Projekt

Sterowniki robotów

Etap III

Kamienie milowe II oraz III

Sterownik lotu drona sterowanego wektorem ciągu „Goose”

TVCG

Skład grupy:
Eryk Możdżeń, 259375

Termin: wtTN19

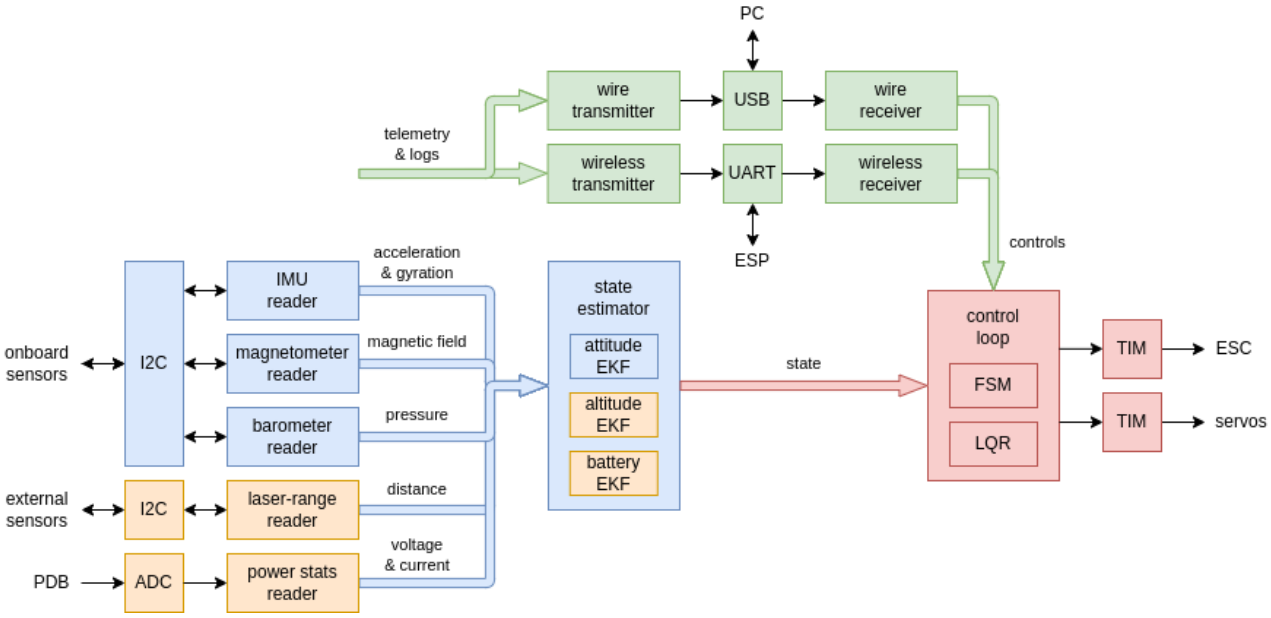
Prowadzący:
dr inż. Wojciech Domski

6 czerwca 2023

Spis treści

1 Zakres wykonanych prac

Wszelkie prace prowadzone i dokumentowane były na repozytorium github[3]. Wykonano wszystkie zadania zdefiniowane w dokumencie „Etap I". Podczas wykonywania projektu zdefiniowane zostały 3 kamienie milowe.



Rysunek 1: Schemat architektury programu,
niebieski – wykonany przed oddaniem I etapu,
zielony – elementy komunikacji (I kamień milowy),
pomarańczowy – sensoryka i estymacja stanu (II kamień milowy),
czerwony – maszyna stanów, regulator i urządzenia wykonawcze (III kamień milowy)

2 Komunikacja

2.1 Przesył danych z użyciem UART oraz ESP8266

Podstawową drogą komunikacji z dronem jest usługa Telnet. Dostęp do niej umożliwia zdalny serwer GDB oraz most UART–Telnet dostępny na mikrokontroler ESP8266 o nazwie blackmagic–espidf[1]. Peryferium UART na głównym mikrokontrolerze STM32F411 zostało skonfigurowane na ciągłe oczekiwanie na przychodzace dane. Odbiór danych z użyciem DMA sygnalizowany jest w przerwaniu wybudzeniem taska. Dedykowany task FreeRTOS odczytuje bajt po bajcie, a następnie formuje wiadomość.

Wysyłanie danych również oddbywa się z pomocą DMA. Dedytkwany task odbiera dane do wysyłki, koduje je, a następnie rozpoczyna transmisje. Przerwanie końca transmisji wybudza task i rozpoczyna następną transmisje. Przerwania zostały pokazane na 1.

```
1  void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
2    if(huart->Instance==USART1) {
3          BaseType_t xHigherPriorityTaskWoken = pdFALSE;
4          vTaskNotifyGiveIndexedFromISR(com_bus_task_tx, 2, &
               xHigherPriorityTaskWoken);
5          portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
6    }
7  }
8
9  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
10   if(huart->Instance==USART1) {
11         BaseType_t xHigherPriorityTaskWoken = pdFALSE;
12         vTaskNotifyGiveIndexedFromISR(com_bus_task_rx, 3, &
               xHigherPriorityTaskWoken);
13         portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
14   }
15 }
```

Listing 1: Przerwania UART – widoczne wybudzanie tasków

2.2 Przesył danych z użyciem USB

W celu łatwiejszej pracy nad oprogramowaniem (oraz programowaniem) zostało wyprowadzone złącze microUSB. Została zastosowana biblioteka ST Middleware USB [5]. Została zasotsowana klasa USB CDC, która tworzy wirutalny port COM. W celu sygnalizacji końca transmisji zostało zdefiniowane przerwanie wybudzające dedykowanego taska do wysyłki danych (analogicznie jak w UART). Przerwanie otrzymania danych pokazane jest na 3.

```
1  static int8_t CDC_TransmitCplt_FS(uint8_t *Buf, uint32_t *Len, uint8_t
       epnum) {
2    (void)Buf;
3    (void)Len;
4    (void)epnum;
5
6    vTaskNotifyGiveIndexedFromISR(wire_transmitter_task, 0, NULL);
7
8    return USBD_OK;
9  }
```

Listing 2: przerwanie końca transmisji USB

Z bliżej nieustalonych powodów podczas obioru danych trzeba zmienić kolejność występowania bajtów w buforze (w UART nie ma takiej konieczności). Funkcja otrzymująca dane pokazana jest na 3.

```
1  static int8_t CDC_Receive_FS(uint8_t* Buf, uint32_t *Len) {
2    USBD_CDC_SetRxBuffer(&hUsbDeviceFS, &Buf[0]);
3
4    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
5
6    for(int i=*Len-1; i>=0; i--) {
7      Transport::getInstance().wire_rx_queue.push_ISR(Buf[i],
           xHigherPriorityTaskWoken);
8    }
9
10   USBD_CDC_ReceivePacket(&hUsbDeviceFS);
11   return USBD_OK;
12 }
```

Listing 3: odiór danych oraz przekazanie bajtów do kolejki

3 Sensoryka i estymacja stanu

3.1 Pomiar odległości z użyciem czujnika VL53L0X

Czujnik VL53L0X jest cyfrowym czujnikiem odległości od firmy STMicroelectronics. Komunikacja przebiega z użyciem protokołu I2C oraz pinu sygnalizującego koniec pomiaru. Do obsługi czujnika została użyta biblioteka dostarczona od producenta [4].

Na początku działania programu czujnik zostaje zresetowany pinem XSHUT, a następnie skonfigurowany w tryb ciągłej konwersji. Parametry pomiaru zostały dobrane tak, aby zmaksymalizować zasięg do 2.2 m dla białej przeszkody. Konfiguracja pokazana jest na 4.

```
1  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
2  vTaskDelay(20);
3  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);
4  vTaskDelay(20);
5
6  VL53L0X_Error status = VL53L0X_ERROR_NONE;
7
8  HAL_NVIC_DisableIRQ(EXTI15_10_IRQn);
9
10 VL53L0X_WaitDeviceBooted(&vlx_sensor);
11 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_DataInit(&vlx_sensor);
12 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_StaticInit(&vlx_sensor);
13 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_PerformRefCalibration(&
       vlx_sensor, &VhvSettings, &PhaseCal);
14 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_PerformRefSpadManagement(&
       vlx_sensor, &refSpadCount, &isApertureSpads);
15 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_SetDeviceMode(&vlx_sensor,
        VL53L0X_DEVICEMODE_CONTINUOUS_RANGING);
16
17 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_SetLimitCheckEnable(&
       vlx_sensor, VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE, 1);
18 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_SetLimitCheckEnable(&
       vlx_sensor, VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE, 1);
19 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_SetLimitCheckValue(&
       vlx_sensor, VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE, (
       FixPoint1616_t)(0.1*65536));
20 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_SetLimitCheckValue(&
       vlx_sensor, VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE, (FixPoint1616_t)
       (60*65536));
21 if(status==VL53L0X_ERROR_NONE) status =
       VL53L0X_SetMeasurementTimingBudgetMicroSeconds(&vlx_sensor, 33000);
22 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_SetVcselPulsePeriod(&
       vlx_sensor, VL53L0X_VCSEL_PERIOD_PRE_RANGE, 18);
23 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_SetVcselPulsePeriod(&
       vlx_sensor, VL53L0X_VCSEL_PERIOD_FINAL_RANGE, 14);
24
25 if(status==VL53L0X_ERROR_NONE) status = VL53L0X_StartMeasurement(&
       vlx_sensor);
26
27 HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
```

Listing 4: konfiguracja VL53L0X

Przerwanie sygnalizujace gotowość danych do odczytu przedstawione jest w 5.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  if(GPIO_Pin==GPIO_PIN_15) {

    // IMU
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    vTaskNotifyGiveIndexedFromISR(imu_task, 1, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

  } else if(GPIO_Pin==GPIO_PIN_5) {

    // magnetometer
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    vTaskNotifyGiveIndexedFromISR(mag_task, 1, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

  } else if(GPIO_Pin==GPIO_PIN_14) {

    // distance
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    vTaskNotifyGiveIndexedFromISR(vlx_task, 1, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);

  }
}
```

Listing 5: konfiguracja VL53L0X

3.2 Estymacja wysokości nad ziemią

Do estymacji wysokości lotu został zastosowany filtr Kalmana sumujący odczyty akcelerometru, estymacji orientacji oraz czujnika odległości zamiennie z barometrem. Wektor mierzonego przyśpieszenia liniowego zostaje obrócony zgodnie tak, aby współrzędne wektora były zdefiniowane we współrzędnych świata. Od osi z odejmowana jest wartość przyśpieszenia grawitacyjnego $g = 9.81\frac{m}{s}$. Dzięki temu uzyskiwane jest przyśpieszenie liniowe w pionie a_z. Równania zostały wyrażone jako 2.

$$x_k = Ax_{k-1} + Bu_{k-1}$$
$$y_k = Cx_{k-1}$$

$$\begin{bmatrix} \hat{z}_k \\ \hat{v}_{zk} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \hat{z}_k \\ \hat{v}_{z_k} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} \cdot \begin{bmatrix} a_z \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} z_k \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} \hat{z}_k \\ \hat{v}_{z_k} \end{bmatrix} \tag{2}$$

Częstotliwość pomiaru przyśpieszenie to 200 Hz, więc $\Delta t = 5$ms. Dzięki temu estymowana jest także prędkość w pionie wymagana do realizacji sterowania.

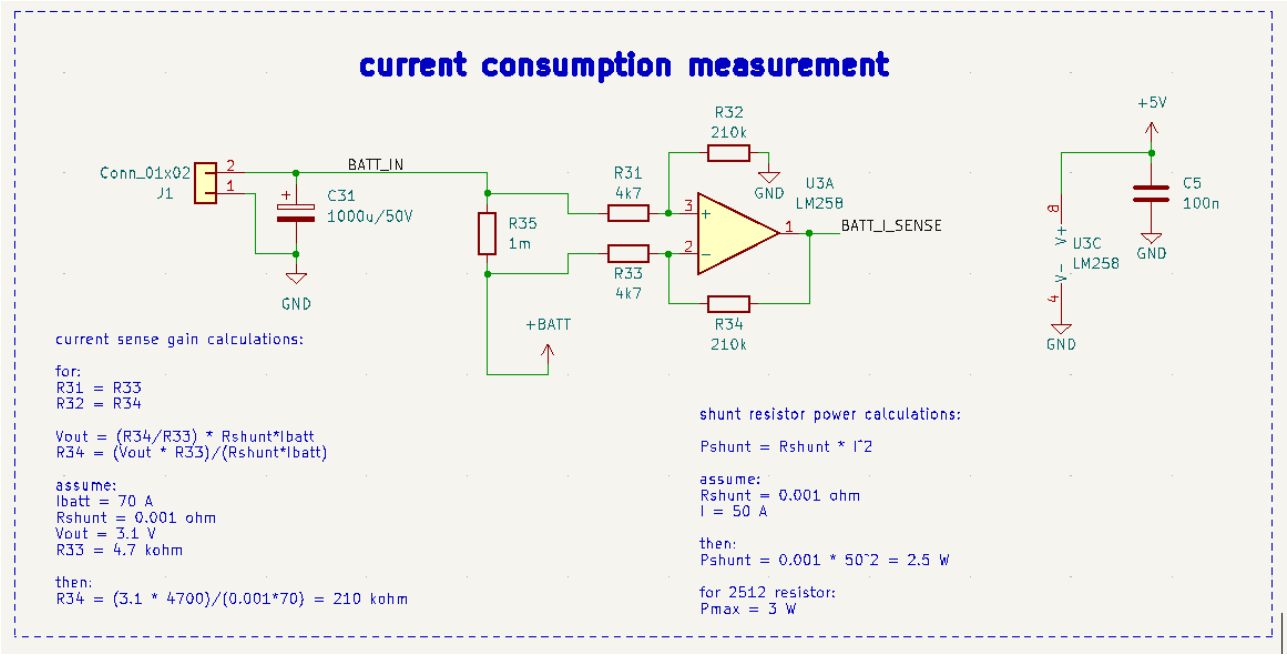
W przypdku braku danych z czujnika odległości (wylot po za zakres) jako wejście filtru podawane jest wartość ciśnienia atmosferycznego. W tym celu użyty jest wzór 3.

$$h = 44330 \cdot \left(1 - \left(\frac{p}{p_0}\right)^{0.1903}\right) \tag{3}$$

3.3 Pomiar napięcia baterii oraz pobieranego prądu

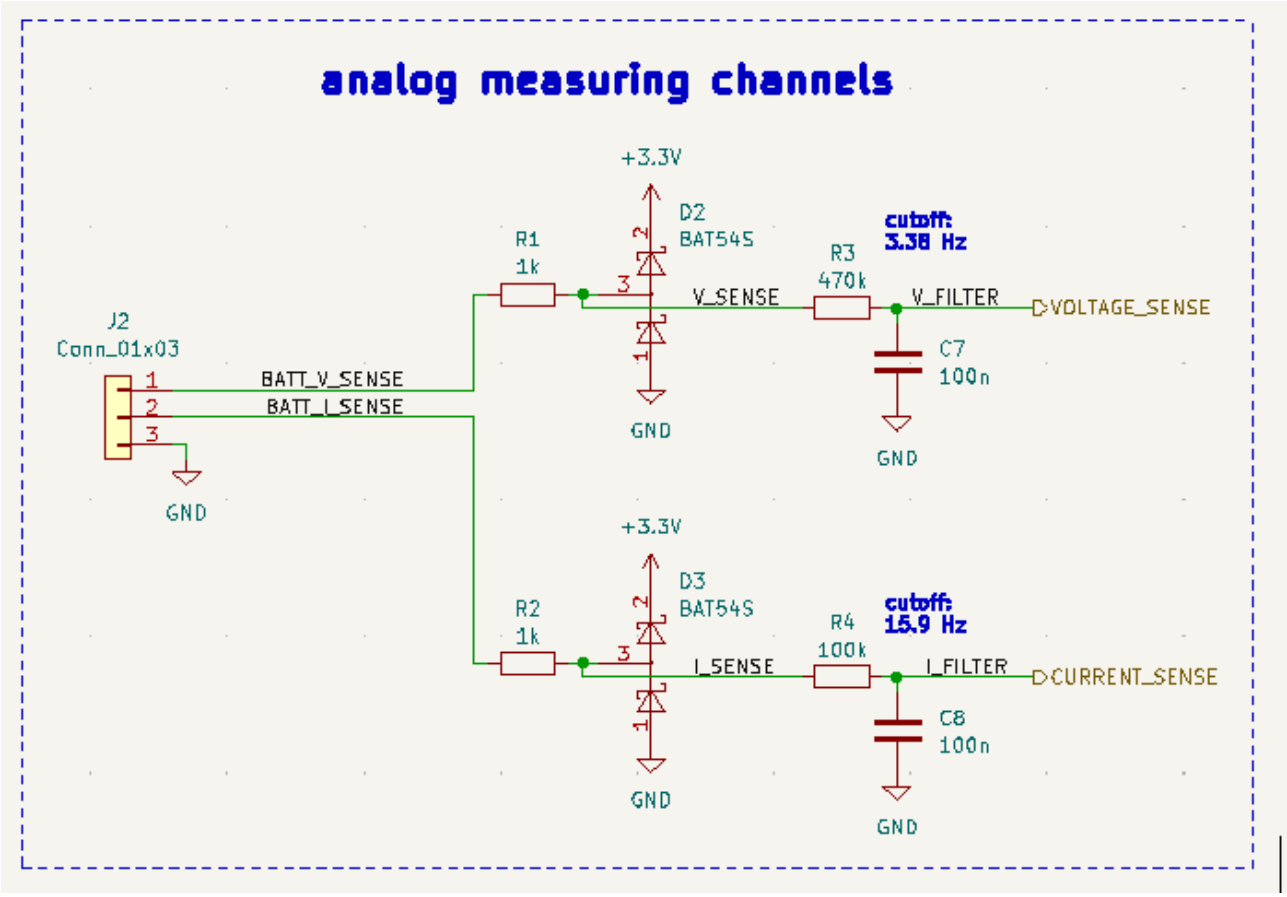
Pomiar napięcia oraz natężenia prądu płynącego z baterii został zrealizowany za pomocą przetwornika ADC w parze z DMA. Przetwornik został skonfigurowany tak, aby pomiar był ciągły. Dedykowany task w stałych interwałach czasowych odczytuje dane z bufora oraz wysyłający je do kolejki mierzonych danych. Pomiar napięcia realizowany jest z użyciem dzielnika napięcia. Filtracja i zabezpieczenie kanałów zostało zaprezentowane na 3.

Pomiar prądu realizowany jest poprzez użycie wzmacniacza operacyjnego wzmacniającego spadek napięcia na rezystorze pomiarowym tak jak przedstawiono na rysunku 2.



Rysunek 2: Schemat układu mierzącego prąd

Początkowo nie zostało sprawdzone czy wzmacniacz jest typu „rail to tail". Okazało się, że napięcie wyjściowe ograniczone jest od dołu napięciem 0.7 V co powoduje brak możliwości pomiaru małych prądów (poniżej 10 A). Niedopatrzenie to skutecznie utrudniło implementacje estymatora oraz uniemożliwiło jego skuteczne przetestowanie i dostrojenie.



Rysunek 3: Kanały pomiarowe przetwornika ADC, zabezpieczenia i filtry

3.4 Estymacja poziomu naładowania baterii

Na podstawie pracy [6] zostały wyprowadzone równanie stanu modelu baterii w postaci 5.

$$\hat{SOC}_k = \hat{SOC}_{k-1} - \frac{1}{C} \cdot I \tag{4}$$

$$V = V(SOC_k) \tag{5}$$

W celu wyznaczenia funkcji odwrotnej zostały użyte stablicowane punkty w pracy [6]. Niestety po uproszczeniu modelu wyłącznie do minimum oraz wpisaniu rzeczywistych parametrów baterii wartości zwracane przez estymator nie mają odzwierciedlenia w rzeczywistości. Spowodowane jest to nieprawidłowym pomiarem prądu, którego nie było możliwe wymuszenie sztuczne jak i testowe (silnik bez śmigła pobiera maksymalnie 1 A, zasilacze laboratoryjne i sztuczne obciążenia generują mniej mocy).

4 Sterowanie

4.1 Abstrakcyjna maszyna stanów

Na potrzeby projektu zostało stworzone osobne repozytorium git [2], które zostało dodane jako submodule do głównego repozytorium. Głównymi założeniami projektowymi biblioteki były:

- interfejs wspierający podejście obiektowe
- tranzycje generowane przez zdarzenia
- logika ukryta przed kodem użytkownika
- użycie wyłącznie statycznie alokowanej pamięci
- brak użycia biblioteki STL

Powyższe założenia zostały spełnione poprzez poniższe klasy pokazane w 8 6 oraz 7. Klasa „Event" służąca jako baza dla klas użytkownika służy do generacji tranzycji. Podczas konfigurowania głównej klasy „StateMachine" definiuje się jakie zdarzenia mają generować przejścia pomiędzy poszczególnymi klasami.

```
1  class Event {
2      bool is_triggered;
3
4      virtual void on_clear() {}
5
6  protected:
7      Event() : is_triggered{false} {
8
9      }
10
11     void trigger() {
12         is_triggered = true;
13     }
14
15 public:
16     bool isTriggered() const {
17         return is_triggered;
18     }
19
20     void clear() {
21         on_clear();
22         is_triggered = false;
23     }
24 };
```

Listing 6: klasa zdarzenia

Klasa „State” jest bazą wszystkich stanów definiowanych przez użytkownika. Odpowiednie nadpisania metod dają możliwość wykonywania akcji na początku, w trakcie jak i przy opuszczaniu stanu.

```
class State {
    virtual void enter() {}
    virtual void execute() {}
    virtual void exit() {}

    template <int S, int E>
    friend class StateMachine;
};
```

Listing 7: klasa stanu

Informacje na temat połączeń w grafie przejść oraz ich obsługa zawarte są w klasie „StateMachine”. Wywołuje ona odpowiednie metody stanów oraz dba o zmianę stanów.

```
template<int S, int E>
class StateMachine {
    struct StateTransition;

    struct Transition {
        Event *event;
        StateTransition *next;
    };

    struct StateTransition {
        State *state;
        Transition transitions[E];
    };

    StateTransition states[S];
    StateTransition *current;

    void clear();

public:
    StateMachine(State *st);
    void transit(State *from, State *to, Event *event);
    void start();
    void update();
};
```

Listing 8: klasa maszyny stanów

4.2 Regulator LQR

W celu realizacja zaproponowanego sterowania w postaci regulatora LQR została wyodrębniona osobna klasa (singleton). Dzięki rozbudowanemu zapleczu matematycznemu w postaci klasy „Matrix" impelmentacja była trywialna, tak jak widać w 9 i 10. Widać macierz przekopiowaną z symulacji w programie Simulink.

```
class Controller {
    static constexpr int X_NUM = 8;
    static constexpr int U_NUM = 5;

    static constexpr Matrix<U_NUM, 1> operating_point = {
        0.2527f,
        0.2527f,
        0.2527f,
        0.2527f,
        0.2229f
    };

    static constexpr Matrix<U_NUM, X_NUM> K = {
         1.4397f,    1.7109f,   -0.0000f,   -0.0000f,    1.2727f,
           -0.4991f,    0.0305f,    0.0479f,
         1.7109f,   -1.4397f,   -0.0000f,    1.2727f,   -0.0000f,
           -0.4991f,    0.0305f,    0.0479f,
        -1.4397f,   -1.7109f,   -0.0000f,    0.0000f,   -1.2727f,
          -0.4991f,    0.0305f,    0.0479f,
        -1.7109f,    1.4397f,   -0.0000f,   -1.2727f,    0.0000f,
          -0.4991f,    0.0305f,    0.0479f,
         0.0000f,   -0.0000f,    0.0000f,    0.0000f,   -0.0000f,
           0.0061f,    0.0998f,    0.1106f
    };

    Matrix<X_NUM, 1> setpoint;
    Matrix<X_NUM, 1> process_value;

    Controller();

public:
    Controller(Controller &) = delete;
  void operator=(const Controller &) = delete;

    void setSetpoint(const Matrix<X_NUM, 1> sp);
    void setProcessValue(const Matrix<X_NUM, 1> pv);
    Matrix<U_NUM, 1> calculate() const;

    Matrix<X_NUM, 1> getSetpoint() const;
    Matrix<X_NUM, 1> getProcessValue() const;

    static Controller& getInstance();
};
```

Listing 9: klasa regulatora

```
1 Matrix<Controller::U_NUM, 1> Controller::calculate() const {
2     const Matrix<X_NUM, 1> error = process_value - setpoint;
3     return operating_point - K*error;
4 }
```

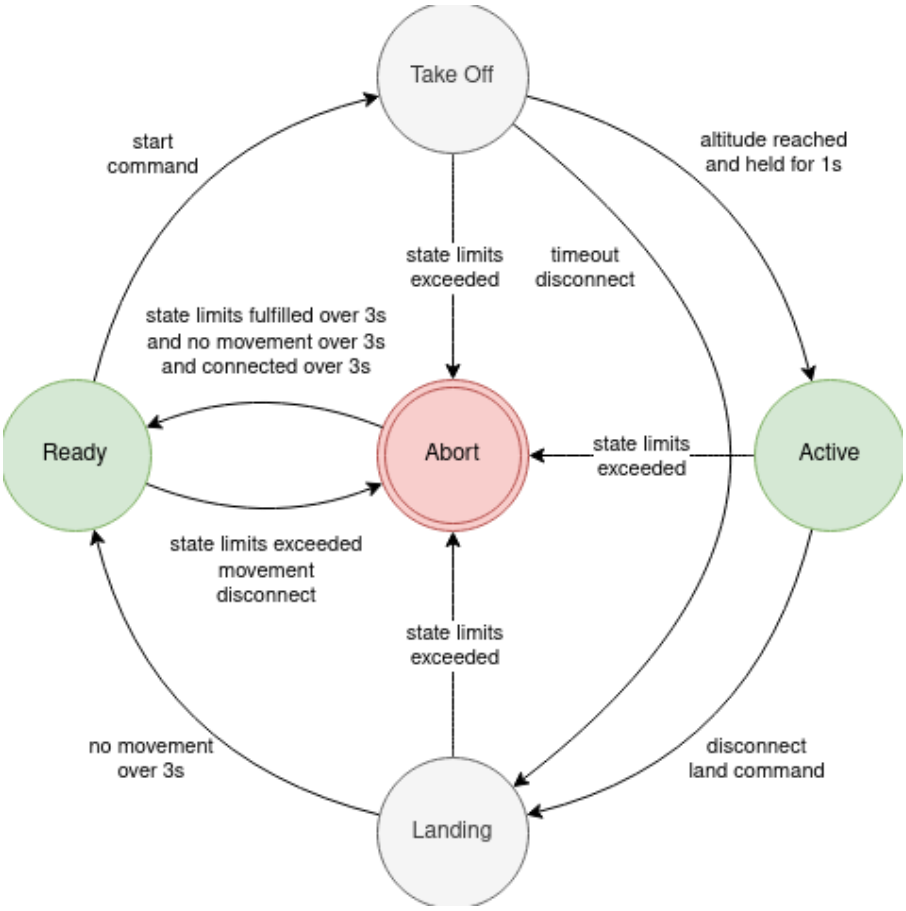
Listing 10: ciało regulatora

4.3 Sterowanie serwomechanizmami oraz ESC

Do sterowania serwomechanizamami oraz steronikiem silnika BLDC zostały użyte dwa układy czasowo-licznikowe w trybie generacji sygnału PWM. Każdy z pięciu sygnałów posiada częstotliwość 50 Hz.

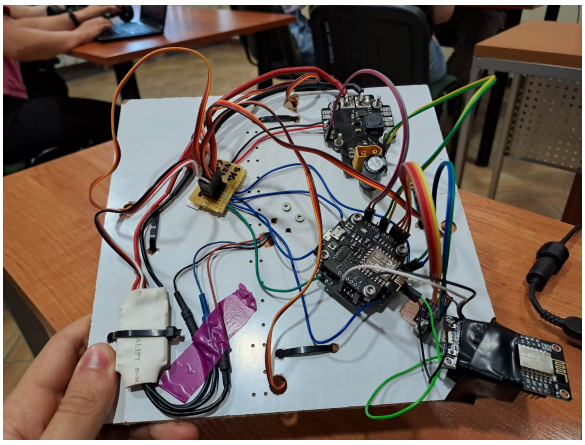
Zastosowane serwa to popularne SG90. Ich charakterystyczną cechą jest praca w niestandardych przedziałach wpełnienia (500 us do 2400 us).

Sterowanie w zakmniętej pętli przebiega z częstotliwością 100 Hz.

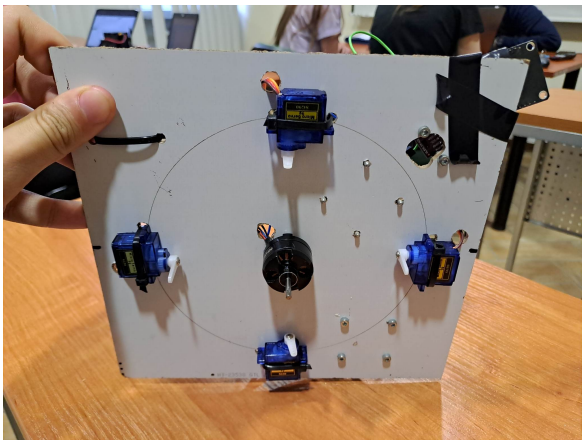


Rysunek 4: Zaimplementowany schemat stanów i przejść

4.4 Makieta



(a) widok z przodu

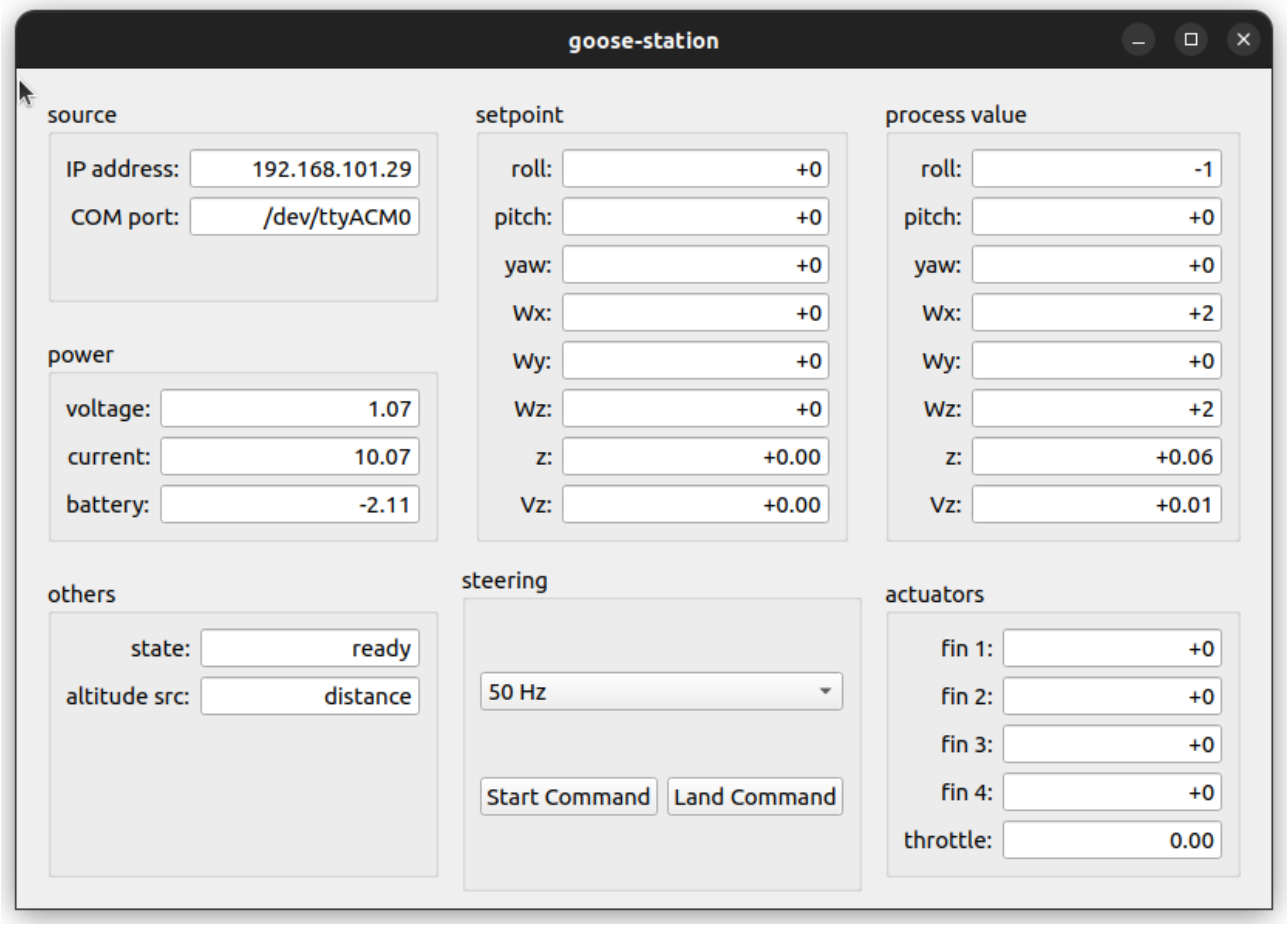


(b) widok z dołu, widoczny silnik oraz serwa

Rysunek 5: Makieta wykonana na potrzeby prezentacji

4.5 Aplikacja okienkowa

Aplikacja oparta o bibliotekę Qt znacznie ułatwiła rozwijanie oprogramowania oraz usuwania błędów na bieżąco. Potrafi zasymulować zerwanie połączenia z operatorem oraz sprawdzać przy jakiej częstotliwości zadanego stanu wykryje brak sygnału z pilota operatora.



Rysunek 6: Aplikacja stworzona do testów oprogramowania

5 Podsumowanie i wnioski

Projekt sterownika lotu jednowirnikowego drona sterowanego wektorem ciągu był wyzwaniem. Udało wypełnić się teoretycznie wszystkie założenia początkowe, lecz niektóre aspekty (takie jak estymacja poziomu baterii) wymagają dopracowania.

Elementem na którym należało poświęcić najwięcej czasu był estymator stanu oraz testy jego poprawnej pracy. Niekiedy problematyczne były błędy hardwareowe, które uniemożlwiały dalszy rozwój programu.

Bibliografia

[1] *Blackmagic Wireless SWD Debug probe hosted on esp-idf SDK (for ESP8266) with UART on Telnet port and HTTP using xterm.js*. URL: https://github.com/walmis/blackmagic-espidf.

[2] Eryk Możdżeń. *Simple finite state machine implementation for various C++ projects*. URL: https://github.com/Eryk-Mozdzen/state-machine-cpp.

[3] Eryk Możdżeń. *UAV TVC Goose*. URL: https://github.com/Eryk-Mozdzen/uav-tvc-goose.git.

[4] STMicroelectronics. *DS11555 VL53L0X Time-of-Flight ranging sensor*. 2022. URL: https://www.st.com/en/imaging-and-photonics-solutions/vl53l0x.html#documentation.

[5] STMicroelectronics. *Provides the USB Device library part of the STM32Cube MCU Component "middleware"for all STM32xx series*. URL: https://github.com/STMicroelectronics/stm32_mw_usb_device.

[6] Ying Zhang. *A Battery SOC Estimation Method Based on AFFRLS-EKF*. 2021. URL: https://www.researchgate.net/publication/354119837_A_Battery_SOC_Estimation_Method_Based_on_AFFRLS-EKF.