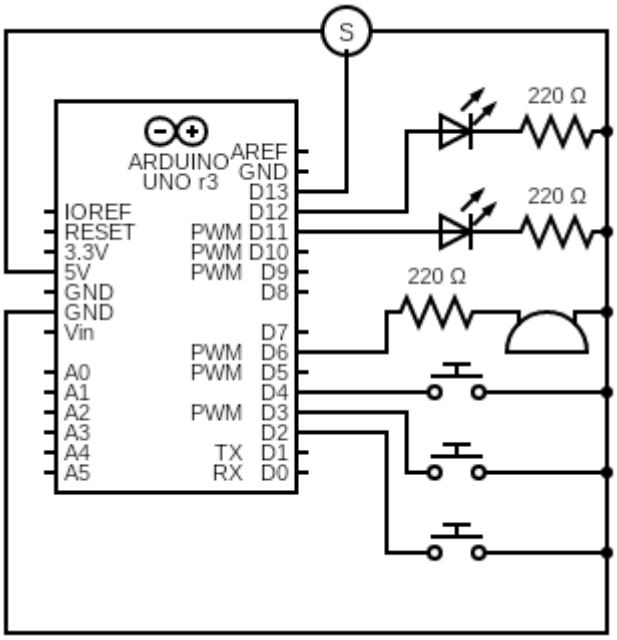
Eksperyment 10: Winda

Autorzy: Dávid Ali, Jan Bensz

Projekt "Winda" to innowacyjna implementacja windy oparta na platformie Arduino. Ta zaawansowana winda nie tylko zapewnia efektywny transport między piętrami, ale także dostarcza informacje o jej stanie za pomocą sygnalizacji wizualnej i dźwiękowej. Projekt ten jest doskonałym przykładem wykorzystania technologii mikrokontrolera do zautomatyzowania codziennych procesów.

Wykorzystane elementy



- **Arduino Uno**: Nasz projekt opiera się na mikrokontrolerze Arduino Uno, który pełni rolę mózgu systemu, zarządzając ruchem windy oraz kontrolując sygnalizację wizualną i dźwiękową.
- **Silnik serwo**: Silnik serwo jest odpowiedzialny za ruch windy. Winda przyspiesza lub zwalnia prędkość w odpowiedzi na sygnały z mikrokontrolera.
- **LEDy**: W projekcie wykorzystywane są dwa LEDy - zielony i czerwony. Zielony LED sygnalizuje, że winda przyspiesza, podczas gdy czerwony LED wskazuje, że winda zwalnia prędkość.
- **Buzzer**: Buzzer dostarcza sygnał dźwiękowy, informując użytkowników, że ich polecenia zostały przyjęte przez system. Dodatkowo, buzzer generuje różne dźwięki na różnych piętrach, aby zasygnalizować przyjazd windy.
- **Guziki**: Dzięki guzikom użytkownicy mogą interaktywnie korzystać z windy, wybierając piętro, do którego chcą się dostać.
- **Oporniki**: Oporniki są używane w projekcie do ochrony przed przepięciami oraz do dostosowania poziomów napięcia w obwodzie.
- **Okablowanie**: Okablowanie to kluczowy element projektu, łączący wszystkie komponenty i umożliwiający przesyłanie sygnałów między nimi. W projekcie konieczne jest odpowiednie okablowanie, które łączy Arduino, silnik serwo, LEDy, buzzer, guziki oraz oporniki w jeden spójny system.

Funkcje i działanie projektu

- **Przyśpieszanie i zwalnianie**: Nasza winda jest zdolna do dynamicznego przyspieszania i zwalniania w odpowiedzi na polecenia użytkownika.
- **Sygnalizacja wizualna**: Zielony i czerwony LED służą do wizualnego informowania o stanie windy. Zielony LED sygnalizuje przyspieszanie, a czerwony LED - zwalnianie.
- **Sygnalizacja dźwiękowa**: Buzzer dostarcza sygnał dźwiękowy, informując użytkowników o przyjęciu polecenia. Dodatkowo, generuje różne dźwięki na różnych piętrach, pomagając pasażerom zorientować się, że winda jest gotowa do przyjęcia.
- **Interfejs użytkownika**: Nasz projekt umożliwia obsługę za pomocą guzików, co pozwala pasażerom na wybór docelowego piętra i inicjowanie ruchu windy.

Kod

```
#include <Servo.h>


#define LEDZWOLNIENIA 12
#define LEDPRZYSPIESZENIA 11
#define BUZZER 6
#define PRZYSPIESZENIE .05
```

Wykorzystujemy bibliotekę `Servo` do kontroli silnika serwo. Definiujemy stałe, takie jak piny LEDów, pin dla buzzera oraz wartość stałoprzyspieszeniową (`PRZYSPIESZENIE`).

```
const int guzik[] = {2, 3, 4};
```

Definiujemy guziki (`guzik[]`) oraz przypisujemy im konkretne piny na Arduino.

```
const int pozycjaPieter[] = {20, 100, 180};
const int frekwencjaSygnalu[] = {262, 330, 392};
```

Określamy położenia pięter w zmiennej `pozycjaPieter[]` oraz częstotliwości dźwięków dla sygnalizacji dźwiękowej w `frekwencjaSygnalu[]`.

```
class Wezel {
public:
  int wartosc;
  Wezel* nastepny;
  Wezel(int val) :
    wartosc(val), nastepny(nullptr) {}
};
```

Tworzymy klasę `Wezel` do reprezentowania elementu kolejki.

```
class Kolejka {
public:
  Kolejka() {
    _poczatek = nullptr;
    _koniec = nullptr;
  }

  ~Kolejka() {
    while (!pusta()) {
      wyjmij();
    }
  }

  bool pusta() {
    return _poczatek == nullptr;
  }

  void wloz(int element) {
    Wezel* nowyWezel = new Wezel(element);
    if (pusta()) {
      _poczatek = nowyWezel;
      _koniec = nowyWezel;
    } else {
      _koniec->nastepny = nowyWezel;
      _koniec = nowyWezel;
    }
  }

  int wyjmij() {
    if (pusta()) {
      return 0; // Kolejka jest pusta
    }
    int element = _poczatek->wartosc;
    Wezel* temp = _poczatek;
    _poczatek = _poczatek->nastepny;
    delete temp;
    return element;
  }

private:
  Wezel* _poczatek;
  Wezel* _koniec;
};
```

Klasa `Kolejka` jest używana do przechowywania poleceń od użytkowników. Odpowiada za dodawanie i usuwanie poleceń.

```
class Winda{
  Servo* serwo;
  int pietroAktualne;
  float przyspieszenie;
  float predkosc;
  float pozycjaSerwo;
  int kierunek;
  int ostatniGuzik;
  public:
  Kolejka polecenia;
  int pietroDocelowe;
  int brzeczyk;
  Winda(){
    pietroAktualne = 0;
    pietroDocelowe = 0;
    przyspieszenie = 0;
```

Nasza klasa zarządza główną logiką projektu windy.

```
  predkosc = 0;
  pozycjaSerwo = pozycjaPieter[pietroAktualne];
  ostatniGuzik=-1;
  brzeczyk=-1;
};
```

Metoda `dolaczSerwo` pozwala nam podłączyć serwo do windy.

```
void dolaczSerwo(Servo* serwoDoDolaczania){
 serwo = serwoDoDolaczania;
 serwo-> write( (int)pozycjaPieter[pietroAktualne]);
 delay(500);
}
```

Metoda `dolaczBrzeczyk` pozwala na podłączenie buzzera.

```
void dolaczBrzeczyk(int numerPinu){
 brzeczyk = numerPinu;
}
```

Metoda `jedz` odpowiada za poruszanie się windy z uwzględnieniem przyspieszania i zwalniania.

```
void jedz(){
  kierunek = (pietroDocelowe > pietroAktualne) –
             (pietroDocelowe < pietroAktualne);
  while(kierunek != 0){
    if(czyDopieroStartuje()){
      przyspiesz();
      digitalWrite(LEDPRZYSPIESZENIA, HIGH);
      digitalWrite(LEDZWOLNIENIA, LOW);
    } else if(czyJuzDojezdza(pietroDocelowe)) {
      digitalWrite(LEDPRZYSPIESZENIA, LOW);
      zwolnij();
      digitalWrite(LEDZWOLNIENIA, HIGH);
    } else {
      digitalWrite(LEDPRZYSPIESZENIA, LOW);
      digitalWrite(LEDZWOLNIENIA, LOW);
      trzymajPredkosc();
    }

    predkosc += przyspieszenie;
    pozycjaSerwo += predkosc;
    serwo->write((int)pozycjaSerwo);
    if(czyDojechal(pietroDocelowe)){
      Serial.println("");
      pietroAktualne = pietroDocelowe;
      Serial.println("Dojechałem");
      predkosc = 0;
    }
    for(int i=0;i<30;i++){
      obslugujGuziki();
      delay(1);
    }
    kierunek = (pietroDocelowe > pietroAktualne) -
               (pietroDocelowe < pietroAktualne);
  }
  digitalWrite(LEDPRZYSPIESZENIA, LOW);
  digitalWrite(LEDZWOLNIENIA, LOW);
  sygnalizujPrzyjazd();
}
inline void sygnalizujPrzyjazd(){
  if(brzeczyk!=-1){
    tone(brzeczyk, frekwencjaSygnalu[pietroAktualne]);
    delay(500);
    noTone(brzeczyk);
  }
}
```

Istnieje wiele funkcji `inline`, które sprawdzają różne warunki i sterują prędkością i przyspieszeniem windy.

```
inline int czyDopieroStartuje(){
  return abs(pozycjaPieter[pietroAktualne]-pozycjaSerwo) < 40;
}
inline int czyJuzDojezdza(int pietroDocelowe){
  return abs(pozycjaPieter[pietroDocelowe]-pozycjaSerwo) < 40;
}
inline int czyDojechal(int pietroDocelowe){
  return kierunek*(pozycjaPieter[pietroDocelowe]-pozycjaSerwo) <=1;
}
inline void przyspiesz(){
  przyspieszenie = kierunek * PRZYSPIESZENIE;
}
inline void zwolnij(){
```

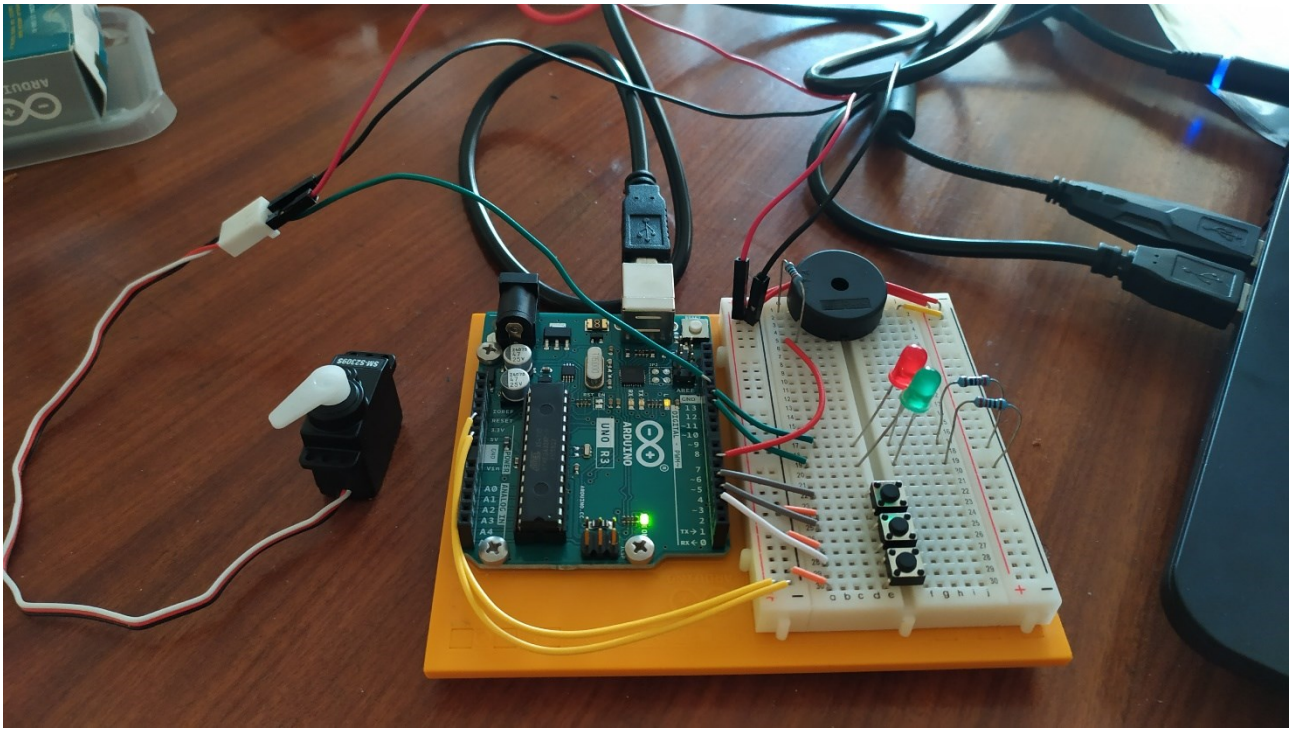
```
    przyspieszenie = -kierunek * PRZYSPIESZENIE;
  }
  inline void trzymajPredkosc(){
    przyspieszenie = 0;
  }

  void obslugujGuziki(){
    for(int i=0;i<3;i++){
      if(digitalRead(guzik[i])==LOW){
        if(ostatniGuzik != i) {
          polecenia.wloz(i);
          ostatniGuzik = i;
          tone(BUZZER,frekwencjaSygnalu[i]/2);
          delay(100);
          noTone(BUZZER);
        }
      }
    }
  }
};
Servo serwo;
Winda winda;
void setup() {
  Serial.begin(9600);
  serwo.attach(13);
  winda.dolaczSerwo(&serwo);
  winda.dolaczBrzeczyk(BUZZER);
  for(int i=0;i<3;i++){
    pinMode(guzik[i], INPUT_PULLUP);
  }
  pinMode(BUZZER, OUTPUT);
  pinMode(LEDZWOLNIENIA, OUTPUT);
  pinMode(LEDPRZYSPIESZENIA, OUTPUT);
}
void loop() {
  winda.obslugujGuziki();
  if(!winda.polecenia.pusta()){
    winda.pietroDocelowe = winda.polecenia.wyjmij();
    winda.jedz();
    for(int i=0;i<100;i++){
      winda.obslugujGuziki();
      delay(20);
    }
  }
}
```

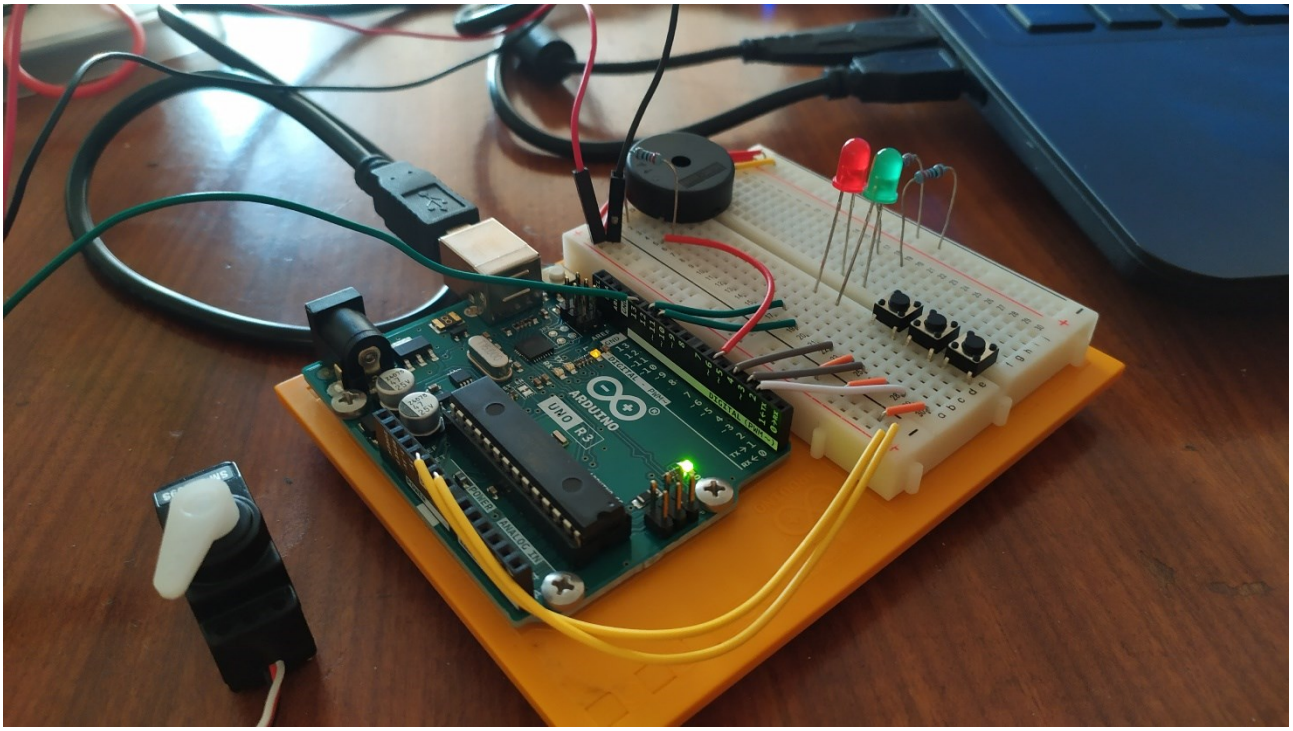
Metoda `obslugujGuziki` monitoruje naciśnięcia guzików i dodaje odpowiednie polecenia do kolejki poleceń.

W `setup`, inicjalizowane są piny guzików, buzzer, LEDy i serwo.

W `loop`, odczytujemy naciśnięcia guzików, a następnie wykonywane są odpowiednie operacje w przypadku wykrycia poleceń.

Zdjęcia





Miejsca rozwoju

1. Zaimplementowanie stanu drzwi w projekcie windy to ważny krok w kierunku zwiększenia bezpieczeństwa i realizmu systemu.
2. Zamiast standardowej kolejki, wykorzystanie metodyki bardziej przypominającej zachowanie windy może uczynić projekt bardziej realistycznym i elastycznym. Można stworzyć stanowy automat, który reprezentuje różne etapy pracy windy, takie jak przyspieszanie, utrzymywanie stałej prędkości, zwalnianie i zatrzymywanie. Każdy stan może reprezentować różne zachowania windy, a przejścia między nimi mogą odpowiadać na różne sytuacje, takie jak naciśnięcie guzika na danym piętrze.
   To podejście pozwoli na lepszą reprezentację rzeczywistego zachowania windy i może być bardziej elastyczne w przypadku przyszłych rozbudów. Można byłoby łatwiej dodawać nowe stany i definiować, jak winda ma zachowywać się w każdym z nich. Na przykład, mogliśmy dodać stan "Awaria" lub "Serwis", który umożliwia obsługę sytuacji wyjątkowych.
   Jednak to podejście wymaga bardziej zaawansowanego programowania i zarządzania stanami, ale może przynieść znacznie bardziej realistyczne i elastyczne efekty. To doskonały przykład na to, jak można dalej rozwijać projekt i uczynić go jeszcze bardziej zaawansowanym.
3. Dodanie guzików zewnętrznych, które pozwalają na przywołanie windy w określonym kierunku, znacznie zwiększa funkcjonalność projektu i zapewnia większą wygodę pasażerom. To wspaniała rozbudowa, która uczyni projekt bardziej użytecznym i dostosowanym do rzeczywistych potrzeb użytkowników.