GJN

5.D
11.12.2022
Gymnázium Jana Nerudy

Stavba a Programování Dronu Dokumentace

Jan Vorel a Filip Hanzlík

Obsah

Zadání projektu

1.1 Motivace

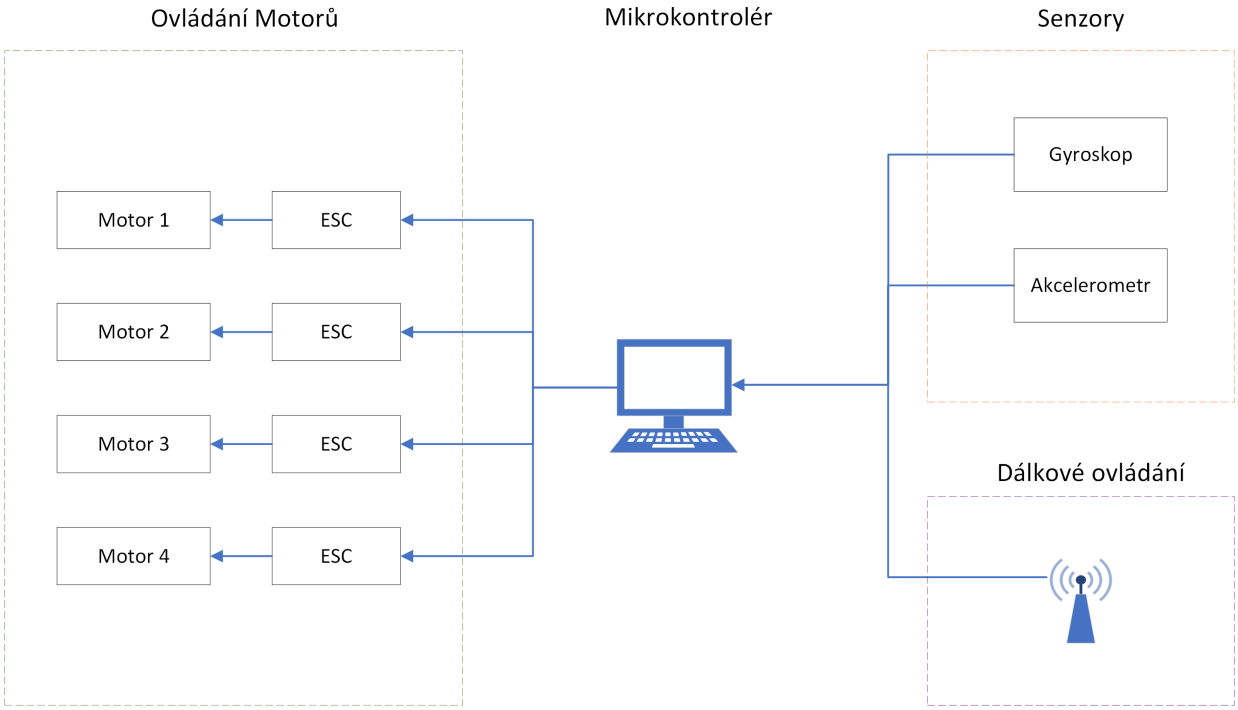
V dnešní době drony rapidně rostou na popularitě a jejich vývoj neustále přináší nové možnosti. Drony se tak stávají velmi atraktivní oblastí pro výzkum a to i na nižší úrovni, což je reflektováno i jejich častější přítomností v médiích. Co drony tvoří tak unikátními, je nejen výborná vzdušná mobilita a ovládání, ale především možnost nadstavby softwarem, jež dále výrazně rozšiřuje využití těchto strojů. Již aktuálně, díky implementaci systémů pro snímání obrazů, automatického předurčování trajektorie a vzájemné koordinace více dronů, je jejich uplatnění velmi široké, a to i v komerční sféře. Jedná se například o mapování a analyzování nedostupných teritorií, pořizování videozáznamů z těžko dostupných vzdušných úhlů, automatické doručování zásilek, aplikace v armádě a mnoho dalších. Jde tedy o technologii se zdánlivě velkým a rozvíjejícím se potenciálem.

1.2 Cíl projektu

Cílem projektu bylo, jak mu název napovídá, sestrojení letu schopného dronu, a to ze základních elektronických součástek a senzorů spolu s vytvořením softwaru pro jeho ovládání. Tento projekt tedy měl důraz na programovací aspekty práce s dronem, zahrnující zejména práci s ovládáním mikrokontroléru, sběr dat ze senzorů/přijímačů, patřičné ovládání motorů a správné zpracování všech těchto dat, pro jejich finální implementaci v algoritmu, který bude provádět korekce v letu dronu. Jako primární cíl tohoto projektu jsme si definovali sestrojení letu schopného dronu, jež bude moct být dálkově ovládán. Takto již sestavený dron by nadále mohl být základem pro vývoj dalších funkcí, které by pak už měli aplikovatelné využití. Jedná se například o instalaci kamery, z níž by mohla být zpracována data, která by dále sloužila k různým aplikacím, jako je například vytváření 3D modelů prozkoumaných oblastí, nebo detekci objektů, či lidí. Dále, vedle samotného vývoje dronu, chceme aby tento projekt sloužil ostatním zájemcům, například studentům naší školy, pro poznání toho jak takový dron funguje a co všechno obnáší jeho sestavení. Chtěli bychom proto, aby tato dokumentace mohla případně sloužit jako takový vstupný bod do tohoto oboru. Budeme tedy klást důraz na přehledný popis našeho projektu tak, aby co nejlépe informoval a doprovázel zájemce, začínající zkoumat fungování dronů. Pro splnění tohoto účelu jsou všechny napsané programy, vytvořené modely a dodatečné dokumenty věřejně k dispozici na Githubu pod tímto odkazem: https://github.com/FilipHanzlik/Stavba-Dronu.

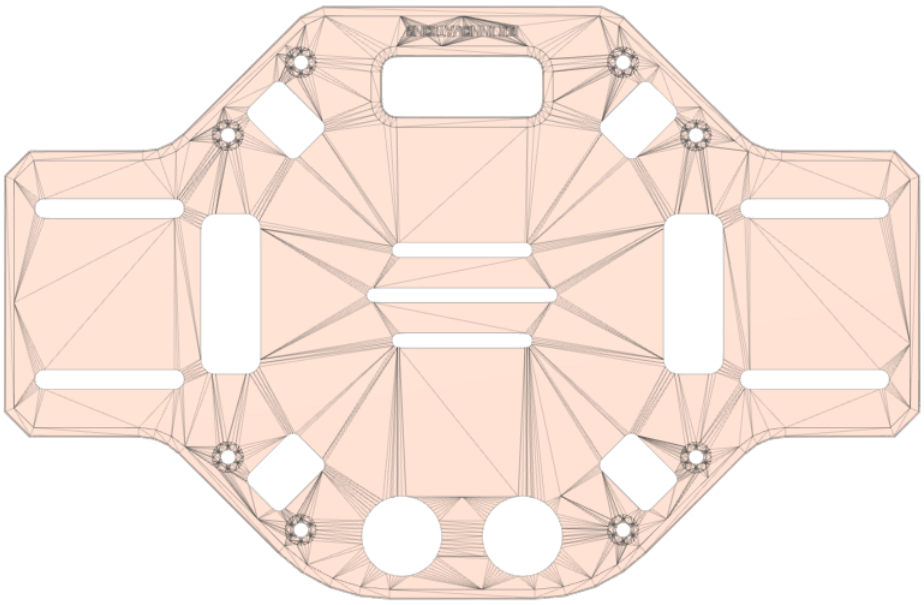
1.3 Úvod

Dron je dle definice bezpilotní letadlo, které může být řízeno na dálku nebo autonomně. V našem případě půjde pouze o dálkové řízení. Drony se obecně řadí mezi tzv. nestabilní systémy. To znamená, že potřebují aktivně korigovat svojí polohu. Tohoto je dosaženo pomocí mikrokontroléru, nejdůležitější součástí dronu. Mikrokontrolér dostává data o požadovaném stavu (orientace a náklon) z dálkového ovládání a poté je porovná s aktuálním stavem, který zjistí z přítomných senzorů. Pomocí algoritmu PID (tento algoritmus bude podrobně popsaný v sekci softwaru) mikrokontrolér určí potřebnou rychlost otáčení jednotlivých motorů a tyto instrukce poté rozešle.
Celé fungování dronu je možné vidět na schématu níže. ESC zde značí elektronické regulátory otáček, pomocí nichž může mikrokontrolér ovládat jednotlivé motory.

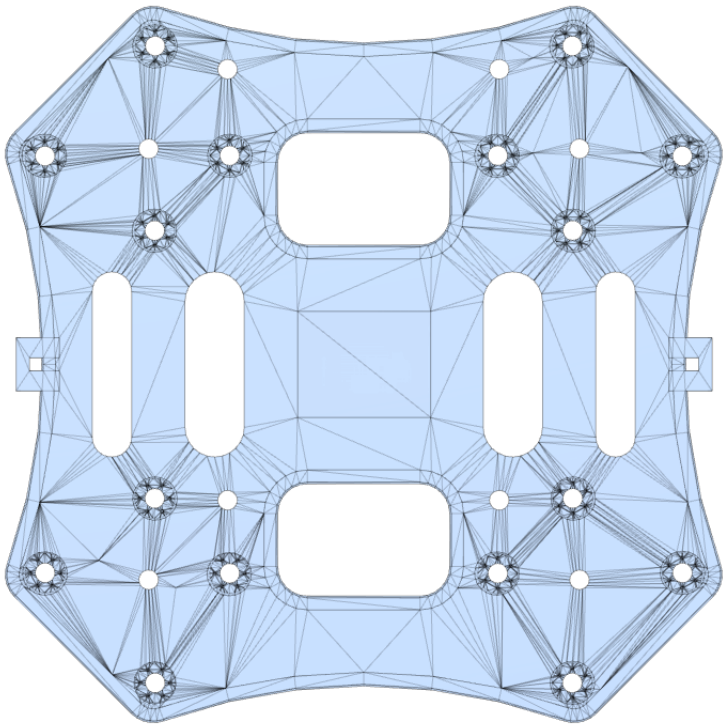


Konstrukce

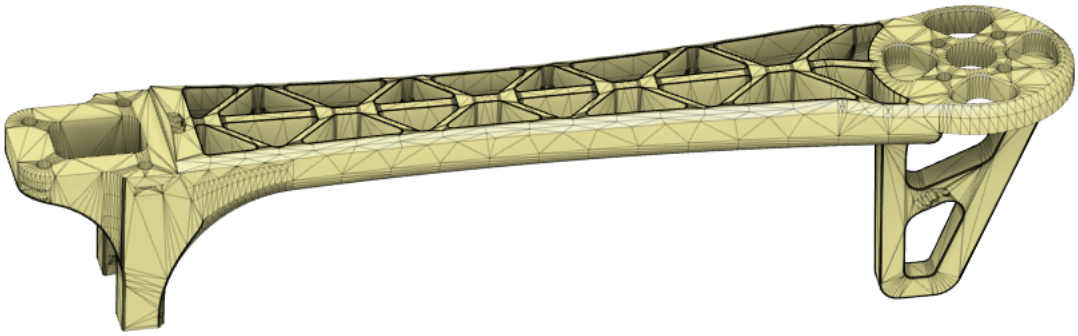
Celý tento systém, který je odpovědný za fungování dronu, potřebuje nosnou konstrukci. K tomuto jsme použili 3D modely, které jsme vytvořili na základě volně dostupných zdrojů z internetu. Tyto modely jsme pak vytiskli na 3D tiskárně z PETG. Celá konstrukce se tak skládá ze tří základních částí: z ramen, ze spodní desky a z horní desky.
Model spodní desky je uveden níže. Na délku měří něco málo přes 18 cm a jeho šířka činí přibližně 12 cm. Její rozměry jsou nastaveny tak, aby se na tuto desku spolehlivě vešla baterie spolu s případním senzorem pro měření aktuální výšky (spodní dvě kulaté díry slouží k umístění tohoto senzoru).



Dále z horní strany přilehne vrchní deska o rozměrech 10.7 x 10.7 cm. Tato deska kromě toho, že umožňuje držet celou konstrukci pohromadě, také slouží jako podpora pro mikrokontrolér a další elektroniku. Právě na této desce bude totiž přimontovaný mikrokontrolér, spolu s dalšími senzory včetně přijímače pro dálkové ovládání. Z tohoto místa tedy vychází veškeré řízení dronu. Model horní desky je opět přiložen níže.



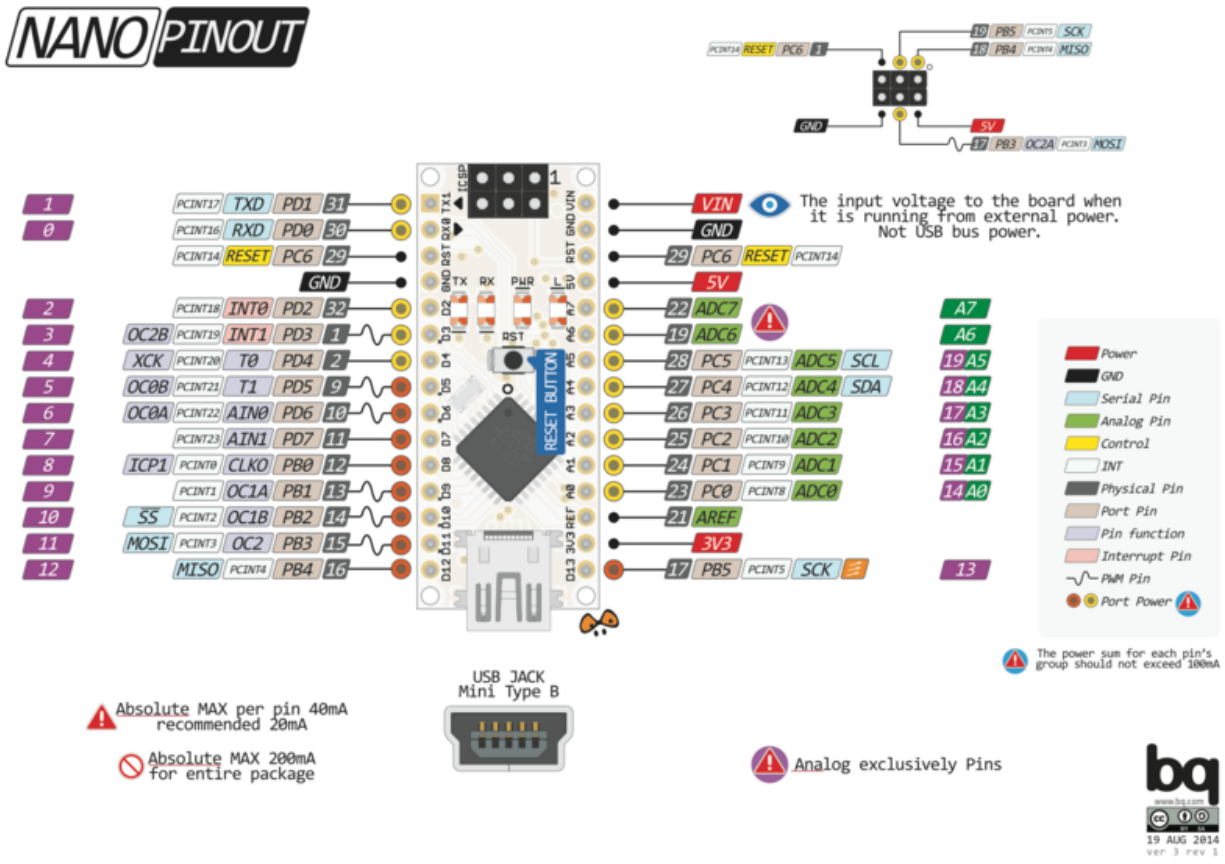
Poslední součástka, která je potřebná ve čtyřech kusech, je rameno. Ramena, s délkou 21.3 cm, jsou vždycky přišroubovaná mezi kraji horní a dolní desky. Jejich umístění je patrné z předpřipravených děr pro šrouby. Každé rameno pak na svém těle drží elektronický regulátor otáček (ESC), který je připojený na mikrokontrolér, přítomný na horní desce. Na regulátor je dál připojený motor. Motor je přišroubovaný na úplném konci ramene na patrně viditelném umístění. Model ramene je opět uveden níže.



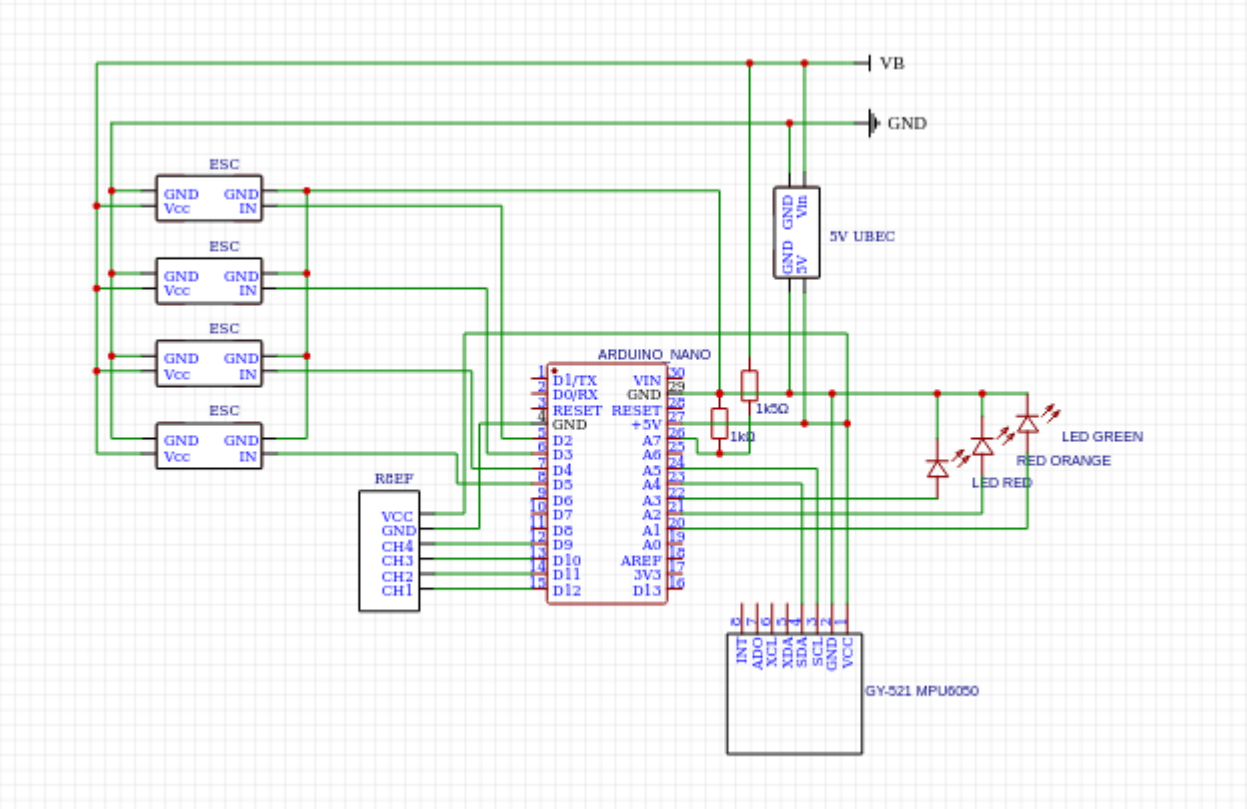
Elektronika

3.1 Elektrické zapojení

Jak již už bylo zmíněno, nejdůležitější částí pro elektroniku je mikrokontrolér, který nese veškerý psaný software. Jako mikrokontrolér jsme si vybrali Arduino Nano a to zejména pro jeho malou velikost, nízkou cenu a hlavně velmi obsáhlou dokumentaci. Další možností by bylo Raspberry Pi Pico, to ale nemá tak rozsáhlou a přehlednou dokumentaci, a tudíž vývoj dronu by byl o něco komplikovanější a delší. Zároveň Arduino nabízí své vlastní vývojové prostředí ve kterém jsme mohli software napsat. Zde je obrázek toho, jak vypadá Arduino Nano. Spolu s označením jednotlivých pinů.



K tomu aby program na Arduinu Nanu mohl zpracovávat data ze senzorů a dálkového ovládání pro následné ovládání motorů, je nejprve zapotřebí připájet všechny komponenty k mikrokontroléru. Na tomto elektronickém schéma je znázorněné zapojení všech komponentů na našem dronu. Oproti předchozímu popisu jsou na schématu navíc přítomná led světla, která slouží jako indikátory stavu dronu (například pro skoro vybitou baterku). Dále je zde použitý jeden regulátor napětí na 5V, jelikož port pro napojení na Arduinu vyžaduje 5V, a v poslední řadě jsou zde ještě přítomné dva rezistory, které opět pomáhají s regulací napětí.



Jak je ze schématu patrné, kromě mikrokontroléru jsou na dronu přítomné ještě čtyři další různé komponenty, plus motory. Začněme od baterie. Celý tento systém samozřejmě potřebuje napájení a jelikož motory dokáží spotřebovat velké množství energie, vybrali jsme relativně velkou 11.1V baterii o kapacitě 4500 mAh. Díky této baterii je možné létat s naším dronem po dobu přibližně patnácti minut.

Dalším komponentem je senzor GY-521 MPU-6050, který disponuje 3-osým akcelerometrem a gyroskopem, díky němuž bude Arduino moct zpracovávat data o aktuální změně náklonu a zrychlení dronu. Vedle gyroskopu a akcelerometru dostává Arduino ještě data z přijímače na dálkové ovládání. Jedná se o přijímač R8EF pro vysílač Radiolink T8FB. Přenos dat z tohoto přijímače do Arduina probíhá pomocí metody PWM (Pulse-width modulation), která umožňuje efektivně přenášet analogový signál.

Poté co Arduino dostane a zpracuje všechna data vyšle instrukce do elektronických regulátorů otáček. V našem případě používáme jako elektronické regulátory otáček Chaos 20A. Každý tento regulátor pak obdržené instrukce zpracuje a pošle je dál motoru. Jako motory používáme bezkartáčové stejnosměrné motory XA2212/1400, které dokáží vyvinout víc než 10000 otáček za vteřinu.

3.2 Software

Veškerý software odpovědný za ovládání dronu je napsaný v Arduino IDE, který využívá jazyk velmi podobný C++. Tento software bychom mohli rozdělit na dvě hlavní části:

1. Konfigurace hardwaru
2. Hlavní cyklus, který se neustále opakuje

3.2.1 Konfigurace Hardwaru

V této části programu dochází k nastavení stavu všech pinů na arduinu k tomu, aby byly schopný správně komunikovat s jednotlivými komponenty dronu. Poté dojde k nastavení počátečních parametrů pro komunikaci mezi komponentami a arduinem. K této komunikaci využíváme protokolu I2C. V případném zájmu o lepší pochopení této problematiky doporučujeme shlédnout toto video: https://www.youtube.com/watch?v=6IAkYpmA1DQ.

Nastavení pinů

Prvním krokem při konfiguraci hardwaru je správné nastavení pinů. V našem případě potřebujeme nastavit piny, které budou ovládat elektronické regulátory otáček a led světla, a piny, přes které budeme dostávat data z dálkového ovládání. Začněme teda od pinů pro ovládání motorů a led světel. K nastavení těchto pinů je pouze potřeba je přepnout do write módu. Piny na něž jsme připojili LED žárovky jsou k nalezení v registru DDRC pod DDC1, DDC2 a DDC3.

13.4.6 DDRC – The Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

A další čtyři piny, které potřebujeme pro ovládání motorů, můžeme najít v registru DDRD pod DDD2, DDD3, DDD4 a DDD5.

13.4.9 DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Pro přepnutí těchto pinů do write módu, tak stačí pouze přepnout uvedené hodnoty v registrech z 0 na 1.

```
DDRC |= B00001110; // LED_green = 15, LED_yellow = 16, LED_red = 17;
DDRD |= B00111100;
```

Nyní nám už zbývá pouze nastavit piny, které jsou spojené s přijímačem na dálkové ovládání. Při každém vyslání informace z přijímače pro dálkové ovládání dojde k zastavení hlavního cyklu, aby se mohla nová data uložit. Tomuto efektu se říká interrupt a při každém zavolání interuptu se spustí námi definovaná funkce, která se postará o uložení právě obdržených dat. V našem programu využíváme masku PCMSK0, která aktivuje piny PCINT0 až PCINT7. Níže je přiložen výstřižek z manuálu ATmega328P (mikročip, který Arduino Nano používá).

12.2.8 PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..0 – PCINT7..0: Pin Change Enable Mask 7..0**

Each PCINT7..0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.

Jelikož jsme přijímač na dálkové ovládání připojili na piny PCINT1 až PCINT4, budou to právě tyto piny na něž nastavíme interrupty v programu.

```
PCMSK0 |= B00000010;
PCMSK0 |= B00000100;
PCMSK0 |= B00001000;
PCMSK0 |= B00010000;
```

Dojde-li tedy k aktivaci interruptu, spustí se funkce, která se postará o zpracování dat z dálkového ovládání. Přijímač dálkového ovládání vysílá signál podobný Pulse-Width modulation (PWM). S frekvencí 50 Hz vysílá signál dlouhý 1000-2000 us, kde délka signálu reprezentuje to jak moc uživatel naklonil páčku na dálkovém ovládání. Tyto hodnoty pak uloží do seznamu *channel_vol*.

Program u každého pinu postupuje následujícím způsobem:

1. Zkontroluje jestli pin byl právě aktivován porovnáním hodnoty pinu a proměnné D#_state (D#_state zaznamenává, zda-li signál už začal), případně uloží počáteční čas signálu.

2. Nebo zkontroluje jestli signál byl ukončen, zrcadlově oproti kroku 1. Délka signálu je určena odečtením času ukončení od počátečního času, a proměnná D#_state je nastavena zpátky na false, aby mohl být změřen příští signál. Délka signálu je uložena do seznamu channel_vol pro budoucí využití.

```
ISR(PCINT0_vect) {
  current_time = micros();
  if (!D9_state && PINB & B00000010) {
    D9_state = true;
    timer_2 = current_time;
  } else if (D9_state && !(PINB & B00000010)) {
    D9_state = false;
    channel_vol[1] = current_time - timer_2;
  }
  if (!D10_state && PINB & B00000100) {
    D10_state = true;
    timer_3 = current_time;
  } else if (D10_state && !(PINB & B00000100)) {
    D10_state = false;
    channel_vol[0] = current_time - timer_3;
  }
  if (!D11_state && PINB & B00001000) {
    D11_state = true;
    timer_4 = current_time;
  } else if (D11_state && !(PINB & B00001000)) {
    D11_state = false;
    channel_vol[2] = current_time - timer_4;
  }
  if (!D8_state && PINB & B00010000) {
    D8_state = true;
    timer_1 = current_time;
  } else if (D8_state && !(PINB & B00010000)) {
    D8_state = false;
    channel_vol[3] = current_time - timer_1;
  }

}
```

I^2C

Nyní nám zbývá pouze nastavit konfiguraci pro komunikační protokol I2C. Toto zajišťuje tato část programu.

```
Wire.begin();
TWBR = 12;
Wire.beginTransmission(MPU);
Wire.write(0x6B);
Wire.write(0x00);
Wire.endTransmission();

Wire.beginTransmission(MPU);
Wire.write(0x1B);
Wire.write(0x08);
Wire.endTransmission();

Wire.beginTransmission(MPU);
Wire.write(0x1A);
Wire.write(0x03);
Wire.endTransmission();
```

První řádek *Wire.begin()* zahájí komunikační protokol a poté $TWBR = 12$ nastaví frekvenci komunikace na 400kHz. MPU je v programu proměnná, která reprezentuje adresu senzoru obsahující gyroskop a akcelerometr. Nejprve je nastavení senzoru resetováno a poté se pro gyroskop nastaví rozsah měření na $\pm$ 500 stupňů za vteřinu. Zde je výstřižek z dokumentace pro konfiguraci gyroskopu, která je k nalezení na tomto odkaze: (https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf)

checking the Data Ready interrupt.

Each 16-bit gyroscope measurement has a full scale defined in *FS_SEL* (Register 27). For each full scale setting, the gyroscopes' sensitivity per LSB in *GYRO_xOUT* is shown in the table below:

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

V poslední fázi je ještě potřeba nastavit filtrování dat, abychom snížili šum z měřících přístrojů. Sensor má několik předem zpracovaných filtrů, které můžeme využít. V našem případě použijeme filtr na frekvenci 42 Hz, protože je blízko k rotační rychlosti motorů.

The DLPF is configured by *DLPF_CFG*. The accelerometer and gyroscope are filtered according to the value of *DLPF_CFG* as shown in the table below.

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

Tímto jsou všechny potřebné konfigurace připraveny a je možné již začít psát samotný program pro ovládání dronu. V případě zájmu o podrobnější pochopení těchto konfigurací, doporučujeme se kouknout na dokumentaci, která byla zmíněna v průběhu této sekce. Tato dokumentace obsahuje podrobně popsané funkce jednotlivých registrů, které by tak bylo zbytečné a zdlouhavé tady znovu rozebírat.

3.2.2 Hlavní cyklus

Jak již už bylo zmíněno, nejpodstatnější část pro ovládání dronu probíhá v hlavním cyklu, který se neustále opakuje. Princip tohoto cyklu je následující:

1. Program zpracuje data z akcelerometru, gyroskopu a dálkového ovládání
2. Tyto data poté pošle do PID algoritmu
3. Na základě výsledku PID připravý instrukce pro ESC a vyšle je

Zpracování dat

Již v předcházející části programu jsme z dálkového ovládání obdrželi délky signálů (1000 us - 2000 us) uložené do seznamu *channel_vol* (V této části kódu používáme seznam *channel*, který je jenom kopií seznamu *channel_vol*). Hodnoty v seznamu reprezentují ve stejném pořadí následující veličiny: požadovanou rychlost otáčení motorů a požadovanou rotaci kolem jednotlivých os dronu (podél vertikální osy, horizontální osy jdoucí od zadní do přední části dronu a vertikální osy kolmé k nám). Potřebujeme proto převést tyto délky signálu na požadované hodnoty. Tyto nové hodnoty zapíše následující program do seznamu *desRate*.

```
void convert_input(){
  for (int i = 0; i < 2; i++){ // Upravení PWM pokynů z vysílače na rychlost změny náklonu
    desRate[i] = 0;
    if (channel[i+2] > 1505)desRate[i] = (channel[i+2] - 1505)/6;
    else if (channel[i+2] < 1495)desRate[i] = (channel[i+2] - 1495)/6;
  }
  desRate[2] = 0;
  if (channel[1] > 1505)desRate[2] = (channel[1] - 1505)/3;
  else if (channel[1] < 1495)desRate[2] = (channel[1] - 1495)/3;

  if (channel[0] < 1200)desRate[2] = 0;

  if (channel[0] > 1800)channel[0] = 1800;
}
```

Pro konvertování rotací kolem os, program určí nové hodnoty podle tohoto postupu:

- Jestliže $x < 1495$, pak nová hodnota $= (x - 1495)/6$
- Jestliže $x > 1505$, pak nová hodnota $= (x - 1505)/6$

Toto neplatí jedině pro rotaci kolem vertikální osy, kde dělíme trojkou (tato korespondují ke citlivosti ovládání - čím vyšší číslo, tím nižší citlovost). Hodnotu rychlosti otáčení motoru nemusíme konvertovat, jelikož 1000 bude stále značit žádné otáčení motorů a 2000 plné otáčky. Nastavili jsme akorát maximální hodnotu pro rychlost otáčení motorů na 1800 aby nedošlo k přehřátí motorů.

Další data jež přijímáme jsou data ze senzoru pro gyroskop a akcelerometr. Aktuálně pro stabilizaci potřebujeme jenom data z gyroskopu. Tyto data dostaneme přes I^2C protokol v registru 0x43. Jelikož hodnota o změně úhlu přichází pod 16-bitovým formátem, může se toto číslo pohybovat v intervalu $[-2^{15}; 2^{15}]$. Tudíž pro převedení hodnoty na stupně za vteřinu je potřeba získanou hodnotu vydělit číslem 65.5. Zde je opět ukázka implementace.

```
void read_sensors(){
  Wire.beginTransmission(MPU);
  Wire.write(0x43); // Data z GyroLatestskopu získány z registru 0x43
  Wire.endTransmission(false);
  Wire.requestFrom(MPU, 6, true); // Read 6 registers total, each axis value is stored in 2 registers
  for (int i = 0; i < 3; i++){
    GyroLatest[i] = (Wire.read() << 8 | Wire.read()) / 65.5 + GyroError[i];
  }
}
```

PID Algoritmus

Nyní jsme se ocitli ve fázi, kde musíme data zpracovat tak, abychom mohli poslat korigující signál do motorů. PID algoritmus je velmi široce využívan díky jednoduchosti jeho implementace. Výsledná hodnota PID algoritmu se skládá ze tří složek, proporcionální, integrační a derivační (podle jejich iniciálů je utvořen název). Každá z nich zpracovává chybu v rotaci dronu oproti požadovaným hodnotám.

Proporcionální složka je nejjednodušší na implementaci. Chyba je vynásobena koeficientem a poslána do motorů. Pokud vnější působící síla překonává sílu kterou proporcionální část algoritmu vydává na základě aktuální chyby, může nastat situace, kdy systém není schopen snižovat chybu. K tomuto problému dochází jelikož proporcionální část algoritmu nedokáže kompenzovat vnější síly, ale pouze využívá vzdálenosti od referenční hodnoty.

Potřebujeme tedy složku schopnou započítat vliv těchto vnějších sil. Tohoto dosáhneme sumací celkové chyby (integrací). Pokud se chyba nesnižuje, integrační složka algoritmu nabývá na hodnotě, což způsobuje zvýšení výkonu systému, který tak dosáhne dostatečné síly na překonání dříve zmíněných vnějších sil.

Poslední složka algoritmu se snaží předejít nadměrné korekci chyby, což by skončilo oscilací kolem referenční hodnoty, která by mohla zabránit přesnému dosažení referenční hodnoty. Zkoumáme tedy velikost změny v chybě (diferenciace). Je třeba zmínit, že tato část je velmi citlivá na šum v signálu, a proto buď musí používat filtrovaná data, anebo být odstraněna úplně (Potom by se ovládací proces nazýval pouze PI).

Ukázka programu níže, počítá PID hodnoty pro korekci rychlosti rotace kolem vertikální osy. Úplně analogicky je implementovaný algoritmus pro zbylé osy. Konečné hodnoty pro všechny tři osy budu nakonec uloženy v jednom seznamu, z něhož se pak dále využijí. Kp, Ki a Kd jsou zde koeficienty složek, které jsme experimentálně určili za nejvhodnější. PIDMAX je nejvyšší hodnota kterou může integrační složka dosáhnout, aby nedošlo k překompenzaci a oscilacím.

```
float calculate_yaw_rate_PID(float Rate, float desRate){
  const float Kp = kpyaw;
  const float Ki = kiyaw;
  const float Kd = kdyaw;
  float Integral, Error, previousError, PID;

  Error = desRate - Rate;
  Integral += Error;

  if (Integral > PIDMAX)Integral = PIDMAX;
  else if (Integral < -PIDMAX)Integral = -PIDMAX;

  PID = Kp * Error + Ki * Integral + Kd * (Error - previousError);
  previousError = Error;

  if (PID > PIDMAX)PID = PIDMAX;
  else if (PID < -PIDMAX)PID = -PIDMAX;

  return PID;
}
```

Ovládání motorů

Nyní už jsou všechna data zpracována a víme tak k jaké změně stavu dronu chceme dojít. Potřebujeme nyní tyto informace převést na instrukce pro motory, respektivě elektronické regulátory otáček. Ty opět přijímají signál s délkou od 1000 us do 2000 us. Kdy při 1000 us nedochází k otáčení motoru a při 2000 us je výkon motoru nastaven na maximum. Pro správné rozdělení signálů mezi motory musíme ale nejdřív použít tzv. Motor mixing algoritmus. Pro případ, že pracujeme s rozložením typu "+", jako je našim případem, je Motor mixing algoritmus založen na těchto principech:

- Pro rotaci dronu kolem vertikální osy je potřeba, aby motory podél jedné horizontální osy se točily rychleji než motory na druhé diagonále.
- Pro rotaci kolem horizontální osy, která je kolmá na nás, je potřeba aby se zadní motor točil rychleji než ten přední. Případně aby se přední motor točil rychleji než zadní, záleží zde na žádaném směru rotace.
- Pro rotaci kolem poslední osy je potřeba, aby se pravý motor točil rychleji než ten na levé straně. Nebo opět opačně pro opačný směr rotace.

Nám už tedy stačí vyjádřit rychlost rotace jednotlivých motorů a poté dosadit hodnoty obdržené z PID algoritmu. Tímto získáme délku signálu, který se později vyšle do motorů. Tyto hodnoty mezitím uložíme do seznamu nazvaném *ESCCommand*. Zde je ukázaná naše impementace *Motor mixing* algoritmu.

```
void motor_mixing_algo(){
  ESCCommand[0] = channel[0] - ratePID[0] + ratePID[2];
  ESCCommand[1] = channel[0] - ratePID[1] - ratePID[2];
  ESCCommand[2] = channel[0] + ratePID[0] + ratePID[2];
  ESCCommand[3] = channel[0] + ratePID[1] - ratePID[2];

  for (int i = 0; i < 4; i++){
    if (batteryVoltage < 1240 && batteryVoltage > 1000)ESCCommand[i] += ESCCommand[i] * ((1240 - batteryVoltage)/3600.0);
    if (ESCCommand[i] < 1100)ESCCommand[i] = 1100;
  }
}
```

Nyní už stačí vyslat samotné signály do elektrických regulátorů otáček. Toho docílíme jednoduchým způsobem. Zpočátku se začne vysílat signál do všech elektronických regulátorů otáček a zároveň se uloží aktuální čas. Nyní dokud nebudou všechny všechny piny, navázané na elektronické regulátory otáček, nastavené na nulu (znamenajíc, že už se žádný signál nevysílají), tak program bude neustále pro každý regulátor otáček kontrolovat jestli platí tato podmínka: $počáteční\_čas + požadovaná\_délka\_signálu <= aktuální\_čas$. Jestli ano, ukončí vysílání signálu do daného regulátoru otáček.

```
PORTD |= B00111100;
for (int i = 0; i < 4; i++)ESCTimer[i] = ESCCommand[i] + currentTime;



while (PORTD != B00000000){
  generalTimer = micros();
  if(ESCTimer[0] <= generalTimer)PORTD &= B11111011;
  if(ESCTimer[1] <= generalTimer)PORTD &= B11110111;
  if(ESCTimer[2] <= generalTimer)PORTD &= B11101111;
  if(ESCTimer[3] <= generalTimer)PORTD &= B11011111;
}
```

A zde program v základním cyklu končí a cyklus se znovu opakuje.

Závěr

Pro shrnutí bychom řekli, že se nám, sice s časovým zpožděním, povedlo úspěšně bez předešlých znalostí zkonstruovat funkční dron. Vidíme to jako důkaz toho, že je to problematika, jež je přístupná i pro studenty na naši úrovni a chceme touto dokumentací tak případným zájemcům pomoct lépe poznat programování hardwaru a fungování dronů. Kompletní verze programu, která už byla úspěšně otestována, bude spolu s dalšími podklady veřejně k dispozici na Githubu pod tímto odkazem: https://github.com/FilipHanzlik/Stavba-Dronu.V případě dotazů nás neváhejte kontaktovat přes Teams (xhanf@gmail.com).