Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Algoritmy strojového učení pro predikci parametrů letu dronu

Bakalářská práce

Martin Skok

Vedoucí práce: doc. Ing. Ivo Bukovský, Ph.D.

České Budějovice 2025

Bibliografické údaje

Skok, M., 2025: Algoritmy strojového učení pro predikci parametrů letu dronu. [Machine Learning Algorithms for Drone Flight Parameter Prediction. Bc. Thesis, in Czech.] – 71 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Cílem této bakalářské práce je vyvinout systém pro predikci parametrů letu malého dronu využívající algoritmy strojového učení. Práce zahrnuje sběr experimentálních dat, analýzu těchto dat, vývoj modelů strojového učení a jejich vyhodnocení. Výstupy práce zahrnují datový soubor z experimentálních měření, implementované algoritmy pro výpočet parametrů dronu, trénované a testované modely strojového učení a analýzu výsledků.

Annotation

The aim of this bachelor's thesis is to develop a system for predicting the flight parameters of a small drone using machine learning algorithms. The work includes collecting experimental data, analyzing this data, developing machine learning models and evaluating them. The outputs of the work include a dataset from experimental measurements, implemented algorithms for calculating drone parameters, trained and tested machine learning models and analysis of the results.

Prohlášení

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

Dále prohlašuji, že jsem nástroje generativní umělé inteligence, konkrétně ChatGPT, využil v souladu s akademickou etikou za účelem kontroly gramatiky a stylistiky, návrhů alternativních formulací textu a generování návrhů struktury některých částí práce.
Veškerý obsah vytvořený s pomocí těchto nástrojů jsem důkladně zkontroloval a upravil, a přebírám za něj plnou odpovědnost.

V Českých Budějovicích dne _____ *Podpis:* _____

Poděkování

Na tomto místě bych rád poděkoval vedoucímu své bakalářské práce doc. Ing. Ivo Bukovskému, Ph.D. za cenné rady, odborné vedení a podporu při zpracování tohoto tématu.
Dále děkuji své rodině a blízkým za trpělivost, podporu a motivaci po celou dobu studia.

Obsah

Seznam zkratek

u	Napětí baterie dronu
$u(t)$	Časová funkce napětí baterie v čase t
PWM	Pulse Width Modulation (pulzně šířková modulace)
m	PWM signál z motorů dronu
$m(t)$	Časová funkce PWM signálu v čase t
v	Počet kroků autoregrese – kolikrát se predikce rekurzivně opakuje
MAE	Mean Absolute Error (střední absolutní chyba)
MSE	Mean Squared Error (střední kvadratická chyba)
R^2	Koeficient determinace
MLP	Multilayer Perceptron (vícevrstvý perceptron)
ReLU	Rectified Linear Unit (nelineární aktivační funkce)
tanh	Hyperbolická tangens (nelineární aktivační funkce)
GD	Gradient Descent (metoda strmého spádu)
SGD	Stochastic Gradient Descent (stochastická verze GD)
Adam	Adaptive Moment Estimation (optimalizační algoritmus)
lr	Learning Rate – velikost kroku při učení modelu
Batch size	Počet vzorků z datasetu zpracovaných v jedné iteraci
Target ($\hat{y}$)	Cílová hodnota z datasetu, ke které se model snaží přiblížit
σ	Obecné označení pro aktivační funkci
h	Počet neuronů ve skryté vrstvě
n	Velikost vstupního okna modelu
$\mathbf{X}$	Vstupní vektor modelu
$\tau(t)$	Zbývající doba letu v čase t
$\Delta t = 100$	Vzorkovací interval
T	Celkový čas letu

Kapitola 1

Úvod

V posledním desetiletí jsme svědky výrazného rozmachu bezpilotních létajících prostředků, obecně známých jako drony. Tyto technologie, původně vyvíjené zejména pro vojenské účely, si dnes nacházejí široké uplatnění i v civilním sektoru. Drony se staly nepostradatelnými nástroji ve stavebnictví, zemědělství, ale i ve výzkumu, záchranných operacích a volnočasových aktivitách. S rostoucím využíváním těchto zařízení narůstá i důraz na jejich autonomii, bezpečnost a efektivitu.
Jedním z klíčových faktorů ovlivňujících provozuschopnost dronů je výdrž baterie, která přímo určuje maximální dobu letu. Včasný a přesný odhad zbývající doby letu je tedy zásadní nejen z praktického hlediska, ale především z hlediska bezpečnosti. Selhání baterie během letu může vést k havárii zařízení, ohrožení lidí nebo ztrátě cenných dat. Spolehlivá predikce doby letu tak představuje důležitý nástroj pro plánování misí i pro autonomní rozhodování samotného dronu.
Vzhledem k tomu, že vybíjení baterie není lineární a je ovlivněno řadou proměnných, jeví se jako vhodný přístup využití metod strojového učení. Tyto algoritmy umožňují odhalit složité nelineární vztahy mezi vstupními parametry a cílovou veličinou, kterou je v tomto případě zbývající čas do vybití baterie.
Tato bakalářská práce se zaměřuje na návrh, implementaci a vyhodnocení různých modelů strojového učení, které jsou schopné predikovat parametry letu malého kvadrokoptérového dronu na základě senzorických dat získaných během letu. Hlavní pozornost je věnována predikci zbývající doby letu pomocí různých přístupů. Experimentální část práce zahrnuje sběr a zpracování dat, návrh datasetů, trénink modelů, a jejich srovnání podle přesnosti predikce.
Výsledkem je funkční predikční systém, který lze implementovat do provozního softwaru dronu a který dokáže v reálném čase odhadnout, kolik času zařízení zbývá do úplného vybití.

1.0.1 Cíle práce

Hlavním cílem této bakalářské práce je navrhnout, implementovat a otestovat modely strojového učení pro predikci parametrů letu malého kvadrokoptérového dronu, se zaměřením na odhad zbývající doby letu. Predikce je založena na reálných senzorických datech zaznamenaných během letu, konkrétně na časových řadách napětí baterie a řídicích PWM signálech jednotlivých motorů.
Specifikem této práce je, že predikce není prováděna jednotným způsobem. Jsou zde testovány a porovnávány tři různé přístupy k predikci. Konkrétně

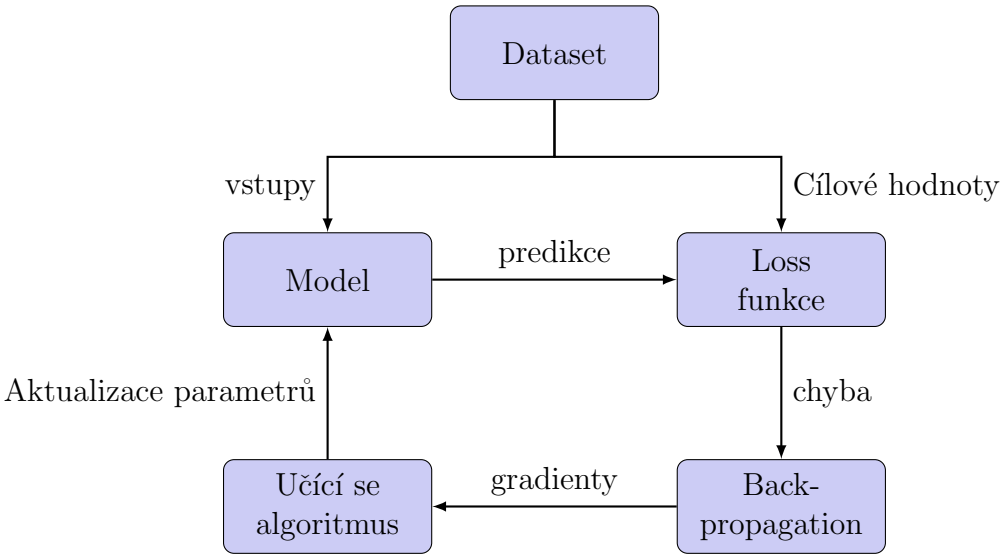
1. predikce zbývající doby letu pouze na základě napětí baterie,
2. predikce zbývající doby letu na základě napětí a PWM signálů motorů,
3. predikce budoucího průběhu křivky napětí s využitím autoregrese.

Každý z těchto přístupů je testován ve spojení s třemi typy modelů strojového učení. Cílem je nalézt kombinaci predikčního přístupu a modelu, která poskytuje nejvyšší přesnost a zároveň stabilitu predikce.
Dílčí cíle práce zahrnují

- **návrh a realizaci měřicího systému** pro záznam letových dat malého kvadrokoptérového dronu včetně řízení letu a logování dat pomocí Python knihovny `cflib`,
- **získání a zpracování dat** ze série experimentálních letů pokrývajících různé scénáře od statického vznášení až po dynamický pohyb v prostoru,
- **vytvoření datasetů** vhodných pro trénink a testování modelů pomocí techniky plovoucího okna (sliding window), s cílem převést časové řady na formát vhodný pro sekvenční učení,
- **návrh, trénink a testování** několika variant modelů strojového učení aplikovaných na jednotlivé predikční přístupy,
- **porovnání přesnosti** jednotlivých kombinací model–predikce na základě kvantitativních metrik (MSE, R^2), a vyhodnocení jejich chování při různých letových režimech,
- **výběr nejvhodnější varianty** a její implementace do reálného systému pro predikci v průběhu letu včetně testu při manuálním řízení dronu.

1.1 Teoretický úvod

Strojové učení (machine learning) se zabývá vývojem algoritmů umožňujících počítačům učit se z dat a na základě tohoto učení dělat rozhodnutí neboli predikce bez explicitního naprogramování, jak podrobně popisuje [9].Základním principem je vytvoření modelu, který na základě vstupních dat a zpětné vazby optimalizuje své vnitřní parametry s cílem minimalizovat chybu predikce.



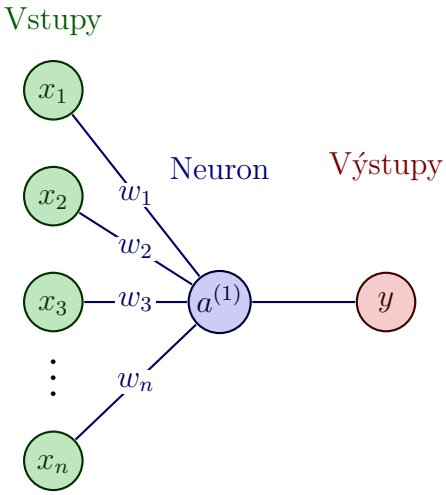
Obrázek 1.1: Cyklus algoritmu

Seznam komponent:

- **Dataset** – Objekt, který drží informace o tom, jak vypadá vstup a příslušný cílový výstup modelu. Na těchto datech se model bude učit.
- **Model** – Neuronová síť, která má svoje parametry (váhy W a biasy b). Na základě vstupních dat vytváří model předpovědi.
- **Chybová funkce (Loss)** – Funkce, která vypočte chybu mezi předpovědí modelu a cílovou výstupovou hodnotou (target) z datasetu.
- **Backpropagation** – Algoritmus pro výpočet gradientů chyby vůči parametrům modelu. Říká, jak máme změnit parametry modelu vůči chybě predikce modelu.
- **Učící se algoritmus** – upravuje parametry modelu na základě gradientů.

1.1.1 LNU model

Základní jednotkou neuronové síťě je LNU (Linear neural unit), jak je popsáno například v [3]. Funguje na podobném principu jako biologický neuron, jak je dále popsáno v [12] a [24].



Obrázek 1.2: Lineární neuronová jednotka

Neuron kombinuje všechny vstupy x_i, násobí je náležitými váhami w_i a poté přidá bias b.

$$y = \Sigma_i^n x_i w_i + b, \tag{1.1}$$

kde

- x_i jsou vstupní hodnoty,
- w_i jsou váhy modelu,
- b je bias a
- y je výstup modelu.

Pomocí matic se rovnice 1.1 vyjádří jako

$$y = \mathbf{X}_{1\times n} \cdot \mathbf{W}_{n\times 1} + b, \tag{1.2}$$

kde

- $\mathbf{X} \in \mathbb{R}^{1\times n}$ je vektor vstupů,
- $\mathbf{W} \in \mathbb{R}^{n\times 1}$ je vektor vah,
- $b \in \mathbb{R}$ je bias.

Backpropagation

Jakmile model provede predikci a spočítá se chyba mezi predikovanou hodnotou a skutečnou cílovou hodnotou, je třeba zjistit, jak tato chyba závisí na parametrech modelu – tedy na vahách W a biasu b. Tento proces se nazývá backpropagation (zpětné šíření chyby) a umožňuje postupně upravit parametry tak, aby model predikoval přesněji. Chybová funkce E vyjadřuje rozdíl mezi výstupem modelu y a cílovou hodnotou. Pro příklad je použita funkce MSE (Mean Squared Error), která je definována jako

$$E = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{1.3}$$

kde $\hat{y}_i$ je cílová hodnota (target) a y_i je predikce modelu.
Chybové funkce jsou do podrobna popsány v sekci 1.1.4

Pro úpravu parametrů modelu je třeba spočítat parciální derivace chyby E podle jednotlivých parametrů modelu, tedy $\frac{\partial E}{\partial \mathbf{W}}$ a $\frac{\partial E}{\partial b}$.

Protože chyba je složená funkce těchto parametrů $E(y(\mathbf{W}, b))$, použijeme pravidla pro derivace složených funkcí

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial b} = \frac{\partial E}{\partial y}, \tag{1.4}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w_i} + \frac{\partial E}{\partial y}x_i \longrightarrow \frac{\partial E}{\partial \mathbf{W}} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial \mathbf{W}} = \frac{\partial E}{\partial y} \cdot \mathbf{X}^T. \tag{1.5}$$

Tyto vztahy umožňují vypočítat gradienty a následně upravit parametry modelu pomocí učícího se algoritmu. Pro Gradient Descent by aktualizace vypadala jako

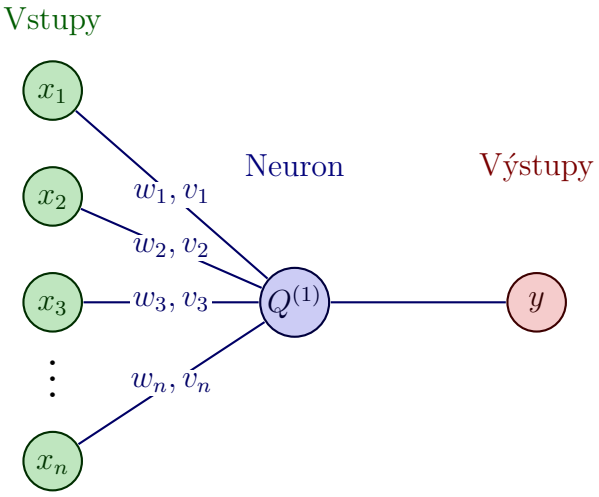
$$\mathbf{W} \longleftarrow \mathbf{W} - \eta \cdot \frac{\partial E}{\partial \mathbf{W}}, \tag{1.6}$$

$$b \longleftarrow b - \eta \cdot \frac{\partial E}{\partial b}, \tag{1.7}$$

kde η je learning rate (rychlost učení). Učící se algoritmy jsou do podrobna popsány v sekci 1.1.5

1.1.2 QNU model

Kvadratická neuronová jednotka (QNU, z anglického Quadratic Neural Unit) představuje rozšíření klasického LNU o kvadratické členy, jak popisuje [3]. Zatímco LNU využívá pouze lineární kombinaci vstupů, QNU umožňuje modelovat i nelineární vztahy mezi vstupními proměnnými, čímž zvyšuje kapacitu modelu učit se složitější závislosti.



Obrázek 1.3: Kvadratická neuronová jednotka

Výstup model lze zapsat jako

$$y = \left(\sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ij}\, x_i\, x_j + b \right), \tag{1.8}$$

kde

- x_i jsou vstupní hodnoty,
- w_i jsou váhy pro lineární členy,
- v_i jsou váhy pro kvadratické členy,
- b je bias,
- y je výstup modelu.

Pomocí maticového zápisu můžeme vztah přepsat na

$$y = \underbrace{\mathbf{X}_{(1\times n)}\, \mathbf{W}_{(n\times 1)}}_{\text{Lineární člen}} + \underbrace{\mathbf{X}_{(1\times n)}\, \mathbf{V}_{(n\times n)}\, \mathbf{X}^{T}_{(n\times 1)}}_{\text{Kvadratický člen}} + b_{(1\times 1)}, \tag{1.9}$$

kde

- $\mathbf{X} \in \mathbb{R}^{1\times n}$ je vektor vstupů,
- $\mathbf{W} \in \mathbb{R}^{n\times 1}$ je vektor lineárních vah,
- $\mathbf{V} \in \mathbb{R}^{n\times n}$ je matice kvadratických vah,
- $b \in \mathbb{R}$ je bias.

Backpropagation

Stejně jako u klasického LNU, i v případě kvadratické neuronové jednotky potřebujeme spočítat, jak se mění chyba modelu vzhledem k jeho parametrům, tedy váhám lineárním W, kvadratickým váhám V, a biasu b. Celková chyba je opět popsána chybovou funkcí $E(y)$, která je funkcí výstupu neuronové jednotky.

Bias je přičten k výstupu jako konstanta, takže derivace chyby podle biasu je přímo derivací chyby podle výstupu

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial y}. \tag{1.10}$$

Lineární člen má tvar XW, takže jeho derivace

$$\frac{\partial E}{\partial \mathbf{W}} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial \mathbf{W}} = \frac{\partial E}{\partial y} \cdot \mathbf{X}^T. \tag{1.11}$$

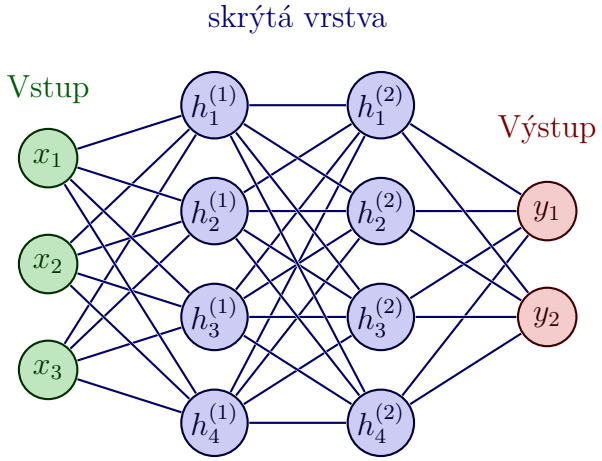
Kvadratický člen má tvar $\mathbf{X}^T\mathbf{V}\mathbf{X}$, což je skalár. Derivace tohoto výrazu podle matice $\mathbf{V}$ je

$$\frac{\partial y}{\partial \mathbf{V}} = \frac{\partial}{\partial \mathbf{V}} \left(\mathbf{X}\,\mathbf{V}\,\mathbf{X}^T\right) = \mathbf{X}^T\mathbf{X}, \tag{1.12}$$

$$\Rightarrow \frac{\partial E}{\partial \mathbf{V}} = \frac{\partial E}{\partial y} \cdot \mathbf{X}^T\mathbf{X}. \tag{1.13}$$

1.1.3 MLP model

Vícevrstvý perceptron (MLP, z anglického Multilayer Perceptron) je základní architekturou neuronové sítě, skládající se z několika vrstev neuronových jednotek. Každý neuron v jedné vrstvě je propojen se všemi neurony ve vrstvě následující. Díky více vrstvám a nelineárním aktivačním funkcím je MLP schopné aproximovat i velmi složité nelineární funkce, jak také popisuje [26] a [9].



Obrázek 1.4: MLP model

Každá vrstva provádí výpočet

$$\mathbf{h}^{(l)} = \sigma\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\right), \tag{1.14}$$

kde

- l značí indexy vrstev,
- $\mathbf{h}^{(l-1)}$ je výstup předchozí vrstvy (resp. vstup $\mathbf{X}$ pro první skrytou vrstvu),
- $\mathbf{W}^{(l)}$ je matice vah pro vrstvu l,
- $\mathbf{b}^{(l)}$ je vektor biasů pro vrstvu l,
- σ je nelineární aktivační funkce (např. ReLU, sigmoid, tanh).

Uvažujme model se dvěma skrytými vrstvami a výstupem jako na obr. 1.4 Celý průchod modelem by vypadal jako

$$\mathbf{h}^{(1)}_{(4\times1)} = \sigma_1\left(\mathbf{W}^{(1)}_{(4\times3)}\mathbf{X}_{(3\times1)} + \mathbf{b}^{(1)}_{(4\times1)}\right), \tag{1.15}$$

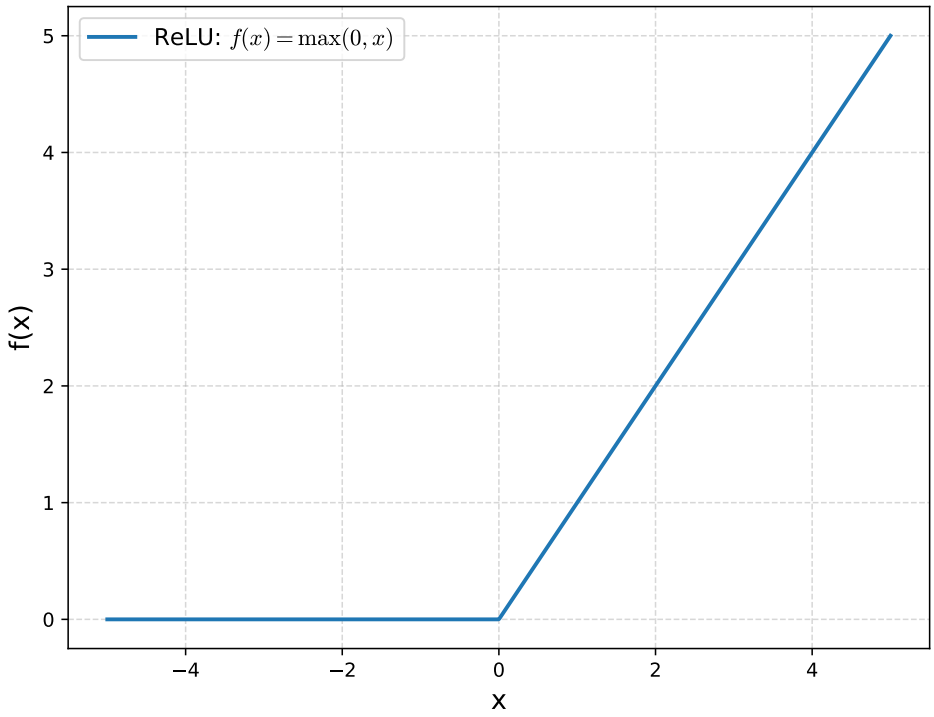
$$\mathbf{h}^{(2)}_{(4\times1)} = \sigma_2\left(\mathbf{W}^{(2)}_{(4\times4)}\mathbf{h}^{(1)}_{(4\times1)} + \mathbf{b}^{(2)}_{(4\times1)}\right), \tag{1.16}$$

$$\mathbf{y}_{(2\times1)} = \sigma_3\left(\mathbf{W}^{(3)}_{(2\times4)}\mathbf{h}^{(2)}_{(4\times1)} + \mathbf{b}^{(3)}_{(2\times1)}\right). \tag{1.17}$$

Aktivační funkce

Pro každou vrstvu je použita aktivační funkce, která do výpočtu zavádí nelinearitu. Model je poté schopen se učit komplexní vzory. Do modelů se většinou přidává jako samostatná vrstva. Různé aktivační funkce mají odlišné vlastnosti a hodí se pro různé typy úloh, jak popisuje [5]. V této práci je aktivační funkce značena pod písmenem σ.

ReLU: Funkce ReLU je jednoduchá a výpočetně velmi efektivní. Vrací zadanou hodnotu, pokud je kladná, jinak vrací nulu. Díky tomu urychluje trénink. Na druhou stranu může vést k tomu, že se některé neurony vypnou, pokud se jejich výstup dlouhodobě nachází v záporné části.

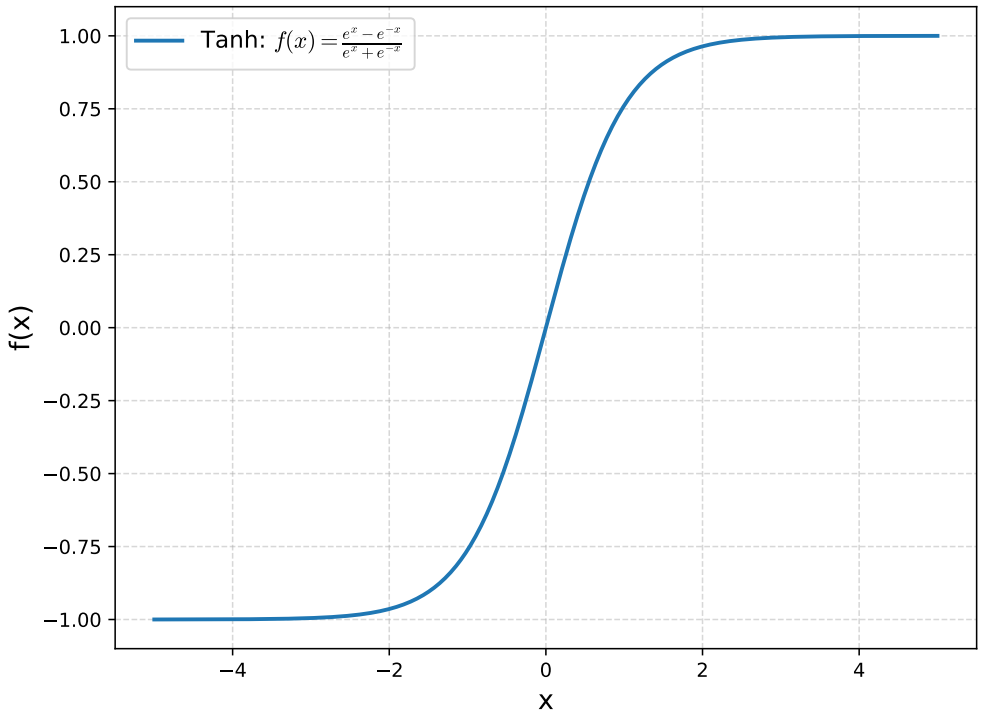


Obrázek 1.5: ReLU

Funkce ReLU je definována jako

$$\mathrm{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{1.18}$$

Tanh: Funkce tanh převádí vstupní hodnoty do intervalu $(-1, 1)$, což je často výhodné, protože výsledky jsou centrované kolem nuly. To může urychlit konvergenci během učení. Tanh ale trpí saturací. Pokud vstup do neuronu spadne do saturační oblasti, kde se křivka zplošťuje, výstup se téměř nemění bez ohledu na změny vah.



Obrázek 1.6: Tanh

Funkce Tanh je definována jako

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{1.19}$$

Backpropagation

V případě MLP se backpropagation používá pro výpočet gradientů vůči všem parametrům ve všech vrstvách sítě.
Cílem je spočítat parciální derivace chyby E vzhledem k parametrům jednotlivých vrstev $\frac{\partial E}{\partial \mathbf{W}^{(l)}}$, $\frac{\partial E}{\partial \mathbf{b}^{(l)}}$, kde l značí index vrstvy.

Nejprve rozdělíme výpočet vrstvy na dvě části.
Na lineární transformaci

$$\mathbf{Z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)} \tag{1.20}$$

a na aktivační funkci

$$\mathbf{h}^{(l)} = \sigma(\mathbf{Z}^{(l)}) \tag{1.21}$$

Gradienty vah a biasů se počítají jako

$$\frac{\partial E}{\partial \mathbf{W}^{(l)}} = \left(\frac{\partial E}{\partial \mathbf{Z}^{(l)}}\right)\left(\mathbf{h}^{(l-1)}\right)^T, \tag{1.22}$$

$$\frac{\partial E}{\partial \mathbf{b}^{(l)}} = \frac{\partial E}{\partial \mathbf{Z}^{(l)}}. \tag{1.23}$$

Gradient výstupu pro předchozí vrstvu se spočítá jako

$$\frac{\partial E}{\partial \mathbf{h}^{(l-1)}} = \begin{pmatrix} \frac{\partial E}{\partial Z_1^{(l)}} w_{11}^{(l)} + \frac{\partial E}{\partial Z_2^{(l)}} w_{21}^{(l)} \\ \frac{\partial E}{\partial Z_1^{(l)}} w_{12}^{(l)} + \frac{\partial E}{\partial Z_2^{(l)}} w_{22}^{(l)} \\ \frac{\partial E}{\partial Z_1^{(l)}} w_{13}^{(l)} + \frac{\partial E}{\partial Z_2^{(l)}} w_{23}^{(l)} \\ \frac{\partial E}{\partial Z_1^{(l)}} w_{14}^{(l)} + \frac{\partial E}{\partial Z_2^{(l)}} w_{24}^{(l)} \end{pmatrix} = \frac{\partial E}{\partial \mathbf{Z}^{(l)}}\left(\mathbf{W}^{(l)}\right)^T. \tag{1.24}$$

Po přidání aktivační funkce budou rovnice vypadat jako

$$\frac{\partial E}{\partial \mathbf{W}^{(3)}} = \frac{\partial E}{\partial \mathbf{Y}}\frac{\partial \mathbf{Y}}{\partial \mathbf{Z}}\left(\mathbf{h}^{(2)}\right)^T, \tag{1.25}$$

$$\frac{\partial E}{\partial \mathbf{b}} = \frac{\partial E}{\partial \mathbf{Y}}\frac{\partial \mathbf{Y}}{\partial \mathbf{Z}}, \tag{1.26}$$

$$\frac{\partial E}{\partial \mathbf{h}^{(2)}} = \frac{\partial E}{\partial \mathbf{Y}}\frac{\partial \mathbf{Y}}{\partial \mathbf{Z}}\left(\mathbf{W}^{(3)}\right)^T. \tag{1.27}$$

kde $\frac{\partial \mathbf{Y}}{\partial \mathbf{Z}}$ je derivace aktivační funkce a $\frac{\partial E}{\partial \mathbf{Y}}$ je derivace chybové funkce.

Tímto způsobem můžeme postupovat od výstupu až po první vrstvu a zpětně zpočítat backpropagation pro celou síť. Na základě těchto derivací jsou následně pomocí učícího se algoritmu aktualizovány všechny parametry sítě.

1.1.4 Chybové funkce

Chybová funkce (loss) měří, jak dobře model predikuje skutečné hodnoty. Přehled běžně používaných chybových funkcí poskytuje [4]. Cílem trénování neuronové sítě je minimalizovat tuto chybu, aby se model co nejlépe přizpůsobil trénovacím datům. Volba správné chybové funkce závisí na typu úlohy a vlastnostech dat.

Střední kvadratická chyba (Mean Squared Error, MSE)

Pro mnohé případy se používá střední kvadratická chyba, která je definována jako

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{1.28}$$

Střední absolutní chyba (Mean Absolute Error, MAE)

Další běžnou chybovou funkcí je střední absolutní chyba, která je definována jako

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{1.29}$$

Metriky jako MSE a MAE jsou běžně používané při hodnocení modelů, jak popisuje [13]. V této práci je však používána pouze MSE, protože klade větší důraz na větší chyby díky kvadratickému členům, což je vhodné pro náš model, který potřebuje penalizovat výrazné odchylky více než malé odchylky.

1.1.5 Učící se algoritmy

Gradient Descent (GD)

Gradient Descent je iterativní optimalizační algoritmus používaný pro hledání minima chybové funkce (loss), jak popisuje [22]. Srovnání výkonu různých algoritmů učení shrnuje [7].
Algoritmus aktualizuje parametry modelu θ v opačném směru gradientu funkce ztráty $E(\theta)$ vzhledem k parametrům

$$\theta \longleftarrow \theta - \eta \cdot \nabla_\theta E(\theta_t) \tag{1.30}$$

kde

- θ jsou parametry modelu
- η je learning rate (rychlost učení)
- $\nabla_\theta E(\theta)$ je gradient funkce ztráty vzhledem k parametrům

Batch Gradient Descent (BGD)

Batch Gradient Descent (BGD) je varianta Gradient Descentu, kde se gradient funkce ztráty vypočítává najednou pro celou trénovací množinu, označovanou jako *batch*. Tímto způsobem algoritmus zohledňuje všechny trénovací příklady při každé aktualizaci parametrů, což vede k přesnějšímu odhadu gradientu, ale může být výpočetně náročnější u velkých datasetů. Gradient je dán vztahem

$$\nabla_\theta E(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta E(\theta; X_i, y_i), \tag{1.31}$$

kde

- m je počet trénovacích příkladů v *batchi* (tedy v celé trénovací množině),
- X_i představuje vstupní data (např. vektor příznaků) pro i-tý trénovací příklad,
- y_i je odpovídající skutečná hodnota (výstup) pro i-tý trénovací příklad.

Stochastic Gradient Descent (SGD)

Aktualizuje parametry pro každý trénovací příklad zvlášť. Vybere náhodná X_i a y_i z batche.

$$\theta \longleftarrow \theta - \eta \cdot \nabla_\theta E(\theta; X_i, y_i) \tag{1.32}$$

Learning Rate

Volba learning rate η je kritická:

- Příliš vysoká η může způsobit divergenci
- Příliš nízká η zpomaluje konvergenci

Momentum

Pro urychlení konvergence se často používá momentum:

$$v_i \longleftarrow \gamma v_i + \eta \nabla_\theta E(\theta_t) \tag{1.33}$$

$$\theta_i \longleftarrow \theta_i - v_i \tag{1.34}$$

kde γ je momentum faktor (typicky 0.9).

Adam

Adam (zkratka pro Adaptive Moment Estimation) je pokročilý optimalizační algoritmus, který kombinuje výhody metody momenta a adaptivního přizpůsobení learning rate pro každý parametr zvlášť, jak také popisuje [26] a [16]. Udržuje si dvě pomocné proměnné. Průměr gradientů (první moment) a průměr druhých mocnin gradientů (druhý moment).

- m – odhad průměrného gradientu (1. moment)
- v – odhad průměrného čtverce gradientu (2. moment)

Algoritmus postupuje následovně.

$$g \longleftarrow \nabla_\theta E(\theta) \quad \text{(aktuální gradient)} \tag{1.35}$$

$$m \longleftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot g \quad \text{(aktualizace 1. momentu)} \tag{1.36}$$

$$v \longleftarrow \beta_2 \cdot v + (1 - \beta_2) \cdot g^2 \quad \text{(aktualizace 2. momentu)} \tag{1.37}$$

$$\hat{m} \longleftarrow \frac{m}{1 - \beta_1^t} \quad \text{(korekce vychýlení pro } m\text{)} \tag{1.38}$$

$$\hat{v} \longleftarrow \frac{v}{1 - \beta_2^t} \quad \text{(korekce vychýlení pro } v\text{)} \tag{1.39}$$

$$\theta \longleftarrow \theta - \eta \cdot \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon} \quad \text{(aktualizace parametrů)} \tag{1.40}$$

kde

- η je learning rate,
- β_1 je váha pro průměr gradientu,
- β_2 je váha pro průměr druhých mocnin gradientu,
- ϵ je malá konstanta kvůli numerické stabilitě.

Výchozí hyperparametry obvykle fungují dobře ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$)

Adam se v praxi velmi často používá jako výchozí optimalizační algoritmus, protože kombinuje to nejlepší. Je rychlý a nevyžaduje složité ladění. Díky adaptivnímu učení pro každý parametr zvlášť zvládá i složité a nelineární optimalizační úlohy s vysokou dimenzionalitou. Dobře funguje i při trénování neuronových sítí s velkým množstvím dat nebo parametrů. Výchozí nastavení hyperparametrů bývá ve většině případů dostatečné, což z něj dělá pohodlnou a efektivní volbu téměř pro jakýkoliv model.

Kapitola 2

Experimentální sběr dat

Pro experimenty byl použit malý kvadrokoptérový dron *Crazyflie 2.1* od společnosti Bitcraze [1].
Komunikace s dronem probíhala přes Python pomocí knihovny `cflib` dostupné od výrobce, která umožňuje řízení letových manévrů i sběr senzorických dat, jak je popsáno na stránkách výrobce [2].

Skript 1 Použité knihovny při ovládání dronu

```
import cflib.crtp
from cflib.crazyflie import Crazyflie
from cflib.crazyflie.log import LogConfig
from cflib.crazyflie.syncCrazyflie import SyncCrazyflie
from cflib.positioning.motion_commander import MotionCommander
from cflib.utils import uri_helper
from cflib.utils import power_switch
```

2.1 Sběr dat

Data byla sbírána během standardních letových manévrů, které zahrnovaly vznášení se dronu na místě, periodický pohyb dopředu a dozadu a periodický pohyb nahoru a dolů.

Skript 2 Program pro ovládání drona pro periodický pohyb dopředu a dozadu

```
def move(scf):
    with MotionCommander(scf, default_height=DEFAULT_HEIGHT) as mc:
        time.sleep(1)
        while True:
            try:
                mc.forward(2, velocity=1)
                time.sleep(0.3)
                mc.back(2, velocity=1)
                time.sleep(0.3)
            except KeyboardInterrupt:
                psw.platform_power_down()
```

Během letů bylo měřeno

- Napětí baterie u $[V]$,
- PWM hodnoty motorů - řídicí signály pro všechny 4 motory m_1 [PWM], m_2 [PWM], m_3 [PWM] a m_4 [PWM],
- Poloha dronu - prostorové souřadnice x $[m]$, y $[m]$ a z $[m]$, které měřily vzdálenost dronu od počáteční startovní polohy v metrech.

Hodnoty byly měřeny po vzorkovacím intervalu $\Delta t = 100$ $[ms]$

Skript 3 Sběr hodnot pomocí programu

```
INTERVAL = 100 #ms
if __name__ == '__main__':
    # Inicializuje drivery
    cflib.crtp.init_drivers()
    with SyncCrazyflie(URI, cf=Crazyflie(rw_cache='./cache')) as scf:
        scf.cf.param.add_update_callback(group='deck', name='bcFlow2',
        cb=param_deck_flow)
        time.sleep(1)

        posconf = LogConfig(name='position', period_in_ms=INTERVAL)
        posconf.add_variable('stateEstimate.x', 'float')
        posconf.add_variable('stateEstimate.y', 'float')
        posconf.add_variable('stateEstimate.z', 'float')
        scf.cf.log.add_config(posconf)
        posconf.data_received_cb.add_callback(acc_callback)

        logconf = LogConfig(name='motor', period_in_ms=INTERVAL)
        logconf.add_variable('motor.m1', 'uint16_t')
        logconf.add_variable('motor.m2', 'uint16_t')
        logconf.add_variable('motor.m3', 'uint16_t')
        logconf.add_variable('motor.m4', 'uint16_t')
        scf.cf.log.add_config(logconf)
        logconf.data_received_cb.add_callback(acc_callback)

        batconf = LogConfig(name='battery', period_in_ms=INTERVAL)
        batconf.add_variable('pm.vbat', 'float')
        scf.cf.log.add_config(batconf)
        batconf.data_received_cb.add_callback(acc_callback)

        if not deck_attached_event.wait(timeout=5):
            print('No flow deck detected!')
            sys.exit(1)

        logconf.start()
        batconf.start()
        posconf.start()
        move(scf)
        logconf.stop()
        batconf.stop()
        posconf.stop()
```

Data byla následně uložena do sourobrů typu `.csv` .
Další výpočty a manipulace s daty byly realizovány pomocí knihoven `NumPy` a `pandas`, které poskytují efektivní nástroje pro práci s numerickými daty v Pythonu, jak popisuje [11, 17].

timestap [ms]	motor m1 [PWM]	motor m2 [PWM]	motor m3 [PWM]	motor m4 [PWM]
43216	35333	35285	33677	33089
43316	37350	30942	36806	38710
43416	31913	39799	43711	29937
43516	33476	32534	37156	33626
43616	38478	28292	33922	41160
43716	43211	38517	41695	46693
43816	49782	54594	48670	39914
43916	37757	40999	31465	23451
44016	23563	46849	40301	12451
44116	34901	25691	34225	38611

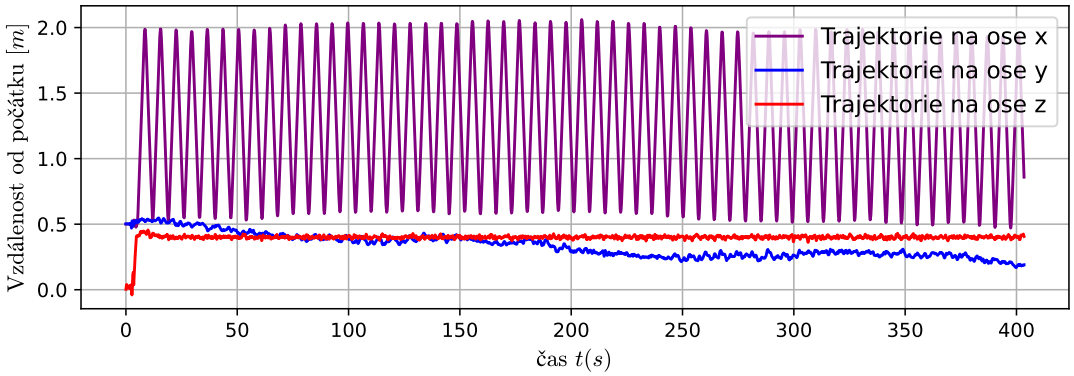
timestap [ms]	x[m]	y[m]	z[m]	timestap [ms]	Voltage [V]
43219	0.50081	0.50238	0.02670	43218	3.61584
43319	0.49911	0.50111	0.01287	43318	3.72141
43419	0.49483	0.48927	0.02019	43418	3.72669
43519	0.49141	0.47594	0.03287	43518	3.76891
43619	0.48677	0.46955	0.03851	43618	3.75308
43719	0.48550	0.48354	-0.00243	43718	3.70557
43819	0.49348	0.51395	-0.03995	43818	3.56305
43919	0.50662	0.52538	0.00675	43918	3.74780
44019	0.51326	0.51582	0.06274	44018	3.83226
44119	0.50738	0.50758	0.09501	44118	3.79003

Tabulka 2.1: Struktura dat v .csv souboru

2.2 Předzpracování dat

Uletěná vdálenost

Pokud se za let dron pohyboval a nevznášel se celou dobu na místě, byla měřena jeho poloha v metrech od počátečního vzletového bodu.



Obrázek 2.1: Ukázka dráhy letu z letu 5

Vzdálenost v každém bodě, tedy vzdálenost, kterou dron uletěl za 100 milisekund, jsem počítal jako

$$d(t) = \sqrt{x(t+\Delta t)^2 + y(t+\Delta t)^2 + z(t+\Delta t)^2}\ [m]; \quad \Delta t = 100\ [ms]. \tag{2.1}$$

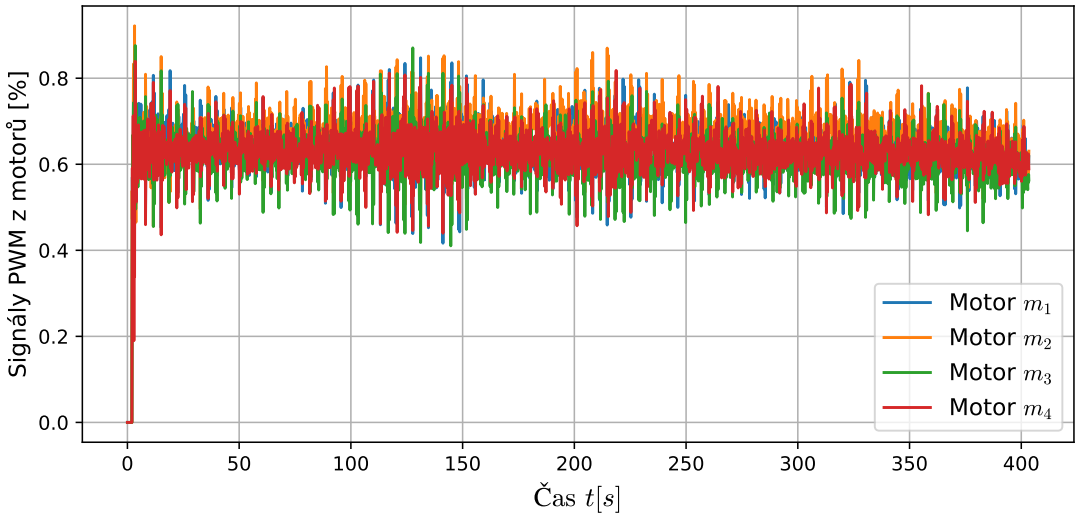
Celková uletěná vzdálenost D se spočítala jako

$$D = \sum_{t}^{T} d(t)\ [m], \tag{2.2}$$

kde T je celková doba letu.

Signál PWM

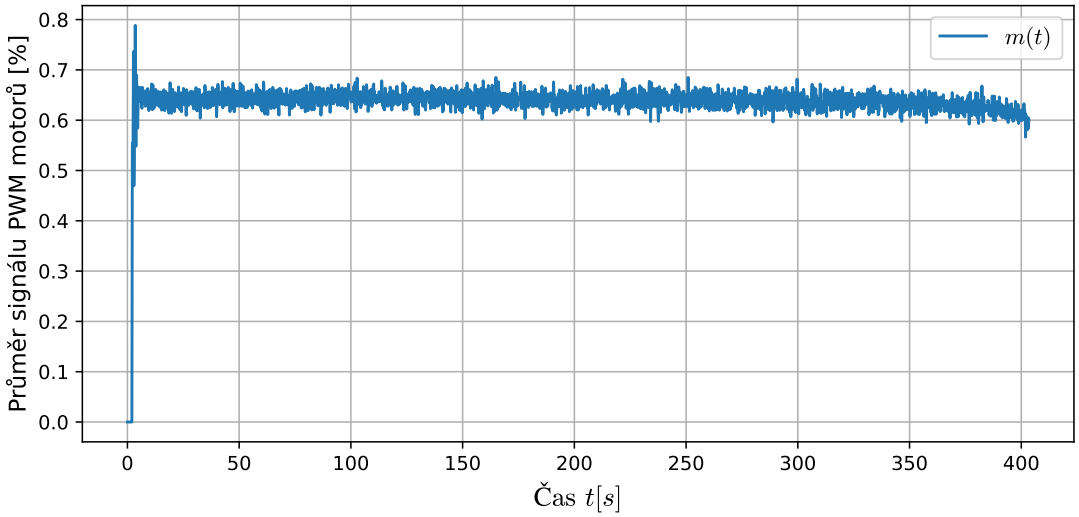
Signál PWM je u jednotlivých motorů normalizován hodnotou 65535, která odpovídá maximální hodnotě signálu.



Obrázek 2.2: Procentuálních signál PWM jednotlivých motorů

Z těchto hodnot je vytvořen aritmetický průměr

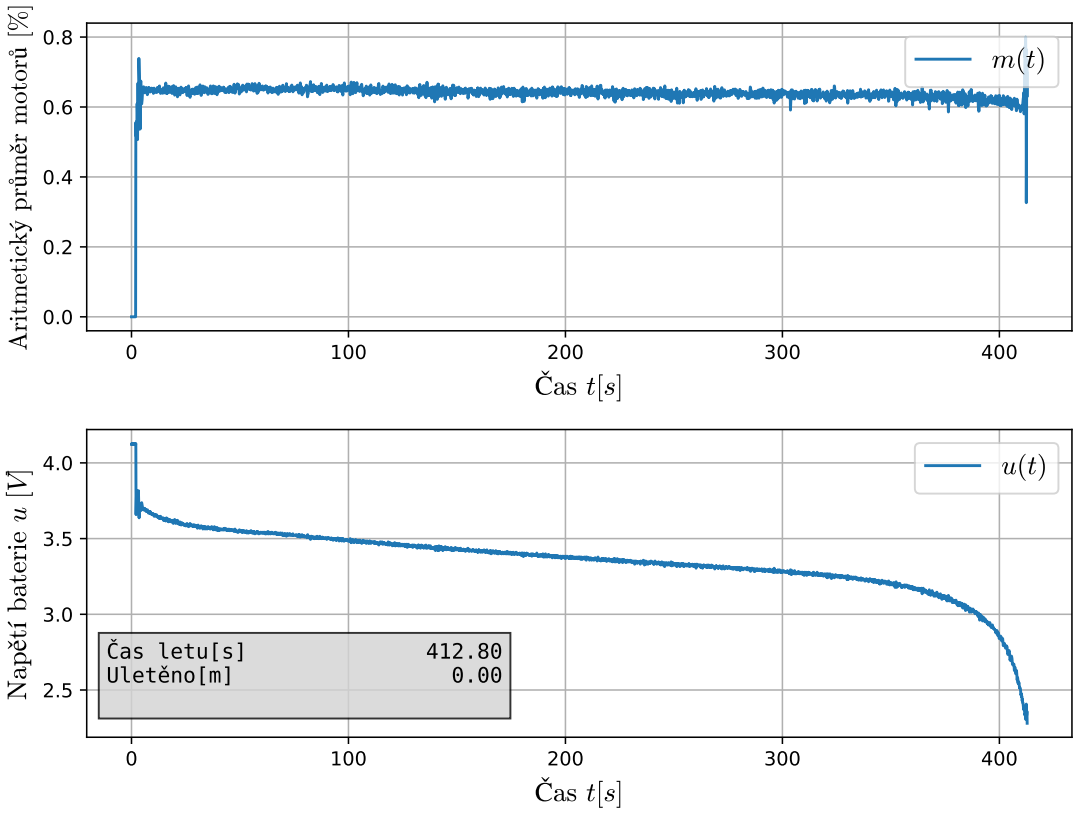
$$m = \frac{m_1 + m_2 + m_3 + m_4}{4}[\%]. \tag{2.3}$$



Obrázek 2.3: Aritmetický průměr procentuálního signálů PWM

2.3 Jednotlivé lety

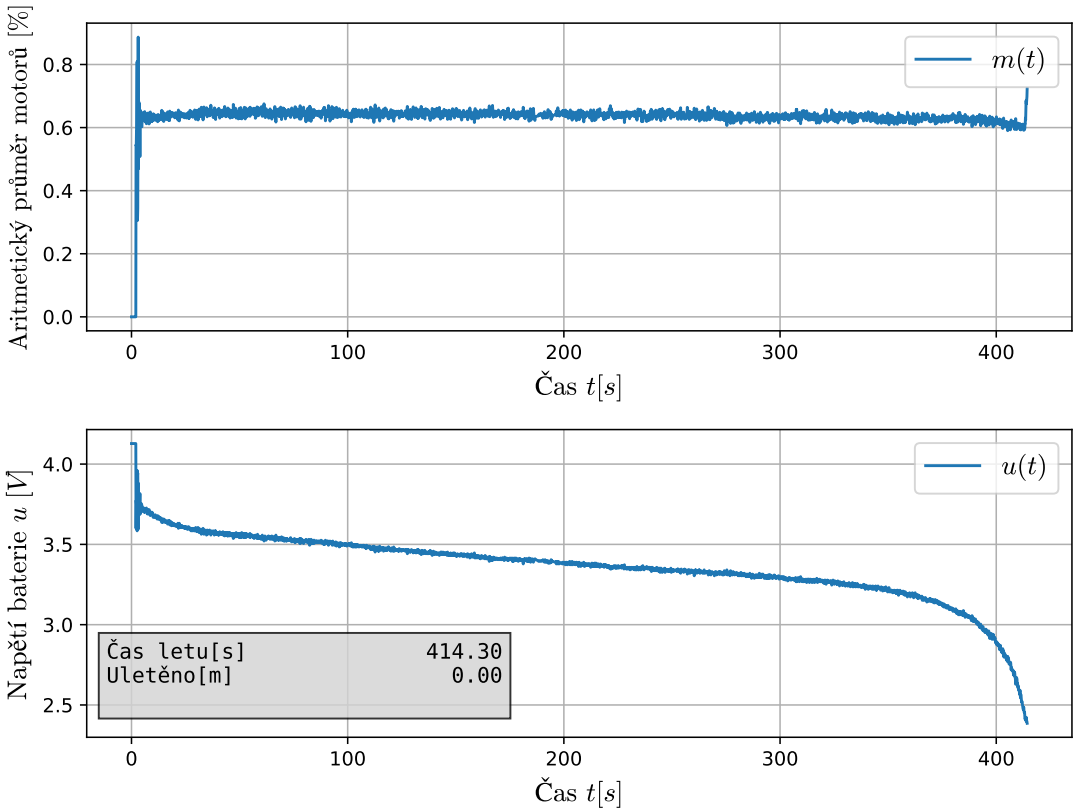
1. let:



Obrázek 2.4: Dron se vznášel nad místem

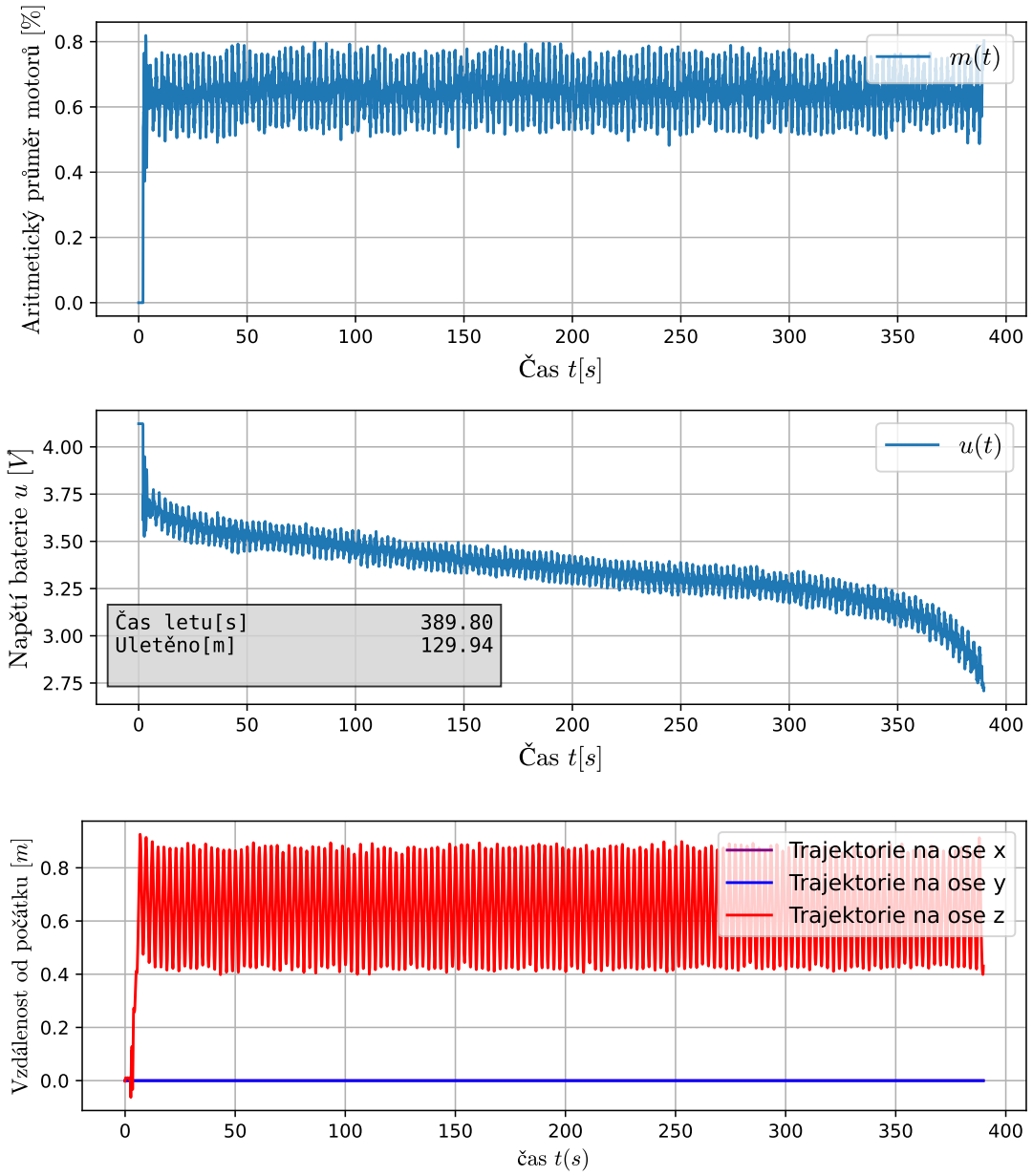
Pozn.: Na obrázku 2.4 lze pozorovat, že ačkoli dron vykonal vzlet ze země, výsledná uletěná vzdálenost je nulová. To je způsobeno tím, že z dat každého letu bylo později odstraněno prvních 5 sekund měření, během nichž dron prováděl vzlet. Tento postup je oprávněný, neboť dron na počátku každého letu dosahuje stejné výšky a výrazné odchylky v počáteční fázi letu negativně ovlivňují přesnost učení modelu.

2. let:



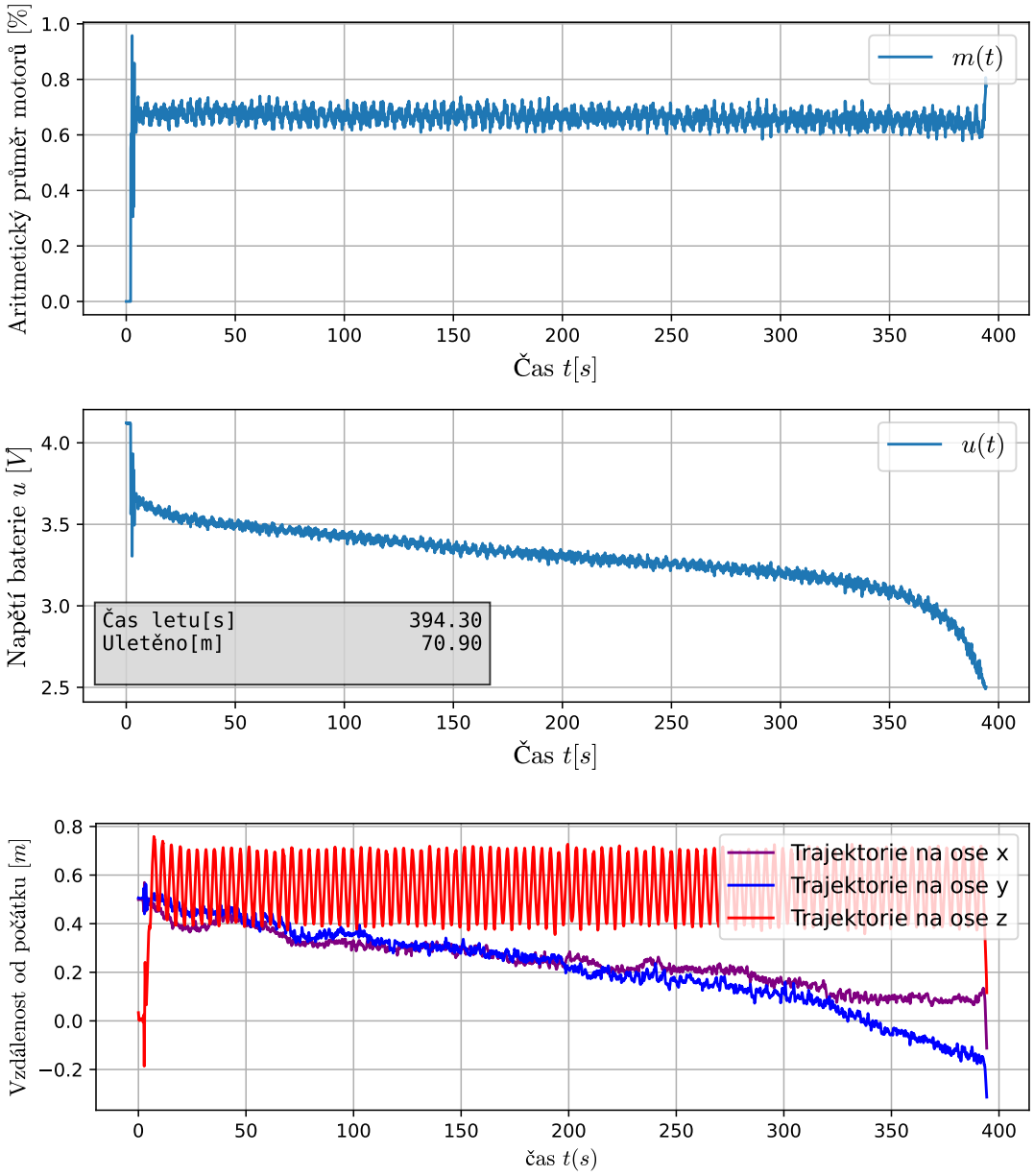
Obrázek 2.5: Dron se vznášel nad místem

3. let:



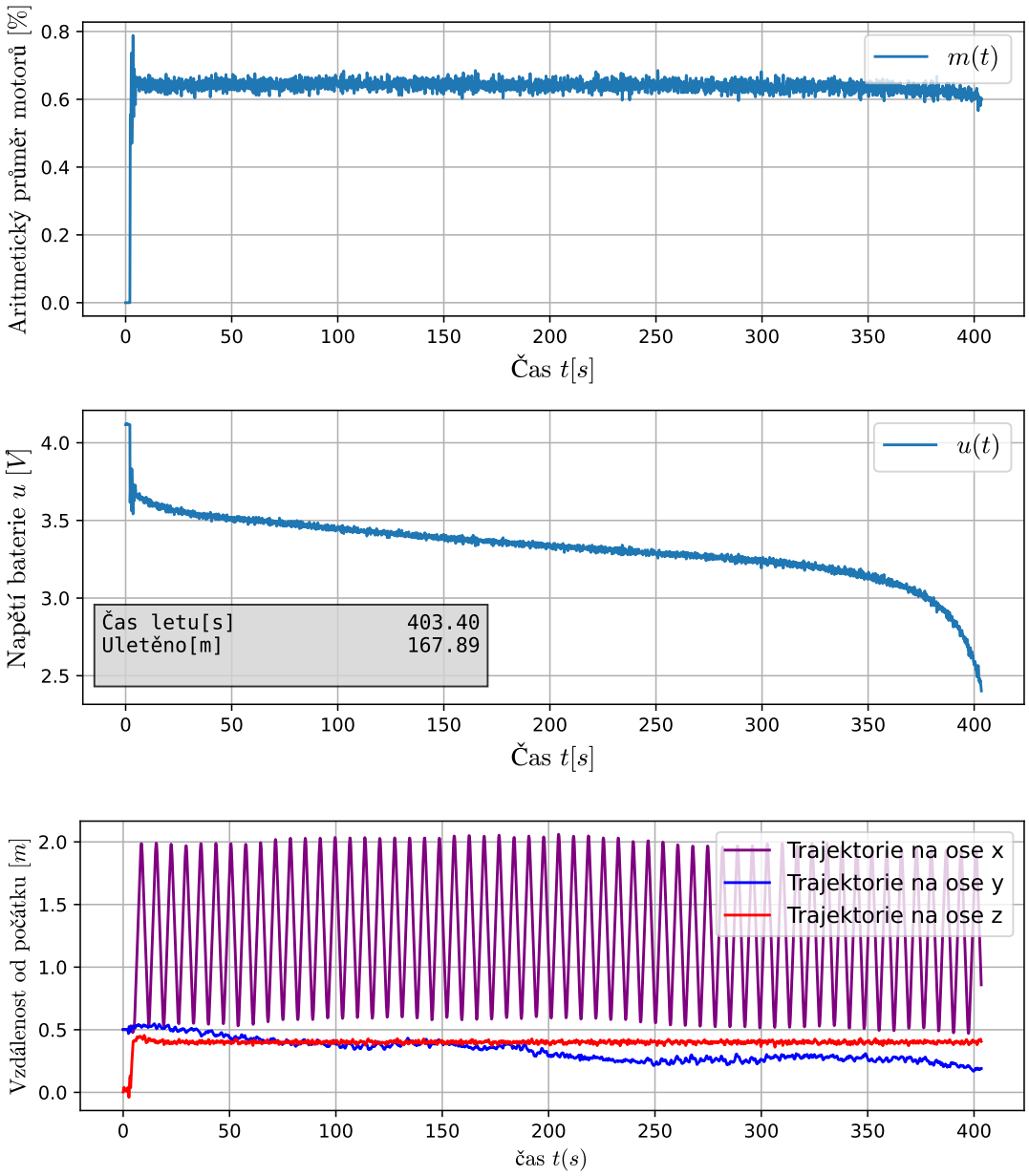
Obrázek 2.6: Dron se pohyboval nahoru a dolů o 0.5 metrů s rychlostí 0.4 $m \cdot s^{-1}$. Po každém vznesení a snesení se dron zastavil na 0.1 sekund. V tomto měření se osy x a y nezaznamenávaly a nastavily se na nulu.

4. let:



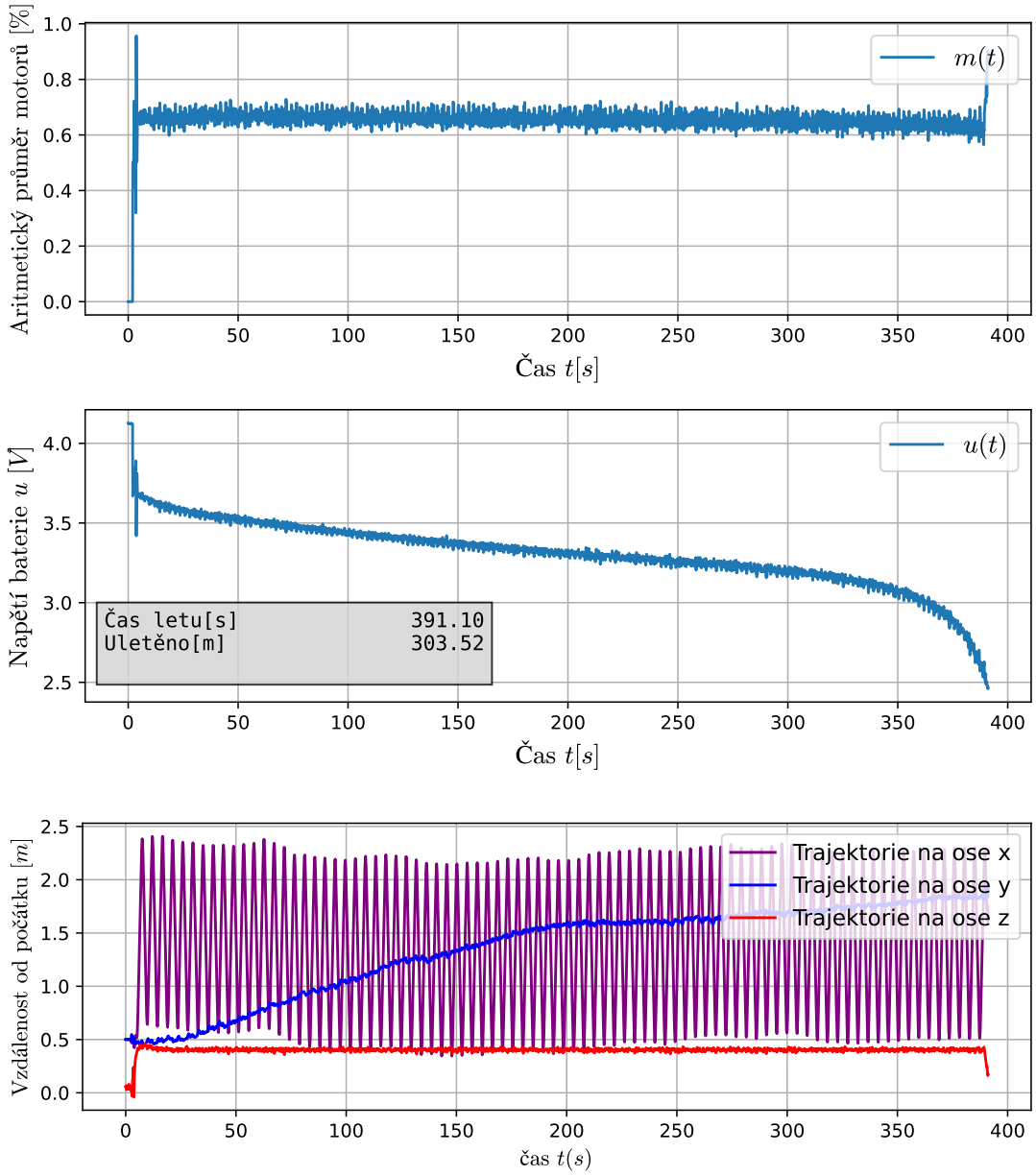
Obrázek 2.7: V tomto letu se dron pohyboval nahoru a dolů o 0.3 metrů s rychlostí 0.2 $m \cdot s^{-1}$. Po každém vznesení a snesení se dron zastavil na 0.5 sekund.

5. let:



Obrázek 2.8: V tomto letu se dron pohyboval dopředu a zpátky o 1.5 metru s rychlostí 0.5 $m \cdot s^{-1}$. Po každém celém posunutí se dron zastavil na 0.5 sekund.

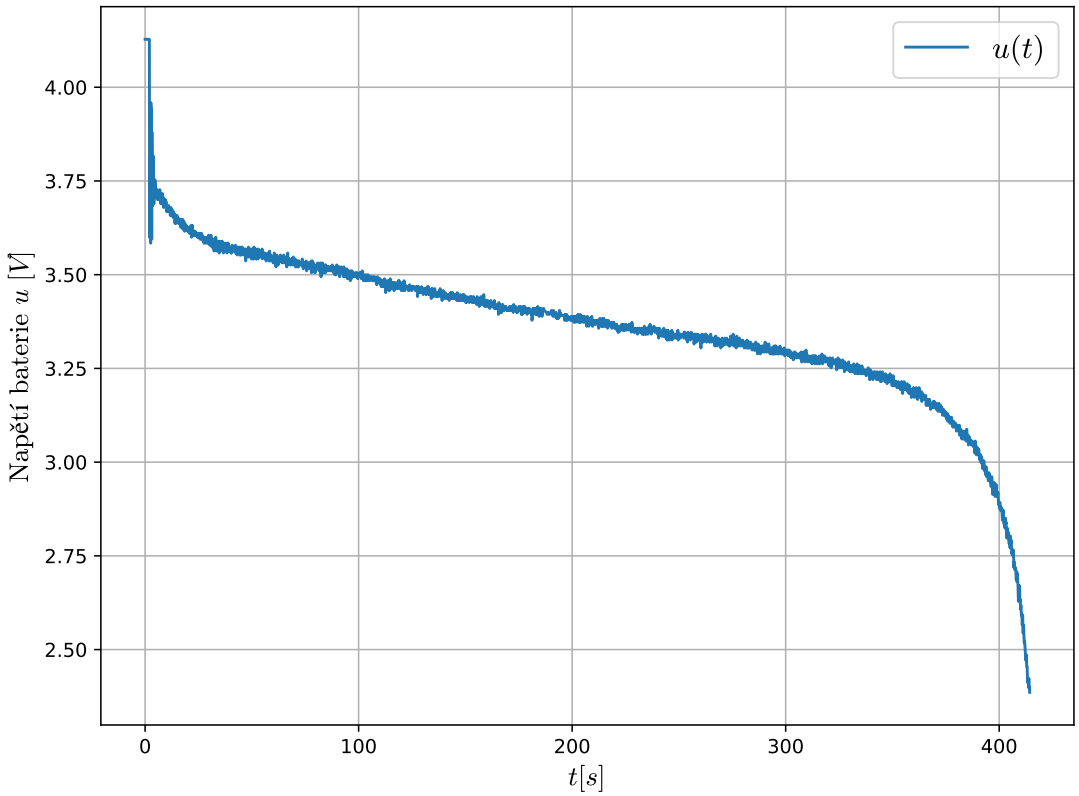
6. let:



Obrázek 2.9: V tomto letu se dron pohyboval dopředu a zpátky o 2 metry s rychlostí $1\ m \cdot s^{-1}$. Po každém celém posunutí se dron zastavil na 0.3 sekund.

2.4 Zpracování dat

2.4.1 Napětí baterie



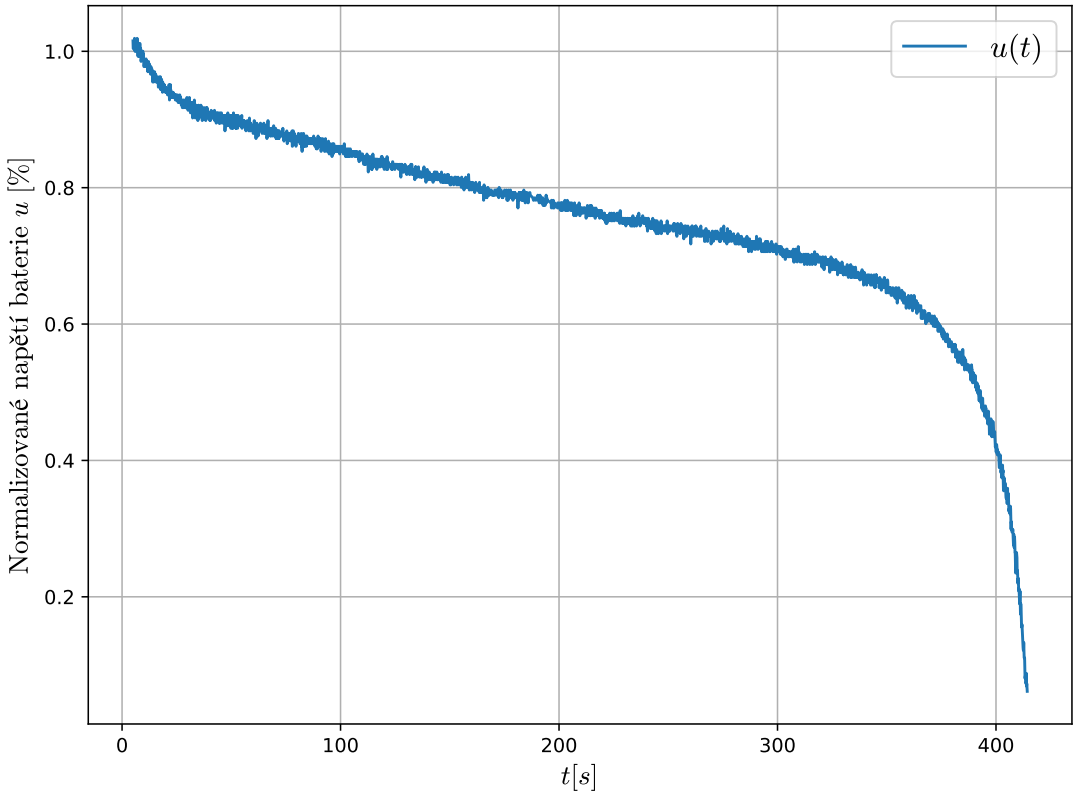
Obrázek 2.10: Napětí baterie za let 2

Naměřené hodnoty napětí baterie všech letů byly pro účely trénování modelů normalizovány na jednotný rozsah jako

$$u \longleftarrow \frac{u - u_{min}}{u_{max} - u_{min}}; \quad u_{max} = 3,7 \quad a \quad u_{min} = 2,3 \ . \tag{2.4}$$

Úprava vstupních dat pomocí škálování a odstranění extrémních hodnot může významně ovlivnit výkonnost regresních modelů, jak ukazuje srovnávací studie [14].
Z obr. 2.10 je patrné, že v počáteční fázi letu dochází k výraznějším výkyvům napětí, které jsou způsobeny vzletem dronu. Tento úsek trvá přibližně prvních

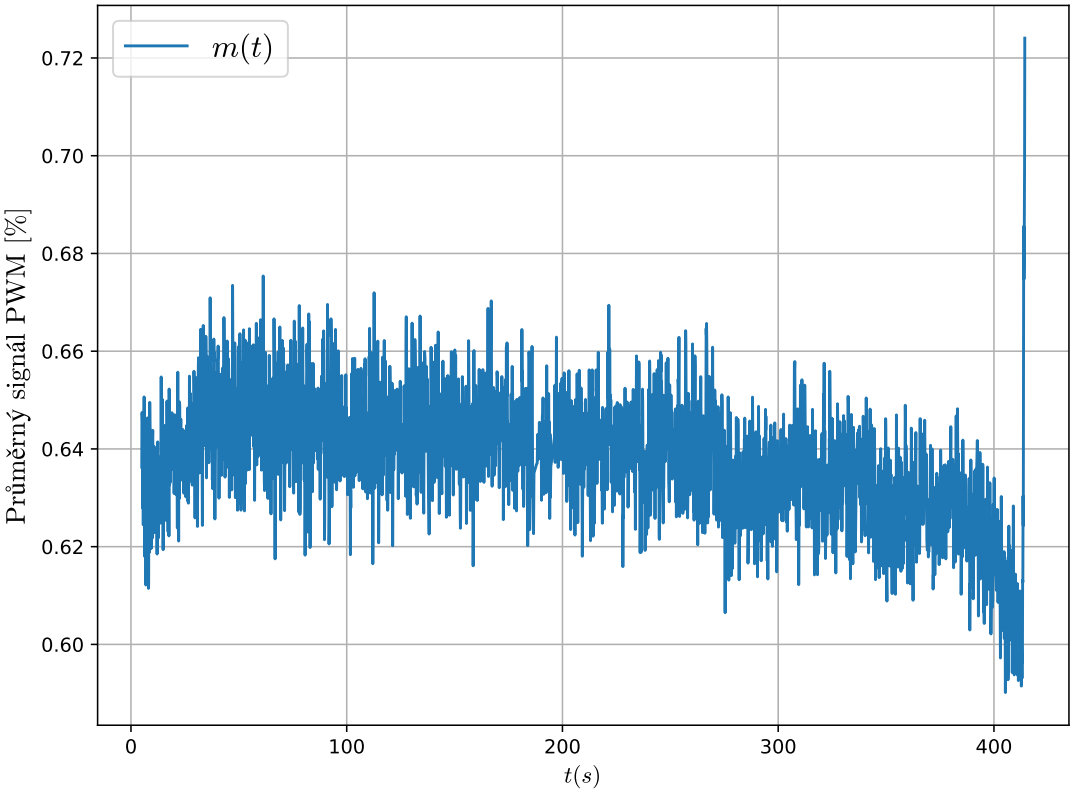
5 sekund letu a byl proto z dat odstraněn, neboť by mohl negativně ovlivnit kvalitu trénování modelu.



Obrázek 2.11: Normalizované napětí baterie za let 2 po odstranění doby vzletu dronu

2.4.2 Signál PWM

Z časové funkce PWM signálu $m(t)$ bylo, stejně jako v případě napětí baterie, odstraněno prvních 5 sekund letu. Tento úsek obsahuje výkyvy související se vzletem dronu, které mohou negativně ovlivnit učení modelu.



Obrázek 2.12: Signál PWM po odstranění prvních 5 sekund

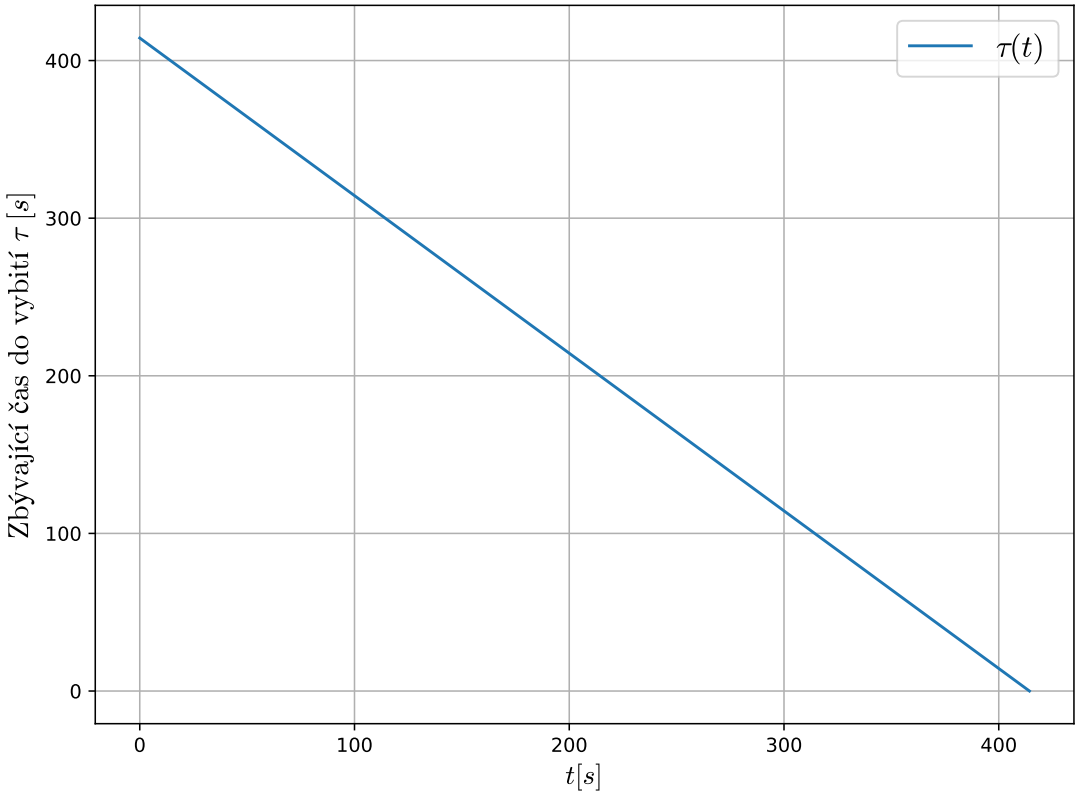
2.4.3 Zbývající čas letu

Označme si funkcí $\tau(t)$ zbývající čas letu v sekundách, tedy čas, který dronu zbývá do úplného vybití baterie a následného pádu. Tato veličina představuje cílovou (target) funkci, kterou se bude model snažit predikovat na základě vstupních dat.

- **První předpoklad**: Předpokládejme, že čas do vybití ubýha v čase lineárně, pokud se dron staticky vznáší na místě.
- **Druhý předpoklad**: Předpokládejme, že když dron bude létat pořád stejně nebo bude dělat krátké periodické manévry, baterie se bude z makroskopického hlediska vybíjet lineárně v čase.

Na základě těchto předpokladů lze funkci $\tau(t)$ definovat jako rozdíl mezi celkovou délkou letu T a aktuálním časem t jako

$$\tau = T - t \ [s]. \tag{2.5}$$

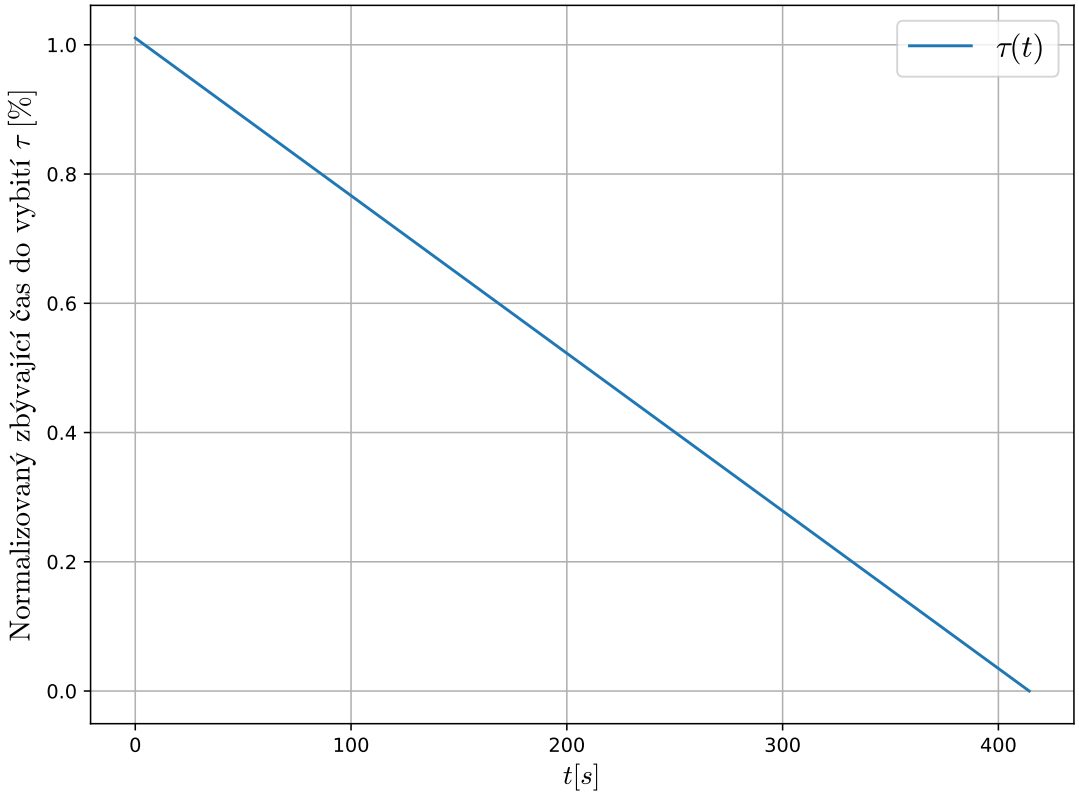


Obrázek 2.13: Zbývající čas letu

Pro zajištění stejné normalizace pro všechny jednotlivé lety, byla funkce τ normalizována pomocí konstanty T_a, která odpovídá průměrné maximální délce letu.

$$\tau \longleftarrow \frac{\tau}{T_a}; \quad T_a = 410. \tag{2.6}$$

Po zpětném vynásobení normalizované funkce konstantou T_a se získá skutečný zbývající čas letu v sekundách.



Obrázek 2.14: Zbývající čas letu po úpravě

Pozn.: Normalizace napětí i zbývajícího času letu nezaručuje, že hodnoty budou vždy mezi 0 a 1. V některých případech může být hodnota vyšší než 1, například u delších letů nebo vyššího napětí. Normalizace slouží hlavně k tomu, aby modely nepracovaly s příliš velkými čísly.

Kapitola 3

Vytváření modelů a datasetů

Predikce zbývající doby letu byla prováděna třemi způsoby.

(a) Predikce doby letu pouze z napětí baterie.

(b) Predikce doby letu z napětí baterie a PWM signálu motorů.

(c) Predikce budoucího průběhu napětí baterie až jeho pokles na prahovou hodnotu.

Pro každý zbůsob je vytvořen typ datasetu. Dataset je objekt, na kterém se model bude učit. Má informace o tom, jak vypadá vstup a cílový výstup na kterém se model bude učit.

Při převodu časové řady na trénovací dataset se často využívá metoda tzv. plovoucího okna (sliding window), využívaná i v práci [25]. Efektivitu různých predikčních modelů na časových řadách analyzuje studie [21]. Tento přístup spočívá v tom, že se přes časovou řadu posouvá pevně definované okno o délce n, které v každém kroku vybere n po sobě jdoucích hodnot jako jeden vstupní a výstupní vzorek:

- n po sobě jdoucích hodnot ze vstupní časové řady jako vstupní vektor $\mathbf{X_k}$,
- n odpovídajících hodnot z cílové výstupní řady jako výstupní vektor $\mathbf{y_k}$.

Každý řádek výsledného datasetu tedy reprezentuje mapování mezi dvěma sekvencemi délky n, kde model predikuje tvar výstupní sekvence na základě tvaru vstupní sekvence. Okno se posouvá o jeden časový krok, čímž vzniká

velký počet překrývajících se trénovacích vzorků. Celý proces je vyobrazen na obr. 3.1.

Takto vygenerované dvojice $(\mathbf{X_k}, \mathbf{y_k})$ tvoří řádky výsledného datasetu. Celý dataset lze poté zapsat ve formě tabulky 3.1,

$\mathbf{X}$	$\hat{\mathbf{y}}$
$\mathbf{X}_1$	$\hat{\mathbf{y}}_1$
$\mathbf{X}_2$	$\hat{\mathbf{y}}_2$
$\mathbf{X}_3$	$\hat{\mathbf{y}}_3$
$\vdots$	$\vdots$
$\mathbf{X}_N$	$\hat{\mathbf{y}}_N$

Tabulka 3.1: Dataset pro predikci doby letu z napětí baterie

kde $\mathbf{X} \in \mathbb{R}^{N \times n}$ je matice vstupních dat, $\mathbf{y} \in \mathbb{R}^{N \times n}$ je matice cílových výstupních hodnot a

$$N = T - n + 1, \tag{3.1}$$

je celkový počet vzorků v datasetu, který odpovídá počtu možných posunutí okna.

Na takto připraveném datasetu lze trénovat modely, jejichž cílem je naučit se mapování mezi vstupní sekvencí $\mathbf{X}_k$ a odpovídající cílovou sekvencí $\hat{\mathbf{y}}_k$

$$\text{Model} : \mathbf{X}_k \longrightarrow \hat{\mathbf{y}}_k. \tag{3.2}$$

Model se trénuje tak, aby pro každý vzorek $\mathbf{X}_k$ minimalizoval rozdíl mezi predikovaným výstupem

$$\mathbf{y}_k = \text{Model}(\mathbf{X}_k) \tag{3.3}$$

a skutečným výstupem (target) $\hat{\mathbf{y}}_k$ pomocí chybové funkce $E(\mathbf{y}_k, \hat{\mathbf{y}}_k)$.

V následujících sekcích je popsán postup tvorby datasetů pro jednotlivé způsoby predikce a následně jsou představeny modely, které byly v práci použity.

3.1 Datasety

3.1.1 Dataset pro predikci doby letu z napětí baterie

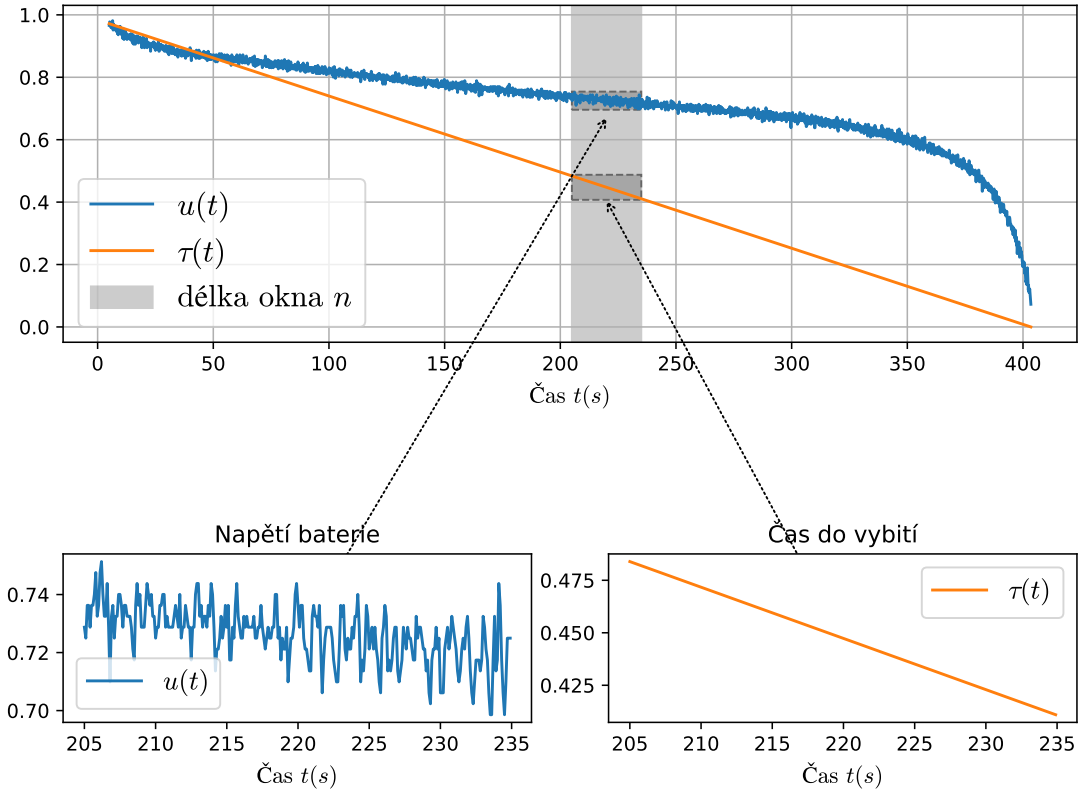
Model mapuje řady funkce napětí baterie $u(t)$ do řad funkce zbývající doby letu $\tau(t)$. Toto mapování lze formálně vyjádřit jako

$$\text{Model} : u(t) \longrightarrow \tau(t). \tag{3.4}$$

Dataset je tvořen pomocí plovoucího okna, které se posouvá po časové řadě s krokem odpovídajícím vzorkovacímu intervalu $\Delta t = 100\ ms$.
Z oken pro napětí baterie u jsou tvořeny příslušné výstupy z oken pro zbývající doby letu τ,

$$\mathbf{X}_k = \begin{pmatrix} u(t=k) \\ u(t=k+\Delta t) \\ u(t=k+2\Delta t) \\ \vdots \\ u(t=k+n\Delta t) \end{pmatrix} \qquad \hat{\mathbf{y}}_k = \begin{pmatrix} \tau(t=k) \\ \tau(t=k+\Delta t) \\ \tau(t=k+2\Delta t) \\ \vdots \\ \tau(t=k+n\Delta t) \end{pmatrix}, \tag{3.5}$$

kde k jsou celá čísla v intervalu $(0, T-n-1)$, kde T je délka letu a tedy délka funkcí $u(t)$ a $\tau(t)$ a n je délka okna. Proces je znázorněn na obr. 3.1.



Obrázek 3.1: Obrázek ukazující okno o délce n. Toto okno se pohybuje od 0 do $T - n - 1$ po jednom bodě.

3.1.2 Dataset pro predikci doby letu z napětí baterie a signálu PWM z motorů

Model nepoužívá pro predikci pouze hodnoty z napětí baterie, ale také informace ze signálů PWM z motorů, které se v literatuře běžně využívají jako vstupní proměnné při modelování spotřeby energie dronů pomocí strojového učení [10, 8, 20].
Jsou vzaty řady z oken z obou funkcí. Jsou zřetězeny do jedné řady, která slouží jako vstup do modelu. Tento způsob konstrukce vstupního vektoru odpovídá běžné praxi časového zřetězení vícekanálových vstupů, jak je využíváno i v práci [6]. Model pak mapuje tuto vstupní řadu na odpovídající sekvenci zbývající doby letu jako

$$\text{Model} : (u(t), m(t)) \longrightarrow \tau(t), \tag{3.6}$$

$$\mathbf{X}_k = \begin{pmatrix} u(t=k) \\ u(t=k+\Delta t) \\ u(t=k+2\Delta t) \\ \vdots \\ u(t=k+n\Delta t) \\ m(t=k) \\ m(t=k+\Delta t) \\ m(t=k+2\Delta t) \\ \vdots \\ m(t=k+n\Delta t) \end{pmatrix} \qquad \hat{\mathbf{y}}_k = \begin{pmatrix} \tau(t=k) \\ \tau(t=k+\Delta t) \\ \tau(t=k+2\Delta t) \\ \vdots \\ \tau(t=k+n\Delta t) \end{pmatrix}, \tag{3.7}$$

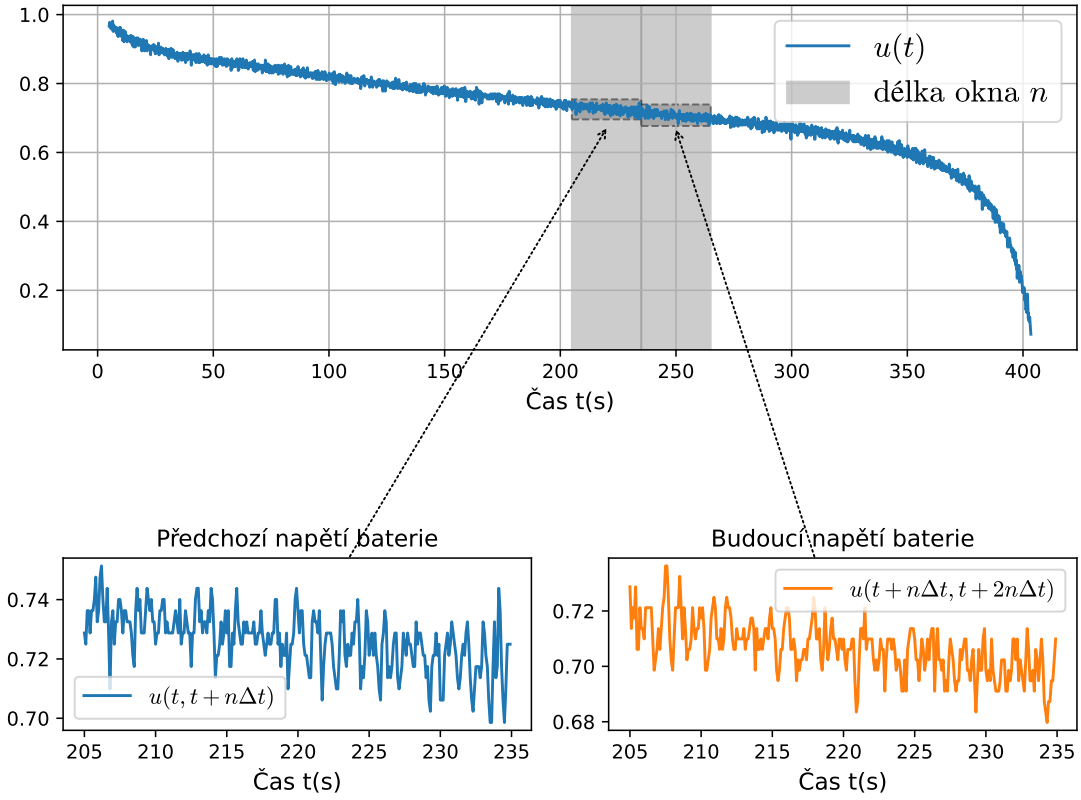
kde k jsou celá čísla v intervalu $(0, T-n-1)$, kde T je délka letu a tedy délka funkcí $u(t)$ a $\tau(t)$ a n je délka okna.

3.1.3 Dataset pro predikci budoucích hodnot napětí baterie

Tento dataset slouží k predikci budoucích hodnot funkce napětí baterie na základě jejího předchozího vývoje. Model mapuje budoucí řady funkce napětí baterie $u(t + n\Delta t, t + 2n\Delta t)$ z předchozí řady $u(t, t + n\Delta t)$. To lze formálně popsat jako

$$\text{Model} : u(t, t + n\Delta t) \longrightarrow u(t + n\Delta t, t + 2n\Delta t), \tag{3.8}$$

$$\mathbf{X}_k = \begin{pmatrix} u(t = k) \\ u(t = k + \Delta t) \\ u(t = k + 2\Delta t) \\ \vdots \\ u(t = k + n\Delta t) \end{pmatrix}, \qquad \hat{\mathbf{y}}_k = \begin{pmatrix} u(t = k + n\Delta t + 1) \\ u(t = k + n\Delta t + 2) \\ u(t = k + n\Delta t + 3) \\ \vdots \\ u(t = k + 2n\Delta t) \end{pmatrix}. \tag{3.9}$$



Obrázek 3.2: Obrázek ukazující okno o délce n. Toto okno se pohybuje od 0 do $T - n - 1$ po jednom bodě, kde T je délka letu a tedy délka funkcí $u(t)$, $m(t)$ a $\tau(t)$.

Autoregrese

Pro predikci delšího úseku budoucnosti může být model aplikován autoregresivně. To znamená, že výstupní predikce z jednoho kroku modelu se zpětně použije jako vstup pro další predikci. Tento postup je možné opakovat vícekrát za sebou a tím generovat predikci v časech, které přesahují rozsah původního vstupního okna. Takový autoregresivní přístup je běžně využíván při predikci napětí baterie, kde se výstupní hodnota rekurzivně vrací zpět na vstup modelu, jak popisuje [27].

Počet iterací takové autoregresivní predikce označíme symbolem $v \in \mathbb{N}$. Výsledný autoregresivní model tedy postupně predikuje $v \cdot n$ hodnot ze vstupních n hodnot. Každá iterace přitom generuje novou výstupní sekvenci délky n, která slouží jako vstup pro následující krok.
Celkový proces lze znázornit jako

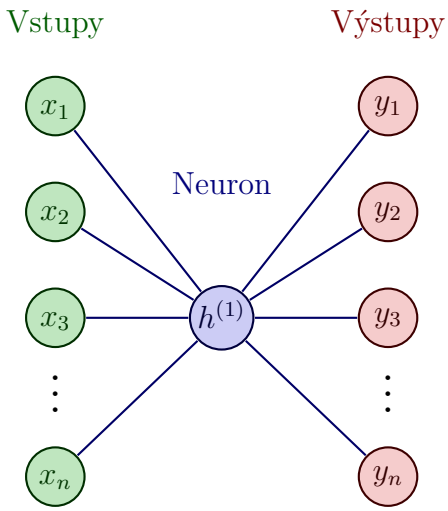
$$\mathbf{y}_1 = \text{model}(\mathbf{X}), \tag{3.10}$$
$$\mathbf{y}_2 = \text{model}(\mathbf{y}_1), \tag{3.11}$$
$$\mathbf{y}_3 = \text{model}(\mathbf{y}_2), \tag{3.12}$$
$$\vdots$$

Nevýhodou autoregresivní predikce je, že v každém kroku model pracuje s výstupem, který sám vygeneroval v předchozím kroku, nikoli se skutečnými daty. Pokud model vygeneruje nepřesnou hodnotu již na začátku řetězce, tato chyba se přenáší do dalších vstupů a může se postupně akumulovat, což vede k výrazné degradaci přesnosti predikce v pozdějších časových krocích.

3.2 Modely

3.2.1 Lineární model

Model vychází z klasického lineárního neuronového modelu typu LNU (Linear Neuron Unit), který využívá váženou sumu vstupů. Vzhledem k tomu, že cílem je generovat výstupní vektor stejné délky jako vstupní, je základní model LNU upraven tak, aby výstupní dimenze odpovídala vstupní.
Za tímto účelem je ke klasické formě přidán jednotkový vektor $\mathbf{E} \in \mathbb{R}^{1\times n}$, jehož všechny složky jsou rovny 1. Tento vektor slouží pouze k rozšíření původního skalárního výstupu na požadovaný tvar výstupního vektoru délky n. Výsledná architektura tak zachovává jednu váhu pro každý vstup.



Obrázek 3.3: Vizualizace mého používaného dvojitého LNU

Rovnice modelu je pak definována jako

$$\mathbf{y}_{1\times n} = \left(\mathbf{X}_{1\times n}\mathbf{W}^{(1)}_{n\times 1} + b^{(1)}\right) \cdot \mathbf{E}^{(1)}_{1\times n}, \tag{3.13}$$

kde

- $X \in \mathbb{R}^{1\times n}$ je vstupní vektor,
- $W^{(1)} \in \mathbb{R}^{n\times 1}$ je vektor váhových koeficientů,
- $b^{(1)} \in \mathbb{R}$ je bias,
- $\mathbf{E} \in \mathbb{R}^{1\times n}$ je jednotkový vektor pro rozšíření výstupu.

Výsledný lineární model, rozšířený o jednotkový vektor pro rozšíření výstupu, je v Pythonu implementován pomocí knihovy PyTorch, která poskytuje flexibilní prostředí pro definici a trénování modelů [19].

Skript 4 Implementace lineárního modelu v jazyce python

```
class linear(nn.Module):
    def __init__(self, input_size, output_size):
        super(linear, self).__init__()
        self.fc1 = nn.Linear(input_size, 1)
        self.register_buffer("hardcoded_matrix", torch.ones(output_size, 1))

    def forward(self, x):
        out = self.fc1(x)
        out = out @ self.hardcoded_matrix.t()
        return out
model = linear(n, n)
```

Pozn.: Při použití datasetu, ve kterém se současně využívají časové řady napětí baterie a signálu PWM, je vstupní dimenze modelu dvojnásobná oproti případům, kdy se používá pouze jeden vstupní signál. V tomto případě je tedy vstupní vektor tvořen zřetězením dvou sekvencí délky n, což vede k celkové vstupní dimenzi $2n$. Modely jsou proto konfigurovány tak, aby odpovídaly této vstupní struktuře. Například u lineárního modelu je velikost nastavena jako

$$model = linear(2n, n)$$

3.3 Kvadratický model

Kvadratický model vychází z klasické architektury typu QNU (Quadratic Neuron Unit), která rozšiřuje lineární neuron o kvadratické členy vstupů. Stejně jako v případě lineárního modelu je i zde výstup klasického QNU pouze skalární. Pro dosažení výstupního vektoru tvaru $1 \times n$, který odpovídá tvaru trénovacích dat, je model rozšířen o jednotkový vektor $\mathbf{E} \in \mathbb{R}^{1\times n}$. Tento vektor slouží k replikaci skalárního výstupu do vektorového tvaru požadované délky, aniž by ovlivnil samotné chování modelu.

$$y = \left(\mathbf{X}_{(1\times n)}\,\mathbf{W}^{(1)}_{(n\times 1)} + \mathbf{X}_{(1\times n)}\,\mathbf{V}_{(n\times n)}\,\mathbf{X}^{T}_{(n\times 1)} + \; b^{(1)}_{(1\times 1)}\right) \cdot \mathbf{E}^{(1)}_{(1\times n)} \tag{3.14}$$

Následující výpis ukazuje implementaci kvadratického modelu v jazyce Python s využitím knihovny PyTorch.

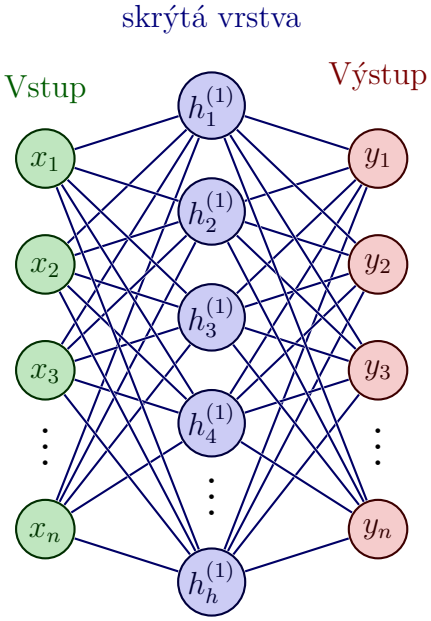
Skript 5 Implementace kvadratického modelu v jazyce python

```
class quadratic(nn.Module):
    def __init__(self, input_size, output_size):
        super(quadratic, self).__init__()
        self.fc1 = QNU(input_size, 1)
        self.register_buffer("hardcoded_matrix", torch.ones(output_size, 1))

    def forward(self, x):
        out = self.fc1(x)
        out = out @ self.hardcoded_matrix.t()
        return out
model = quadratic(n, n)
```

3.3.1 MLP model

Model MLP je v této práci použit ve své základní podobě, jak je běžně uváděna v literatuře. Model se skládá z jedné skryté vrstvy o dimenzi $(1 \times h)$, na kterou navazuje výstupní vrstva odpovídající délce výstupního vektoru. Jako aktivační funkce je ve skryté vrstvě použita funkce ReLU (Rectified Linear Unit), která je v některých experimentech nahrazována jinými funkcemi. Architektura použité sítě je znázorněna na obrázku 3.4.



Obrázek 3.4: Schéma neuronové sítě modelu MLP

Rovnice modelu lze zapsat jako

$$\mathbf{h}^{(1)}_{(1\times h)} = \mathrm{ReLU}\left(\mathbf{W}^{(1)}\mathbf{X} + \mathbf{b}^{(1)}\right), \tag{3.15}$$

$$\mathbf{y}_{(1\times n)} = \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}, \tag{3.16}$$

kde

- $X \in \mathbb{R}^{1\times n}$ je vstupní vektor,
- $W^{(1)}, W^{(2)}$ jsou váhové matice jednotlivých vrstev,
- $b^{(1)}, b^{(2)}$ jsou biasy,
- $h^{(1)} \in \mathbb{R}^{1\times h}$ je výstup ze skryté vrstvy po aplikaci aktivační funkce,
- y je výstup modelu o dimenzi $(1 \times n)$.

Skript 6 Implementace modelu v jazyce Python s využitím knihovny PyTorch

```
class MLP(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, output_size)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

model = MLP(n, h, n)
```

Kapitola 4

Testování modelů

Cílem této kapitoly je vyhodnotit přesnost navržených modelů za různých experimentálních podmínek. Testování se zaměřuje především na vliv délky vstupního okna n. U modelu typu MLP jsou navíc analyzovány také parametry velikost skryté vrstvy h a různé aktivační funkce σ.

Pro účely základního porovnání je zvolena následující výchozí konfigurace trénovacích parametrů.

Batch size	16
Learning rate (LR)	10^{-4}
Počet epoch	100
Učící se algoritmus	Adam

Tabulka 4.1: Výchozí trénovací parametry použité pro testování modelů

Tato konfigurace byla zvolena jako vyvážený kompromis mezi rychlostí a stabilitou trénování. Algoritmus Adam byl vybrán díky své schopnosti adaptivně upravovat velikost kroku učení pro jednotlivé parametry, což zajišťuje stabilnější trénink i při menších datasetech.

Při každém testu budou systematicky měněny pouze vybrané parametry, zatímco ostatní budou zachovány. Pro každý experiment budou zaznamenány

- konkrétní hodnoty použitých parametrů (pokud se liší od výchozích),
- chyba MSE a koeficient determinace R^2 pro trénovací i testovací data,
- čas trénování modelu.

Chyba MSE vyjadřuje, jak moc se predikované hodnoty v průměru liší od skutečných, přičemž větší chyby jsou díky druhé mocnině výrazně penalizovány. Je užitečná hlavně tehdy, když chceme model, který se co nejvíce přibližuje skutečnosti a zároveň se vyhýbá velkým odchylkám.
Koeficient determinace R^2 slouží jako doplňková metrika k MSE a vyjadřuje přesnost predikce vzhledem k rozptylu skutečných hodnot. Hodnota R^2 se pohybuje od $-\infty$ do 1, přičemž čím blíže je k 1, tím lépe model odpovídá skutečným datům. Tyto metriky patří k běžným standardům při hodnocení regrese, jak o tom píše [23].

Trénovací a testovací sady pokrývají odlišné režimy chování dronu jako je statický let, vertikální pohyb (stoupání/sestup) a horizontální pohyb. Zároveň je zajištěno, že žádný z modelů není testován na datech, která byla použita při jeho trénování.

Trénovací data	**Popis letu**	**Testovací data**	**Popis letu**
Let 1	Statické vznášení na místě.	Let 2	Statické vznášení na místě.
Let 3	Vertikální pohyb o 0.5 m a rychlostí 0.4 ms^{-1}.	Let 4	Vertikální pohyb o 0.3 m a rychlostí 0.2 ms^{-1}
Let 5	Horizontální pohyb o 1.5 m a rychlostí 0.5 ms^{-1}.	Let 6	Horizontální pohyb o 2 m a rychlostí 1 ms^{-1}.

Tabulka 4.2: Rozdělení letových dat a stručný popis letových režimů

Na následujících grafech a v tabulkách jsou znázorněny výsledky predikce zbývajícího času do vybití baterie pomocí různých modelů strojového učení. Testování probíhá ve třech letových scénářích jak je ukázáno v tab. 4.2. Každý řádek grafů odpovídá jednomu typu modelu, zatímco jednotlivé sloupce reprezentují odlišné letové režimy. V grafech jsou vždy zobrazeny funkce

- $u(t)$ – normalizované napětí baterie,
- $\hat{\tau}(t)$ – skutečný zbývající čas letu (target),
- $y(t)$ – predikovaný zbývající čas letu (výstup modelu).

V testovacích tabulkách níže jsou vždy zvýrazněny testy s nejnižší chybou (MSE). Následující grafy pak zobrazují výstup těchto nejlepších variant modelů na testovacích datech.

4.1 Predikce času do vybití z napětí baterie

Pro lineární model:

n	LR	MSE $\cdot 10^3$		$R^2 \cdot 10^3$		čas
		train	test	train	test	trénování (s)
20	10^{-4}	13.49	13.63	827.69	825.26	36.5
40	10^{-4}	13.18	12.86	831.1	834.64	36.5
60	10^{-4}	12.19	14.22	843.8	817.18	36.6
80	10^{-4}	12.52	14.44	838.51	814.28	36.6

Tabulka 4.3: Parametry a výsledky pro lineární model

Pro kvadratický model:

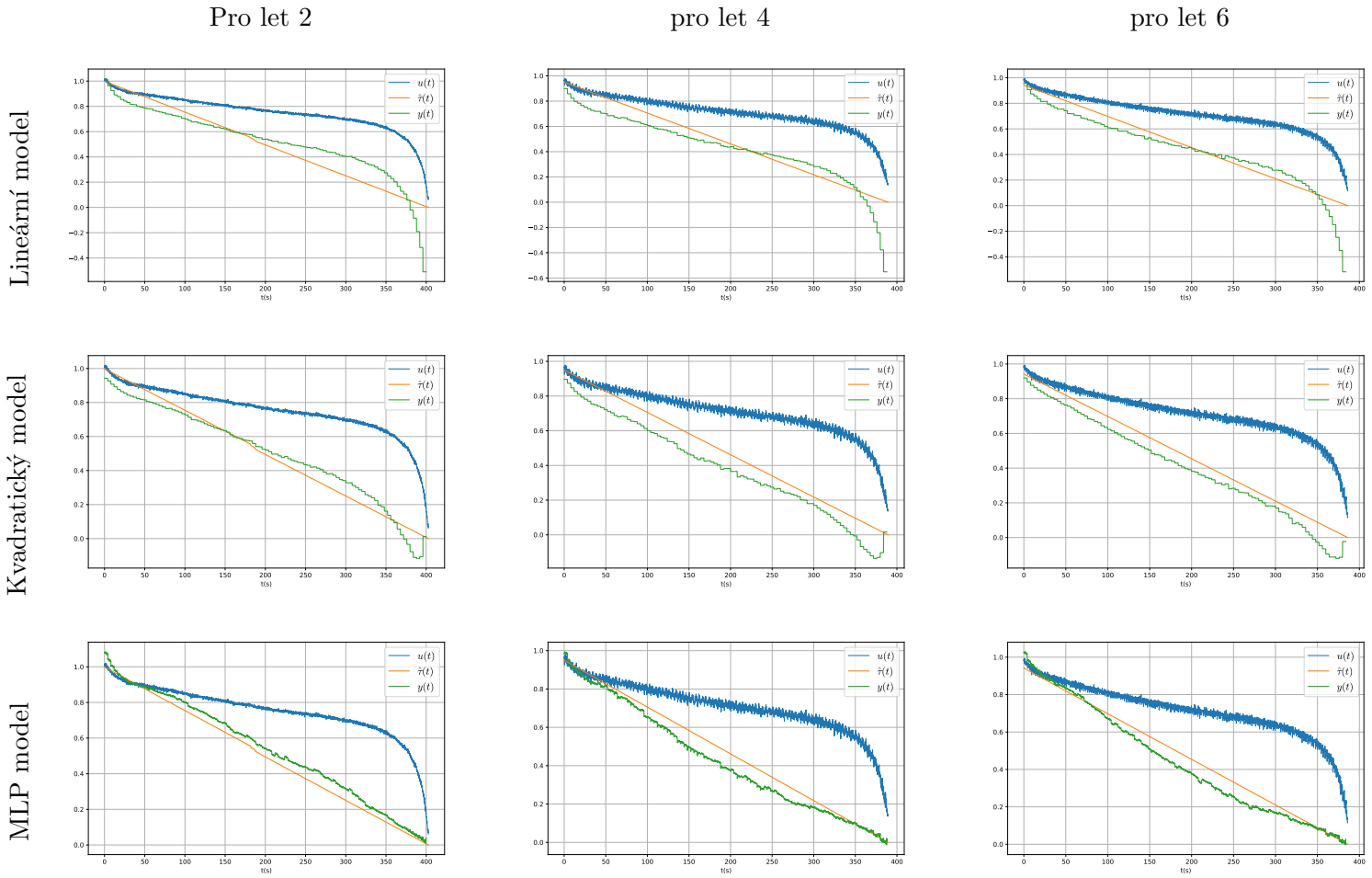
n	LR	MSE $\cdot 10^3$		$R^2 \cdot 10^3$		čas
		train	test	train	test	trénování (s)
20	10^{-4}	4.56	7.2	941.68	907.77	58.6
40	10^{-4}	3.89	4.52	950.19	941.93	56.7
60	10^{-4}	3.97	5.41	949.14	930.47	55.7
80	10^{-4}	4.25	5.06	945.2	934.92	59.6

Tabulka 4.4: Parametry a výsledky pro kvadratický model

Pro MLP model:

n	h	LR	σ	MSE $\cdot 10^3$		$R^2 \cdot 10^3$		čas
				train	test	train	test	trénování (s)
20	10	10^{-3}	ReLU	1.3	4.2	983.35	946.1	49.8
40	10	10^{-3}	ReLU	1.35	3.92	982.7	949.57	49.9
60	10	10^{-3}	ReLU	1.7	3.54	978.16	954.49	50.4
20	40	10^{-3}	ReLU	1.04	2.93	986.67	962.38	50.4
40	40	10^{-3}	ReLU	1.2	2.85	984.56	963.33	51.1
60	40	10^{-3}	ReLU	1.46	3.03	981.26	961.08	52.0
20	70	10^{-3}	ReLU	0.83	3.09	989.4	960.37	51.5
40	70	10^{-3}	ReLU	1.42	2.68	981.79	965.59	51.8
60	70	10^{-3}	ReLU	1.1	3.59	985.92	953.79	52.8

Tabulka 4.5: Parametry a výsledky pro MLP model



Obrázek 4.1: Parametry a výsledky pro predikci z napětí baterie

Z obr. 4.1 je patrné, že **lineární model** nedokáže dostatečně zachytit nelineární vztahy mezi napětím a zbývající dobou letu. Přesnost jeho predikce se výrazně zhoršuje v pozdějších fázích letu, kdy napětí klesá rychleji. **Kvadratický model** již vykazuje podstatně lepší shodu s realitou, a to zejména v dynamických letových režimech, kde model lépe reflektuje změny ve vybíjecí křivce. Nejlepších výsledků však dosahuje **MLP model**, který je schopen velmi přesně rekonstruovat průběh zbývajícího času letu ve všech testovacích scénářích. Celkově lze tedy konstatovat, že i samotné napětí baterie jako jediný vstupní parametr postačuje k velmi přesné predikci zbývající doby letu, za předpokladu použití dostatečně expresivního modelu, jako je vícevrstvá neuronová síť.

4.2 Predikce času do vybití z napětí baterie a signálu PWN z motorů

Pro lineární model:

n	LR	MSE $\cdot 10^3$		$R^2 \cdot 10^3$		čas
		train	test	train	test	trénování (s)
10	10^{-4}	14.66	15.21	813.03	805.01	46.8
30	10^{-4}	13.36	16.78	829.61	784.22	46.5
50	10^{-4}	13.12	16.94	832.35	781.84	47.0

Tabulka 4.6: Parametry a výsledky pro lineární model

Pro kvadratický model:

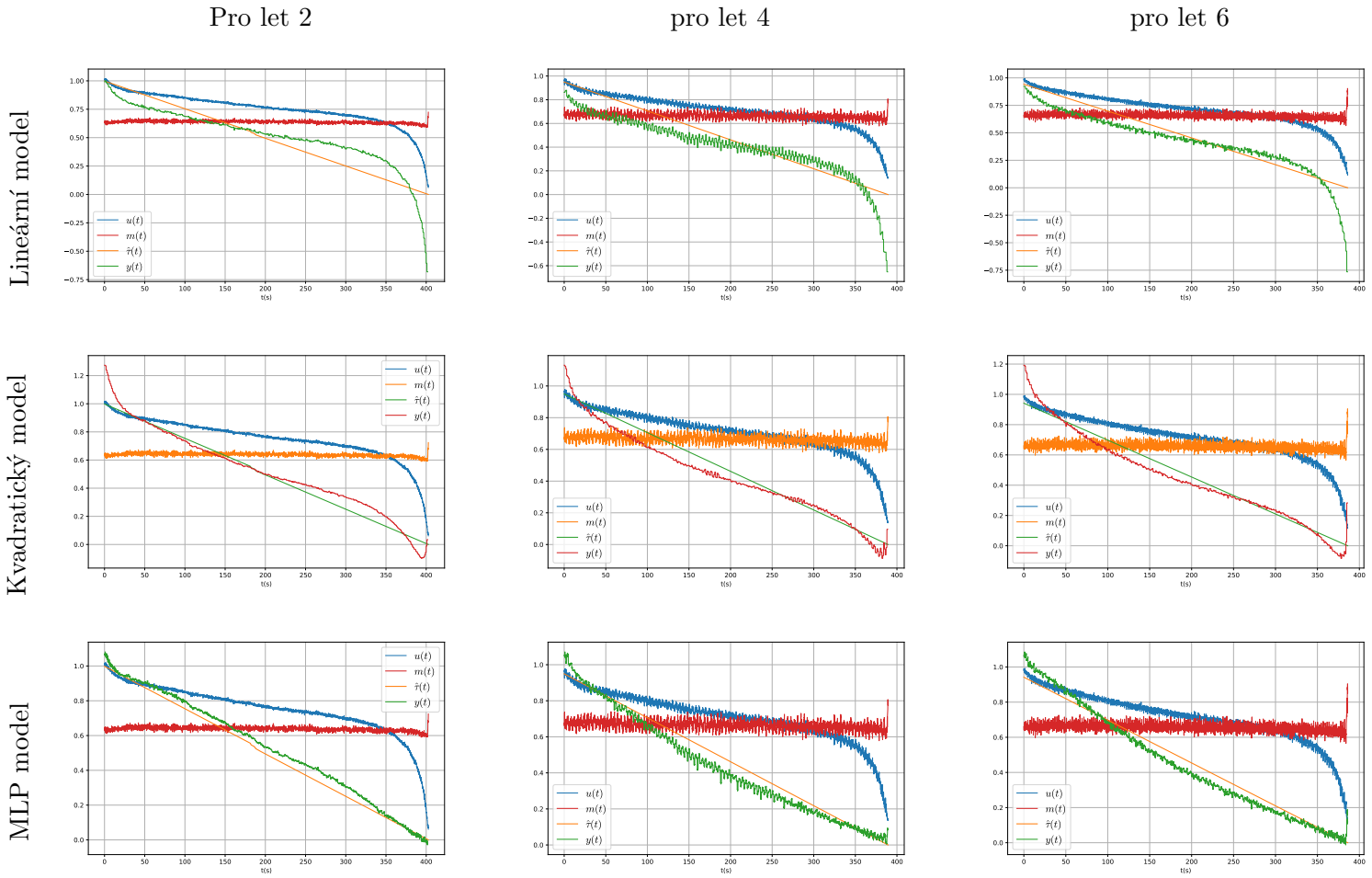
n	LR	MSE $\cdot 10^3$		$R^2 \cdot 10^3$		čas
		train	test	train	test	trénování (s)
10	10^{-4}	3.74	3.72	952.3	952.29	61.2
30	10^{-4}	4.03	5.09	948.62	934.57	63.1
50	10^{-4}	6.75	6.65	913.82	914.28	68.6

Tabulka 4.7: Parametry a výsledky pro kvadratický model

Pro MLP model:

n	h	LR	σ	MSE $\cdot 10^3$		$R^2 \cdot 10^3$		čas
				train	test	train	test	trénování (s)
10	20	10^{-4}	ReLU	1.76	2.96	977.6	962.03	58.8
30	20	10^{-4}	ReLU	1.21	10.01	984.52	871.28	59.3
50	20	10^{-4}	ReLU	1.35	9.93	982.77	872.1	59.4
10	40	10^{-4}	ReLU	1.35	2.77	982.76	964.52	59.2
30	40	10^{-4}	ReLU	1.12	8.4	985.67	891.97	60.8
50	40	10^{-4}	ReLU	1.24	8.61	984.12	889.15	61.4
10	60	10^{-4}	ReLU	1.16	2.38	985.2	969.46	59.5
30	60	10^{-4}	ReLU	1.01	8.11	987.17	895.66	60.2
50	60	10^{-4}	ReLU	1.04	8.83	986.73	886.29	61.4

Tabulka 4.8: Parametry a výsledky pro MLP model



Obrázek 4.2: Parametry a výsledky pro predikci z napětí baterie a PWM signálu z motorů

Na obr 4.2 jsou zobrazeny výsledky predikce zbývajícího času letu při použití jak napětí baterie, tak signálu PWM z motorů jako vstupních veličin. Ve všech grafech je vidět, že přidání PWM signálu zlepšilo predikční schopnosti modelů, především u kvadratického a MLP modelu. Lineární model má stále problémy zachytit přesnější průběh vývoje zbývající doby letu. Nejlépe si i v tomto scénáři vede model MLP, jehož predikce téměř přesně kopíruje průběh skutečné hodnoty zbývajícího času $\hat{\tau}(t)$.

4.3 Predikce budoucích hodnot napětí baterie

Pro predikci budoucích hodnot napětí baterie byl zaveden parametr v, který určuje počet iterací autoregresivní predikce, jak je podrobněji popsáno v sekci 3.1.3. Hodnota v byla pro každý model volena specificky, aby byl dosažen co nejdelší predikční horizont bez výrazné akumulace chyby, která by mohla negativně ovlivnit výsledky modelu. Pro každý model byly vybrány dvě hodnoty parametru v, což umožnilo posoudit vliv rostoucího počtu autoregresivních iterací na přesnosti predikce. Pro každou hodnotu v byla vyhodnocena střední kvadratická chyba (MSE) a koeficient determinace R^2.

Pro lineární model:

		$v = 5$				$v = 10$				
n	LR	MSE $\cdot 10^3$		$R^2 \cdot 10^3$		MSE $\cdot 10^3$		$R^2 \cdot 10^3$		čas
		train	test	train	test	train	test	train	test	trénování (s)
30	10^{-4}	1.18	1.0	934.67	949.48	3.04	3.35	831.24	831.19	37.5
50	10^{-4}	2.15	2.19	879.45	888.8	7.88	8.34	557.88	575.68	37.4
70	10^{-4}	3.38	3.8	808.86	804.77	19.43	24.69	−99.26	−268.73	37.9

Tabulka 4.9: Parametry a výsledky pro lineární model

Pro kvadratický model:

		$v = 10$				$v = 20$				
n	LR	MSE $\cdot 10^3$		$R^2 \cdot 10^3$		MSE $\cdot 10^3$		$R^2 \cdot 10^3$		čas
		train	test	train	test	train	test	train	test	trénování (s)
30	10^{-4}	1.05	1.31	941.95	934.04	4.68	142.26	739.86	−6163.19	52.2
40	10^{-4}	2.28	2.78	872.83	859.18	14.04	23.0	216.85	−164.19	51.8
50	10^{-4}	4.94	8.69	723.1	557.89	45.62	84.46	−1559.24	−3297.24	55.3

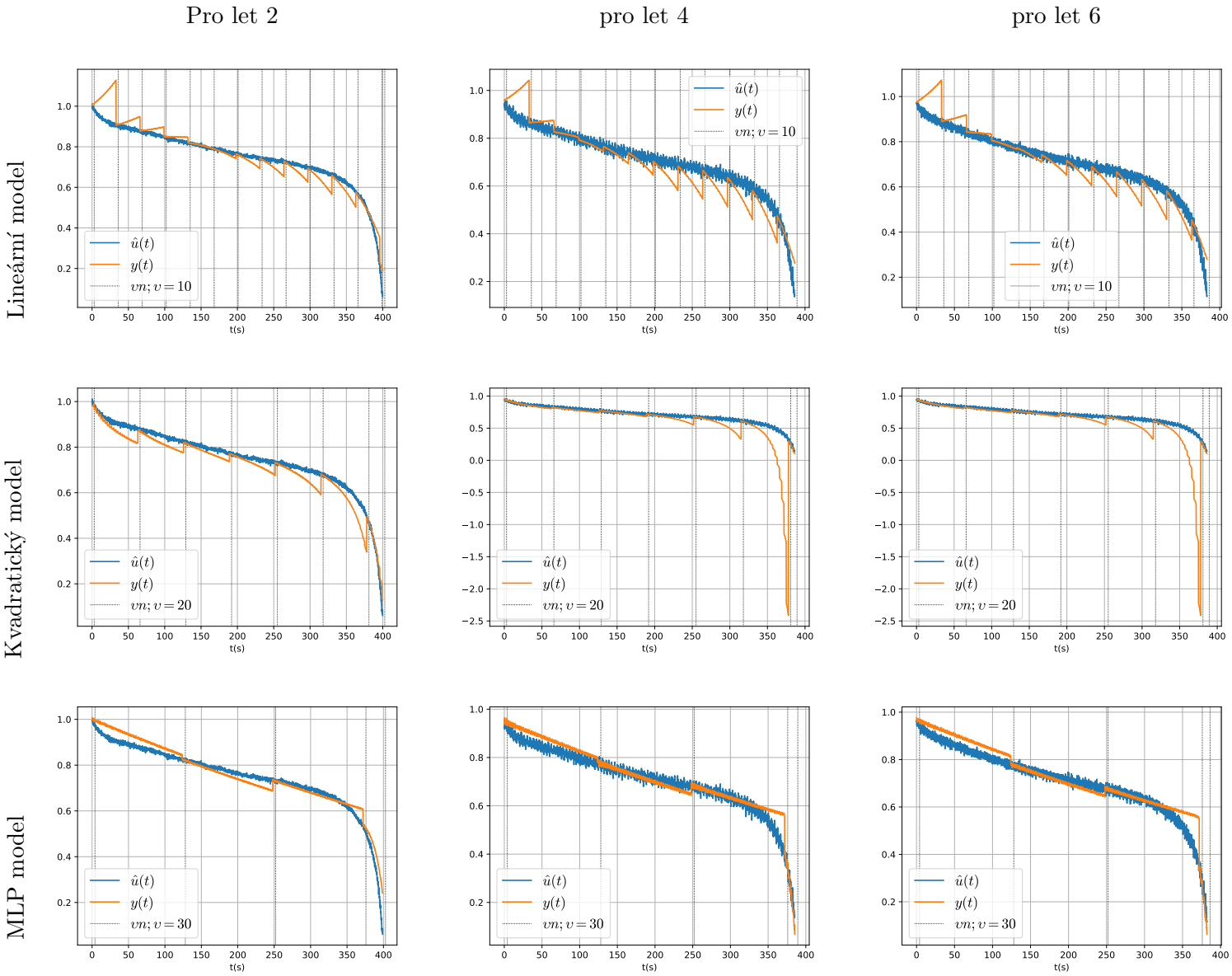
Tabulka 4.10: Parametry a výsledky pro kvadratický model

Pro MLP model:

				$v = 20$				$v = 30$				
n	h	LR	σ	MSE $\cdot 10^3$		$R^2 \cdot 10^3$		MSE $\cdot 10^3$		$R^2 \cdot 10^3$		čas
				train	test	train	test	train	test	train	test	trénování (s)
30	40	10^{-4}	ReLU	1.34	1.71	925.55	913.85	1.53	1.87	915.24	906.08	42.7
40	40	10^{-4}	ReLU	58.53	71.01	−2265.43	−2594.88	429.39	538.62	−22954.75	−26267.21	42.0
50	40	10^{-4}	ReLU	8.3	21.71	534.14	−104.57	7.99	31.0	551.99	−577.14	41.8
30	60	10^{-4}	ReLU	1.5	3.37	916.69	830.16	1.87	2.33	896.04	882.6	42.4
40	60	10^{-4}	ReLU	3.88	3.03	783.49	846.79	1.14	1.44	936.56	927.16	46.3
50	60	10^{-4}	ReLU	2.7	7.61	848.7	613.06	2.97	14.22	833.59	276.71	46.4
30	40	10^{-4}	tanh	1.11	36.54	938.47	−839.7	4.23	104.33	765.26	−4253.42	45.4
40	40	10^{-4}	tanh	4.38	10.49	755.45	468.73	14.81	156.57	173.58	−6926.54	43.9
50	40	10^{-4}	tanh	12.22	37.85	314.29	−925.9	10.56	60.18	407.62	−2062.03	43.8
30	60	10^{-4}	tanh	2.12	49.72	882.3	−1503.71	15.09	150.42	162.01	−6574.36	45.6
40	60	10^{-4}	tanh	2.72	7.52	848.38	619.14	0.73	87.42	959.46	−3425.55	46.2
50	60	10^{-4}	tanh	4.77	16.3	732.14	170.49	5.95	36.37	666.15	−850.22	47.2

Tabulka 4.11: Parametry a výsledky pro MLP model

V rámci experimentů v tab. 4.11 byly testovány dvě aktivační funkce `ReLU` a `tanh`. Cílem bylo zjistit, zda typ použité nelinearity ovlivní přesnost predikce. Výsledky ukazují, že ReLU dosahuje o něco lepších výsledků. Nejlepších výsledků dosáhl model MLP s délkou okna $n = 40$ a velikostí skryté vrstvy $h = 60$. Z tabulky je však zároveň patrné, že při parametru autoregrese $v = 30$ je chyba v některých případech nižší než při $v = 20$, což nedává smysl, protože s rostoucím v by se měla chyba spíše zvětšovat. Tato skutečnost poukazuje na to, že pro větší v predikce pomocí autoregrese není dostatečně přesná. Velikost chyby tedy závisí spíše na tom, zda se model „trefí“ do správného trendu v dané iteraci.



Obrázek 4.3: Parametry a výsledky pro predikci z napětí baterie

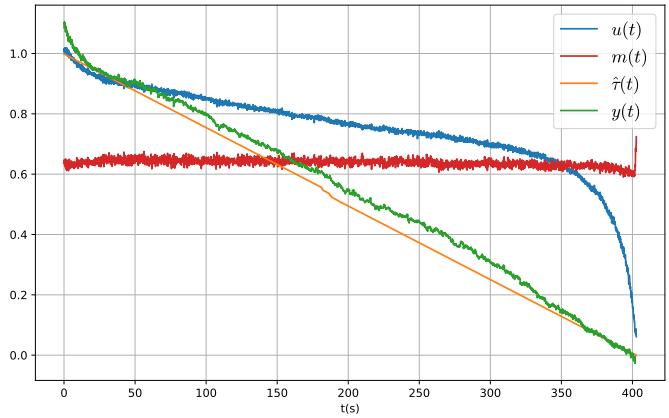
4.4 Výsledný model a implementace

Z experimentů vyplývá, že predikce budoucích hodnot napětí baterie není příliš spolehlivá. Dochází k výrazné akumulaci chyb, a to i u složitějších modelů. Technika autoregrese se proto neukázala jako vhodná pro predikci celého průběhu vybíjecí křivky.
Nejlepších výsledků bylo dosaženo při predikci zbývajícího času do vybití baterie na základě napětí a signálu PWM. Nejlépe si vedl model MLP s délkou okna $n = 10$ a velikostí skryté vrstvy $h = 60$.
Tento model byl dále testován. Pro zvýšení přesnosti byl počet trénovacích epoch navýšen na 200. Zároveň byla pro zrychlení výpočtu mírně zmenšena velikost skryté vrstvy na $h = 50$.
Dále byl model porovnán při použití dvou různých aktivačních funkcí *ReLU* a *tanh*.

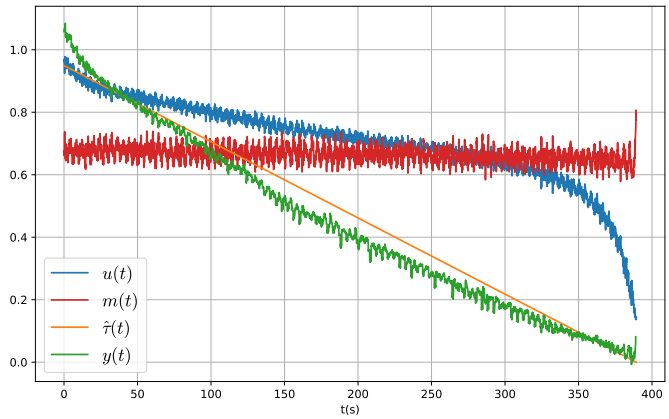
n	h	LR	σ	MSE $\cdot 10^3$		$R^2 \cdot 10^3$		čas
				train	test	train	test	trénování (s)
10	50	10^{-4}	ReLU	1.1	2.18	985.92	972.01	123.4
10	50	10^{-4}	tanh	1.66	2.49	978.82	968.07	118.9

Tabulka 4.12: Caption

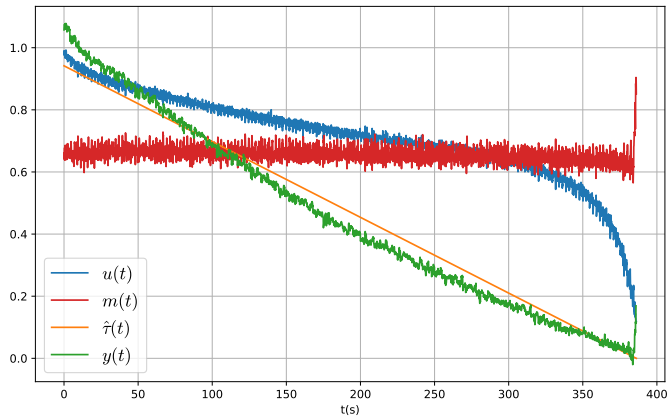
Tento finální model byl dále důkladně testován s navýšeným počtem trénovacích epoch na 200, což vedlo ke zvýšení přesnosti predikce. V rámci experimentování s různými aktivačními funkcemi se jako nejvhodnější ukázala funkce *ReLU*, která dosahovala nejnižší průměrné chyby v porovnání s *tanh*. Výsledný model s parametry $n = 10$ a $h = 50$, trénovaný s *ReLU* aktivací, dosáhl střední kvardatické chyby MSE $= 2.18 \cdot 10^{-3}$ a koeficientu determinace $R^2 = 972.01 \cdot 10^{-3}$ na testovací množině.



(a) Finální model testovaný na datech letu 2



(b) Finální model testovaný na datech letu 4



(c) Finální model testovaný na datech letu 6

Obrázek 4.4: Testy finálního modelu

Během trénování modelů byla pravidelně sledována hodnota ztrátové funkce (MSE), která udává průměrnou chybu predikce modelu na trénovacích datech. Každých 10 epoch byl zaznamenán průměrný výstup chyby, což umožnilo průběžně sledovat, zda se model stále zlepšuje nebo zda nevznikají velké oscilace v chybě či jiné problémy s učením. Výstup z trénovací smyčky finálního modelu je vyobrazen níže.

```
Epoch [10/200], Average Loss: 0.00576737
Epoch [20/200], Average Loss: 0.00329506
Epoch [30/200], Average Loss: 0.00220535
Epoch [40/200], Average Loss: 0.00175334
Epoch [50/200], Average Loss: 0.00155402
Epoch [60/200], Average Loss: 0.00144704
Epoch [70/200], Average Loss: 0.00138168
Epoch [80/200], Average Loss: 0.00133548
Epoch [90/200], Average Loss: 0.00128120
Epoch [100/200], Average Loss: 0.00124078
Epoch [110/200], Average Loss: 0.00120300
Epoch [120/200], Average Loss: 0.00118624
Epoch [130/200], Average Loss: 0.00114672
Epoch [140/200], Average Loss: 0.00112648
Epoch [150/200], Average Loss: 0.00111026
Epoch [160/200], Average Loss: 0.00109591
Epoch [170/200], Average Loss: 0.00109254
Epoch [180/200], Average Loss: 0.00107228
Epoch [190/200], Average Loss: 0.00106504
Epoch [200/200], Average Loss: 0.00104703
Training completed in 123.4142 seconds
```

Samotné sledování chyby pouze na trénovacích datech však neumožňuje odhalit přetrénování modelu (overfitting). To nastává tehdy, když se model naučí trénovací data příliš přesně a tím ztratí schopnost generalizace na nová, dosud neviděná data. Pro rozpoznání přetrénování je proto nutné na konci trénování vyhodnotit výkon modelu také na oddělené testovací sadě. Typickým příznakem přetrénovaného modelu je výrazně nižší chyba na trénovacích datech než na testovacích datech, jak popisuje práce zaměřená na praktické aspekty tohoto problému [18].
Například v tab. 4.8, kde MLP model s parametry $n = 30$, $h = 20$ vykazuje velmi nízkou chybu na trénovacích datech ($\text{MSE}_{\text{train}} = 1.21 \cdot 10^{-3}$), ale výrazně vyšší chybu na testovacích datech ($\text{MSE}_{\text{test}} = 10.01 \cdot 10^{-3}$).

Tento natrénovaný model byl uložen ve formátu PyTorch jako `final_model.pth`.

Skript 7 Uložení finálního modelu ve formátu .pth

```
checkpoint = {
        'seq_length': n,
        'input_size': 2*n,
        'hidden_size': k,
        'output_size': n,
        'cutoff': cutoff,
        'sig_norm': sig_norm,
        'sec_norm': sec_norm,
        'state_dict': model.state_dict(),
    }
    torch.save(checkpoint, 'final_model.pth')
```

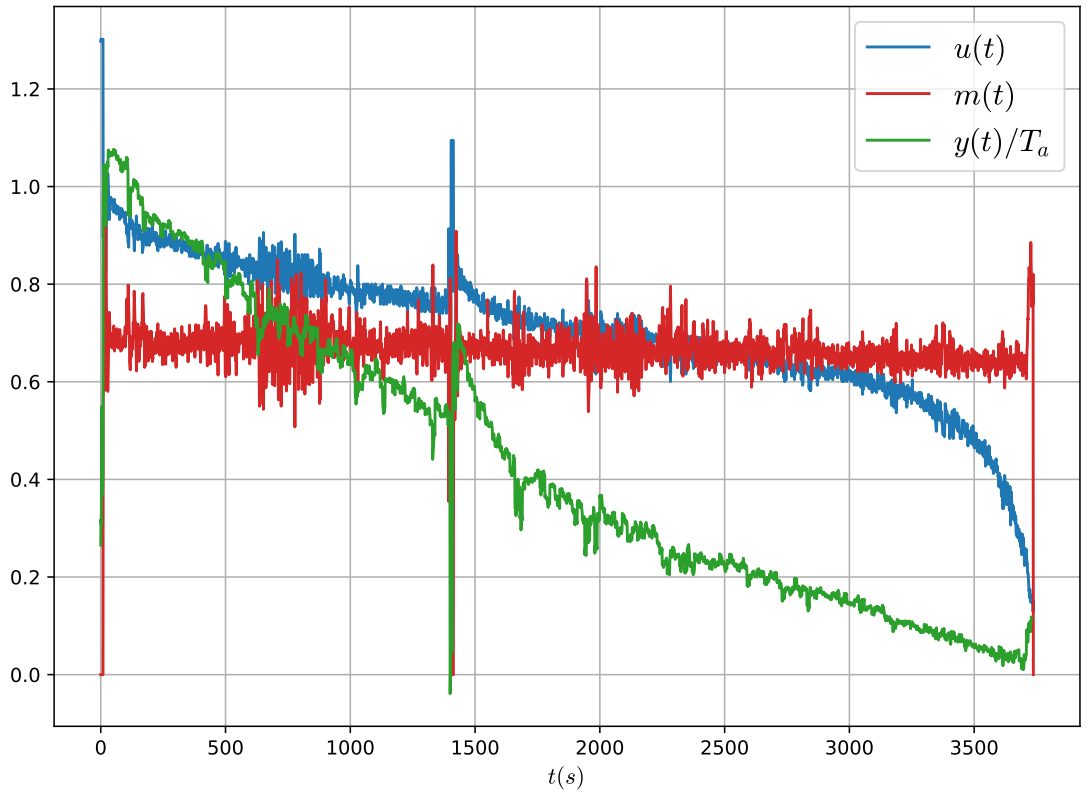
Pro účely praktického nasazení byl model implementován do Python skriptu, který ovládá dron. Na základě vstupů z napětí baterie a PWM signálu z motorů, model vypočítá aktuální odhad zbývajícího času do vybití baterie, který je vyjádřen v sekundách.

Skript 8 Úryvek skriptu, ve kterém byl model implementován

```
def run_model(self, u_seq, m_seq, timestamp):
        u_seq = self.normalize_u(np.array(u_seq))
        m_seq = np.array(m_seq)
        data = np.concatenate((u_seq, m_seq))
        test_input = torch.tensor(data, dtype=torch.float32).unsqueeze(0)
        with torch.no_grad():
            pred = self.model(test_input).squeeze().numpy()
            pred_sec = pred * self.sec_norm #sec_norm = T_a
            self.g = np.append(self.g, pred_sec)
            self.signal = np.append(self.signal, u_seq)
            print(f"prediction: {pred_sec.mean():.2f} s")
            self.write_to_file(timestamp, pred_sec)
```

Model byl dále otestován při reálném letu, kdy byl dron ovládán manuálně pomocí joysticku. Cílem bylo ověřit, zda model dokáže i v méně strukturovaném letovém režimu správně predikovat zbývající čas do vybití baterie.

Během letu byl výstup modelu průběžně sledován. Jakmile predikovaný čas poklesl přibližně na hodnotu 200 sekund, byl dron manuálně přistán. Po krátké pauze byl znovu proveden vzlet a bylo pozorováno, zda model pokračuje v predikci s ohledem na nový průběh napětí.
Na následujícím grafu je zobrazen celý průběh tohoto experimentálního letu. Výstup modelu byl normalizován hodnotou T_a, protože program vrací zbývající čas letu v sekundách.



Obrázek 4.5: Průběh reálného testovacího letu s predikcí zbývajícího času do vybití.

Kapitola 5

Závěr a diskuze

5.1 Závěr

Tato bakalářská práce se zabývala návrhem systému pro predikci parametrů letu malého kvadrokoptérového dronu pomocí metod strojového učení. Hlavním cílem bylo vytvořit modely schopné odhadnout zbývající dobu letu na základě dat získaných z napětí baterie a řídicích PWM signálů motorů.
V rámci práce byly navrženy a otestovány tři různé přístupy k predikci, konkrétně predikce pouze z napětí, kombinace napětí a PWM signálů a predikce budoucího vývoje napětí pomocí autoregrese. Pro každý přístup byly vytvořeny příslušné datasety a testovány různé varianty modelů. Nejlepších výsledků bylo dosaženo při predikci zbývající doby letu na základě kombinace napětí a PWM signálu, která vedla k nejnižší chybě predikce v různých letových scénářích. Přesto se ukázalo, že i predikce založená pouze na napětí pomocí MLP modelu poskytovala velmi dobré výsledky a vzhledem ke své jednoduchosti by v mnoha případech zcela postačovala k řešení dané úlohy. Naopak predikce pomocí autoregrese se ukázala jako méně vhodná. Tento přístup, který využívá výstup modelu jako vstup pro další krok, vedl k rychlé akumulaci chyb a ztrátě přesnosti při delších predikčních horizontech. Pro tento typ úlohy by bylo vhodnější použít modely určené pro sekvenční data, například RNN nebo LSTM, které umožňují zpracování celé časové řady bez nutnosti postupného navazování výstupů. Autoregresivní metoda se tedy ukázala jako nejméně spolehlivá a představuje jednu z hlavních oblastí pro zlepšení.
Během měření byly kromě základních senzorických dat zaznamenávány také prostorové souřadnice dronu, tedy jeho trajektorie. Tato data nebyla v rámci této práce dále využita, ale jsou uložena a připravena k budoucímu zpra-

cování, například pro výpočet energeticky nejefektivnější trajektorie nebo rozšíření modelu o prostorový kontext letu.
Závěrem lze říci, že i se základními prostředky bylo možné vytvořit funkční systém schopný predikce doby letu. Přes dosažené výsledky zůstává řada možností pro další rozvoj, ať už směrem k přesnějším modelům, bohatším vstupním datům nebo hlubšímu zapojení reálného letového kontextu. Tato práce tak tvoří pevný základ pro budoucí zkoumání a rozšiřování prediktivních systémů v oblasti bezpilotního letectví.

5.2 Plánovaný rozvoj

Jedním ze směrů dalšího vývoje je rozšíření vstupních dat. Přesnost a adaptabilita modelu by mohly být zvýšeny přidáním dalších senzorických informací, jako je proudový odběr, teplota baterie nebo dynamika pohybu, například akcelerace, rotace nebo externí zátěž, jak také navrhuje práce [15]

Další možností je využití trajektorií letu. Během měření byly zaznamenávány také prostorové souřadnice dronu, které však nebyly dále využity v rámci modelování. Tyto trajektorie představují cenný zdroj informací o tom, jak konkrétní manévry ovlivňují spotřebu energie. Do budoucna by bylo možné tyto informace využít pro výpočet optimální trajektorie letu, která by minimalizovala spotřebu baterie, zkrátila dobu letu nebo kombinovala více optimalizačních cílů, například co nejrychlejší a zároveň nejúspornější cestu k cíli.

Za důležitý směr rozvoje lze považovat také nasazení pokročilejších modelů. V této práci byly použity jednodušší architektury s pevnou strukturou, které již poskytly velmi dobré výsledky. Pokročilejší neuronové sítě, jako jsou rekurentní modely (RNN), LSTM nebo GRU, by mohly výrazně lépe zachytit časové závislosti v datech a zpracovávat celé sekvence bez nutnosti použití autoregresivního přístupu.

Dalším krokem by mohla být integrace predikčního systému do autonomního řízení. V současném stavu slouží výstup modelu pouze jako pasivní informace. V budoucnu by mohl být využit k aktivnímu rozhodování, například k včasnému přistání, úpravě rychlosti nebo plánování trasy podle aktuálního stavu baterie a předpokládané spotřeby.

Zajímavou oblastí rozšíření je také zajištění generalizace modelu na různé hardwarové platformy. Současný model byl trénován a testován na jednom

konkrétním typu dronu, avšak v praxi se mohou lišit motory, baterie i celkové letové vlastnosti. V tomto směru by mohl být velmi užitečný přístup založený na reinforcement learningu. Tento způsob učení umožňuje modelu zlepšovat se na základě zpětné vazby přímo během provozu a přizpůsobovat se konkrétní konfiguraci hardwaru. Reinforcement learning by tak mohl nejen usnadnit přenositelnost modelu na jiné platformy, ale také podpořit jeho průběžné zdokonalování při reálném nasazení.

Bibliografie

[1] Bitcraze AB. *Crazyflie 2.1 - Product Information*. Oficiální dokumentace k platformě Crazyflie 2.1. 2023. URL: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/docs/hardware_cf2/.

[2] Bitcraze AB. *Crazyflie Python API (cflib) and Logging Documentation*. Dokumentace Python knihovny pro ovládání a logování dat z Crazyflie 2.1. 2023. URL: https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/api/.

[3] Ivo Bukovský et al. "Quadratic neural unit is a good compromise between linear models and neural networks for industrial applications". In: *IEEE International Conference on Cognitive Informatics*. 2010. DOI: 10.1109/COGINF.2010.5599677. URL: https://doi.org/10.1109/COGINF.2010.5599677.

[4] Lorenzo Ciampiconi et al. "A survey and taxonomy of loss functions in machine learning". In: *arXiv preprint arXiv:2301.05579* (2023). URL: https://arxiv.org/abs/2301.05579.

[5] Shiv Ram Dubey, Satish Kumar Singh a B. B. Chaudhuri. "Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark". In: *arXiv preprint arXiv:2109.14545* (2021). URL: https://arxiv.org/abs/2109.14545.

[6] Hatice Vildan Dudukcu, Murat Taskiran a Nihan Kahraman. "Unmanned Aerial Vehicles (UAVs) Battery Power Anomaly Detection Using Temporal Convolutional Network with Simple Moving Average Algorithm". In: *INISTA 2022 - IEEE International Symposium on INnovations in Intelligent SysTems and Applications*. 2022. DOI: 10.1109/INISTA55318.2022.9894193.

[7] O.S. Eluyode a A.A. Akinyede. "Comparative Study of Biological and Artificial Neural Networks". In: *European Journal of Applied Engineering and Scientific Research* 2.1 (2013). Odborné srovnání biologických a umělých neuronových sítí, s. 36–46.

[8] Julian Gatscher, Johannes Breitenbach a Ricardo Buettner. "Machine Learning-Based Power Consumption Prediction for Unmanned Aerial Vehicles in Dynamic Environments". In: *Proceedings of the Hawaii International Conference on System Sciences*. 2023. DOI: 10.24251/hicss.2023.839.

[9] Ian Goodfellow, Yoshua Bengio a Aaron Courville. *Deep Learning*. Základní učebnice o neuronových sítích a metodách hlubokého učení. Cambridge, MA: MIT Press, 2016. ISBN: 9780262035613.

[10] Krystian Góra et al. "Machine Learning in Creating Energy Consumption Model for UAV". In: *Energies* 15.18 (2022), s. 6810. DOI: 10.3390/en15186810.

[11] Charles R Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (2020). Doporučená citace pro NumPy, s. 357–362. DOI: 10.1038/s41586-020-2649-2.

[12] Simon Haykin. *Neural Networks and Learning Machines*. 3. vyd. Základní učebnice obsahující srovnání umělých a biologických neuronů. Upper Saddle River, NJ: Pearson Education, 2009.

[13] Munzir Hussain et al. "Differential Flatness Based Control for Aggressive Maneuvers with Crazyflie 2.1". In: *IEEE Access* 8 (2020). Modelování a řízení Crazyflie 2.1 pomocí konceptu plochosti, s. 117135–117147. DOI: 10.1109/ACCESS.2020.3004940.

[14] Arshmeet Kaur a Morteza Sarmadi. "Comparative Analysis of Data Preprocessing Methods, Feature Selection Techniques and Machine Learning Models for Improved Classification and Regression Performance". In: *arXiv preprint arXiv:2402.14980* (2024). DOI: 10.48550/arXiv.2402.14980.

[15] Jiwon Kim et al. "Voltage prediction of drone battery reflecting internal temperature". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 2022. DOI: 10.1145/3489517.3530448.

[16] Diederik P Kingma a Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980* (2015). Původní článek popisující optimalizační algoritmus Adam.

[17] Wes McKinney. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference (SciPy)*. Ed. Stéfan van der Walt a Jarrod Millman. Základní článek k pandas. SciPy. 2010, s. 51–56.

[18] V. Parasich et al. “Overfitting in Machine Learning: Problems and Solutions”. In: *Cybernetics and Information Technologies* (2024). DOI: 10.14529/ctcr240202.

[19] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems*. Sv. 32. Oficiální článek popisující PyTorch. 2019. URL: https://arxiv.org/abs/1912.01703.

[20] P. Siva Kota Reddy, K. Deepa a S. Sangeetha. “Case Study on Predictive Modeling of UAV Battery Remaining Useful Life Using Machine Learning Regression Techniques”. In: *IEEE ICEMPS* (2024). DOI: 10.1109/icemps60684.2024.10559377.

[21] Alessio Rinaldi, Adriano Fagiolini a Antonio Bicchi. “Comparison of Control Strategies for Quadrotor Stabilization: PID, LQR, MPC, Sliding and Adaptive Control”. In: *Applied Sciences* 13.4 (2023). Srovnání řídicích strategií pro kvadrokoptéry, s. 2231. DOI: 10.3390/app13042231.

[22] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016). Přehled optimalizačních algoritmů používaných při trénování neuronových sítí.

[23] A.A. Shah et al. “Nonlinear autoregressive models for high accuracy early prediction of Li-ion battery end-of-life”. In: *Journal of Energy Storage* 68 (2023), s. 109014. DOI: 10.1016/j.est.2023.109014.

[24] Charles Simon. *Machine Learning Is Not Like Your Brain: Perceptrons vs Neurons*. https://www.kdnuggets.com/2022/05/machine-learning-not-like-your-brain-part-2.html. Populárně-odborný článek s vysvětlením rozdílů mezi perceptronem a biologickým neuronem. 2022.

[25] Changhoon Song a Sukhan Lee. “Accurate RUL Prediction Based on Sliding Window with Sparse Sampling”. In: *International Conference on Ubiquitous Information Management and Communication (IMCOM)*. 2021. DOI: 10.1109/IMCOM51814.2021.9377402.

[26] Dattatray G. Takale et al. “Comparative Analysis of LSTM, RNN, CNN and MLP Machine Learning Algorithms for Stock Value Prediction”. In: *Journal of Future Smart Networks* (2024). DOI: 10.48001/jofsn.2024.211-10. URL: https://doi.org/10.48001/jofsn.2024.211-10.

[27] Søren B. Vilsen a Daniel-Ioan Stroe. “An auto-regressive model for battery voltage prediction”. In: *Applied Power Electronics Conference (APEC)*. 2021. DOI: 10.1109/APEC42165.2021.9487060.

Přílohy

Níže uvedené skripty jsou součástí jednoho ZIP archivu této práce.

Ve složce `data` jsou uloženy CSV soubory získané z jednotlivých letů dronu, které tyto skripty využívají.

Skripty `flight_fnb.py`, `flight_hover.py` a `flight_upndown.py` slouží k řízení dronu a záznamu dat během různých letových režimů.
Skript `flight.py` umožňuje manuální ovládání pomocí joysticku.
Skript `flight_w_pred.py` obsahuje finální implementaci predikčního modelu, který průběžně odhaduje zbývající čas do vybití baterie a ukládá výsledky do CSV souboru.

Skript `maketab.py` načítá CSV soubory z letů a převádí je do jednotné formy vhodné pro trénování modelů. Zahrnuje základní úpravy dat, jako je výběr relevantních sloupců a výpočet odvozených veličin.

Soubory `bat_map_*.py` a `batmot_map_*.py` slouží k trénování regresních modelů pro odhad zbývajícího času letu na základě různých vstupních veličin, jako je napětí baterie nebo signál PWM. Jednotlivé skripty se liší použitou architekturou modelu.

Skripty `sigpred_*.py` slouží k predikci budoucího vývoje napětí na základě časových řad naměřených hodnot. I zde jsou využívány různé typy modelových struktur a přístupů.