Oliwier Pszeniczko – Projekt

RC-Robot Car

Rozdział I

Tematem projektu jest budowa oraz zaprogramowanie systemu pojazdu charakteryzującego się pracą w dwóch niesymultanicznych trybach tzn. autonomicznym w którym to przemieszczanie pojazdu uwarunkowane jest poprzez otoczenie i przeszkody, na które może natrafić pojazd. Dodatkowo pojazd w tym trybie realizuje rolę sondy badawczej dokonując pomiarów temperatury powietrza w otoczeniu po każdym napotkaniu przeszkody i informując operatora w razie niespodziewanie wysokich wskazań tego pomiaru.

Drugi tryb, czyli praca pojazdu w trybie zdalnego sterowania umożliwiająca operatorowi pełną kontrolę nad procesem poruszania się pojazdu przy pomocy specjalnie do tego zadania skonstruowanego kontrolera bezprzewodowego. Bieżący tryb pracy jest w pełni zależny od wyboru operatora.

Rozdział II

W projekcie jako jednostki centralne stanowiące trzon w pełni działającego systemu zostały wykorzystane mikrokontrolery Arduino Uno pełniący tę rolę dla systemu pojazdu oraz Arduino Nano dla systemu kontrolera bezprzewodowego. Napięcie do obu tych niezależnych systemów doprowadzone zostało z zewnętrznych źródeł zasilania o wysokości 6 oraz 9V. Komunikacja między tymi podsystemami zapewniona została z wykorzystaniem modułu radiowego nRf24L04 zasilanego napięciem o wysokości 3.3V działającego w trybie komunikacji dwukierunkowej.

Przemieszczanie pojazdu odbywa się poprzez wprawienie w ruch czterech gumowych kół przyłączonych do karoserii pojazdu przy udziale w sposób bezpośredni silników prądu stałego. Za ich obsługę odpowiedzialny jest dwukanałowy moduł sterownika silników L298N, Ten fragment systemu również został zaopatrzony w dodatkowe źródło zasilania w wysokości 9V.

Pojazd został wyposażony w ultradźwiękowy czujnik odległości HC-SR04 przymocowany do serwomechanizmu, cyfrowy czujnik temperatury DS18B20 przyłączony poprzez rezystor o rezystancji 4.7 kOhm. Te komponenty w sposób bezpośredni wpływają na zachowanie systemu, podejmując interakcje z otoczeniem determinują pewne zachowania pojazdu w konkretny sposób, w zarówno autonomicznym trybie pracy jak i zdalnego sterowania.
To znaczy kolejno, czujnik HY-SRF05 odpowiedzialny za wyliczenie odległości do przeszkody, w przypadku konieczności zmiany trasy jazdy to znaczy wykryciu w odległości 40cm w linii prostej od pojazdu

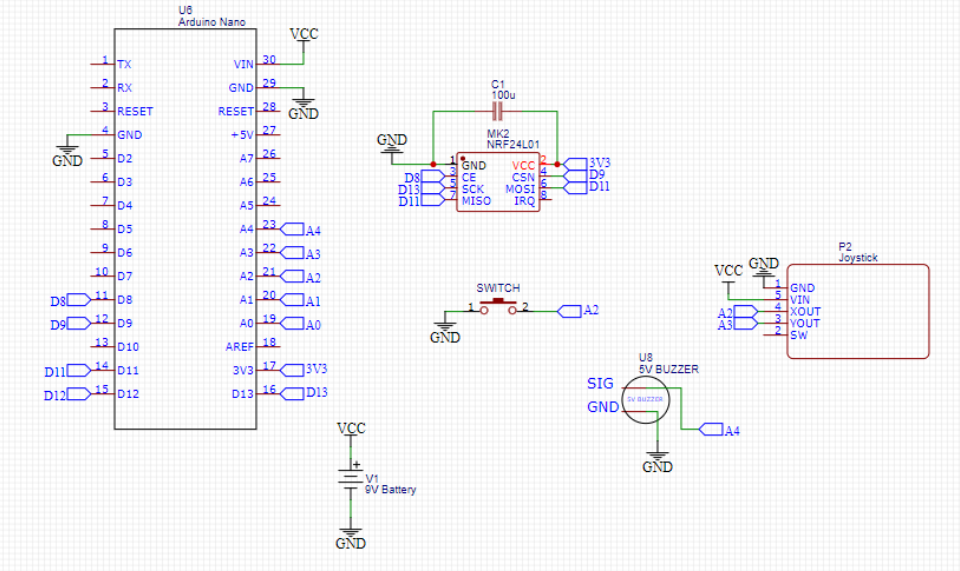
przeszkody, spowoduje jego zatrzymanie i dzięki obrotu serwomechanizmu w zakresie 120 stopni i tym samym umożliwieniu dalszego dokonania pomiaru czujnikowi ultradźwiękowemu, na podstawie którego to zostanie wybrany nowy, korzystniejszy pod względem odległości od kolejnej przeszkody kierunek jazdy.

Z kolei rolą czujnika DS18B20 będzie monitorowanie i ewentualne informowanie o wysokim tzn. Przekraczającym 30 stopni w skali Celsjusza pomiarze.

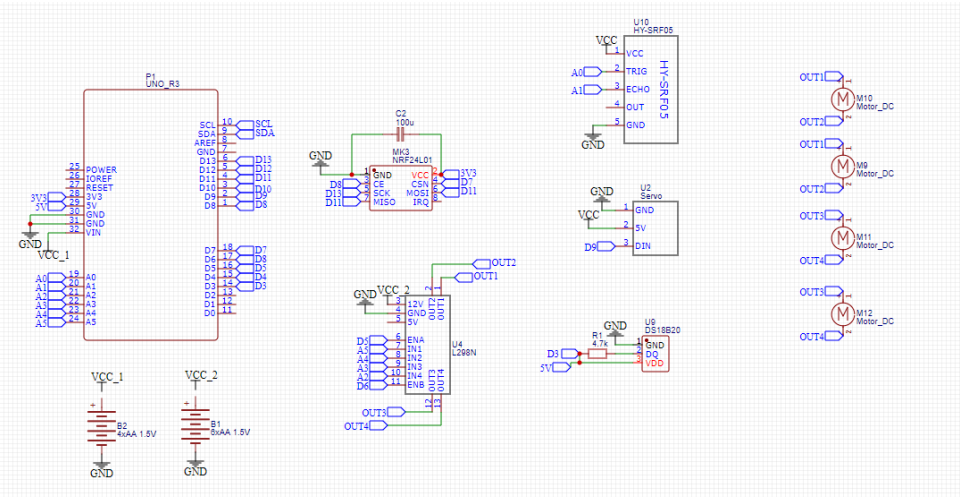
Do ich zasilenia wykorzystano napięcie o wysokości 5V.

Budowa kontrolera bezprzewodowego została przeprowadzona na osobnej płytce uniwersalnej prototypowej. Po za mikrokontrolerem w skład tych podzespołów wchodzą, joystick zasilany napięciem w wysokości 5V, odpowiedzialne za realizacje poruszania pojazdem w dwóch osiach z uwzględnieniem stopnia odchylenia gałki przekładającej się odpowiedni stopień modulacji sygnału PWM po stronie kontrolera silników w trybie pracy zdalnego sterowania, brzęczyk wydający głośny, jednostajny dźwięk w przypadku, gdy na system kontrolera ze strony systemu pojazdu trafi stosowna informacja docierająca z pomiaru czujnika temperatury oraz przełącznik typu tact, służący do wyboru pomiędzy trybami pracy systemu zmieniający ten tryb w przypadku odczytu zmiany jego stanu cyfrowego.

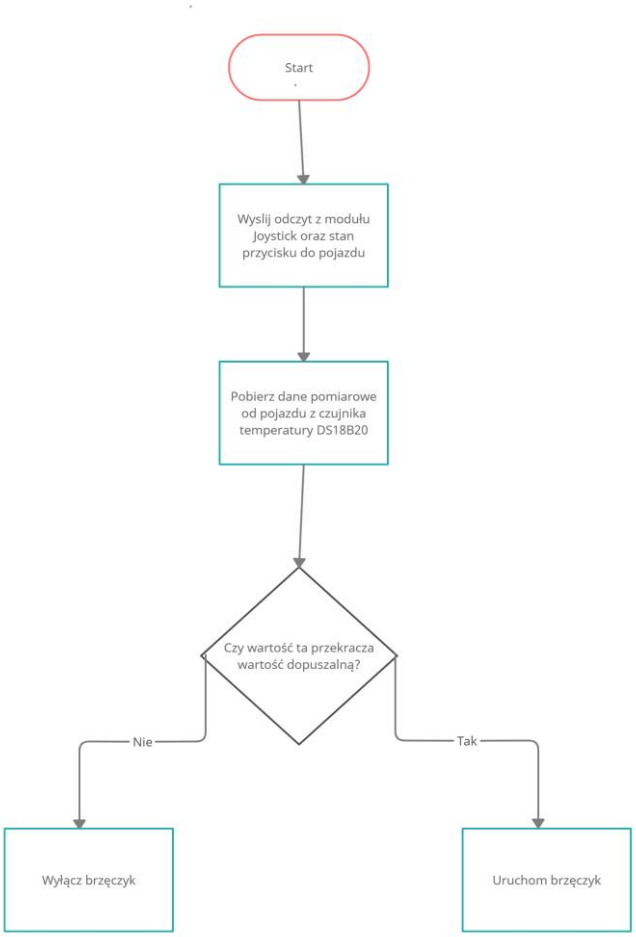
Schemat logiczny Kontrolera



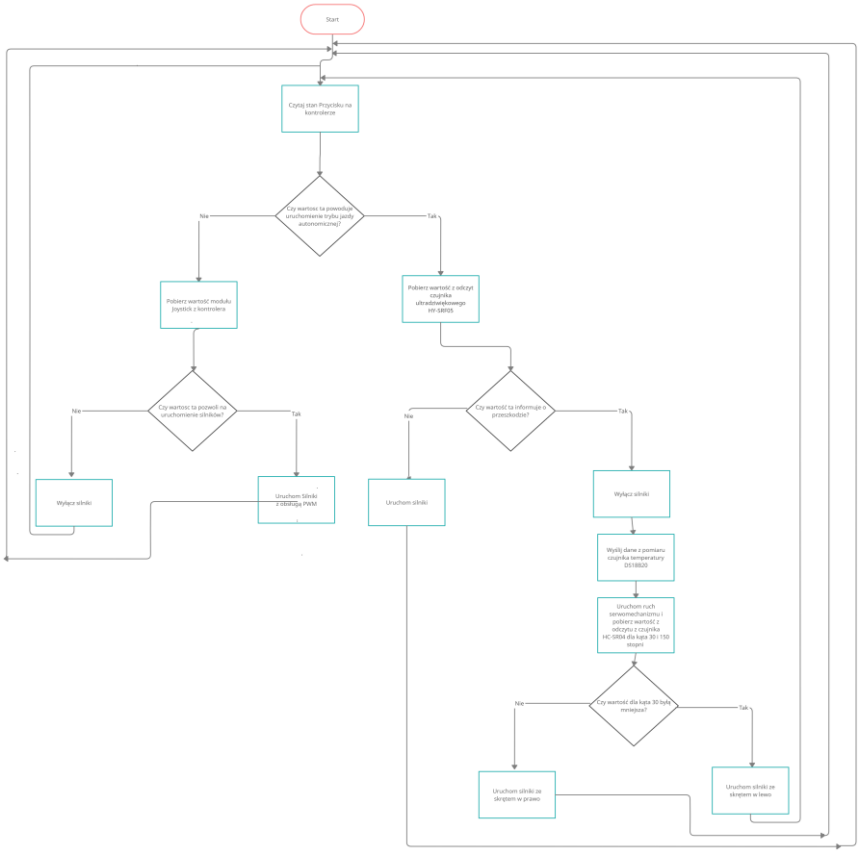
Schemat logiczny Pojazdu



Schemat blokowy Kontrolera



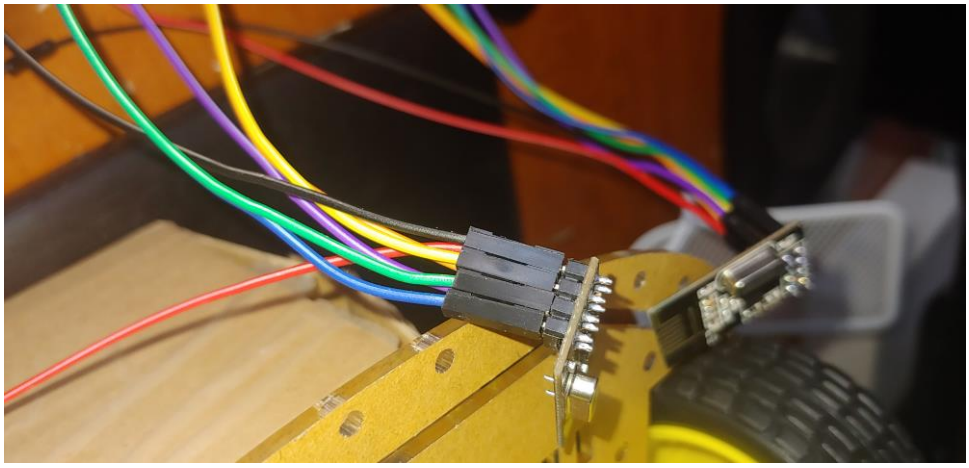
Schemat blokowy Pojazdu



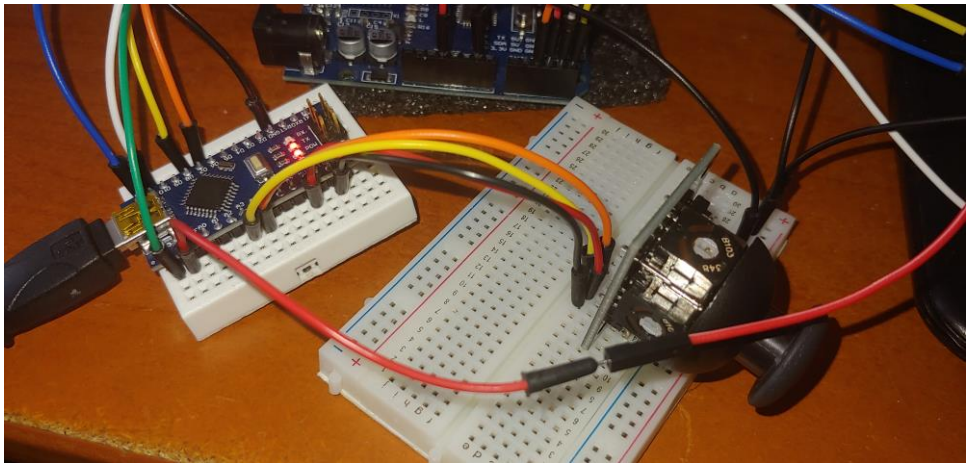
Rozdział III - Przebieg realizacji projektu

Konstrukcja sprzętowa systemu

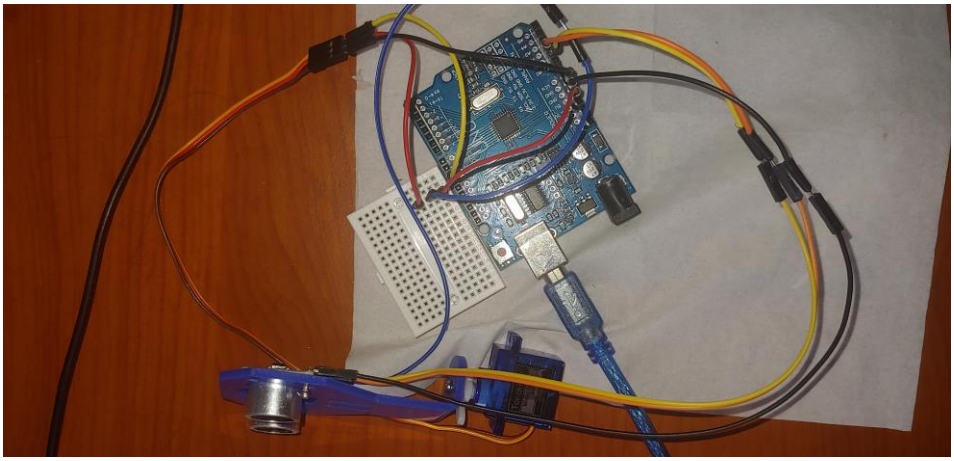
Przed przystąpieniem do konstrukcji właściwej przedmiotu projektu poszczególne komponenty wchodzące w jego skład zostały poddane szeregowi testów, kalibracji i poprawek. Pomocne okazały się przy tym wszelkiego rodzaju skonstruowane na tę potrzebę jednostki testowe sprawdzające poszczególne komponenty, pod kątem sposobu ich działania, interakcji z pozostałymi komponentami wchodzących w skład systemu wraz z wyszukaniem ewentualnych nieprawidłowości pozwalających na dokonanie działań prowadzących do ich eliminacji. Poniżej zostały przedstawione przykłady takich testów.



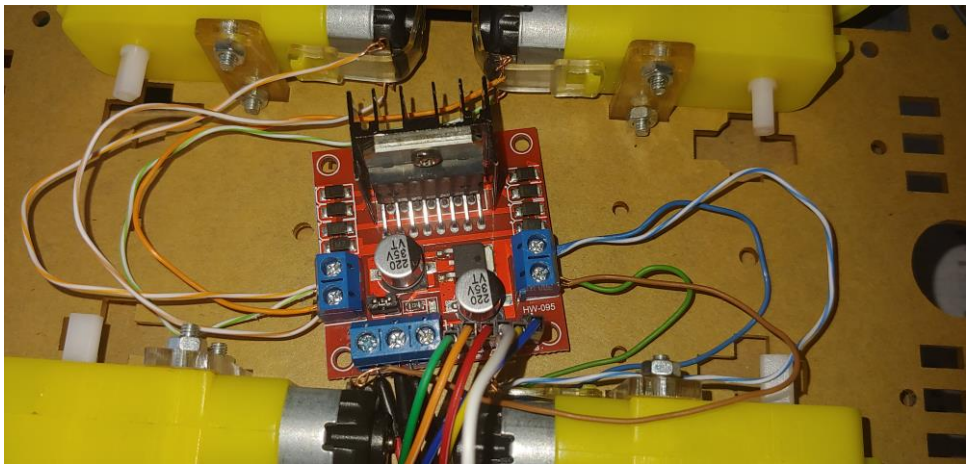
komunikacji pomiędzy modułami NRF24L01



połączenie mikrokontrolera z modułem joystick



połączenie czujnika ultradźwiękowego oraz serwomechanizmu

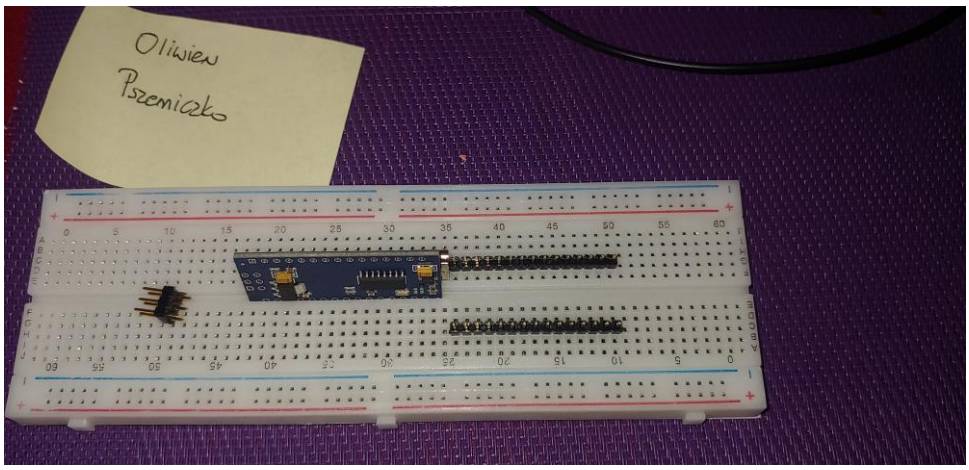


obsługi silników przy pomocy sterownika L298N przy wykorzystaniu żył pochodzących z kabla ethernetowego

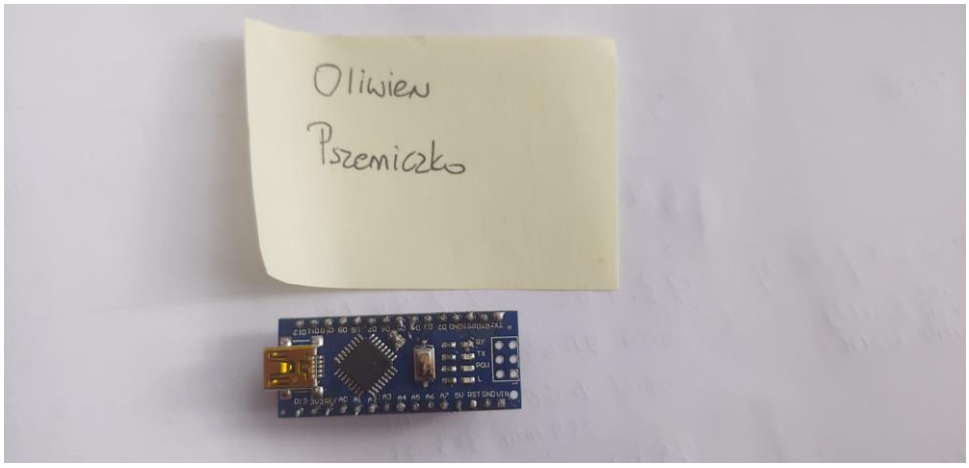
W części właściwej konstrukcji zostały wykorzystane takie czynności modelarskie jak, lutowanie z wykorzystaniem lutownicy kolbowej o mocy 40W, łączenie ze sobą elementów przy użyciu pistoletu do kleju na gorąco i oczywiście skręcanie pomniejszych elementów przy użyciu śrubokręta. Szereg wyszczególnionych poniżej czynności doprowadziły do uzyskania konstrukcji zdanej do niezakłóconej jazdy.

Proces konstrukcyjny Kontrolera

Jako mikrokontroler obsługujący podsystem kontrolera bezprzewodowego wykorzystany został samodzielnie zlutowany model Arduino Nano z rzędami goldpinów.



widok na płytkę oraz rzędy pinów przed ich przylutowaniem

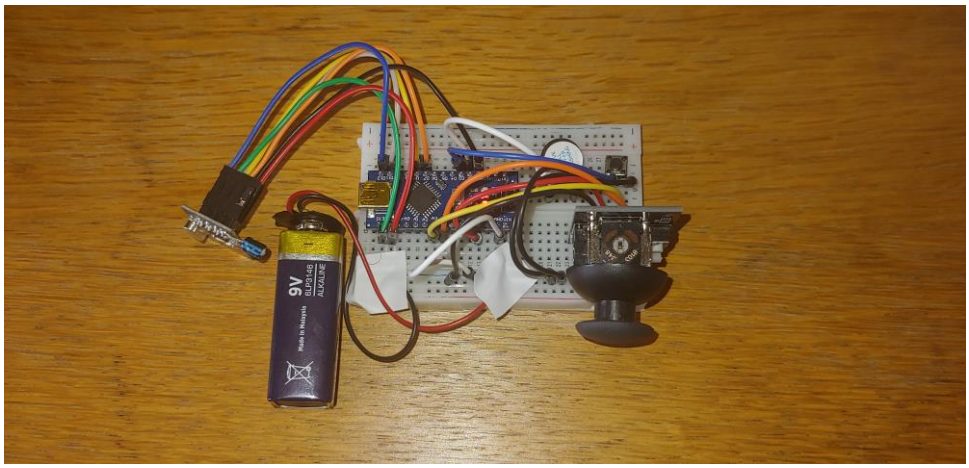


Widok na zlutowany komponent

Do modułu radiowego w celu zapewnienia lepszej jakości oraz większej niezawodności wlutowany równoległe pomiędzy złączem zasilania a masą kondensator 10 uF. Analogiczna operacją wykonana zostaje w przypadku modułu połączonego do systemu pojazdu.

Widok na moduł radiowy po wykonanym przylutowaniu kondesatora

W ramach kontrolera na płytce według schematu zostają umieszczone wcześniej wymienione elementy.

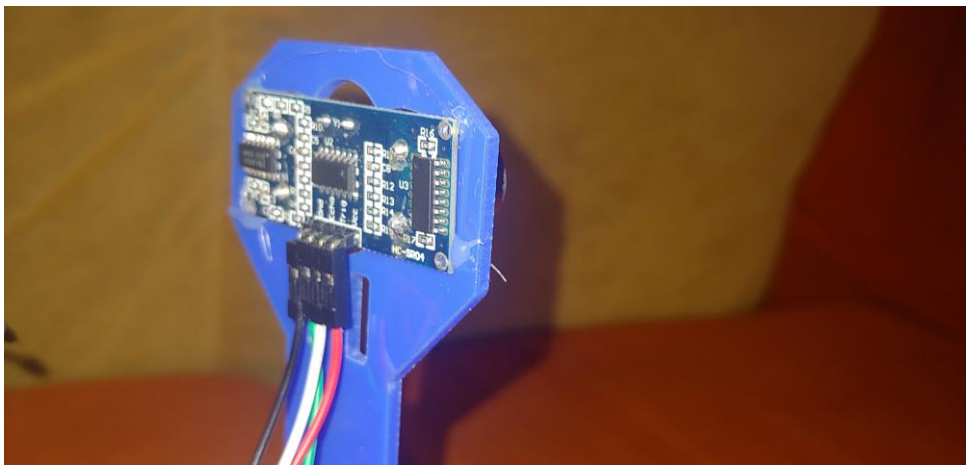


Widok na gotowy kontroler

Proces konstrukcyjny Pojazdu

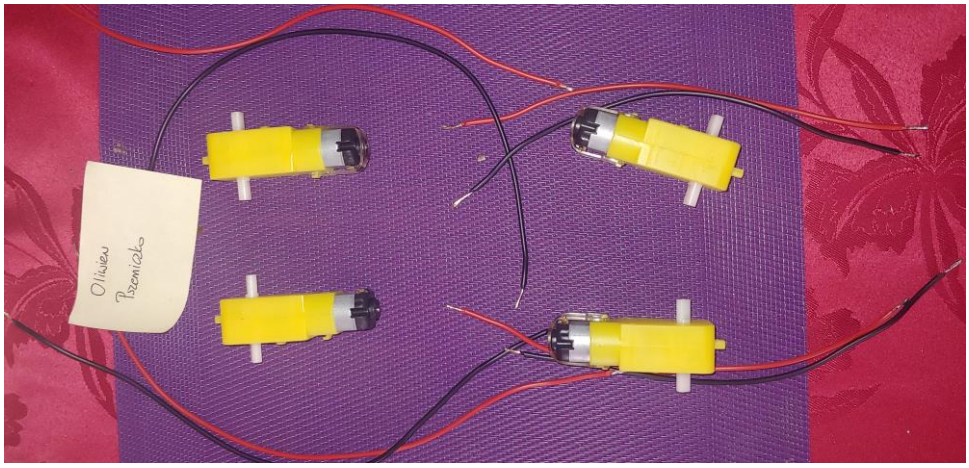
Serwomechanizm wraz czujnikiem ultradźwiękowym złączone zostały poprzez zakupiony do tego celu uchwyt. Po stronie serwomechanizmu został on przykręcony do orczyka poprzez zamieszony w komplecie wkręt. Z kolei sam czujnik został sklejony z uchwytem przy pomocy kleju na gorąco.

widok na elementy wchodzące w skład podsystemu wykrywania przeszkód przed połączniem

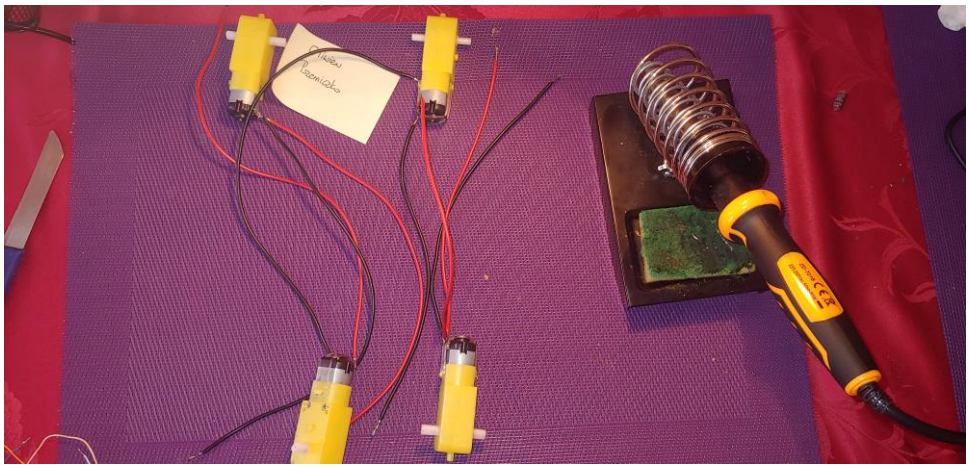


widok po sklejeniu czujnika z uchwytem

Do silników prądu stałego zostały przylutowane wcześniej pobielone przewody instalacyjne doprowadzające do nich zasilanie.

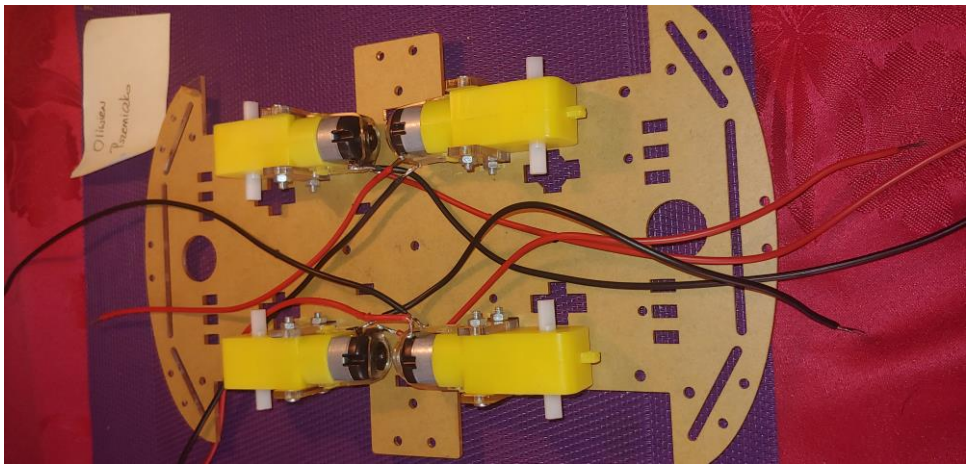


widok przed przylutowaniem przewodów do silników



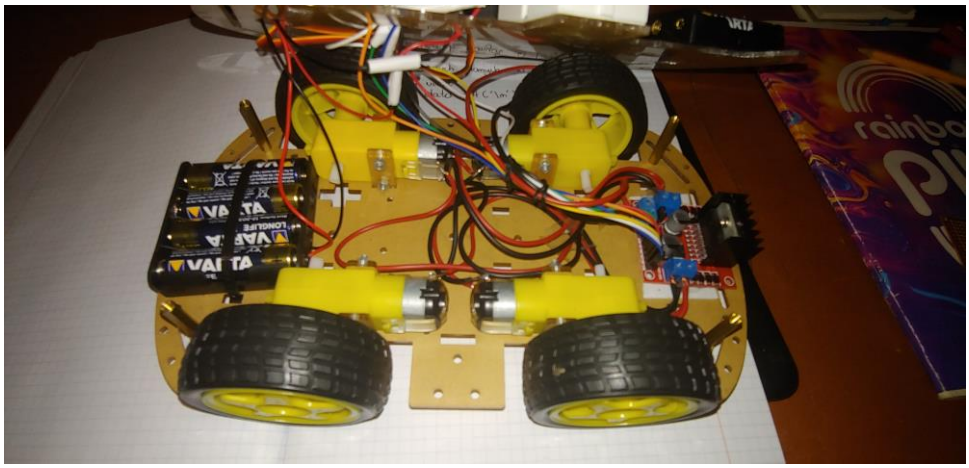
widok po przylutowaniu przewodów do silników

Silniki została umieszczone i zamocowana przy użyciu znajdujących się w komplecie elementów połączeniowych dołączonych do zakupionego stelażu.



widok po zamontowaniu silników do stelaża

Dolny poziom pojazdu uzupełniony został o kontroler silników L298N do którego doprowadzone zostały przewody wychodzące od silników oraz baterię 4x1.5V zasilającą płytkę Arduino Uno znajdującą się z kolei na górnym poziomie.

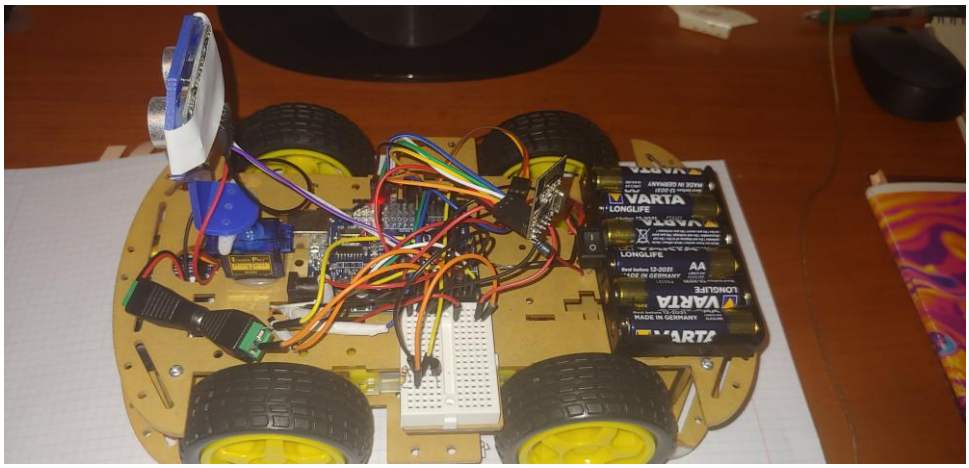


widok na dolny poziom pojazdu

Na górnej powierzchni pojazdu znalazł się mikrokontroler Arduino wraz małą 170 polową płytką stykową przez którą połączony został czujnik temperatury DS18B20.

Po za tym znajduje się tam pojemnik z sześcioma bateriami R6 AA z przyłączonych do wyjścia napięcia przełącznikiem kołyskowym i stanowiącym zasilanie dla silników prądu stałego.

Na froncie pojazdu osadzony został czujnik ultradzwiękowy na wieżyczce z serwomechanizmu i uchwytu.



widok na górny poziom pojazdu

.

Oprogramowanie dla systemu

Do realizacji projektu od strony programistycznej wykorzystane zostało dedykowane w tym celu popularne środowisko, Arduino IDE.

Program został napisany z poszanowaniem takich paradygmatów programowania jak modularność, funkcyjność. Został on rozbity na dwa modułu z każdym przypadającym do obsługi danego podsystem z całości. Mnogość zdefiniowanych procedur dla fragmentów kodu wykonywanych w sposób cykliczny pozwoliło na pozostawienie głównej procedury loop w kompaktowym i przejrzystym widoku.

Kod pisany był starannie, schludnie, tak aby był w miarę możliwości jak najbardziej zrozumiały dla potencjalnej osoby zainteresowanej.

Zaimplementowane zostały również gotowe rozwiązania takie jak np. Procedura obsługująca czujnik ultradźwiękowy, gdzie to, aby otrzymać jak najdokładniejszy wynik pomiaru odległości od przedmiotu należało posłużyć się gotową do tego celu formułą.

Kod źródłowy dla Kontrolera

```
1 #include <SPI.h>
2 #include <nRF24L01.h>
3 #include <RF24.h>
4 #include <Servo.h>
```

Wykorzystane biblioteki

```
#define buttonPin 3
#define buzzerPin 2
```

Definicja nazw dla pinów służących do obsługi przycisku switch oraz brzęczyka

```
bool prevButton = true;
bool currButton = true;
```

Deklaracja dwóch zmiennych typu bool przechowujące odpowiednio poprzedni i aktualny stan na przełączniku

```
bool buzzerState = LOW;
```

Deklaracja zmiennej służącej do określenia stanu na wyjściu brzęczyka

```
bool autoMode = false;
```

Deklaracja zmienna pomocnicza służąca do określenia czy aktualny tryb pracy pojazdu jest trybem autonomicznym

```
float sentData;
```

Deklaracja zmiennej służącej do pobierania pomiara z czujnika temperatury nadesłanego z pojazdu

```
RF24 radio (7, 8);
```

Konstrukcja obiektu klasy RF24

```
byte address[][6] = {"00001" , "00002"};
struct package
{
  int X = 0;
  int Y = 0;
  bool Mode = false;
  };

typedef struct package Package;
Package data;
```

Deklaracja struktury, która w postaci pakietu zawiera w sobie informacje mające trafić do pojazdu

Fragment kodu należące do procedury setup:

```
32    pinMode(buttonPin, INPUT_PULLUP);
```

Ustawienie trybu pracy pinu przełącznika na input wraz z podciągniętym wewnętrznym rezystorem

```
34    radio.begin();
35    radio.openWritingPipe( address[1]);
36    radio.openReadingPipe(1, address[0]);
37    radio.setPALevel(RF24_PA_MIN);
```

Szereg instrukcji związanych z wywołaniem połączenia w komunikacji radiowej

```
31  void setup() {
32    pinMode(buttonPin, INPUT_PULLUP);
33    delay(100);
34    radio.begin();
35    radio.openWritingPipe( address[1]);
36    radio.openReadingPipe(1, address[0]);
37    radio.setPALevel(RF24_PA_MIN);
38    delay(100);
39    currButton = digitalRead(buttonPin);
40
41  }
```

widok na pełną procedurę setup

Fragment kodu należące do procedury loop:

```
46    data.X = analogRead(A2);
47    data.Y = analogRead(A3);
```

Przypisanie do zmiennych wewnątrz pakietu odczytów z pinów połączonych do osi modułu joystick

```
49    prevButton = currButton;
50    currButton = digitalRead(buttonPin);
```

Przypisanie do zmiennej informująca o poprzednim stanie przełącznika aktualny, a do aktualnego nowy odczyt z pinu.

```
51    if (prevButton == HIGH && currButton == LOW){
52       autoMode = !autoMode;
53       data.Mode = autoMode;
54    }
55    else{
56     data.Mode = autoMode;
57    }
58   delay(50);
59   radio.write(&data, sizeof(data));
```

Fragment kodu odpowiedzialny za obsługę przypadku, gdy przełącznik został naciśniety i do pakiety trafi wartość informująca o konieczności zmiany trybu działania pojazdu

```
59   radio.startListening();
60   while(!radio.available());
61   radio.read(&sentData, sizeof(sentData));
62
63   if (sentData >= 30 && buzzerState == LOW){
64     buzzerState = HIGH;
65     digitalWrite(buzzerPin, buzzerState);
66   }
67
68   else if (sentData >= 30 && buzzerState == HIGH){
69
70   }
71
72   else{
73     buzzerState = LOW;
74     digitalWrite(buzzerPin, buzzerState);
75   }
76
77 }
```

Fragment kodu odpowiedzialny za pobranie wartości przychodzącej na kontroler ze strony pojazdu i w przypadku odczytu o określonym parametrze uruchomiony zastaje brzęczyk

```
43 void loop() {
44   delay(5);
45   radio.stopListening();
46   data.X = analogRead(A2);
47   data.Y = analogRead(A3);
48 
49   prevButton = currButton;
50   currButton = digitalRead(buttonPin);
51    if (prevButton == HIGH && currButton == LOW){
52       autoMode = !autoMode;
53       data.Mode = autoMode;
54    }
55    else{
56     data.Mode = autoMode;
57    }
58   delay(50);
59   radio.write(&data, sizeof(data));
60 
61   delay(5);
62   radio.startListening();
63   while(!radio.available());
64   radio.read(&sentData, sizeof(sentData));
65 
66   if (sentData >= 30 && buzzerState == LOW){
67     buzzerState = HIGH;
68     digitalWrite(buzzerPin, buzzerState);
69   }
```

widok na pełną procedurę loop

Kod źródłowy dla Pojazdu

```
1 #include <SPI.h>
2 #include <nRF24L01.h>
3 #include <RF24.h>
4 #include <Servo.h>
5 #include <OneWire.h>
6 #include <DallasTemperature.h>
```

Wykorzystane biblioteki

```
#define rightBack A3
#define rightFront A2
#define leftBack A5
#define leftFront A4

#define enableLeft 5
#define enableRight 6

#define cePin 8
#define csnPin 7

#define trigPin A0
#define echoPin A1

#define servoPin 9

#define temprPin 2
```

Definicje nazw dla konkretnych pinów będących odpowiedzialne za obsługę poszczególnych podzespołów

```
RF24 radio(cePin, csnPin);

Servo towerServo;
```

Konstrukcja obiektów klas RF24 oraz Servo

```
OneWire oneWire(temprPin);
DallasTemperature sensors(&oneWire);
```

Konstrukcja obiektów z klas powiązanych z czujnikiem temperatury

```
int autoMode;
```

Deklaracja zmiennej typu int, do której przesyłany z kontrolera jest aktualny tryb pracy. W przypadku przyjęcia wartości 1 pozwalająca na przejście w tryb autonomiczny.

```
float sentData[1];
```

Deklaracja tablicy typu float, do której zapisywany jest pomiar z czujnika temperatury i przesyłany dalej do kontrolera

```
49 int datagram[3];
```

Deklaracja tablicy typu int do której trafiają dane odczytane z kontrolera.

```
51 int leftMotorSpeed = 0;
52 int rightMotorSpeed = 0;
```

Deklaracja zmiennych odpowiedzialnych określenie wartości prędkości dla silników przy użyciu modulacji PWM w trybie pracy zdalnego sterowania

```
54 byte address[][6] = {"00008", "00009"};
```

Deklaracja tablicy znaków zawierających adresy, po których komunikują wzajemnie się moduły NRF24L01

Fragmenty kodu należące do procedury setup:

```
60   radio.begin();
61   radio.openWritingPipe(address[0]);
62   radio.openReadingPipe(1, address[1]);
63   radio.setPALevel(RF24_PA_MAX);
```

Szereg instrukcji związanych z wywołaniem połączenia w komunikacji radiowej

```
64   towerServo.attach(servoPin);
```

Podłączanie obiektu klasy Servo.h

```
65   sensors.begin();
```

Inicjalizacja czujnika temperatury

```
  pinMode(echoPin, INPUT);
  pinMode(trigPin, OUTPUT);

  pinMode(enableLeft, OUTPUT);
  pinMode(enableRight, OUTPUT);
  pinMode(leftFront, OUTPUT);
  pinMode(leftBack, OUTPUT);
  pinMode(rightFront, OUTPUT);
  pinMode(rightBack, OUTPUT);

  digitalWrite(leftFront, LOW);
  digitalWrite(leftBack, LOW);
  digitalWrite(rightFront, LOW);
  digitalWrite(rightBack, LOW);
```

Konfiguracja trybu pracy dla pinów czujnika ultradźwiękowego oraz silników wraz z ustalenie na nich stanu niskiego

```
void setup(){
  Serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(address[0]);
  radio.openReadingPipe(1, address[1]);
  radio.setPALevel(RF24_PA_MAX);
  towerServo.attach(servoPin);
  sensors.begin();

  delay(1000);

  pinMode(echoPin, INPUT);
  pinMode(trigPin, OUTPUT);

  pinMode(enableLeft, OUTPUT);
  pinMode(enableRight, OUTPUT);
  pinMode(leftFront, OUTPUT);
  pinMode(leftBack, OUTPUT);
  pinMode(rightFront, OUTPUT);
  pinMode(rightBack, OUTPUT);

  digitalWrite(leftFront, LOW);
  digitalWrite(leftBack, LOW);
  digitalWrite(rightFront, LOW);
  digitalWrite(rightBack, LOW);
}
```

widok na pełną procedurę setup

Fragment kodu należące do procedury loop:

```
110 long ultrasonicRead(){
111   long duration, distance;
112   digitalWrite(trigPin, LOW);
113   delayMicroseconds(2);
114   digitalWrite(trigPin, HIGH);
115   delayMicroseconds(10);
116   digitalWrite(trigPin, LOW);
117   duration = pulseIn(echoPin, HIGH);
118   distance = (duration / 2) / 29.1;
119   return distance;
120 }
```

Funkcja zwracająca odległość wyliczoną za pomocą czujnika ultradźwiękowego

```
122 void moveForward(){
123   digitalWrite(leftBack, HIGH);
124   digitalWrite(leftFront, LOW);
125   digitalWrite(rightBack, HIGH);
126   digitalWrite(rightFront, LOW);
127 }
128
129 void moveBackward(){
130   digitalWrite(leftBack, LOW);
131   digitalWrite(leftFront, HIGH);
132   digitalWrite(rightBack, LOW);
133   digitalWrite(rightFront, HIGH);
134 }
135
136 void turnRight(){
137   digitalWrite(enableLeft, HIGH);
138   digitalWrite(enableRight, HIGH);
139   digitalWrite(leftBack, HIGH);
140   digitalWrite(leftFront, LOW);
141   digitalWrite(rightBack, LOW);
142   digitalWrite(rightFront, HIGH);
143 }
144
145 void turnLeft(){
146   digitalWrite(enableLeft, HIGH);
147   digitalWrite(enableRight, HIGH);
148   digitalWrite(leftBack, LOW);
149   digitalWrite(leftFront, HIGH);
150   digitalWrite(rightBack, HIGH);
151   digitalWrite(rightFront, LOW);
152 }
```

Szereg procedur odpowiedzialnych za obsługę silników wykorzystanych w autonomicznym trybie jazdy

```
154 void Stop(){
155   digitalWrite(enableLeft, LOW);
156   digitalWrite(enableRight, LOW);
157   digitalWrite(leftBack, LOW);
158   digitalWrite(leftFront, LOW);
159   digitalWrite(rightBack, LOW);
160   digitalWrite(rightFront, LOW);
161 }
162
```

Procedura zatrzymania silników

```
void obstacleAvoidance(){
  long currDistance = ultrasonicRead();
  delay(100);
  if(currDistance <= 40){
    Stop();
    delay(10);
    radio.stopListening();
    radio.write(sentData, sizeof(sentData));
    delay(100);
    digitalWrite(enableLeft, HIGH);
    digitalWrite(enableRight, HIGH);
    moveBackward();
    delay(200);
    Stop();
    delay(10);
    radio.startListening();
    if (radio.available()){
      while (radio.available()){
      radio.read(datagram, sizeof(datagram));
    }}
    autoMode = datagram[2];
    if (autoMode == 1){
      delay(500);
      towerServo.write(30);
      delay(300);
      long rightDistance = ultrasonicRead();
      // delay(100);
      delay(10);
      radio.startListening();
      if (radio.available()){
        while (radio.available()){
        radio.read(datagram, sizeof(datagram));
      }}
      autoMode = datagram[2];
      if (autoMode == 1){
        delay(500);
        towerServo.write(150);
        delay(300);
        long leftDistance = ultrasonicRead();
```

```
201         long leftDistance = ultrasonicRead();
202         delay(500);
203         Serial.println("");
204         Serial.println(rightDistance);
205         Serial.println(leftDistance);
206         Serial.println("");
207         if (leftDistance < rightDistance){
208           turnRight();
209           delay(200);
210           }
211         else if (rightDistance < leftDistance){
212           turnLeft();
213           delay(200);
214           }
215         else {
216           digitalWrite(enableLeft, HIGH);
217           digitalWrite(enableRight, HIGH);
218           moveBackward();
219           delay(200);
220           }
221       }
222       else{
223         towerServo.write(90);
224         return;
225       }
226     }
227     else{
228         towerServo.write(90);
229         return;
230     }
231   }
232   else{
233     towerServo.write(90);
234     analogWrite(enableLeft, 150);
235     analogWrite(enableRight, 150);
236     moveForward();
237   }
238 }
```

Główna procedura dla autonomicznego trybu jazdy wraz z przesłaniem do kontrolera odczytu z czujnika temperatury.

```
void remoteControl(){

  if (yValue < 470){
    moveBackward();
    rightMotorSpeed = map(yValue, 470, 0, 0, 255);
    leftMotorSpeed = map(yValue, 470, 0, 0, 255);
  }

   else if (yValue > 550){
    moveForward();
    rightMotorSpeed = map(yValue, 550, 1023, 0, 255);
    leftMotorSpeed = map(yValue, 550, 1023, 0, 255);
  }

  else{
    rightMotorSpeed = 0;
    leftMotorSpeed = 0;
  }

  if (xValue < 470){
    int xValueScaled = map(xValue, 470, 0, 0, 255);
    rightMotorSpeed = rightMotorSpeed + xValueScaled;
    leftMotorSpeed = leftMotorSpeed - xValueScaled;

    if (leftMotorSpeed > 255){
      leftMotorSpeed = 255;
    }

    if (rightMotorSpeed < 0) {
      rightMotorSpeed = 0;
    }
  }

  else if (xValue > 550){
    int xValueScaled = map(xValue, 550, 1023, 0, 255);
    rightMotorSpeed = rightMotorSpeed - xValueScaled;
    leftMotorSpeed = leftMotorSpeed + xValueScaled;


    if (rightMotorSpeed > 255){
      rightMotorSpeed = 255;
    }

    if (leftMotorSpeed < 0) {
      leftMotorSpeed = 0;
    }
  }

  if (rightMotorSpeed < 70){
    rightMotorSpeed = 0;
  }

  if (leftMotorSpeed < 70){
    leftMotorSpeed = 0;
  }

  analogWrite(enableLeft, leftMotorSpeed);
  analogWrite(enableRight, rightMotorSpeed);
}
```

Główna procedura dla trybu zdalnego sterowania z obsługą modulacji PWM

```
85 void loop(){
86   delay(10);
87   radio.startListening();
88   if (radio.available()){
89     while (radio.available()){
90       radio.read(datagram, sizeof(datagram));
91     }
92       xValue = datagram[0];
93       yValue = datagram[1];
94       autoMode = datagram[2];
95       Serial.println(xValue);
96       Serial.println(yValue);
97       Serial.println(autoMode);
98   }
99
100  if (autoMode == 1){
101    delay(25);
102    obstacleAvoidance();
103  }
104
105  else{
106    remoteControl();
107  }
108 }
```

widok na pełną procedurę loop

Wszystkie te podjęte czynności, czyli zarówno od strony oprogramowania jak i strony sprzętowej złożyły się na całokształt, którym to jest produkt finalnym projektu w postaci działającego system realizującego postawione mu zadania.

Rozdział IV

Projekt jako konstrukcja złożona nie uchronił się od pewnych mankamentów zaistniałych na przebiegu procesu jego tworzenia. Do takich niedoskonałości zaliczyć można, wykorzystanie nieodnawialnego źródła zasilania bądź przynajmniej takiego, które było zdolne do ponownego ładowania co niewątpliwe narażać może na dodatkowe koszty w postaci konieczności zakupu nowego egzemplarza baterii, jak również stanowiące realne zagrożenie dla środowiska w przypadku niepoddania ich procesowi utylizacji odpadów elektronicznych. Jednak mimo to możliwość, że system zostanie poddany do tego stopnia nadmiernej eksploatacji jest na tyle znikoma więc pozwala z dużo dozą stanowczości stwierdzić, że taka sytuacja nie będzie miała miejsca w przyszłości zbyt prędko.

Następną kwestią, która niewątpliwie można zarzucić konstrukcji jest jej surowy wygląd. System ubogacić można było o pewne wstawki wizualne mające na celu w jakimś stopniu podnieść walory estetyczne i uprzyjemnić sam odbiór optyczny systemu, mimo tego według realnego zastosowania zawartego w rozdziale pierwszym wobec systemu nie mającego nic wspólnego z możliwością pełnienie funkcji narażonych na nadmierną ekspozycję na ludzkie oko nie uznałem tego za konieczne. Jednakże mimo to zostały poczynione starania o zachowanie względnej estetyki, której obraz dopatrywać można w np. szczególnie obranej usystematyzowanej metodyce porządkowania złączek pod względem kolorystycznym.

Bez zarzutów nie można pozostawić również pewnych rozwiązań czysto ze sfery kodu.

Jednym z takich jest, wykorzystanie w obrębie programu funkcji delay, której to zadaniem jest zatrzymanie działania programu na pewien zadany czas, która to tym samym może nieść wiele nie porządnych konsekwencji takich jak zakłócenie w sposobie działania

programu. Mimo to z instrukcji tej korzystano tylko tam, gdzie było to potrzebne bądź nawet konieczne z uwzględnieniem pozostałych obrębów kodu mogących funkcjonować w sposób niezakłócony.

Następnym narzucającym się niekoniecznie najbardziej optymalnym rozwiązaniem jest wykorzystanie w kodzie instrukcji digitalWrite. Instrukcja ta odpowiedzialna jest za manipulację stanem logicznym na konkretnych pinach lecz nie koniecznie robi to w satysfakcjonującym tempie. Bardziej korzystnym rozwiązaniem byłoby w tym miejscu posłużenie się odwołaniem bezpośrednim do rejestrów, do których to właśnie przez szereg pośrednich operacji odwołuje się wyżej wymieniona instrukcja co ma swoje odzwierciedlenie w szybkości wykonywania tej operacji. Proces ten niewątpliwie miałaby swoje odzwierciedlanie w poprawie szybkości wykonywania programu. Jednak są to marginalne różnicę, które tak naprawdę nie mogą w żaden sposób dać się odczuć w rzeczywistości przy normalnym użytkowaniu systemu.

W czasach zegarków posiadających więcej funkcjonalności niż godzin, których to według ich nominalnego zastosowania powinny one pokazywać, ilość możliwość oferowanych przez wyżej wyszczególniony system może wydawać się znikoma i w tym aspekcie dopatrywać można się pewnego jego mankamentu, jednakże system swoją kompozycją nie zamyka się na przyszłe modernizację i możliwe wzbogacenia o dodatkowe funkcjonalności systemu dzięki dołożeniu dodatkowego “czujnika bądź dwóch” co niewprowadziło by go w stan zupełnego chaosu. Realne jest zarówno rozbudowanie już istniejących trybów pracy jak i implementacja nowych wcale zależnych od poprzednich.