Poduszkowiec

IEiT
Informatyka
Technika Mikroprocesorowa 2
2015/2016

Mateusz Jarosz
Wojciech Zagrajczuk

Spis treści

1. Temat projektu

Tematem naszego projektu była budowa oraz zaprogramowanie sterowania poduszkowca. Należało obsłużyć skręcanie, przyspieszanie oraz wznoszenie/opadanie pojazdu.

2. Harmonogram pracy

Data	Postęp
2015-10-26	Wybór tematu projektu
2015-11-27	Ukończenie budowy głównej części modelu
2015-12-09	Uzyskanie przewodowej kontroli nad sterem
2015-12-16	Uzyskanie łączności z modułem ESP oraz postawienie serwera HTTP
2015-12-21	Uzyskanie bezprzewodowej kontroli nad sterem
2016-01-02	Uzyskanie kontroli nad sterem w dużej prędkości
2016-01-04	Stworzenie aplikacji na system Android do kontroli modelu
2016-01-05	Uzyskanie kontroli bezprzewodowej nad silnikami
2016-01-11	Usztywnienie fizyczne konstrukcji układu
2016-01-15	Ukończenie budowy owiewki kadłuba
2016-01-16	Malowanie modelu i organizacja kabli
2016-01-18	Oddanie projektu

3. Wykorzystywany sprzęt

- Arduino Leonardo
- ESP8266-07
- 2 x silnik EMAX XA2212
- Śmigło 2 płatowe
- Śmigło 3 płatowe
- 2 x regulator napięcia ESC ABC POWER 30A
- Styrodur 3mm oraz 6mm

4. Sposób działania poduszkowca

Poduszkowiec posiada dwa silniki. Pierwszy z nich odpowiada za wznoszenie się. Kręcąc śmigłem zagarnia powietrze z nad siebie i kieruje je pod siebie. Tworzy to siłę, a przez to, że powietrze ma bardzo ograniczone możliwości aby uciec i w efekcie podnosi poduszkowiec. Dzięki temu silnikowi pojazd ma bardzo znikome tarcie lub kontakt z podłożem w ogóle nie występuje. Drugi silnik odpowiada za napęd. Odpycha powietrze i zgodnie z 3 zasadą

zdynamiki sam także otrzymuje pewną siłę (o przeciwnym zwrocie niż ta, którą sam wytworzył).

5. Sposób komunikacji z poduszkowcem

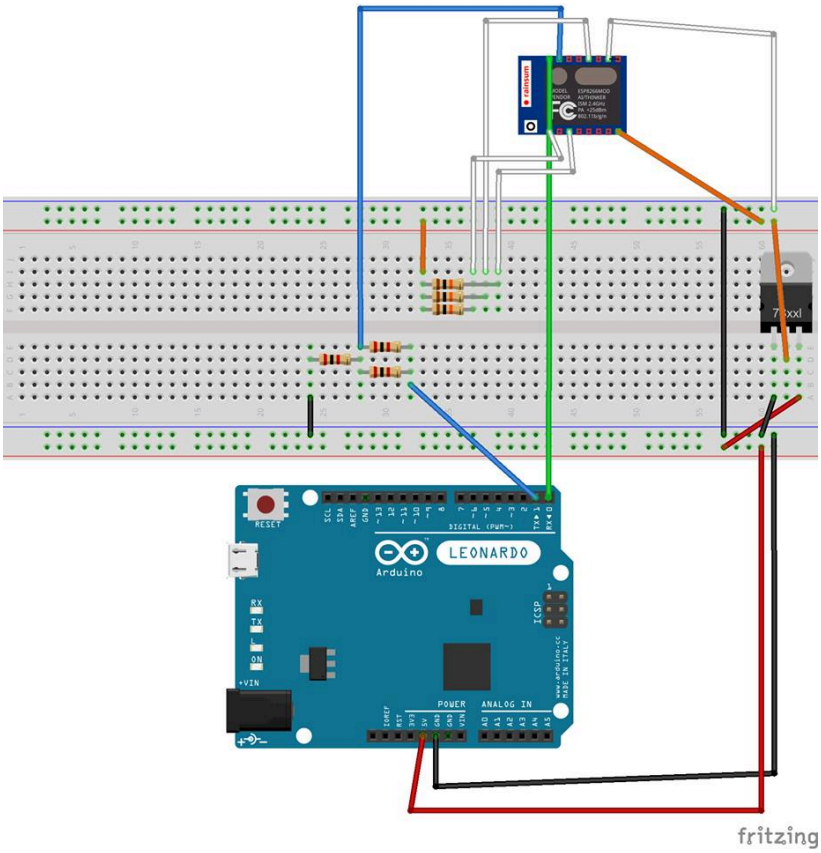
W komunikacji z urządzeniem posłużyliśmy się modułem komunikacji WiFi tj. EPS8266-E07. Tworzymy na nim sieć lokalną o adresie 192.168.4.1 i tworzymy serwer HTTP na porcie 80. Następnie korzystając z urządzenia z systemem Android zaopatrzonym w przygotowaną przez nas aplikację łączymy się z siecią stworzoną przez moduł ESP. Używając aplikacji łączymy się z serwerem przez socket używający protokołu TCP. Następnie cyklicznie co 0.1s wysyłamy do modułu informacje o ustawieniach zadanych przez obsługującego aplikację. Moduł odbiera te informacje, dzieli je i wysyła na odpowiednie lokacje (tj. do steru oraz obu silników).

6. Uruchamianie poduszkowca

1. Podłączyć płytkę Arduino do komputera z zainstalowym oprogramowaniem Arduino IDE przy pomocy kabla USB ze złączem microUSB z jednej strony oraz standardowym wyjściem męskim z drugiej
2. W Arduino IDE należy wybrać na pasku w Narzędzia -> Płyta używaną wersję mikrokontrolera oraz numer portu (Narzędzia -> Port) do którego podpięliśmy nasze urządzenie
3. Połączyć model ESP i Arduino tak jak jest to narysowane na schemacie w punkcie "Schemat układu głównego i łączenie się z siecią"
4. Następnie należy skopiować do środowiska kod znajdujący się w punkcie "Programowanie płytki Arduino" tej dokumentacji oraz wgrać oprogramowanie na płytkę
5. W przypadku braku błędów powinna się pojawić nowa dostępna sieć
6. Następnie należy stworzyć aplikację do sterowania poduszkowcem. Aby to zrobić należy na komputerze z zainstalowanym kompilatorem Python, biblioteką *kivy*, narzędziem *Buildozer* oraz Android SDK wykonać punkt "Programowanie kontrolera do sterowania poduszkowcem".
7. Podłączyć ster na podstawie punktu "podłączanie się do steru i silników". Następnie włączyć aplikację. Jeżeli nie otrzymujemy ekranu pokazanego w punkcie "Programowanie kontrolera do sterowania poduszkowcem", ale aplikacja zostaje natychmiast zamknięta należy sprawdzić czy na pewno jesteśmy podłączeni do odpowiedniej sieci.

8. W przypadku powodzenia należy wybrać jeden z regulatorów a następnie połączyć czerwony kabel wychodzący z regulatora do pinu Vin na płytce Arduino. Dzięki temu zasilanie będzie prowadzone z baterii a nie przez kabel USB
9. Następnie należy odpiąć kabel USB od płytki i na podstawie punktu 8 "Podłączenie się do steru i silników" podłączyć jeden z silników do regulatora, który wybraliśmy do zasilania płytki.
10. Do tego regulatora należy podłączyć baterię. Arduino powinno mieć zasilanie (czyli odpowiednia dioda powinna być zapalona). Powinien odpalić się także program na nią wgrany co objawiać się będzie ponownym pojawieniem się naszej sieci. Jeżeli silnik będzie wydawał ciągłe piknięcia, oznacza to, że nie dostał mu podany stan niski na starcie. Aby to zrobić należy włączyć aplikację i jednym z dwóch suwaków szukać w celu znalezienia odpowiedniego stanu (objawi się to zagraniem przez silnik krótkiej melodii). W przypadku gdy nie istnieje taka wartość, która daje stan niski to należy przeprogramować regulator (jest to także dobry pomysł nawet po uzyskaniu kontroli nad silnikiem w celu posiadania pełnego zakresu pracy silnika). Aby to uczynić należy podążać zgodnie z poleceniami dostępnymi w instrukcji regulatora (link w bibliografii). W tym momencie uzyskujemy pełną kontrolę nad jednym z silników.
11. Analogicznie postąpić do drugiego z silników

7. Schemat układu głównego i łączenie się z siecią



Schemat ten zapewnie pełną komunikację pomiędzy płytką Arduino oraz modułem ESP. Zasilenie takiego układu powinno skutkować pojawieniem się nowej sieci o nazwie "SIECTESTOWA" z hasłem dostępuj "HELLOWORLD". Aby uzyskać pełną funkcjonalność należy się do tej sieci podłączyć.

8. Podłączanie się do steru i silników

Aby podłączyć ster należy połączyć następujące kable:

- kabel czarny wychodzący ze steru do pinu GND lub wyprowadzić go na płytkę stykową
- kabel czerwony wychodzący ze steru do pinu 5V na płytce Arduino
- kabel żółty wychodzący ze steru do pinu 10 na płytce Arduino

Aby podłączyć silnik podnoszący należy połączyć następujące kable:

- 3 czarne kable wychodzące z tego silnika z 3 kablami wychodzącymi z regulatora w taki sposób, aby kolory zacisków się zgadzały parami (jest to ważne dla prawidłowego działania silnika)
- Kabel czarny wyprowadzony przez wtyczkę żeńską należy podłączyć do pinu GND lub płytki stykowej na wcześniej wyprowadzone miejsce
- Kabel biały wyprowadzony przez wtyczkę żeńską należy podłączyć do pinu 9 na płytce Arduino

Aby podłączyć silnik napędowy należy połączyć następujące kable:

- 3 czarne kable wychodzące z tego silnika z 3 kablami wychodzącymi z regulatora w taki sposób, aby kolory zacisków się zgadzały parami (jest to ważne dla prawidłowego działania silnika)
- Kabel czarny wyprowadzony przez wtyczkę żeńską należy podłączyć do pinu GND lub płytki stykowej na wcześniej wyprowadzone miejsce
- Kabel biały wyprowadzony przez wtyczkę żeńską należy podłączyć do pinu 11 na płytce Arduino

9. Programowanie płytki Arduino

```
#include <Servo.h>

int pos = 90;
int vel;
int hig;
Servo myservo;
Servo velocity;
Servo hight;

bool hasStarted = false;

void setup()
{
	Serial.begin(9600);
	Serial1.begin(115200); // wartość zależna od modułu, najczęściej 9600 lub 115200

	myservo.attach(10);
	velocity.attach(11);
	hight.attach(9);


	sendData("AT+RST\r\n",2000); // reset modułu
	sendData("AT+CWMODE=2\r\n",1000); // ustawienie w trybie Access Point
	sendData("AT+CWSAP=\"SIECTESTOWA\",\"HELLOWORLD\",5,3",1000);

	sendData("AT+CIFSR\r\n",1000); // Uzyskanie adresu IP (domyślnie 192.168.4.1)
	sendData("AT+CIPMUX=1\r\n",1000); // Tryb połączeń wielokrotnych
	sendData("AT+CIPSERVER=1,80\r\n",1000); // Ustawienie serwera na porcie 80
}
void loop()
{
	if(!hasStarted)
	{
		velocity.write(35);
		myservo.write(90);
		hight.write(35);
	}

	if(Serial1.available()) // sprawdzenie czy moduł otzrymał dane
	{
		if(Serial1.find("+IPD,"))
		{
			delay(10); // czekanie na zapełnienie bufora danymi
			hasStarted = true;
			Serial1.find("serwo="); // wyszukanie frazy "serwo="
```

```
                        char h = Serial1.read();
                        char v = Serial1.read();
                        char a = Serial1.read();

                        pos = int(a);

                        if(pos < 0)
                        pos = 127 + (pos + 127);


                        if(pos>180||pos<0){
                        pos=90;
                        Serial.println("Im wrong");
                        }
                        myservo.write(180-pos);

                        vel = int(v);
                        velocity.write(vel);

                        hig = int(h);
                        hight.write(hig);
                }
        }
}


String sendData(String command, const int timeout)
{
        String response = "";

        Serial1.print(command); // wysłanie polecenia do Serial1

        long int time = millis();

        while( (time+timeout) > millis())
        {
                while(Serial1.available()) //jeśli są jakieś dane z modułu, wtedy następuje ich
odczyt
                {
                        char c = Serial1.read(); // odczyt kolejnego znaku
                        response += c;
                }
        }
        Serial.print(response);
        return response;
}
```

Funkcja *setup* wywołuje się tylko raz po podłączeniu płytki do źródła zasilania. W niej kolejno inicjujemy wyjście serialowe (które służą głównie do debuggowania kodu oraz podglądania stanu modułu ESP), przypisujemy do pinów 9, 10 i 11 sygnały wysyłane do silników i serwa (sterującego sterem). Ostatni blok jest to przy wykorzystaniu funkcji sendData wysyłanie do modułu ESP wysyłanie komend AT, które inicjują sieć i ustawiają jej parametry.

Funkcja *loop* jest wykonywana stale po zakończeniu funkcji setup. Zanim aplikacja do sterowania zacznie wysyłać sygnał podaje ona stan niski do obu silników oraz ustawia skręt steru na 90 (czyli jazdę na wprost). Następnie jeżeli otrzyma sygnał od kontrolera to szuka napisu "serwo=". Zaraz za nim odbiera 3 znaki, które określają wartości jakie zostały podane na kontrolerze. Następnie przekształca te znaki w sposób czytelny dla ich docelowych nadawców (tj. silników lub serwa) a następnie je tam wysyła.

Funkcja *sendData* posiada dwa paramatry: *command* jest to komenda AT, która jest bezpośrednio wysyłana wyjście modułu ESP oraz *timeout* czyli czas oczekiwania na odpowiedź od modułu ESP. Dodatkowo funkcja ta drukuje na wyjście wszystkie informacje odnośnie stworzonej sieci, które podczas konfigurowania zostały zwrócone przez moduł ESP.

10. Programowanie kontrolera do sterowania poduszkowcem

W przypadku braku posiadania któregoś z narzędzi należy go pobrać. Linki do główny stron można znaleźć w bibliografii.

main.py

```
from kivy.app import App
from kivy.uix.widget import Widget
from kivy.properties import ObjectProperty
from kivy.clock import Clock
import socket

hval = 0
vval = 0
aval = 90

TCP_IP = '192.168.4.1'
TCP_PORT = 80
BASE = "serwo="

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))

class HeightSlider(Widget):
    pass
```

```
class VelocitySlider(Widget):
   pass

class AngleSlider(Widget):
   pass

class HoovercraftDriver(Widget):
   hslider = ObjectProperty(None)
   vslider = ObjectProperty(None)
   aslider = ObjectProperty(None)

   def update(self, dt):
      global hval, vval, aval
      lhval = hval + 35.0
      lvval = vval + 35.0
      s.sendall(BASE + chr(int(lhval)) + chr(int(lvval)) + chr(int(aval)) + "\n")


   def on_touch_move(self, touch):
      if touch.x < self.width / 5:
         if touch.y >= 5 and touch.y <= self.height - 10:
            global hval
            self.hslider.pos = 5, touch.y
            hval = (92.0 * touch.y)/(self.height - 10)

      if touch.x < 2 * self.width / 5 and touch.x > self.width / 5:
         if touch.y >= 5 and touch.y <= self.height - 10:
            global vval
            self.vslider.pos = self.width / 5 + 5, touch.y
            vval = (92.0 * touch.y)/(self.height - 10)

      if touch.x > 2 * self.width / 5:
         if touch.x < self.width - 10:
            global aval
            self.aslider.pos = touch.x, 5
            aval = (180.0 * (touch.x - 2.0/5.0 * self.width))/(3.0/5.0 * self.width - 5.0)


class HeightSlider(Widget):
   pass

class VelocitySlider(Widget):
   pass

class AngleSlider(Widget):
   pass
```

```
class HoovercraftApp(App):
    def build(self):
        app = HoovercraftDriver()
        Clock.schedule_interval(app.update, 1.0 / 10.0)
        return app

if __name__ == '__main__':
    HoovercraftApp().run()
```

hoovercraft.kv

```
#:kivy 1.0.9

<HeightSlider>:
    canvas:
        Rectangle:
            pos: self.pos
            size: self.size

<VelocitySlider>:
    canvas:
        Rectangle:
            pos: self.pos
            size: self.size

<AngleSlider>:
    canvas:
        Rectangle:
            pos: self.pos
            size: self.size


<HoovercraftDriver>:
    hslider: h_id
    vslider: v_id
    aslider: a_id

    HeightSlider:
        id: h_id
        pos: 5, 5
        size: root.width / 5 - 10, 5

    VelocitySlider:
        id: v_id
        pos: root.width / 5 + 5, 5
        size: root.width / 5 - 10, 5
```

```
AngleSlider:
    id: a_id
    pos: root.width * 7 / 10 - 3, 5
    size: 6, root.height - 10
```

buildozer.spec

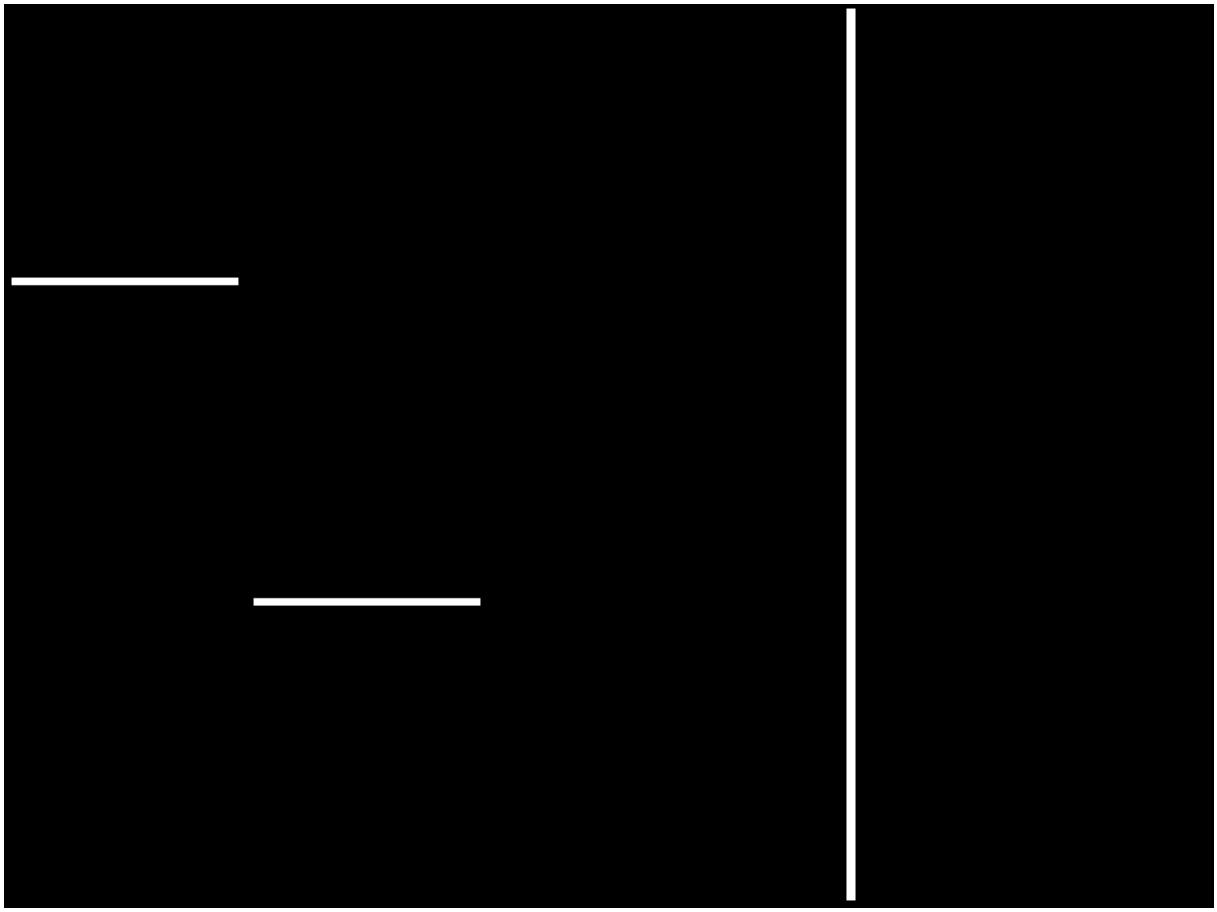
```
title = HooverCraft Driver
package.name = main
package.domain = org.test
source.dir = .
source.include_exts = py,png,jpg,kv,atlas
version = 1.0
requirements = kivy
orientation = landscape
fullscreen = 1
android.permissions = INTERNET
log_level = 1
warn_on_root = 1
```

Plik *main.py* zawiera główną część aplikacji. Klasa HoovercraftApp tworzy jeden obiekt typu HoovercraftDriver oraz zajmuje się pilnowaniem wysyłania danych do modułu. Klasa HoovercraftDriver zawiera referencję na obiekty które mają zajmować się obsługą serwa, silnika wznoszącego i silnika napędowego. Metoda *update* jest wywoływana co 0.1s i wysyła ona przez otwarty socket na adres i port naszego serwera na module ESP stringa postaci "serwo=hva\n", gdzie h jest to typ znakowy określający na jakiej wysokości ma znajdować się poduszkowiec (czyli wartość dla silnika wznoszacego), v czyli prędkość, z jaką poduszkowiec ma się poruszać (informacja dla silnika napędowego) oraz a czyli kąt jaki ma być ustawiony na serwo (czyli informacja dla steru). Metoda *onTouch* zajmuje się ustawieniem jednego z trzech suwaków w odpowiednim miejscu i ustawieniu odpowiedniej zmiennej globalnej (z której później informacje bierze funkcja *update.*

Plik *hoovercraft.kv* zawiera informacje o tym jak dana aplikacja powinna wyglądać czyli jeżeli rozpatrzymy tę aplikację z perspektywy wzorca projektowego *Model-View-Controller* to zajmuje się tylko i wyłączenie *view* (a main.py model i controller).

Plik *buildozer.spec* jest generowany przez narzędzie buildozer. Zawiera on konfigurację, która jest informacją dla rzeczonego już buildozera, którego głównym zadaniem jest stworzenie pliku .apk dzięki któremu możemy zainstalować i używać napisana przez nas aplikację na systemach typu Android (buildozer umożliwia też budowanie aplikacji na urządzenia z systemem Apple, ale jest to dopiero wersja niestabilna i jest obecnie rozwijane.

Wygląd aplikacji



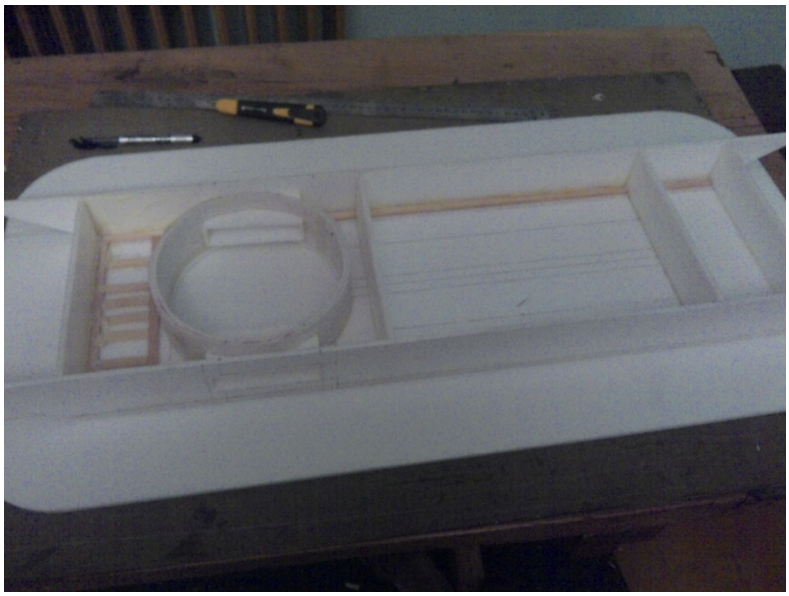
Pierwszy pasek od lewej ma szerokość jednej piątej szerokości ekranu. Biały prostokąt może być przesuwany po całej wysokości ekranu. Odpowiada on za wysokość podnoszenia się poduszkowca.
Drugi pasek od lewej ma także szerokość jednej piątej szerokości ekranu. Odpowiada on za prędkość poruszania sie poduszkowca.
Ostatnia strefa odpowiada za sterowanie kierunkiem lotu poduszkowca. Jego zakres zaczyna się od dwóch piątych szerokości ekranu (patrząc od lewej strony) aż do prawej krawędzi ekranu. W przeciwieństwie do poprzednich dwóch suwaków, jego położenie może zmieniać się na płaszczyźnie poziomej (a nie pionowej jak w przypadku dwóch poprzednich).

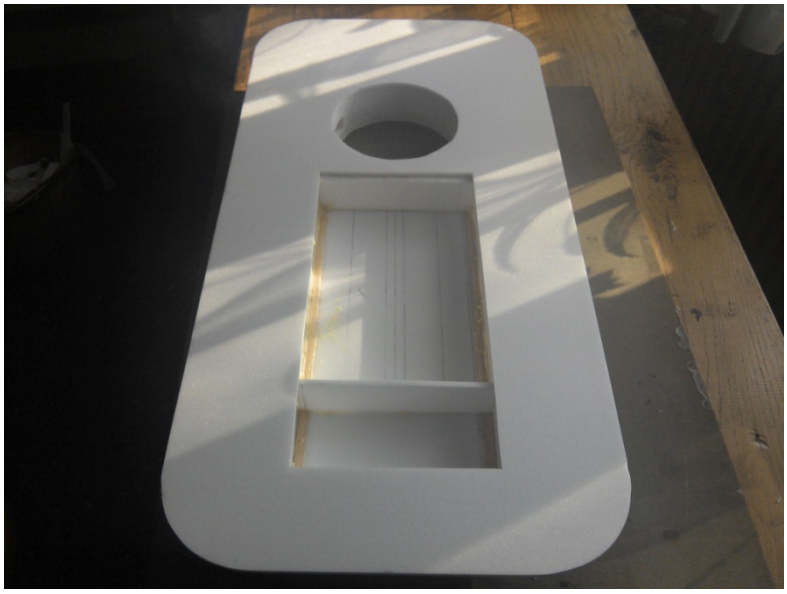
11. Składanie głównej części modelu

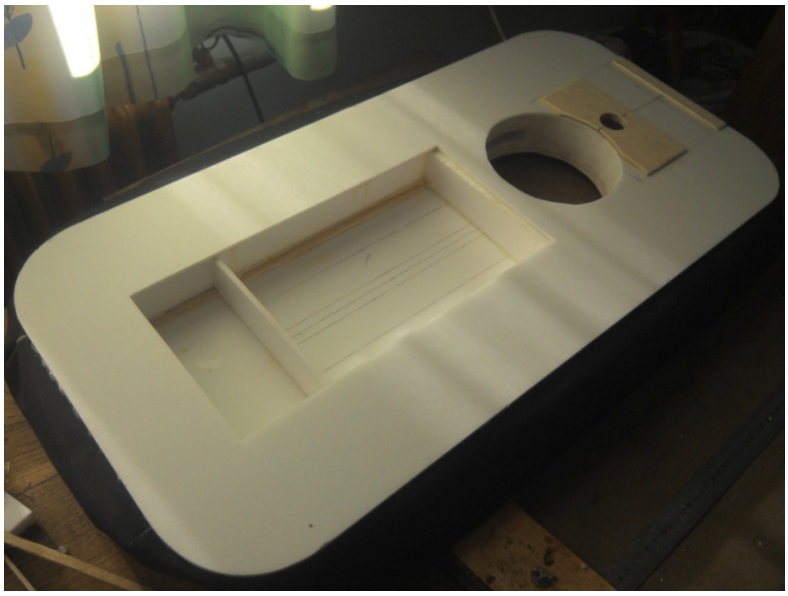
Projekt rozpoczęliśmy od zbudowania głównej części modelu tj. pojazdu, który po podłączeniu do niego innego sterowania nadawałby się do jazdy.

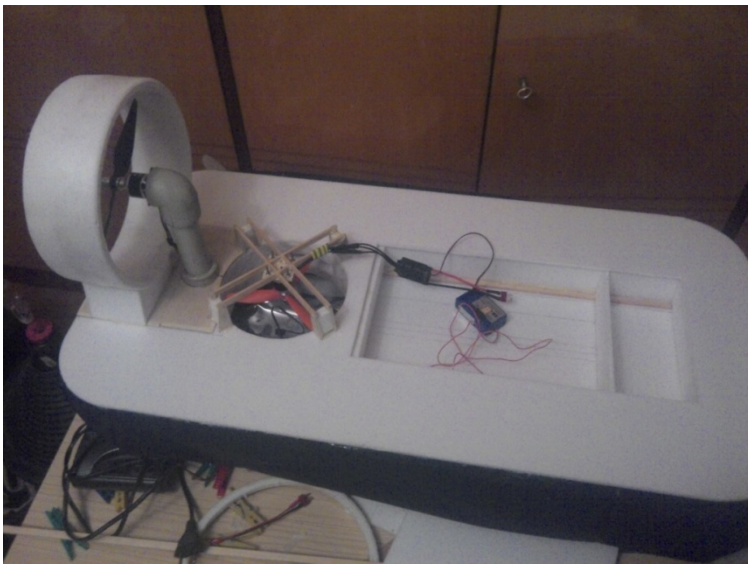
Pod poniższym linkiem można znaleźć próby sterowania poduszkowcem po podłączeniu do niego zewnętrznego modułu sterowania

http://student.agh.edu.pl/~mateusja/mikro/video/





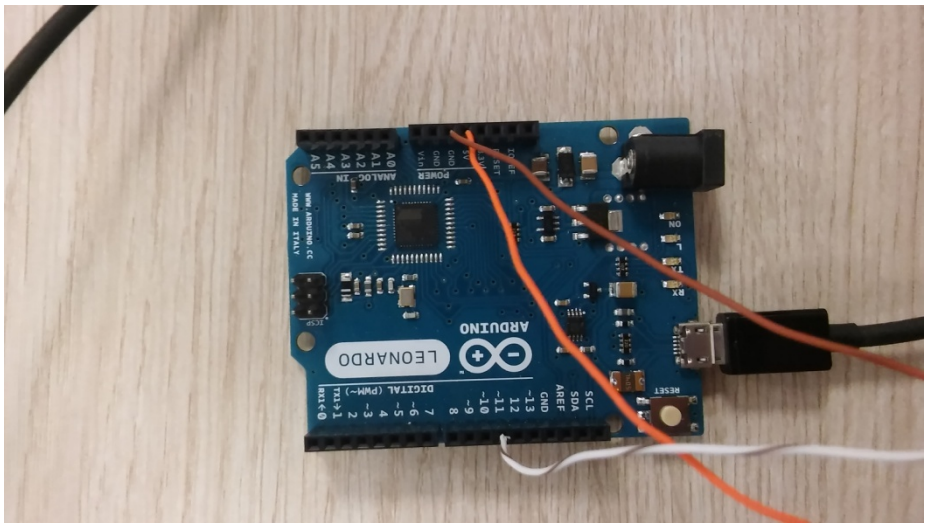




12. Uzyskanie przewodowej kontroli nad sterem

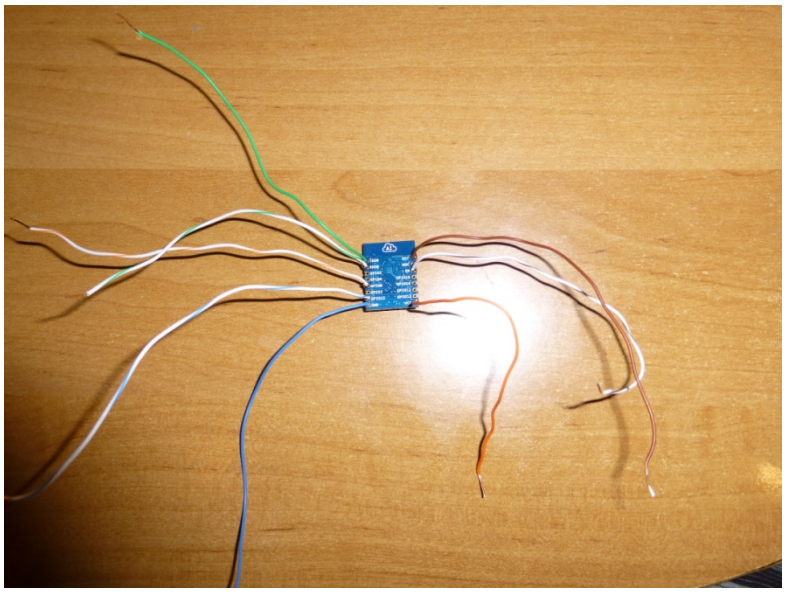
Pierwszym elementem budowy własnego układu sterowania była kontrola steru przy pomocy podawania na konsolę liczb. Ster miał ustawiać się pod odpowiednim kątem.

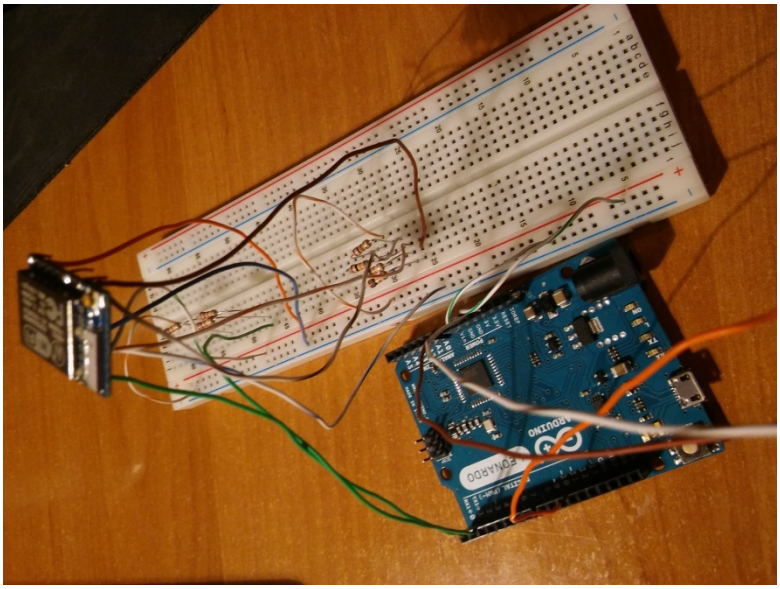
Poniższe zdjęcia prezentują układ który obsługiwał taką funkcjonalność.



13. Uzyskanie łączności z modułem ESP oraz postawienie serwera HTTP

Następnie uzyskaliśmy łączność z modułem ESP. Ze względu na niekorzystną dla nas budowę modułu musieliśmy przylutować kable, które następnie wpięliśmy przez płytkę stykową do Arduino.





Przez to całość była bardzo delikatna i bardzo ryzykowne byłoby jakiekolwiek próby terenowe. Dlatego też postanowiliśmy usztywnić całą konstrukcję.

14. Uzyskanie bezprzewodowej kontroli nad sterem

Pierwotnym zamysłem było stworzenie kontrolera przez stronę WWW. Jednakże pomysł ten legł w gruzach ponieważ jQuery, które chcieliśmy zaprząc do przesyłania danych, nie poradził sobie z taką prędkością przesyłania danych. Jako zamiennik zdecydowaliśmy się zmienić sposób przesyłania danych na proste wysyłanie na socket TCP.

15. Uzyskanie kontroli nad sterem w dużej prędkości oraz stworzenie aplikacji na system Android do kontroli steru

Po napisaniu aplikacji na system Android przeprowadziliśmy testy. Można je znaleźć pod następującym niepublicznym linkiem:

https://youtu.be/tRewklb0bVI

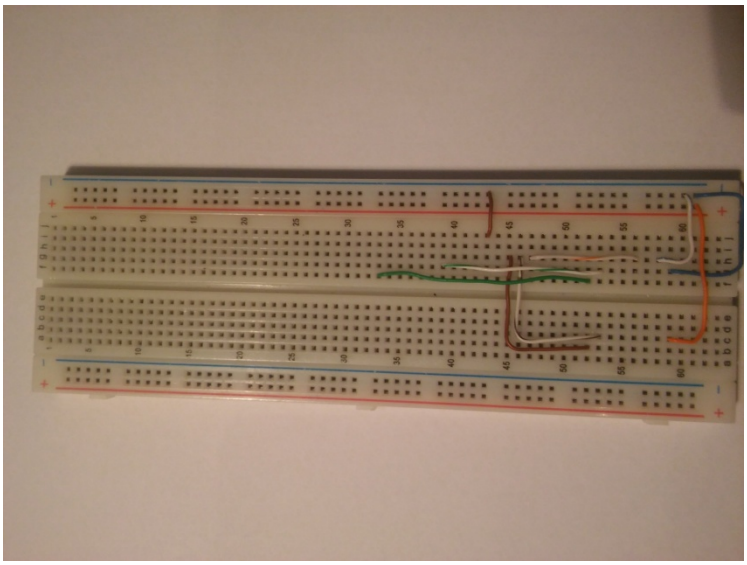
16. Uzyskanie kontroli bezprzewodowej nad silnikami

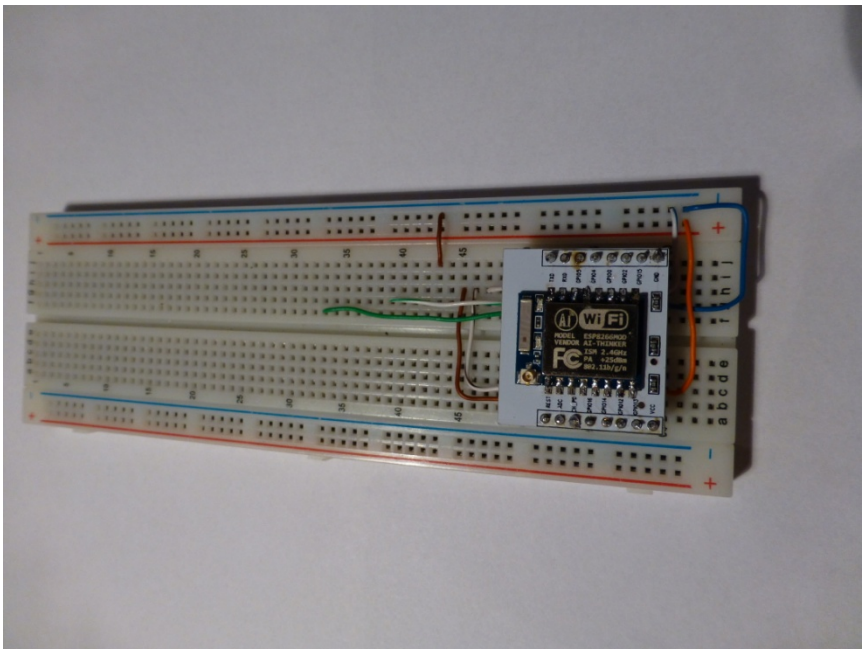
Ze względu na to, że ruch, który jest główną dziedziną naszego projektu, ciężko uchwycić na zdjęciach to kręciliśmy filmy. Poniższy film przedstawia próby sterowania poduszkowcem z podłączonymi silnikami

https://youtu.be/StERsE_sHQQ

17. Usztywnienie fizyczne konstrukcji układu

Ze względu na problem, który zaobserwowaliśmy omówiony w punkcie 13 postawiliśmy kupić specjalny adapter dedykowany pod układ ESP8266





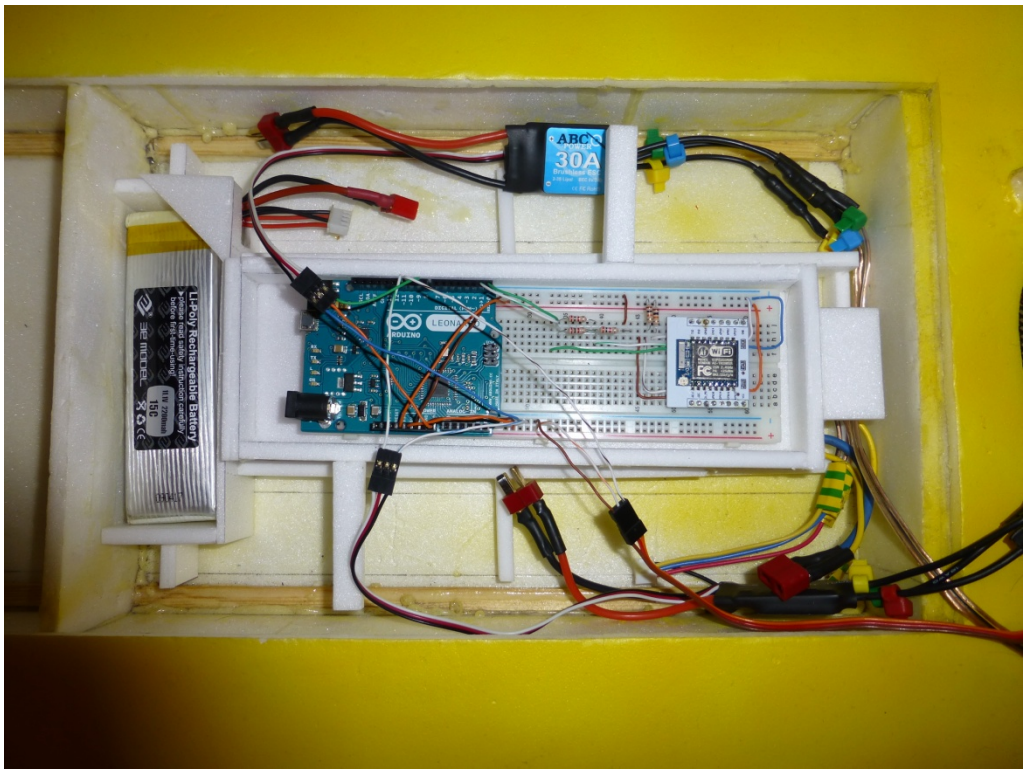
18. Ukończenie budowy owiewki kadłuba

Owiewka kadłuba ma za zadanie osłonić pokrywę miejsca, gdzie jest schowana elektronika oraz chronić silnik podnoszący przez ewentualnymi latającymi przedmiotami bez zmniejszania jego ciągu i dostępu powietrza.

19. Malowanie modelu i organizacja kabli

Model pomalowaliśmy na kolory żółty i pomarańczowy ze względu na jego dużą widoczność. Użyliśmy specjalnej farby w spreju przeznaczonej specjalnie do malowania styropianu.

Postanowiliśmy też zadbać o odpowiednie ustawienie kabli aby można było łatwo podłączyć i rozłączyć baterię i był swobodny dostęp do wszystkich pinów i elementów układu. Poniższe zdjęcie przedstawia zaproponowane przez nas układ kabli wykorzystując specjalne przygotowane formy utrzymujące układ oraz baterie na miejscu



20. Efekt końcowy

Udało nam się zbudować w pełni działający model poduszkowca wraz z jego sterowaniem. Warto tutaj nadmienić, że sterowanie poduszkowcem jest trudnym wyzwaniem ponieważ elementy skrętne są usytuowane z tyłu modelu. Przypomina to ciągłą jazdę autem na biegu wstecznym. Nie pomaga także fakt, że najlepsze efekty uzyskuje się jadąc z pewną prędkością.

Warto tutaj także wspomnieć, że użyte w projekcie silniki są bardzo silne i przeważnie ustawianie pracy obu poniżej połowy jest wystarczające. Ustawianie powyżej 3/4 mocy jest odradzane i użytkownik robi to na własną odpowiedzialność (ponieważ przy takim ciągu śmigła prędkość jaką uzyskuje poduszkowiec jest niszcząca zarówno dla konstrukcji wieży bądź fartucha), nie wspominając o niezwykle trudnym manewrowaniu sterem przy takiej prędkości ponieważ serwo może mieć problemy wykonywaniem ruchów.

21. Podsumowanie

Stworzony przez nas projekt pokazuje, że przy odrobinie chęci można stworzyć własnoręcznie coś, co na półkach sklepowych sprzedaje się jako produkt gotowy. Dodatkowo zyskujemy pełną świadomość jak dana część działa co pozwala nam na dostosowanie takich parametrów jakich chcemy oraz na ewentualne naprawy lub poprawienie wydajności.

W czasie tworzenia modelu napotkaliśmy na problemy związane głównie z utrzymaniem sztywności i wytrzymałości konstrukcji przy utrzymaniu jej lekkości, z uzyskaniem częstotliwości wysyłania danych do modułu o wartości, która pozwalała na swobodne kontrolowanie poduszkowca oraz stworzenie takiego układu kabli z których składał się moduł komunikacji, aby nie wypadały one i nie trzymały się "na słowo honoru".

22. Bibliografia i przydatne linki

1. Instrukcja do regulatora ABC-POWER: http://abc-rc.pl/templates/images/files/995/1384185241-instrukcja-esc-abc-power-4881.pdf
2. Dokumentacja Pythona: https://www.python.org/doc/
3. Android SDK: http://developer.android.com/sdk/index.html
4. Arduino IDE: https://www.arduino.cc/en/Main/Software

5. Forum ESP8266:
   http://www.esp8266.com/
6. Kivy:
   https://kivy.org/
7. Buildozer:
   https://github.com/kivy/buildozer