Radionica izrade i programiranja bespilotnih letjelica

dr. sc. Jurica Kundrata
Centar za svemirsku i inovativnu tehnologiju
26. i 27. travnja 2025.

Raspored

26. travnja 2025., subota	
9.00 - 12.00	Uvod u bespilotne letjelice; Sastavljanje i programiranje pomoćnih funkcionalnosti
12.00 - 12.45	Pauza za ručak
12.45 - 15.00	Programiranje inercijalne mjerne jedinice i motora
27. travnja 2025., nedjelja	
9.00 -12.00	Estimacija orijentacije i detekcija sudara
12.00 - 12.45	Pauza za ručak
12.45 - 15.00	Programiranje kontrolera leta (PID regulator i logika leta)

CSIT

perAsperaAdAstra

Upoznavanje

Dan I

Uvod u bespilotne letjelice

Uvod u bespilotne letjelice

- Bespilotne letjelice (dronovi) su **zračni sustavi bez posade** koji se upravljaju daljinski ili autonomno, koristeći napredne algoritme. Primjenjuju se u raznim industrijama, uključujući poljoprivredu, dostavu, snimanje, znanstvena istraživanja i vojsku.

- Ključne komponente uključuju **pogonski sustav** (motori i propeleri za uzgon), **senzore** (IMU za stabilizaciju, GPS za navigaciju), **letno računalo** (flight controller) za obradu podataka i upravljanje te bateriju i regulator za napajanje svih sustava.

Primjeri bespilotnih letjelica

Primjeri bespilotnih letjelica

Primjeri bespilotnih letjelica

- **Tupoljev Tu-141 “Strič”** je sovjetska bespilotna izviđačka letjelica iz razdoblja Hladnog rata, razvijena za prikupljanje obavještajnih podataka na velikim udaljenostima.

- Pogonjen **turbomlaznim motorom KR-17A**, Tu-141 postiže brzine **do 1,000 km/h** i domet preko 1,000 km, što ga čini brzim i dugodometnim izviđačkim sustavom.

- Koristi unaprijed programiranu putanju leta uz pomoć inercijalne navigacije i zemaljsku kontrolu za povratak, a dizajniran je za višekratnu upotrebu zahvaljujući sustavu za padobransko slijetanje.

Primjeri bespilotnih letjelica

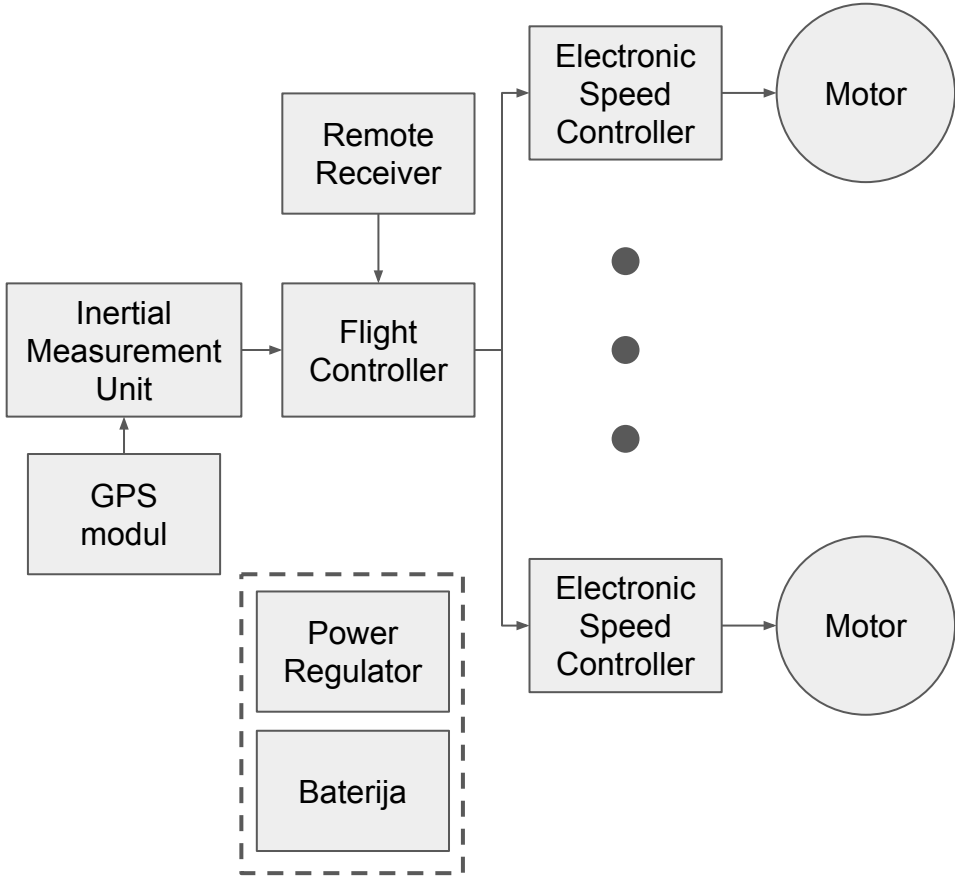
Primjeri bespilotnih letjelica

- Amazon Prime Air MK-30 dron koristi **VTOL (Vertical Take-Off and Landing)** sposobnosti za okomito polijetanje i slijetanje te prelazak na učinkoviti horizontalni let, omogućujući dostavu u različitim okruženjima.

- Može nositi **pakete težine do 2,3 kg**, leti dvostruko dalje od prethodnih modela, i operira u uvjetima lagane kiše, proširujući područje dostave.

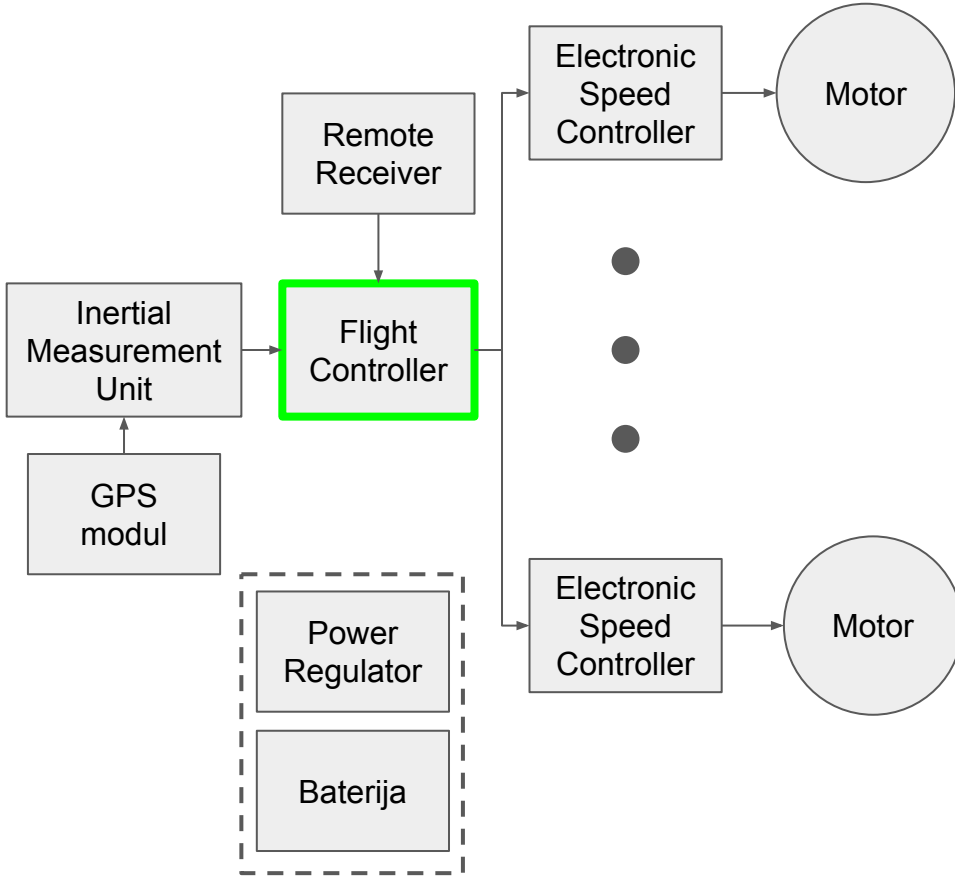
Opća građa bespilotne letjelice

- Letjelicom upravlja **Flight Controller** koji prikuplja podatke iz senzora (**IMU i GPS**) i prima komande od **daljinskog upravljača**.
- **Elektronički kontroleri brzine** (ESC) reguliraju rad motora na temelju signala iz kontrolera leta.
- **Baterija i regulator napona** osiguravaju stabilno napajanje svih komponenti letjelice.



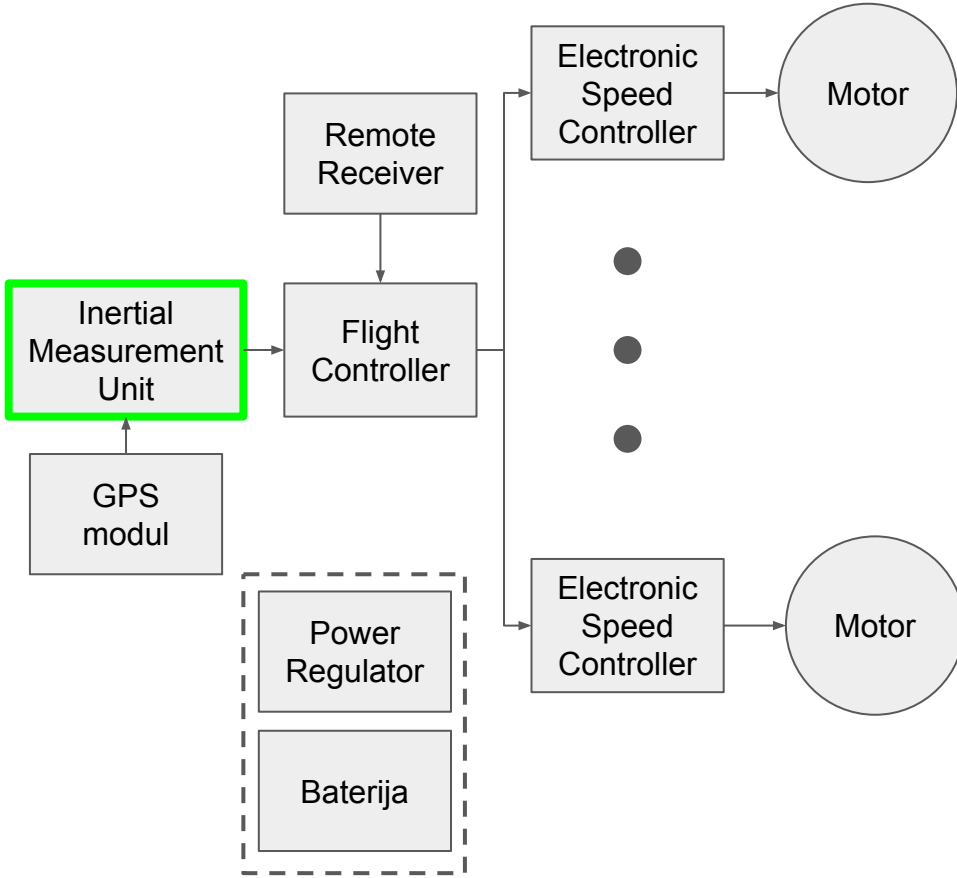
Opća građa bespilotne letjelice

- Flight controller **prikuplja podatke** iz senzora (IMU, GPS) i daljinskog upravljača te **izračunava komande** za motore.
- Osigurava **stabilan let** kroz kontrolu orijentacije (roll, pitch, yaw) i pozicije letjelice.
- Omogućuje prilagodbu **algoritama leta** i integraciju **dodatnih senzora** za specifične zadatke.



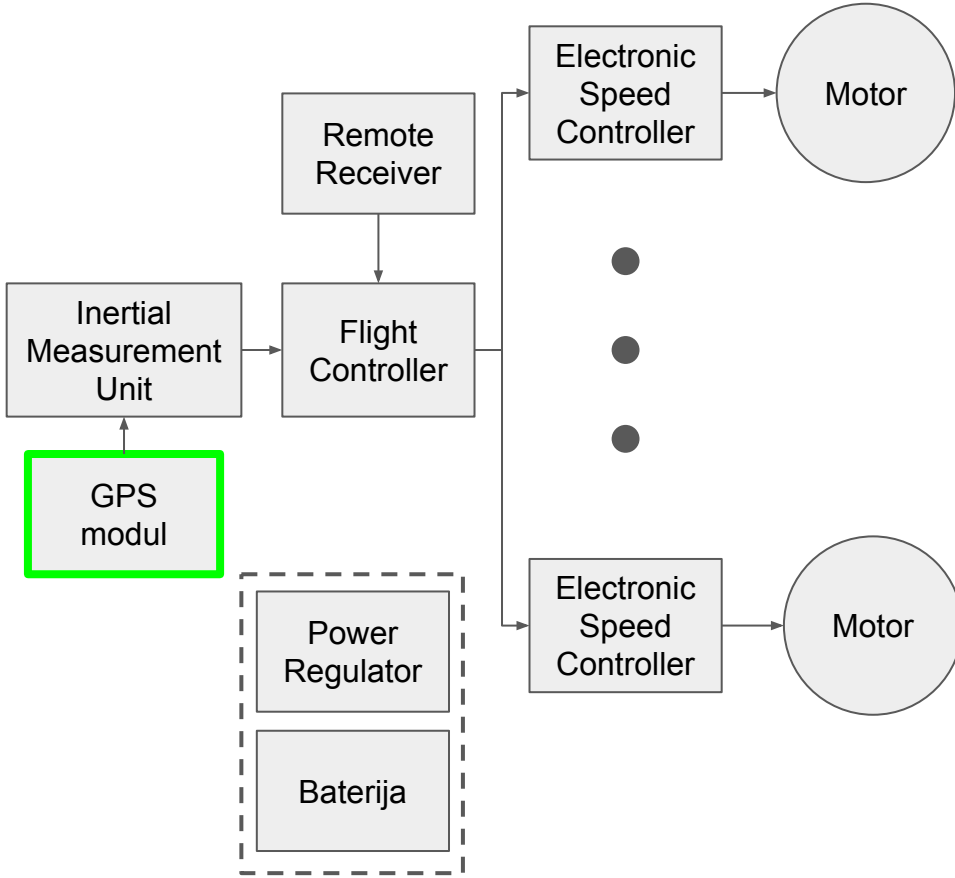
Opća građa bespilotne letjelice

- IMU (Inertial Measurement Unit) **mjeri ubrzanja i kutne brzine** kako bi procijenila orijentaciju letjelice (roll, pitch, yaw).
- Mjeri podatke za **stabilizaciju i kontrolu leta** kroz akcelerometar i žiroskop.
- Brzo reagira na promjene kretanja što je potrebno za **precizno upravljanje letjelicom**.



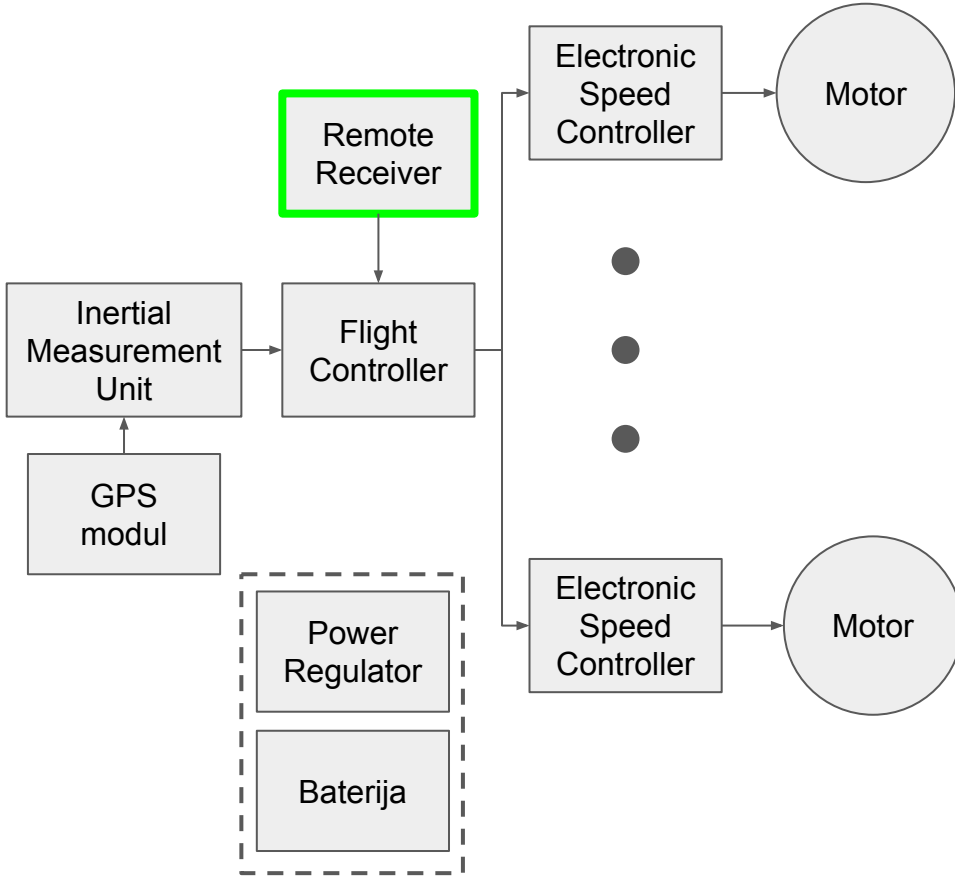
Opća građa bespilotne letjelice

- GPS modul određuje **točnu poziciju** letjelice u globalnom koordinatnom sustavu.
- Ključan je za funkcije poput **povratka na početnu točku** (Return to Home) i **programirane rute**.
- Osim pozicije, pruža podatke o **brzini kretanja** i relativnoj visini u odnosu na GPS referencu.



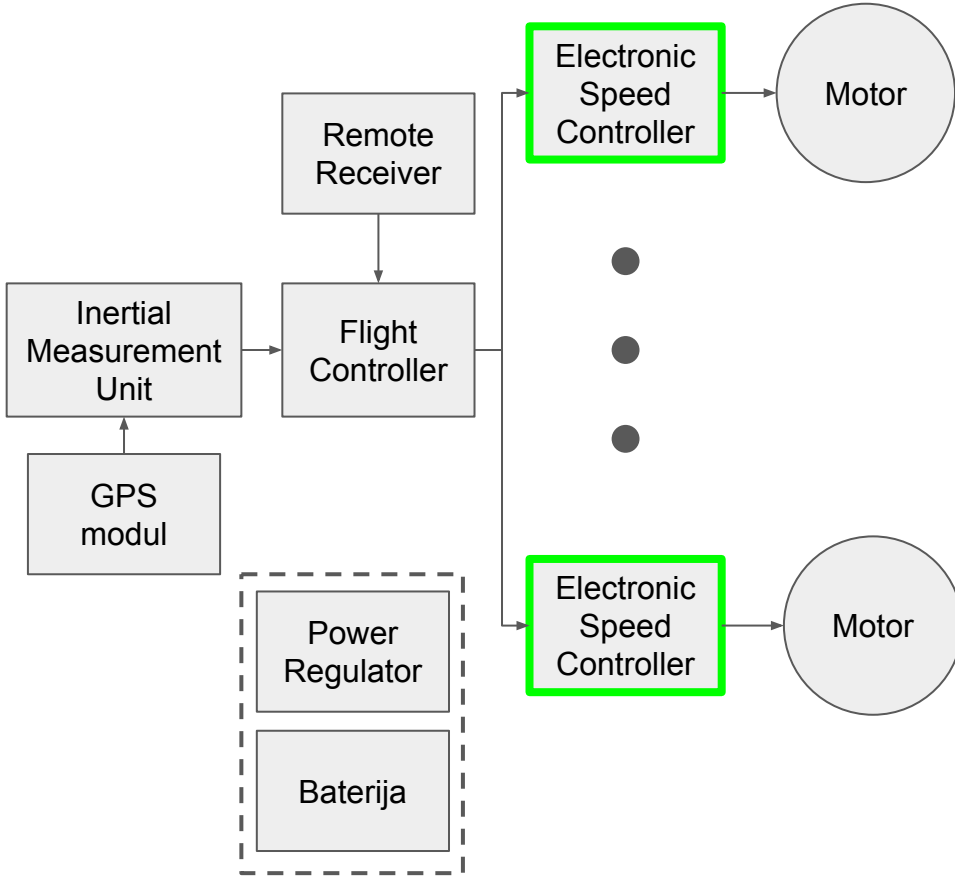
Opća građa bespilotne letjelice

- Remote receiver **prima korisničke komande** s daljinskog upravljača, poput promjene smjera, visine i brzine.
- Koristi **radio frekvencije** za pouzdanu vezu između pilota i letjelice.
- Prosljeđuje primljene signale flight controlleru za **izvršavanje upravljačkih naredbi**.



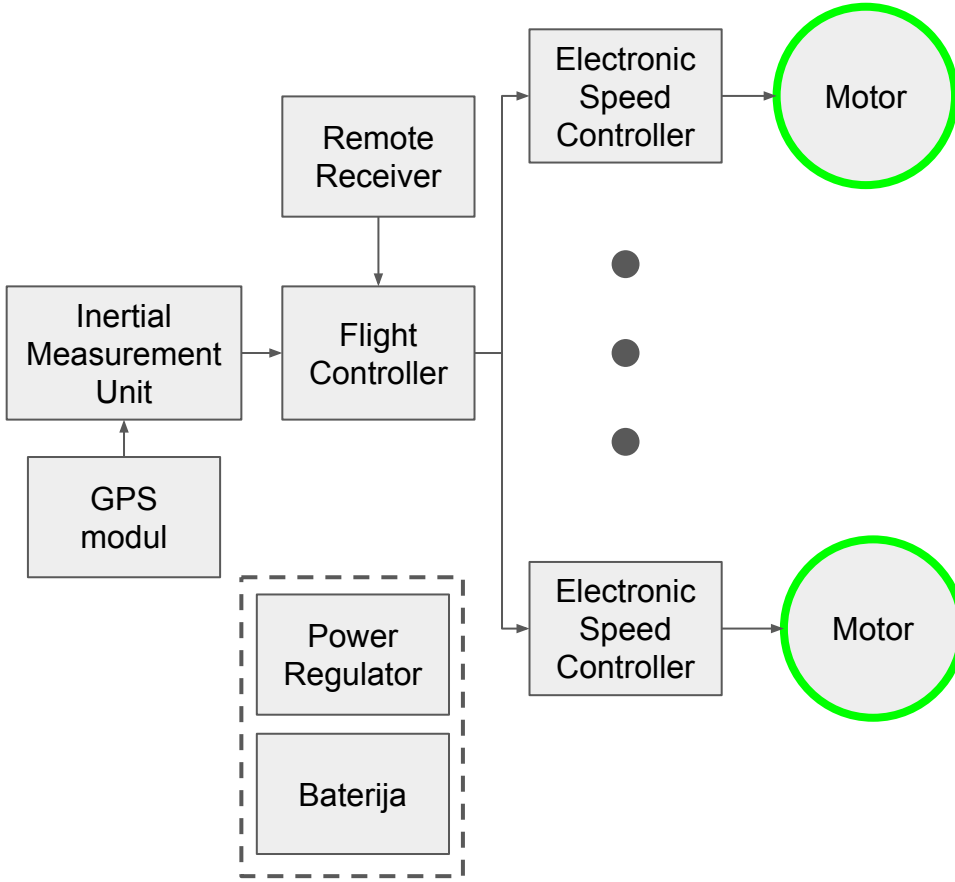
Opća građa bespilotne letjelice

- Elektronički kontroleri brzine (ESC) **upravljaju radom motora** reguliranjem snage na temelju signala iz kontrolera leta.
- Pretvaraju **PWM signale** iz flight controllera u preciznu regulaciju brzine i smjera vrtnje motora.
- Ključni su za **balansiranje snaga između motora**, osiguravajući stabilan let i reakciju na promjene u letu.



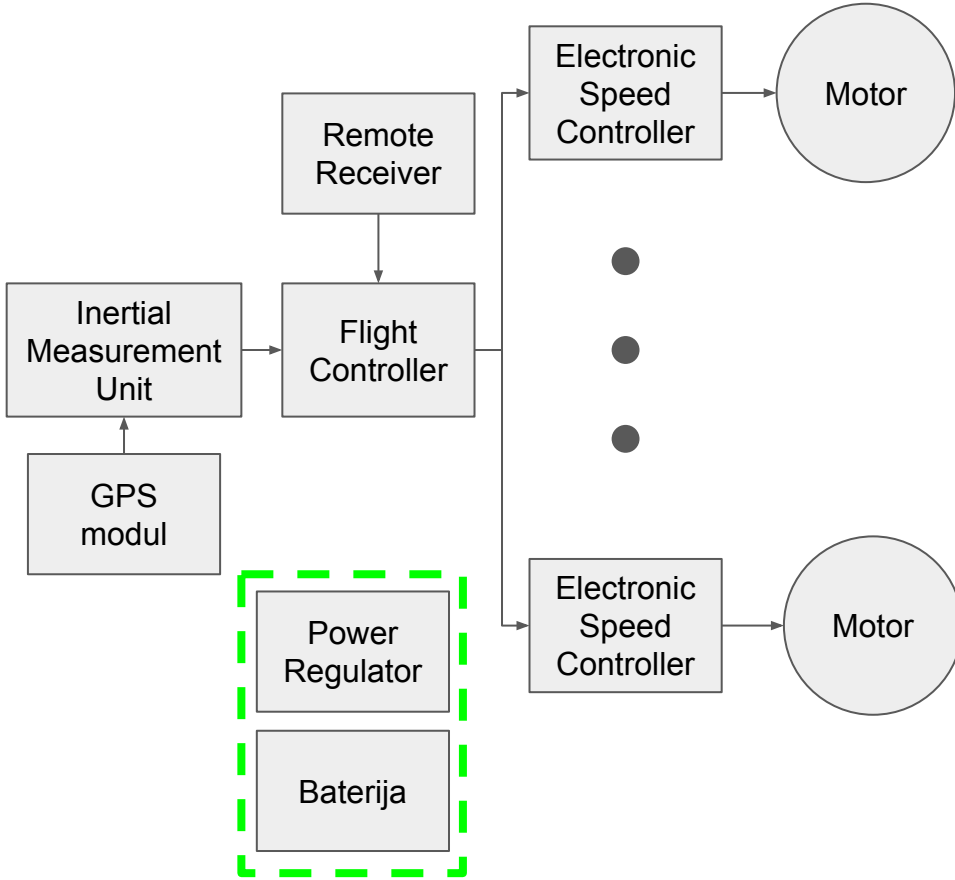
Opća građa bespilotne letjelice

- Motori **generiraju uzgon** potreban za let i omogućuju kontrolu kretanja letjelice.
- Postavljeni su u **suprotnim smjerovima (CW i CCW)** kako bi se neutralizirali zakretni momenti.
- Brzina i smjer vrtnje motora određuju **stabilnost, visinu i orijentaciju** letjelice.



Opća građa bespilotne letjelice

- Baterija **napaja el. energijom** sve komponente letjelice, uključujući motore, kontroler leta i senzore.
- Power regulator **prilagođava napon** iz baterije kako bi odgovarao zahtjevima različitih elektroničkih modula.
- Kapacitet baterije određuje **maksimalno vrijeme leta** i zahtijeva balans između težine i energetskog kapaciteta.



eduCopter


SCL
SDA
VCC
GND
INT2
INT1
easyc


IRF540N
77
WS

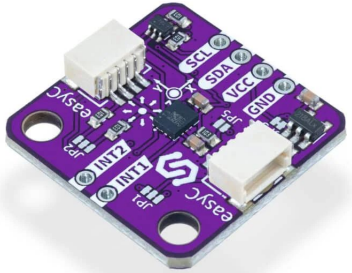

LED
USB
BOOTSEL
Raspberry Pi Pico
© 2020
DEBUG

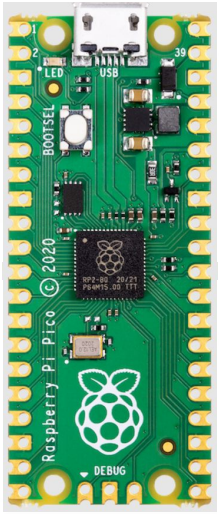

IRF540N
77
WS


IRF540N
77
WS

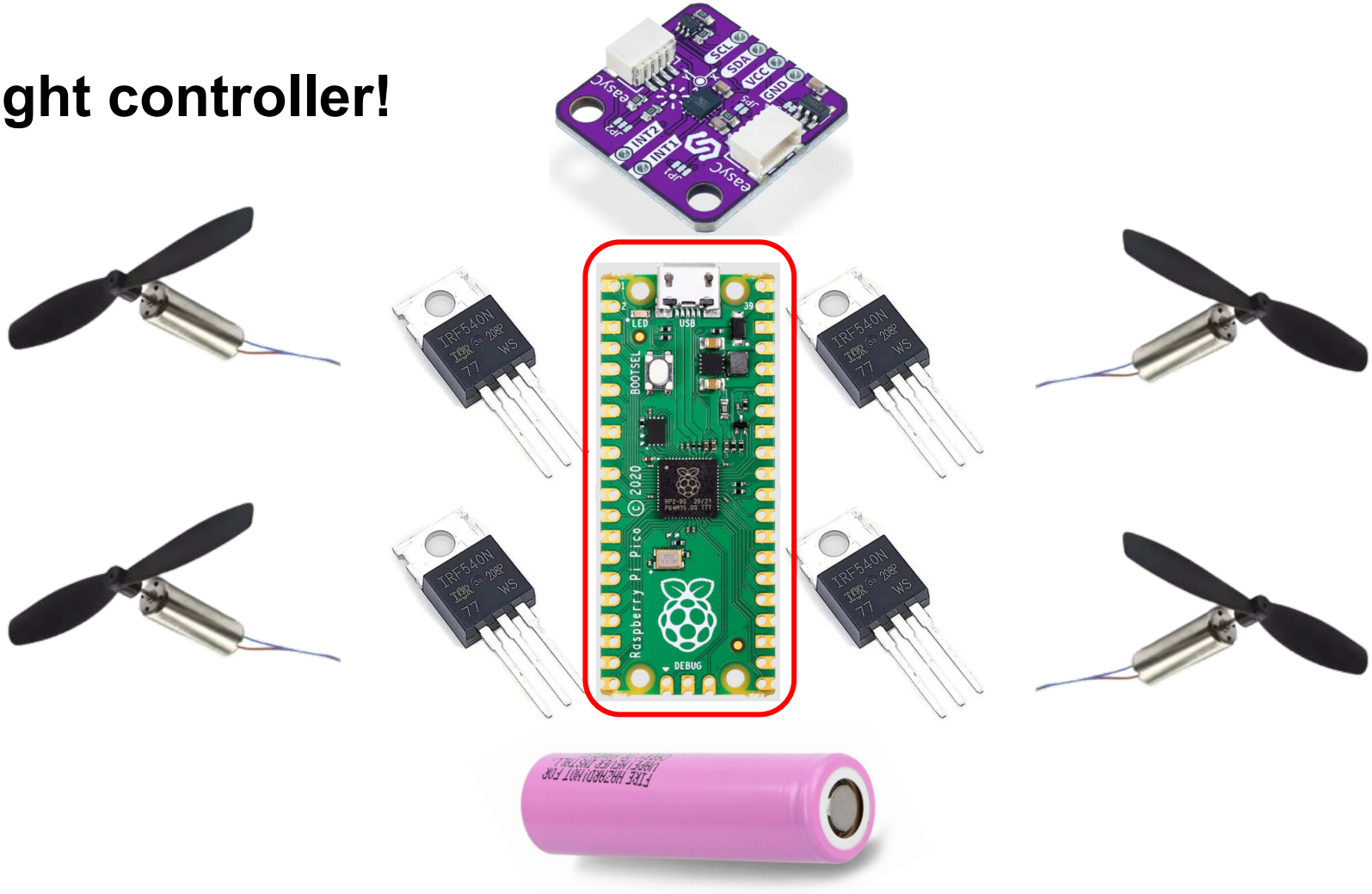

IRF540N
77
WS

Flight controller?

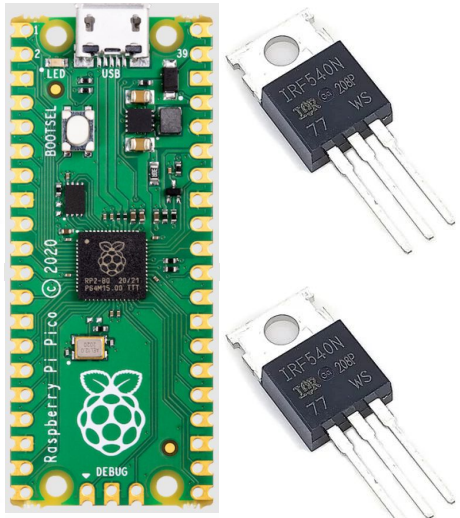




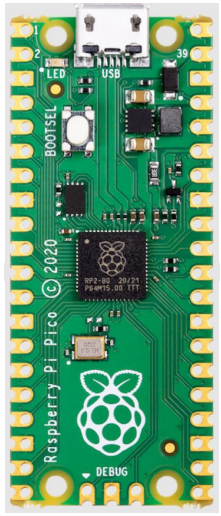
Flight controller!



Inertial Meas. Unit?

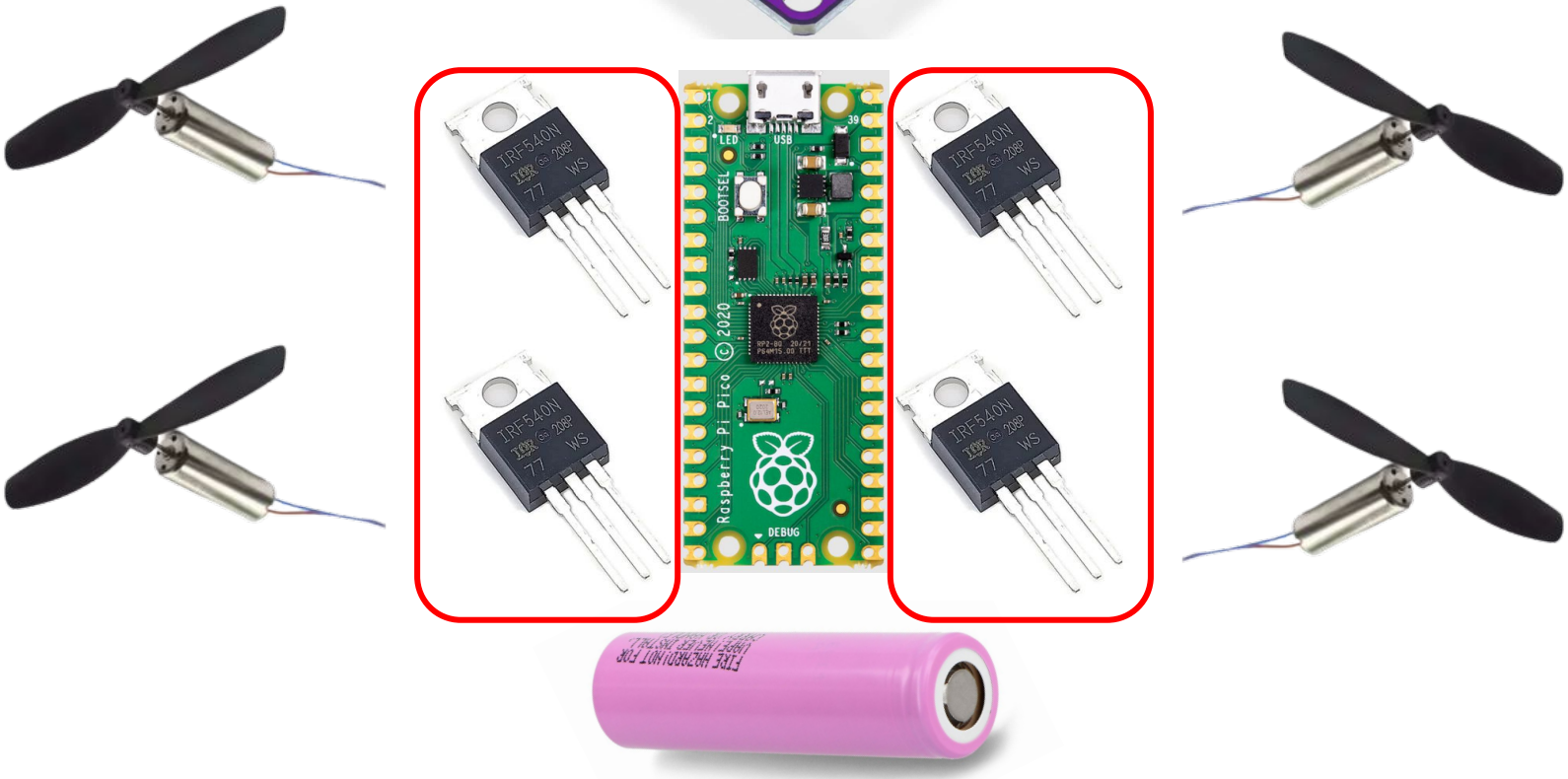


Inertial Meas. Unit!



El. Speed Controller?

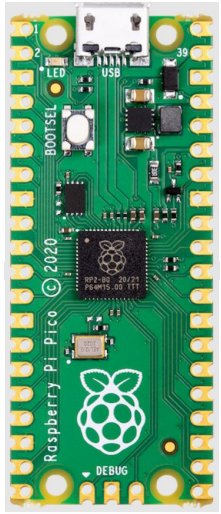
El. Speed Controller!



Motor?


SCL
SDA
VCC
GND
INT2
INT1
easyc


IRF540N
77
WS


LED
USB
BOOTSEL
Raspberry Pi Pico
© 2020
DEBUG

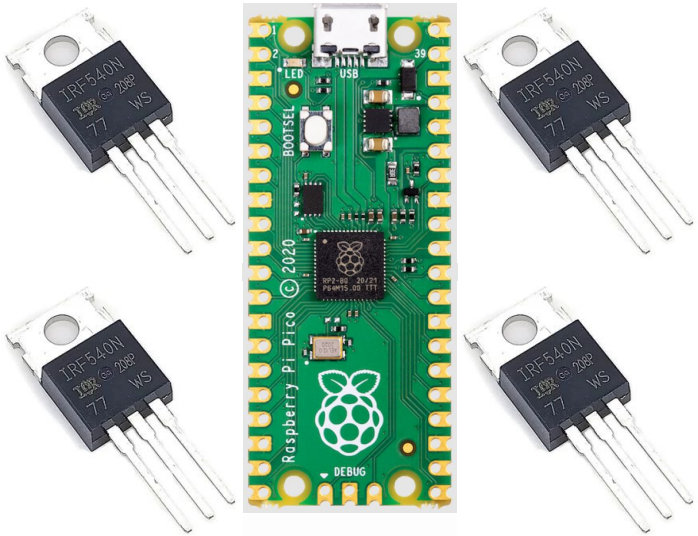

IRF540N
77
WS


IRF540N
77
WS


IRF540N
77
WS

Motor!


SCL
SDA
VCC
GND
INT2
INT1


IRF540N
77
WS
LED
USB
BOOTSEL
Raspberry Pi Pico
© 2020
DEBUG

Baterija?


SCL
SDA
VCC
GND
INT2
INT1
easyC


IRF540N
77
WS


LED
USB
BOOTSEL
Raspberry Pi Pico
© 2020
DEBUG

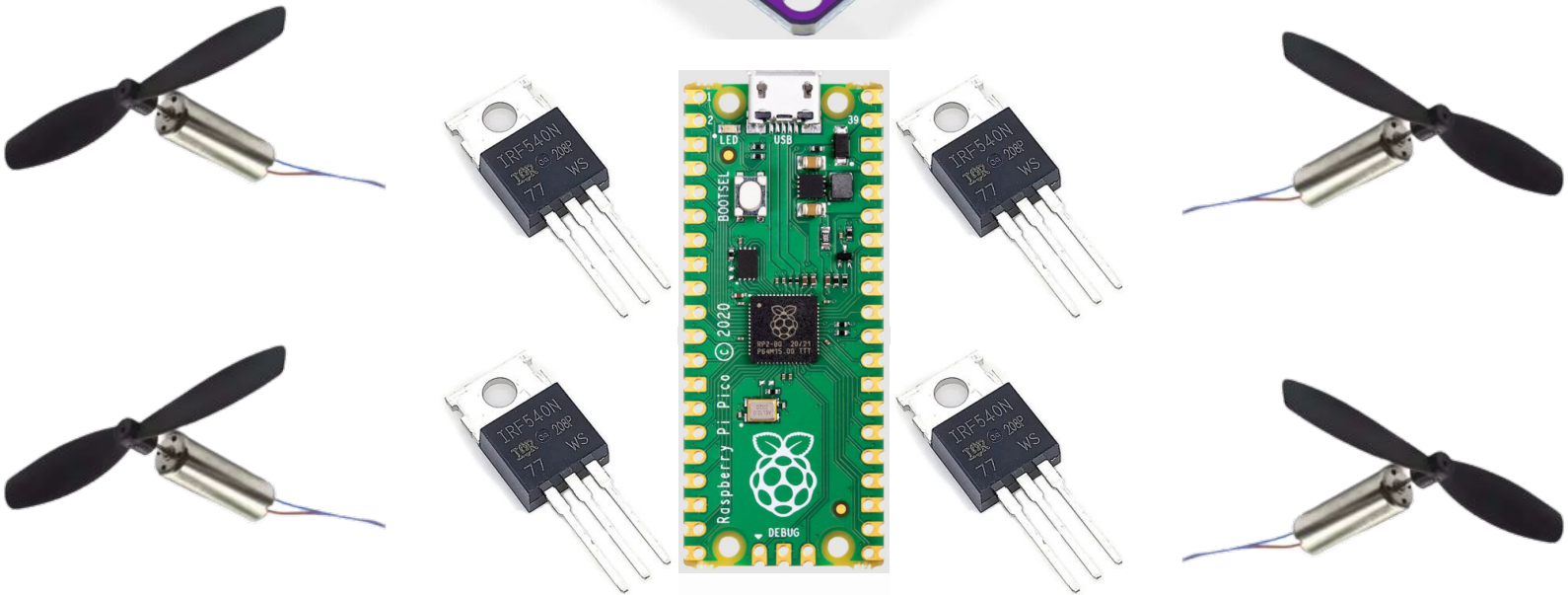

IRF540N
77
WS


IRF540N
77
WS

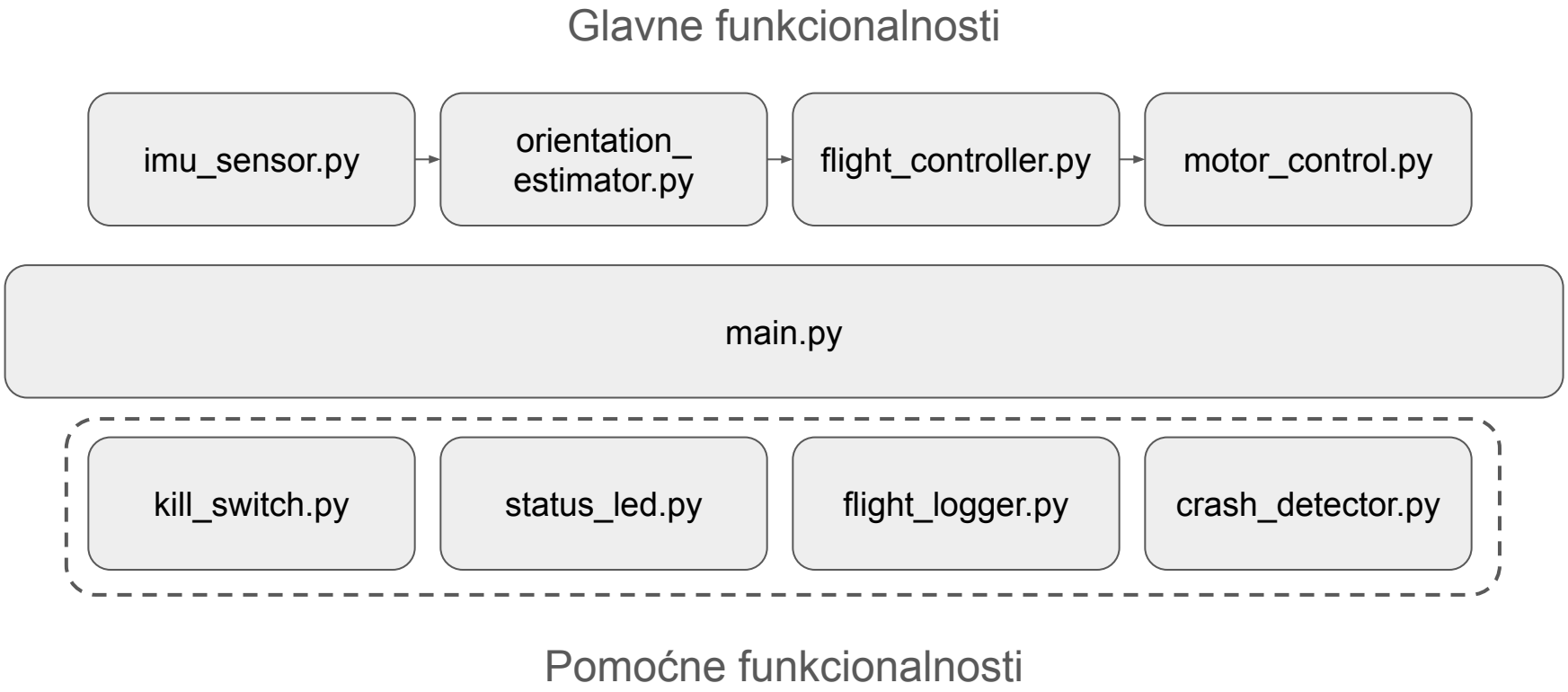

IRF540N
77
WS

Baterija!


SCL
SDA
VCC
GND
INT2
INT1
easyc


IRF540N
77
WS
BOOTSEL
LED
USB
DEBUG
Raspberry Pi Pico
© 2020

Organizacija programske podrške *(firmware)*



Dan I

Sastavljanje i programiranje pomoćnih funkcionalnosti

Sadržaj kita bespilotne letjelice

1. Šasija s Raspberry Pi Pico razvojnom pločicom i pogonskim sklopovima IRLZ44N i motorima 816 4 kom.

2. Propeler 75mm 4kom. (2 para po 2 propelera suprotnih orijentacija)

3. Modul s IMU senzorom LSM6DS3

4. Kratkospojnik

5. Shema izvoda Raspberry Pi Pico pločice

Sastavljanje i programiranje pomoćnih funkcionalnosti

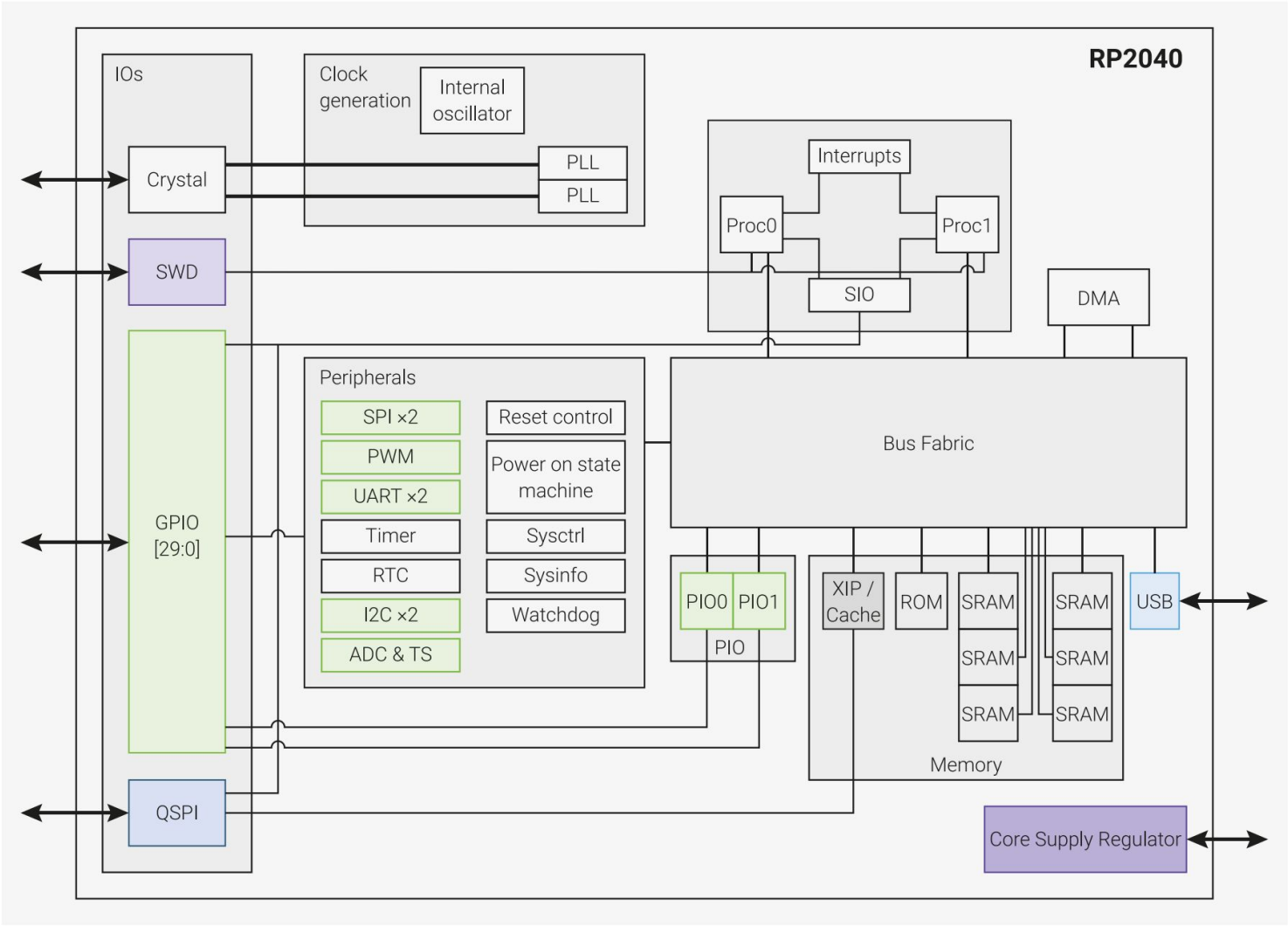
- Postavit ćemo razvojno okruženje za letno računalo
- Isprobat ćemo sigurnosni prekidač
- Iterativno i modularno ćemo graditi programsku podršku počevši od pomoćnih funkcionalnosti:
  - Statusne LEDice
  - Sigurnosnog prekidača (Kill Switch)
  - Zapisa dijagnostičkih podataka

Letno računalo - Raspberry Pi Pico

- Raspberry Pi Pico služi kao **kompaktno i energetski učinkovito** letno računalo za upravljanje senzorima i izvršavanje algoritama kontrole leta.

- Raspberry Pi Pico ima dvije **ARM Cortex-M0+** jezgre, **264 KB SRAM-a**, **2 MB Flash** memorije i podržava do **16 PWM** pinova. Podržava MicroPython i C/C++.

- Podržava protokole poput **I2C, SPI i UART** što omogućuje integraciju sa senzorima, motorima i drugim perifernim uređajima. Integrirana LEDica daje vizualni indikator statusa rada.


RP2040
IOs
Crystal
SWD
GPIO
[29:0]
QSPI
Clock
generation
Internal
oscillator
PLL
PLL
Interrupts
Proc0
Proc1
SIO
DMA
Peripherals
SPI ×2
PWM
UART ×2
Timer
RTC
I2C ×2
ADC & TS
Reset control
Power on state
machine
Sysctrl
Sysinfo
Watchdog
Bus Fabric
PIO0
PIO1
PIO
XIP /
Cache
ROM
SRAM
SRAM
SRAM
SRAM
SRAM
SRAM
Memory
USB
Core Supply Regulator

Razvojno okruženje

https://thonny.org/

- Alat za uređivanje, debugiranje i izvođenje Python i MicroPython koda
- Jednostavno povezivanje i rad s MicroPython uređajima
- Idealan za početnike u programiranju

Početna programska podrška

- Raspoloživa je na:

tinyurl.com/jfddmmet

Statusna LEDica - modul **status_led.py**

- Modul omogućuje vizualnu **indikaciju rada drona** (uključen, isključen, ili signalizacija).

- **Kontinuirano treptanje LED-a** pomoću Timer modula pruža informacije poput statusa povezivanja ili grešaka.

- LED se može ručno uključiti ili isključiti za prilagođene indikacije tijekom rada drona.

Statusna LEDica - modul **status_led.py**

- Pregled koda modula

```
1  from machine import Pin, Timer
2
3  class StatusLED:
4      def __init__(self, pin_num=  ):
5          self.led = Pin(pin_num, Pin.OUT)
6          self.timer = Timer()
7
8      def start_blinking(self, interval):
9          self.timer.init(freq=1/interval, mode=Timer.PERIODIC, callback=lambda t: self.led.toggle())
10
11     def stop_blinking(self):
12         self.timer.deinit()
13         self.led.off()
14
15     def turn_on(self):
16         self.led.on()
17
18     def turn_off(self):
19         self.led.off()
```

Statusna LEDica - modul **status_led.py**

- Pregled koda primjera uporabe modula

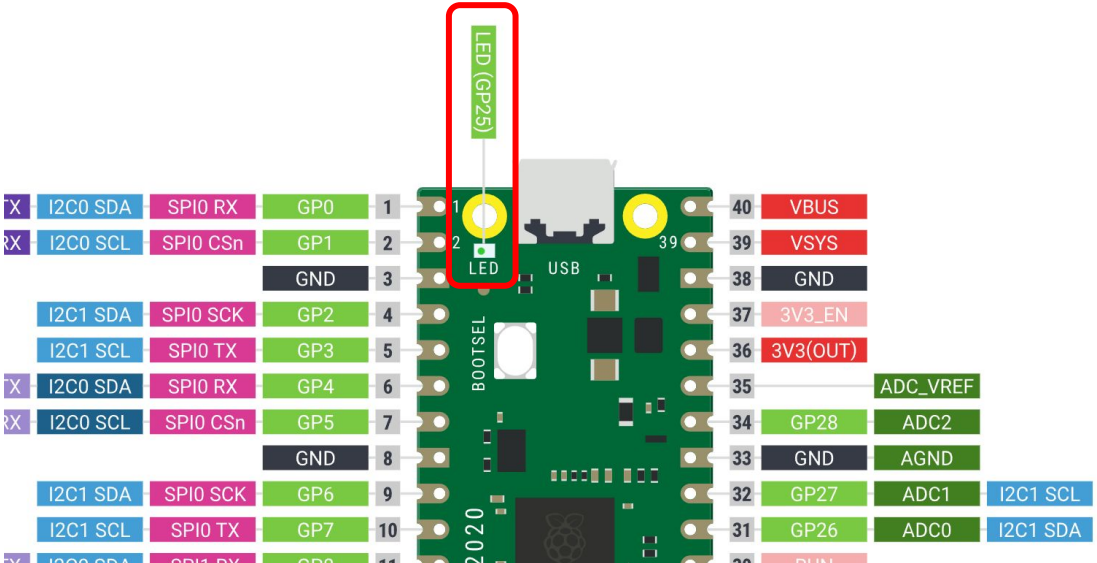
```
21 if __name__ == "__main__":
22     led = StatusLED()
23     led.start_blinking(0.5)
24     import time
25     time.sleep(3)
26     led.stop_blinking()
27     led.turn_on()
28     time.sleep(2)
29     led.turn_off()
```

Statusna LEDica - modul **status_led.py**

- Na kojem je GPIO pinu spojena LEDica?

Statusna LEDica - modul **status_led.py**

- Na kojem je GPIO pinu spojena LEDica? **25**



Statusna LEDica - modul **status_led.py**

Zadatak:

- Promijenite frekvenciju treptanja LEDice na 4 Hz.
- Promijenite kod tako da LEDica na kraju ostane upaljena.

Kill Switch (Sigurnosni prekidač) - modul **kill_switch.py**

- Prekidač **sprječava aktivaciju drona** dok je kratkospojnik prisutan.
- **Uklanjanjem kratkospojnika** dron se aktivira i započinje letnu sekvencu.
- Funkcija is_activated() koristi se za detekciju stanja prekidača i kontrolu nad pokretanjem drona.
- Koristi **Pin.PULL_UP** kako bi osigurao stabilno stanje prekidača dok nije aktiviran.

Kill Switch (Sigurnosni prekidač) - modul **kill_switch.py**

- Izgled kratkospojnika za sigurnosni prekidač

Kill Switch (Sigurnosni prekidač) - modul **kill_switch.py**

- Aktivacija sigurnosnog prekidača

Kill Switch (Sigurnosni prekidač) - modul **kill_switch.py**

- Pregled koda modula

```
1  from machine import Pin
2  import time
3
4  class KillSwitch:
5      def __init__(self, pin_num=□):
6          self.switch = Pin(pin_num, Pin.IN, Pin.PULL_UP)
7
8      def is_activated(self):
9          """Check if the kill switch is activated (pulled to GND)."""
10         return self.switch.value() == 0
```

Kill Switch (Sigurnosni prekidač) - modul **kill_switch.py**

- Pregled koda primjera uporabe modula

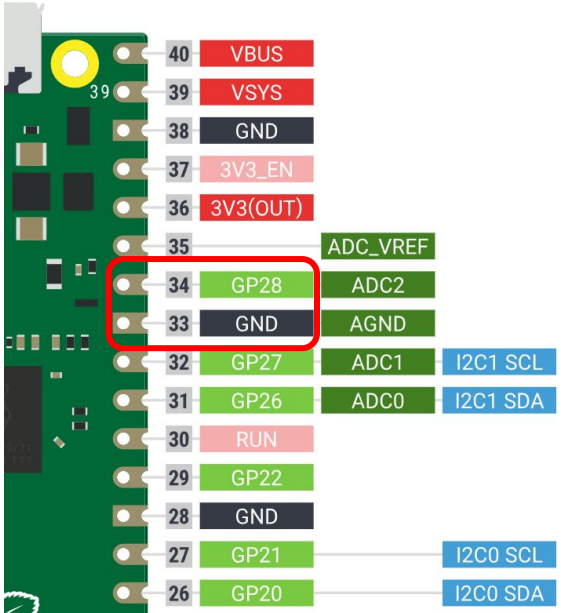
```
14  from status_led import StatusLED
15
16  # Initialize Kill Switch and Status LED
17  kill_switch = KillSwitch()
18  led = StatusLED()
19
20  print("Deactivate the kill switch to test.")
21  led.start_blinking(0.5)  # Blink LED to indicate waiting for input
22
23  try:
24      while True:
25          if not kill_switch.is_activated():
26              print("Kill switch deactivated!")
27              led.stop_blinking()
28              led.turn_on()  # Solid LED indicates activation
29              break
30          time.sleep(0.1)
31      while True:
32          time.sleep(1)
33  except KeyboardInterrupt:
34      print("Exiting test.")
35  finally:
36      led.stop_blinking()
37      led.turn_off()
```

Kill Switch (Sigurnosni prekidač) - modul **kill_switch.py**

- Između kojih pinova je spojen sigurnosni prekidač?
- U kojem stanju ulaznog pina je sigurnosni prekidač aktivan?

Kill Switch (Sigurnosni prekidač) - modul **kill_switch.py**

- Između kojih pinova je spojen sigurnosni prekidač? **28**
- U kojem stanju ulaznog pina je sigurnosni prekidač aktivan? **NISKOM**



Kill Switch (Sigurnosni prekidač) - modul **kill_switch.py**

- Za izlazak iz testa pritisnite Ctrl+C

Zadatak:

- Promijenite frekvenciju treptanja LEDice nakon deaktivacije sigurnosnog prekidača na 2 Hz.

Zapis dijagnostičkih podataka - modul **flight_logger.py**

- Modul bilježi **vremenski označene događaje** u letu drona **za kasniju analizu** ponašanja sustava.

- Koristi funkcije time.ticks_ms() i time.ticks_diff() za **precizno praćenje vremena** od početka zapisivanja.

- Zapisuje događaje u **tekstualnu datoteku** i osigurava trajnost podataka putem funkcije flush().

Zapis dijagnostičkih podataka - modul **flight_logger.py**

- Pregled koda modula

```
3  class FlightLogger:
4      def __init__(self, file_name="flight_log.txt"):
5          self.file_name = file_name
6          self.file = None
7          self.start_time = None
8
9      def start(self):
10         """Start the logger and initialize the log file."""
11         self.file = open(self.file_name, 'w')
12         self.start_time = time.ticks_ms()  # Record start time
13         self.file.write("Time_ms,Event\n")
14
15     def log(self, event):
16         """Log an event with a timestamp."""
17         if self.file:
18             elapsed_time = time.ticks_diff(time.ticks_ms(), self.start_time)
19             self.file.write(f"{elapsed_time},{event}\n")
20
21     def flush(self):
22         """ Flush the log content to flash memory. """
23         if self.file:
24             self.file.flush()
25
26
27     def stop(self):
28         """Stop the logger and close the file."""
29         if self.file:
30             self.file.close()
```

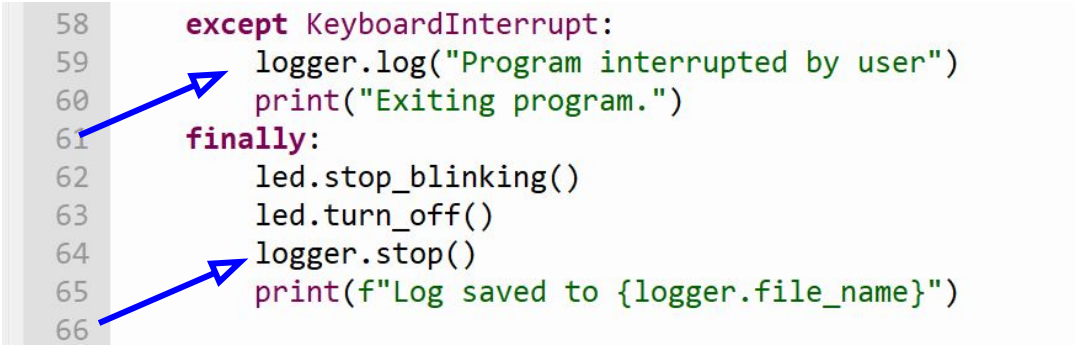
Zapis dijagnostičkih podataka - modul **flight_logger.py**

- Pregled koda primjera uporabe modula

```
33  from status_led import StatusLED
34  from kill_switch import KillSwitch
35  # Initialize components
36  led = StatusLED()
37  kill_switch = KillSwitch()
38  logger = FlightLogger()
39
40  print("System starting...")
41  logger.start()
42  logger.log("System initialized, waiting for kill switch deactivation")
43
44  led.start_blinking(0.5)  # Blink LED to indicate waiting for kill switch deactivation
45
46  print("Kill switch active. Deactivate it to proceed.")
47  try:
48      while kill_switch.is_activated():
49          time.sleep(0.1)  # Wait for the kill switch to be deactivated
50
51      logger.log("Kill switch deactivated")
52      print("Kill switch deactivated!")
53      led.stop_blinking()
54      led.turn_on()  # Solid LED indicates readiness
55      time.sleep(1)  # Hold the LED for visibility
56      logger.log("System ready to proceed")
57
```

Zapis dijagnostičkih podataka - modul **flight_logger.py**

- Pregled koda primjera uporabe modula



Zapis dijagnostičkih podataka - modul **flight_logger.py**

- Kako dodajemo nove module, proširivat ćemo stavke dijagnostičkog zapisa.

- One će uključivati:
  - Izmjerene vrijednosti IMU senzora
  - Procijenjenu orijentaciju u prostoru
  - Izračunate korekcije regulatora
  - Brzine motora

Dan I

Programiranje inercijalne mjerne jedinice i motora

Programiranje inercijalne mjerne jedinice i motora

- Montirat ćemo IMU modul i motore s propelerima na unibody šasiju letjelice
- Nastavit ćemo gradnju programske potpore dodavanjem modula za:
  - IMU senzor
  - Upravljanje motora

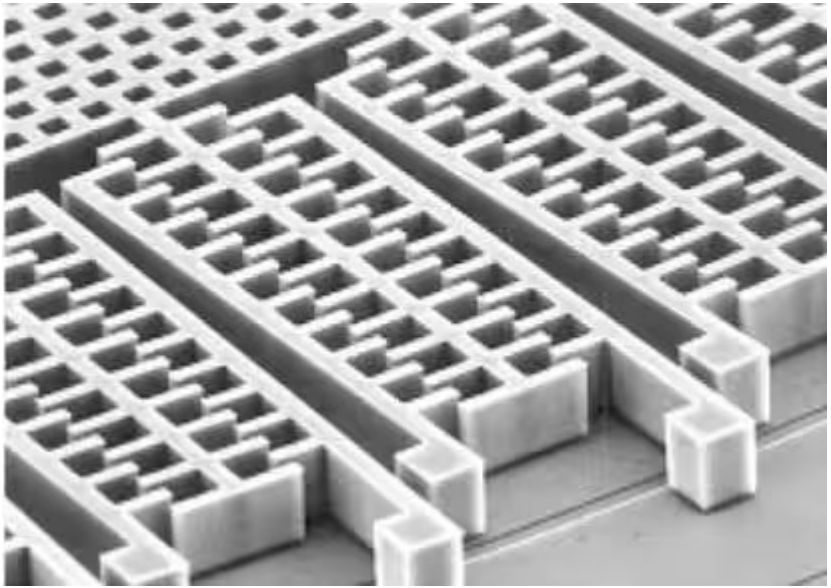
Inercijalna mjerna jedinica - modul **imu_sensor.py**

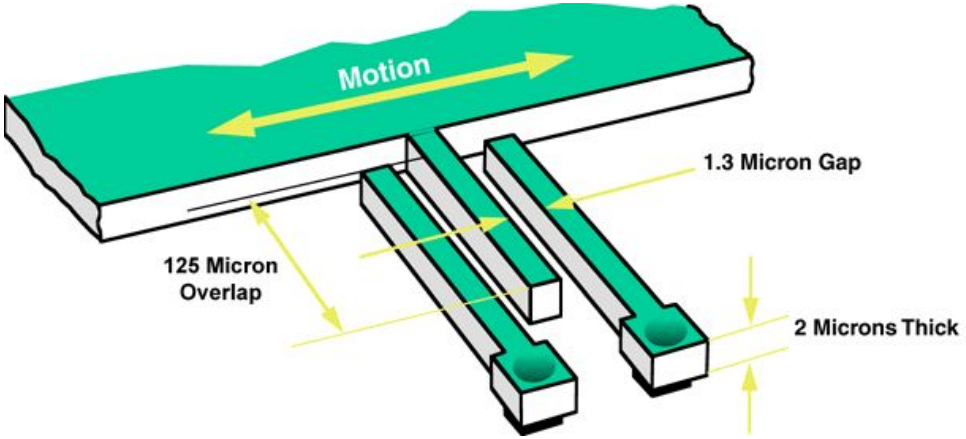
https://content.arduino.cc/assets/st_imu_lsm6ds3_datasheet.pdf

1. LSM6DS3 kombinira **3-osni akcelerometar** i **3-osni žiroskop** za precizno praćenje kretanja i orijentacije letjelice.

2. Senzor pruža podatke visoke preciznosti za **stabilizaciju, navigaciju i autonomne operacije drona**.

3. Dizajniran za **učinkovito korištenje energije**, idealan za primjenu u bespilotnim letjelicama s ograničenom autonomijom.




Motion
1.3 Micron Gap
125 Micron
Overlap
2 Microns Thick

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- IMU modul u kitu

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Spajanje IMU modula

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Korištenje **I2C protokola** za komunikaciju sa senzorom LSM6DS3 omogućuje precizno podešavanje akcelerometra i žiroskopa putem kontrolnih registara.

- Funkcija **read_imu()** očitava podatke o ubrzanju i kutnoj brzini, uz primjenu osjetljivosti i rotacijskih transformacija za pravilno poravnanje.

- **Kalibracijom senzora** se uklanjanjaju greške mjerenja uzrokovane sustavnim greškama akcelerometra i žiroskopa.

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Pregled koda modula

```
class IMUSensor:
    def __init__(self):
        # GPIO Pins Configuration
        self.I2C_SDA_PIN = 
        self.I2C_SCL_PIN = 
        self.I2C_VDD_PIN = 
        self.I2C_GND_PIN = 

        # IMU Configuration
        self.IMU_ADDRESS = 
        self.I2C_FREQ = 400000
        self.WHO_AM_I_REG = 0x0F
        self.CTRL1_XL = 0x10
        self.CTRL2_G = 0x11
        self.OUTX_L_G = 0x22
        self.OUTX_L_XL = 0x28

        # Sensor Calibration Data
        self.gyro_offset = {'x': 0, 'y': 0, 'z': 0}
        self.accel_offset = {'x': 0, 'y': 0, 'z': 0}

        # Initialize IMU Power Pins
        self.vdd_pin = Pin(self.I2C_VDD_PIN, Pin.OUT)
        self.gnd_pin = Pin(self.I2C_GND_PIN, Pin.OUT)

        # Initialize I2C
        self.i2c = None
```

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Pregled koda modula

```
33      def power_on(self):
34          """Power on the IMU and initialize communication."""
35          self.vdd_pin.on()
36          self.gnd_pin.off()
37          time.sleep(0.1)
38          self.i2c = I2C(0, scl=Pin(self.I2C_SCL_PIN), sda=Pin(self.I2C_SDA_PIN), freq=self.I2C_FREQ)
39
40          # Configure IMU Registers
41          self.i2c.writeto_mem(self.IMU_ADDRESS, self.CTRL1_XL, bytes([0b0101_00_01]))  # Accelerometer
42          self.i2c.writeto_mem(self.IMU_ADDRESS, self.CTRL2_G, bytes([0b0101_00_00]))  # Gyroscope
```

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Pregled koda modula
- Tiskana pločica IMU senzora je zarotirana naspram referente orijentacije

```
50      def read_imu(self):
51          """Read data from the IMU."""
52          # Read accelerometer data
53          accel_data = self.i2c.readfrom_mem(self.IMU_ADDRESS, self.OUTX_L_XL, 6)
54          accel_x = self.twos_complement(accel_data[1] << 8 | accel_data[0], 16)
55          accel_y = self.twos_complement(accel_data[3] << 8 | accel_data[2], 16)
56          accel_z = self.twos_complement(accel_data[5] << 8 | accel_data[4], 16)
57
58          # Read gyroscope data
59          gyro_data = self.i2c.readfrom_mem(self.IMU_ADDRESS, self.OUTX_L_G, 6)
60          gyro_x = self.twos_complement(gyro_data[1] << 8 | gyro_data[0], 16)
61          gyro_y = self.twos_complement(gyro_data[3] << 8 | gyro_data[2], 16)
62          gyro_z = self.twos_complement(gyro_data[5] << 8 | gyro_data[4], 16)
63
64          # Adjust for sensitivity
65          accel_sensitivity = 0.000061  # ±2g full scale
66          gyro_sensitivity = 0.00875    # ±245 dps full scale
67
68          accel_x *= accel_sensitivity
69          accel_y *= accel_sensitivity
70          accel_z *= accel_sensitivity
71
72          gyro_x *= gyro_sensitivity
73          gyro_y *= gyro_sensitivity
74          gyro_z *= gyro_sensitivity
75
76          # Apply 90-degree CCW rotation around Z-axis
77          accel_x, accel_y = -accel_y, -accel_x
78          gyro_x, gyro_y = gyro_y, -gyro_x
79
```

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Pregled koda modula

```
80          # Apply calibration offsets
81          accel_x -= self.accel_offset['x']
82          accel_y -= self.accel_offset['y']
83          accel_z -= self.accel_offset['z']
84
85          gyro_x -= self.gyro_offset['x']
86          gyro_y -= self.gyro_offset['y']
87          gyro_z -= self.gyro_offset['z']
88
89          return {'accel': {'x': accel_x, 'y': accel_y, 'z': accel_z},
90                  'gyro': {'x': gyro_x, 'y': gyro_y, 'z': gyro_z}}
91
```

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Pregled koda modula

```
92      def calibrate(self):
93          """Calibrate sensors and log the results."""
94
95          samples = 100
96          accel_sum = {'x': 0, 'y': 0, 'z': 0}
97          gyro_sum = {'x': 0, 'y': 0, 'z': 0}
98
99          for _ in range(samples):
100             data = self.read_imu()
101             for axis in ['x', 'y', 'z']:
102                 accel_sum[axis] += data['accel'][axis]
103                 gyro_sum[axis] += data['gyro'][axis]
104             # Subtract 1g expected in z-direction
105             accel_sum['z'] -= 1.0
106             time.sleep(0.01)
107
108         for axis in ['x', 'y', 'z']:
109             self.accel_offset[axis] = accel_sum[axis] / samples
110             self.gyro_offset[axis] = gyro_sum[axis] / samples
111
112
113         print("Calibration done!")
114         print(f"Accelerometer offsets: {self.accel_offset}")
115         print(f"Gyroscope offsets: {self.gyro_offset}")
```

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Pregled koda primjera uporabe modula

```
120    from status_led import StatusLED
121    from kill_switch import KillSwitch
122    from flight_logger import FlightLogger
123
124    # Initialize modules
125    kill_switch = KillSwitch()
126    led = StatusLED()
127    logger = FlightLogger()
128    imu = IMUSensor()
129
130    logger.start()
131    logger.log("IMU test program started")
132
133    print("Waiting for kill switch to be deactivated...")
134    led.start_blinking(0.5)
```

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Pregled koda primjera uporabe modula

```
136     try:
137         while kill_switch.is_activated():
138             time.sleep(0.1)
139 
140         logger.log("Kill switch deactivated")
141         print("Kill switch deactivated!")
142         led.stop_blinking()
143         led.turn_on()
144 
145         # Power on and calibrate the IMU
146         imu.power_on()
147         logger.log("IMU powered on")
148         print("IMU powered on, starting calibration...")
149         led.start_blinking(0.1)
150         imu.calibrate()
151         logger.log(f"Calibration completed: accel_offset={imu.accel_offset}, gyro_offset={imu.gyro_offset}")
152         led.stop_blinking()
153         led.turn_on()
154 
155         # Run indefinitely to allow workshop participants to play
156         logger.log("IMU entering interactive mode")
157         print("\nIMU interactive mode: Move the IMU and observe readings below.\nPress Ctrl+C to exit.")
158         while True:
159             data = imu.read_imu()
160             print(f"Accelerometer: {data['accel']}, Gyroscope: {data['gyro']}")
161             time.sleep(0.2)
162 
```

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Pregled koda primjera uporabe modula

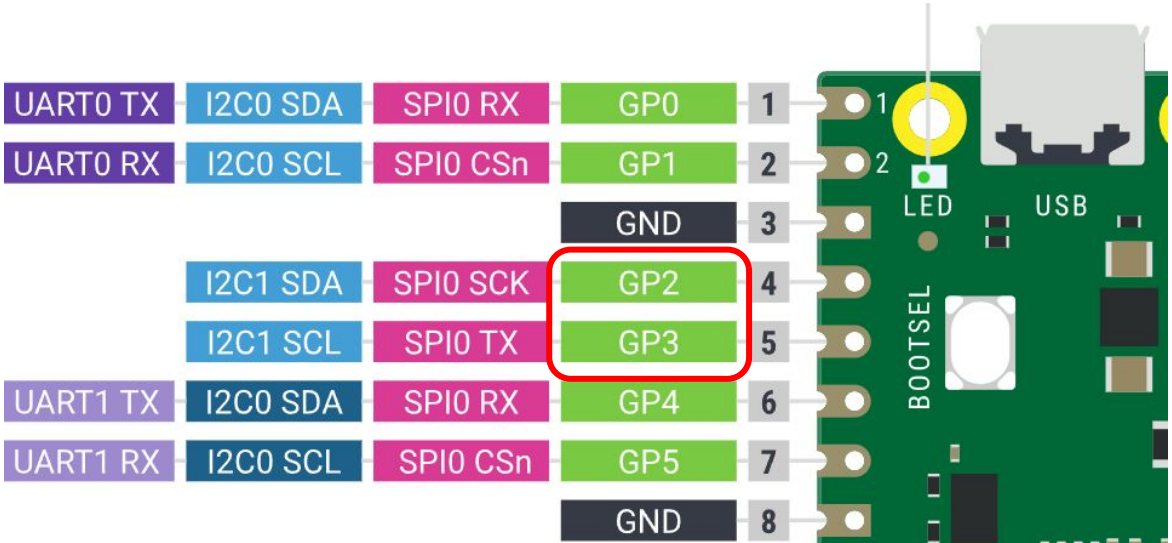
```
163     except KeyboardInterrupt:
164         logger.log("Program interrupted by user")
165         print("\nExiting program.")
166 
167     finally:
168         led.turn_off()
169         logger.stop()
170         print(f"Log saved to {logger.file_name}")
171 
```

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Na koje pinove je spojeno napajanje IMU senzora?

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Na koje pinove je spojeno napajanje IMU senzora? **3 (VDD) i 2 (GND)**

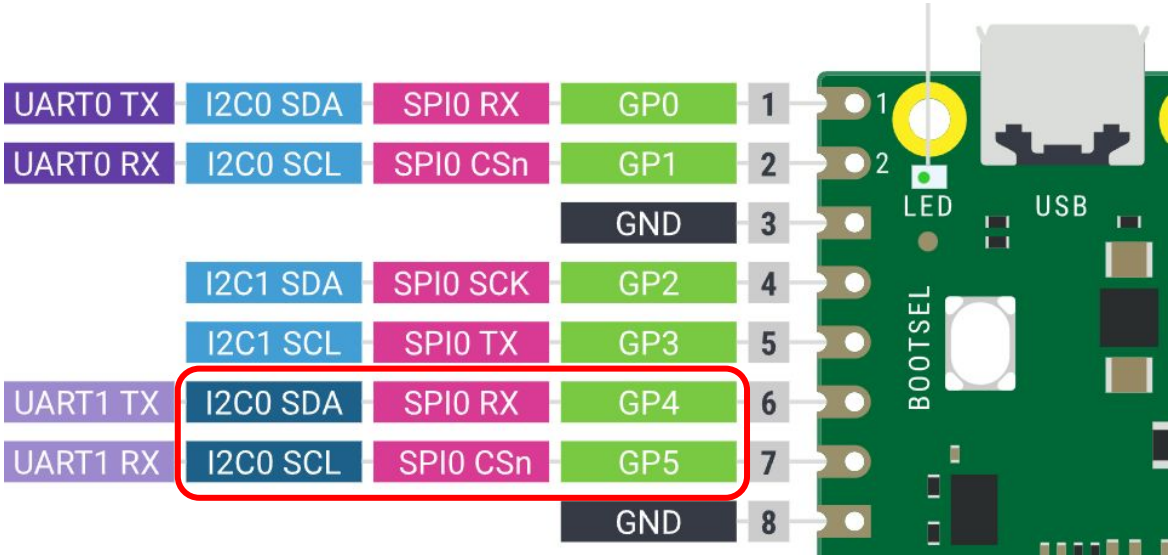


Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Na koje pinove je spojena I2C sabirnica IMU senzora?

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Na koje pinove je spojena I2C sabirnica IMU senzora? **5 (SCL) i 4 (SDA)**



Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Koja je I2C adresa IMU senzora ako je pin SA0 = 1?

Inercijalna mjerna jedinica - modul **imu_sensor.py**

- Koja je I2C adresa IMU senzora ako je pin SA0 = 1? **0b110_1011**

  0x6B

The Slave ADdress (SAD) associated to the LSM6DS3 is 110101xb. The SDO/SA0 pin can be used to modify the less significant bit of the device address. If the SDO/SA0 pin is connected to the supply voltage, LSb is '1' (address 1101011b); else if the SDO/SA0 pin is connected to ground, the LSb value is '0' (address 1101010b). This solution permits to connect and address two different inertial modules to the same I^2C bus.

Inercijalna mjerna jedinica - modul **imu_sensor.py**

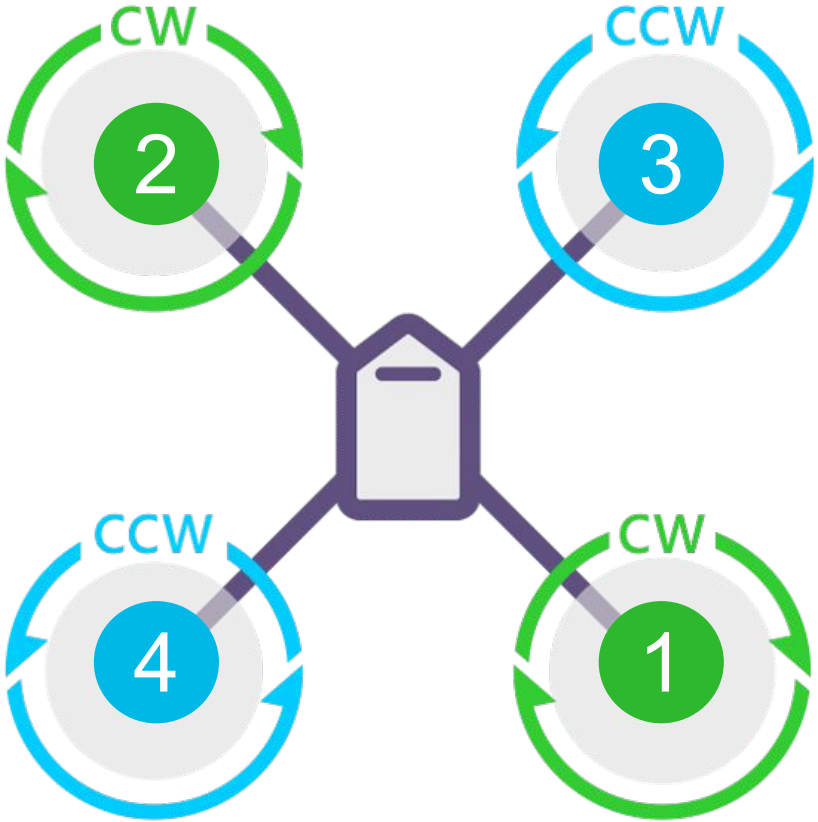
Zadatak:

- Promijenite broj uzoraka za kalibraciju na 300 uzoraka.

Upravljanje motorima - modul **motor_control.py**

- Prikazuje raspored četiri motora drona, gdje se parni motori (1 i 2) okreću u suprotnom smjeru kazaljke na satu (CCW), a neparni (3 i 4) u smjeru kazaljke na satu (CW).

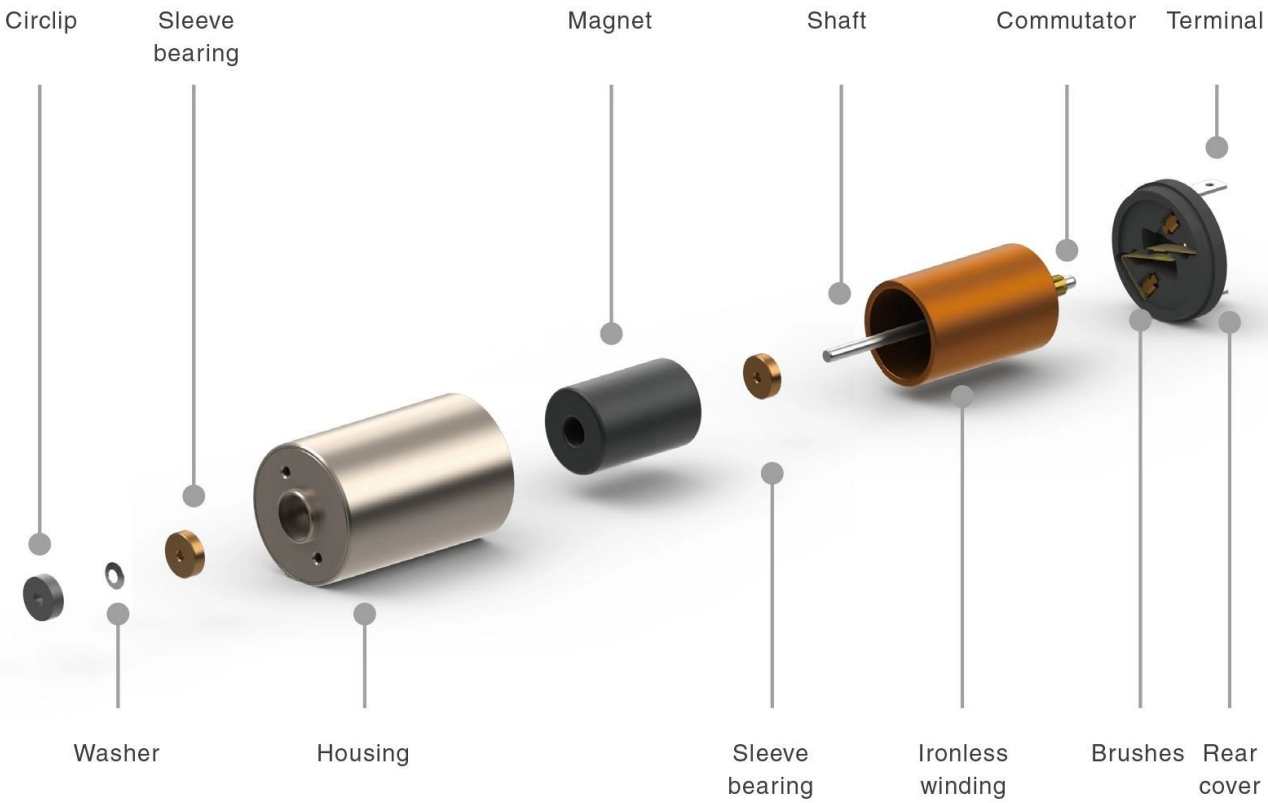
- Suprotni smjerovi rotacije motora **balansiraju zakretni moment** i stabilnost drona tijekom leta.



Upravljanje motorima - modul **motor_control.py**

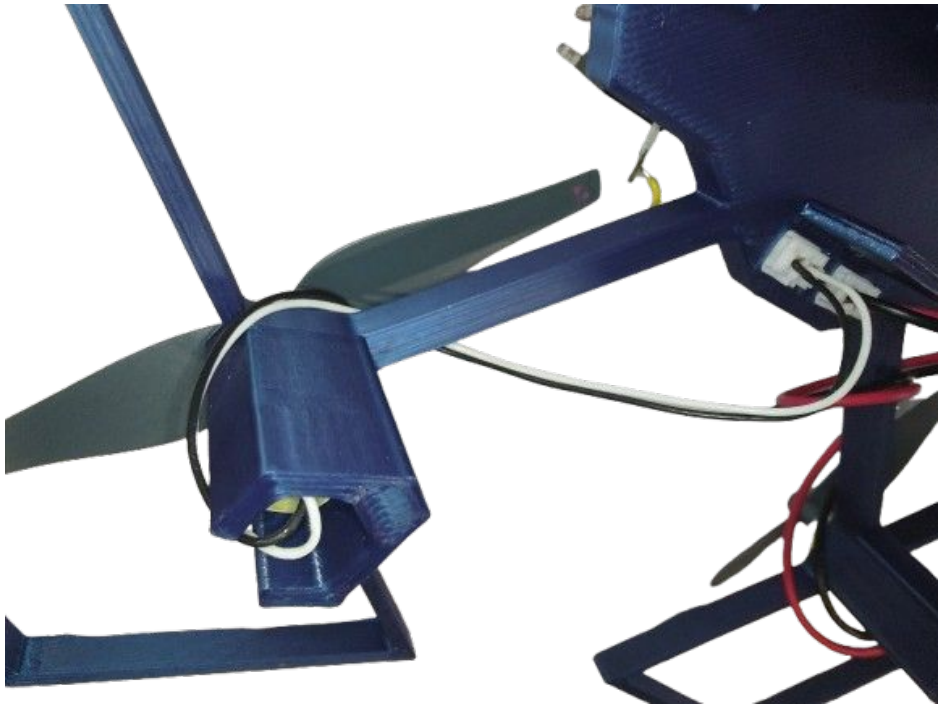
- DC **motor 816** ima **promjer od 8 mm** i **duljinu od 16 mm**, dizajniran za rad na **nominalnom naponu od 3.7 V** (tipično litij-ionska baterija).

- Motor postiže brzinu od **~50,000 RPM** pri punom naponu, što s **propelerom od 75 mm** generira potisnu silu od **30–35 grama po motoru**, omogućujući stabilan let **mikro dronova težine do 150 grama**.


Circlip
Sleeve bearing
Magnet
Shaft
Commutator
Terminal
Washer
Housing
Sleeve bearing
Ironless winding
Brushes
Rear cover

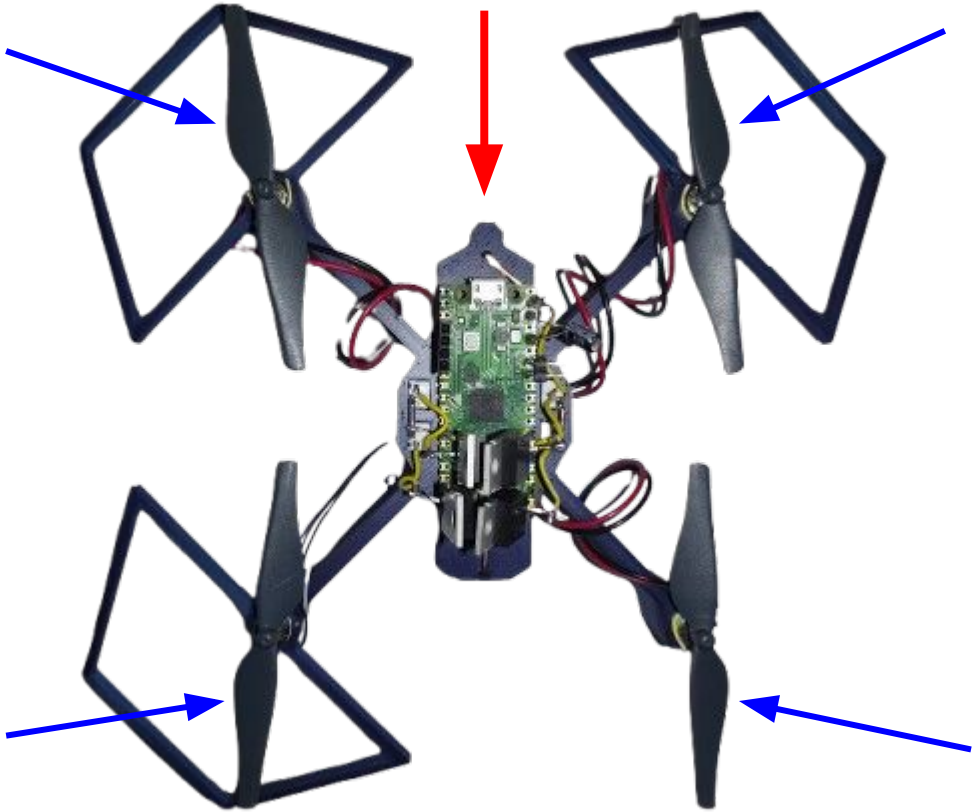
Upravljanje motorima - modul **motor_control.py**

- Montaža i ožičenje motora



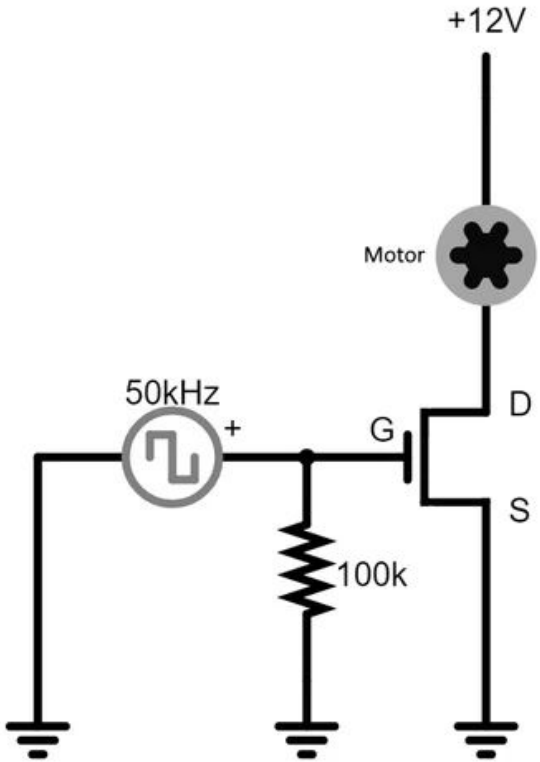
Upravljanje motorima - modul **motor_control.py**

- Montaža propelera



Upravljanje motorima - modul **motor_control.py**

- **Precizna kontrola brzine** pojedinih motora pomoću **PWM signala** i funkcije set_motor_throttle().
- Tokom inicijalizacije se postavlja PWM frekvencija i pokreće svaki motor s brzinom 0 **sigurnosti** radi.
- Funkcija **stop_all_motors()** brzo isključuje sve motore u hitnim slučajevima ili pri završetku letne sekvence.



Upravljanje motorima - modul **motor_control.py**

- Pregled koda modula

```
4  class MotorControl:
5      MAX_THROTTLE = 65535  # Maximum PWM duty cycle for operational use
6
7      def __init__(self):
8          self.MOTOR_PINS = {
9              "front_right":    # Motor 3
10             "rear_right":     # Motor 1
11             "front_left":     # Motor 2
12             "rear_left":      # Motor 4
13         }
14         self.motors = {}
15         self.initialize_motors()
16
17     def initialize_motors(self):
18         """Initialize PWM for each motor."""
19         for name, pin_num in self.MOTOR_PINS.items():
20             pwm = PWM(Pin(pin_num))
21             pwm.freq(200)  # ESCs typically require 50-500 Hz PWM frequency
22             pwm.duty_u16(0)  # Start with motors off
23             self.motors[name] = pwm
24
25     def set_motor_throttle(self, motor_name, throttle):
26         """Set the throttle for a specific motor."""
27         throttle = max(0, min(self.MAX_THROTTLE, int(throttle)))  # Clamp throttle
28         self.motors[motor_name].duty_u16(throttle)
29
30     def stop_all_motors(self):
31         """Turn off all motors."""
32         for motor in self.motors.values():
33             motor.duty_u16(0)
```

Upravljanje motorima - modul **motor_control.py**

- Pregled koda primjera uporabe modula

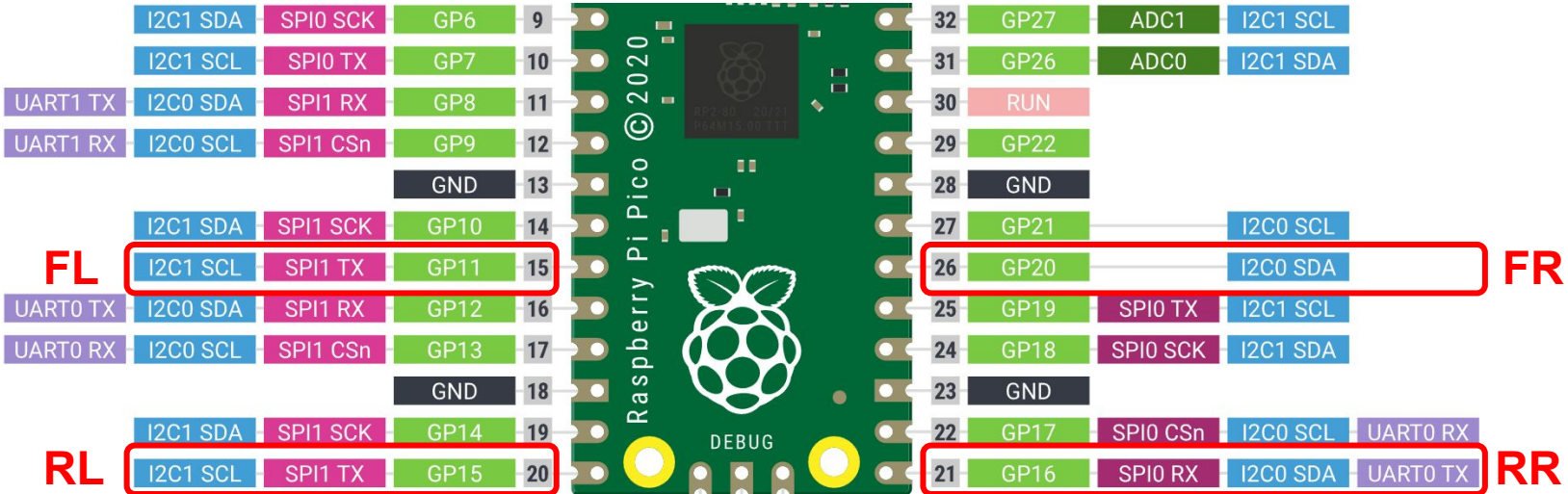
```
67        # Test each motor
68        for motor_name in motor_control.MOTOR_PINS.keys():
69            print(f"Testing motor: {motor_name}")
70            logger.log(f"Testing motor: {motor_name}")
71
72            # Ramp up
73            for throttle in range(0, MAX_TEST_THROTTLE + 1, MAX_TEST_THROTTLE // STEPS):
74                motor_control.set_motor_throttle(motor_name, throttle)
75                logger.log(f"{motor_name} throttle={throttle}")
76                time.sleep(RAMP_DURATION / STEPS)
77
78            # Ramp down
79            for throttle in range(MAX_TEST_THROTTLE, -1, -MAX_TEST_THROTTLE // STEPS):
80                motor_control.set_motor_throttle(motor_name, throttle)
81                logger.log(f"{motor_name} throttle={throttle}")
82                time.sleep(RAMP_DURATION / STEPS)
83
84            # Turn off the motor
85            motor_control.set_motor_throttle(motor_name, 0)
86            logger.log(f"{motor_name} test completed")
87
88        print("Motor testing completed. All motors are off.")
89        logger.log("Motor testing completed")
```

Upravljanje motorima - modul **motor_control.py**

- Na koji pin je spojen koji motor?
  - Prednji lijevi (**F**ront **L**eft)
  - Stražnji lijevi (**R**ear **L**eft)
  - Prednji desni (**F**ront **R**ight)
  - Stražnji desni (**R**ear **R**ight)

Upravljanje motorima - modul **motor_control.py**

- Na koji pin je spojen koji motor?
  - Prednji lijevi (**F**ront **L**eft) **11**
  - Stražnji lijevi (**R**ear **L**eft) **15**
  - Prednji desni (**F**ront **R**ight) **20**
  - Stražnji desni (**R**ear **R**ight) **16**



Upravljanje motorima - modul **motor_control.py**

Zadatak:

- Provjerite stvaraju li svi motori potisak tj. pušu s donje strane.
- Odredite frekvenciju PWM-a pri kojoj se više ne čuje zujanje motora?

Dan II

Estimacija orijentacije i detekcija sudara

Estimacija orijentacije i detekcija sudara

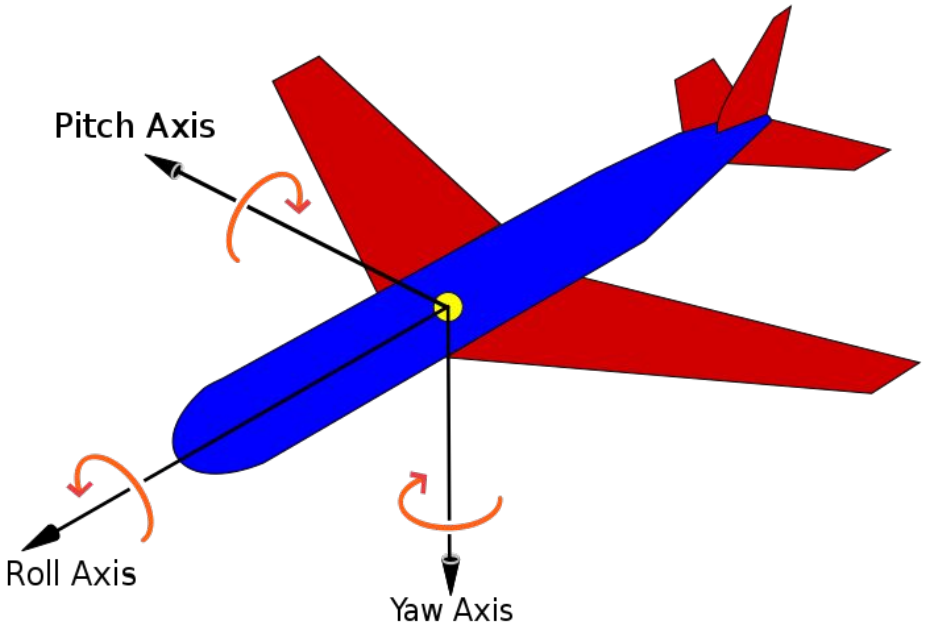
- Kombinira podatke iz IMU senzora (akcelerometar i žiroskop) koristeći algoritme poput komplementarnog filtra ili Kalmanovog filtra za precizno praćenje orijentacije letjelice (roll, pitch, yaw), što je **ključno** za stabilnost i **kontrolu leta**.

- Analizira **nagle promjene akceleracija** ili **prekoračenje kutnih praga** (npr. ±45° roll/pitch) kako bi otkrila sudare ili nestabilnosti u letu.

- Algoritmi omogućuju **pravovremeno isključivanje motora** ili pokretanje sigurnosnih protokola za minimiziranje štete pri sudaru i očuvanje sigurnosti sustava.

Orijentacija u prostoru

- Roll, pitch and yaw (heading)
- Hrvatsko_zrakoplovno_nazivlje



Orijentacija u prostoru - Propinjanje

- engl. pitch → propinjanje, gibanje letjelice oko poprečne osi

Orijentacija u prostoru - Valjanje

- engl. roll → valjanje, gibanje oko uzdužne osi letjelice

Orijentacija u prostoru - Kurs

- engl. heading → kurs, kut između temeljnoga smjera sjever i crte kursa

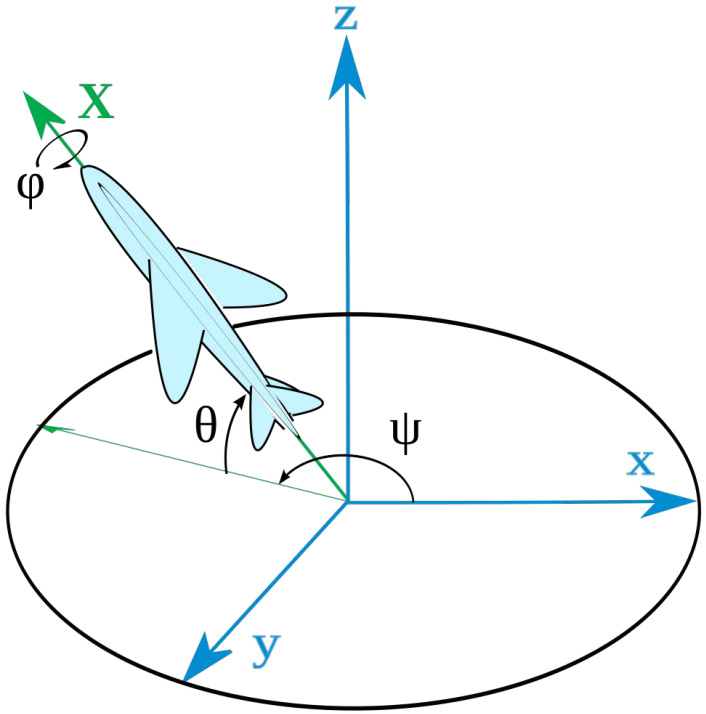
Estimacija orijentacije - modul **orientation_estimator.py**

- Procjena temeljem žiroskopa:

$$\begin{aligned}\text{Roll}\ (\phi) &= \phi_{\text{prev}} + \omega_x \cdot \Delta t,\\ \text{Pitch}\ (\theta) &= \theta_{\text{prev}} + \omega_y \cdot \Delta t,\\ \text{Yaw}\ (\psi) &= \psi_{\text{prev}} + \omega_z \cdot \Delta t,\end{aligned}$$

- Procjena temeljem akcelerometra:

$$\begin{aligned}\text{Roll}\ (\phi) &= \arctan 2\left(a_y, a_z\right),\\ \text{Pitch}\ (\theta) &= \arctan 2\left(-a_x, \sqrt{a_y^2 + a_z^2}\right),\end{aligned}$$



Estimacija orijentacije - modul **orientation_estimator.py**

- Koristi **komplementarni filter** za procjenu kutova pitch, roll, i yaw kombiniranjem podataka akcelerometra i žiroskopa.
- Komplementarni filter kombinira **kratkoročno precizne podatke žiroskopa** s **dugoročnom stabilnošću akcelerometra** što daje pouzdanu procjenu orijentacije u stvarnom vremenu.
- Normalizira vrijednost yaw kuta na raspon od -180° do +180° kako bi spriječio prelijevanje i održao točnost.
- Parametar **alpha** omogućuje podešavanje težine između žiroskopa i akcelerometra za optimizaciju procjene u različitim uvjetima leta.

Estimacija orijentacije - modul **orientation_estimator.py**

- Komplementarni filter koristi formule:

$$\phi = \alpha \cdot (\phi_{\text{prev}} + \omega_x \cdot \Delta t) + (1 - \alpha) \cdot \phi_{\text{acc}},$$

$$\theta = \alpha \cdot (\theta_{\text{prev}} + \omega_y \cdot \Delta t) + (1 - \alpha) \cdot \theta_{\text{acc}},$$

Estimacija orijentacije - modul **orientation_estimator.py**

- Web aplikacija simulacije komplementarnog filtra:

  https://educopter-fwtbnspreugkicoefwv8ru.streamlit.app/

Estimacija orijentacije - modul **orientation_estimator.py**

- Pregled koda modula

```
5  class OrientationEstimator:
6      def __init__(self):
7          self.angle = {'pitch': 0.0, 'roll': 0.0, 'yaw': 0.0}
8          self.alpha = 0.9  # Complementary filter coefficient
9
10     def normalize_yaw(self, yaw):
11         """Normalize yaw to the range -180° to +180°."""
12         while yaw > 180:
13             yaw -= 360
14         while yaw < -180:
15             yaw += 360
16         return yaw
```

Estimacija orijentacije - modul **orientation_estimator.py**

- Pregled koda modula

```python
def complementary_filter(self, data, dt):
    """Update orientation using a complementary filter."""
    gyro_rate = {
        'pitch' : data['gyro']['y'],  # Already calibrated
        'roll'  : data['gyro']['x'],   # Already calibrated
        'yaw'   : data['gyro']['z'],    # Already calibrated
    }

    # Gyroscope integration for angular rate
    self.angle['pitch'] += gyro_rate['pitch'] * dt
    self.angle['roll'] += gyro_rate['roll'] * dt
    self.angle['yaw'] += gyro_rate['yaw'] * dt

    # Normalize yaw to avoid overflow
    self.angle['yaw'] = self.normalize_yaw(self.angle['yaw'])

    # Accelerometer angle estimation
    accel_angle_pitch = math.atan2(
        data['accel']['x'],
        math.sqrt(data['accel']['y']**2 + data['accel']['z']**2)
    ) * (180 / math.pi)

    accel_angle_roll = math.atan2(
        data['accel']['y'],
        data['accel']['z']
    ) * (180 / math.pi)

    # Complementary filter
    self.angle['pitch'] = self.alpha * self.angle['pitch'] + (1 - self.alpha) * accel_angle_pitch
    self.angle['roll'] = self.alpha * self.angle['roll'] + (1 - self.alpha) * accel_angle_roll
    # Note: Yaw is gyroscope-only (requires magnetometer for full correction)

    # Return the current pitch, roll, and yaw angles.
    return self.angle
```

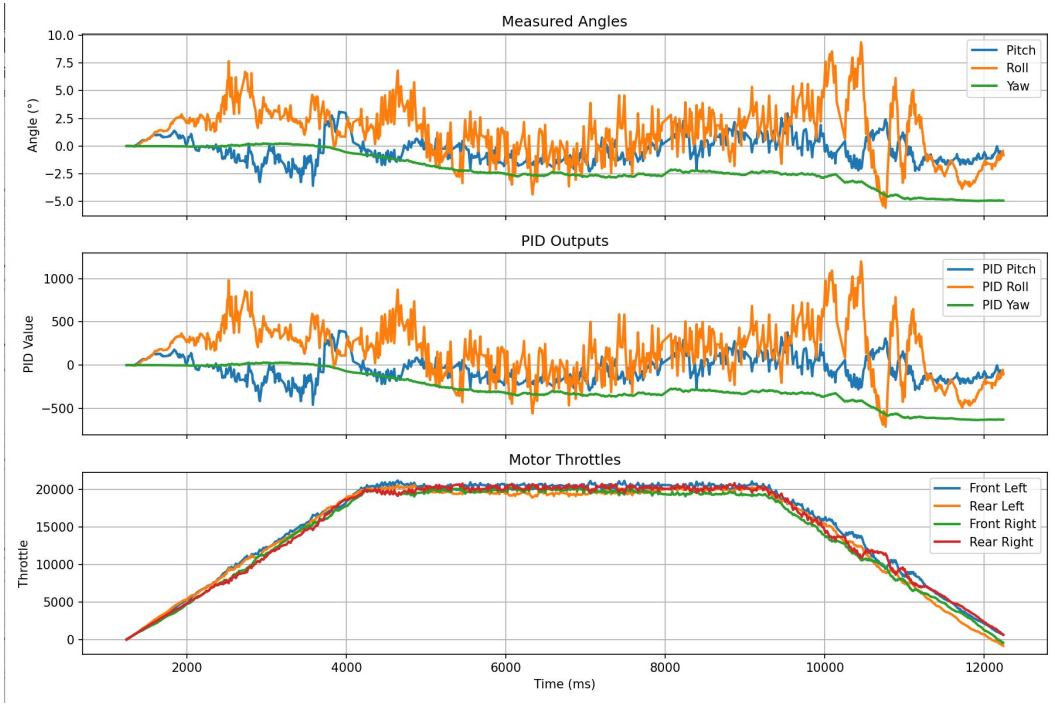
Estimacija orijentacije - modul **orientation_estimator.py**

- Pregled koda primjera uporabe modula

```
93        while True:
94            current_time = time.ticks_ms()
95            dt = time.ticks_diff(current_time, last_time) / 1000.0  # Convert to seconds
96            last_time = current_time
97
98            # Read IMU data (already calibrated)
99            data = imu.read_imu()
100
101           # Update orientation
102           angles = orientation.complementary_filter(data, dt)
103
104           # Retrieve angles and log
105           print(f"Pitch: {angles['pitch']:.2f}°, Roll: {angles['roll']:.2f}°, Yaw: {angles['yaw']:.2f}")
106           logger.log(f"Orientation: pitch={angles['pitch']:.2f}, roll={angles['roll']:.2f}, yaw={angles['yaw']:.2f}")
107
108           time.sleep(0.2)
```

Estimacija orijentacije - modul **orientation_estimator.py**

- Prikaz podataka iz dijagnostičkog zapisa



- Instalirati:
  - Python
  - **pip install matplotlib** u CMD

- Pokretanje skripte:
  - python .**visualize_flight_log_orientation_estimator.py**
  - python .**visualize_flight_log_flight_controller.py**

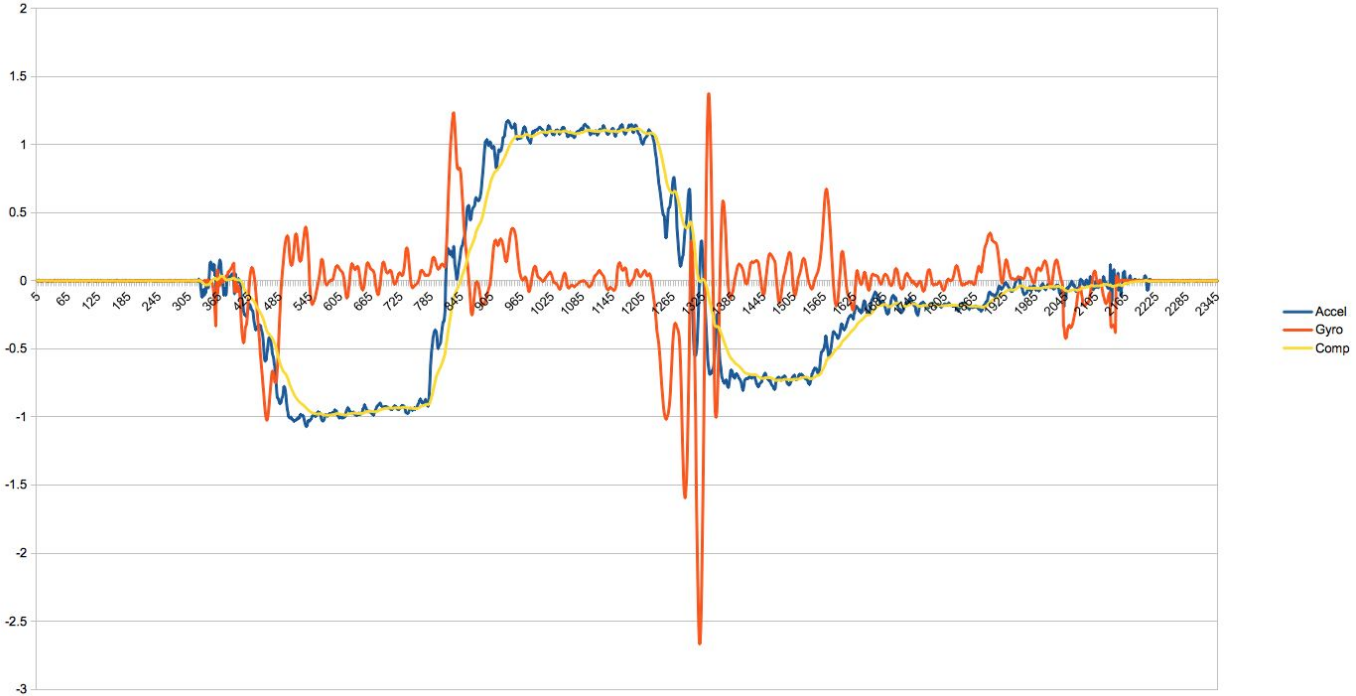
Estimacija orijentacije - modul **orientation_estimator.py**

Zadatak:

- Isprobajte iznose komplementarnog filtra:
  - alpha = 0.1
  - alpha = 0.95
  - alpha = 0.99

Estimacija orijentacije - modul **orientation_estimator.py**

- Primjer podataka za komplementarni filter



Detekcija sudara - modul **crash_detector.py**

- Analizira kutove nagiba (pitch i roll) kako bi otkrio **nenormalne položaje drona**.
- Aktivira detekciju kada kutovi premaše **unaprijed definirani prag** od ±45°.
- Jednostavna logika omogućuje **brzo prepoznavanje** potencijalnih sudara ili nestabilnosti.

Detekcija sudara - modul **crash_detector.py**

- Pregled koda modula

```python
class CrashDetector:
    ANGLE_THRESHOLD = 45

    def detect_crash(self, angle):
        """
        Detect a crash based on pitch and roll angles.
        Returns True if either roll or pitch exceeds ±45°.
        """
        return abs(angle['roll']) > self.ANGLE_THRESHOLD or abs(angle['pitch']) > self.ANGLE_THRESHOLD

```

Detekcija sudara - modul **crash_detector.py**

- Pregled koda primjera uporabe modula

```
67          # Check for crash
68          if crash_detector.detect_crash(angles):
69              print("Crash detected!")
70              logger.log(f"Crash detected at angles: pitch={angles['pitch']:.2f}, roll={angles['roll']:.2f}")
71              led.start_blinking(0.1)  # Rapid blinking to signal crash
72              time.sleep(2)  # Allow observation
73              led.stop_blinking()
74              break
```

Detekcija sudara - modul **crash_detector.py**

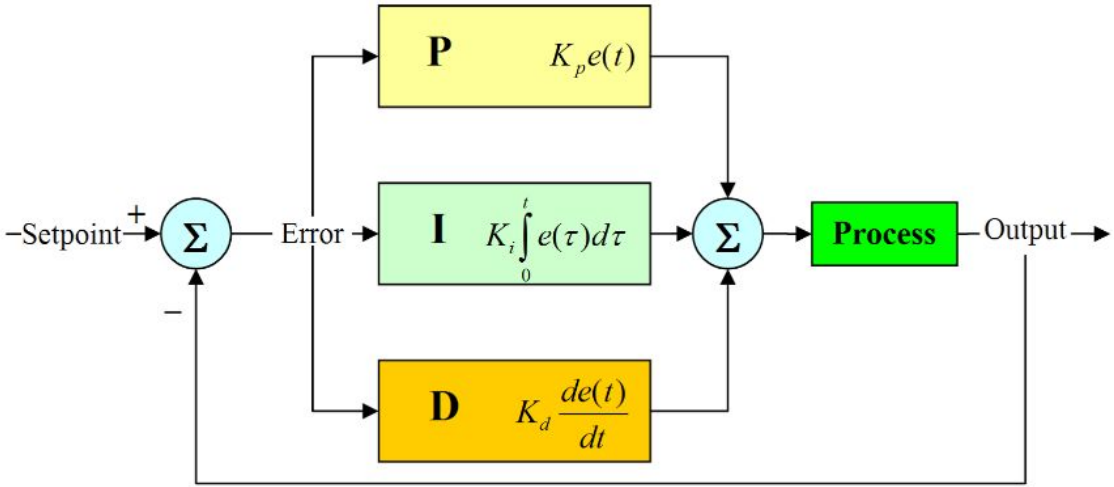
Zadatak:

- Promijenite granicu otklona roll and pitch na npr. 15° i 60°.

Dan II

Programiranje kontrolera leta (PID regulator i logika leta)

PID regulator orijentacije



- PID regulator kombinira **proporcionalni, integracijski i derivacijski dio** kako bi minimizirao pogrešku između zadane vrijednosti (Setpoint) i stvarnog izlaza procesa, pri čemu se podešava parametrima **K_p** (proporcionalnost), **K_i** (integracija), i **K_d** (derivacija).

PID regulator orijentacije

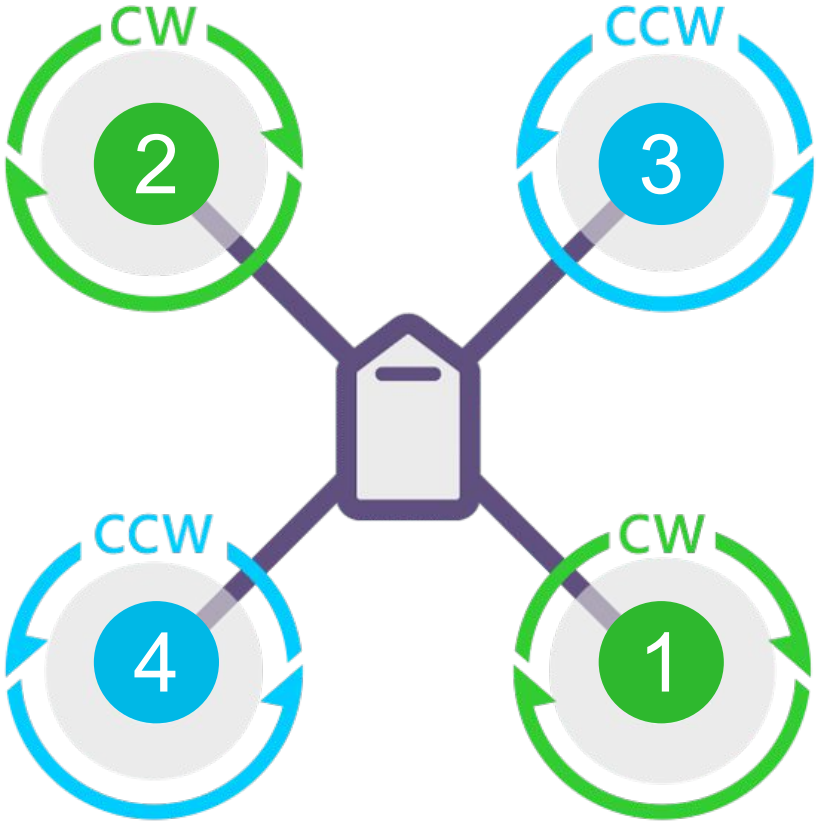
- Web aplikacija simulacije komplementarnog filtra:

https://educopter-chem8scbexdfjuugub8kdx.streamlit.app/

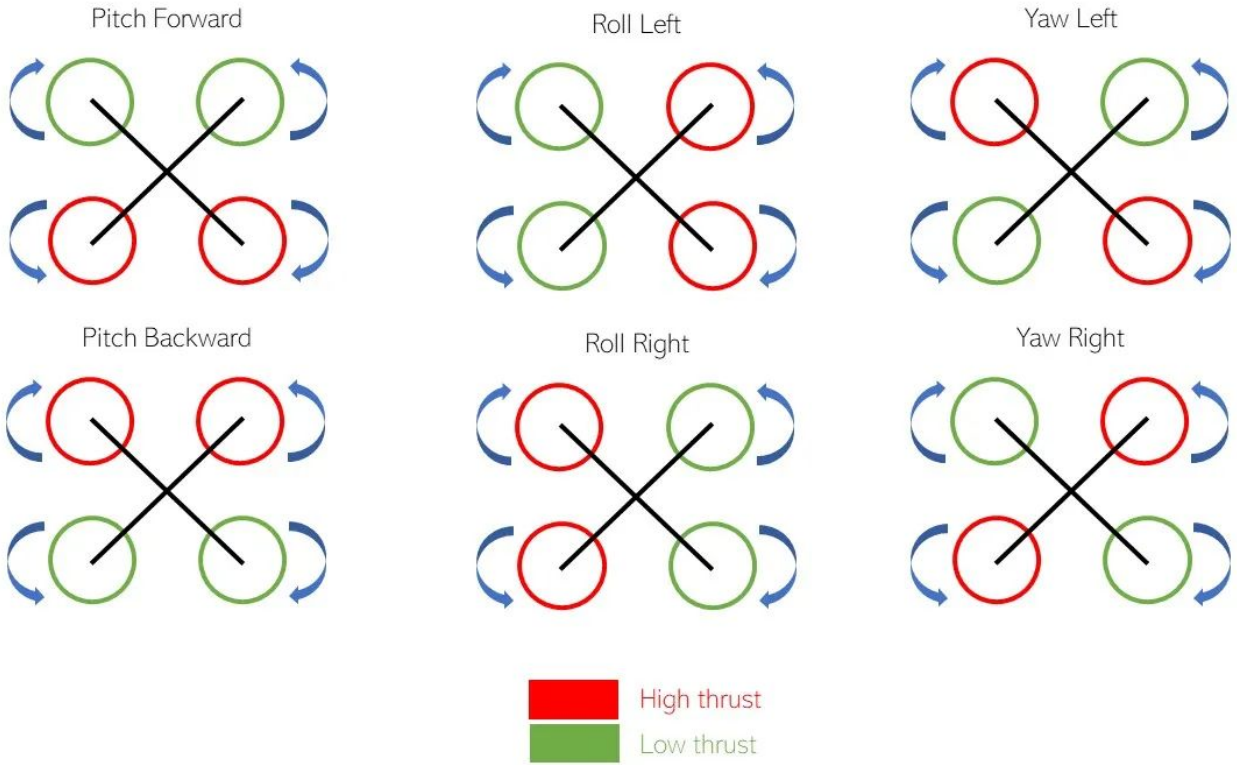
Raspodjela korekcija po motorima

- Prikazuje raspored četiri motora drona, gdje se parni motori (1 i 2) okreću u suprotnom smjeru kazaljke na satu (CCW), a neparni (3 i 4) u smjeru kazaljke na satu (CW).

- Suprotni smjerovi rotacije motora **balansiraju zakretni moment** i stabilnost drona tijekom leta.



Raspodjela korekcija po motorima



Kontroler leta - modul **flight_controller.py**

- **PID regulator** osigurava stabilno upravljanje letom drona koristeći proporcionalne, integracijske i derivacijske komponente za minimiziranje pogreške.

- Omogućuje postavljanje **granica na izlaz** kako bi se spriječilo prekomjerno ubrzanje ili nestabilno ponašanje motora.

- Distribucija brzine po motorima **kombinira osnovnu brzinu s PID korekcijama** za roll, pitch i yaw radi stabilizacije i kontrole letjelice.

Kontroler leta - modul **flight_controller.py**

- Pregled koda modula

```python
class PID:
    def __init__(self, Kp, Ki, Kd, output_limits=(None, None)):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.output_limits = output_limits

        self._integral = 0
        self._previous_error = 0

    def compute(self, error, dt):
        self._integral += error * dt
        derivative = (error - self._previous_error) / dt if dt > 0 else 0

        output = self.Kp * error + self.Ki * self._integral + self.Kd * derivative

        # Apply output limits
        min_output, max_output = self.output_limits
        if min_output is not None:
            output = max(min_output, output)
        if max_output is not None:
            output = min(max_output, output)

        self._previous_error = error

        return output
```

Kontroler leta - modul **flight_controller.py**

- Pregled koda modula

```python
class FlightController:
    def __init__(self):
        # Initialize PID regulators
        self.pid_roll   = PID(Kp=128.0, Ki=0.0, Kd=0.0, output_limits=(-10000, 10000))
        self.pid_pitch  = PID(Kp=128.0, Ki=0.0, Kd=0.0, output_limits=(-10000, 10000))
        self.pid_yaw    = PID(Kp= 128.0, Ki=0.0, Kd=0.0, output_limits=(-10000, 10000))


    def compute_motor_throttles(self, measured_angles, target_angles, dt, base_throttle = 55000):
        """Compute motor throttles using PID controllers."""
        pid_roll    = self.pid_roll.compute(measured_angles['roll'] - target_angles['roll'], dt)
        pid_pitch   = self.pid_pitch.compute(measured_angles['pitch'] - target_angles['pitch'], dt)
        pid_yaw     = self.pid_yaw.compute(measured_angles['yaw'] - target_angles['yaw'], dt)

        self.pid_outputs = {'roll'  : pid_roll,
                            'pitch' : pid_pitch,
                            'yaw'   : pid_yaw}

        # Compute motor speeds corrections
        return {
            "front_right"   : base_throttle - pid_roll - pid_pitch + pid_yaw,
            "rear_right"    : base_throttle - pid_roll + pid_pitch - pid_yaw,
            "front_left"    : base_throttle + pid_roll - pid_pitch - pid_yaw,
            "rear_left"     : base_throttle + pid_roll + pid_pitch + pid_yaw
        }
```

Kontroler leta - modul **flight_controller.py**

- Pregled koda primjera uporabe modula

```
while True:
    current_time = time.ticks_ms()
    dt = time.ticks_diff(current_time, last_time) / 1000.0  # Convert to seconds
    last_time = current_time

    # Read IMU data and update orientation
    imu_data = imu.read_imu()
    measured_angles = orientation.complementary_filter(imu_data, dt)

    # Check for crash
    if crash_detector.detect_crash(measured_angles):
        print("Crash detected! Stopping all motors.")
        logger.log(f"Crash detected at angles: pitch={measured_angles['pitch']:.2f}, roll={measured_angles['roll']:.2f}")
        motor_control.stop_all_motors()
        led.start_blinking(0.1)  # Rapid blinking to signal crash
        time.sleep(5)
        led.stop_blinking()
        break

    # Compute motor throttles
    motor_throttles = flight_controller.compute_motor_throttles(measured_angles, target_angles, dt, base_throttle = 10_000)

    # Apply motor throttles
    for motor_name, throttle in motor_throttles.items():
                motor_control.set_motor_throttle(motor_name, throttle)

    logger.log(     f"{measured_angles['pitch']:.2f}" +','
                    f"{measured_angles['roll']:.2f}" +','
                    f"{measured_angles['yaw']:.2f}" +','
                    f"{flight_controller.pid_outputs['pitch']:.2f}" +','
                    f"{flight_controller.pid_outputs['roll']:.2f}" +','
                    f"{flight_controller.pid_outputs['yaw']:.2f}" +','
                    f"{motor_throttles['front_left']}" +','
                    f"{motor_throttles['rear_left']}" +','
                    f"{motor_throttles['front_right']}" +','
                    f"{motor_throttles['rear_right']}")
```

Kontroler leta - modul **flight_controller.py**

Zadatak:

- Odredite frekvenciju osvježavanja glavne upravljačke petlje.

```
print(f”{1.0/dt if dt > 0.0 else 0.0}”)
```

Kontroler leta - modul **flight_controller.py**

Zadatak:

- Okrećite letjelicu u zraku po svakoj osi i promatrajte kako se mijenjanju brzine pojedinih motora:
  - Roll
  - Pitch
  - Yaw

Kontroler leta - modul **flight_controller.py**

Postupak ugađanja PID kontrolera:

1. Početna konfiguracija:
    a. Postavite Kp, Ki i Kd na nulu.
    b. Započnite s ugađanjem roll/pitch osi, dok je yaw os deaktivirana

2. Podešavanje proporcionalnog dijela (Kp):
    a. Postepeno povećavajte dok letjelica održava stabilnost.
    b. Preveliki uzrokovat će oscilacije. Smanjite ga do točke gdje oscilacije prestanu.

Kontroler leta - modul **flight_controller.py**

Postupak ugađanja PID kontrolera:

3. Dodavanje derivacijskog dijela (Kd):

   a. Postepeno povećavajte Kd kako biste smanjili preostale oscilacije i poboljšali odziv na nagle promjene.
   b. Preveliki Kd može uzrokovati spor odziv ili drhtanje letjelice.

4. Dodavanje integracijskog dijela (Ki):

   a. Polako povećavajte kako biste eliminirali dugoročne pogreške (npr. odstupanje od ciljanog kuta).
   b. Pazite na povećanje oscilacija jer preveliki Ki može destabilizirati sustav.

Kontroler leta - modul **flight_controller.py**

Postupak ugađanja PID kontrolera:

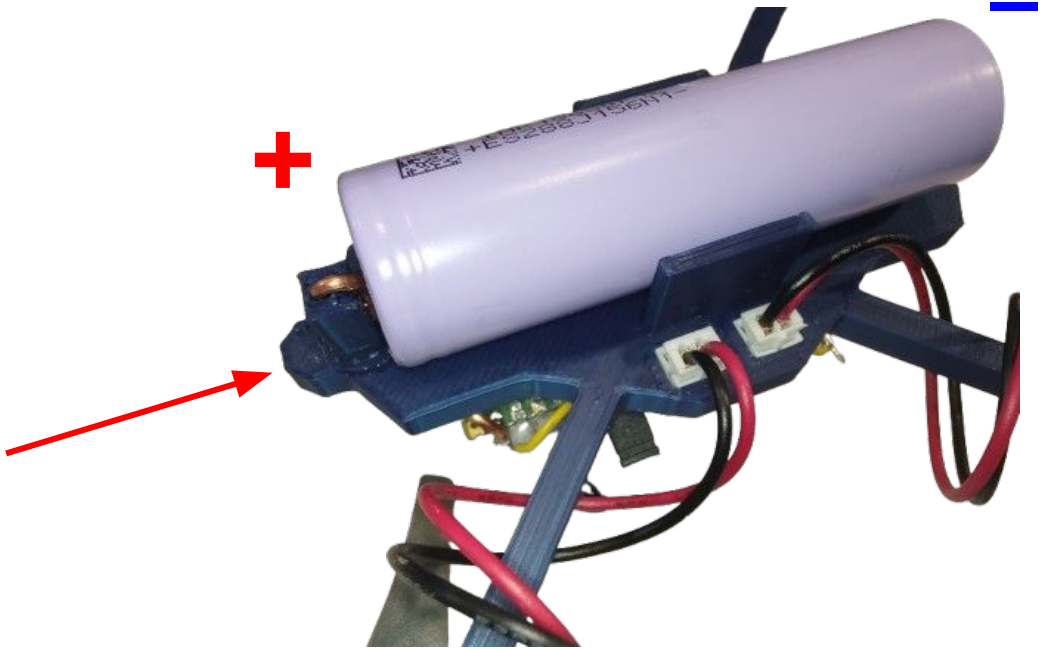
5. Ugađanje yaw osi:

    a. Nakon što su roll i pitch stabilizirani, primijenite isti postupak za yaw.
    b. Yaw obično zahtijeva manje koeficijente jer nema potrebe za velikim korekcijama osim u slučaju brzih rotacija.

6. Testiranje i fine-tuning:

    a. Testirajte letjelicu u stvarnim uvjetima (mirnom zraku).
    b. Prilagodite vrijednosti na temelju ponašanja letjelice dok ne postignete stabilan i gladak let.

Umetanje baterije



Testiranje uživo

- Testiranje uživo započeti s nižim iznosima MAX_BASE_THROTTLE
- Obratiti pažnju na ravnomjernost uzgona i po potrebi dodati kalibracijske faktore kod komentara
  - # Compute motor speeds corrections
-

THE END