Energy-Meter-Watts-Out*
	7	*Energy-Meter-All*
100	*0*	*IIC-Byte*
	1	*IIC-WORD*
101	*0*	*NXT-Test*
102		*Free*
…	…	…
120		*Free*

Źródło: LEGO MINDSTORMS EV3 Firmware Developer Kit

2.3. Korzystanie z biblioteki

Znając mechanizmy odpowiadające za wykonywanie poleceń w jednostce sterującej, można płynnie przejść do korzystania ze stworzonej biblioteki.

Pierwszym krokiem, który należy postawić, jest połączenie jednostki sterującej z urządzeniem wysyłającym komendy. Jak wspomniano w drugim rozdziale tego artykułu, można tę operację przeprowadzić na trzy sposoby. Na potrzeby tego artykułu użyto kabla USB, łącząc jednostkę sterującą LEGO Mindstorms z mikrokomputerem Raspberry Pi 4. W przypadku połączenia typu Daisy Chain, wystarczy podać adres MAC pierwszej jednostki sterującej, czyli tej bezpośrednio połączonej w urządzeniem wysyłającym komendy.

Kolejnym krokiem jest utworzenie pliku w języku Python, który będzie plikiem odpowiadającym za rozpoczęcie komunikacji. Wstępną postać tego pliku pokazuje kod na *Listingu 2*. W tym momencie powinno się przetestować połączenie, aby wykluczyć przyczyny ewentualnie powstałych błędów w trakcie rozwijania programu.

Ostatni krok to utworzenie instancji czujników, które użytkownik chce obsługiwać podczas działania swojego programu. Wystarczy zaimportować bibliotekę *sensors*, odszukać w niej podklasę wybranego czujnika odpowiadającą typowi jednostki sterującej (*NXT* lub *EV3*), a następnie utworzyć jego instancję obiektu. Dla przykładu, poniższy skrypt tworzy obiekt klasy czujnika ultradźwiękowego, po czym wypisuje na konsolę zmierzoną przez niego odległość podaną w centymetrach:

```
ultrasonicSensor1 = sensors.EV3Sensors.UltrasonicSensor(my_ev3,
4, 0)
print(ultrasonicSensor1.GetDistanceInCm())
```

Listing 5. Kod tworzący instancję obiektu czujnika ultradźwiękowego oraz pomiar odległości

Fragment kodu odpowiadający za wywołanie funkcji odczytania odległości z czujnika ultradźwiękowego zamieszczony został poniżej. Taką strukturę reprezentuje każda z dostępnych w bibliotece podklas. Klasy EV3Sensors oraz NXTSensors są jedynie kontenerami dla czujników poszczególnych typów, dlatego nie należy tworzyć instancji obiektów tych klas.

```
import ev3_dc as ev3
import struci



class EV3Sensors(object):
    def ConstructCommand(self, daisyChainLayer, port, sensorType,
sensorMode,values = 1, values1 = 0):
```

```
        return b''.join((
            ev3.opInput_Device,       # Moduł
            ev3.READY_SI,             # Czynność
            ev3.LCX(daisyChainLayer), # Warstwa Daisy Chain
            ev3.LCX(port),            # Numer portu
            ev3.LCX(sensorType),      # Typ czujnika
            ev3.LCX(sensorMode),      # Tryb czujnika
            ev3.LCX(values),          # Liczba zwracanych warto-
ści
            ev3.GVX(values1),         # Dodatkowe wartości
       ))

    class UltrasonicSensor(object):

        def __init__(self, ev3Instance, port, daisyChainLayer):
            self.ev3Instance = ev3Instance
            self.port = port - 1
            self.daisyChainLayer = daisyChainLayer

        def GetDistanceInCm(self):
            command = EV3Sensors.ConstructCommand(self, self.dai-
syChainLayer, self.port, 30, 0)
            reply = self.ev3Instance.send_direct_cmd(command,
global_mem=4)
            return struct.unpack('<f', reply)[0]

        def GetDistanceInInch(self):
            command = EV3Sensors.ConstructCommand(self, self.dai-
syChainLayer, self.port, 30, 1)
            reply = self.ev3Instance.send_direct_cmd(command,
global_mem=4)
            return struct.unpack('<f', reply)[0]

        def Listen(self):
            command = EV3Sensors.ConstructCommand(self, self.dai-
syChainLayer, self.port, 30, 2)
            reply = self.ev3Instance.send_direct_cmd(command,
global_mem=4)
            return struct.unpack('<f', reply)[0]

        def GetDistanceInSICm(self):
            command = EV3Sensors.ConstructCommand(self, self.dai-
syChainLayer, self.port, 30, 3)
            reply = self.ev3Instance.send_direct_cmd(command,
global_mem=4)
            return struct.unpack('<f', reply)[0]

        def GetDistanceInSIInch(self):
            command = EV3Sensors.ConstructCommand(self, self.dai-
syChainLayer, self.port, 30, 4)
            reply = self.ev3Instance.send_direct_cmd(command,
global_mem=4)
            return struct.unpack('<f', reply)[0]
```

```
        def GetDistanceInDCCm(self):
            command = EV3Sensors.ConstructCommand(self, self.dai-
syChainLayer, self.port, 30, 5)
            reply = self.ev3Instance.send_direct_cmd(command,
global_mem=4)
            return struct.unpack('<f', reply)[0]

        def GetDistanceInDCInch(self):
            command = EV3Sensors.ConstructCommand(self, self.dai-
syChainLayer, self.port, 30, 6)
            reply = self.ev3Instance.send_direct_cmd(command,
global_mem=4)
            return struct.unpack('<f', reply)[0]
```

Listing 6. Klasa agregująca funkcje czujników EV3 oraz podklasa czujnika fal ultradźwiękowych

3. PODSUMOWANIE

Biblioteka pozwala na łatwe rozszerzenie możliwości oferowanych przez zestaw LEGO Mindstorms. Dodatkowym atutem jest rozwiązanie tego zagadnienia w powszechnie znanym i lubianym z swej prostoty języku Python, który znacząco zmniejsza próg wejścia dla osób dopiero rozpoczynających swoją przygodę z programowaniem. Głównym minusem obsługi jednostki sterującej w sposób opisany w tym artykule jest wydajność. Czas reakcji na podaną komendę lub oczekiwanie na odczyt z czujnika w złożonych programach okazuje się zbyt długi, aby móc go zastosować w zaawansowanych i precyzyjnych maszynach. Pomimo tej wady pozostaje świetną platformą do większości projektów.

Rys. 3. Kod QR zawierający adres do repozytorium projektu
Źródło: opracowanie własne.

LITERATURA

1. Ceder N., *Python szybko i prosto*, Wydanie III, Wydawnictwo Helion, 2019.

ŹRÓDŁA INTERNETOWE

2. Gaukel Ch., *EV3-DC Documentation*, https://ev3-dc.readthedocs.io/en/latest/intro.html (dostęp: 14.04.2021).
3. Gaukel Ch., *EV3 Direct Commands Blog*, http://ev3directcommands.blogspot.com/ (dostęp: 15.04.2021).
4. The LEGO Group, *LEGO® MINDSTORMS® EV3 Firmware Developer Kit*, https://www.lego.com/cdn/cs/set/assets/blt77bd61c3ac436ea3/LEGO_MINDSTORMS_EV3_Firmware_Developer_Kit.pdf (dostęp: 14.04.2021).

Piotr STOREK, Karol SIWIEC
Rafał NAZARKO, Filip BŁĄDZIŃSKI

dr inż. Bartosz PAWŁOWICZ
opiekun naukowy

BEZZAŁOGOWY STATEK POWIETRZNY TYPU QUADROCOPTER

Artykuł opisuje sposób wykonania i zasadę działania drona czterosilnikowego typu quadrocopter. Jest to najczęściej stosowany rodzaj statku powietrznego, z uwagi na jego stosunkowo dużą stabilność lotu, znajdujący zastosowanie w każdej gałęzi przemysłu, poprzez możliwość użycia owej konstrukcji na rynku usług fotograficznych i wideo, do wykorzystania bezzałogowca przez kartografów i rolników. Artykuł zawiera także szczegółowy opis konfiguracji i wdrożenie w oprogramowanie ArduPilot wraz z kontrolerem lotu Pixhawk 4 autopilot. Do konfiguracji oprogramowania zostało wybrane GUI (ang. *Graphical User Interface*) Mission Planner, który umożliwia także automatyczne planowanie lotów.
Słowa kluczowe: bezzałogowy statek powietrzny, dron, silniki BLDC, ArduPilot.

WPROWADZENIE

Zapotrzebowanie na zastosowanie bezzałogowych statków powietrznych wciąż rośnie, głównie przez wzgląd na zróżnicowane zastosowanie. W naszym przypadku postanowiliśmy wykorzystać najbardziej popularną konstrukcję, czyli ramę typu quadrocopter. Jest to wysokiej jakości, wykonana z włókna węglowego, rama Tarot IRON MAN 650 przeznaczona do zastosowań profesjonalnych. Popularne oznaczenie 650 pochodzi od rozstawu silników w odległości 650 mm od siebie. Docelowo konstrukcja ma udźwig wynoszący około 2400 g. Bezzałogowiec, dzięki zastosowaniu odpowiednich silników, może podnieść różnorodne czujniki pomiarowe, niewielkie, profesjonalne aparaty fotograficzne, do wykonywania specjalistycznych zdjęć oraz różnorodne elementy wykonawcze przykładowo specjalne uchwyty stworzone do podnoszenia i spuszczania stosunkowo lekkich elementów. W naszym projekcie zastosowaliśmy jeden z najlepszych kontrolerów lotu, oparty na otwartym oprogramowaniu, który oferuje dwa niezależne moduły IMU (ang. *Inertial Measuring Unit*), barometr, dwa niezależne żyroskopy, również dwa niezależne akcelerometry oraz obsługę kilku systemów

GPS (ang. *Global Positioning System*). Dzięki możliwości zastosowania aż 14 wyjść, przeznaczonych dla serwomechanizmów PWM, możemy kontrolować do ośmiu silników i sześciu urządzeń peryferyjnych. Pixhawk posiada interfejs CAN Bus, trzy wejścia analogowe i dwa wyjścia UART (od ang. *Universal Asynchronous Receiver – Transmitter*). Bezpośrednio zastosowaliśmy GPS Here 3 GNSS. Zastosowanie protokołu CAN zapewnia między innymi większą prędkość transmisji danych czy większą odporność na zakłócenia. W poprawnej nawigacji pomaga wbudowany magnetometr. Kontrola bezzałogowca może odbywać się na kilka różnych sposobów, nie jesteśmy ograniczeni jedynie do sterowania Transmiterem RC. Dodatkowo zastosowany tu został mikrokomputer Raspberry Pi połączony bezpośrednio z kontrolerem lotu po magistrali UART, w celu umożliwienia przyszłego rozwijania projektu. Ponadto zastosowaliśmy składane podwozie na dwóch niezależnych serwomechanizmach sprzężonych ze sobą, które daje nam możliwość automatycznego złożenia, w przypadku gdy wykorzystywane są dodatkowe zewnętrzne elementy wykonawcze.

1. BUDOWA BEZZAŁOGOWEGO STATKU POWIETRZNEGO TYPU QUADROCOPTER

Planowanie budowy quadrocoptera zostało podzielone na kilka części związanych z poszczególnymi etapami budowy. Zastosowane zostały cztery silniki od bardzo popularnego producenta na rynku dronów, mianowicie DJI e600 o maksymalnym poborze prądu wynoszącym 20 A i częstotliwości sygnału od 30 do 450 Hz. Bezpośrednio do nich zamontowane zostały oryginalne dwudziestoamperowe regulatory o częstotliwości zgodnej z parametrami silników wykorzystanych w naszej konstrukcji, bezpośrednim zasilaniu mogącym wynosić od 14,8 do 22,2 V (akumulator 4S do 6S) oraz minimalnej wydajności prądowej wynoszącej 40 C.

Do bezzałogowego statku powietrznego zostały wykorzystane silniki trójfazowe bezszczotkowe BLDC firmy DJI. Silniki te mogą poszczycić się ciągle rosnącą popularnością w przemyśle przez wzgląd na bardzo dobre parametry i stosunkowo długą żywotność. Charakteryzują się dokładną sterowalnością. Do najważniejszych zalet takich silników należą między innymi brak łuku elektrycznego, dzięki czemu silniki nie wydzielają iskier, duża sprawność sięgająca nawet 95%,względnie niskie natężenie hałasu, które pozwala na łatwiejsze dostosowanie do określonych warunków pracy silnikowej, kontrola prędkości obrotowej prawie niezależnie od momentu silnika, stosunkowo długa żywotność oraz trwałość, dzięki wyeliminowaniu z konstrukcji silnika szczotek.

Do dystrybucji napięcia zasilania poszczególnych elementów użytych do budowy bezzałogowego statku powietrznego wykorzystany został moduł zasilania Power Brick przeznaczony specjalnie do kontrolera PixHawk Radiolink. Zaawansowany kontroler lotu PixHawk Radiolink charakteryzuje się bardzo dużą odpornością na zakłócenia, posiada względnie dużą precyzję kompasu, co przekłada się

na większe bezpieczeństwo w trakcie lotu. Jest on kompatybilny ze sporą ilością systemów satelitarnych, takich jak przykładowo GPS (ang. *Global Positioning System*) stworzony przez Departament Obrony Stanów Zjednoczonych, GLONASS stworzony w Rosji, QZSS (ang. *Quasi-Zenith Satellite System*), BeiDou, co umożliwia mu połączenie się z dużą ilością satelitów w stosunkowo krótkim, wynoszącym kilka sekund, czasie. Kontroler ten jest także względnie precyzyjny, potrafi zmierzyć pozycję z dokładnością sięgającą nawet 50 cm, prędkość z precyzją wynoszącą aż 0,1m/s. Kontroler również bardzo dobrze sprawuje się nawet w pomieszczeniach zamkniętych. Parametry graniczne, jakie posiada ten kontroler, pozwalają nawet na osiągnięcie realnych, w przypadku bezzałogowych statków powietrznych, wartości takich jak maksymalna prędkość lotu, gwarantująca poprawne pomiary, wynosząca zawrotne 515 m/s bądź maksymalna wysokość, zapewniająca poprawny pomiar, AGL (ang. *Above Ground Level*) wynosząca 50 km.

Procesor główny, w który producent wyposażył wybrany przez nas kontroler lotu PixHawk Radiolink, jest 32 bitową jednostką STM (STM32F427 Cortex M4 z FPU) posiadającą 256 KB pamięci RAM, 168 MHz częstotliwości taktowania oraz 2 MB pamięci Flash.

Sensory, które posiada PixHawk Radiolink, to 16 bitowy żyroskop (ST Micro L3GD20H), 14 bitowy akcelerometr/magnetometr (ST Micro LSM303D), 3 – osiowy żyroskop/akcelerometr (Invensense MPU 6000) oraz barometr (MEAS MS5611).

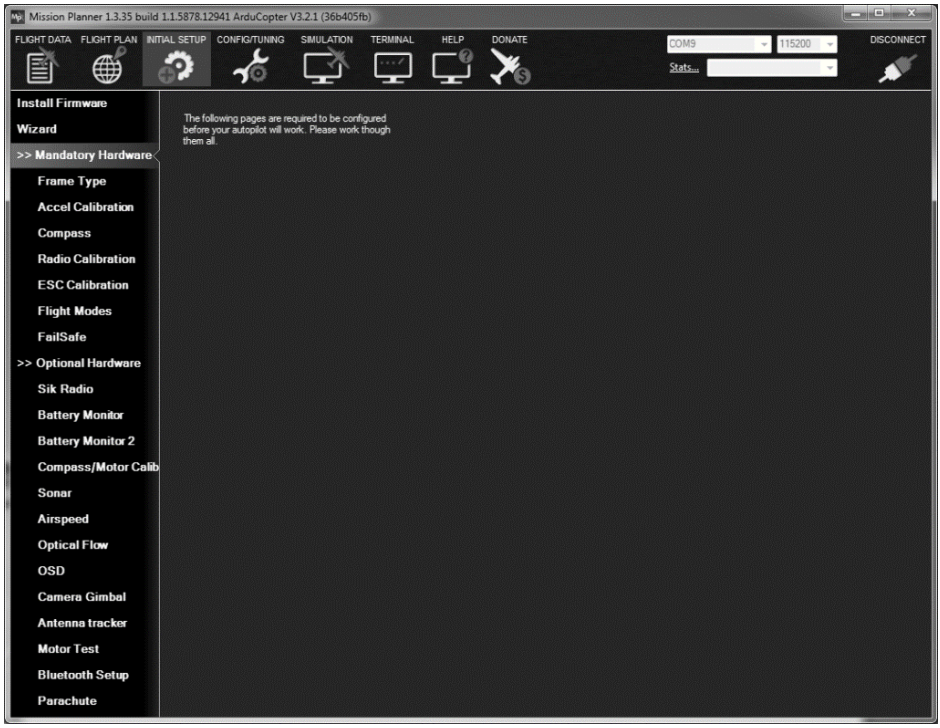
Za zasilanie bezzałogowego statku powietrznego odpowiada akumulator pięcio celowylitowo-polimerowy o napięciu nominalnym 18,5 V oraz pojemności 60000 mAh. Akumulatory litowo-polimerowe to w chwili obecnej jedne z najlepszych akumulatorów na rynku, gdyż charakteryzują się mniejszą wagą w porównaniu z akumulatorami niklowo-metalowo-wodorkowymi bądź akumulatorami niklowo-kadmowymi, posiadają znikome tempo samorozładowania oraz słyną z dużej wydajności prądowej.

Sterowanie bezzałogowym statkiem powietrznym oparte jest na odbiorniku i nadajniku RC. Transmiter RC posiada 16 sterowalnych kanałów, dzięki czemu możemy sterować dronem wzdłuż każdej osi. Odbiornik FRSKY, użyty do budowy prototypu, posiada dziewięć sterowalnych kanałów PWM oraz jeden protokół komunikacyjny SBUS. Teoretyczny zasięg dwukierunkowej wymiany parametrów, pomiędzy użytymi przez nas modułami, wynosi od 3 do 5 kilometrów.

Wykorzystaliśmy ramę Tarot IRON MAN 650 wykonaną z włókna węglowego oraz aluminium. Jest to bardzo stabilna, sztywna oraz lekka konstrukcja, posiadająca rozstaw silników 650 mm i możliwość założenia śmigieł od 10-13 cali.

2. KONFIGURACJA KONTROLERA LOTU PIXHAWK, POPRZEZ GUI MISSIONPLANNER

Konfiguracja kontrolera zaczyna się od wgrania odpowiedniego oprogramowania w wybranej przez nas wersji w zależności od wymagań i założeń projektu. W naszym przypadku postawiliśmy na odpowiedni system w ArduCopter w najnowszej stabilnej wersji, kompatybilny z MissionPlannerem. Do bezpośredniego połączenia kontrolera lotu z GUI odpowiednio przygotowanym na naszym komputerze systemem, potrzebujemy odpowiednich sterowników, które odpowiedzialne są za komunikację poprzez USB. W prawym górnym rogu, bezpośrednio po podłączeniu kontrolera widzimy możliwość nawiązania połączenia poprzez wybranie odpowiedniego portu COM.



Rys. 1. Wygląd GUI oprogramowania Mission Planner od ArduPilot służącego do konfiguracji bezzałogowca
Źródło: opracowanie własne.

Bezpośrednio po połączeniu kontrolera i wgraniu odpowiedniego oprogramowania musimy wybrać odpowiedni typ ramy, w naszym przypadku jest to quadrocopter, czyli dron z czterema silnikami w standardowej konfiguracji obrotu silników. Kolejno, zaczynając od lewego przodu, mamy silniki (CCW, CW, CCW, CW). Następnym niezbędnym krokiem jest kalibracja akcelerometrów w zakładce widocznej po lewej stronie na zrzucie ekranu oraz kalibracja dwóch niezależnych kompasów w zakładce Compass. Równie niezbędnym krokiem jest kalibracja naszego transmitera RC. Kolejno musimy sprawdzić, czy wszystkie osie są odpowiednio ustawione według wybranego MOD'U, w którym chcemy latać. Odpowiednio ustawione i wycentrowane gimbale zwiększają pewność bezpiecznego lotu. Oprócz ustawienia kanałów wykorzystywanych podczas lotów, musimy ustawić przełącznik, który będzie odpowiedzialny za przełączanie trybów lotu w naszym bezzałogowym statku powietrznym. Na pierwszej pozycji przełącznika ustawiamy tryb lotu STABILZE. Tryb ten jest podstawowym i często używanym trybem lotu. W trybie model jest stabilizowany przez żyroskop w pozycji poziomej, by zapewnić równowagę, jednak model sam nie utrzyma swojej pozycji, jest podatny na złe wyważenie modelu, wiatr i różne czynniki atmosferyczne. W trybie cały czas mamy kontrolę nad modelem za pomocą transmitera i cały czas jesteśmy zmuszeni do kontroli kierunku oraz wysokości lotu. Drążek gazu cały czas działa w trybie ręcznym, czyli kontroler nie ma wpływu na wysokość modelu i zgodnie z założeniami tego trybu cały czas dryfuje, co oznacza, że ciągle musimy kontrolować jego położenie.

Na drugiej pozycji przełącznika trójpozycyjnego ustawiamy tryb Loiter, który ma za zadanie utrzymanie wysokości modelu, ale także na podstawie GPS stara się utrzymać pozycję w przestrzeni powietrznej. W trybie Loiter używane są wszystkie czujniki pokładowe bezzałogowego statku powietrznego. Przypadkowe puszczenie wszystkich drążków powoduje zapamiętanie i utrzymanie bieżącej pozycji naszego statku powietrznego.

Na trzeciej pozycji ustawiamy tryb awaryjny lotu RTL (ang. *Return to Launch*), który pozwala w sytuacjach awaryjnych na automatyczny powrót do domu. Tryb RTL znajduje się w zakładce Mandatory Hardware, kolejno w zakładce FAILSAFE. W przypadku gdy bezzałogowiec z różnych przyczyn straci połączenie transmitera RC z odbiornikiem umieszczonym na bezzałogowym statku powietrznym, w takiej sytuacji automatycznie dron wzlatuje do ustawionej wcześniej pozycji RTL_ALT, czyli wysokości, na którą dron wzniesie się podczas zmiany na ten tryb i wróci po linii poziomej (najkrótszej od miejsca startu). Stąd musimy względnie przed każdym startem w zależności od miejsca, w którym się znajdujemy, odpowiednio ustawić wysokość RTL_ALT, tak aby w razie niebezpiecznej sytuacji była odpowiednia (wysokość ta musi być większa niż maksymalna wysokość przeszkód nad danym terenem), biorąc pod uwagę obostrzenia panujące w danej strefie, w której planujemy lot. Opcję RTL_ALT znajdziemy w zakładce configuration w drzewie wszystkich parametrów pod wyżej wymienioną nazwą.

Przed pierwszym lotem musimy skonfigurować jeszcze nastawy regulatora PID oraz sprawdzać ciągle wskazania z czujników pokładowych. W naszym przypadku musimy ustawić na oddzielny przełącznik tryb ALT-HOLD (to specjalny tryb lotu, który stara się zachować aktualną wysokość, na jakiej model się znajduje na podstawie wskazań barometru), bezpośrednio z włączonym trybem ALT-HOLD wykonujemy start i włączamy bezpośrednio tryb AUTO-TUNE, który służy do automatycznego doboru nastaw regulatora PID, dzięki czemu dron sam bada i reaguje na swoje zachowanie podczas lotu i dobiera nastawy optymalnie pod płynność lotu.

W przypadku sterowania bezzałogowym statkiem powietrznym wybraliśmy najbardziej popularny tryb lotu MODE 2 (tryb najbardziej zbliżony do prawdziwego układu lotniczego, w którym bezpośrednio osie YAW oraz regulacja przepustnicą gazu jest ustawiona na lewym drążku aparatury, osie PITCH i ROLL kolejno na prawym drążku manipulatora.

3. LOTY BEZZAŁOGOWYM STATKIEM POWIETRZNYM

Po odpowiednim skonfigurowaniu bezzałogowego statku powietrznego kolejnym etapem było przeprowadzenie testów, które z uwagi na pokaźny rozmiar naszej konstrukcji, kwalifikujący się do kategorii otwartej, podkategorii A3, musiały zostać przeprowadzone na przestrzeni otwartej z dala od budynków i innych zabudowań.

Podczas testów omawiana konstrukcja bezzałogowego statku powietrznego zachowywała się nadzwyczaj dobrze. Stabilność lotu można ocenić na wręcz wzorową, gdyż dzięki zastosowaniu czterech silników BLDC oraz zarządzania ich pracą przy pomocy kontrolera lotu PixHawk Radiolink wyposażonego w różne sensory, między innymi żyroskopy, cała konstrukcja znakomicie kompensowała wpływ wszelkich czynników atmosferycznych takich jak powiewy wiatru, zarówno te słabsze, jak i mocniejsze. Widoczne było to szczególnie podczas ustawienia bezzałogowca w trybie swobodnego zawisu (ang. *Hovering Mode*), podczas którego zadaniem bezzałogowego statku powietrznego było utrzymanie swojej pozycji w wyznaczonym przez osobę sterującą miejscu.

W warunkach sprzyjających lotom, czyli przykładowo podczas bezwietrznej pogody, czas lotu na jednym, w pełni naładowanym, akumulatorze wynosi około 25 minut. Dla porównania czas lotu podczas wietrznej pogody, gdy bezzałogowiec musi ciągle kompensować parametry lotu, przeciwdziałając niesprzyjającym warunkom atmosferycznym, wynosi około 15 minut, co stanowi nieco ponad połowę czasu lotu podczas sprzyjającej pogody.

Podczas lotów sprawdziliśmy także wszystkie pozostałe tryby lotów, które zostały nam umożliwione dzięki odpowiedniej konfiguracji kontrolera lotu PixHawk Radiolink w ogólnodostępnej aplikacji Mission Planner. Należały do nich między innymi Stabilize, Alt Hold i Loiter. Dzięki przeprowadzeniu testów na wszystkich dostępnych trybach byliśmy w stanie ustalić, który z nich sprawuje

się najlepiej. W naszym przypadku był to tryb Loiter, który stabilizuje lot przy użyciu wszystkich aktualnie dostępnych sensorów, głównie opierając się na pozycjonowaniu GPS.

4. MOŻLIWOŚĆ ROZWINIĘCIA PROJEKTU

Opisany w artykule bezzałogowy statek powietrzny typu quadrocopter jest maszyną, w której jest wiele możliwości ulepszenia w zależności od przyszłego zadania, do którego miałby zastosowanie.

W planach rozwojowych naszego bezzałogowca jest bezpośrednie połączenie z minikomputerem typu Raspberry Pi, który bezpośrednio będzie wydawał rozkazy dotyczące lotu oraz wysyłał dane telemetryczne bezpośrednio do stacji naziemnej połączonej z dronem poprzez protokół TCP/IP (ang. *Transmission Control Protocol/Internet Protocol*). Dodatkowo dodanie zewnętrznej telemetrii pozwoli nam na automatyczne loty po zaprogramowanych wcześniej punktach oraz ciągłe monitorowanie czujników pokładowych. Dzięki temu wcześniej możliwe będzie zaprogramowanie misji i odtworzenie jej bezpośrednio w locie. Możliwość podłączenia bezpośrednio do minikomputera Raspberry Pi czujników pokładowych spowoduje znaczny wzrost funkcjonalności bezzałogowego statku powietrznego. Przykładowo po podłączeniu i zaprogramowaniu niezbędnych czujników takich jak: czujniki pyłu pm2.5 oraz pm10, czujnik wilgotności, czujnik formaldehydu (HCHO), czujnik lotnych związków organicznych (LDO), możemy monitorować jakość spalanych produktów w piecach domowych, dzięki czemu mielibyśmy realny wpływ na nasze środowisko i walkę z najbardziej popularnym trucicielem czyli smogiem. Dodatkowo podłączenie kamery poprzez mikrokomputer Raspberry Pi umożliwi ciągłe przesyłanie obrazu transmitowanego bezpośrednio na pokładzie bezzałgowca. Rozwój bezzałogowych statków powietrznych powoduje, że miejsc, w których możemy spotkać bezzałogowe statki powietrzne, ciągle przybywa, a wykorzystanie ich potencjału ciągle będzie zaskakiwać innowacyjnością. Jednosilnikowy, bezzałogowy statek powietrzny jest urządzeniem, którego konstrukcja posiada niezaprzeczalnie liczne wady. Jednakże pomimo tych wad urządzenie to jest w stanie odbyć relatywnie stabilny lot, pod warunkiem, iż odbędzie się on w pomieszczeniu zamkniętym. Na podstawie przeprowadzonych testów wynika jednoznacznie, że odbycie lotu z dala od budynków, na przestrzeni otwartej jest bardzo utrudnione, czego powodem jest duża podatność omawianej konstrukcji na czynniki atmosferyczne, których nie sposób uniknąć, przykładowo podmuchy wiatru. Reasumując, cały proces tworzenia konstrukcji jednosilnikowego bezzałogowca objął wiele dziedzin, poprzez projektowanie i druk 3D, aż po samodzielne łączenie elementów elektronicznych.

5. PODSUMOWANIE

Czterosilnikowy, bezzałogowy statek powietrzny typu quadrocopter jest niewątpliwie urządzeniem, którego konstrukcja należy do jednej z najbardziej opłacalnych i popularnych, przez wzgląd na omawianą podczas testów stabilność lotu, która jest dostępna dla użytkownika w przystępnej cenie. Czas lotu takiego urządzenia jest również sporym atutem, gdyż pomimo użycia czterech silników, których zapotrzebowanie na energię elektryczną jest spore, maksymalny udźwig pozwala na zastosowanie akumulatora o większej pojemności, co pozytywnie wpływa na czas lotu na jednym akumulatorze. Kolejnym, niepodważalnym atutem takowej konstrukcji jest bardzo szeroka możliwość rozbudowy omawianej konstrukcji, przykładowo poprzez dodanie różnych sensorów i innych dodatków, między innymi kamery, co znacznie zwiększy funkcjonalność tego urządzenia.

LITERATURA

1. Ballaster R., Firman Andrew, Clot Eleanor, *A practical Guide to Drone Law*, Wydawnictwo Law Brief Publishing Ltd, 2017.
2. Halliday B., *Drones: Mastering Flight Techniques*, Wydawnictwa CreateSpace Independent Publishing Platform, 2017.
3. Halliday B., *Drones: The Professional Drone Pilot's Manual*, Wydawnictwa CreateSpace Independent Publishing Platform, 2016.
4. Krykowski K., *Silniki PM BLDC: właściwości, sterowanie, aplikacje*, Wydawnictwo BTC, Legionowo 2015.

Piotr STOREK, Karol SIWIEC
Michał BAZAN, Rafał NAZARKO

dr inż. Bartosz PAWŁOWICZ
opiekun naukowy

STEROWNIK DO MATRYCY LED 7 x 21

Artykuł opisuje zasadę działania, schemat budowy oraz instrukcję obsługi sterownika do matrycy LED. Pozwala na wykorzystanie układu do celów praktycznych takich jak wskazywanie aktualnej godziny, daty oraz jako budzik.
Słowa kluczowe: układ elektroniczny, automatyka, systemy wbudowane.

WPROWADZENIE

Opisywany sterownik to oprogramowanie mikroprocesora oraz układ elektroniczny składający się z:

- części analogowej odpowiedzialnej za odpowiednie wysterowanie napięcia zasilania matrycy LED,
- części cyfrowej, która ma na celu multipleksowanie sygnału pochodzącego z części analogowej oraz jest odpowiedzialna za relatywnie dokładne utrzymywanie godziny i daty wyświetlanej na matrycy,
- układu mikroprocesorowego, w którego skład wchodzą najprostsze komponenty, takie jak kondensatory, rezystory oraz mikroprocesor.

Opis oprogramowania znajduje się w innej sekcji. Sterownik w połączeniu z matrycą LED pozwala na wykorzystanie układu do celów praktycznych, jak wskazywanie aktualnej godziny, daty oraz jako budzik, który udostępnia pięć budzików, które po wyłączeniu alarmu pozostają nieaktywne do następnych nastaw. Oprogramowanie pozwala na ustawienie długości drzemki w zakresie od jednej do sześćdziesięciu minut. Poza użytkiem praktycznym, matryca wraz ze sterownikiem stanowi ciekawą ozdobę. Obsługa tego urządzenia jest bardzo prosta i intuicyjna. Instrukcja postępowania znajduje się w następnym rozdziale.

1. INSTRUKCJA OBSŁUGI

1.1. Podstawowe funkcje

Aby wejść do menu nastaw należy nacisnąć przycisk „FUNCTION” i przytrzymać go przez co najmniej dwie sekundy. Podczas oczekiwania na przejście do

menu nastaw na matrycy wyświetla się napis „HOLD”, który sugeruje, aby przycisk był wciśnięty. Po upływie dwóch sekund użytkownik znajduje się w menu nastaw. Oczom użytkownika powinna się ukazać pierwsza możliwa nastawa, tj. „TIME”, czyli ustawienia związane ze zliczanym przez układ czasem. Po przejściu do menu, użytkownik może nawigować przyciskami „+” oraz „–”. Przyciśnięcie któregokolwiek z przycisków nawigacji zmienia wyświetlaną wartość wyłącznie o jedną opcję. Przewijanie listy nastaw, podczas gdy wciśnięty jest przycisk, jest zablokowane. Wybranie interesującej użytkownika nastawy odbywa się poprzez wciśnięcie przycisku „OK”. UWAGA: wielkości takie jak godzina, data czy czas drzemki są zapisywane natychmiast po każdej zmianie przez przyciski nawigacyjne. Po braku aktywności przez piętnaście sekund następuje wyjście do menu głównego, a po dwudziestu sekundach braku aktywności następuje wyjście z menu. Podczas gdy użytkownik jest w menu nastaw budzików czy brzęczyka, po upłynięciu piętnastu sekund od ostatniej aktywności ZMIANY NIE SĄ ZAPISYWANE.

1.2. Ustawianie czasu – „TIME”

Po wybraniu opcji „TIME” możliwe jest ustawienie bieżącej godziny, liczby minut oraz wyjście do głównego menu. Wielkość, która będzie ustawiana po zatwierdzeniu przyciskiem „OK”, jest wyświetlana. Po wybraniu interesującej użytkownika opcji i wciśnięciu przycisku „OK” wyświetlane są:

- XX : HH,
- MM : XX, gdzie XX to aktualnie ustawiana wielkość, HH – sugestia, że ustawiana jest liczba godzin, MM sugestia, że nastawiana jest liczba minut.

Zmiana wielkości następuje poprzez wykorzystanie przycisków nawigacji, a zatwierdzenie przyciskiem „OK” pozwala powrócić użytkownikowi do podmenu. Wybranie opcji „EXIT” pozwala wyjść do menu głównego nastaw.

1.3. Ustawianie aktualnej daty – „DATE”

Po wybraniu opcji „DATE” możliwe jest ustawienie bieżącej daty oraz wyjście do głównego menu. Wielkość, która będzie ustawiana po zatwierdzeniu przyciskiem „OK”, jest wyświetlana. Po wybraniu interesującej użytkownika opcji i wciśnięciu przycisku „OK” wyświetlane są:

- XX : DD,
- MM : XX,
- 20XX, gdzie XX to aktualnie ustawiana wielkość, DD to sugestia, że nastawiany jest dzień miesiąca, MM to sugestia, że aktualnie ustawiany jest miesiąc.

20XX – możliwość nastawy roku znajduje się w przedziale od 2000 do 2099, co jest spowodowane ograniczeniami układu czasu rzeczywistego DS1307.

Zmiana wielkości następuje poprzez wykorzystanie przycisków nawigacji, a zatwierdzenie przyciskiem „OK" pozwala powrócić użytkownikowi do podmenu. Wybranie opcji „EXIT" pozwala wyjść do menu głównego nastaw.

1.4. Ustawianie długości drzemki – „BUZZ"

Możliwość włączenia brzęczyka, który powiadamia brzęczeniem o rozpoczętej nowej godzinie. Możliwe nastawy:

- „OFF" – wyłączony,
- „ON" – włączony,
- „EXIT" – wyjście z podmenu.

1.5. Ustawianie długości drzemki – „SNZE"

Po wybraniu opcji „SNZE" możliwe jest ustawienie długości drzemki alarmu (opis postępowania z alarmem w innym podrozdziale). Po wejściu do tego podmenu dostępne są następujące opcje:

- XXMM, gdzie XX to puste, białe znaki (NULL), MM to ilość minut określająca długość drzemki,
- „EXIT".

Po zatwierdzeniu „XXMM" przyciskiem „OK", wyświetlana jest następująca kombinacja „SNMM", gdzie SN ma sugerować, że ustawiana wielkość to drzemka. Zmiana wielkości następuje poprzez wykorzystanie przycisków nawigacji, a zatwierdzenie przyciskiem „OK" pozwala powrócić użytkownikowi do podmenu. Wybranie opcji „EXIT" pozwala wyjść do menu głównego nastaw.

1.6. Ustawianie jednorazowego alarmu – „S_AL"

Po wybraniu tej opcji wyświetla się menu, po którym użytkownik może nawigować przyciskami „+" i „–", aby wybrać pożądany alarm lub wyjść z podmenu. Po wybraniu dowolnego alarmu dostępne są następujące opcje:

- „OFF" – wyłącza alarm,
- „ON" – włącza alarm,
- „SET" – przejście do ustawienia godziny,
- „EXIT" – wyjście z nastaw danego alarmu.

UWAGA: gdy alarm jest aktywny, wciśnięcie przycisku „+" wywołuje drzemkę, a „OK" wyłącza alarm. Po dokonaniu nastaw danego alarmu jest on automatycznie aktywowany.

1.7. Wyjście z menu nastaw – „EXIT"

Zatwierdzenie tej opcji powoduje wyjście z menu nastaw i normalny stan pracy urządzenia w sekwencji wyświetlanie godziny (dwadzieścia sekund), daty (pięć sekund, format DDMM) oraz roku (przez pięć sekund).

2. DOKUMENTACJA TECHNICZNA

2.1. Repozytorium projektu

Niniejsza sekcja zaczyna się od opisu oprogramowania, a kończy na układzie elektronicznym. Nie wszystkie pliki są opisane w całości. Aby dokonać głębszej analizy, należy udać się na stronę, na której znajduje się repozytorium: https://github.com/DevxMike/led_matrix.

Rys. 1. Kod QR zawierający adres do repozytorium projektu
Źródło: opracowanie własne.

2.2. Opis operacji niskopoziomowych

Plik „twi.h" zawiera definicje funkcji konwertujących kod BCD na kod dziesiętny, kod dziesiętny na kod BCD oraz funkcję do sprawdzenia, czy rok jest przestępny. W pliku znajdują się również deklaracje funkcji realizujących inicjalizację układu RTC, przesłanie bufora informacji, odczytania bufora informacji z układu czasu rzeczywistego oraz funkcji określającej dzień tygodnia. Plik „twi.c" zawiera definicje funkcji, których deklaracje znajdują się w pliku oraz tych, które powinny zostać ukryte przed potencjalnym „użytkownikiem" kodu, takie jak np. warunek początku transmisji.

W „timers.h" oraz „timers.c" znajdują się definicje funkcji inicjalizującej 16-bitowy timer sprzętowy. Tryb CTC, preskaler 1, zawartość rejestru przechowującą zawartość wzorcową 0x1F3Fh (7999). Przerwania generowane są z częstotliwością 2 kHz.

Plik „spi.h" zawiera definicje typu danych, który przechowuje dane przesyłane do rejestrów matrycy LED oraz deklaracje dwóch funkcji: inicjalizacja SPI oraz przesył trzech bajtów do matrycy. W „spi.c" znajdują się definicje wyżej wymienionych funkcji oraz takich, które nie powinny znaleźć się w globalnej przestrzeni nazw.

Deklaracje funkcji potrzebnych do obsługi klawiszy (nie zostały wykorzystane przerwania) oraz zmiennych globalnie dostępnych znajdują się w pliku „controls.h". W pliku z rozszerzeniem „.c" z tą samą nazwą znajdują się definicje funkcji, które pozwalają na podpięcie klawiszy do dowolnego portu procesora. Plik definiuje zmienne wskaźnikowe „ulotne", których zadaniem jest przechowywanie adresu portu każdego z przycisków, zmienne całkowite, które przechowują wagę bitu danego przycisku.

Plik „chars.h" zawiera definicję typu wyliczeniowego, który pomaga określić, ile „kropek" ma zostać wyświetlonych na matrycy (none – zero, one – separator daty, jedna kropka, both – dwie kropki, separator godzin i minut). W pliku zawarta jest również deklaracja funkcji przygotowującej zestaw danych dla matrycy zależnie od tego, co „użytkownik" kodu chce wyświetlić oraz czy potrzebuje kropki. W „chars.c" można znaleźć definicję tablicy dwuwymiarowej znaków zdefiniowanych dla matrycy przechowywanej w pamięci EEPROM mikroprocesora oraz definicję funkcji przygotowującej zestaw danych dla rejestrów matrycy.

Poniżej został zamieszczony kod jednego z najważniejszych plików dla tego projektu. Znajdują się w nim definicje struktur danych potrzebnych do przechowywania takich wielkości jak czas, data, flagi oraz stany alarmów, typu wyliczeniowego, przez który kod sprawdzający maksymalną ilość dni w miesiącu przy nastawach jest bardziej czytelny. W pliku również zostały zawarte: deklaracje funkcji inicjalizującej alarmy, sprawdzającej stan alarmów oraz zadeklarowana została zmienna dostępna globalnie, która przechowuje flagi przydatne w określaniu niektórych stanów systemu takich jak sprawdzenie, czy alarm jest aktywny.

```
#ifndef data_structs_h
#define data_structs_h
#include <avr/io.h>
#include "controls.h"
#define ALARM_MASK 0x01

void prepare_set(uint8_t first, uint8_t second, uint8_t third,
uint8_t fourth, uint8_t row, reg_data_t* set, uint8_t dots){
first = eeprom_read_byte(&characters[first][row]); //read bytes
from eemem defined by char codes
second = eeprom_read_byte(&characters[second][row]);
third = eeprom_read_byte(&characters[third][row]);
fourth = eeprom_read_byte(&characters[fourth][row]);
set->first = set->second = set->third = 0x00; //zero out data
set->first |= (first << 3); //set first digit
set->first |= (second & 0xF8) >> 3; //set 2 cols second digit
set->second |= (second & 0x07) << 5; //set second digit
set->second |= (third & 0xFE) >> 1; //4 rows third digit
set->third |= (third & 0x01) << 7; //set third digit
```

```
set->third |= fourth << 1;

enum Month{
january = 1, february, march, april, may,
june, july, august, september, october, november, december
};
typedef struct{
uint8_t hours : 5;
uint8_t mins : 6;
uint8_t seconds : 6; //seconds, max val 59
}time_t;
typedef struct{
uint8_t year : 7; //max val = 99
uint8_t month : 4; //max val = 15
uint8_t day_1 : 5; //max val = 31
uint8_t day_2 : 3; //day_1 - day of month, day_2 - day of week
}date_t;
typedef struct{
date_t date;
time_t time;
}time_data_t;
typedef struct{
uint8_t days_flags; //days of week when alm has to be triggered
uint8_t other_flags : 2; //second snoze, first alm trigerred
}flags_t;
typedef struct{
time_t alm_time;
flags_t flags;
uint32_t tim; //max 7200000 -> 60 min
uint8_t state;
}alarm_t;
extern uint8_t flags;
void check_alarm(alarm_t*, const time_data_t*, uint8_t, const
uint8_t);
void init_alarms(alarm_t*, uint8_t);
#endif
```

Listing 1. Plik „data_structs.h”

Równie ważny z plikiem nagłówkowym jest plik z rozszerzeniem „.c”. Zawiera definicje funkcji inicjalizującej alarmy (ustawia wszystkie nastawy na zera, alarmy wyłączone) oraz funkcji, która sprawdza stany alarmów (sprawdzanie, czy alarm powinien zostać aktywowany, w razie gdy jest aktywny, czy powinien zostać wyłączony lub wyłączony na czas drzemki i ponownie włączony). Poniżej znajduje się jego struktura:

```
#include "data_structs.h"
uint8_t flags;//first bit stands for alarm, second for buzzing
while mins == secs == 00, third exit, fourth is set if buzzer
enabled
//fifth for check wheter data or time is being set
void check_alarm(alarm_t* alarms, const time_data_t* time,
uint8_t quantity, const uint8_t coef){
alarm_t* pt = alarms;
uint8_t tmp = 0;
for(uint8_t i = 0; i < quantity; ++i){
switch(pt[i].state){
case 1:
tmp = (pt[i].alm_time.hours == time->time.hours) &&
(pt[i].alm_time.mins == time->time.mins) &&
(pt[i].flags.days_flags & (1 << time->date.day_2)) && !time-
>time.seconds;
if((pt[i].flags.other_flags = tmp? 1 : 0)){
if(!(flags & ALARM_MASK)) flags |= ALARM_MASK;
pt[i].state = 2;
}
break;
case 2:
if(!(flags & ALARM_MASK)) flags |= ALARM_MASK;
if(okK){
pt[i].tim = 60;
pt[i].state = 3;
}
else if(incK){
pt[i].tim = 60;
pt[i].state = 5;
}
break;
case 3:
if(pt[i].tim && !okK){
pt[i].state = 2;
}
else if(!pt[i].tim && okK){
if(i >= 5){ //5 is a first non repetive alm
pt[i].flags.days_flags = 0;
}
pt[i].flags.other_flags = 0;
pt[i].state = 4;
}
break;
case 4:
if(!okK){
```

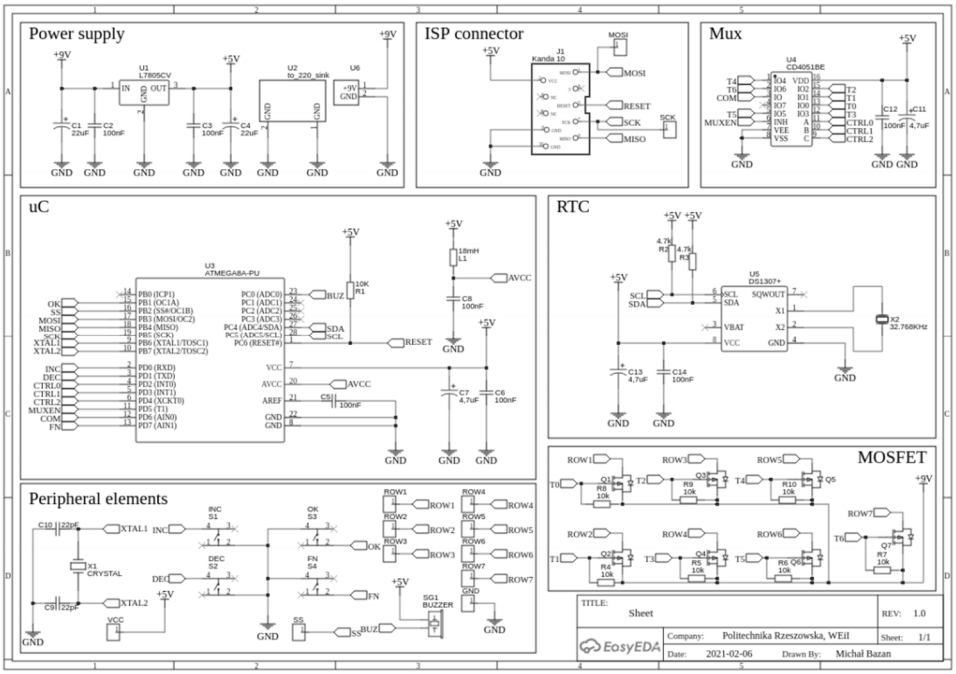
```
pt[i].state = 1;
}
break;
case 5:
if(pt[i].tim && !incK){
pt[i].state = 2;
}
else if(!pt[i].tim && incK){
pt[i].flags.other_flags = 2;
pt[i].state = 6;
}
break;
case 6:
if(!incK){
pt[i].tim = (uint32_t)coef * 2000 * 60;
pt[i].state = 7;
}
break;
case 7:
if(pt[i].tim == 0){
pt[i].flags.other_flags = 1;
pt[i].state = 2;
}
break;
}
if(pt[i].tim > 0) --pt[i].tim;
}
for(uint8_t i = 0; i < quantity; ++i){
if(pt[i].flags.other_flags == 1) ++tmp;
}
if(!tmp) flags &= ~ALARM_MASK;
else if(tmp) if(!(flags & ALARM_MASK)) flags |= ALARM_MASK;
}
void init_alarms(alarm_t* alm, uint8_t q){
for(uint8_t i = 0; i < q; ++i){
alm[i].tim = alm[i].flags.other_flags = alm[i].flags.days_flags =
0;
alm[i].state = 1;
alm[i].alm_time.hours = alm[i].alm_time.mins = 0;
}
}
```

Listing 2. Plik „data_structs.c”

2.3. Skrócony opis pliku z funkcją main

Definicje makr upraszczające sprawdzanie stanów zapisanych w zmiennej flags, zmiennych wykorzystywanych do operacji niskopoziomowych i przechowywania nastaw alarmów oraz załączenie odpowiednich plików nagłówkowych znajdują się przed wejściem do funkcji głównej *int main(void)*. W jej środku natomiast, pierwszym krokiem jest inicjalizacja peryferiów, timera sprzętowego oraz przypisanie 0x00 zmiennej flags. Następnie znajdują się definicje zmiennych wykorzystywanych przez funkcje *main* do manipulacji peryferiami, zarządzania wewnętrznymi stanami oraz wewnętrznymi timerami, których baza czasowa opiera się na timerze sprzętowym (przez małą dokładność timera sprzętowego, do zliczania czasu zastosowany został układ RTC). Pętlę nieskończoną *while(1)* można opisać w następujący sposób. W cyklach trwających $\frac{1}{2000}$ sekundy wykonywane są wszystkie czynności potrzebne do poprawnego funkcjonowania sterownika. Pierwszym grafem, który determinuje to, w jakim stanie znajduje się system, jest „main graph", który cyklicznie sprawdza pojawienia się warunku przejścia do menu nastaw (tj. wciśnięty przycisk „FUNCTION"), jest to graf binarny, którego tablice warunków skoku, tablice adresów oraz tablice stanów wyjść zawarte są w pamięci EEPROM procesora. Po przejściu do stanu, w którym przeprowadzane są nastawy podstawowe, sprawdzany jest stan alarmu, jeśli jest on aktywowany, to następuje bezwarunkowe przejście do stanu podstawowego (wyświetlanie godziny i daty). Jeśli zaś alarm jest nieaktywny, po przejściu do którejkolwiek opcji w menu nastaw graf oczekuje na warunek wyjścia („flags & EXIT_CONDITION"), aby powrócić do głównego menu, lub na alarm, by wrócić do podstawowego stanu. Następnym grafem jest automat aktualizujący wyświetlaną datę, która pobierana jest z układu RTC, a następnie konwertowana na system dziesiętny. Aktualizacja daty dzieje się we wszystkich stanach poza tymi, które wymagają wprowadzania zmian w zliczanej przez RTC zawartości (nastawy czasu i daty). Kolejnym automatem jest graf sterujący brzęczykiem zależnie od stanów flag. Odpowiedzialny jest za brzęczenie w trakcie alarmu w taki sposób, aby brzmiał jak zwykły budzik. Następny blok instrukcji warunkowych pozwala na sterowanie zawartością wyświetlanej na matrycy LED, zależnie od stanów grafu głównego i ewentualnie podgrafów pozwalających na nastawy podstawowe. Kolejnymi automatami są grafy, które pozwalają na zmianę czasu, daty, czasu drzemki, nastawy alarmów oraz przełączanie brzęczyka. Wszystkie te grafy mają swoje własne podgrafy, które manipulują zawartością wyświetlaną. Ostatnim do omówienia jest blok sterujący multiplekserem, który wyłącza wyjścia układu cyfrowego, przełącza aktywne wyjście, przygotowuje zestaw danych dla rejestrów matrycy, przesyła dane oraz włącza multiplekser. Pod blokiem multipleksera znajdują się instrukcje sterujące „timerami" programowymi oraz pętla, która oczekuje na przerwanie zegarowe timera sprzętowego (otrzymanie cyklu $\frac{1}{2000}$ sekundy).

3. SCHEMAT IDEOWY STEROWNIKA MATRYCY

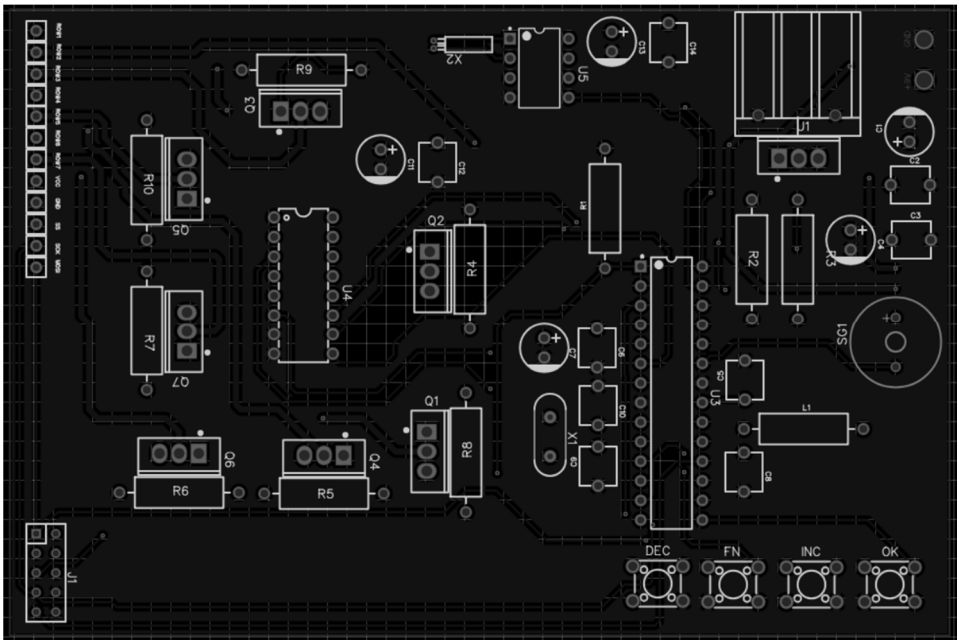


Rys. 2. Schemat ideowy sterownika matrycy
Źródło: opracowanie własne.

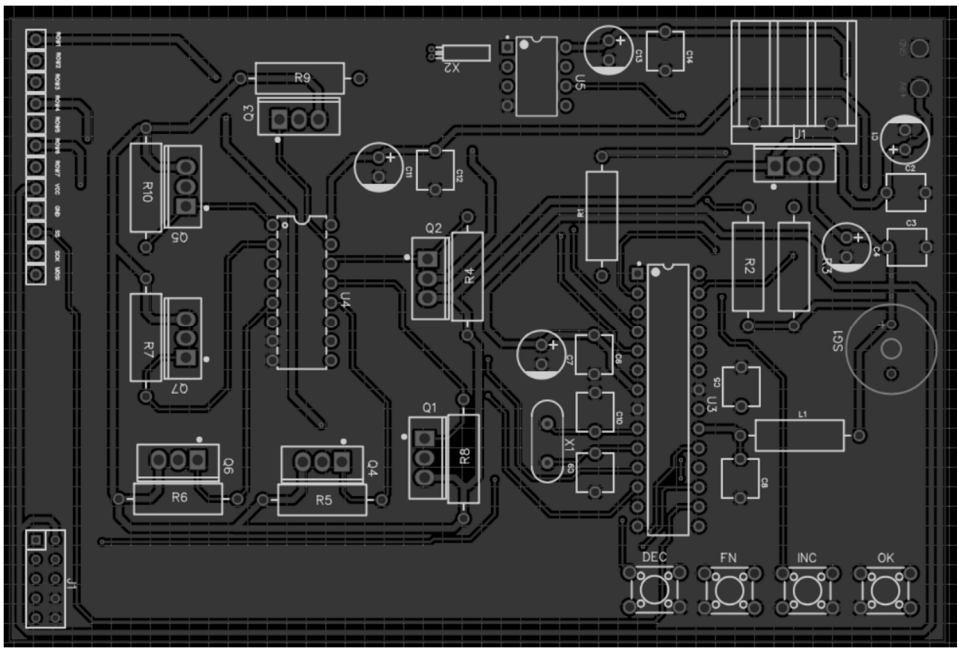
Powyżej zamieszczony został schemat ideowy sterownika matrycy. Opis poszczególnych sekcji w tym widoku można znaleźć poniżej:

- **Power supply** – ta sekcja układu zawiera stabilizator napięcia stałego doprowadzonego do układu poprzez pady oznaczone jako „U6", zawarty również został radiator odprowadzający ciepło ze stabilizatora.
- **ISP connector** – standardowa KANDA pozwalająca na przeprogramowanie procesora oraz wyprowadzenia sygnałów SPI.
- **Mux** – sekcja zawiera multiplekser, który pozwala na zasilenie odpowiedniego tranzystora zasilającego pożądany wiersz matrycy LED.
- **uC** – ta część układu zawiera mikroprocesor, jego zasilanie oraz wyprowadzenia odpowiednich pinów procesora użytych do sterowania i komunikacji z innymi komponentami. RTC Sekcja zawiera układ DS1307, tj. układ czasu rzeczywistego, który pozwala na relatywnie dokładny pomiar czasu wyświetlanego na matrycy LED. Komunikacja opiera się na interfejsie I 2C . Układ nie ma zasilania bateryjnego.

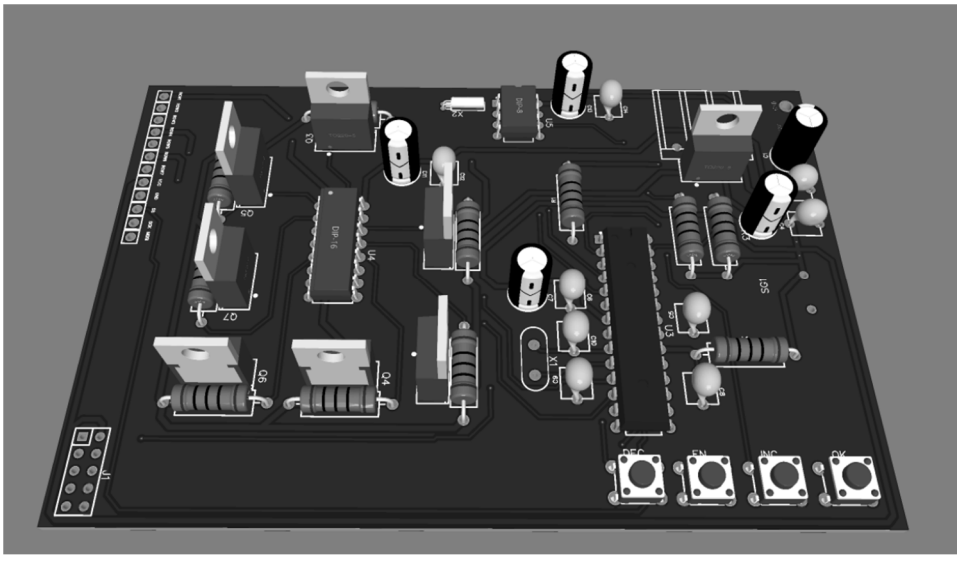
- **MOSFET** – ta część układu zawiera tranzystory doprowadzające zasilanie do wierszy matrycy. Są to MOSFETy z kanałem typu P, dlatego aktywowane są stanem niskim dostarczanym przez multiplekser.
- **Peripheral elements** – sekcja zawiera wszystkie elementy, które powinny być kojarzone z doprowadzonym/odprowadzonym sygnałem od/do części wykonawczych układu. W skład wchodzą: wyjścia tranzystorowe zasilające wiersze matrycy oraz wyjście buzzera, przyciski wykorzystywane do dokonywania nastaw oraz taktowanie procesora.



Rys. 3. Projekt płytki PCB, połączenia na tylnej stronie
Źródło: opracowanie własne.



Rys. 4. Projekt płytki PCB, połączenia na przedniej stronie
Źródło: opracowanie własne.



Rys. 5. Wizualizacja 3D płytki
Źródło: opracowanie własne.

4. PODSUMOWANIE

Przedstawiony w tym artykule sterownik do matrycy LED w bardzo prosty, ale wydajny i niezawodny sposób realizuje powierzone mu zadanie. Może się sprawdzić w wielu praktycznych zastosowaniach takich jak np. budzik. Możliwe jest zastosowanie tego komponentu również w technologiach typu IoT, po odpowiednim zmodyfikowaniu kodu źródłowego. Planowanym kierunkiem rozwoju tego projektu jest możliwość sterowania matrycą za pomocą pilota na podczerwień lub modułu bluetooth. Pozwoli to na bardziej uniwersalne zastosowanie tego rozwiązania w systemach zintegrowanych.

ŹRÓDŁA INTERNETOWE

1. *ATmega8/L datasheet – Microchip Technology*, https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf (dostęp: 04.02.2021).
2. *CD4051B, CD4052B, CD4053B datasheet – Texas Instruments*, https://datasheet.octopart.com/CD4051BE-Texas-Instruments-datasheet-7280354.pdf (dostęp: 06.02.2021).

Piotr STOREK, Karol SIWIEC
Rafał NAZARKO, Michał BAZAN

dr inż. Bartosz PAWŁOWICZ
opiekun naukowy

BADANIE DRONA JEDNOSILNIKOWEGO

Artykuł opisuje sposób wykonania i zasadę działania drona jednosilnikowego. Jest to specjalny rodzaj statku powietrznego, który docelowo miałby być przeznaczony do inspekcji trudnodostępnych miejsc dzięki swoim niewielkim rozmiarom. W naszym przypadku sterowanie odbywa się za pomocą odpowiedniej regulacji przepływu powietrza, dzięki czemu statek powietrzny może być kontrolowany w każdej osi. Celem badania jest skonstruowanie i przeprowadzenie testów omawianego bezzałogowego statku powietrznego pod kątem zredukowania liczby silników bezszczotkowych BLDC oraz zminimalizowania kosztów budowy takiego statku powietrznego.
Słowa kluczowe: bezzałogowy statek powietrzny, dron, silniki BLDC.

WPROWADZENIE

W ostatnich latach coraz większy udział na rynku stanowią bezzałogowe statki powietrzne. Popularność dronów wzrasta przez wzgląd na coraz łatwiejsze możliwości sterowania oraz na bardzo dużą liczbę zastosowań. Swoje zadania wykonują, pomagając ratownikom medycznym przy poszukiwaniu zaginionych osób szczególnie w terenach wysokogórskich, dzięki użyciu specjalnych kamer termowizyjnych o dużej rozdzielczości, w rolnictwie jako źródła do zapobiegania przesuszeniu dzięki użyciu kamer multispektralnych, w wojsku do lokalizowania i likwidowania celów, w przemyśle do monitorowania stanu linii energetycznych i wiele więcej. Zastosowanie statków powietrznych jest nieograniczone i zależne tylko i wyłącznie od ludzkiej wyobraźni oraz ograniczeń sprzętowych, jakimi są przykładowo źródła zasilania bezzałogowców. Na przestrzeni lat, dzięki spopularyzowaniu odbiorników GPS, coraz dokładniejszym barometrom i układom żyroskopowym, drony są w stanie określać swoją pozycję z dokładnością rzędu centymetrów.

Sterowanie może odbywać się nie tylko na podstawie zmiany prędkości obrotowej silników, ale także na podstawie zmianie przepływu powietrza, tak jak w naszym przypadku, dzięki zmianie położenia kątowego serw, na których łopatki bezpośrednio wyprofilowane sterują obrotem silników i przechyleniem drona.

Celem naszego badania jest dokładna analiza i możliwość wykorzystania bezzałogowego statku powietrznego przy zastosowaniu przemysłowym. Dzięki badaniom możemy określić stosunek bezzałogowych statków powietrznych wielowirnikowych do statku jednosilnikowego.

Metody badawcze, które zostały przeprowadzone, to swobodny zawis bezzałogowego statku powietrznego oraz sprawdzenie sterowalności pod kątem dokładnego przemieszczania poprzez odpowiednie wysterowanie przepływu powietrza.

1. BUDOWA BEZZAŁOGOWEGO STATKU POWIETRZNEGO

Do budowy naszego bezzałogowego statku powietrznego użyte zostały następujące podzespoły: silniki BLDC brother hobby 1700 KV, regulatory odpowiednie do silników, PDB (ang. *Power Distribution Board*), akumulator 3s 400mAh, transmiter RC, odbiornik RCL9R FRSKY. Funkcje ramy spełnia, odpowiednio przystosowany do naszych potrzeb, a następnie wydrukowany, ogólnie dostępny model 3D.

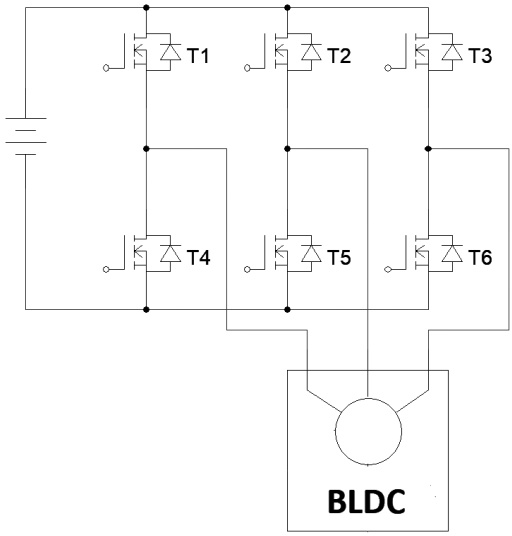
Do prototypu został wykorzystany silnik trójfazowy bezszczotkowy BLDC brother hobby 1700 KV. Silniki te cieszą się ciągle zwiększającą się popularnością w przemyśle przez wzgląd na bardzo dobre parametry i stosunkowo długą żywotność. Charakteryzują się dokładną sterowalnością. Do najważniejszych zalet takich silników należą między innymi:

- brak łuku elektrycznego, dzięki czemu silniki nie wydzielają iskier,
- duża sprawność sięgająca nawet 95%,
- względnie niskie natężenie hałasu, które pozwala na łatwiejsze dostosowanie do określonych warunków pracy silnikowej,
- kontrola prędkości obrotowej prawie niezależnie od momentu silnika,
- stosunkowo długa żywotność oraz trwałość, dzięki wyeliminowaniu z konstrukcji silnika szczotek.

Sterowanie silnikami trójfazowymi bezszczotkowymi BLDC odbywa się poprzez przepływ prądu przez połączone uzwojenia stojana, każde z tych uzwojeń wytwarza pole elektromagnetyczne, które oddziałuje z magnesami umieszczonymi na rotorze, powodując obrót wirnika. Kierunek wirowania oraz prędkość obrotu uwarunkowane są od częstotliwości i kolejności załączania sześciu tranzystorów mocy typu MOSFET. Ideowy schemat podłączenia silnika BLDC przedstawiono na rys. 1. W naszym przypadku do sterowania silnikiem BLDC wykorzystaliśmy regulator ESC (ang. *Electronic Speed Control*) umożliwiający płynną zmianę prędkości oraz zmianę kierunku wirowania. Podczas pracy silnika BLDC, w tym samym momencie, zasilane są tylko dwie fazy, które sterowane są przez dwie pary przeciwległych tranzystorów. W silnikach BLDC istotną rolę odgrywają czujniki położenia, oparte na zjawisku Halla, enkoderach lub resolverach,

gdyż układ sterownika musi posiadać informację o chwilowym położeniu wirnika względem uzwojeń stojana.

Rys. 1. Rzeczywisty model bezzałogowego statku powietrznego przeznaczony do badań
Źródło: opracowanie własne.



Rys. 2. Ideowy schemat sterowania silnikiem trójfazowym BLDC
Źródło: opracowanie własne.

Do dystrybucji napięcia zasilania poszczególnych elementów użytych do budowy bezzałogowego statku powietrznego wykorzystana została płytka dystrybucji PDB (ang. *Power Distribution Board*). Na wejście układu płytki może zostać podłączone napięcie od 6 do 27 V. Dzięki zastosowaniu płytki PDB zmniejszyliśmy ilość zastosowanych obwodów zasilających, gdyż płytka ta posiada dwie sztuki BEC (ang. *Battery Eliminator Circuit*) dla stabilizowanego napięcia wyjściowego 5 V oraz 3,3 V. Przy napięciu wyjściowym 5 V mamy dostępne maksymalnie 2,5 A, natomiast w przypadku napięcia 3,3 V mamy dostępne maksymalnie 1 A.

Za zasilanie prototypu odpowiada akumulator trzy celowy LiPo o napięciu nominalnym 11,1 V oraz pojemności 400 mAh. Akumulatory litowo-polimerowe to w chwili obecnej jedne z najlepszych akumulatorów na rynku, gdyż charakteryzują się mniejszą wagą w porównaniu z akumulatorami niklowo-metalowo-wodorkowymi bądź akumulatorami niklowo-kadmowymi, posiadają znikome tempo samorozładowania oraz słyną z dużej wydajności prądowej.

Sterowanie naszym bezzałogowym statkiem powietrznym oparte jest na odbiorniku i nadajniku RC. Transmiter RC posiada 16 sterowalnych kanałów, dzięki czemu można sterować dronem wzdłuż każdej osi. Odbiornik FRSKY, użyty do

budowy prototypu, posiada dziewięć sterowalnych kanałów PWM oraz jeden protokół komunikacyjny SBUS. Teoretyczny zasięg dwukierunkowej wymiany parametrów, pomiędzy użytymi przez nas modułami, wynosi od 3 do 5 kilometrów.

Do sterowania przepływem powietrza zostały użyte cztery serwomechanizmy, których napięcie robocze wynosi od 3,5 V do 6 V. Kąt obrotu zastosowanego serwomechanizmu wynosi 180 stopni. Sygnał sterowania PWM pochodzi od naszego odbiornika RC, dzięki czemu można płynnie zmieniać obrót wału naszych serwomechanizmów. Każdy serwomechanizm podłączony jest do oddzielnego kanału odbiornika RC, dzięki czemu możliwe jest sterowanie każdym z serwomechanizmów osobno.

Wykorzystana rama jest w pewnym stopniu zmodyfikowanym ogólnie dostępnym modelem 3D. Modyfikacja polegała głównie na zmianie mocowania serwomechanizmów i dostosowaniu powierzchni górnej do użytego w naszym prototypie PDB. Po wydrukowaniu i skręceniu elementów wraz z elektroniką bezzałogowy statek powietrzny był gotowy do konfiguracji.

2. DOBÓR ODPOWIEDNIEGO REGULATORA DO SILNIKA BEZSZCZOTKOWEGO BLDC

Podczas konstruowania bezzałogowego statku powietrznego niezwykle ważny jest dobór podzespołów elektronicznych. Jednym z takich urządzeń, które jest wręcz niezbędne w konstrukcjach napędzanych silnikiem trójfazowym, jest regulator obrotów – ESC (ang. *Electronic Speed Control*). Rolą omawianego regulatora jest wysterowanie cewek w silniku trójfazowym.

Podczas doboru odpowiedniego regulatora do silnika trójfazowego należy zwrócić uwagę na kilka istotnych cech regulatora. Są to między innymi maksymalny prąd danego regulatora, chłodzenie oraz napięcie zasilania danego regulatora. Przez maksymalny prąd regulatora należy rozumieć prąd, którego natężenie, według danych producenta, dany regulator jest w stanie wytrzymać bez jakiejkolwiek szkody. Istotne jest także, aby dobierać regulator o odpowiednio większej wydajności prądowej, biorąc pod uwagę pobór prądu przez silnik. Przykładowo dla silnika pobierającego maksymalnie 19 A należy wybrać regulator będący w stanie wytrzymać pobór prądu o natężeniu 25 A. W przypadku kolejnej omawianej cechy, jaką jest chłodzenie, należy pamiętać, iż dodatkowe elementy chłodzące, najczęściej radiatory, nie występują we wszystkich regulatorach. Podyktowane jest to brakiem konieczności posiadania dodatkowego chłodzenia przez regulatory o stosunkowo małym prądzie, najczęściej poniżej 30 A. W przypadku regulatorów o większym prądzie możemy, relatywnie często, napotkać na umiejscowienie w ich budowie elementów chłodzących w postaci radiatorów. Jednak bez względu na to czy dany regulator posiada radiator, czy też nie, należy zapewnić odpowiedni przepływ powietrza dla regulatora. W tym celu należy, już podczas projektowania ramy bezzałogowca, zaplanować budowę tak, aby regulator

nie był całkowicie zabudowany oraz aby elementy chłodzące regulatora znajdowały się, o ile jest to możliwe, w miejscu ciągłego obiegu powietrza. Dzięki temu unikniemy niebezpiecznego uszkodzenia regulatora bądź, w najgorszym przypadku, całej konstrukcji. Ostatnią, przytoczoną przez nas cechą jest napięcie zasilania regulatora. Najczęściej jest ono podawane bezpośrednio na regulatorze przykładowo „2-4S", co oznacza że dany regulator może być zasilany akumulatorami dwu, trzy lub czterocelowymi.

Każdy regulator ESC do silników trójfazowych posiada na swoim wyjściu trzy przewody. W momencie gdy po odpowiednim podłączeniu regulatora z silnikiem trójfazowym uruchomimy nasz układ i kierunek obrotu silnika będzie przeciwny do oczekiwanego, należy zamienić ze sobą dowolnie przez nas wybrane dwa przewody, z trzech, łączące regulator z silnikiem trójfazowym. Należy także pamiętać, że o ile zmiana kolejności przewodów znajdujących się na wyjściu regulatora nie uszkodzi go, a jedynie zmieni kierunek obrotu silnika, to jeżeli zmienimy polaryzację przewodów zasilających regulator, zostanie trwale uszkodzony dany regulator ESC.

3. KONFIGURACJA POŁĄCZENIA ODBIORNIKA RC Z TRANSMITEREM RC

Konfiguracja połączenia pomiędzy odbiornikiem RC a transmiterem odbywa się poprzez proces, który nazywany jest bindowaniem. Ważnym aspektem w przypadku próby połączenia jest fakt, iż odbiornik i transmiter muszą posiadać tę samą wersję oprogramowania. W naszym przypadku stosujemy oprogramowanie FCC EU (ang. *Federal Communications Commission European*). Nierozerwalnym elementem konfiguracji jest odpowiednie ustawienie kanałów transmitera RC oraz miksera, który pozwala na połączenie działania dwóch kanałów proporcjonalnych w ten sposób, aby za pomocą serwomechanizmów mieć możliwość sterowania jedną płaszczyzną sterową. Dzięki zastosowaniu miksera możemy połączyć ze sobą kanały. W naszym przypadku połączony jest kanał drugi z czwartym, dzięki czemu możemy sterować przepływem powietrza i wykonywać obrót wokół osi YAW (obrót w osi pionowej). Dzięki takiemu ustawieniu wały serwomechanizmu drugiego oraz czwartego w zależności od przechylenia drążka w prawo lub w lewo stopniowo obracają się niesymetrycznie, ale pod stałym kątem do poziomu ramy bezzałogowego statku powietrznego. Również w przypadku kanału drugiego i czwartego występuje połączenie dzięki mikserowi, kanały te odpowiadają za pochylenie bezzałogowego statku powietrznego (Pitch). Kanał piąty z odbiornika RC przyporządkowany jest bezpośrednio do silnika BLDC, którym płynnie, poprzez zmianę wychylenia lewego manipulatora (Throttle), odpowiedzialnego za otwarcie przepustnicy, jesteśmy w stanie wysterować prędkością obrotową silnika.

Proces sterowania odbywa się za pomocą najpopularniejszego ustawienia drążków transmitera, czyli w MODE 2. Układ drążków jest przyporządkowany

do określonego MODE. W Mode-2 (układzie, który wybraliśmy), ponieważ jest on najbardziej zbliżony do prawdziwego układu lotniczego, na drążku lewym odpowiednio w pionie mamy przepustnicę gazu (Throttle), a w poziomie występuje ster kierunku (Rudder). Na prawym manipulatorze w pionie ustawiona jest możliwość pochylenia statku powietrznego w przód bądź w tył w zależności od położenia względem pilota. W poziomie na prawym manipulatorze ustawiona jest możliwość wykonania ruchu poziomego (Roll), który to odpowiada za przechylenie modelu, odpowiednio z ruchem manetki w lewo bądź w prawo. Uwzględniając nasz silnik trójfazowy bezszczotkowy, teoretyczna aproksymowana, generowana siłą ciągu wynosi 950 g przy maksymalnym poborze prądu rzędu 25 A. Zaplanowany ciąg silników jest duży, co przenosi się bezpośrednio na trudność sterowalności bezzałogowego statku powietrznego. Odpowiednie ustawienie wartości Rate w transmiterze Rc pozwala zachować precyzyjną kontrolę przepustnicy (Throttle) przy delikatnych ruchach, umożliwiając delikatny wzrost wartości wychylenia przepustnicy, dzięki temu bezzałogowiec odpowiednio wolno reaguje na szybką zmianę wartości położenia przepustnicy, co zwiększa sterowalność i stabilność szczególnie potrzebną przy sterowaniu dronem wyposażonym tylko w jeden silnik bezszczotkowy.

4. BADANIE DZIAŁANIA BEZZAŁOGOWEGO STATKU POWIETRZNEGO

Głównym celem stworzenia przez nas jednosilnikowego, bezzałogowego statku powietrznego było przeprowadzenie badania działania omawianego urządzenia. Testy te zostały podzielone na dwie główne części, pierwsza z nich została przeprowadzona w pomieszczeniu, dzięki czemu na testy działania nie wpłynęły czynniki atmosferyczne, przykładowo wiatr. Natomiast druga część badania została już przeprowadzona z uwzględnieniem czynników atmosferycznych – odbyła się na przestrzeni otwartej, z dala od wszelakich budynków.

Podczas pierwszej części jednosilnikowy, bezzałogowy statek powietrzny zachowywał się stosunkowo stabilnie, biorąc pod uwagę fakt, że owa konstrukcja posiadała wyłącznie jeden silnik wytwarzający siłę nośną. Nasz bezzałogowiec bardzo dobrze reagował na zmianę przepustnicy gazu, dzięki temu kontrolowanie wysokości lotu odbywało się bez większych problemów – wznoszenie, jak i opadanie, przebiegało w sposób ciągły i ustabilizowany. Obrót wokół osi YAW, a także przechylenie wokół osi PITCH, przebiegało poprzez odpowiednią regulację strumienia powietrza generowanego przez śmigło silnika, przebiegającego przez łopatki ustawiane przy pomocy micro serwomechanizmów. Przy naszej konstrukcji i odpowiednio dobranym do niej śmigle, maksymalny ciąg wyrażony w gramach wynosi około 800. Biorąc pod uwagę fakt, że wspomniany maksymalny ciąg był osiągany podczas maksymalnego poboru prądu, należy pamiętać, że rzeczywista wartość ciągu wyrażona w gramach, podczas której możliwe będzie sterowanie bezzałogowym statkiem powietrznym, wynosi około 400. Jest to

relatywnie niska wartość, którą w większości posiada już sama konstrukcja wraz z niezbędnymi elementami elektronicznymi. Dodatkowo dołączyliśmy do podstawowego wyposażenia analogowy nadajnik VTX bezpośrednio z kamerą, nadający na częstotliwości 5,8 GHz o mocy 25 mW. Dzięki temu w czasie rzeczywistym bezzałogowiec ma możliwość badania, co znajduje się w pomieszczeniach zamkniętych, w trudno dostępnych miejscach. Poprzez zastosowany odbiornik obraz z kamery pokładowej może być odtwarzany w czasie rzeczywistym na dowolnym urządzeniu posiadającym wejście HDMI (ang. *High Definition Multimedia Interface*).

Kolejnym etapem przeprowadzonych przez nas testów jednosilnikowego, bezzałogowego statku powietrznego była zmiana otoczenia na bardziej otwarte, znajdujące się z dala od wszelakich budowli. Pomimo faktu, iż w pobliżu miejsca testów nie znajdowały się żadne budynki, ani nie przebywały osoby postronne, według aktualnie obowiązujących przepisów nadanych przez Urząd Lotnictwa Cywilnego wymagane było posiadanie przez osoby sterujące bezzałogowcem podstawowych uprawnień umożliwiających loty w kategorii otwartej, podkategorii A1, które dotyczą bezzałogowych statków powietrznych o maksymalnej masie startowej MTOM (ang. *Maximum Take-Off Mass*) nieprzekraczającej 900 gram. Loty na świeżym powietrzu, wedle naszych przewidywań, okazały się względnie trudne – bezzałogowiec był niestabilny, narażony na powiewy nawet lekkiego wiatru, co w znacznym stopniu utrudnia kontrolę nad badanym obiektem.

5. MOŻLIWOŚĆ ROZWINIĘCIA PROJEKTU

Opisany w artykule jednosilnikowy, bezzałogowy statek powietrzny jest urządzeniem, w którym znajduje się wiele elementów nadających się do ulepszenia. Są to przykładowo łopatki, znajdujące się przy serwomechanizmach, które po przeprowadzeniu dodatkowych badań można wymienić na lepiej zaprojektowane i wykonane z lżejszego i trwalszego materiału, przykładowo włókna węglowego. Istnieje także możliwość zminimalizowania ilości serwomechanizmów, poprzez wprowadzenie specjalnie do tego zaprojektowanych przekładni. Zmniejszy to całkowitą masę konstrukcji, umożliwiając przenoszenie przez bezzałogowca dodatkowych elementów zwiększających funkcjonalność urządzenia. Kolejnym ulepszeniem może być zwiększenie siły nośnej poprzez zastosowanie mocniejszego silnika.

6. PODSUMOWANIE

Jednosilnikowy, bezzałogowy statek powietrzny jest urządzeniem, którego konstrukcja posiada niezaprzeczalnie liczne wady. Mimo to urządzenie jest w stanie odbyć relatywnie stabilny lot pod warunkiem, że odbędzie się on w pomiesz-

czeniu zamkniętym. Na podstawie przeprowadzonych testów wynika jednoznacznie, że odbycie lotu z dala od budynków, na przestrzeni otwartej jest bardzo utrudnione, czego powodem jest duża podatność omawianej konstrukcji na czynniki atmosferyczne, których nie sposób uniknąć, przykładowo podmuchy wiatru. Reasumując, cały proces tworzenia konstrukcji jednosilnikowego bezzałogowca objął wiele dziedzin, poprzez projektowanie i druk 3D, aż po samodzielne łączenie elementów elektronicznych.

LITERATURA

1. Ballaster R., Firman Andrew, Clot Eleanor, *A practical Guide to Drone Law*, Wydawnictwo Law Brief Publishing Ltd, 2017.
2. Halliday B., *Drones: Mastering Flight Techniques*, Wydawnictwa CreateSpace Independent Publishing Platform, 2017.
3. Halliday B., *Drones: The Professional Drone Pilot's Manual*, Wydawnictwa CreateSpace Independent Publishing Platform, 2016.
4. Krykowski K., *Silniki PM BLDC: właściwości, sterowanie, aplikacje*, Wydawnictwo BTC, Legionowo 2015.

ŹRÓDŁA INTERNETOWE

5. https://www.thingiverse.com/thing:4635873 (dostęp: 05.05.2021).