UNIVERSITATEA TITU MAIORESCU

FACULTATEA: INFORMATICĂ
DEPARTAMENT: INFORMATICĂ
Programa de studii: INFORMATICĂ
DISCIPLINA: INTELIGENȚĂ ARTIFICIALĂ

IA - *Testul de evaluare nr.* 5

Robot autonom aerian – QUADROCOPTER

EXAMEN

Nr. crt.	Grupa	Numele și prenumele	Semnătură student	Notă evaluare
1				
2				
3				
4				
5				
6				

Data: __/__/_____

CS I dr.ing.
Lucian Ștefăniță GRIGORE

Conf.dr.ing.
Iustin PRIESCU

Ș.l. dr.ing.
Dan-Laurențiu GRECU

Cuprins

1. SOFTWARE QUADROCOPTER

Ca și în cazul tuturor schițelor Arduino[1], există o rutină de configurare și o bucla principală. Setarea inițializează totul, iar bucla principală efectuează intrarea, calculul și ieșirea.

Fig. 1-1 Quadrocopter Turbolinx-TL-300
https://www.rcgroups.com/forums/showthread.php?2400369-Turbolinx-Tl-300

Semnalele PWM de la receptorul RC sunt preluate cu ajutorul "întreruperilor de schimbare a pinului". Când se schimbă tensiunea pe pinul de intrare, se apelează rutina de întrerupere. Lățimea pulsului se calculează pe baza diferențelor dintre timbrele de timp.

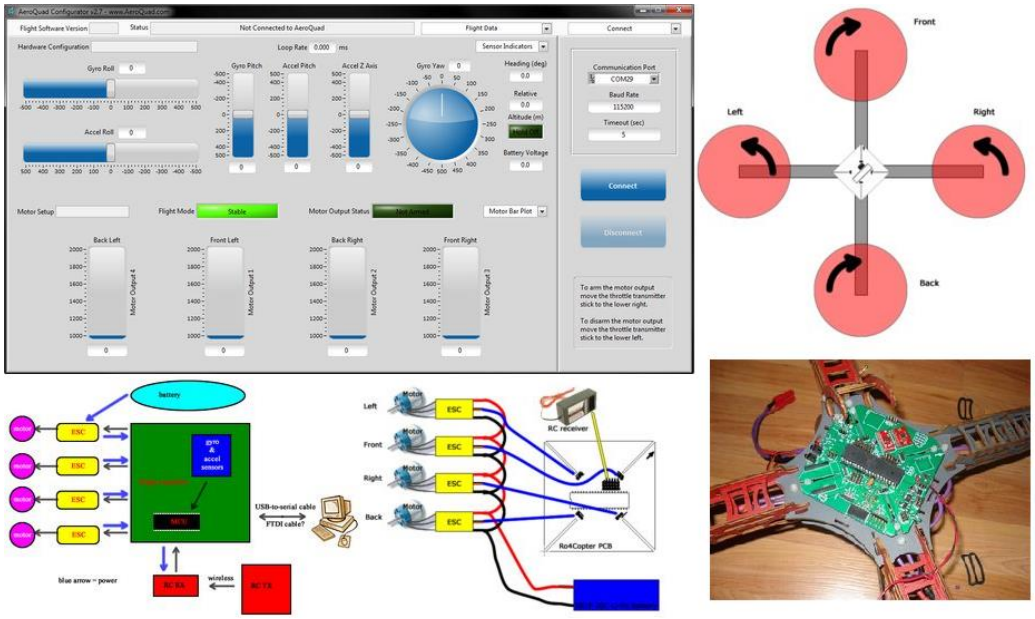


Fig. 1-2 Schematizarea sistemului de comandă și control.

[1] http://www.instructables.com/id/RC-Quadrotor-Helicopter/

Semnalul de ieșire a semnalului PWM către ESC este generat folosind mai mulți cronometre cu "comparați ieșirile de potrivire", ceea ce înseamnă că lățimea pulsului este setată și cronometrul va crea automat impulsul, atribuit pinului corespunzător, pentru perioada de timp stabilită.

Senzorii comunică utilizând magistrala I2C, care este o magistrală sincronă de date care are un suport excelent pentru dispozitivele multiple, folosind numai două fire.

Buclele principale sunt împărțite în sarcini, fiecare executată la o frecvență diferită. Acest lucru dă prioritate fiecărei sarcini. Codul care păstrează stabilitatea quadrocopterului are cea mai mare prioritate. Lucrurile cum ar fi comunicarea cu computerul au mai puțină prioritate. Este posibil să se adauge un codul de navigare pentru GPS dacă este acordată o prioritate mai mică.

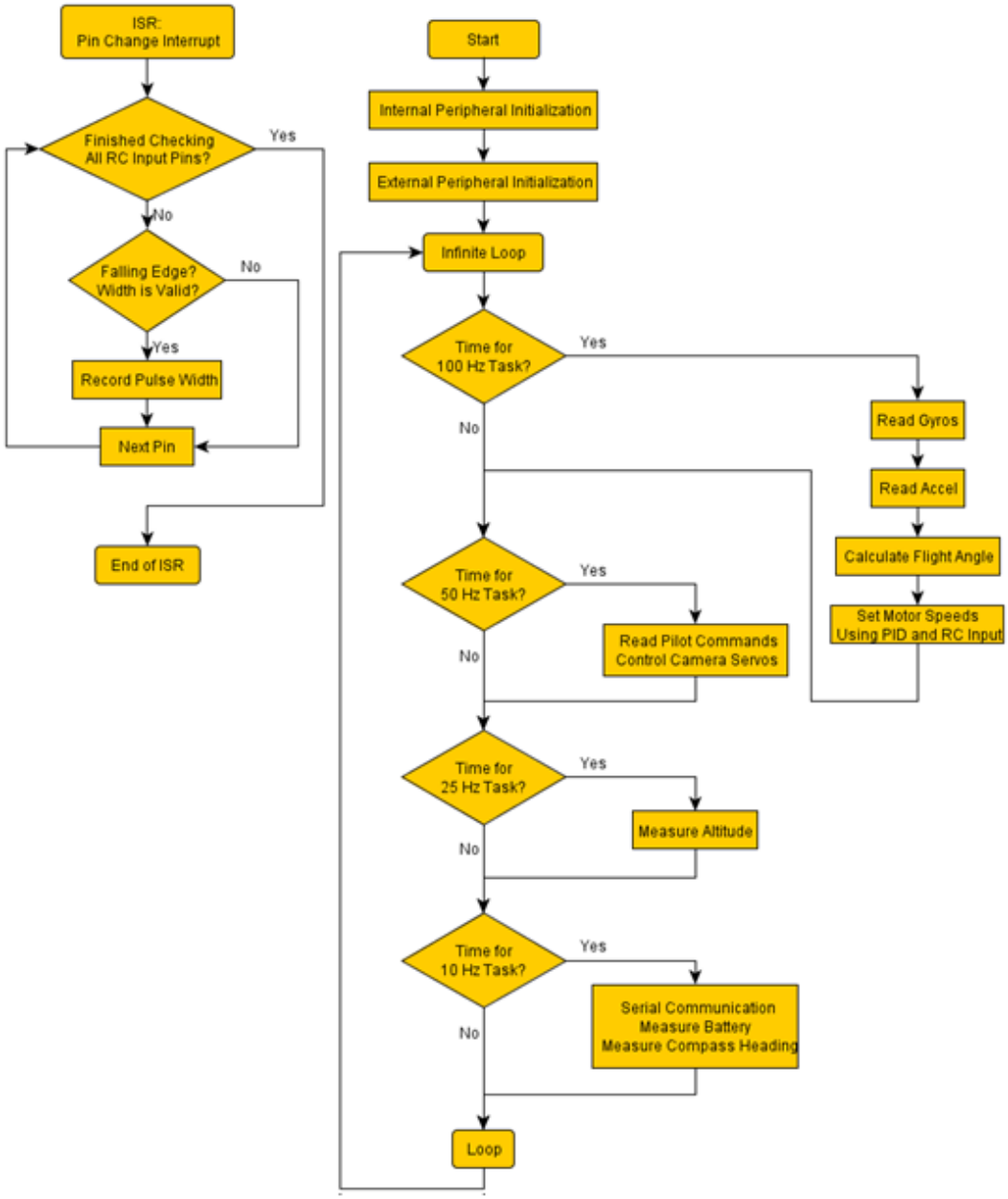
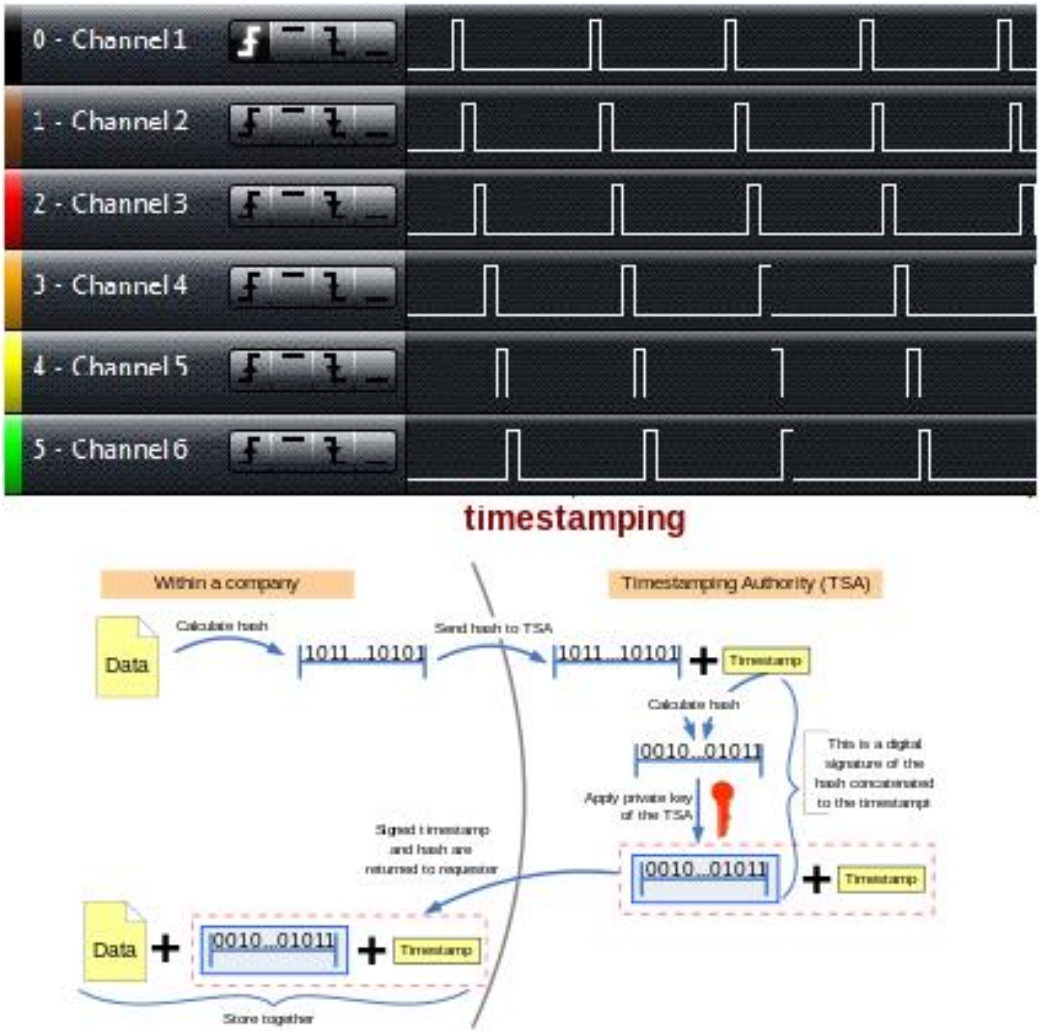


Fig. 1-3 Schema logică.

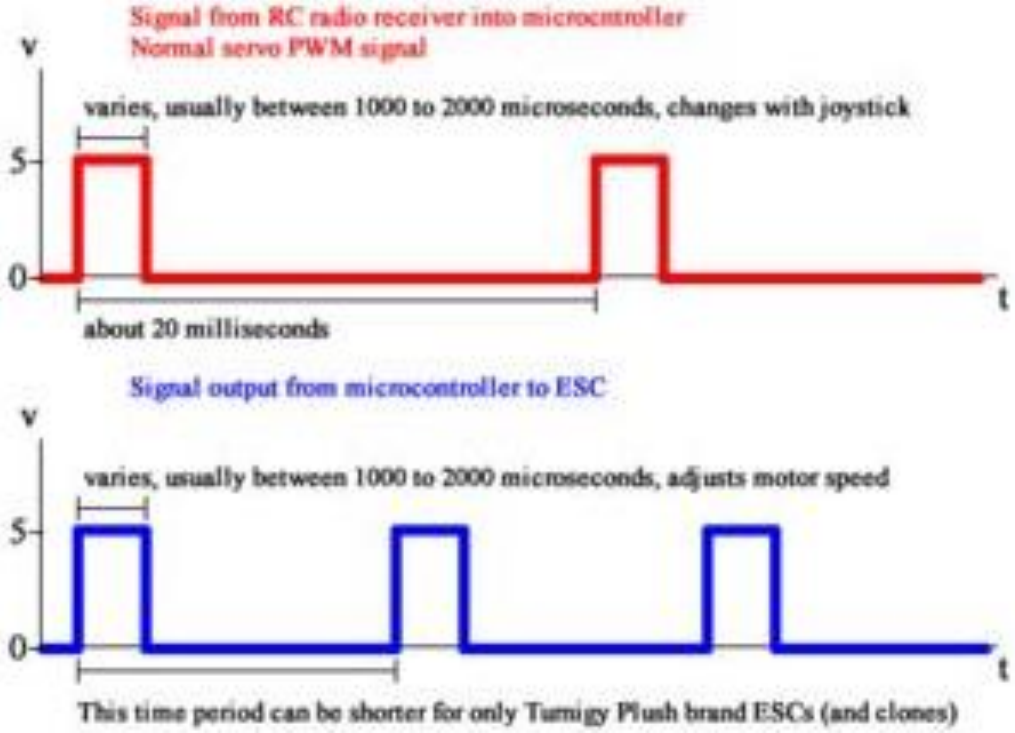
1.1 PWM Input

Sub Arduino pin-change are rolul de a întrerupe și un cronometru pentru a citi lățimea impulsuri de la mai multe semnale, pe diferitele fire de alimentare. Ori de câte ori se detectează o schimbare a stării pinului, vectorul de întrerupere înregistrează un timestamp[2] cu ajutorul cronometrului, iar diferența dintre marcatorii de timp este lățimea impulsului. Semnalele receptoarelor radio RC, ca impulsuri au o lățime între 1000 și 2000 de microsecunde, perioada fiind de aproximativ 20 de milisecunde:

- PWM;s
- semnale servo;
- ISR în AVR (routina de întrerupere a serviciului);
- PCINT, întreruperea schimbării pinului;
- timpi de 16 biți în AVR



[2] https://tools.ietf.org/html/rfc3628




```
void setup()
{
  DDRC = 0; // pins as input

  // enable PCINT 18 to 23
  PCICR |= (1 << PCIE2);
  PCMSK2 = 0xFC;

  Serial.begin(115200);
}

typedef struct {
  byte edge;
  unsigned long riseTime;
  unsigned long fallTime;
  unsigned int lastGoodWidth;
} tPinTimingData;
volatile static tPinTimingData pinData[6 + 1];
volatile static uint8_t PCintLast;
```

```
ISR(PCINT2_vect)
{
  uint8_t bit;
  uint8_t curr;
  uint8_t mask;
  uint32_t currentTime;
  uint32_t time;

  // get the pin states for the indicated port.
  curr = PINC & 0xFC;
  mask = curr ^ PCintLast;
  PCintLast = curr;

  currentTime = micros();

  // mask is pcint pins that have changed.
  for (uint8_t i=0; i < 6; i++) {
    bit = 0x04 << i;
    if (bit & mask) {
      // for each pin changed, record time of change
      if (bit & PCintLast) {
        time = currentTime - pinData[i].fallTime;
        pinData[i].riseTime = currentTime;
        if ((time >= 10000) && (time <= 26000))
          pinData[i].edge = 1;
        else
          pinData[i].edge = 0; // invalid rising edge detected
      }
      else {
        time = currentTime - pinData[i].riseTime;
        pinData[i].fallTime = currentTime;
        if ((time >= 800) && (time <= 2200) && (pinData[i].edge == 1)) {
          pinData[i].lastGoodWidth = time;
          pinData[i].edge = 0;
        }
      }
    }
```

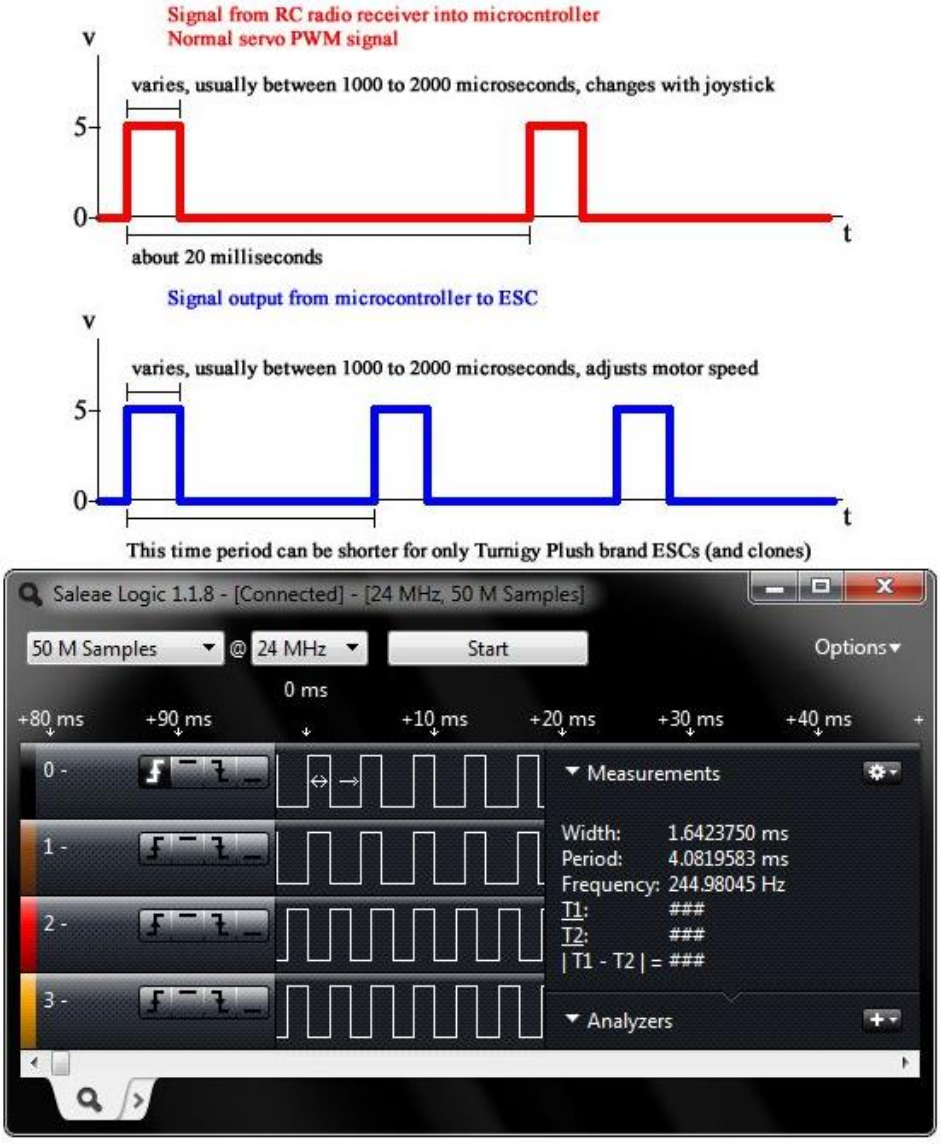
```
  }
}

void loop()
{
  Serial.println();
  for (byte i = 0; i < 6; i++) {
    Serial.print("C");
    Serial.print((int)i + 1);
    Serial.print(": ");
    Serial.print(pinData[i].lastGoodWidth);
    Serial.print(", ");
  }

  delay(500);
}
```

1.2 PWM Output

În continuare se prezintă codul care generează semnale PWM pentru fiecare ESC, astfel încât, fiecare motor să genereze o rotație corespunzătoare regimului de funcționare solicitat.

Funcția de temporizare a timpului de ieșire este utilizată pentru a genera semnale PWM. Când cronometrul ajunge la 0, acesta pornește un pin, când atinge un anumit număr (pe care îl specificăm), oprește pinul. Aceasta generează un semnal pătratic corespunzător unui ciclu de sarcină variabil.

Deși semnalele servo normale au de obicei o perioadă de aproximativ 20 ms (adică o frecvență de aproximativ 50 Hz), ieșirea de la microcontrolerul nostru are o perioadă mai scurtă (deci o frecvență mai mare, aproximativ 250÷300 Hz). Este cunoscut faptul că ESC-urile Turnigy Plush (și clonele sale sub marca Hobby King) sunt capabile să se lucreze la frecvențe mai ridicate, lucru care va permite mai multe ajustări ale turației motorului pe secundă, astfel încât quadrocopterul va deveni mai stabil.



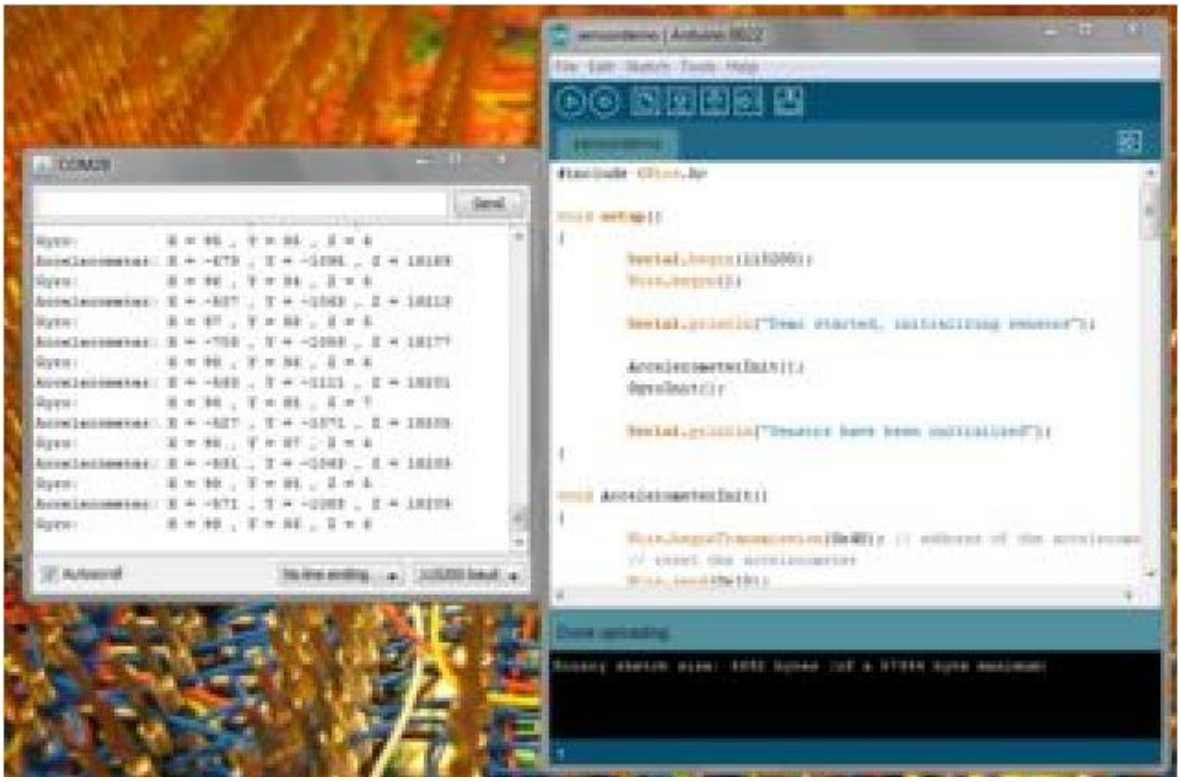
```
#define PWM_FREQUENCY 300 // in Hz
#define PWM_PRESCALER 8
#define PWM_COUNTER_PERIOD (F_CPU/PWM_PRESCALER/PWM_FREQUENCY)

void setup()
{
  // pins as output
  DDRD |= (1 << 4) | (1 << 5) | (1 << 6) | (1 << 7);

  // default to 1000 microsecond pulse width
  OCR1A = 1000 * 2 ;
  OCR1B = 1000 * 2 ;
  OCR2A = 1000 / 16 ;
  OCR2B = 1000 / 16 ;

  // the setup is:
  // Clear OCnA/OCnB on compare match, set OCnA/OCnB at BOTTOM (non-inverting mode)

  // setup timer 1
  TCCR1A = (1<<WGM11)|(1<<COM1A1)|(1<<COM1B1);
  TCCR1B = (1<<WGM13)|(1<<WGM12)|(1<<CS11);
  ICR1 = PWM_COUNTER_PERIOD;

  // setup timer 2
  TCCR2A = (1<<WGM20)|(1<<WGM21)|(1<<COM2A1)|(1<<COM2B1);
  TCCR2B = (1<<CS22)|(1<<CS21);
  // the period is fixed for timer 2, it's about 244 Hz

  // note that timer1 is a 16-bit timer and timer2 is a 8-bit timer
}

void loop()
{
  int pw;
  for (pw = 1000; pw <= 2000; pw += 20)
  {
    OCR1A = pw * 2 ;
    OCR1B = pw * 2 ;
```

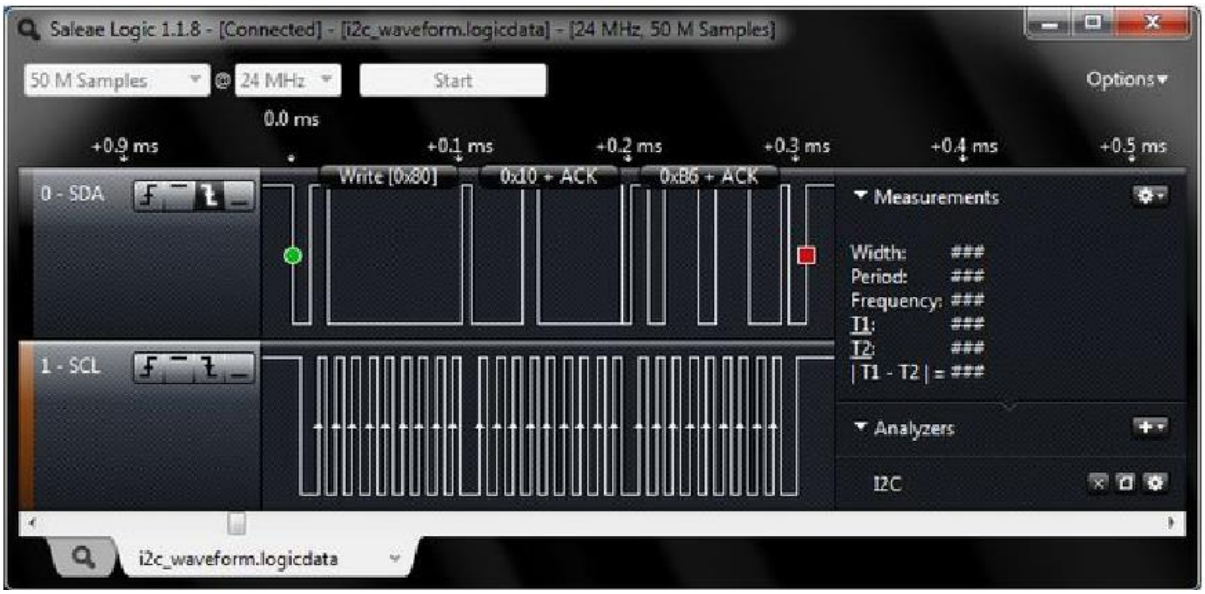
```
    OCR2A = pw / 16 ;
    OCR2B = pw / 16 ;

    delay(10);
  }

  for (pw = 2000; pw >= 1000; pw -= 20)
  {
    OCR1A = pw * 2 ;
    OCR1B = pw * 2 ;
    OCR2A = pw / 16 ;
    OCR2B = pw / 16 ;

    delay(10);
  }
}
```

2. ATMEGA168A PULSE WIDH MODULATION – PWM

Coordonarea senzorilor, cum ar fi: accelerometrul BMA180, gyro ITG-3200. Senzorii sunt conectați la microcontrolerul Atmega168A printr-o magistrală I2C. Acest tip de magistrală este conceput pentru a permite mai multor dispozitive să comunice împreună folosind doar două fire.

În următoarea figură se prezintă forma de undă a unui analizor logic al semnalelor de magistrală I2C.





Software-ul pentru controlerul de zbor va iniția accelerometrul inițial resetându-l, apoi stabilind un filtru de trecere de 10 [Hz] și setând intervalul de citire la +/- 2 G.

Software-ul pentru controlerul de zbor va iniția senzorul giroscopului, resetându-l mai întâi, apoi instalându-se filtrul său de trecere de 10 [Hz] și comandând-ui să utilizeze propriul oscilator intern. Acest senzor permite utilizarea unui oscilator extern, care de asemenea are ieșiri digitale pentru senzorii de temperatură.

```
#include <Wire.h>
void setup()
{
  Serial.begin(115200);
  Wire.begin();

  Serial.println("Demo started, initializing sensors");

  AccelerometerInit();
  GyroInit();

  Serial.println("Sensors have been initialized");
}

void AccelerometerInit()
{
  Wire.beginTransmission(0x40); // address of the accelerometer
  // reset the accelerometer
  Wire.send(0x10);
  Wire.send(0xB6);
  Wire.endTransmission();
  delay(10);

  Wire.beginTransmission(0x40); // address of the accelerometer
  // low pass filter, range settings
  Wire.send(0x0D);
  Wire.send(0x10);
  Wire.endTransmission();

  Wire.beginTransmission(0x40); // address of the accelerometer
```

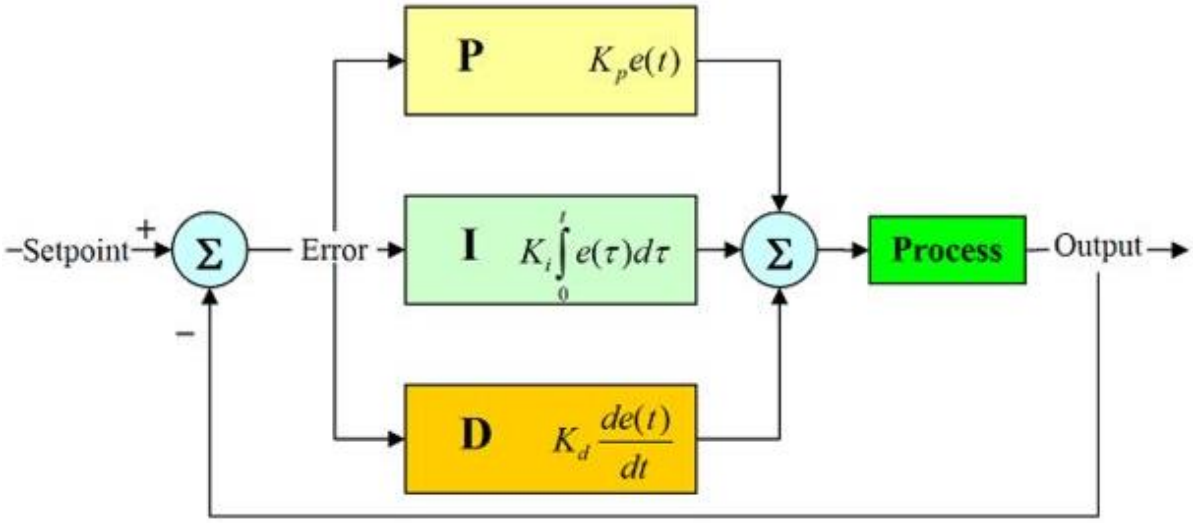
```
  Wire.send(0x20); // read from here
  Wire.endTransmission();
  Wire.requestFrom(0x40, 1);
  byte data = Wire.receive();
  Wire.beginTransmission(0x40); // address of the accelerometer
  Wire.send(0x20);
  Wire.send(data & 0x0F); // low pass filter to 10 Hz
  Wire.endTransmission();

  Wire.beginTransmission(0x40); // address of the accelerometer
  Wire.send(0x35); // read from here
  Wire.endTransmission();
  Wire.requestFrom(0x40, 1);
  data = Wire.receive();
  Wire.beginTransmission(0x40); // address of the accelerometer
  Wire.send(0x35);
  Wire.send((data & 0xF1) | 0x04); // range +/- 2g
  Wire.endTransmission();
}

void AccelerometerRead()
{
  Wire.beginTransmission(0x40); // address of the accelerometer
  Wire.send(0x02); // set read pointer to data
  Wire.endTransmission();
  Wire.requestFrom(0x40, 6);

  // read in the 3 axis data, each one is 16 bits
  // print the data to terminal
  Serial.print("Accelerometer: X = ");
  short data = Wire.receive();
  data += Wire.receive() << 8;
  Serial.print(data);
  Serial.print(" , Y = ");
  data = Wire.receive();
  data += Wire.receive() << 8;
  Serial.print(data);
  Serial.print(" , Z = ");
```

```
  data = Wire.receive();
  data += Wire.receive() << 8;
  Serial.print(data);
  Serial.println();
}

void GyroInit()
{
  Wire.beginTransmission(0x69); // address of the gyro
  // reset the gyro
  Wire.send(0x3E);
  Wire.send(0x80);
  Wire.endTransmission();

  Wire.beginTransmission(0x69); // address of the gyro
  // low pass filter 10 Hz
  Wire.send(0x16);
  Wire.send(0x1D);
  Wire.endTransmission();

  Wire.beginTransmission(0x69); // address of the gyro
  // use internal oscillator
  Wire.send(0x3E);
  Wire.send(0x01);
  Wire.endTransmission();
}

void GyroRead()
{
  Wire.beginTransmission(0x69); // address of the gyro
  Wire.send(0x1D); // set read pointer
  Wire.endTransmission();

  Wire.requestFrom(0x69, 6);

  // read in 3 axis of data, 16 bits each, print to terminal
  short data = Wire.receive() << 8;
  data += Wire.receive();
```

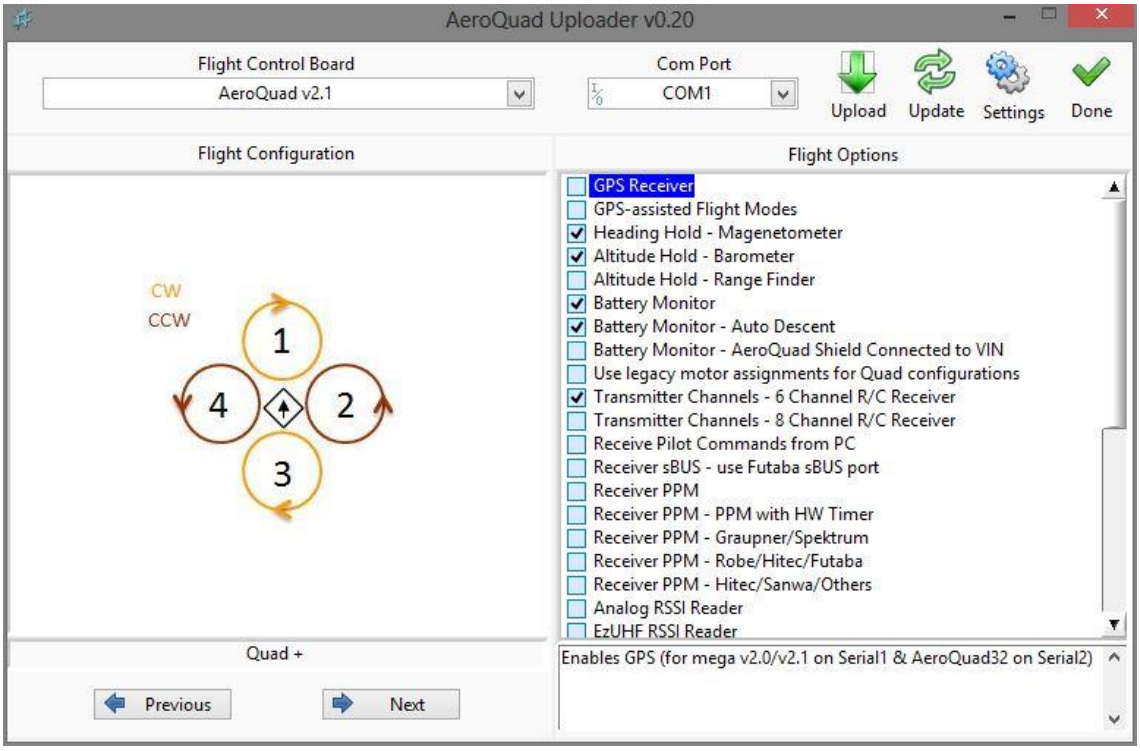
```
  Serial.print("Gyro: X = ");
  Serial.print(data);
  Serial.print(" , Y = ");
  data = Wire.receive() << 8;
  data += Wire.receive();
  Serial.print(data);
  Serial.print(" , Z = ");
  data = Wire.receive() << 8;
  data += Wire.receive();
  Serial.print(data);
  Serial.println();
}

void loop()
{
  AccelerometerRead();
  GyroRead();

  delay(500); // slow down output
}
```

3. CONTROLUL ZBORULUI

Codurile prezentate până acum facilitează controlul zborului în două moduri.

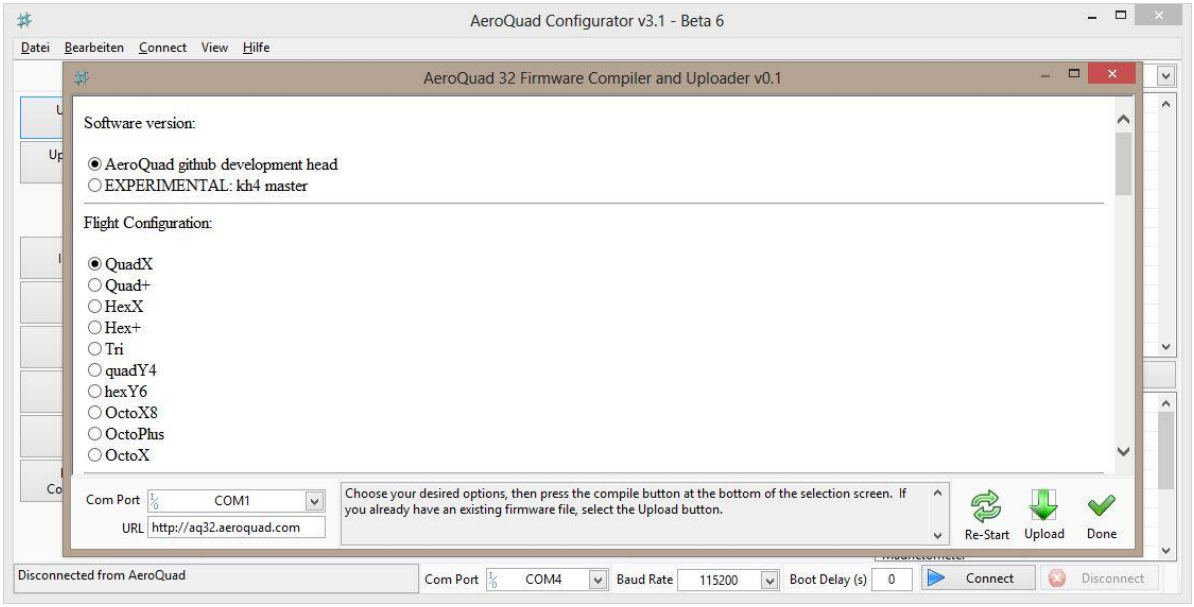


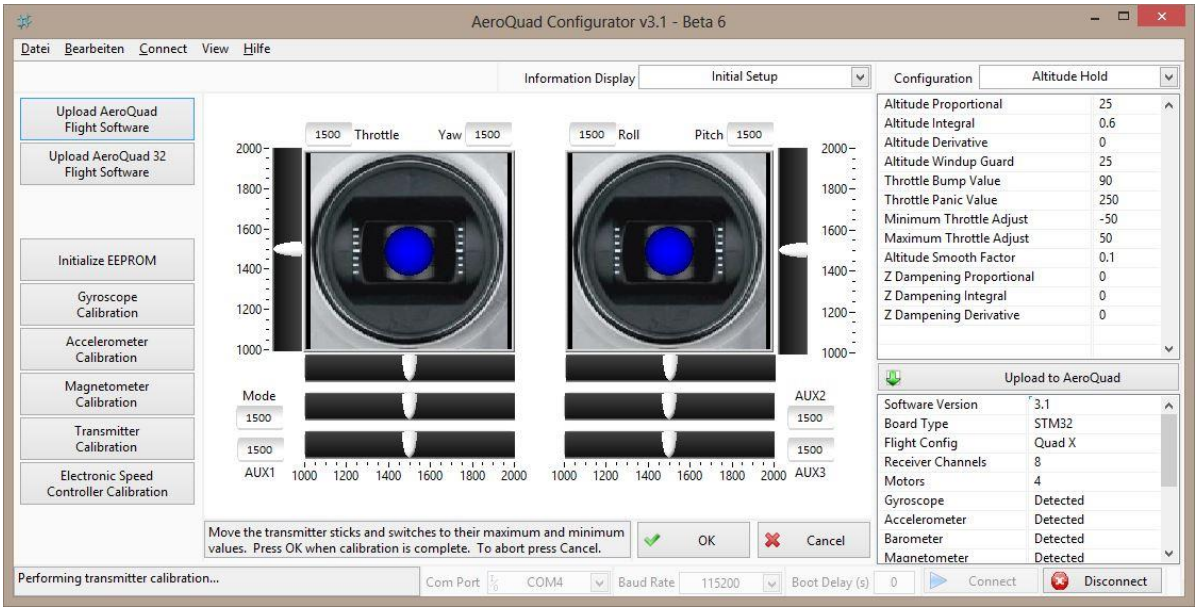
Controlul zborului quadrocopterului cu un joystick ne conduce la ideea unui zbor acrobatic. Aceasta din cauza faptului că accelerometrul este ignorat, controlul fiind efectuat doar asupra vitezei unghiulare. În acest mod codul permite rotirea dronei în orice moment, indiferent de semnalele transmise de către senzori, de asemenea poate fi oprit oricând, poate efectua looping-uri (răsuciri peste cap) sau tonouri (rulouri / rotiri în jurul axului transversal).

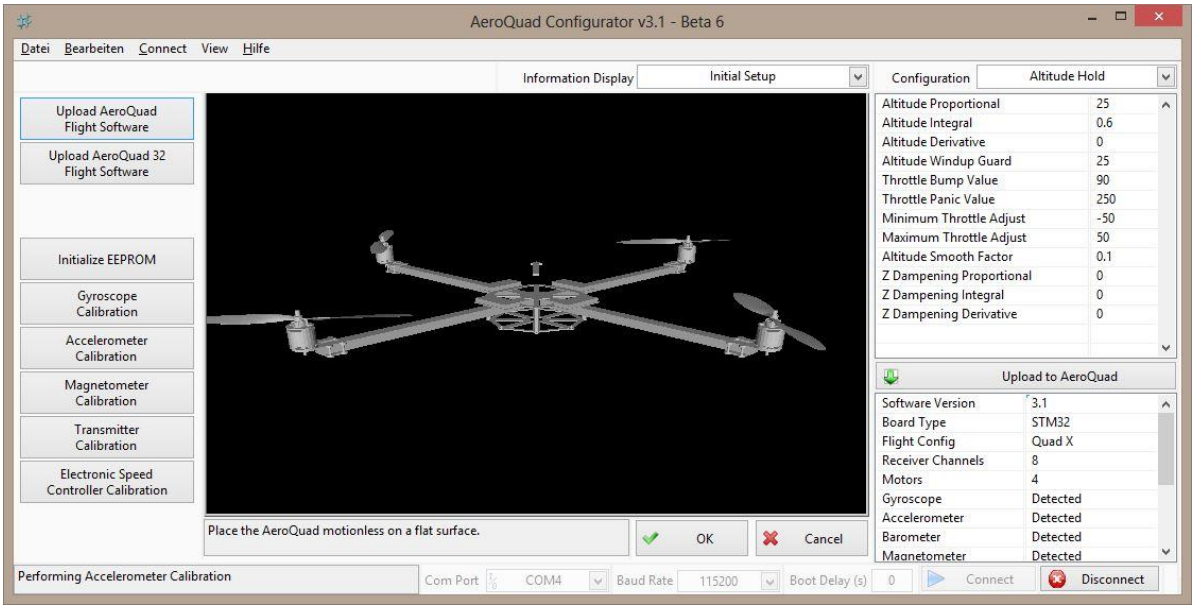


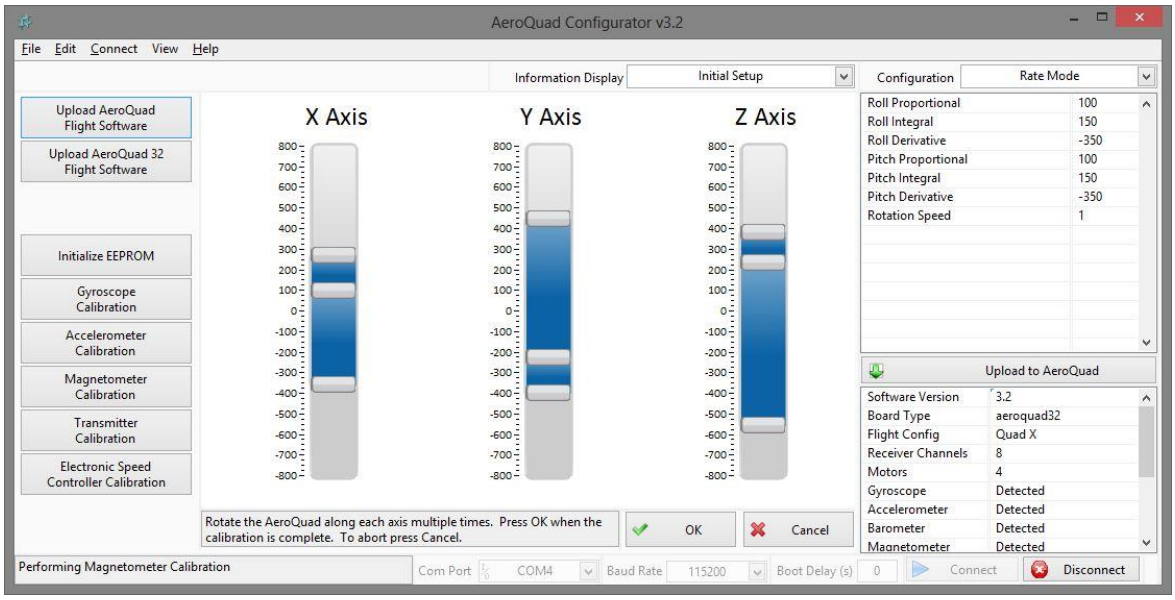
Al doilea mod de coordonare al quadrocopterului este cel care ține cont de informațiile primite de la senzori - accelerometru, giroscop - care calculează unghiul de atac curent al dronei. Acest mod presupune renunțarea la joystick, altfel spus vom observa un zbor orizontal (echilibrat). Mutarea joystick-ului într-o poziție non-centrală, va imprima robotului o mișcare de răsucire (modificare a unghiului de atac) iar imediat ce manșa este eliberată, quadrocopterul va reveni la poziția stabilitățprin cod. Acest mod se numește stabil (stable / attitude). Acest mod nu permite

Calculul unghiul curent al quadrocopterului se face în baza citirii vitezei unghiulare de către senzorul giroscopic. Viteza unghiulară înmulțită cu timpul este egală cu cantitatea de rotație din timpul respectiv. Există o mulțime de erori în acest proces datorită deviației și zgomotului din senzorul giroscopic. Pentru a corecta această eroare, accelerometrul este utilizat pentru a măsura accelerația gravitațională.





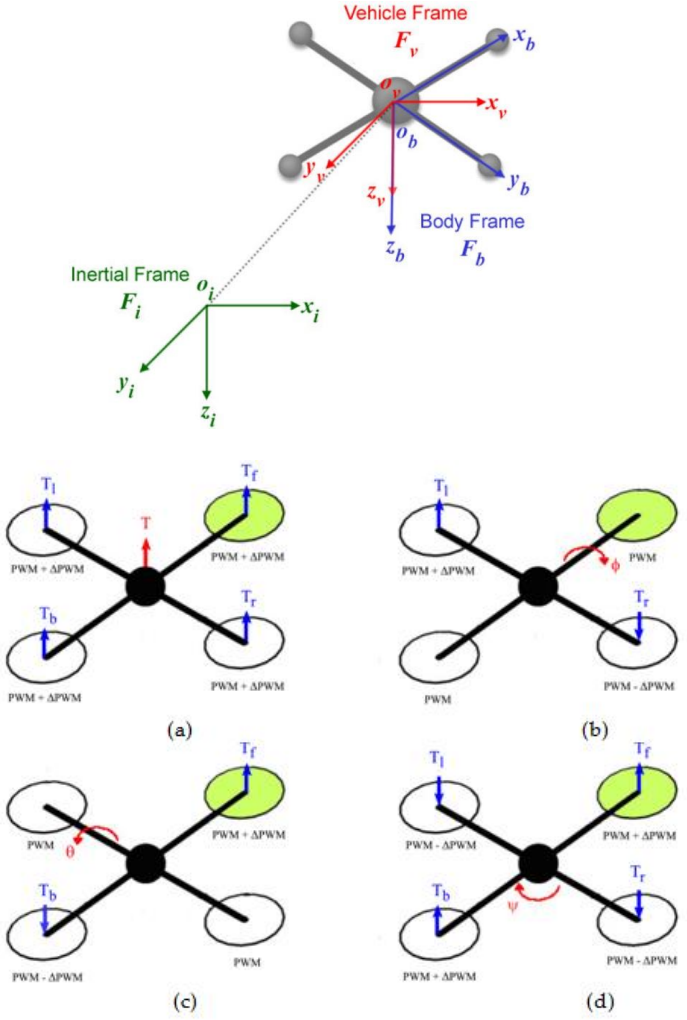


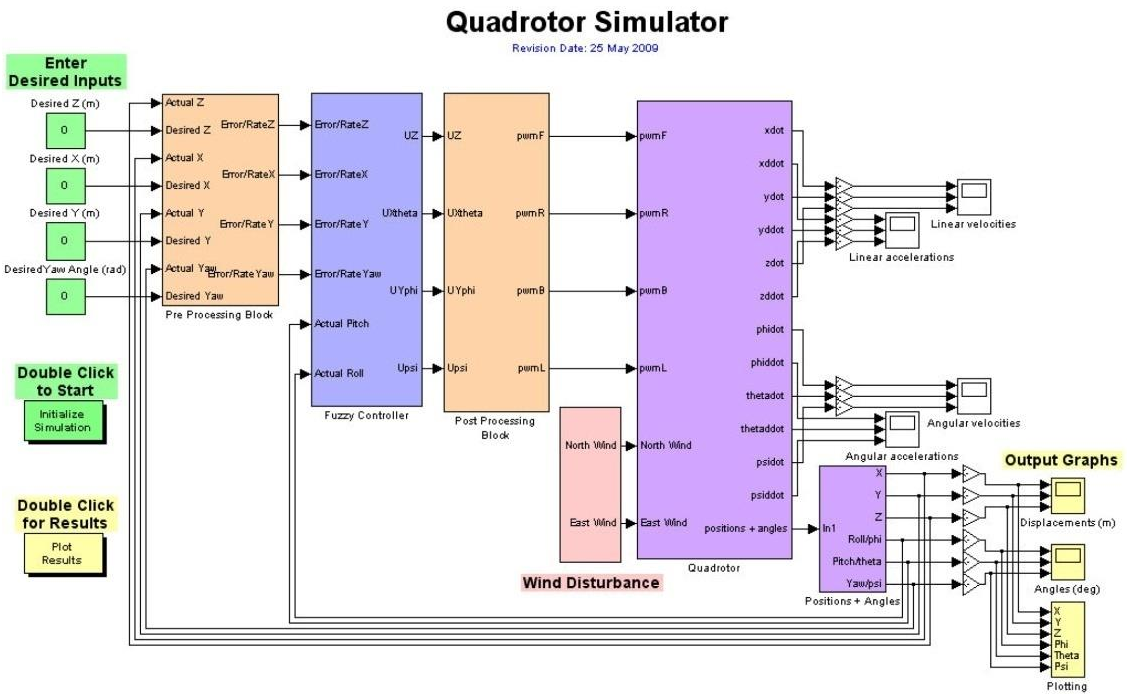


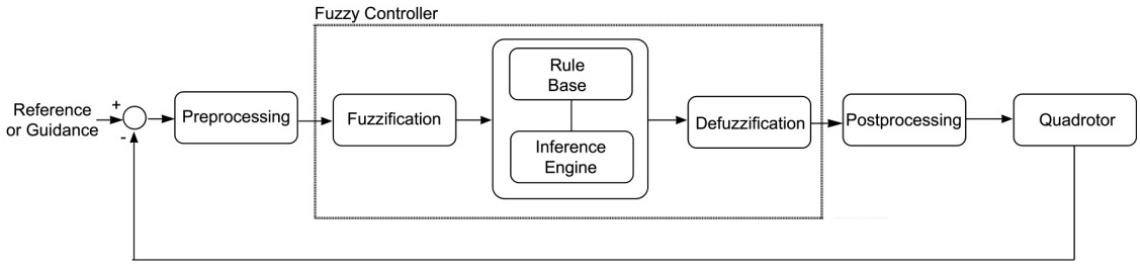
Sistemul de control are la bază PID:

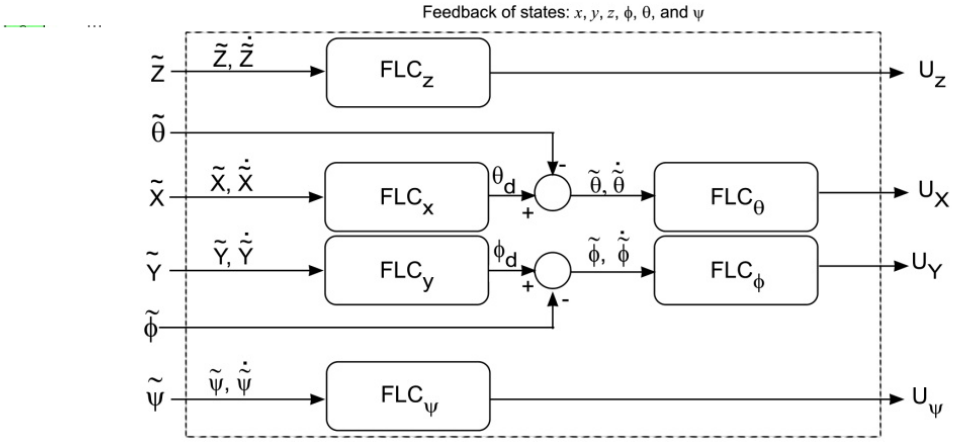
- termenul "P", are rolul de a gestiona atingerea unei ținte – dacă este la distanță se accelerează, când se aproprie trebuie redusă viteza, etc.;
- termenul "I", dacă robotul este înfrânat de un motiv sau altul motoarele vor primi comanda de creștere a vitezei, iar semnalul este unul de tip integrat;
- termenul "D", calculează viteza pe baza datelor despre diferența dintre poziția actuală și cea dorită;

Prin urmare un PID în cazul quadrocopterelor ajustează unghiurile de atac în funcție de viteza de rotație a motorului.






Fuzzy Controller
Reference or Guidance
+
-
Preprocessing
Fuzzification
Rule Base
Inference Engine
Defuzzification
Postprocessing
Quadrotor


Feedback of states: x, y, z, ϕ, θ, and ψ
FLC_z
FLC_x
FLC_y
FLC_θ
FLC_ϕ
FLC_ψ
U_z
U_X
U_Y
U_ψ

4. CALCULATOR CARACTERISTICI QUADROCOPTER

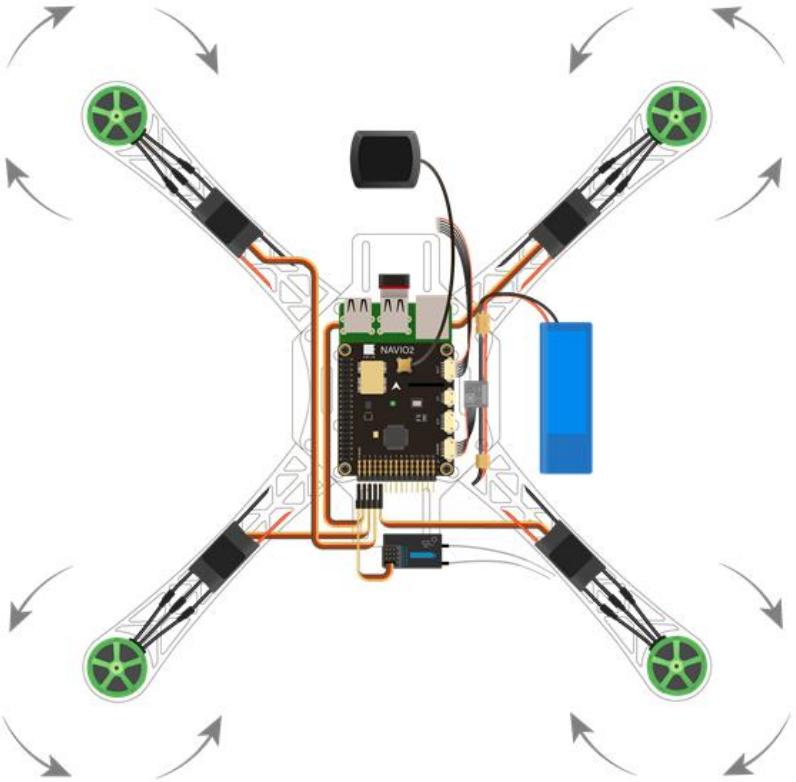
Calculator multicopter[3], [4] (MotoCalc - MotoWizard) este util pentru a ne ajuta să găsim cea mai bună setare pentru modelul ales.

De asemenea (MotoCalc – Workbench) permite dimensionarea completă, plecând de la următoarele date: motorul, tipul de celule, numărul de celule (serie și / sau paralel), gama de dimensiuni ale elicei și/sau rapoarte de viteză, controlul vitezei, modul în care trebuie să fie conectate împreună și informații mai detaliate despre modelul în sine.

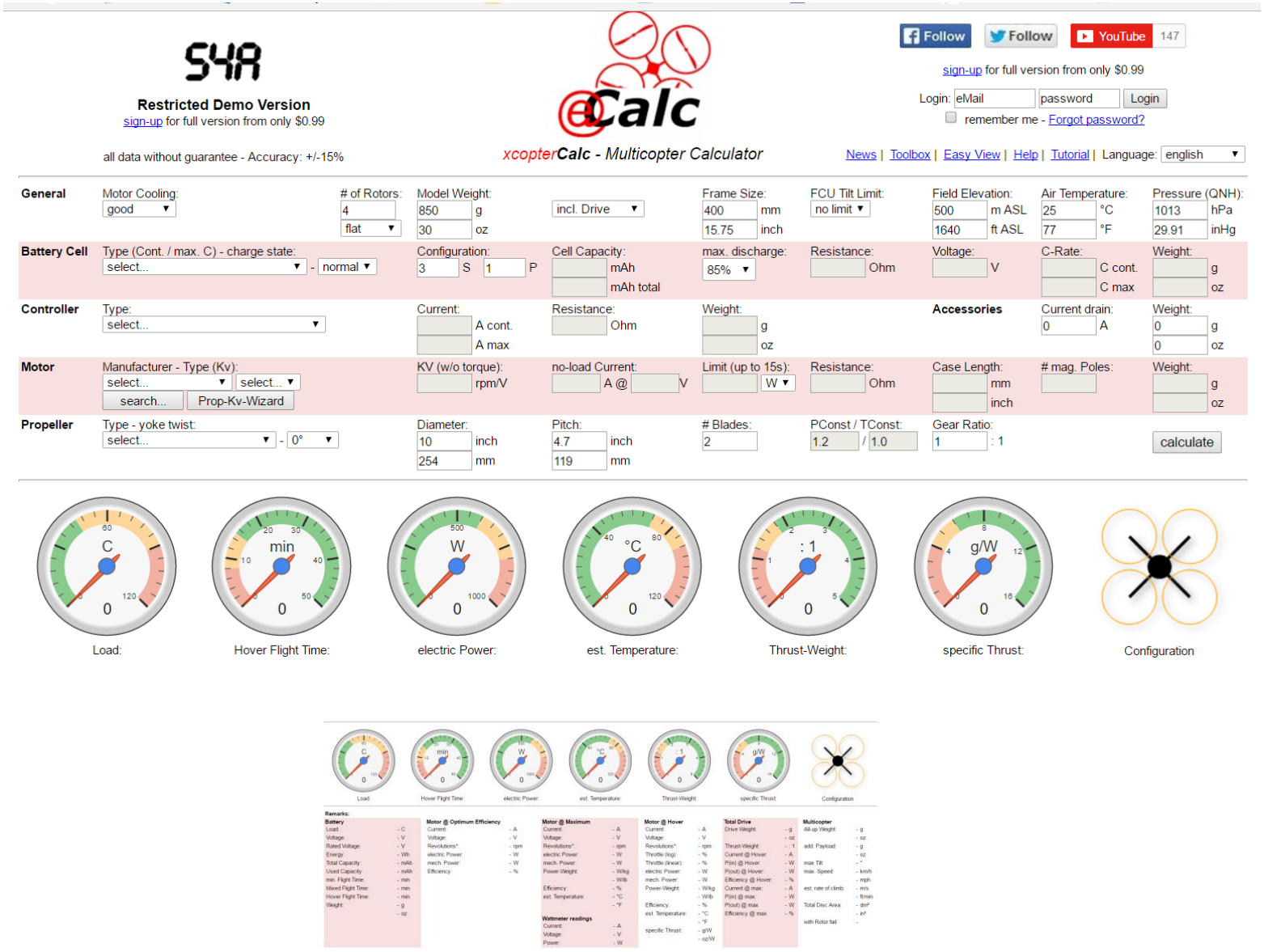
Previziunile statice permit calculul tensiunii, curentului, tensiunea terminalelor motorului, puterea de intrare și puterea de încărcare, pierderea de putere, puterea de ieșire și puterea de încărcare, eficiența motorului / transmisiei, eficiența bateriei-arbore, viteza de împingere, viteza pitch, rata de urcare și timpul de execuție.

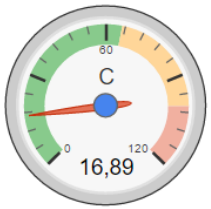
Această facilitate crează tabele de predicții, pentru a determina dimensiunea optimă a elicei și / sau raportul de transmisie pentru aplicația dorită.

Predicția de zbor permite efectuarea unei analize a zborului pentru o anumită combinație de componente, prezicând ridicarea, tracțiunea, curentul, tensiunea, puterea, eficiența, turația, diferite viteze de zbor.



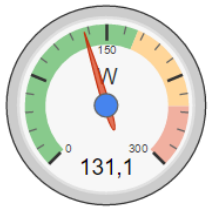
[3] https://www.ecalc.ch/xcoptercalc.php?ecalc&lang=en
[4] http://controls.ae.gatech.edu/dbershad/EMSTAirTimeCalculator.html


Restricted Demo Version
sign-up for full version from only $0.99
all data without guarantee - Accuracy: +/-15%
eCalc
xcopterCalc - Multicopter Calculator
Follow
Follow
YouTube 147
sign-up for full version from only $0.99
Login: eMail password Login
remember me - Forgot password?
News | Toolbox | Easy View | Help | Tutorial | Language: english
General
Motor Cooling: good
# of Rotors: 4 flat
Model Weight: 850 g 30 oz incl. Drive
Frame Size: 400 mm 15.75 inch
FCU Tilt Limit: no limit
Field Elevation: 500 m ASL 1640 ft ASL
Air Temperature: 25 °C 77 °F
Pressure (QNH): 1013 hPa 29.91 inHg
Battery Cell
Type (Cont. / max. C) - charge state: select... - normal
Configuration: 3 S 1 P
Cell Capacity: mAh mAh total
max. discharge: 85%
Resistance: Ohm
Voltage: V
C-Rate: C cont. C max
Weight: g oz
Controller
Type: select...
Current: A cont. A max
Resistance: Ohm
Weight: g oz
Accessories
Current drain: 0 A
Weight: 0 g 0 oz
Motor
Manufacturer - Type (Kv): select... select... search... Prop-Kv-Wizard
KV (w/o torque): rpm/V
no-load Current: A @ V
Limit (up to 15s): W
Resistance: Ohm
Case Length: mm inch
mag. Poles:
Weight: g oz
Propeller
Type - yoke twist: select... - 0°
Diameter: 10 inch 254 mm
Pitch: 4.7 inch 119 mm
Blades: 2
PConst / TConst: 1.2 / 1.0
Gear Ratio: 1 : 1
calculate
Load:
Hover Flight Time:
electric Power:
est. Temperature:
Thrust-Weight:
specific Thrust:
Configuration

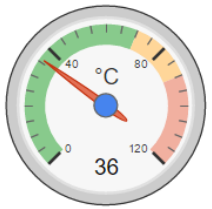


Load:

Hover Flight Time:

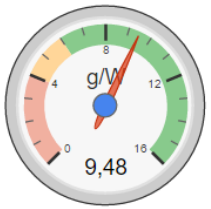


electric Power:

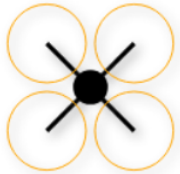


est. Temperature:

Thrust-Weight:



specific Thrust:


Configuration

Remarks:

Battery		
Load:	16.89	C
Voltage:	10.54	V
Rated Voltage:	11.10	V
Energy:	33.3	Wh
Total Capacity:	3000	mAh
Used Capacity:	2550	mAh
min. Flight Time:	3.0	min
Mixed Flight Time:	10.4	min
Hover Flight Time:	18.7	min
Weight:	252	g
	8.9	oz

Motor @ Optimum Efficiency		
Current:	8.55	A
Voltage:	10.59	V
Revolutions*:	8908	rpm
electric Power:	90.5	W
mech. Power:	75.6	W
Efficiency:	83.5	%

Motor @ Maximum		
Current:	12.67	A
Voltage:	10.35	V
Revolutions*:	8280	rpm
electric Power:	131.1	W
mech. Power:	107.6	W
Power-Weight:	616.8	W/kg
	279.8	W/lb
Efficiency:	82.1	%
est. Temperature:	36	°C
	97	°F

Wattmeter readings		
Current:	50.68	A
Voltage:	10.54	V
Power:	534.2	W

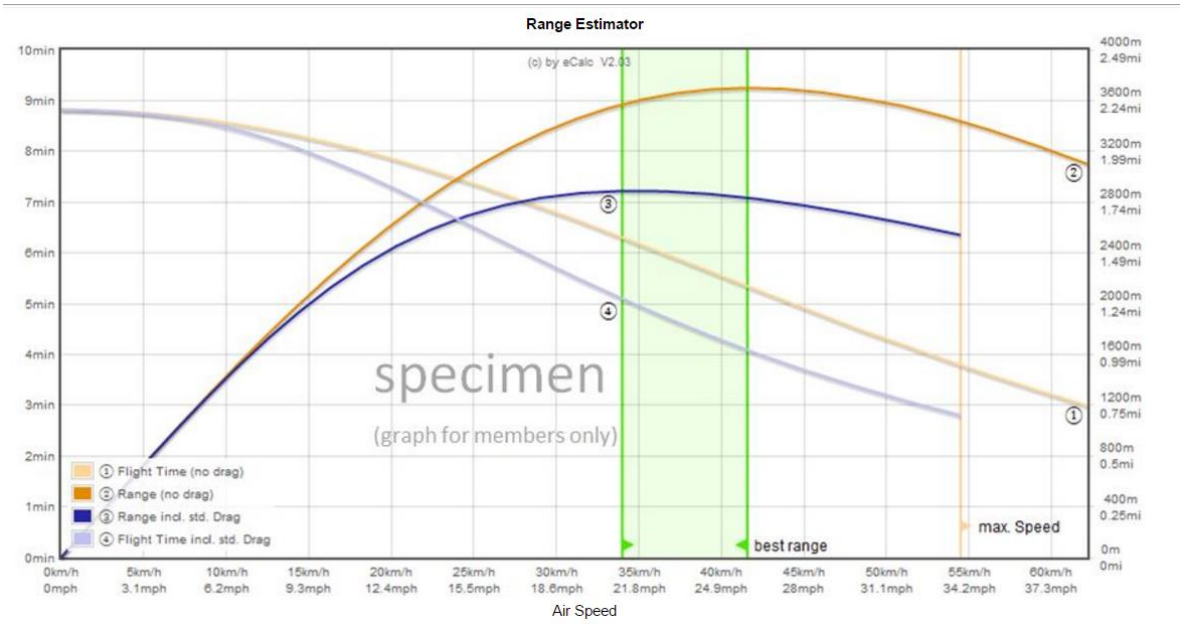
Motor @ Hover		
Current:	2.04	A
Voltage:	10.98	V
Revolutions*:	4047	rpm
Throttle (log):	29	%
Throttle (linear):	45	%
electric Power:	22.4	W
mech. Power:	17.7	W
Power-Weight:	106.7	W/kg
	48.4	W/lb
Efficiency:	79.1	%
est. Temperature:	27	°C
	81	°F
specific Thrust:	9.48	g/W
	0.33	oz/W

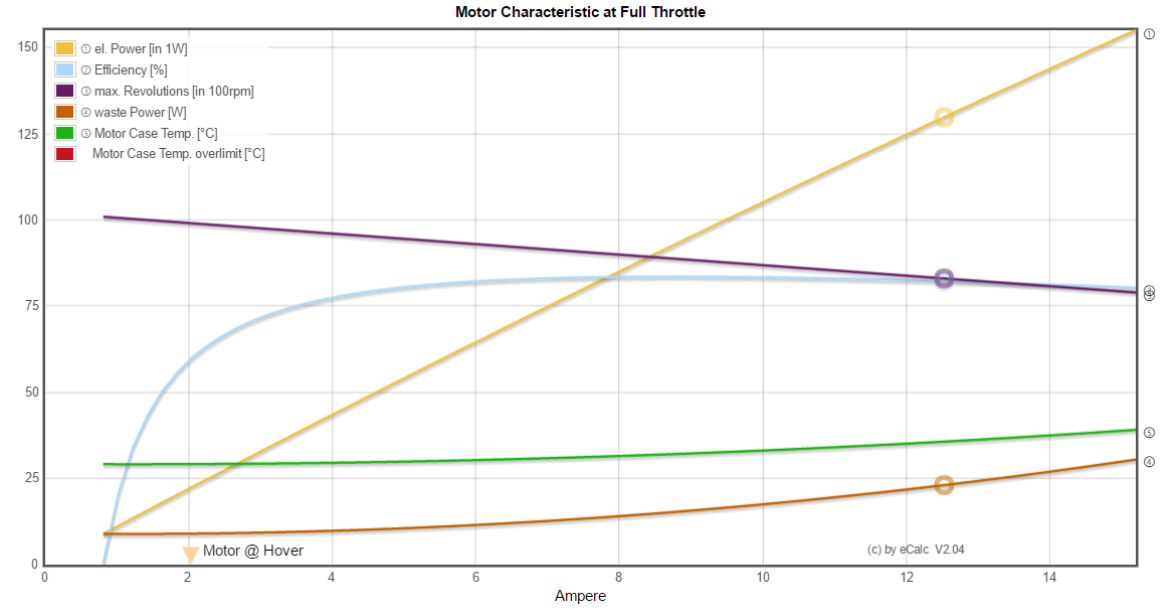
Total Drive		
Drive Weight:	810	g
	28.6	oz
Thrust-Weight:	3.2	: 1
Current @ Hover:	8.17	A
P(in) @ Hover:	90.7	W
P(out) @ Hover:	70.9	W
Efficiency @ Hover:	78.2	%
Current @ max:	50.66	A
P(in) @ max:	562.4	W
P(out) @ max:	430.6	W
Efficiency @ max:	76.6	%

Multicopter		
All-up Weight:	850	g
	30	oz
add. Payload:	1516	g
	53.5	oz
max Tilt:	69	°
max. Speed:	44	km/h
	27.3	mph
est. rate of climb:	9.2	m/s
	1811	ft/min
Total Disc Area:	20.27	dm²
	314.19	in²
with Rotor fail:		

share https:// https://www.eCalc.ch... add to >> Download .csv (0) << clear

Range Estimator


Range Estimator
(c) by eCalc V2.03
10min
9min
8min
7min
6min
5min
4min
3min
2min
1min
0min
4000m 2.49mi
3600m 2.24mi
3200m 1.99mi
2800m 1.74mi
2400m 1.49mi
2000m 1.24mi
1600m 0.99mi
1200m 0.75mi
800m 0.5mi
400m 0.25mi
0m 0mi
specimen
(graph for members only)
① Flight Time (no drag)
② Range (no drag)
③ Range incl. std. Drag
④ Flight Time incl. std. Drag
best range
max. Speed
0km/h 0mph
5km/h 3.1mph
10km/h 6.2mph
15km/h 9.3mph
20km/h 12.4mph
25km/h 15.5mph
30km/h 18.6mph
35km/h 21.8mph
40km/h 24.9mph
45km/h 28mph
50km/h 31.1mph
55km/h 34.2mph
60km/h 37.3mph
Air Speed

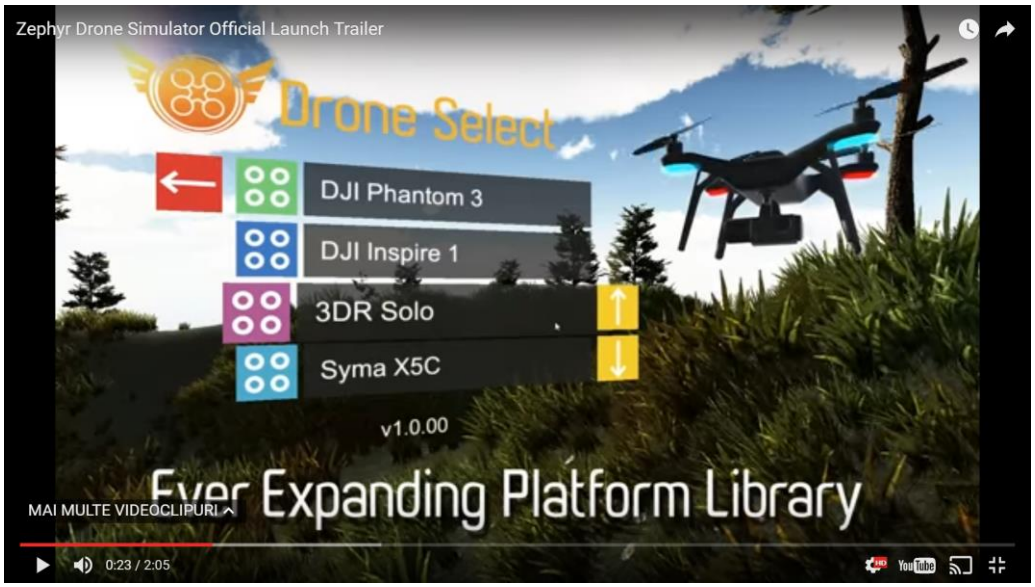

Motor Characteristic at Full Throttle
el. Power [in 1W]
Efficiency [%]
max. Revolutions [in 100rpm]
waste Power [W]
Motor Case Temp. [°C]
Motor Case Temp. overlimit [°C]
150
125
100
75
50
25
0
Motor @ Hover
(c) by eCalc V2.04
0
2
4
6
8
10
12
14
Ampere

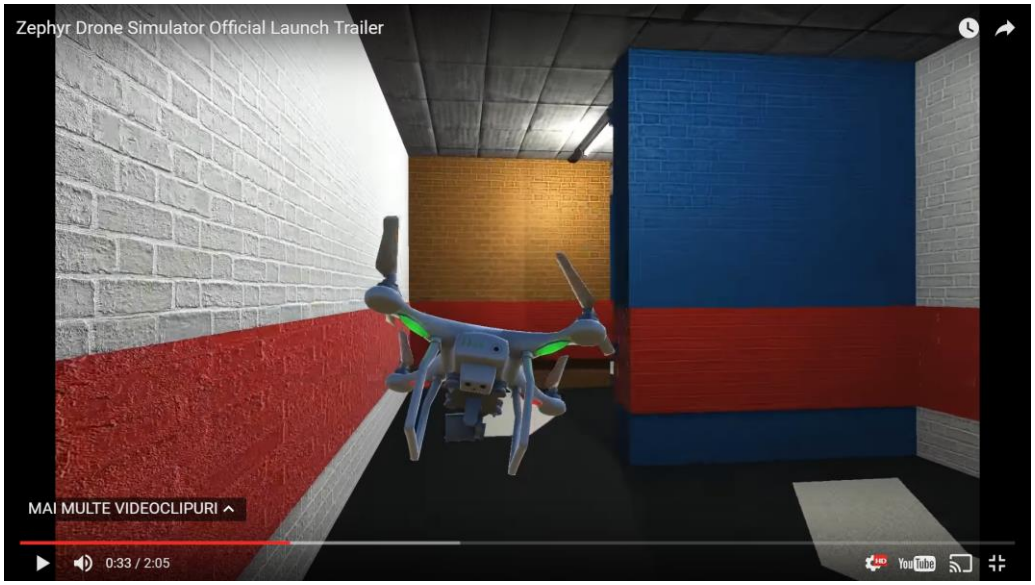
5. SIMULAREA VIRTUALĂ A ZBORULUI

Una dintre firmele specializate pentru simulatoare de zbor este Little Arms Studios[5], care se află în spatele celui mai recent si mai avansat simulator sUAS de pe piață - Zephyr.

Zephyr este un simulator de antrenament piloți virtuali. Rezultatul efortului de dezvoltare a integrat feedback-ul din partea clienților și a profesioniștilor pentru a atinge un nivel de flexibilitate.

5 https://www.suasnews.com/2016/11/little-arms-studios-team-behind-zephyr-drone-simulator-avisight-announce-strategic-partnership/


Zephyr Drone Simulator Official Launch Trailer
Drone Select
DJI Phantom 3
DJI Inspire 1
3DR Solo
Syma X5C
v1.0.00
Ever Expanding Platform Library
MAI MULTE VIDEOCLIPURI
0:23 / 2:05


Zephyr Drone Simulator Official Launch Trailer
MAI MULTE VIDEOCLIPURI
0:33 / 2:05


Zephyr Drone Simulator Official Launch Trailer
Apasă pe Esc pentru a ieși din ecranul complet
MAI MULTE VIDEOCLIPURI
0:42 / 2:05


Zephyr Drone Simulator Official Launch Trailer
MAI MULTE VIDEOCLIPURI
0:49 / 2:05


Zephyr Drone Simulator Official Launch Trailer
MAI MULTE VIDEOCLIPURI
1:03 / 2:05


Zephyr Drone Simulator Official Launch Trailer
MAI MULTE VIDEOCLIPURI
1:35 / 2:05


Zephyr Drone Simulator Official Launch Trailer
Apasă pe Esc pentru a ieși din ecranul complet
MAI MULTE VIDEOCLIPURI
1:39 / 2:05
YouTube

6. CONCLUZII

Această lucrare a avut drept scop prezentarea caracteristicilor unui zbor autonom/teleoperat al unui quadrocopter.

Quadrocopterul realizat, ca proiect de cercetare internă, în cadrul laboratorului Centrului de Cercetare al Facultății de Informatică, Universitatea Titu Maiorescu.

Modelul analitic a fost scris în limbaj C++ și introdus în controlerul Arduino. Toate elementele care țin de cinematica și dinamica zborului au fost salvate în varianta basic. Nu s-au ținut cont de parametri variabili. Abordarea din punct de vedere al sistemelor automate s-a centrat pe regula fuzzy modulară pentru controlul robust al motoarelor, acest aspect a fost generat de faptul că s-a renunțat la un control foarte precis al elementelor de comandă și control.

Controlerul are sarcina de a gestiona turația motoarelor în funcție de informațiile transmise de către GPS, senzorul de accelerație, telemetrul radio.

Simularea în SolidWorks a permis stabilirea poziției centrului de greutate, astfel încât, echilibrul față de sistemul de referință să poată fi asigurat doar prin modificarea alimentării motoarelor.

Una dintre concluziile importante constă în faptul că senzorii introduc zgomote, care pot induce în eroare comanda robotului.

Acest tip de QuadCopter poate fi utilizat în multe domenii:

- observare aeriană a unor obiective industriale;
- filmări aeriene, putând înlocui cu succes operatorii umani.