VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

LETECKÝ ÚSTAV

INSTITUTE OF AEROSPACE ENGINEERING

REGISTRACE PŘISTÁVAJÍCÍHO LETOUNU Z OBRAZOVÉHO ZÁZNAMU

REGISTRATION OF THE LANDING AIRCRAFT BASED ON IMAGE RECOGNITION TECHNIQUES

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jan Juroška

AUTHOR

VEDOUCÍ PRÁCE — Ing. Miroslav Červenka, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Letecký ústav
Student:	**Bc. Jan Juroška**
Studijní program:	Letecká a kosmická technika
Studijní obor:	Technologie provozu letadlové a letištní techniky
Vedoucí práce:	**Ing. Miroslav Červenka, Ph.D.**
Akademický rok:	2023/24

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Registrace přistávajícího letounu z obrazového záznamu

Stručná charakteristika problematiky úkolu:

Zaznamenání přistávajícího letounu na neřízených letištích je možné automatizovat za použití počítačového vidění. Tato práce si klade za cíl vytvořit program a instrukce k jeho použití, které tento postup co nejvíce automatizují.

Cíle diplomové práce:

1. Přehled možností sledování provozu na letištích.
2. Rešerše a test relevantních postupů.
3. Návrh koncepce a struktury programu.
4. Ověřování a testování limitů implementované aplikace.
5. Dokumentace programu, sepsání doporučení a instrukcí k využití.

Seznam doporučené literatury:

DUDA, Richard O, Peter E. (Peter Elliot) HART a David G STORK. Pattern classification. New York: J. Wiley, 2001, xx, 654 s. ISBN 0-471-05669-3.

RUSSELL, Stuart J. (Stuart Jonathan) a Peter NORVIG. Artificial intelligence: a modern approach. London: Pearson, 2016, xviii, 1132 s. ISBN 978-1-292-15396-4.

RUSS, John C. The image processing handbook. Boca Raton: Taylor & Francis, 2007, 817 s. ISBN 0-8493-7254-2.

SONKA, Milan, HLAVAC, Vaclav a BOYLE, Rogert. Image Processing, Analysis and Machine Vision. Toronto: Thomson, 2008. 829 s. ISBN 978-0-495-24438-7. (EN).

SZELISKI, Richard. Computer Vision: Algorithms and Applications. London: Springer, 2010, xx, 812 s. ISBN 978-1-84882-934-3.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2023/24

V Brně, dne

L. S.

doc. Ing. Jaroslav Juračka, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

Abstrakt

V rámci této práce byl vytvořen postup, který automatizuje zaznamenávání letadel na neřízených letištích pomocí počítačového vidění. Letadlo je nejdříve rozpoznáno pomocí neuronové sítě z rodiny YOLO a následně je na něm přečtena imatrikulace pomocí sítě Tesseract. V teoretické části práce jsou uvedeny možné přístupy k této problematice a teoretické základy, které stály za výběrem použité metodiky. V praktické části je program zdokumentován, jsou testovány limitní případy jeho detekčních schopností a je poskytnuta příručka s instrukcemi k uvedení do provozu a použití.

Abstract

In this diploma thesis, a method automating for the process of registering an aircraft on an uncontrolled airfield using computer vision was created. First, aircraft is detected using neural network form the YOLO family, next, its registration is read using the Tesseract network. In the theoretical part of this work, various methods for solving this issue are introduced, as well as the theoretical basis behind the chosen method. In the practical part of this work, the documentation of the created program is provided. Testing of programs limits is conducted, and instructions for setting up and using the method are provided.

Klíčová slova

Sledování letištního provozu, rozpoznávání přistávajících letadel, neuronové sítě, automatizace, zpracování obrazu, detekce textu.

Keywords

Airport traffic surveillance, registering of landing aircraft, neural networks, automation, image processing, text detection.

Bibliografická citace

Citace tištěné práce:

JUROŠKA, Jan. Registrace přistávajícího letounu z obrazového záznamu. Brno, 2024. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/153946. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Letecký ústav. Vedoucí práce Miroslav Červenka.

Citace elektronického zdroje:

JUROŠKA, Jan. Registrace přistávajícího letounu z obrazového záznamu [online]. Brno, 2024 [cit. 2024-02-14]. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/153946. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Letecký ústav. Vedoucí práce Miroslav Červenka.

LETECKÝ ÚSTAV
Institute of Aerospace Engineering

Čestné prohlášení

Prohlašuji, že jsem tuto diplomovou práci na téma Registrace přistávajícího letounu z obrazového záznamu vypracoval samostatně, pod vedením Ing. Miroslava Červenky, Ph.D., s použitím literatury a dalších podkladů uvedených v seznamu použitých zdrojů.

V Brně dne ……

Bc. Jan Juroška

Poděkování

V první řadě bych rád poděkoval svému vedoucímu, panu Ing. Miroslavu Červenkovi, Ph.D., za veškerou jeho ochotu, trpělivost, vstřícný přístup a cenné rady, které mi během psaní mé práce poskytl. Také bych rád poděkoval Leteckému ústavu, který mi poskytl skvělé zázemí nutné ke zvládnutí magisterského studia i diplomové práce. Také bych rád poděkoval své rodině, protože jejich podpora pro mne na cestě celým mým studiem znamenala víc než si kdokoliv z nich dokáže představit.

Obsah

1. Úvod 1
2. Přehled možností sledování provozu na letištích 2
3. Počítačové vidění 5
   3.1. Úloha počítačového vidění a její řešení 5
   3.2. Překážky v řešení úlohy počítačového vidění 7
   3.3. Shrnutí 9
4. Návrh a test základní myšlenky programu 10
   4.1. Teorie 10
   4.2. Testování 11
   4.3. Shrnutí 14
5. Struktura a dokumentace programu 15
   5.1. Databáze a operace s ní 20
   5.2. Zpracovávací čas a výpočet snímku k zpracování 24
   5.3. Zpracování snímku 27
   5.4. Shrnutí 45
6. Testování limitů 46
7. Instrukce k použití 48
   7.1. Před použitím 48
   7.2. Spuštění 48
   7.3. Použití 50
   7.4. Shrnutí 55
8. Závěr 56
9. Bibliografie 57
10. Seznam příloh 60

1. Úvod

Udržení přehledu o přistávajících letadlech na neřízených letištích je užitečné zejména pro získání povědomí o činnostech třetích stran na letišti během nepřítomnosti majitelů či provozovatelů dané plochy, jako pomocník pro správné vybírání letištních poplatků a ulehčení práce provozovatelům letiště, jako je udržení řádu, dodržování hlukových limitů a prostředí nutného pro bezpečný a příjemný provoz pro všechny zúčastněné. Udržovat tento přehled v dostatečné míře je ale relativně náročné na lidskou práci, které na neřízeném letišti nebývá dostatek. Proto se v dnešní době přímo nabízí využití moderních technologií na automatizování tohoto procesu.

Tato práce si klade za cíl vytvořit systém, který by dosáhl jistého zautomatizování tohoto procesu, dále vytvoření rad a instrukcí k použití. V rešeršní části jsou krátce shrnuty možnosti sledování provozu na letištích a jejich vhodnost pro použití na neřízených letištích, jisté základní přístupy k problematice počítačového vidění a návrh základní myšlenky programu. Praktická část podrobně popisuje funkci celého systému a shrnuje výsledky testů jeho schopností. Práce je ukončena příručkou k použití celého systému, která osvětlí, jak program zprovoznit a správně používat.

2. Přehled možností sledování provozu na letištích

Tato kapitola stručně shrnuje několik způsobů, kterými lze sledovat provoz na letišti a uvádí některé jejich výhody a nevýhody. V závěru zdůvodňuje, jaká metoda byla pro práci vybrána a proč.

Vizuální sledování

Nejjednodušším způsobem sledování provozu na letištní ploše je prostý pohled okem. Tato metoda je jednoduchá z hlediska technologického, má ovšem rezervy z hlediska provozního.

V první řadě je nutné zajistit zodpovědnou osobu, která bude sledování provádět, což může být na neřízených letištích nepraktické. Dále je pak nepoužitelná na větších a komplikovanějších letištích, kde může být problém udržet si čistě vizuálně dostatečný přehled o provozu. Z těchto důvodů bývá často vizuální sledování rozšířeno technologiemi.

Radarové sledování

Nejrozšířenější technologií, která pomáhá zlepšit přehled o provozu na letišti, je použití radaru pro řízení pohybu na letištní ploše (SMR). Jedná se o primární radar, který sleduje pohyb předmětů po ploše letiště, typicky vozidel a letadel. Na rozdíl od klasických radarů je přizpůsoben na práci v prostředí se silným rušením a kratší vzdálenosti. Pracuje v pásmu X nebo K, díky tomu má takovýto radar mnohem lepší rozlišovací schopnost, ale také podléhá ve větší míře rušení deštěm a podobnými meteorologickými podmínkami. Použití primárního radaru by samo o sobě neumožňovalo identifikaci letadla, je proto často používán v kombinaci s vizuálním sledováním nebo je doplněn o sekundární radar. Pro nejvyšší úroveň přesnosti zaměření se používá systém více sekundárních radarů, který určuje polohu zaměřovaného objektu multilaterací. V případě použití sekundárního radaru je také vhodné vybavit všechna vozidla a zařízení pohybující se po letišti odpovídačem. [1]

Tato metoda dobře řeší problém sledování větších a komplikovaných letišť. Použití sekundárního radaru také řeší problematiku sledování při nepříznivých meteorologických podmínkách. Tato metoda je ale drahá a technologicky i provozně náročná, je nutné zajistit radary a jejich obsluhu. Pro využití na menších neřízených letištích je proto vyloženě nevhodná.

Využití online služeb třetích stran

Dalším způsobem, jak tento problém řešit, je využití online služeb jako je např. Flightradar24 nebo ADS-B Exchange. Pomocí těchto služeb lze zjistit, která letadla využívající patřičný odpovídač přistála na letišti. Informace lze získat buďto přímo z jejich webových stránek v reálném čase [2] jako je na obrázku 2, nebo je od nich lze vyžádat jako data zpětně [3], viz obrázek 1. Tyto informace by se musely zpracovávat buďto ručně, nebo by bylo možné je automatizovat pomocí softwaru.

```
{"icao":"000001",
"noRegData":true,
"timestamp": 1714521600.000,
"trace":[
[1962.97,35.867386,-78.773346,400,47.5,112.2,1,640,null,"adsb_icao",400,null,null,null],
[1963.46,35.867340,-78.773232,400,48.5,111.8,0,640,null,"adsb_icao",400,null,null,null],
[1963.94,35.867289,-78.773083,400,49.8,112.4,0,640,null,"adsb_icao",400,null,null,null],
[1964.44,35.867242,-78.772966,400,50.7,112.0,0,704,{"type":"adsb_icao","flight":"ZEUS32
","alt_geom":400,"track":112.01,"baro_rate":704,"emergency":"none","category":"A7",
"nic":10,"rc":25,"version":2,"nic_baro":1,"nac_p":10,"nac_v":4,"sil":3,
"sil_type":"perhour","gva":2,"sda":2,"alert":0,"spi":0},"adsb_icao",400,null,null,null],
```


obrázek 1: data dodaná společností ADS-B Exchange zpětně na vyžádání [3]

obrázek 2: aktuální informace poskytnuté společností ADS-B Exchange v reálném čase [4]

Tato metoda je technologicky nejjednodušší ze všech uvedených, je pro ni potřeba pouze přístup k internetu a počítač, dokonce ani není třeba mít přístup k letišti. Je zde ovšem několik problémů z hlediska provozního. V první řadě je možné, že pilot letadla vypne svůj odpovídač, a tím znemožní jeho detekci. Vzhledem k povaze cílů, za kterými chceme letadlo detekovat, jako je například výběr letištních poplatků, se nezdá moudré ponechat pilotům tuto možnost. V druhé řadě je použití podobných prostředků pro jiné než osobní účely zpravidla zpoplatněno. Například cena za roční využití dat z webu Flightradar24 je v době psaní této práce 499 USD. [5] Na trhu jsou ovšem i levnější varianty. Příkladem může být ADS-B Exchange. Tato firma své ceny za takové komerční využití sice nezveřejňuje, zveřejňuje ovšem své ceny v jiných kategoriích, ve kterých se pohybuje zhruba na pětině ceny oproti firmě Flightradar24. [6] V porovnání s jinými metodami je tedy třeba vzít v úvahu nejen spolehlivost detekce, ale i finanční náklady.

Automatizace kamerovými systémy

Vizuální sledování lze také doplnit, nebo zcela nahradit pomocí počítačového vidění. V tomto případě jsou na letiště nainstalovány kamery, a zpracováním jejich záznamu jsou získány potřebné informace. Jedná se tedy o jakousi technologickou analogii sledování okem. Počítačové vidění je velmi široká disciplína a lze jej pojmout mnoha různými způsoby. [7]

Je možné využít klasickou kameru fungující ve viditelném spektru k rozšíření plochy sledované řídícím. Dalším příkladem je využití infračervené kamery, která krom pohybu dopravních prostředků po letišti pomůže ulehčit sledování nežádoucího pohybu osob nebo zvířat po areálu letiště [8].

Všechny různé kamerové vstupy je možné dále zpracovávat počítačem, a tím snížit práci potřebnou pro sledování provozu. Pomocí takového systému je pak možné převést obraz na potřebná data, jako jsou pozice, imatrikulační značky nebo třeba i typy letadel.

Technologická a provozní náročnost takového systému se odvíjí od jeho účelu [7]. Může se jednat o jedinou poměrně levnou kameru připojenou k počítači, která obraz pouze zpracuje a uloží pro pozdější užití, nebo třeba systém mnoha infračervených kamer, jejichž software pracuje ve spolupráci s řídícím a jinými zařízeními.

Shrnutí

V rámci této kapitoly byl poskytnut přehled možností sledování provozu na letištích. Už z popisu jednotlivých metod je zřejmé, že prosté vizuální a radarové sledování pro tuto práci není vhodné. V rámci tohoto shrnutí jsou proto srovnány pouze využití online služeb třetích stran s automatizací kamerovými systémy.

První metoda se zdá jednoduchá z hlediska provedení. Není třeba žádného složitého systému pro její provoz. Stačí pouze počítač a přístup k internetu. Dále se zdá vhodná z hlediska obsluhy, není pro ni třeba nikoho, krom technické podpory a příležitostné obsluhy. Internetové služby často nabízejí přístup k datům z minulosti, takže zpracování může proběhnout jednou za určitý časový úsek. Tyto služby ovšem bývají zpoplatněny a závisí na férovosti pilotů.

Druhá metoda je z hlediska provedení složitější. Je potřeba systém skládající se z počítače a kamery se sběračem nebo přenašečem dat. Na druhou stranu není nutný přístup k internetu. Dále je třeba zajistit podmínky pro provoz kamery na letišti, kde může být celý systém ovlivněn vnějšími vlivy jako jsou např. meteorologické podmínky. Jednoznačnou výhodou tohoto systému je jeho nízká cena. Cena vhodné kamery na trhu se momentálně pohybuje od 600 korun výše, a mnohá letiště jsou již kamerou vybavena.

Pro mou práci se zdá vhodnější druhá varianta, a to z provozních i finančních důvodů. Náročnější provedení a obsluha jsou jimi více než vyváženy.

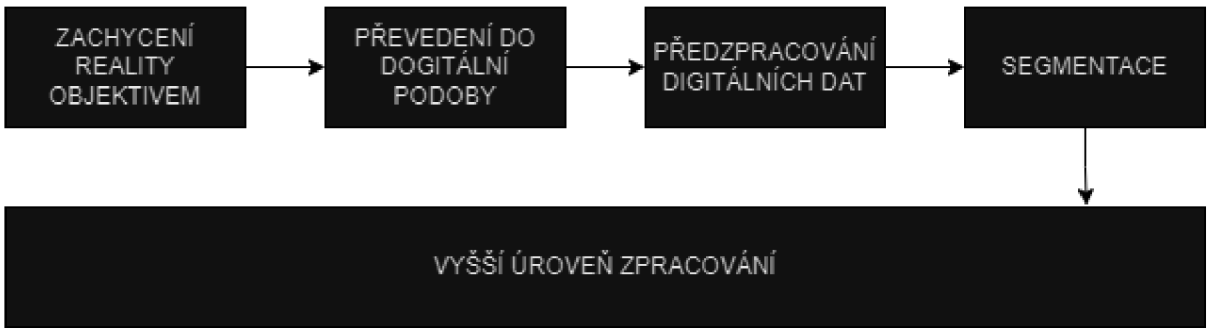
3. Počítačové vidění

V této kapitole je popsáno počítačové vidění jako vědní disciplína. Je představena jeho obecná úloha a nastíněn přístup k jejímu řešení, jakožto i problémy, které se při něm běžně vyskytují.

Počítačové vidění se řadí do mezioborových vědních disciplín. Jeho předmětem je porozumění obrazu nebo obrazovému záznamu s použitím počítačů. V praxi se dá říct, že jde o převedení obrazových dat na jiná požadovaná data. [7]

3.1. Úloha počítačového vidění a její řešení

Základní úloha počítačového vidění se dá charakterizovat jako pokus najít vztah mezi vstupním obrazem a předem zavedeným modelem světa. Tento proces obyčejně probíhá v mnoha krocích, ve kterých jsou obrazová data postupně rafinována do výsledné podoby, ta je pak interpretována předem stanoveným modelem. Je velmi důležité zdůraznit, že tento proces nemá žádnou univerzálně platnou metodiku nebo postup. Každý problém si žádá vysoce individuální a nezřídka kreativní přístup. Přesto lze nastínit orientační postup, kterým jsou data obyčejně zpracována [9], viz obrázek 3. Je ovšem třeba mít na paměti, že některé kroky postupu často nelze zcela přesně vymezit, prolínají se, nebo zcela splývají. [7]



obrázek 3: obecný postup počítačového vidění

Prvním krokem je zpravidla zachycení reality přístrojem, a tím získání analogové verze potřebných dat. V případě této práce, stejně jako v drtivé většině případů, je toto provedeno klasickou kamerou, může se ale jednat o podobná zařízení pracující v jiných spektrech, jako jsou např. termokamery nebo radary.

V druhém kroku jsou data převedeny z analogové do digitální verze. Toto typicky probíhá přímo v kameře. V případě některých netradičních způsobů zachycení dat je možné, že je tento krok proveden až v počítači. Cílem je umožnit počítači s daty pracovat.

Dalším krokem je tzv. předzpracování obrazu. Jeho cílem je příprava dat na vyšší úrovně zpracování. Vstupní i výstupní data jsou obrazová, většinou ve stejné nebo velmi podobné formě. Tento krok nijak nezvyšuje počet informací v obrázku, pouze je lehce upravuje nebo potlačuje, typicky tedy snižuje entropii jasové funkce obrazu. Teoreticky lze říct, že v ideálním případě je nejlepší žádné předzpracování nepoužít, a zakomponovat veškeré informace obsažené v obrázku do jeho analýzy.

V praxi je ovšem předzpracování velmi užitečné k odlišení rušivých informací od těch, které jsou pro budoucí analýzu důležité. Příkladem může být částečné nebo úplné potlačení barev, zostření nebo rozostření obrázku, odstranění šumu nebo převedení na jiné rozlišení. Některé příklady předzpracování můžeme vidět na obrázku 4obrázek 4. Cílem předzpracování je obecně zlepšení obrazových dat zmírněním nedůležitých prvků a zvýrazněním důležitých. [7]

obrázek 4: různé druhy předzpracování vlevo nahoře původní obrázek, vpravo nahoře převedení na šedobílý obraz, vlevo dole převedení na černobílý obraz, vlevo dole detekce hran

Následujícím krokem postupu je segmentace. To je jedna z nejdůležitějších částí celého postupu, úspěšnost celého procesu se často odvíjí právě od úspěšnosti segmentace. Cílem je vyčlenit z obrazu oblasti, které mají velmi silnou korelaci s objekty v reálném světě, čili poznat, kterými regiony obrázku je třeba se na vyšší úrovni zpracování zaobírat. K tomuto účelu je možno použít mnoho postupů, od nejjednodušších jako je např. prahování, až po složité unikátní algoritmy využívající apriori znalosti týkající se daného konkrétního problému. [7]

Příklad segmentace můžeme vidět níže na obrázcích, kdy je z původního obrázku 5 vystřiženo letadlo a začerněny pixely které letadlu nebo jeho bezprostřednímu okolí nenáleží, výsledek je pak na obrázku 6.

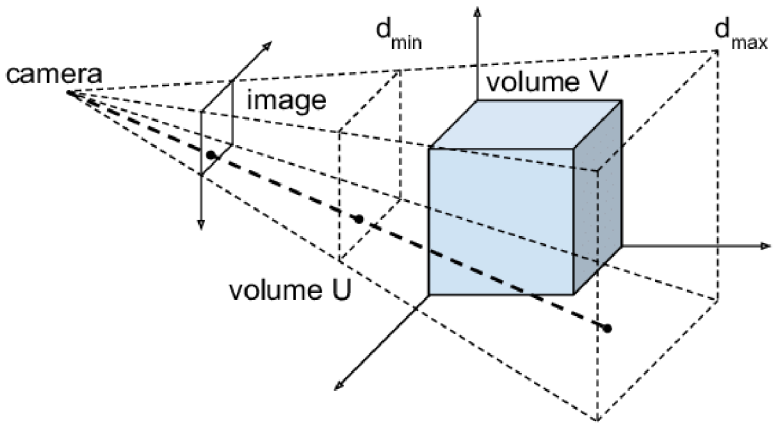
obrázek 5: původní kamerový snímek

obrázek 6: segmentovaná část snímku s letadlem

Následuje vyšší úroveň zpracování, kdy je předzpracovaný a segmentovaný obraz interpretován do již zavedeného modelu. To může být například podrobné zpracování leteckých fotografií [10] nebo pozorování populací létavého hmyzu [11]. Tato část je individuální danému konkrétnímu problému, a nelze ji nijak konkrétněji rozepsat. Obecně je jejím cílem získat požadovaná výstupní data z obrazu, který k tomu byl vhodně připraven. [7]

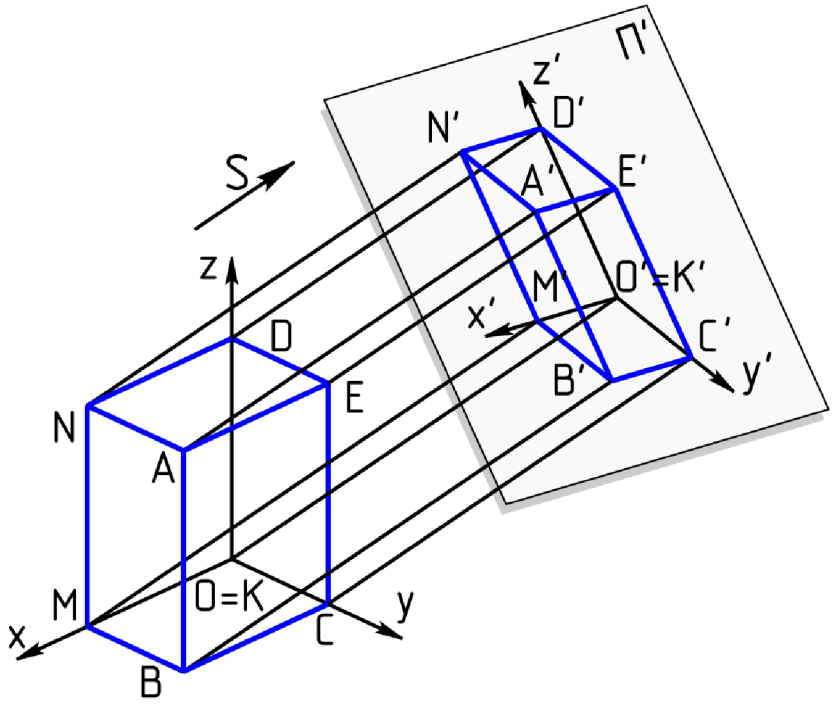
3.2. Překážky v řešení úlohy počítačového vidění

Ačkoliv každá konkrétní úloha si nese svá úskalí, existují problémy, které je nutné v rámci počítačového vidění řešit téměř vždy. Tyto problémy jsou počítačovému vidění neoddělitelně vlastní a vycházejí z jeho povahy. [7]



obrázek 7: perspektivní projekce [12]

Prvním problémem je ztráta informace při převodu 3D reality na 2D obraz. Nejběžnější metoda převodu používaná u kamer je perspektivní projekce, kterou můžeme vidět znázorněnou na obrázku 7, ta má za následek, že z 2D obrazu nedokážeme určit vzájemnou velikost a vzdálenost předmětů. Toto lze řešit více způsoby. Jeden ze způsobů je použití telecentrického objektivu, jehož zobrazení je znázorněno na obrázku 8, toto zobrazení ovšem velmi omezuje prostor, který jsme schopni objektivem snímat. Další ze způsobů je použití soustavy několika kamer, podobně jako to dělají lidské oči. Tato metoda ovšem obrovským způsobem komplikuje všechny ostatní procesy zpracování. [7]



obrázek 8: projekce telecentrického objektivu [13]

Druhým běžným problémem je interpretace významu obrázku. Když se člověk snaží zjistit co je na obrázku, využívá vrozených instinktů a informací nasbíraných za celý život. Toto je dále podpořeno lidskou schopností abstraktně myslet. Při použití

strojů tyto možnosti nemáme. K obrazovému zpracování jsou běžně použity dva přístupy, logické modely reality, nebo umělé inteligence. Při použití logických modelů se musí předem vytvořit takový model, v jehož kontextu budou extrahovaná data dávat smysl. Příkladem může být kontrola kvality součásti, kdy je z obrázku vysegmentována výrobní vada, a v rámci modelu je posouzena její velikost a charakter [14].

Třetím problémem je šum. Za šum je obecně považováno znečištění měřených dat, v případě počítačového vidění typicky kamerového záznamu. Může se jednat o znečištění objektivu, husté sněžení nebo třeba znečištění vzniklé poškozením disku kde je obrázek uložen. Je neoddělitelně více či méně přítomen v každém měření. Toto znečištění je třeba odstínit matematickými nástroji, jako je teorie pravděpodobnosti. Čím složitější jsou operace s daty a forma dat, tím složitější matematické nástroje je třeba použít.

Čtvrtým faktorem je příliš mnoho dat. Na rozdíl od standardních měření v laboratořích či průmyslu, jako je měření vzdáleností, tlaků nebo teplot, je v obraze přítomno mnoho různých dat, které nejsou z hlediska měření důležité, a i když důležitá jsou, bývají tato data velmi objemná. Toto s sebou nese značnou náročnost z hlediska výpočetního a také to vytváří zvýšené požadavky na datová úložiště. I když je dnes výpočetní technika velmi pokročilá, častokrát je složité zpracovat data v únosných časech, zvláště pokud je třeba zpracovávat v reálném čase. [7]

Páté hledisko, které je třeba brát v úvahu je, že obraz nereprezentuje realitu dokonale. Obraz vzniká poměrně komplikovanými fyzikálními procesy. Jas každého pixelu je ovlivněn osvětlením, geometrií povrchu, pozicí kamery a optickými vlastnostmi povrchu. V praxi se proto objevily pokusy obrazové snímání doplnit jinými pomocnými metodami, jako je např. lidar a podobná zařízení, které informace z prostoru extrahují přímo. [7]

Šestý a poslední problém, který zde bude uveden, je problém lokálního vhledu, a to ve více formách. Jednak obraz nikdy není schopen zachytit realitu zcela, vždy se může něco nacházet mimo obraz, nebo v zákrytu za jiným předmětem. Jednak se ukazuje velmi obtížné zpracovávat předměty v obraze v kontextu s jinými, většinou je schůdné zpracovat pouze daný segmentovaný region a jeho bezprostřední okolí. [7]

3.3. Shrnutí

Při použití počítačového vidění je nutné především určit model, podle kterého bude rozpoznáno přistávající letadlo, a způsob kterým do něj budou letadla zařazena. Dále je nutné zjistit, jak obraz na tento proces připravit. Při návrhu celé metody je nutné mít na paměti šest základních problémů počítačového vidění, a pokud možno je co nejvíce eliminovat nebo v rámci možností zohlednit.

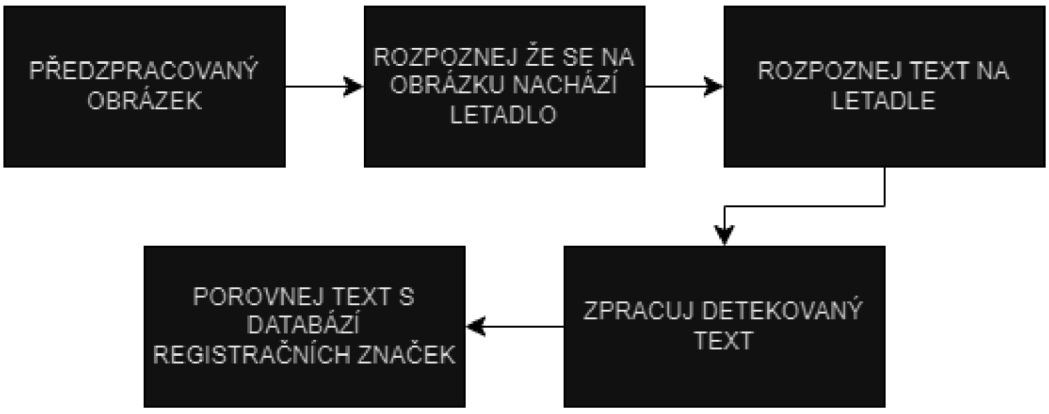
4. Návrh a test základní myšlenky programu

Tato kapitola zodpoví základní otázky, které vyplývají z použití počítačového vidění pro řešení úlohy. Jaký model bude použit pro rozpoznávání letadel? Jak bude obraz do tohoto modelu zařazován? Jaká bude zvolená segmentace? Jaké bude nutné předzpracování? Jinými slovy určí, jakým způsobem bude každý snímek zpracován. V první části kapitoly bude nastíněn teoretický základ zvolených postupů. V druhé budou prezentovány výsledky testů jejich validnosti.

4.1. Teorie

Každé letadlo je jednoznačně určeno svou registrační značkou. Jako model, do kterého bude obraz zařazen, se tedy seznam registračních značek nebo jejich formátů zdá přímo ideální. Tyto informace jsou navíc veřejně dostupné v leteckých rejstřících. Dále se jako vhodný postup jeví najít v obraze letadlo. To sice jednoznačně neurčí, o jaké letadlo se jedná, ale v dnešní době jsou dostupné umělé inteligence, které velmi spolehlivě dokážou letadla v obraze detekovat, což může při hledání registrační značky výrazně pomoct.

Zvolený postup tedy bude hledat v obraze letadlo, a pokud bude nalezeno, snažit se na něm přečíst jeho registrační značku. Nahrubo detekovaný text potom zušlechtit, a pokud bude v předem vytvořeném seznamu registračních značek získaného z leteckého rejstříku Úřadu pro civilní letectví, nebo bude odpovídat některému z obecných formátů registračních značek, bude uvažováno jednoznačně detekované letadlo. Tento proces je znázorněn na obrázku 9.



obrázek 9: zvolený postup pro detekování letadla

Tímto také prakticky zodpovíme otázku segmentace. Pokud nejdříve rozpoznáme, že se na obrázku nachází letadlo, a až potom na této části obrázku rozpoznáváme text, jedná se prakticky o segmentační úlohu. Je záhodno poznamenat, že neuronové sítě na rozpoznání textu mají typicky vlastní interní segmentaci [15], v rámci které detekují text na obrázku, dá se tedy říct že je zde segmentace provedena ve dvou krocích.

Testované předzpracování se bude odvíjet podle potřeb jednotlivých neuronových sítí. Obecně je pro neuronové sítě nutné zvážit použití odstranění šumu, např. pomocí gaussovské nebo podobné masky, a zvětšení nebo zmenšení obrázku [15] [16]. U této úlohy nepředpokládám žádný závažný šum, ten by sice mohl vzniknout za nepříznivých meteorologických podmínek, za těch se ale neřízená letiště, na kterých je zamýšlené použití tohoto systému, běžně nepoužívají. Co se týče zvětšení nebo zmenšení obrázku, každá síť je trénována na detekování objektů určité velikosti v pixelech, pokud pak máme na obrázku objekt, který chceme detekovat, měl by být tento konkrétní objekt takto veliký, bez ohledu na rozlišení samotného obrázku. Díky tomu může, od určitého bodu, zvyšování kvality obrazu paradoxně zhoršit detekci, protože neuronová síť na tak velké a ostré objekty není trénovaná. [16]

Čtecí neuronové sítě typicky fungují nejlépe s textem v černobílé, ideálně knižní podobě [15], je proto na místě testovat převedení obrázku na černobílý, nebo šedobílý. Je ovšem nutno podotknout, že tyto sítě jsou v omezené míře schopné rozpoznat i barevný text na barevném pozadí [15]. Důležitá je také velikost textu, často je vhodné obrázek zvětšit nebo zmenšit pro umělé upravení počtu pixelů na písmeno [17]. Dále je problematické rozpoznání textu, který je nějak otočený nebo nakloněný [18], teoreticky by bylo možné obraz uměle rotovat, a tak získat všechny možné varianty natočení a detekovat text na každé z nich. Toto by ale bylo výpočetně náročné a tedy by bylo obtížné použití v reálném čase, proto tato metoda nebude v rámci této práce zvažována.

Neuronové sítě na rozpoznávání objektů většinou nepotřebují speciální předzpracování [16]. Bývají trénovány na rozličných sadách dat, zachycených za různých optických podmínek. Jediné předzpracování, které se tady nabízí, je zvětšení nebo zmenšení obrázku.

Následně je třeba vybrat neuronové sítě, které budou na plnění úloh použity. Hlavní kritéria výběru budou rychlost zpracování a jednoduchost použití. Rychlost zpracování je extrémně důležitá, protože letadlo může být na záznamu zachyceno pouze po dobu pár vteřin, za kterou bude muset být spolehlivě detekováno. Jednoduchost použití je významná z hlediska uživatelské příjemnosti.

Pro detekci a rozpoznání textu byla vybrána síť Tesseract [15], která je v dostupných zdrojích považována za bezkonkurenčně nejrychlejší, a jejíž použití je jednoduché a relativně široké. Za tyto vlastnosti ovšem platí nižší přesností. Tato nižší přesnost se bude muset zohlednit při dalším navrhování programu. [19] [20] [21]

Pro detekci letadel byla vybrána síť YOLOv8 [16]. Tato síť je považována za jednu z nejrychlejších sítí, přitom ovšem stále dosahuje velmi vysoké přesnosti detekce [22] [23] [24]. Její použití je velmi jednoduché, je součástí knihovny ultralytics a není třeba ji nijak zvlášť instalovat. Ve své nativní podobě také umožňuje rozpoznávání letadel, což je pro tuto práci ideální.

4.2. Testování

Na sadu několika desítek videí přistávajících, pojíždějících a vzlétajících letadel, která byla natočena na mobil Xiaomi POCO X3 PRO na medláneckém letišti, bylo použito rozpoznání letadel pomocí sítě YOLOv8n z knihovny ultralytics verze 8.2.8. V oblasti s detekovaným letadlem byl dále detekován text pomocí sítě Tesseract verze

0x0000024322921630 s různými nastaveními, ze kterých se nakonec nejlépe osvědčilo nastavení psm 12 oem 3. V rámci předzpracování bylo testováno několik postupů a funkcí:

- převedení na černobílý obraz vs. převedení na šedobílý obraz, a to v různých bodech procesu
- změna rozlišení obrázků a částí obrázků s detekovanými letadly
- potlačení šumu gaussovskou maskou

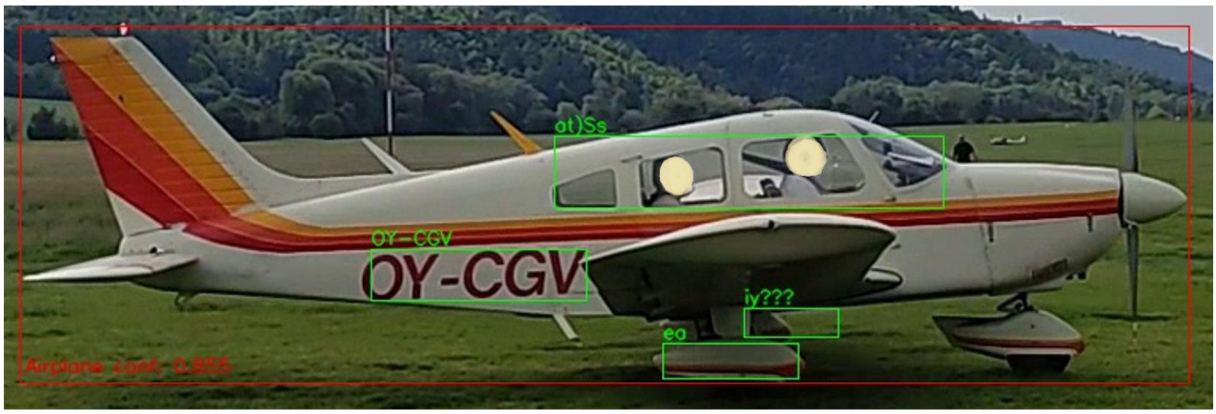
Nejspolehlivější metodou se ukázalo být nejdříve rozpoznat letadlo a až následně převést obraz na šedobílý, a to bez gaussovského rozostření. Změna rozlišení byla použita nepřímo. Pokud bylo letadlo detekováno s nízkou jistotou, byla detekce provedena znovu pouze na výřezu původního obrázku, viz kapitolu 5.3, síť YOLO si u všech obrázků interně změní rozlišení tak, aby byl počet pixelů strany vždy násobek 32 pixelů a ne méně než 640 [16]. Letadla byla vždy spolehlivě detekována, text nebyl vždy zcela spolehlivě detekován, příklady můžeme vidět na obrázcích 10, 11, 12, 13 a 14. Důvod simplistického řešení bez složitějších předzpracovávacích postupů vychází z toho, že detekované letadlo v obrázku má vždy jinou velikost. A to jak v pixelech, tak relativně. Toto nastává jednak z důvodu různé vzdálenosti a přiblížení kamery, jednak protože skutečné letadlo zabírá v detekovaném výřezu pokaždé jinou část. Proto nelze jednoduše určit potřebnou velikost Gaussovské masky ani jiného podobného nástroje.

obrázek 10: špatně detekovaný text, namísto OK-CUS 76 bylo detekováno OK – CUS.2.6,

obrázek 11: špatně detekovaný text, text zde nebyl vůbec detekován

obrázek 12: špatně detekovaný text, text zde nebyl vůbec detekován

obrázek 13: správně detekovaný text



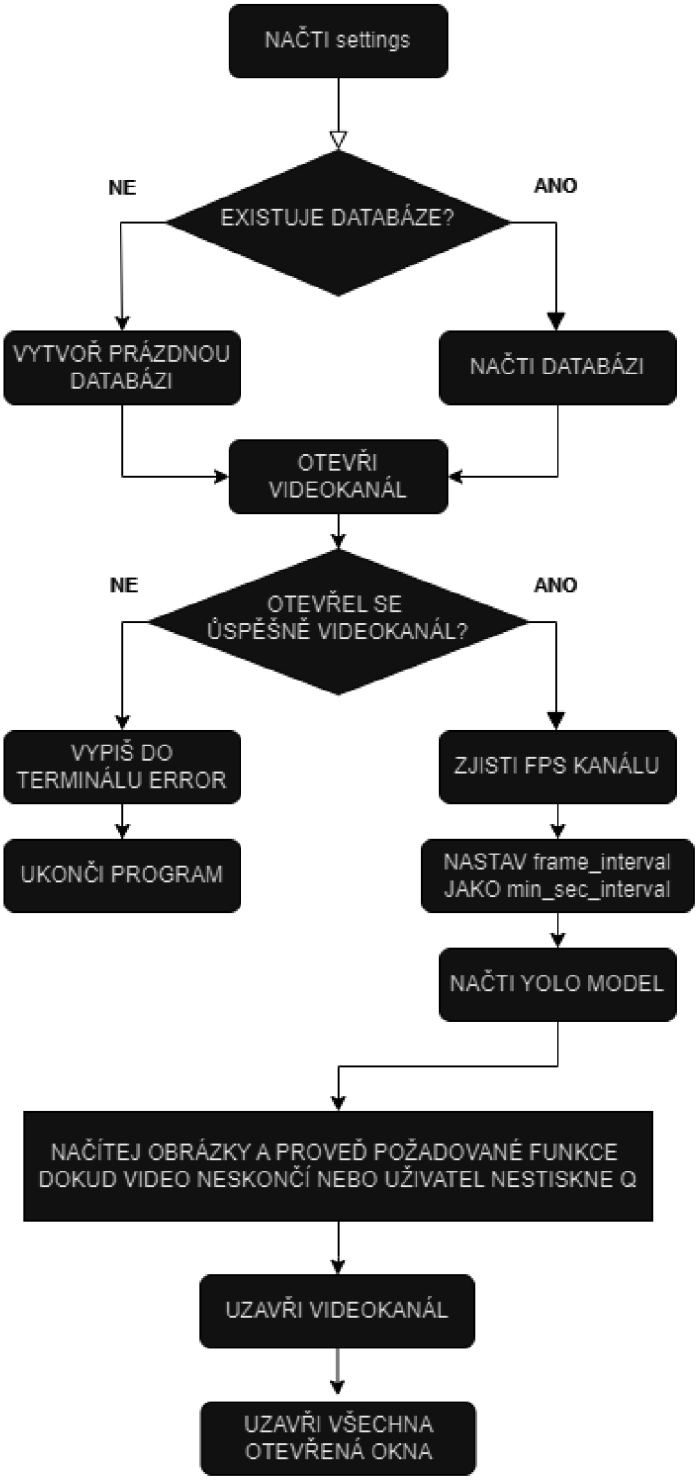
obrázek 14: správně detekovaný text

4.3. Shrnutí

V rámci této kapitoly byla provedena rešerše a úvodní testování postupů relevantních pro tuto práci. Rovněž zde byla navržena základní koncepce programu. Pro detekci textu byla zvolena síť Tesseract, která je oproti svým konkurentům výrazně rychlejší, a je poměrně jednoduchá na použití. Pro detekci letadel byla vybrána síť YOLOv8, která dosahuje velmi dobré přesnosti detekce v kombinaci s vysokou rychlostí zpracování. Rozpoznání textu nebylo spolehlivě dosaženo. Rozpoznání letadel bylo dosaženo v drtivé většině případů. Je třeba dodat, že obrázky byly pořízeny za velmi pěkného počasí. Nedokonalá detekce textu vede k tomu, že bude nutné v rámci postprocesingu detekovaný text upravit před zařazením do seznamu registračních značek a zavést uživatelskou kontrolu. Jako uživatelská kontrola bude použito uložení každého obrázku s detekovanou značkou v kombinaci s počtem detekcí značky, při malých počtech detekcí půjde pravděpodobně o špatné zařazení, a obrázky takových detekcí by se měly dodatečně zkontrolovat uživatelem.

5. Struktura a dokumentace programu

V rámci této kapitoly bude nejdříve představena idea a struktura celého programu, potom podrobně rozebrány jednotlivé funkční celky a jejich podfunkce. Na obrázku 15 vidíme základní strukturu programu, velká dlaždice s nápisem „načítej obrázky a proveď požadované funkce dokud video neskončí nebo uživatel nestiskne q“ je dále pro přehlednost rozepsaná v dalším diagramu na obrázku 17.

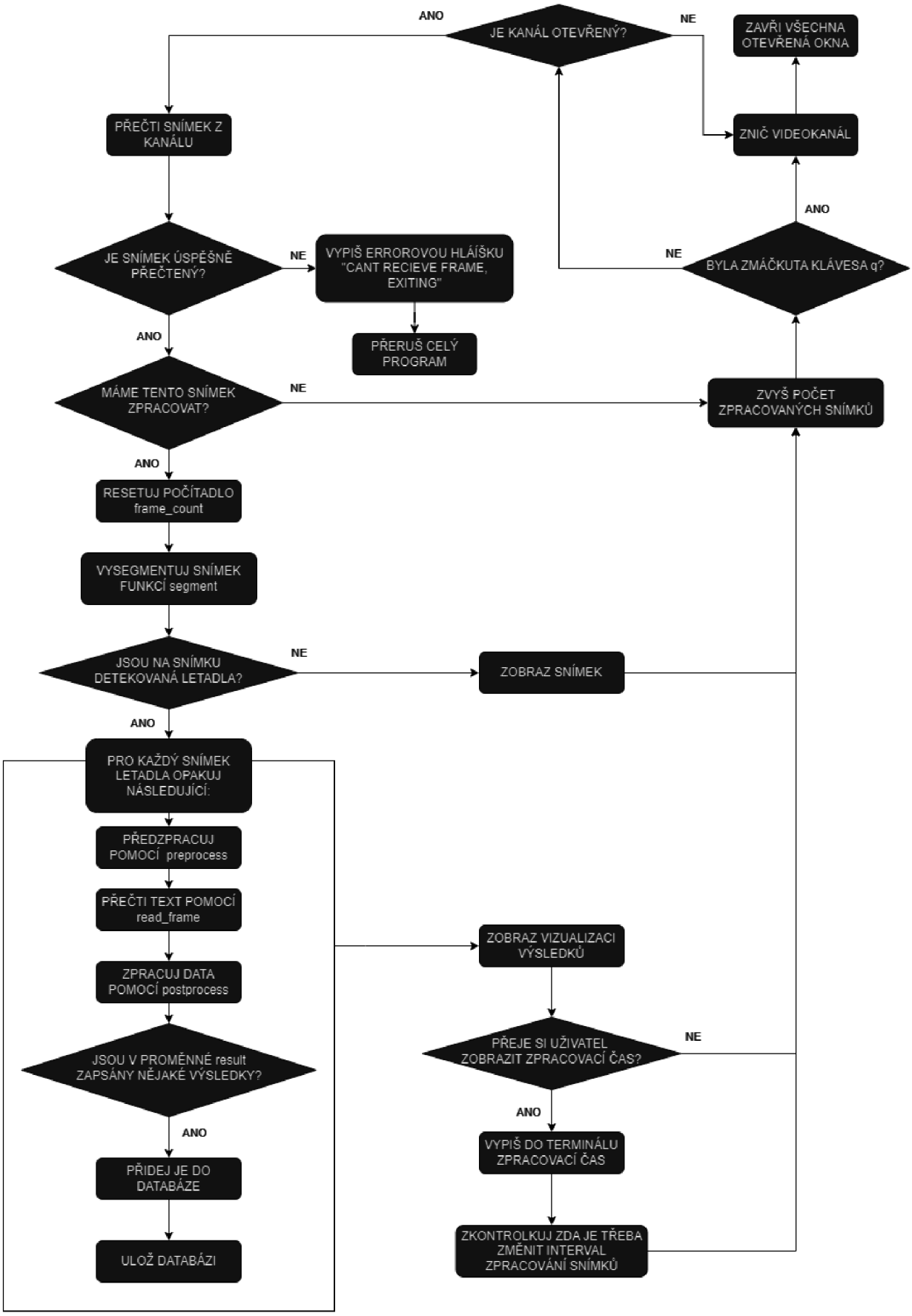


obrázek 15: základní struktura programu

Na obrázku 16 můžeme vidět kód se vším, co je třeba připravit pro načtení a zpracování snímku. Na řádku 31 načteme nastavení pomocí funkce settings, tak jak si je uživatel zadal. Na řádku 34 až 39 načítáme databázi, a když neexistuje vytvoříme novou, prázdnou. Následně na řádku 41 vytvoříme videokanál, který bude zachycovat video z kamery a na řádcích 44 až 46 zkontrolujeme, jestli se správně načetl, pokud ne, vypíšeme chybovou hlášku do terminálu a ukončíme program. Na řádku 48 zjistíme, kolik snímků za sekundu kamera pořizuje, na řádku číslo 49 určíme, jak často se mají snímky zpracovávat, a na řádku 50 zajistíme, aby byl i první snímek zpracovaný (viz kapitolu 5.2). Konečně na řádku 53 načteme model, který později použijeme pro rozpoznání letadel.

```
30  # load settings
31  tess_config, min_sec_interval, desired_conf_plane, desired_conf_text, draw_boxes_text, draw_boxes_planes, print_prosecing_time, save_unread_img = settings()
32
33  # if database exists
34  if os.path.isfile('database.json'):
35      # load database
36      database = load_database('database.json')
37  else:
38      # create new empty database
39      database = []
40
41  cap = cv2.VideoCapture(0) # Create a VideoCapture object and read from input file or camera
42
43  # Check if camera opened successfully
44  if not cap.isOpened():
45      print("Error opening video file")
46      exit()
47
48  fps = int(cap.get(cv2.CAP_PROP_FPS)) # get captured videos fps
49  frame_interval = min_sec_interval # define minimal interval for processing first frame
50  frame_count = load_frame_num(fps, frame_interval) # define frame_count so that the first frame is included
51
52  # Load a segmentation model
53  model = YOLO('yolov8n-seg.pt')
```

obrázek 16: kód přípravy programu na zpracování



obrázek 17: struktura zpracování snímků

Kód následující po přípravě na zpracování vidíme na obrázku 18, a jeho schéma na obrázku 17. Cyklus (řádek 56) bude postupně procházet jednotlivé snímky videa nebo kamerového záznamu, dokud bude existovat videokanál. Nejdříve začne odpočet doby na zpracování (řádek 59). Pak program přijme z videokanálu snímek (řádek 61). Zkontroluje, zda byl v pořádku přečten (řádek 63), pokud ne, vypíše chybovou hlášku (řádek 64) a ukončí cyklus (řádek 65). Program zpracuje jeden snímek každých několik sekund díky podmínce na řádku 68, a následně na řádku 71 resetuje počítadlo snímků (viz kapitolu 5.2). Poté je na řádku 74 provedena segmentace, v rámci té je na obrázku pomocí neuronové sítě YOLOv8n-seg nalezeno letadlo a je z něj vystřiženo, viz obrázek 19. Tento obrázek letadla bez pozadí je pak v rámci předzpracování převeden na šedobílý (řádek 80), což můžeme vidět na obrázku 20, a je na něm provedena detekce textu pomocí sítě Tesseract (řádek 81). Program po detekci ukáže vizualizaci detekovaných textů a letadel, pokud nejsou žádná nalezena, ukáže původní snímek (řádky 91 a 94).

```
55  # Read until video is completed
56  while(cap.isOpened()):
57
58      # start counting the time neede to process the frame
59      start_time = time.time()
60
61      ret, frame = cap.read() # Capture frame
62
63      if not ret: # if frame is read correctly ret is True
64          print("Can't receive frame. Exiting.")
65          break
66
67      # after a defined time, process the frame
68      if frame_count % load_frame_num(fps, frame_interval) == 0: # this if condition says, that the frame will be processed every min_sec_interval seconds
69
70          # reset frame count
71          frame_count = 0
72
73          # isolate plane region
74          planes_iso, planes_visual, iso_xy = segment(frame, desired_conf_plane, model, draw_boxes_planes)
75          # if there are any dedected planes
76          if planes_iso != []:
77              # for each detected plane
78              for img_num in range(len(planes_iso)):
79                  # read words from its bbox
80                  plane_prepro = preprocess(planes_iso[img_num])
81                  detected_words, visual = read_frame(plane_prepro,  planes_visual[img_num], iso_xy[img_num], tess_config, desired_conf_text, draw_boxes_text)
82                  # postprocess frame and detected words
83                  result = postprocess(detected_words, visual, save_unread_img)
84                  # if there are any results
85                  if result != []:
86                      # add them to database
87                      database = add_to_database(database, result)
88                      save_database(database)
89
90                  # Display the resulting frame
91                  cv2.imshow('Frame', visual)
92          else:
93              # Display the original frame
94              cv2.imshow('Frame', frame)
```

obrázek 18: kód pro zpracování snímků

obrázek 19: letadlo vystřižené z původního snímku

obrázek 20: převedení letadla na šedobílé

V rámci postprocesingu je pak veškerý detekovaný text zkontrolován (řádek 83). Jsou z něj odstraněny všechny znaky, které nemohou být součástí registrační značky, a pokud je nějaká jeho část zapsaná v leteckém rejstříku Úřadu pro civilní letectví, nebo alespoň odpovídá formátu registrační značky, je na obrázku potvrzeno přistávající letadlo. Tento snímek letadla, jeho registrační značka, aktuální datum a čas, počet dnešních detekcí tohoto letadla a umístění snímku v úložišti se při úspěšné detekci zapíšou do databáze (řádky 85 až 88). Příklad zápisu letadla s registrační značkou OK-VUT, které bylo detekováno 5 krát dne 1.5.2024 v časech mezi 17:45:05 a 17:45:20 můžeme vidět na obrázku 21.

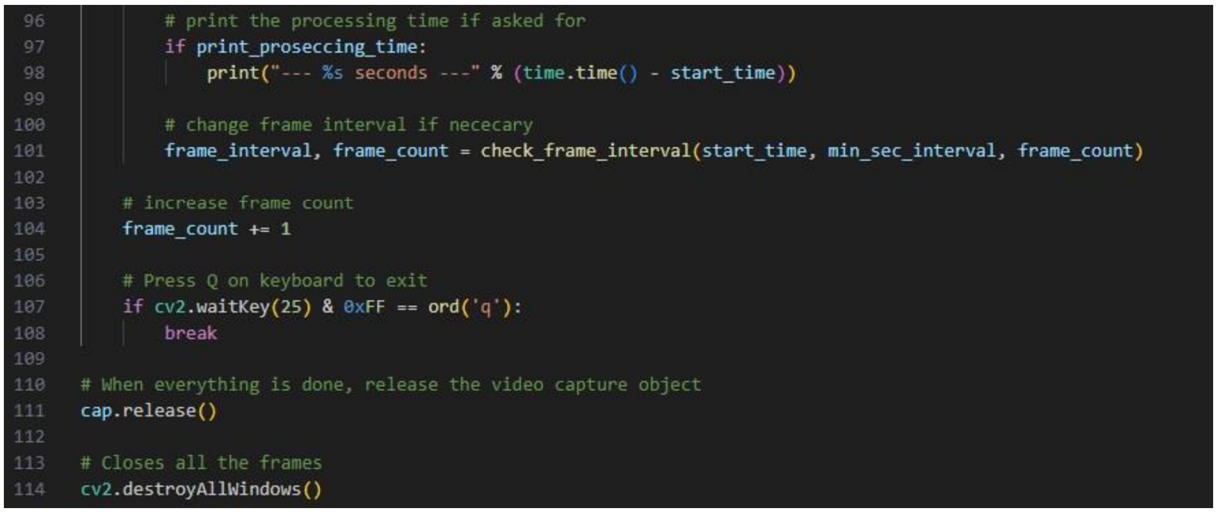

```
[
    [
        "2024-05-01",
        [
            [
                "OK-VUT"
            ],
            [
                5
            ],
            [
                "17-45-05",
                "17-45-08",
                "17-45-11",
                "17-45-17",
                "17-45-20"
            ],
            [
                "D:\\python_projects\\prumyslovy_projekt\\script\\frame_database\\2024-05-01\\17-45-05OK-VUT.jpg",
                "D:\\python_projects\\prumyslovy_projekt\\script\\frame_database\\2024-05-01\\17-45-08OK-VUT.jpg",
                "D:\\python_projects\\prumyslovy_projekt\\script\\frame_database\\2024-05-01\\17-45-11OK-VUT.jpg",
                "D:\\python_projects\\prumyslovy_projekt\\script\\frame_database\\2024-05-01\\17-45-17OK-VUT.jpg",
                "D:\\python_projects\\prumyslovy_projekt\\script\\frame_database\\2024-05-01\\17-45-20OK-VUT.jpg"
            ]
        ]
    ],
```


obrázek 21: úsek databáze se zaznamenaným letadlem s registrační značkou OK-VUT

Uživatel má také možnost si zvolit, že se do databáze zapíšou všechna detekovaná letadla, bez ohledu na to, jestli je u nich značka úspěšně zařazená, v takovém případě se místo registrační značky zapíše údaj „Unknown“, jak je možné vidět na obrázku 22.

obrázek 22: úsek databáze s neúspěšně přečtenými značkami

Kód následující po zpracování snímku můžeme vidět na obrázku 23. Pokud si to uživatel vyžádá, program následně vypočítá čas, který byl potřeba na zpracování snímku, a vypíše ho do terminálu (řádky 97 a 98). Podle času, potřebného na zpracování minulého snímku, se také určí minimální potřebný čas na zpracování následujícího snímku (řádek 101). Bez ohledu na to, jestli byl snímek zpracován nebo ne, se zvýší počet snímků, které jsme již prošli (řádek 104). Řádky 107 a 108 udávají, že při zmáčknutí klávesy q se celé zpracování zastaví. Po ukončení celého cyklu se uvolní videokanál (řádek 111) a zavřou všechna otevřená okna (řádek 114).



obrázek 23: kód s operacemi probíhajícími po zpracování snímku

5.1. Databáze a operace s ní

Databáze (tak jak ji lze vidět na obrázku 21 a 22) je pythonovský list, kde každá buňka odpovídá jednomu dni. Na nultém místě má uložené datum dne, kterému odpovídá, na prvním místě list, jehož každá buňka odpovídá jedné detekované registrační značce. Ta je pak dále dělena. Na místě nula je detekovaná značka, na místě jedna počet dnešních detekcí, na místě dva časy dnešních detekcí a na místě tři je umístění obrázku detekovaného letadla. Databáze se ukládá do formátu JSON.

Počet detekcí je tu pro zjednodušení vnímání uživatelem, pokud byla daná značka detekovaná jen jednou, pravděpodobně jde o chybnou detekci. Pokud byla detekována vícekrát v rámci jednoho časového úseku, jde s nejvyšší

pravděpodobností o správnou detekci. Toto je poté možno zkontrolovat na uložených snímcích.

S databází zdrojový kód interaguje několika funkcemi: load_database, add_to_database a save_database.

Zavolání funkce load_database můžeme vidět na obrázku 24. Hned na počátku celého programu se zjistí, jestli nějaká databáze už v minulosti byla vytvořena. Pokud ano, tato databáze se načte, pokud ne, vytvoříme novou prázdnou databázi.

```
24  # if database exists
25  if os.path.isfile('database.json'):
26      # load database
27      database = load_database('database.json')
28  else:
29      # create new empty database
30      database = []
```

obrázek 24: načtení databáze

Samotná funkce potom pomocí knihovny json jednoduše otevře dříve vytvořenou databázi a přepíše ji do proměnné, jak lze vidět na obrázku 25.

```
1   import json
2
3   def load_database(database_name):
4       # Opening JSON file
5       f = open(database_name,)
6       # returns JSON object as a dictionary
7       database = json.load(f)
8       # close JSON file
9       f.close()
10      return database
```

obrázek 25: funkce load_database

Funkce add_to_database a save_database jsou v kódu volány později. Po zpracování všech dat ze snímku se nám detekovaná značka letadla se všemi potřebnými údaji uloží do proměnné result, jak uvidíme níže v této kapitole. Nejdříve je zjištěno jestli jsou v ní nějaká uložená data, pokud ano, přidáme je do existující databáze a poté ji uložíme. Tento proces lze vidět na obrázku 26.

```
83                  if result != []:
84                      # add them to database
85                      database = add_to_database(database, result)
86                      save_database(database)
87
```

obrázek 26: uložení do databáze

Princip fungování save_database je analogický k load_database, viz obrázek 27.

```python
import json

def save_database(database):
    # Open JSON file
    out_file = open("database.json", "w")
    # dump JSON object
    json.dump(database, out_file, indent = 6)
    # close JSON file
    out_file.close()
```

obrázek 27: aktualizace souboru database.json

Funkce add_to_database je už poněkud složitější, a její pochopení vyžaduje znalost struktury proměnné result, pro ilustraci uvádím konkrétní příklad podoby této proměnné:

[

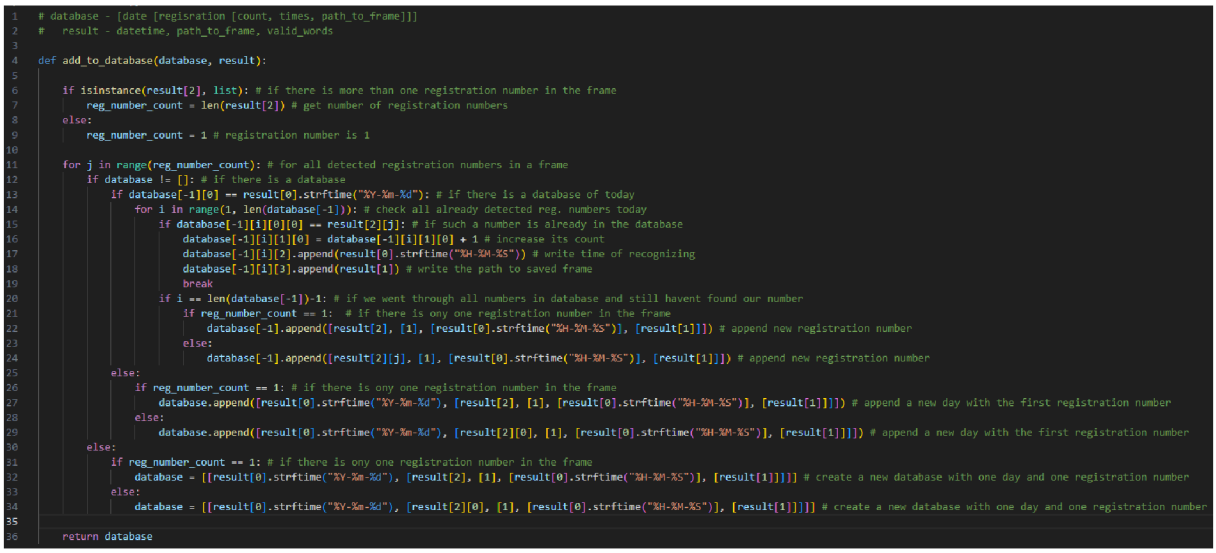
datetime.datetime(2024, 5, 16, 10, 2, 21, 761361),

'D:\\python_projects\\prumyslovy_projekt\\script\\frame_database\\2024-05-16\\10-02-21_OY-CGV.jpg',

['OY-CGV']

]

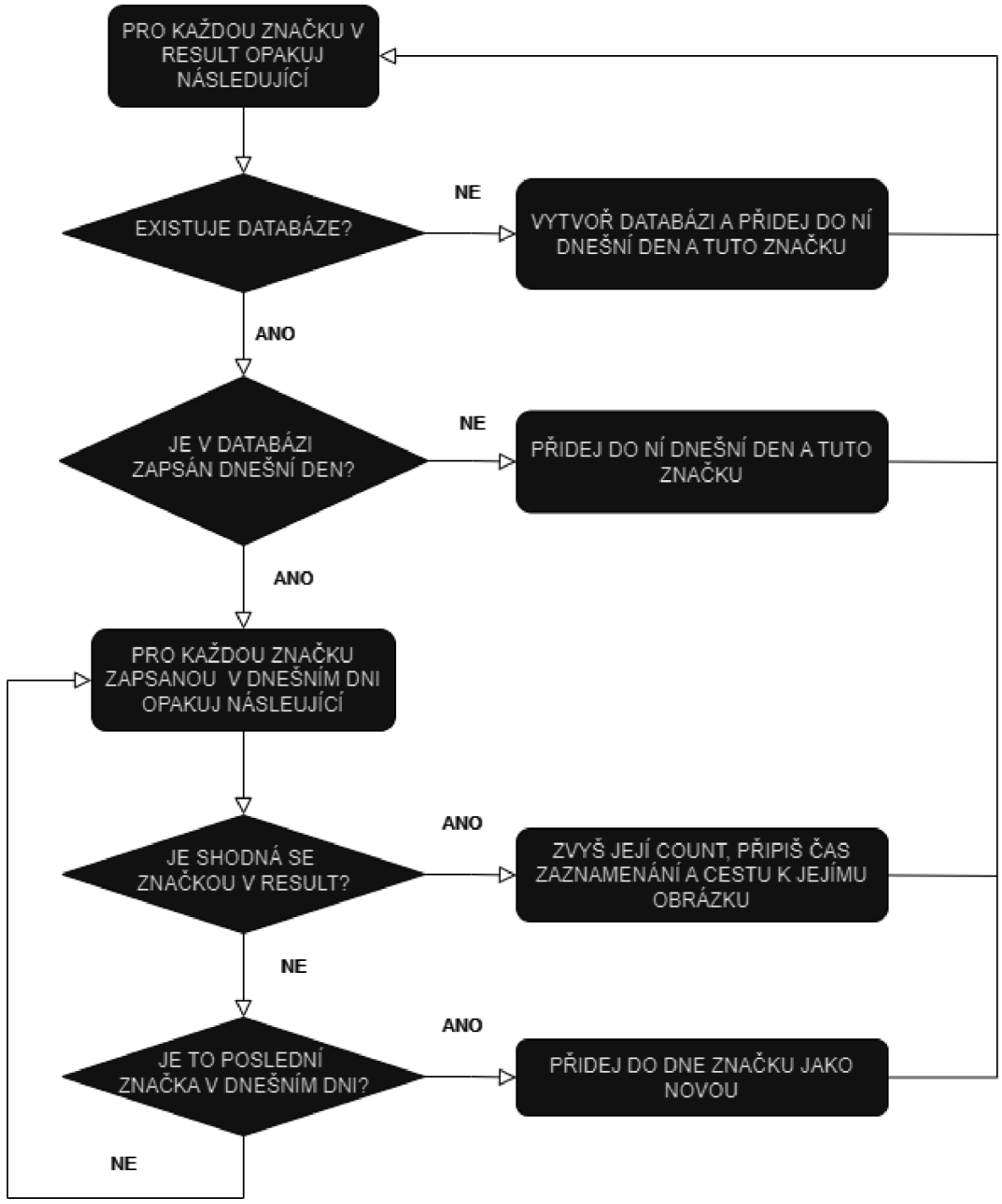
Na nultém místě je datetime objekt z knihovny datetime, ten jednoznačně určuje datum a čas, kdy byly tato data zpracována. Časové údaje jsou tu uvedeny v tomto pořadí: rok, měsíc, den, hodina, minuta, vteřina, sta tisíciny vteřin. Na prvním místě je uvedena cesta k složce, kam se uložil snímek, ze kterého byla data získána. Na druhém místě je pak samotná detekovaná registrační značka.



```
1  # database - [date [regisration [count, times, path_to_frame]]]
2  #   result - datetime, path_to_frame, valid_words
3
4  def add_to_database(database, result):
5
6      if isinstance(result[2], list): # if there is more than one registration number in the frame
7          reg_number_count = len(result[2]) # get number of registration numbers
8      else:
9          reg_number_count = 1 # registration number is 1
10
11     for j in range(reg_number_count): # for all detected registration numbers in a frame
12         if database != []: # if there is a database
13             if database[-1][0] == result[0].strftime("%Y-%m-%d"): # if there is a database of today
14                 for i in range(1, len(database[-1])): # check all already detected reg. numbers today
15                     if database[-1][i][0][0] == result[2][j]: # if such a number is already in the database
16                         database[-1][i][1][0] = database[-1][i][1][0] + 1 # increase its count
17                         database[-1][i][2].append(result[0].strftime("%H-%M-%S")) # write time of recognizing
18                         database[-1][1][3].append(result[1]) # write the path to saved frame
19                         break
20                 if i == len(database[-1])-1: # if we went through all numbers in database and still havent found our number
21                     if reg_number_count == 1:  # if there is ony one registration number in the frame
22                         database[-1].append([result[2], [1], [result[0].strftime("%H-%M-%S")], [result[1]]]) # append new registration number
23                     else:
24                         database[-1].append([result[2][j], [1], [result[0].strftime("%H-%M-%S")], [result[1]]]) # append new registration number
25             else:
26                 if reg_number_count == 1: # if there is ony one registration number in the frame
27                     database.append([result[0].strftime("%Y-%m-%d"), [result[2], [1], [result[0].strftime("%H-%M-%S")], [result[1]]]) # append a new day with the first registration number
28                 else:
29                     database.append([result[0].strftime("%Y-%m-%d"), [result[2][0], [1], [result[0].strftime("%H-%M-%S")], [result[1]]]) # append a new day with the first registration number
30         else:
31             if reg_number_count == 1: # if there is ony one registration number in the frame
32                 database = [[result[0].strftime("%Y-%m-%d"), [result[2], [1], [result[0].strftime("%H-%M-%S")], [result[1]]]] # create a new database with one day and one registration number
33             else:
34                 database = [[result[0].strftime("%Y-%m-%d"), [result[2][0], [1], [result[0].strftime("%H-%M-%S")], [result[1]]]] # create a new database with one day and one registration number
35
36     return database
```

obrázek 28: funkce add_to_database

Na obrázku 28 můžete vidět celou funkci add_to_database. Nejdříve se zjistí, kolik je detekovaných značek, a toto číslo se zapíše do proměnné reg_num_count (řádky 6 až 9). Poté následuje postup znázorněný na obrázku 29. Cyklus na řádku 11 prochází všechny právě detekované registrační značky, a pokud existuje databáze ve které už je něco zapsáno (řádek 13), projde všechny značky, které byly dnes do databáze zapsány (řádek 14) a zkontroluje, jestli některá z nich není právě detekovaná značka (řádek 15), pokud ano, přidá do databáze další zápis jejího zaznamenání a už dále databázi neprochází (řádky 16 až 19). Pokud bychom prošli celou databázi dnešního dne, a naši značku v ní nenašli (řádek 20), přidáme tam novou značku (řádek 21 až 24). Pokud neexistuje žádný dnešní zápis v databázi (řádek 25), přidáme do ní nový den s naší detekovanou značkou (řádky 26 až 29). Pokud v databázi zatím není vůbec žádný zápis (řádek 30), vytvoříme zcela nový zápis do databáze (řádky 31 až 34).



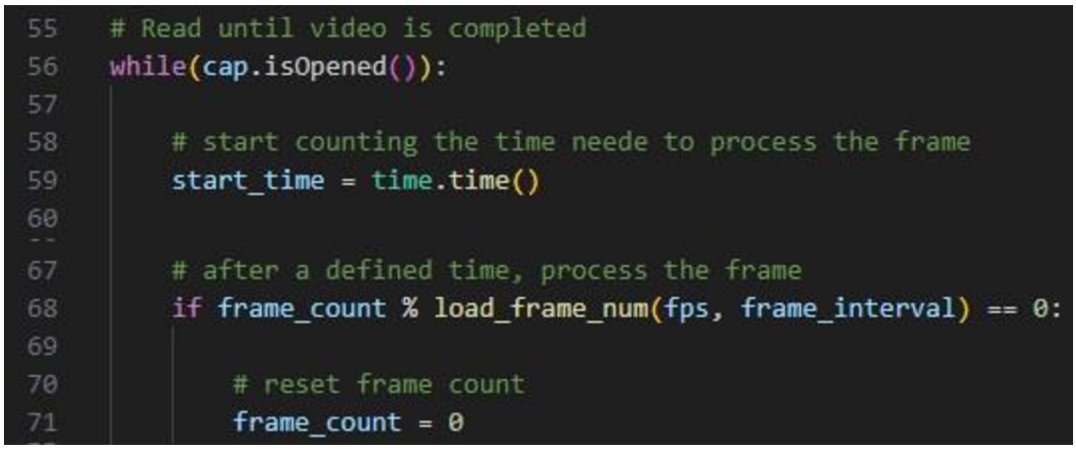
obrázek 29: zápis detekované značky do databáze

5.2. Zpracovávací čas a výpočet snímku k zpracování

V závislosti na hardwarových specifikacích kamery a počítače, na kterém bude program běžet, pravděpodobně nebude možné zpracovat každý snímek, který kamera pořídí. Proto je třeba zjistit, jak dlouho zpracování trvalo a odpočítat, kolik snímků mezitím proběhlo. Jako další snímek ke zpracování potom vezmeme až ten, který byl

zaznamenán s dostatečným odstupem. K tomu, jak tento proces přesně funguje budeme potřebovat znát několik proměnných:

- frame_count – počítá kolik snímků proběhlo od posledního zpracovaného snímku
- start_time – zaznamená čas ve kterém se začal zpracovávat snímek
- fps – udává kolik snímků za sekundu kamera pořizuje
- min_sec_interval – uživatelem zadaný minimální časový interval mezi dvěma snímky
- frame_interval – skutečný vyžadovaný časový interval mezi právě zpracovaným a dalším snímkem



obrázek 30: podmínka pro zpracování snímku, řádky 61 až 66 zde nejsou uvedeny, protože souvisí s jiným podsystémem programu

Na obrázku 30 můžeme vidět rozhodovací podmínku, která rozhoduje o tom, zda bude snímek zpracován, nebo ne. Na řádku 56 se nachází cyklus, který prochází snímky videa a na řádku 59 začne odpočet doby, kterou trvá snímek zpracovat (tato doba se na rozhodovacím procesu nepodílí). Na řádku 68 je podmínka, která říká, že se bude snímek zpracovávat pouze v případě, že lze beze zbytku vydělit počet uběhnutých snímků počtem snímků které měly uběhnout. Poté se resetuje počítání uběhnutých snímků (řádek 71).

Počet snímků, které měly uběhnout, je dán funkcí load_frame_num, kterou můžeme vidět na obrázku 31. Počet snímků který má uběhnout, se vypočítá jako počet sekund, který má uběhnout krát počet snímků za sekundu kamery (řádek 6). V případě že by toto číslo bylo desetinné, zaokrouhlí se na celá (řádek 6), a pokud by bylo menší než 1 (např. nula), nastaví se na 1 (řádky 9 a 10).

```
1   # frame interval gives interval between frames in seconds
2
3   def load_frame_num(fps, frame_interval):
4
5       # the number of frames, at wich we open a new frame
6       num = round(fps * frame_interval, 0)
7
8       # if frame_interval is too low, set it to 1
9       if 1 > num:
10          num = 1
11
12      return num
```

obrázek 31: funkce load_frame_num

Dále proběhne celé zpracování snímku, a až po jeho dokončení se s těmito časovými proměnnými opět pracuje. Tyto operace můžeme vidět na obrázku 32. Nejdříve na řádcích 98 a 99, pokud si to uživatel nastaví, program v terminálu vypíše jak dlouho trvalo snímek zpracovat. Poté na řádku 102 pomocí funkce check_frame_interval definujeme čas, který bude muset uplynout do dalšího zpracování. Nakonec zaznamenáme, že cyklus z řádku 56 prošel další snímek.

```
97          # print the processing time if asked for
98          if print_proseccing_time:
99              print("--- %s seconds ---" % (time.time() - start_time))
100
101         # change frame interval if nececary
102         frame_interval, frame_count = check_frame_interval(start_time, min_sec_interval, frame_count)
103
104     # increase frame count
105     frame_count += 1
106
```

obrázek 32: poslední operace s časovými proměnnými v rámci cyklu

Funkce check_frame_interval

Funkce check_frame_interval je vidět na obrázku 33. V jejím rámci se vypočítá čas který uběhl od začátku počítání času na zpracování snímku (řádek 6). Pokud je tento čas větší než min_sec_interval (řádek 9), bude následující snímek zpracován za hodnotu tohoto času zvýšenou o 0,01 vteřiny (řádek 11 a 12), a začne se od začátku počítat počet zpracovaných snímků (řádek 14). Jinak je čas zpracování dalšího snímku nastaven na čas udaný proměnnou min_sec_interval (řádky 15 až 17).

```python
import time

def check_frame_interval(start_time, min_sec_interval, frame_count):

    # calculate processing time of this iteration
    processing_time = (time.time() - start_time)

    # if the time to process last frame was higher than set minimal frame interval
    if processing_time > min_sec_interval:
        # round it to 1 decimal and increase it by 0.01 sec
        processing_time = round(processing_time, 2) + 0.01
        frame_interval = processing_time
        # reset frame counting
        frame_count = 0
    else:
        # otherwyse just leave it at min_sec_interval
        frame_interval = min_sec_interval

    return frame_interval, frame_count
```

obrázek 33: funkce check_frame_interval

5.3. Zpracování snímku

Kód pro zpracování každého snímku můžeme vidět na obrázku 34. Nejdříve na řádku 74 provedeme segmentaci. Na řádku 78 je poté cyklus, který pro každé detekované letadlo provede následující: předzpracuje vysegmentovaný region (řádek 80), provede na něm rozpoznání textu (řádek 81) a zpracuje výsledky které jsme při čtení dostali (řádek 83).

```
73      # isolate plane region
74      planes_iso, planes_visual, iso_xy = segment(frame, desired_conf_plane, model, draw_boxes_planes)
75      # if there are any dedected planes
76      if planes_iso != []:
77          # for each detected plane
78          for img_num in range(len(planes_iso)):
79              # read words from its bbox
80              plane_prepro = preprocess(planes_iso[img_num], max_px_value, treshold_C, relTrhMaskSize)
81              detected_words, visual = read_frame(plane_prepro,  planes_visual[img_num], iso_xy[img_num], tess_config, desired_conf_text, draw_boxes_text)
82              # postprocess frame and detected words
83              result = postprocess(detected_words, visual, save_unread_img)
```

obrázek 34: zpracování jednotlivého snímku

Na první pohled nás zarazí, že nejdříve probíhá segmentace a až potom předzpracování. Při druhém pohledu se ale tato zvláštnost hned objasní. Ve funkci segment totiž používáme síť YOLO k zjištění regionu, na kterém se nachází letadlo, a pro tu není předzpracování potřeba, jak bylo objasněno v rámci kapitoly 4. Ovšem ve funkci read_frame používáme síť Tesseract k zjištění a čtení textu v obrázku, a tato síť už si určité předzpracování vyžaduje, opět viz kapitolu 4.

Funkce segment

Pro pochopení funkce segment budeme potřebovat znát význam těchto proměnných:

- results – proměnná, do které se ukládají výstupy sítě YOLO
- img – kamerový snímek na vstupu
- desired_conf – jistota, se kterou chceme mít letadla detekovaná
- iso_crop – obrázek ořezaný na bounding box letadla, na kterém se nachází pouze letadlo, ostatní pixely jsou černé
- visual – obrázek, který slouží k vizualizaci výsledku

- iso_xy – souřadnice obrázku iso_crop v původním obrázku
- planes_iso – proměnná pro uložení všech iso_crop obrázků
- planes_visual – proměnná pro uložení všech visual obrázků
- planes_iso_xy – proměnná pro uložení všech iso_xy souřadnic

Definice funkce segment je vidět na obrázku 35 a její logické schéma na obrázku 36. Nejdříve se na řádcích 8 až 11 deklarují proměnné na uložení výsledků, poté na řádku 14 použijeme síť YOLO k nalezení všech letadel v obrázku, a uložení dat do proměnné results.

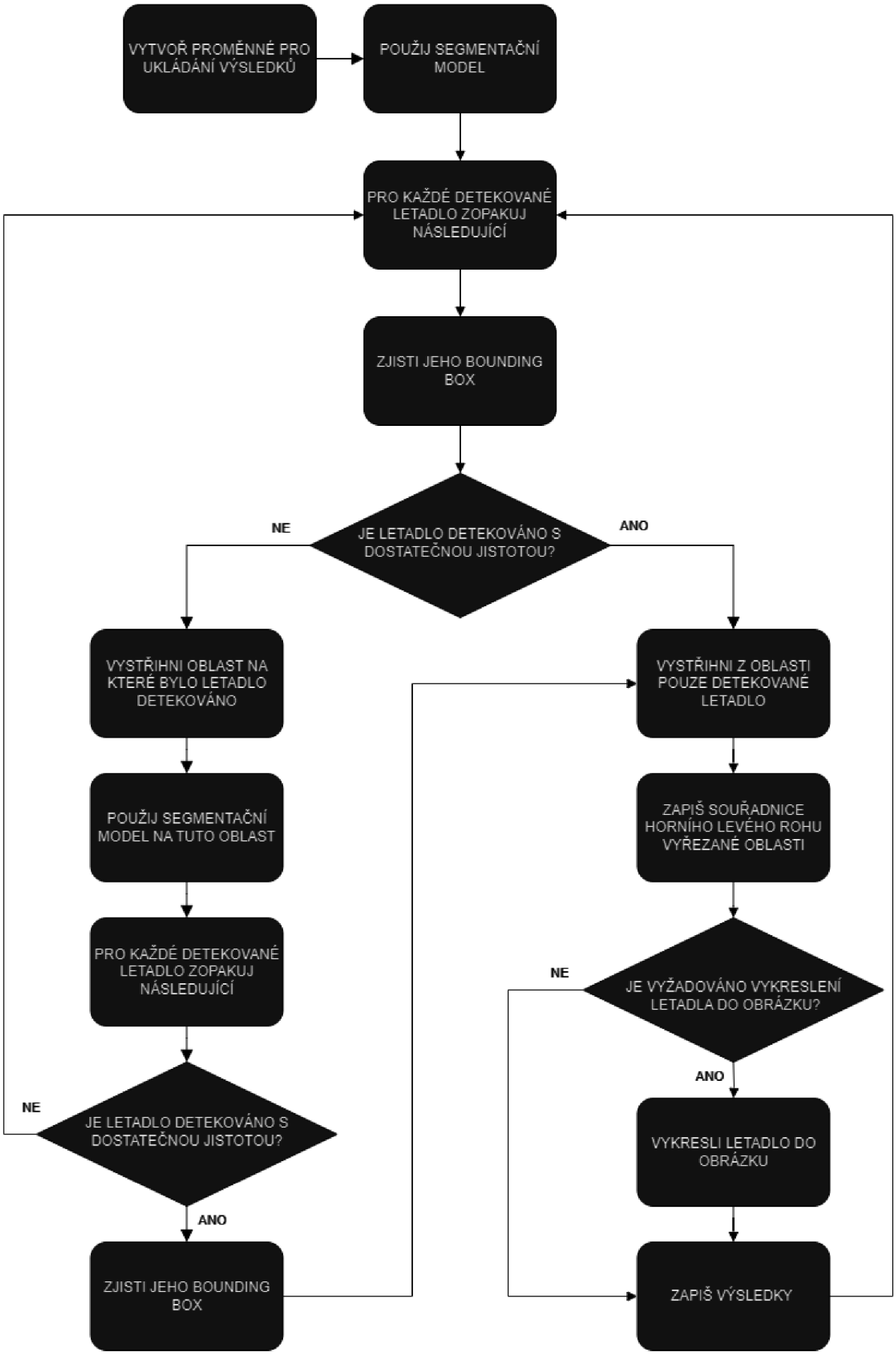
Následně na řádku 17 projdeme všechna detekovaná letadla, zjistíme souřadnice jejich bounding boxu (řádek 19) a jestli byla detekována s požadovanou přesností (řádek 22).

```
1  from ultralytics import YOLO
2  from make_croped_imgs import make_croped_imgs
3  from draw_visual_plane import draw_visual_plane
4  import numpy as np
5
6
7  def segment(img, desired_conf, model, draw_boxes_planes):
8
9      planes_visual = []
10     planes_iso = []
11     planes_iso_xy = []
12
13     # Predict with the model
14     results = model(img, classes=[4], verbose=False)
15
16     # for each detected airplane
17     for r in range(len(results[0].boxes.conf)):
18         # get bbox values
19         x1, y1, x2, y2 = results[0].boxes.xyxy[r].cpu().numpy().squeeze().astype(np.int32)
20
21         # if there an airplane is detected with sufficient confidence
22         if results[0].boxes.conf[r].numpy() >= desired_conf:
23             # get image with only the detected region and the rest black, cropped to the region bbox
24             iso_crop = make_croped_imgs(img, results, x1, y1, x2, y2, r)
25             # make visualization by drawing bbox with confidence numbler into original frame
26             if draw_boxes_planes:
27                 visual = draw_visual_plane(img, x1, y1, x2, y2, results[0].boxes.conf[r].numpy())
28             else:
29                 visual = img
30
31             # write bbox starting coordinates
32             iso_xy = [x1, y1]
33             # add images to their respective lists
34             planes_iso.append(iso_crop)
35             planes_visual.append(visual)
36             planes_iso_xy.append(iso_xy)
37
38         else:
39             # crop out a detail of suspected airplane
40             # from original img
41             zoomed_img = np.array(img)
42             zoomed_img  = zoomed_img[y1:y2, x1:x2]
43
44             # predict on this detail
45             results_zoomed = model(zoomed_img, classes=[4], verbose=False)
46
47             # for each detected airplane
48             for i in range(len(results_zoomed[0].boxes.conf.numpy())):
49                 # if there is an airplane with sufficient confidence
50                 if results_zoomed[0].boxes.conf[i] >= desired_conf:
51                     # get its bbox values
52                     x1_zoomed, y1_zoomed, x2_zoomed, y2_zoomed = results_zoomed[0].boxes.xyxy[i].cpu().numpy().squeeze().astype(np.int32)
53                     # get image with only the detected region and the rest black, cropped to only region bbox
54                     iso_crop = make_croped_imgs(zoomed_img, results_zoomed, x1_zoomed, y1_zoomed, x2_zoomed, y2_zoomed, i)
55                     # write bbox starting coordinates
56                     iso_xy = [x1+x1_zoomed, y1+y1_zoomed]
57                     # make visualization by drawing bbox with confidence numbler into original frame
58                     if draw_boxes_planes:
59                         visual = draw_visual_plane(img, x1+x1_zoomed, y1+y1_zoomed, x2+x2_zoomed, y2+y2_zoomed, results_zoomed[0].boxes.conf[i].numpy())
60                     else:
61                         visual = img
62                     # add images to their respective lists
63                     planes_iso.append(iso_crop)
64                     planes_visual.append(visual)
65                     planes_iso_xy.append(iso_xy)
66
67     return planes_iso, planes_visual, planes_iso_xy
```

obrázek 35: funkce segment

Pokud ano, vystřihneme z obrázku letadlo a začerníme všechny pixely okolo něj (řádek 24). Pak vykreslíme vizualizaci detekovaného letadla, nebo, pokud to není žádáno, ponecháme originální snímek (řádky 26 až 29). Potom přepíšeme získaný obrázek izolovaného letadla, vizualizaci a pozici letadla v obrázku do patřičných proměnných (řádky 32 až 36).

Pokud letadlo nebylo detekováno s dostatečnou přesností, vyřízneme z obrázku část, ve které máme podezření na letadlo (řádky 41 a 42). A opět na něm zkusíme provést rozpoznání pomocí modelu YOLO. Model totiž vždy převede rozměry obrázku na nejbližší násobek 32 pixelů, nejméně však na 640 [16]. Z toho vyplývá, že pokud se jedná o obrázek s příliš nízkým rozlišením, tímto mu můžeme v detekci pomoct. Poté provedeme algoritmus analogický s algoritmem předchozího odstavce. Pro každé detekované letadlo v přiblíženém obrázku (řádek 48) zjistíme, jestli si je model dostatečně jistý, že je to skutečně letadlo (řádek 50), zjistíme jeho souřadnice v rámci přiblíženého obrázku (řádek 52), vytvoříme obrázek, kde je pouze letadlo s černým pozadím (řádek 54), zapíšeme nové souřadnice letadla v původním obrázku (řádek 56), zakreslíme vizualizaci je-li to požadováno (řádky 58 až 61), a nakonec vše uložíme do patřičných proměnných (řádky 63 až 65).



obrázek 36: schéma funkce segment

Funkce make_croped_imgs

Kód definující funkci make_croped_imgs můžeme vidět na obrázku 37. Nejdříve vytvoříme prázdnou proměnnou knihovny numpy o velikosti původního obrázku (řádek 7). Poté z výsledků detekovaných modelem extrahujeme kontury detekovaného regionu (řádek 10), a převedeme je na potřebný typ dat (řádek 13). Převedeme konturu na 3D objekt, abychom ji mohli dále používat s naší proměnnou vytvořenou na řádku 7 [25] (řádek 16), a zakreslíme ji do ní (řádek 19 až 23). Pak převedeme masku na barevnou verzi (řádek 26). Je třeba si uvědomit, že maska je stále složena pouze z černých a bílých pixelů, a proto uděláme průnik masky s původním obrázkem (řádek 29). Nakonec vystřihneme z celého obrázku pouze region, ve kterém se nachází letadlo (řádek 32).

```
1  import numpy as np
2  import cv2
3
4
5  def make_croped_imgs(frame, results, x1, y1, x2, y2, r):
6      # Create binary mask
7      b_mask = np.zeros(frame.shape[:2], np.uint8)
8
9      # Extract contour result
10     contour = results[0].masks[r].xy.pop()
11
12     # Changing the type
13     contour = contour.astype(np.int32)
14
15     # Reshaping
16     contour = contour.reshape(-1, 1, 2)
17
18     # Draw contour onto mask
19     _ = cv2.drawContours(b_mask,
20                          [contour],
21                          -1,
22                          (255, 255, 255),
23                          cv2.FILLED)
24
25     # Create 3-channel mask
26     mask3ch = cv2.cvtColor(b_mask, cv2.COLOR_GRAY2BGR)
27
28     # Isolate object with binary mask
29     isolated = cv2.bitwise_and(mask3ch, frame)
30
31     # Crop image to object region
32     iso_crop = isolated[y1:y2, x1:x2]
33
34     return iso_crop
```

obrázek 37: funkce make_croped_imgs

Funkce draw_visual_plane

Tato poměrně jednoduchá funkce na vstupu přijme obrázek jako proměnnou frame, souřadnice boundong boxu jako x1, x2, y1 a y2, a jistotu, s jakou bylo letadlo detekováno jako proměnnou conf. Na řádku 5 nakreslíme na místo určené souřadnicemi červený čtverec. Na řádku 7 vytvoříme text, který se napíše do tohoto čtverce. Konečně na řádku 9 tento text zapíšeme do levého spodního rohu nakresleného čtverce. Výsledný obrázek potom vrátíme na výstupu. Kód této funkce můžeme vidět na obrázku 38.

```
1  import cv2
2
3  def draw_visual_plane(frame, x1, y1, x2, y2, conf):
4      # draw red redtangle around detected airplane
5      visual = cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255))
6      # create string to be written
7      char = 'Airplane conf: ' + str(conf)[0:5]
8      # write string into the frame
9      visual = cv2.putText(visual, char, (x1 + 3, y2 - 7), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 0, 225), 1, cv2.LINE_AA)
10     return visual
```

obrázek 38: funkce draw_visual_plane

Funkce preprocess

Tuto funkci můžeme vidět na obrázku 39. V rámci této funkce prostě a jednoduše převedeme obraz na odstíny šedi (řádek 6), tím se obraz převede na šedobílý a do šedobílého formátu. Poté, na řádku 8, ho převedeme zpátky do barevného formátu, tím pádem s ním můžeme dále pracovat a například do něj barevně kreslit, ale obraz zůstane šedobílý.

```
1  import cv2
2
3  def preprocess(img):
4
5      # convert to greyscale
6      img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
7      # convert back to BGR img (image will stay greyscale, but we can now draw into img)
8      img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
9      return img
```

obrázek 39: funkce preprocess

Funkce read_frame

V rámci této funkce, kterou můžeme vidět na obrázku 47, získáme z obrázku text. Pro pochopení této relativně složité funkce je třeba nejdříve pochopit její části. První část, na kterou se zaměříme, je funkce check_conf. Tuto funkci můžeme vidět na obrázku 40.

```python
def check_conf(data, desired_conf):
    # prepare variables for confidence validation
    amount_boxes = len(data['text'])
    detected_words = []
    detected_boxes = []

    # for every detected text
    for box in range(amount_boxes):
        # if we achived desired confidence
        if float(data['conf'][box]) >= desired_conf:
            # put detected text to database
            detected_words.append(data['text'][box])
            # put its bbox to database
            detected_boxes.append([data['left'][box], data['top'][box], data['width'][box], data['height'][box]])

    return detected_words, detected_boxes
```

obrázek 40: funkce check_conf

Vstupní proměnná jménem data je dodaná sítí Tesseract, a obsahuje všechna data extrahovaná z obrázku. Vstupní proměnná desired_conf je číslo, které udává s jakou jistotou musí být text detekován, abychom ho považovali za validní. Na řádku 3 zjistíme, kolik textů bylo detekováno. Na řádcích 4 a 5 pak vytvoříme proměnné k budoucímu ukládání validních slov a jejich bounging boxů. Na řádku 8 až 14 pak prověříme, zda bylo slovo detekováno s požadovanou jistotou, a pokud ano, zapíšeme ho a jeho bounding box do patřičných proměnných, tyto proměnné nám pak funkce vrací.

Druhou funkcí, kterou bude třeba vysvětlit, je funkce get_word_dimensions, která převádí souřadnice bounding boxů obou slov z formátu x, y, šířka, výška na x, y, x, y. Tato jednoduchá funkce je vidět na obrázku 41.

```python
def get_word_dimensions(detected_boxes, word_1, word_2):
    # get coordinates of bbox sides for word_1
    word_1_lef = detected_boxes[word_1][0]
    word_1_top = detected_boxes[word_1][1]
    word_1_rig = detected_boxes[word_1][0] + detected_boxes[word_1][2]
    word_1_bot = detected_boxes[word_1][1] + detected_boxes[word_1][3]
    # get coordinates of bbox sides for word_2
    word_2_lef = detected_boxes[word_2][0]
    word_2_top = detected_boxes[word_2][1]
    word_2_rig = detected_boxes[word_2][0] + detected_boxes[word_2][2]
    word_2_bot = detected_boxes[word_2][1] + detected_boxes[word_2][3]

    return word_1_lef, word_1_top, word_1_rig, word_1_bot, word_2_lef, word_2_top, word_2_rig, word_2_bot
```

obrázek 41: funkce get_word_dimensions

V některých případech Tesseract sice přečte text správně, ale rozdělí jej na více separátních částí, toto můžeme vidět na obrázku 42. Pokud tedy chceme registrační značku správně kategorizovat, je třeba nejdříve spojit všechny texty, které jsou podezřele blízko sebe. Ke zjištění, zdali je třeba texty spojit, slouží funkce words_should_merge, kterou můžeme vidět na obrázku 43.

obrázek 42: správně detekovaný text rozdělený na dvě části, místo „OK-CES“ je detekováno „OK“ a „-CES“

```
1  def words_should_merge(detected_boxes, detected_words, word_1, word_2, word_1_lef, word_1_top, word_1_rig, word_1_bot, word_2_lef, word_2_top, word_2_rig, word_2_bot):
2      # if top bbox borders are aproximately aligned
3      if abs(word_1_top - word_2_top) <= (detected_boxes[word_1][3] + detected_boxes[word_2][3])/2 :
4          # if bot bbox borders are aproximately aligned
5          if abs(word_1_bot - word_2_bot) <= (detected_boxes[word_1][3] + detected_boxes[word_2][3])/2 :
6
7              # get box distance
8              box_distance = min(abs(word_1_lef - word_2_rig), abs(word_2_lef - word_1_rig))
9              # get average width of a letter in both boxes
10             letter_width = detected_boxes[word_1][2]/len(detected_words[word_1]) + detected_boxes[word_2][2]/len(detected_words[word_2])/2
11
12             # if the boxes are apart by not more than a letter and a half
13             if box_distance <= letter_width*1.5 :
14                 should_merge = True
15             else:
16                 should_merge = False
17
18         else:
19             should_merge = False
20
21     else:
22         should_merge = False
23
24     return should_merge
```

obrázek 43: funkce words_should_merge

Do této funkce vstupuje seznam detekovaných bounding boxů a slov, dále číslo word_1 a word_2, které určují které slova a boxy v pořadí budou prošetřovány, a pak souřadnice stěn obou boxů. Na řádku 3 se prověří, jestli jsou horní stěny boxů obou slov na obrázku v podobné úrovni na ose y. Za dostatečně blízké považujeme, že je jejich vzdálenost menší nebo rovná průměru výšek obou boxů. Analogicky na řádku 5 zjistíme, jestli jsou v podobné úrovni i spodní stěny boxů. Na řádku 8 potom zjistíme vzdálenost boxů v horizontální ose jako menší ze vzdáleností pravé od levé stěny obou boxů. Poté na 10 řádku zjistíme průměrnou šířku jednoho písmene obou boxů. A to tak že pro oba boxy vydělíme jejich šířku počtem písmen v nich, a tato dvě čísla pak zprůměrujeme. Poslední podmínka je na řádku 13, tady prověřujeme, jestli od sebe nejsou boxy vzdáleny o více než písmeno a půl. Pokud boxy projdou všemi těmito podmínkami, vrací funkce proměnnou should_merge jako pravdivou, v opačném případě jako nepravdivou.

Další funkce, kterou je třeba pochopit, je funkce join_detected_words. Tato funkce spojí dvě slova dohromady ve správném pořadí, tak aby slovo vlevo bylo připojeno zleva a slovo vpravo bylo připojeno zprava. Tuto funkci můžeme vidět na obrázku 44. Na vstupu přijmeme souřadnice levé stěny obou slov jako word_1_lef a word_2_lef, a text obou slov jako word_1 a word_2. Podmínka na řádku 4 je splněna, pokud je slovo word_1 více vlevo než slovo word_2. Pak slova spojíme v pořadí word_1 word_2. Pokud podmínka není splněna, spojíme je v pořadí word_2 word_1. Vracíme spojené slovo.

```python
def join_detected_words(word_1_lef, word_2_lef, word_1, word_2):

    # if the word more to the left is word_1
    if min(word_1_lef, word_2_lef) == word_1_lef:
        # join two words woth word_1 to the left and word_2 to the right
        new_word = word_1 + word_2
    else:
        # join two words woth word_2 to the left and word_1 to the right
        new_word = word_2 + word_1

    return new_word
```

obrázek 44: funkce join_detected_words

Předposlední funkce, potřebná pro pochopení read_frame, je funkce get_new_bbox, kterou můžeme vidět na obrázku 45. Get_new_bbox na vstupu přijme souřadnice stěn dvou bounding boxů ve formátu x, y, x, y, a převede je na nový bounding box, který obsahuje oba staré ve formátu x, y, šířka, výška.

```python
def get_new_bbox(word_1_lef, word_1_top, word_1_rig, word_1_bot, word_2_lef, word_2_top, word_2_rig, word_2_bot):

    # find the correct one of the two possible new bbox borders
    new_left = min(word_1_lef, word_2_lef)
    new_top = min(word_1_top, word_2_top)
    new_height = max(word_1_bot, word_2_bot) - new_top
    new_width = max(word_1_rig, word_2_rig) - new_left

    # write new bbox
    new_bbox = [new_left, new_top, new_width, new_height]

    return new_bbox
```

obrázek 45: funkce get_new_bbox

Na řádcích 4 a 5 zjistíme, která ze starých souřadnic je víc nahoře a vlevo, a definujeme ji jako novou horní a levou souřadnici bounding boxu. Na řádcích 6 a 7 zjišťujeme, která ze starých souřadnic je více dole a vpravo, odečteme od nich novou levou a horní hranici, a definujeme výsledek jako novou šířku a výšku bounding boxu. Vracíme nový bounding box.

```python
import cv2

def draw_visual_text(detected_words, detected_boxes, visual, img_xy):
    # for all detected words
    for word in range(len(detected_words)):
            # get bbox of text
            (x, y, width, height) = (detected_boxes[word][0], detected_boxes[word][1], detected_boxes[word][2], detected_boxes[word][3])
            # put rectangle around text
            visual = cv2.rectangle(visual, (img_xy[0] + x, img_xy[1] + y), (img_xy[0] + x + width, img_xy[1] + y+height), (0, 255, 0))
            # write recognized text
            visual = cv2.putText(visual, detected_words[word], (img_xy[0] + x, img_xy[1] + y - 3), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 255, 0), 1, cv2.LINE_AA)

    return visual
```

obrázek 46: funkce draw_visual_text

Poslední funkcí kterou popíšeme pro pochopení read_frame je funkce draw_visual_text, kterou můžeme vidět na obrázku 46. Na vstupu této funkce přijímáme všechna detekovaná slova jako detected_words, jejich bounding boxy ve vyřezaném obrázku letadla jako detected_boxes, původní snímek do kterého se celý

proces vizualizuje jako visual, a souřadnice vyřezaného obrázku letadla v původním snímku jako img_xy. Na řádku 5 je potom cyklus, který pro každé detekované slovo zjistí jeho bounding box (řádek 7), do obrázku visual zakreslí okolo detekovaného textu zelený obdélník (řádek 9), a nakonec k obdélníku zapíše detekovaný text zeleným písmem (řádek 11). Vracíme potom obrázek se všemi zakreslenými texty jako visual.

```
40  def read_frame(img, visual, img_xy, myconfig, desired_conf, draw_boxes_text):
41
42      # read text in preprocessed img
43      data = pytesseract.image_to_data(img, config=myconfig, output_type=Output.DICT) # use tesseract to read
44      # check if text was detected with sufficient confidence
45      detected_words, detected_boxes = check_conf(data, desired_conf)
46
47      # we will go through every detected word, some of them multiple times
48      word_1 = 0
49      while word_1 <= len(detected_words):
50          # for every word in the list after word_1
51          for word_2 in range(word_1+1, len(detected_words)):
52              # get dimensions of words
53              (word_1_lef, word_1_top, word_1_rig, word_1_bot, word_2_lef, word_2_top, word_2_rig, word_2_bot) = get_word_dimensions(detected_boxes, word_1, word_2)
54              # check if words are next to each other on aproximately same (height) level, and therefore are possibly one word and should be merged
55              if words_should_merge(detected_boxes, detected_words, word_1, word_2, word_1_lef, word_1_top, word_1_rig, word_1_bot, word_2_lef, word_2_top, word_2_rig, word_2_bot):
56                  # join the two detected words together
57                  new_word = join_detected_words(word_1_lef, word_2_lef, detected_words[word_1], detected_words[word_2])
58                  # get new bbox
59                  new_bbox = get_new_bbox(word_1_lef, word_1_top, word_1_rig, word_1_bot, word_2_lef, word_2_top, word_2_rig, word_2_bot)
60                  # replace word_1 bbox and word with the new one, and delete ones belonging to word_2
61                  detected_boxes[word_1] = new_bbox
62                  detected_words[word_1] = new_word
63                  detected_boxes.pop(word_2)
64                  detected_words.pop(word_2)
65                  # iterate again with the new word
66                  word_1 -= 1
67                  break
68          word_1 += 1
69
70      # if you want to draw boxes around detected text
71      if draw_boxes_text:
72          visual = draw_visual_text(detected_words, detected_boxes, visual, img_xy)
73
74      return detected_words, visual
```

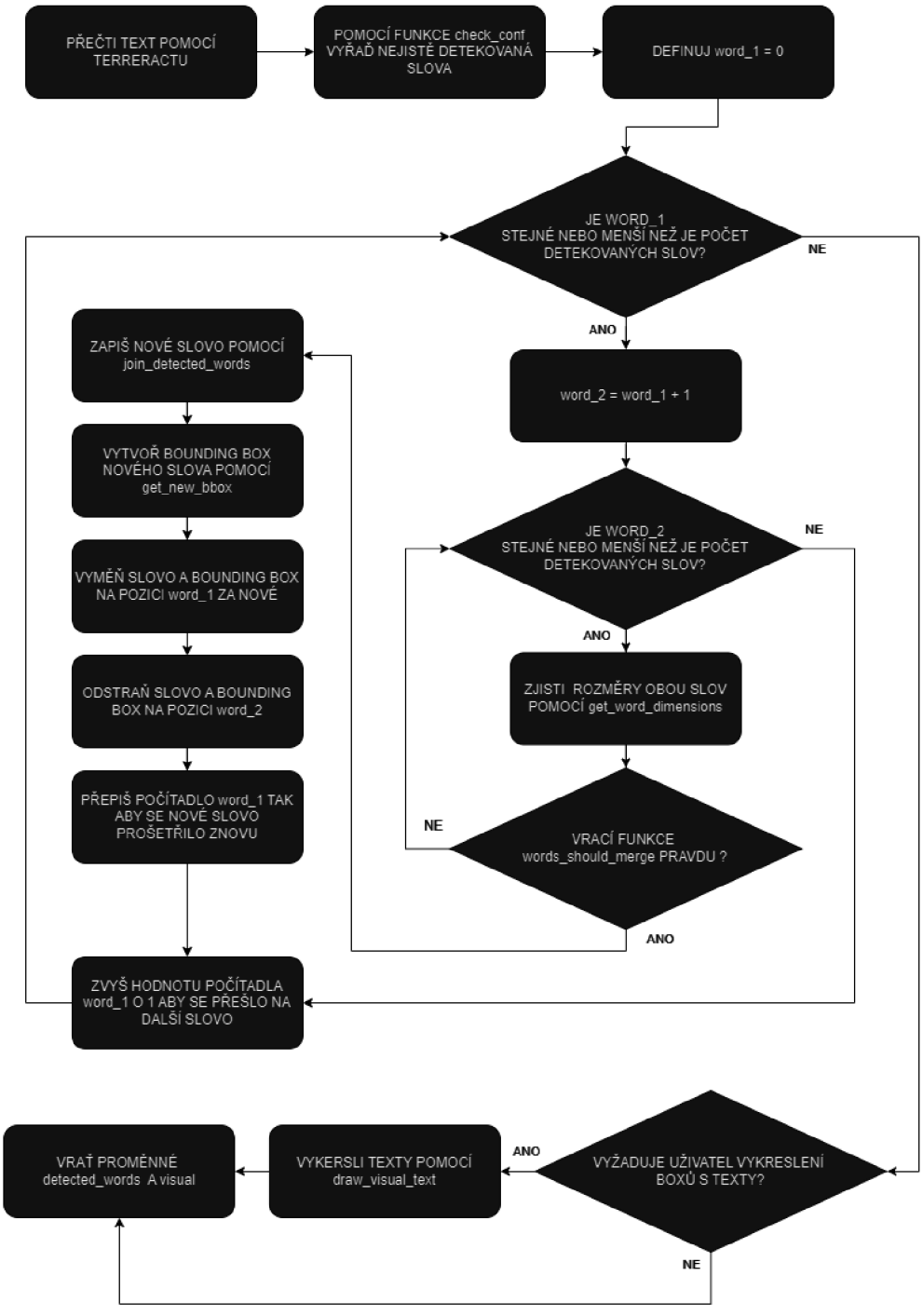
obrázek 47: funkce read_frame

Princip samotné funkce read_frame je znázorněný na obrázku 48. A její kód je na obrázku 47. Vstupují do ní následující proměnné:

- img - izolované letadlo vyřezané z původního snímku
- visual - potom původní snímek upravený pro vizualizaci detekovaných dat jako
- img_xy - souřadnice obrázku img ve snímku visual
- myconfig – požadované nastavení pro detekci textu
- desired_conf – požadovaná jistota detekce textu
- draw_boxes_text – určuje, jestli budeme vizualizovat detekovaný text

Na řádku 43 použijeme síť Tesseract k detekci a přečtení textu na obrázku. Následně na řádku 45 pomocí funkce check_conf vytřídíme pouze ta slova, která byla detekována s dostatečnou jistotou. Řádek 48 nám definuje proměnnou word_1. Tato proměnná slouží jako počítadlo, které nám bude v následujícím cyklu určovat kolikáté slovo seznamu právě porovnáváme s ostatními. Cyklus začínající na řádku 49 totiž slouží k porovnání všech slov mezi sebou, a pokud jsou daná slova v obrázku těsně vedle sebe a ve stejné výšce, spojí je dohromady. Podmínka cyklu vlastně udává, že cyklus skončí, pokud už žádné další slovo v proměnné detected_words není. V těle cyklu na řádku 51 vidíme další cyklus, který projde všechna detekovaná slova, která následují za slovem na pozici word_1, a přiřadí jim číslo word_2. Zjistí souřadnice bounding boxů obou slov (řádek 53). Zkontroluje, jestli nejsou těsně za sebou (řádek 55) a pokud ano, spojí je dohromady (řádek 57). Dále získá bounding box nového slova (řádek 59) a nahradí jím původní dvě slova v proměnných detected_words a detected_boxes (řádky 61 až 64). Potom odečte od počítadla word_1 jedničku (řádek 66), čímž způsobí že se bude v další iteraci cyklu z řádku 49 prošetřovat toto nově

vytvořené slovo. Nakonec ukončí cyklus z řádku 51 (řádek 67), který už nemá smysl, protože bylo vytvořeno zcela nové slovo. Úplně nakonec přičteme jedničku k word_1, čímž přejdeme na další detekované slovo. Po dokončení celého cyklu z řádku 51 a zkontrolovaní kombinací všech slov se tato slova vizualizují, pokud je to uživatelovo přání (řádky 71 a 72). Funkce vrací proměnné detected_words a visual.



obrázek 48 schéma funkce read_frame

Funkce postprocess

V rámci funkce postprocess zjistíme, jestli nějaké z detekovaných slov odpovídá formátu registrační značky (řádek 6). Pokud ano (řádek 8), zapíšeme do snímku aktuální datum a čas a uložíme ho (řádek 10). Nakonec vytvoříme proměnou result (řádek 12), do které uložíme detekovanou registrační značku, cestu k uloženému snímku a aktuální čas. Pokud žádné detekované slovo neodpovídá registrační značce, ale uživatel si přeje uložit všechny snímky s detekovaným letadlem (řádek 14), definujeme registrační značku jako „Unknown“ (řádek 16), stejně jako výše do snímku přepíšeme momentální datum a čas (řádek 18) a vše uložíme do proměnné result (řádek 20). Pokud nesplňujeme ani jednu z těchto podmínek, definujeme result jako prázdnou proměnnou. Funkce postprocess pak na výstupu vrací proměnnou result. Kód funkce můžeme vidět na obrázku 49.

```
4   def postprocess(detected_words, frame, save_unread_img):
5       # get any possible registration numbers from the words
6       valid_words = postprocess_words(detected_words)
7       # if there are any valid words
8       if valid_words != []:
9           # save frame and add timedate to it
10          path_to_frame, datetime = postprocess_frame(frame, valid_words)
11          # write down the results
12          result = [datetime, path_to_frame, valid_words]
13      # if there are any no words, but we want to save frame anyway
14      elif save_unread_img:
15          # define that the words are unknown
16          valid_words = ['Unknown']
17          # save frame and add timedate to it
18          path_to_frame, datetime = postprocess_frame(frame, valid_words)
19          # write down the results
20          result = [datetime, path_to_frame, valid_words]
21      # if there are any no words, and we dont want to save frame anyway
22      else:
23          # return empty results
24          result = []
25
26      return result
```

obrázek 49: funkce postprocess

Teď objasníme použité podfunkce. Jako první vysvětlíme funkci postprocess_words, kterou můžeme vidět na obrázku 50. Do této funkce vstupuje pouze seznam detekovaných textů jako old_list. Pro pochopení této funkce musíme nejdříve pochopit následující proměnné:

- break_flag – tato proměnná říká, že máme skončit cyklus prohledávání registračních značek, k tomu nastane pokud značka byla nalezena v seznamu registračních značek UCL
- new_list – seznam značek, které byly správně kategorizovány

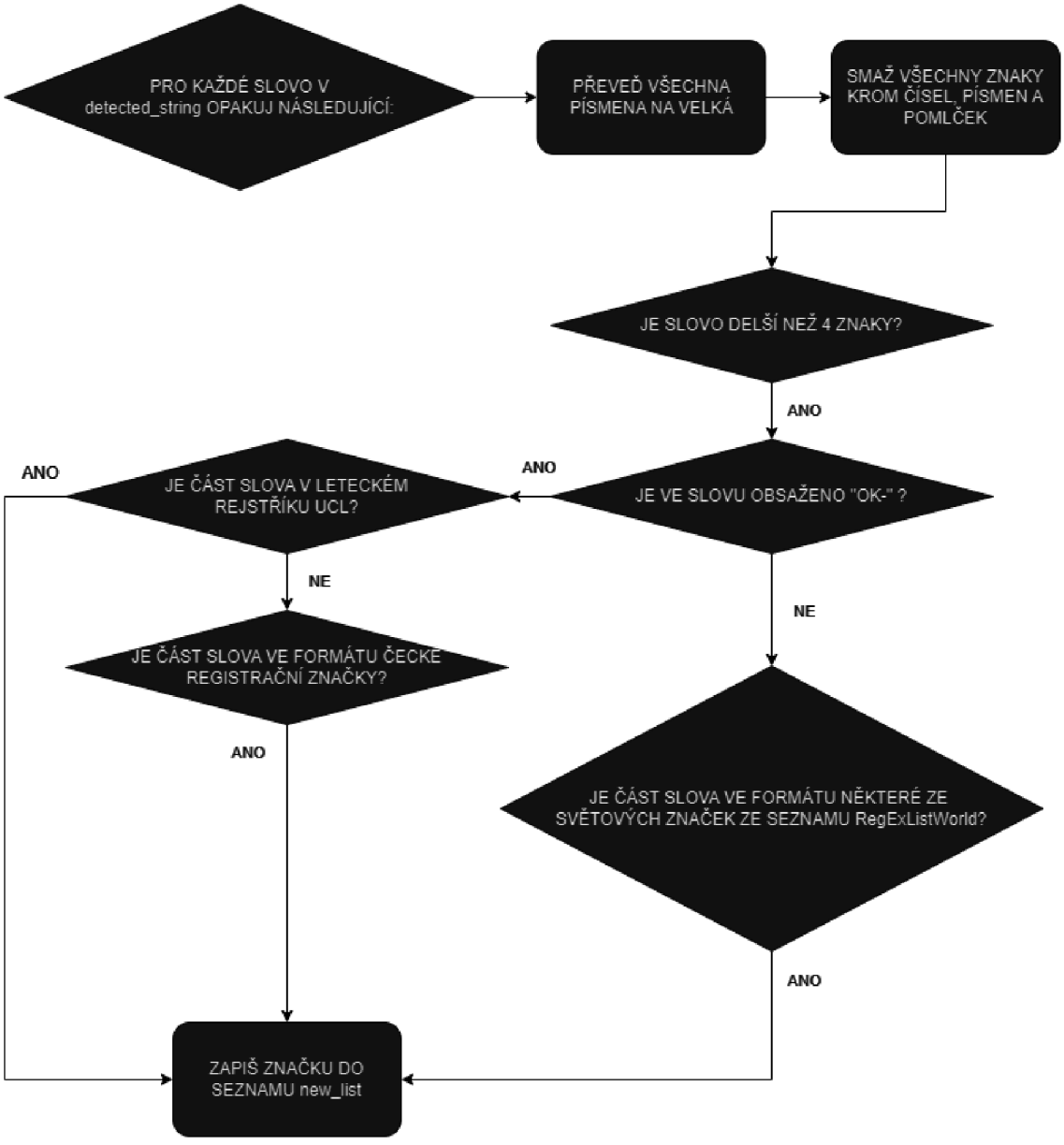
- ragistracni_znacky_database – seznam všech registračních značek uvedených v leteckém rejstříku UCL
- RegExListCzech – seznam možných formátů českých registračních značek
- RegExListWorld – seznam možných formátů registračních značek některých světových států, jedná se o všechny evropské státy a státy blízko Evropy, jako jsou např. státy severní Afriky nebo kavkazské státy, a jiné velké země jako např. Spojené státy americké
- detected_string – jedno z přečtených slov
- match – značka, která byla úspěšně detekována, protože se nachází v detekovaném textu, a sedí na některý z formátů v seznamech RegExListCzech nebo RegExListWorld

```
4   def postprocess_words(old_list):
5
6       break_flag = False # set breakflag to false
7       new_list = [] # new list for output
8       registracni_znacky_database = pd.read_csv('registracni_znacky.csv') # database of registration numbers
9       RegExListCzech = pd.read_csv('RegEx_list_czech.csv')
10      RegExListWorld = pd.read_csv('RegEx_list_world.csv')
11
12      for i in range(len(old_list)): # for each detected string
13          detected_string = old_list[i].upper() # turn it to upper case
14
15          # erase unwanted characters, udělat přes regexy
16          detected_string = re.sub('[^A-Z0-9-]', "", detected_string, count=0, flags=0)
17
18          if len(detected_string) >= 4: # if the string is equal or longer than 4 char (reg. plates cant be shorter)
19              if 'OK-' in detected_string: # if the string includes 'OK-'
20
21                  for j in range(len(registracni_znacky_database)): # look if any of the registration plates in database are included in the string
22                      if registracni_znacky_database.iat[j, 0] in detected_string: # if they are
23                          new_list.append(registracni_znacky_database.iat[j, 0]) # write the registration plate to a new list
24                          # go to next detected string
25                          break_flag = True
26                          # dont check the rest of registration numbers
27                          break
28
29                  if break_flag :
30                      break_flag = False
31                      break
32
33                  # check if it fits any regex for czech registrations numbers
34                  for RegEx in range(len(RegExListCzech)):
35                      match = re.findall(RegExListCzech.iat[RegEx, 0], detected_string)
36                      # if there is a match
37                      if match != []:
38                          # write the registration plate to a new list
39                          new_list.append(match[0])
40                          # dont check the rest of regexes
41                          break
42
43              else:
44                  # check if it fits any regex for foreign registrations numbers
45                  for RegEx in range(len(RegExListWorld)):
46                      match = re.findall(RegExListWorld.iat[RegEx, 0], detected_string)
47                      if match != []: # if there is a match
48                          new_list.append(match[0]) # write the registration plate to a new list
49                          # to go to next detected string
50                          break
51      return new_list
```

obrázek 50: funkce postprocess_words

Celý algoritmus funkce je znázorněn na obrázku 51. Nejdříve na řádku 6 definujeme break_flag jako False, na řádku 7 new_list jako prázdnou proměnnou a na řádcích 8 až 10 načteme jednotlivé databáze. Na řádku 12 nám potom začíná cyklus, který postupně prošetří všechny detekované texty. V rámci cyklu nejdříve na řádku 13 získáme detekovaný text, a všechny detekovaná písmena převedeme na velká. Následně na řádku 16 odstraníme všechny znaky, které nemůžou být v registrační značce, tedy všechny krom čísel, písmen a pomlček. Pokud je takový text delší nebo roven čtyřem znakům (řádek 18), zjistíme jestli se jedná o registrační značku. Pokud

je jeho součástí text „OK-“ (řádek 19), nejdříve prohledáme letecký rejstřík (řádek 21), pokud se v detected_string nachází některá ze značek rejstříku (řádek 22), přidáme tuto registrační značku do proměnné new_list (řádek 23), do proměnné break_flag zadáme, že můžeme přejít na další detekovaný text, protože došlo k úspěšnému rozpoznání značky (řádek 25) a přestaneme prohledávat značky z leteckého rejstříku (řádek 27). Kód na řádcích 29 až 31 zajistí, že pokud byla úspěšně rozpoznaná značka, přejdeme na další detekovaný text. Pokud zatím nebyla úspěšně rozpoznaná značka, zkontrolujeme na řádku 34 a 35 jestli část detected_string alespoň nesedí na nějaký formát české registrační značky. Pokud ano (řádek 37), zapíšeme ji na řádku 39 do proměnné new_list, a přestaneme prohledávat (řádek 41). Pokud se v daném textu nenachází „OK-“ (řádek 43), tak zjistíme, jestli text nesedí na některý ze světových formátů (řádky 45 a 46). Pokud ano (řádek 47), zapíšeme ji do proměnné new_list a přejdeme na další detekovanou značku (řádek 50).



obrázek 51: schéma funkce postprocess_words

Další objasníme funkci postprocess_frame, jejíž kód můžeme vidět na obrázku 52. Tato funkce je zodpovědná za vytvoření konečné podoby vizualizace a její uložení. Na vstupu přímá dosavadní verzi vizualizace jako frame, a seznam úspěšně detekovaných značek, vrací cestu k uloženému obrázku a čas uložení.

```
6   def postprocess_frame(frame, valid_words):
7       date_time = datetime.datetime.now() # get date and time
8       time_str = date_time.strftime("%H-%M-%S")
9       date_str = date_time.strftime("%Y-%m-%d")
10
11      # if there is more than one valid word
12      if type(valid_words) == list:
13          # merge them into single string
14          valid_words = tuple(valid_words)
15          valid_words = "_".join(valid_words)
16
17      # get frame shape
18      height, width, _ = frame.shape
19
20      # write time and date into frame
21      frame = cv2.putText(frame, date_str + "  " + time_str, (int(width/10), int(height/10)), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2, cv2.LINE_AA)
22
23      # get path for saving img
24      path_here = str(os.path.dirname(os.path.abspath(__file__))) # get path to directory of script being run
25      path_to_frame_dir = os.path.join(path_here, 'frame_database', date_str) # get path to directory for saving images
26      path_to_frame = os.path.join(path_to_frame_dir, time_str + "_" + valid_words + '.jpg') # get path to the frame
27
28      # check if dir to save frames exists, if not then create it
29      if not os.path.isdir(path_to_frame_dir):
30          os.mkdir(path_to_frame_dir)
31
32      # save img
33      cv2.imwrite(path_to_frame, frame)
34
35      return path_to_frame, date_time
```

obrázek 52: funkce postprocess_frame

Na řádku 7 zjistíme aktuální datum a čas, na řádcích 8 a 9 každý z nich zvlášť převedeme na text. Čas je zapsán ve formátu HH-MM-SS a datum ve formátu RR-MM-DD. Pokud je značek více než jedna (řádek 12), spojíme je do jednoho textu ve formátu ZNACKA1_ZNACKA2_ZNACKA3 a tak dále (řádky 14 a 15). Na řádku 18 zjistíme rozměry obrázku, a následně do horního levého rohu zapíšeme aktuální datum a čas (řádek 21). Na řádcích 24 až 26 získáme potřebnou cestu pro uložení obrázku. Podmínka na řádku 29 zjistí, zda existuje složka pro ukládání obrázku, a když ne, vytvoří ji. Nakonec se obrázek uloží (řádek 33).

Funkce make_planes_database

Tato funkce do hlavního programu přímo nevstupuje, je ovšem velmi důležitá pro jeho funkci, vytváří totiž seznam registračních značek leteckého rejstříku UCL. Proklikává automaticky jednotlivé stránky rejstříku a kopíruje z nich zapsané registrační značky. Pokud by bylo v budoucnu žádoucí databázi aktualizovat, je třeba k tomu využít právě make_planes_database. Vzhledem k její povaze je ale nutné zdůraznit, že je závislá na podobě internetových stránek UCL, a pokud je někdo v budoucnu změní, tento program už nebude fungovat. Můžeme ji vidět na obrázku 53.

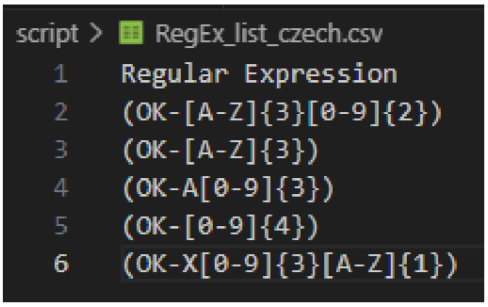
```python
# do not phisicaly open chrome
options = Options()
options.add_argument("--headless=new")

# go to desired website
driver = webdriver.Chrome(options=options)
driver.get("https://lr.caa.cz/letecky-rejstrik")

time.sleep(2)  # Wait for the new results to load
page_source = driver.page_source

# Parse the page source with BeautifulSoup
driver.find_element(By.CLASS_NAME, "btn-primary").click()
soup = BeautifulSoup(page_source, 'html.parser')

# get pagination
pagination = (soup.find('span',class_='my-auto').getText())
splitPagination = pagination.split()

# define how many pages we have to click through
how_many_pages = int(splitPagination[4]) / int(splitPagination[2])
how_many_pages = math.ceil(how_many_pages)

list1 = []

# for each page in registery
for i in range(how_many_pages):
    time.sleep(2)  # Wait for the new results to load

    # get registation numbers
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'html.parser')
    search_results = soup.find_all('td', class_='column-3')

    # add registation numbers to list
    for result in search_results:
        list1.append(result.strong.getText())

    # if you are not on the last page, click next page
    if i != how_many_pages - 1:
        driver.find_element(By.CLASS_NAME, "fa-forward").click()

# sort list from longest to shortest string

list1.sort(key=len)
list1.reverse()

df = pd.DataFrame({'znacka':list1}) # write name of list data
df.to_csv('registracni_znacky.csv', index = False) # save list as .csv

driver.quit() # close driver
```

obrázek 53: funkce make_planes_database

Na řádcích 11 a 12 nastavujeme, že se použitý prohlížeč fyzicky neotevře, ale bude běžet na pozadí. Na řádcích 15 a 16 otevíráme prohlížeč Chrome a jdeme na webovou adresu leteckého rejstříku. Poté dvě sekundy počkáme, až se stránka načte (řádek 18), a následně načteme obsah stránky do proměnné page_source (řádek 19). Dále na řádku 22 odklikneme soubory cookies a na řádku 23 zdrojový kód stránky importujeme do knihovny BeautifulSoap, která s takovými kódy umí pracovat. Řádky 26 a 27 zjišťují, kolik je v rejstříku zapsaných letadel a kolik se jich zobrazuje na stránku, díky těmto datům potom na řádcích 30 a 31 zjistíme, kolika stránkami se musíme dohromady proklikat. Na řádku 33 vytvoříme list, do kterého budeme značky zapisovat. Cyklus na řádku 36 prochází každou stránku rejstříku. Vždy počká dvě vteřiny, aby se stránka načetla (řádek 37). Na řádku 40 a 41 opět předá zdrojový kód nově načtené stránky do BeautifulSoap knihovny, a na řádku 42 vždy zkopíruje všechny elementy z třetího sloupce, tedy registrační značky. Na řádcích 45 a 46 je přepíše do seznamu registračních značek. Pak se přesouvá na další stánku. Po dokončení celého procesu list seřadí podle velikosti od nejdelší po nejkratší (řádky 54 a 55), což je důležité pro jednodušší optické hledání v seznamu a řešení možných problémů (například při vývoji programu bylo zjištěno že značka OK-EXS má v rejstříku za sebou omylem napsanou mezeru a bylo třeba ji manuálně opravit). Nakonec databázi uložíme (řádky 57 a 58) a zavřeme prohlížeč (řádek 60).

Funkce make_regex_list

Tato funkce podobně jako předchozí make_planes_database do hlavního programu přímo nevstupuje, ale má na svědomí generování seznamu možných formátů registračních značek cizích zemí. Seznam formátů České republiky byl vytvořen ručně a můžeme ho vidět na obrázku 54.



obrázek 54: regulérní výrazy pro možné formáty registračních značek české republiky

Funkce make_regex_list se skládá ze dvou částí, v první části programu definuje jaké kombinace znaků před a za pomlčkou jsou pro registrační značky možné, a to ve formě regulárních výrazů, druhá je pak skládá dohromady a vytváří jejich seznam.

```
# define possible prefix/suffix combinations
pre_suf_fix_list = [
    [
        [
            '[A-Z]{2}'
        ],
        [

            'F'
        ]
    ],

    [
        [
            '[A-Z]{2}[0-9]{1}'
        ],
        [
            'EC'
        ]
    ],

    [
        [
            '[A-Z]{3}'
        ],
        [
            'YA','ZA','7T','C3','D2','V2','LQ','EK', '4O','P4','VH','OE','C6','A9C','S2','8P','EW','OO','V3','TY','VP',
            'VQ-B','A5','E7','A2','PP','PR','PS','PT','PU','V8','LZ','XT','9U','XU','TJ','CF','D4','VQ-C','TL',
            'TT','CC','D6','TN','9S','E5','TI','9A','5B-D','OY','J2','J7','HI','4W','HC','SU','YS','3C','ES','3DC',
            'ET','DQ','OH','F','TR','C5','4L','D','9G','SX','J3','TG','3X','J5','8R','HH','HR','B-H','B-K','B-L',
            'HA','TF','VT','PK','EP','YI','EI','4X','TU','6Y','JA','JR','ZJ','JY','4YB','5Y','T3','9K','RDPL','YL',
            'OD','7P','A8','5A','LY','LX','9H','5T','ER','3A','CN','C9','XY','XZ','C2','PH','PJ','YN','5U','5N','Z3','LN',
            'A40','SP','CR', 'CS', 'A7', 'YR','Z6','VQ-H', 'T7', '6V', '6W', 'YU', 'OM', 'S5', 'H4', '6O', 'Z8',
            'EC','EM','4R','ST','PZ','SE','HB','YK','TS','TC','EZ','UR','4U','N','7O','9J','Z'
        ]
    ],

```

obrázek 55: úsek první části programu make_regex_list

Na obrázku 55 vidíme příklad toho, jak první část programu vypadá. Na řádku 5 až 13 je definováno, že pro dvě písmena za pomlčkou je možné mít před pomlčkou pouze F. Na řádku 15 až 22 zase že pro dvě písmena, za kterými je jedno číslo je možné mít pouze předponu EC. Na řádcích 24 až 38 vidíme, že pro tři písmena za pomlčkou je možných mnoho různých předpon.

```
# create list for storing regular expresions
RegEx_list_world = []

# create all possible combinations of regular expressions
for suffix_num in range(len(pre_suf_fix_list)): # for every possible suffix
    plane_suffix = str(pre_suf_fix_list[suffix_num][0][0]) # get the suffix
    for country in range(len(pre_suf_fix_list[suffix_num][1])): # for every of its possible prefixes
        country_prefix = str(pre_suf_fix_list[suffix_num][1][country]) # get prefix

        # combine it together
        if country_prefix == 'N': # remember that USA does not use - after prefix
            RegEx = '(' + country_prefix + plane_suffix + ')'
            RegEx_list_world.append(RegEx)
        else: # every other country
            RegEx = '(' + country_prefix + '-' + plane_suffix + ')'
            RegEx_list_world.append(RegEx)

# sort list alphabeticaly
RegEx_list_world.sort()
# reverse it (this is nessesary to detect longer regexes first)
RegEx_list_world.reverse()
# write name of list data
df_wd = pd.DataFrame({'Regular Expression':RegEx_list_world})
# save list as .csv
df_wd.to_csv('RegEx_list_world.csv', index = False)
```

obrázek 56: druhá část programu make_regex_list

Na obrázku 56 můžeme vidět druhou část programu, která je zodpovědná za vytvoření seznamu. Na řádku 258 vytvoříme zatím prázdný seznam. Na řádku 261 začíná cyklus, který pro každou možnou příponu načte tuto příponu (řádek 262) a projde každou z jejích možných předpon (řádek 263). Načte tuto předponu (řádek 264), skombinuje je dohromady (řádky 268 a 271) a přidá do seznamu (řádky 269 a 272). Je zde kladena speciální podmínka (řádek 267) pro značky letadel Spojených států amerických, které nepoužívají po své předponě pomlčku, toto je pak při tvoření značky zohledněno (viz rozdíly řádků 268 a 271). Po přidání všech kombinací list seřadíme podle abecedy (řádky 275) a převrátíme (řádek 277), díky čemuž budeme nejprve detekovat delší regulární výrazy, a až pak ty kratší, a nedojde k záměně. Nakonec seznam uložíme (řádky 279 a 281).

5.4. Shrnutí

V rámci této kapitoly byl kompletně předveden návrh a popsána struktura programu. Ke všem použitým částem byla poskytnuta nutná dokumentace a podrobně vysvětleno jejich fungování. U složitějších funkcí jsou k dispozici diagramy znázorňující jejich fungování.

6. Testování limitů

Tato kapitola se věnuje situacím, ve kterých metoda selhává. Program byl použit na soubor videí o celkové délce 24 minut a 53 vteřin. Videozáznamy byly natočené na medláneckém letišti v Brně za slunečného počasí a v různou denní dobu. Na tomto souboru videí jsou nepřetržité záběry prolétávajících, přistávajících, vzlétajících a pojíždějících letadel. Soubor byl získán jednak osobním natočením, jednak extrahován z youtube kanálu František Brablc [26], kterému tímto děkuji za poskytnutí záběrů. Z výsledků zpracování jsou vyvozeny čtyři případy, ve kterých nejsme schopni značku detekovat, a uvedeny příklady těchto situací.

První a nejčastější důvod špatné detekce je špatná velikost písma. Tesseract vyžaduje písmo o výšce minimálně 20 pixelů, a dobré spolehlivosti dosahuje přibližně do výšky písma 35 pixelů [17], tuto chybu můžeme vidět na obrázku 57, kde není detekovaný vůbec žádný text.

obrázek 57: příklad nepřečtení textu z důvodu malého počtu pixelů na písmeno

Druhý častý důvod je nedostatečně kontrastní imatrikulace. Pokud není písmo od pozadí jednoznačně rozeznatelné, síť Tesseract ji považuje za součást pozadí [15] toto můžeme vidět na obrázku 58.

obrázek 58: neregistrovaná nekontrastní imatrikulace (kamera byla natočena aby se vykompenzoval náklon písma)

Třetí častý důvod je nečitelnost imatrikulace kvůli odleskům na povrchu letadla. Takovéto odlesky mohou udělat imatrikulaci zcela nečitelnou i pro lidské oko, jak vidíme na obrázku 59.

obrázek 59: registrační značka nečitelná kvůli odleskům

Čtvrtý častý důvod je nečitelnost daného typu písma. V rámci testů se ukázalo, že Tesseract není schopen přečíst některá písma, zejména nakloněná písmena a ta psaná kurzívou. Toto můžeme vidět na obrázku 60.

obrázek 60: příliš nakloněné písmo, je přečteno QN-BAN namísto OY-BTY

Shrnutí

V rámci této kapitoly byly testovány limity a omezení aplikace. Hlavním limitem se ukázalo být neschopnost přečíst jakákoliv písmena s výškou nižší než 20 pixelů. Další odhalené limity jsou neschopnost přečíst nekontrastní imatrikulace, nečitelnost imatrikulací kvůli odleskům slunečních paprsků od povrchu letadla, a nečitelnost určitých typů písma, zejména typů s nakloněnými písmeny a kurzívou.

7. Instrukce k použití

V rámci této kapitoly je popsáno všechno, co je třeba ke zprovoznění a obsluze programu. V tomto ohledu předpokládám použití operačního systému Windows. Dále jsou zde popsány rady a tipy jak systém správně nastavit a používat.

7.1. Před použitím

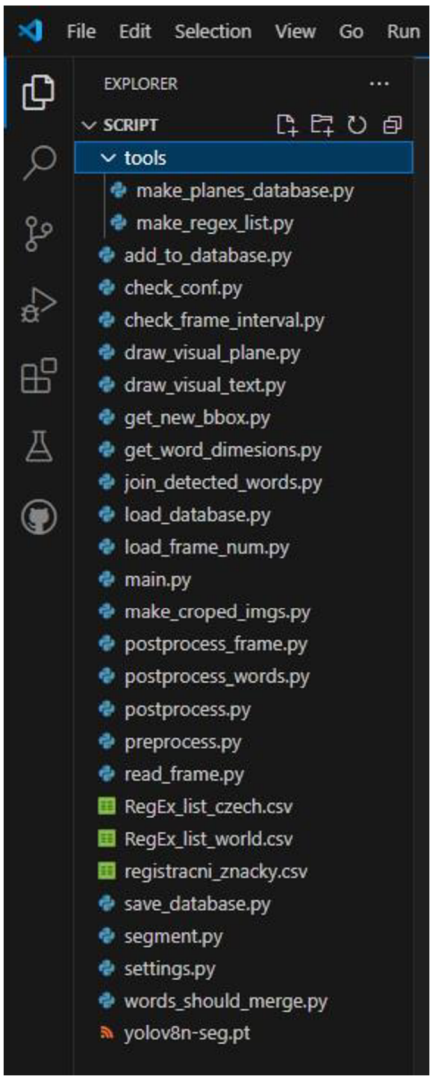
V první řadě bude potřeba nainstalovat programovací prostředí, jako je například Visual Studio Code, které bylo použito při vývoji této aplikace. Instalace tohoto programovacího prostředí je dostupná z internetové stránky společnosti Visual Studio [27]. Dále je nutné nainstalovat Python a připojit ho k tomuto programovacímu prostředí, návod jak na to najdeme opět na stránkách Visual Studio [28]. Jako poslední je třeba se ujistit, že je nainstalován Python PIP, což je prográmek, který Pythonu umožňuje stahovat si různé knihovny. Ten by měl být nativně naistalován v Pythonu pro všechny verze od 3.4 dále [29]. Jak zjistit, zda je nainstalován a jak ho používat najdeme v oficiální dokumentaci programu Python [30].

Teď bychom měli být připravení na nainstalování požadovaných knihoven. Pro program bude nutné nainstalovat knihovny cv2, re, pandas, numpy, bs4, selenium a ultralytics. Dále bude nutné naistalovat a zprovoznit Pytesseract, jak na to se dozvíme v oficiálním repositáři volně dostupných doplňků k Pythonu, který se nazývá pypi [31] [32].

Pro úspěšné použití programu musíme také mít k počítači připojenou kameru, na kterou chceme letadla zachytit. Rady jak tuto kameru umístit, můžete najít v kapitole 7.3.

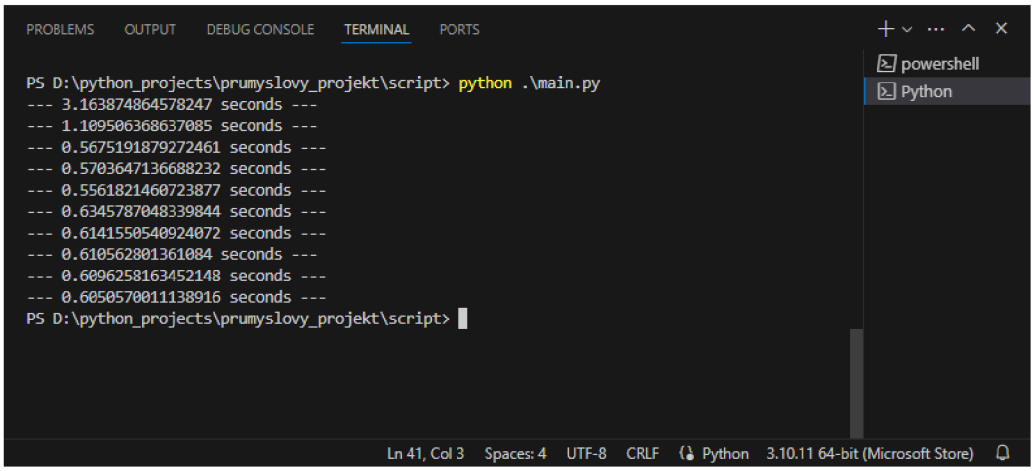
7.2. Spuštění

Abychom mohli program použít, je třeba rozbalit složku s programem, která je připojena v příloze. V této složce najdeme 21 Pythonovských souborů, které jsou potřeba k fungování celého programu, včetně souboru main.py, přes který se celý program spouští. Také jsou zde tři seznamy ve formátu .csv. V seznamu registracni_znacky.csv jsou uložené registrační značky, které byly přepsány z leteckého rejstříku UCL. V souborech RegEx_list_czech.csv a RegEx_list_world.csv jsou seznamy regulérních výrazů pro české a různé světové registrační značky. Dále zde najdeme složku tools. V té jsou uloženy funkce make_planes_database.py a make_regex_list.py, které do programu přímo nevstupují, ale s jejich pomocí se vytvářejí výše zmíněné .csv soubory. Tyto funkce jsou blíže popsány v kapitole 5.3. Jako poslední se zde nachází soubor yolo8n-seg.pt, což je neuronová síť použitá k rozpoznávání letadel. Jak tyto soubory vypadají po otevření ve Visual Studio Code, si můžeme prohlédnout na obrázku 61.



obrázek 61: náhled celého programu

Následně spustíme soubor main.py. Spuštění tohoto souboru i s následným výpisem zpracovacích časů je vidět na obrázku 62, čas zpracování prvního obrázku je vždy trochu delší, protože v první iteraci se inicializují potřebné podsystémy. Pro ukončení přenosu stačí stisknout klávesu q.



obrázek 62: spuštění souboru main.py skrze příkazový řádek

7.3. Použití

V této kapitole budou shrnuty postupy které jsou potřebné pro běžnou obsluhu programu.

Jak umístit kameru

V této podkapitole je nejdříve bodově popsáno jak správně umístit kameru, každý bod je potom dále rozveden. Níže najdeme obrázek 63, na kterém je příklad záběru s komentářem jeho vhodnosti. Kameru je třeba umístit tak aby:

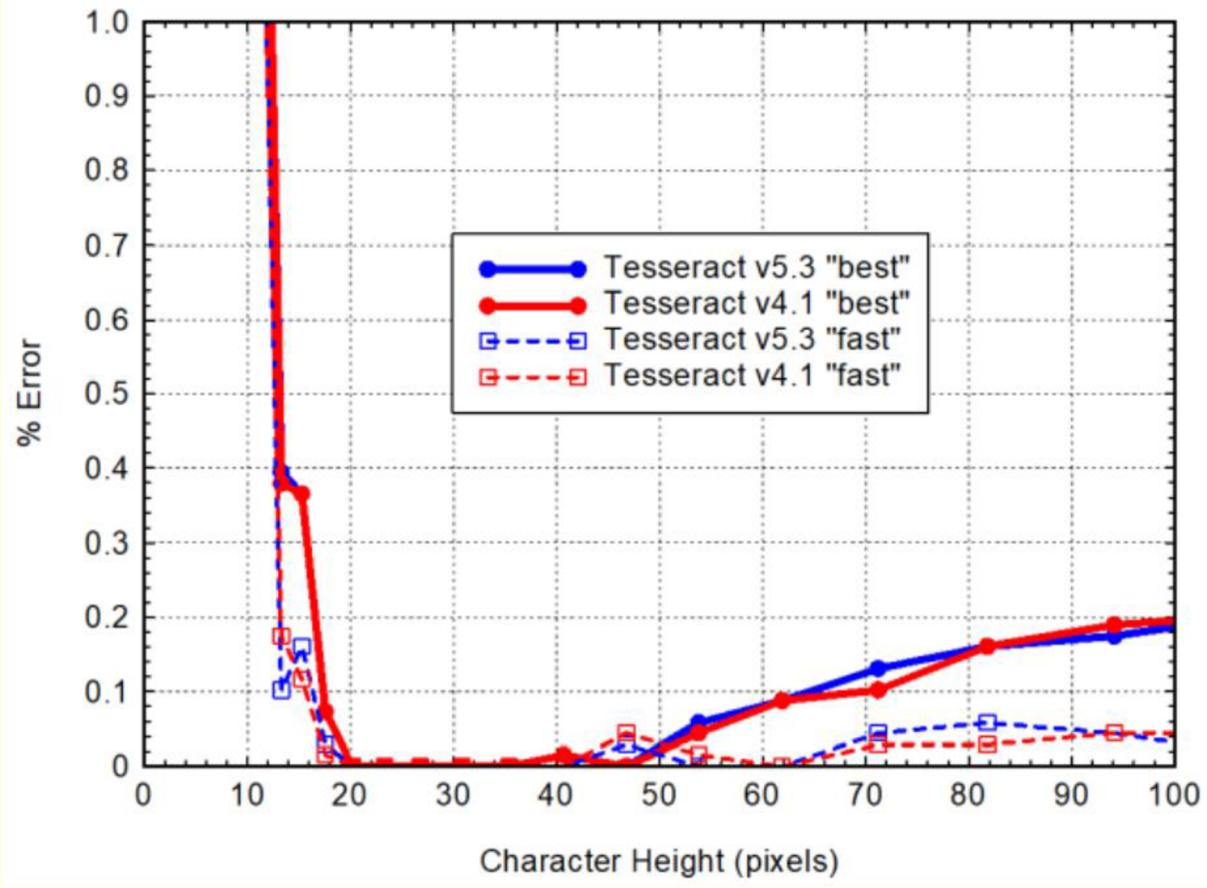
- Bylo letadlo v záběru co nejdéle
- Text měl odpovídající velikost
- Bylo letadlo zabráno ze správného úhlu
- Obraz registrační značky nebyl znehodnocen odlesky slunce od povrchu letadla

obrázek 63: Umístění kamery. Kamera je umístěna na výjezdu z přistávací plochy, což zajišťuje zabrání letadla při nízké rychlosti. Text má velikost 21 pixelů, což znamená že kamera není optimálně umístěna. Pokud by letadlo pojíždělo ve větší vzdálenosti, text už by nebylo možné spolehlivě detekovat. Úhel záběru je sice hraniční, ale letadlo je zabráno na velmi širokém prostoru, což znamená že eventuelně se do správného úhlu dostane, a to i s větším zkosením křídel. Kamera je orientována na jih, díky tomu nemůže dojít k oslnění kamery odlesky od povrchu letadla.

Jelikož letadla mohou využít pro přistání teoreticky celou přistávací plochu, je obtížné zajistit, aby byla při přistání v záběru. Další komplikací je, že při přistání se letadla pohybují relativně rychle. Letadlo by mělo být v záběru co nejdéle, aby bylo nasbíráno co nejvíce jeho snímků, a v rámci kontroly databáze bylo možné lépe poznat chybné detekce od správných. Při psaní této práce se osvědčilo umístit kameru na výjezd z přistávací plochy. Zde přistávající letadla musí projet, pohybují se při tom výrazně pomaleji než při přistání, a navíc je pojezdová dráha typicky užší než dráha přistávací, což napomůže při plnění následujícího bodu.

Letadlo i text mají určitou optimální velikost v pixelech, pro kterou jsou nejjednodušeji detekovatelné. Rozpoznání letadla je zpravidla mnohem snadnější než rozpoznání textu, proto se soustředíme na správnou velikost textu. Optimální velikost

pro rozpoznání je udána jako výška písmene 20 až 35 pixelů [15], viz obrázek 64 [17]. Při umístění kamery bychom tedy měli zvolit takovou vzdálenost, aby měla písmena přibližně v tomto rozmezí, nikdy však méně než 20px, jak z obrázku vyplývá.



obrázek 64: závislost chyby detekce textu na výšce textu v pixelech [17]

Problematika zabrání ze správného úhlu spočívá v zabrání letadla tak, aby jeho křídlo nezakrylo registrační značku. Výsledek takového chybného umístění je na obrázku 65. Tohoto docílíme, když kameru umístíme lehce šikmo po směru pojezdové dráhy. Ovšem ne moc šikmo, protože pro přečtení je žádoucí zabrat text co nejvíce zpříma.

obrázek 65: špatný úhel záběru

V poslední řadě musíme zajistit, že odlesky slunce na povrchu letadla neznemožní detekci, tak jak to můžeme vidět na obrázku 66. K tomuto musíme umístit kameru tak, aby nebylo slunce v zádech kamery, typicky ji tedy směrovat na jih.

obrázek 66: neúspěšná detekce z důvodu odlesků na povrchu vrtulníku.

Výběr vstupu

Pro program je možné jako vstup vybrat živý kamerový přenos nebo starší kamerový záznam. Pokud je k počítači připojeno více kamer, je volitelné, kterou z nich pro program použijeme. K těmto manipulacím slouží argument funkce cv2.videocapture() ve funkci main řádek 41. Pro živý přenos je třeba zadat číslo kamery, kterou chceme použít, pokud máme připojenu jen jednu kameru, je toto číslo 0 (viz obrázek 68), dvě kamery jsou číslovány 0 a 1 atd.. Pro nahrání kamerového záznamu je třeba do argumentu napsat cestu k danému záznamu a jeho název. Příklad použití programu na soubor VIDEO.mp4, který je umístěný ve stejné složce jako soubor main.py, vidíme na obrázku 67.

```
cap = cv2.VideoCapture('VIDEO.mp4')
```

obrázek 67: výběr předem nahraného videa jako vstupu pro zpracování

```
cap = cv2.VideoCapture(0)
```

obrázek 68 výběr živého záznamu kamery jako vstupu pro zpracování

Nastavení

Všechny uživatelem nastavitelné veličiny a funkce je možné nastavit ve funkci settings, tu můžeme vidět na obrázku 70. Na řádku 5 je definováno nastavení sítě Tesseract. Nastavení psm (page segmentation models) volí jakou interní segmentaci obrázku model použije. Nastavení oem (optical character recognition engine mode) určuje jakým způsobem se pokusí síť Tesseract text přečíst [15]. V rámci této práce se nejvíce

osvědčilo nastavení --psm 12 --oem 3. Je ovšem možné, že pro konkrétní případy bude optimální jiné nastavení, a je vhodné, aby každý uživatel vyzkoušel, které bude v jeho případě nejvhodnější. Jak tato nastavení fungují, je napsáno na řádcích 32 až 57, viz obrázek 69.

```
32      """
33      instructions: https://pyimagesearch.com/2021/11/15/tesseract-page-segme
34
35      page segmentation modes (psm in myconfig)
36      0   orientation and script detection only
37      1   automatic page segmentation with OSD
38      2   automatic page segmentation with no OSD or OCR
39      3   fully automatic page segmentation with no OSD (default)
40      4   assume a single column of text of variible sizes
41      5   assume a single uniform block of verticaly aligned text
42      6   assume a single uniform block of text
43      7   treat the image as a single text line
44      8   treat the image as a single text word
45      9   treat the image as a single text word in a circle
46      10   treat the image as a single character
47      11   sparse text, find as much text as possible in no particular order
48      12   sparse text with OSD
49      13   raw line, treat the image as a single text line
50
51      OCR engine mode (oem in myconfig)
52      0   legacy engien only
53      1   neural nets LSTM engine only
54      2   legacy + LSTM engines
55      3   default, based on what is avalible
56      LSTM = long short term memory
57      """
```

obrázek 69: možná nastavení modelu Tesseract

Na řádku 9 je nastaven minimální časový interval mezi zpracováním dvou snímků. Pokud je zadán nižší čas než je potřeba na zpracování snímku, zpracuje se kolik snímků umožní zpracovávací čas. Tuto proměnnou lze použít na ušetření místa na disku, protože se díky ní uloží méně obrázků. Řádek 13 určuje, s jakou jistotou musíme sítí YOLO detekovat letadlo, aby bylo uznáno za úspěšně detekované. V rámci této práce se osvědčila hodnota 0.7, skutečná letadla byla detekována se zpravidla vyšší jistotou, a zároveň je taková hodnota dostatečně vysoká na to, aby odstínila většinu chybně detekovaných objektů. Na řádku 17 je určena jistota, se kterou bude text detekovaný sítí Tesseract považovaný za validní. Tato hodnota byla v rámci testů vybrána relativně nízká, počítá se totiž s validací čtených textů pomocí zařazení do systému registračních značek. Řádek 20 dává uživateli na výběr, zda chce okolo detekovaného textu nechat vykreslit zelené obdélníky s detekovaným textem. Tato funkce může být vypnuta například pokud se uživatel rozhodne, že nechce detekci textu obecně používat, a bude fotky s letadly procházet ručně, aby pro něj vykreslené texty nebyly rušivé. Podobné nastavení je na řádku 23, kde si je možné ze stejného

důvodu vypnout vykreslování obdélníků okolo letadel. Řádek 26 dává možnost vypsat, jak dlouho trvalo zpracování každého snímku (pozor, ne jaký byl interval mezi snímky). V poslední řadě se na řádku 30 nachází nastavení, které umožňuje uložení snímků, na kterých bylo detekováno letadlo, ale nebyla u nich detekována jeho značka.

```
1   def settings():
2
3       # configuration of tesseract CNN, for details
4       # see lines 28 to 51
5       tess_config = r"--psm 12 --oem 3"
6
7       # sets minimal interval between processed frames,
8       # if set too low (like 0.01) will process every frame
9       min_sec_interval = 0.01
10
11      # define confidence of detection at witch plane
12      # is considered to be detected
13      desired_conf_plane = 0.7
14
15      # define confidence of detection at witch text
16      # is considered to be detected
17      desired_conf_text = 0.5
18
19      # set true if you want to draw boxes around detected text
20      draw_boxes_text = True
21
22      # set true if you want to draw boxes around detected planes
23      draw_boxes_planes = True
24
25      # set true if you want to print processing time of each frame
26      print_proseccing_time = True
27
28      # set true if you want to save frames with detected frames
29      # but undetected registration numbers as "unknown"
30      save_unread_img = True
31
```

obrázek 70: funkce settings

Jak kontrolovat výsledky

Po ukončení programu (typicky skončením nahraného videa nebo stiskem klávesy q), je vhodné výsledky nějakým způsobem kontrolovat. Metoda předpokládá že pokud je jedna značka detekována dostatečně mnohokrát, je detekována správně. Kolik množství detekcí je nutných, aby bylo jisté, že jsou správné, to závisí na každé jednotlivé implementaci systému. Cíl takové kontroly je proto dvojí. Zaprvé bude třeba získat dostatečné praktické zkušenosti, které nám prozradí, kolik detekcí je dostatečných. Druhé na řadě je potom využití těchto zkušeností k samotné kontrole.

Kontrola by měla probíhat přečtením záznamů v databázi database.json z daného dne a projitím uložených obrázků k nim. Je zapotřebí vysledovat určitý počet detekcí, od kterého je letadlo i se značkou detekováno vždy správně. Pokud máme tento počet, je možné procházet pouze značky o nízkém počtu detekcí.

Aktualizace seznamu registračních značek UCL

Pro aktualizaci seznamu registračních značek UCL je třeba použít soubor make_planes_database.py, který se nachází ve složce tools. Stačí soubor spustit, počkat na jeho ukončení, toto může trvat pár desítek minut až hodinu. Po dokončení se vytvoří nový soubor s názvem registracni_znacky.csv, kterým je třeba přepsat ten starý.

Je nutné zdůraznit, že tento program je závislý na podobě stránek UCL, a pokud budou v budoucnu změněny, už nebude fungovat!

Změna šablony registrační značky jiného státu

Registrační značky jiných států jsou uloženy v souboru RegEx_list_world.csv, který se tvoří souborem make_regex_list.py, ten se nachází ve složce tools. Pokud chceme umožnit rozpoznání dalších formátů, nebo naopak některé formáty nerozpoznávat, musíme přepsat soubor make_regex_list.py, a seznam vygenerovat znovu.

Stačí v první části programu (řádek 5 až 256) najít formát, který chceme přidat, a připsat k němu předponu státu. Alternativně najít předpony státu, který chceme odebrat, a odstranit je od patřičných přípon. Přípony jsou seřazeny podle počtu čísel, které se v nich nacházejí, nejdříve jsou přípony bez čísel, pak s jedním číslem a tak dále.

Po přepsání seznamu značek dle libosti uživatele je třeba soubor spustit, vygeneruje se nový soubor RegEx_list_world.csv, a tím přepíšeme starý.

7.4. Shrnutí

V rámci této kapitoly byly poskytnuty doporučení a instrukce k použití. Bylo popsáno, jak program spustit, jaké jsou k tomu softwarové požadavky a jaké jsou požadavky na kameru použitou pro získávání záznamu. Bylo objasněno, jaká je zamýšlená metodika použití.

8. Závěr

Cílem této diplomové práce bylo vytvořit metodu, kterou lze automatizovat zápis letadel přistávajících na neřízených letištích. K tomuto účelu byl vytvořen program v prostředí Python, který s pomocí neuronových sítí hledá v kamerovém záznamu letadla a jejich registrační značky.

V rámci rešerše byly představeny možné přístupy k sledování provozu na letištích. Následně bylo hlouběji prozkoumáno počítačové vidění jako perspektivní metoda pro řešení tohoto problému. V závěru rešeršní části byly vybrány neuronové sítě YOLOv8 a Tesseract, které byly v praktické části použity k detekci letadel a textu. Síť Tesseract byla vybrána z důvodu její bezkonkurenční rychlosti a jednoduchosti použití. Síť YOLOv8 byla vybrána z kvůli její unikátní kombinaci rychlosti a vysoké přesnosti.

V praktické části diplomové práce byla nejdříve navrhnuta základní koncepce řešení problému. Ta byla následně implementována, podrobně popsána a testována. V rámci testů byly odhaleny konkrétní případy, ve kterých metoda selhává. Program není schopen přečíst písmena o výšce nižší než 20 pixelů. Pouze s velkými obtížemi je schopen přečíst registrační značky které nejsou kontrastní vůči pozadí. Dalším limitem je neschopnost přečíst určité typy písma, zejména nakloněné písmo a kurzívu. V poslední řadě je potom problematické přečíst značky, pokud je kamera oslněna odlesky od povrchu letadla. V závěru práce je sepsána příručka k použití, která provede uživatele uvedením metody do provozu, možnými nastaveními a všemi záležitostmi, které se týkají každodenního používání.

9. Seznam použitích zdrojů

[1] WOLFF, Christian. Radar pro řízení pohybů na letištní ploše — Radar Basics. In: WOLFF, Christian. *Základy radaru — Radar Basics* [online]. 1998 [cit. 2024-03-17]. Dostupné z: https://www.radartutorial.eu/02.basics/rp17.cz.html

[2] ADSB-EXCHANGE. Enterprise API - ADS-B Exchange. In: ADSB-EXCHANGE. *Serving the Flight Tracking Enthusiast - ADS* - [online]. 2024 [cit. 2024-05-23]. Dostupné z: https://www.adsbexchange.com/products/enterprise-api/

[3] ADSB-EXCHANGE. Historical Data - ADS-B Exchange. In: ADSB-EXCHANGE. *Serving the Flight Tracking Enthusiast - ADS* - [online]. 2024 [cit. 2024-05-23]. Dostupné z: https://www.adsbexchange.com/products/historical-data/

[4] ADS-B EXCHANGE. ADS-B Exchange - track aircraft live. In: *Serving the Flight Tracking Enthusiast - ADS* - [online]. 2024 [cit. 2024-05-23]. Dostupné z: https://globe.adsbexchange.com

[5] FLIGHTRADAR24. Flightradar24: Live Flight Tracker - Real-Time Flight Tracker Map. In: FLIGHTRADAR24. *Flightradar24: Live Flight Tracker - Real-Time Flight Tracker Map* [online]. 2024 [cit. 2024-03-17]. Dostupné z: https://www.flightradar24.com/premium

[6] ADS-B EXCHANGE. Subscriptions - ADSBexchange.com. In: ADS-B EXCHANGE. *ADSBexchange.com store* [online]. 2024 [cit. 2024-03-17]. Dostupné z: https://store.adsbexchange.com/collections/subscriptions

[7] ŠONKA, Milan, Václav HLAVÁČ a Roger BOYLE. *Image processing, analysis, and machine vision*. 4th edition. United States of America: Cengage Learning, 2015, xxxv, 870 stran : ilustrace, grafy ; 23 cm. ISBN 978-1-133-59369-0.

[8] HGH. Airport Surveillance I HGH Infrared Systems. In: HGH. *HGH I Infrared Systems* [online]. 2024 [cit. 2024-03-17]. Dostupné z: https://hgh-infrared.com/en/airport-surveillance-airfield-surveillance/?gad_source=1&gclid=Cj0KCQiAw6yuBhDrARIsACf94RVKwr5ZjF-b32qtWLIMoQWe-qwuXz-AnBTkJBA41NKmVzIlRjQ7ISwaAqUOEALw_wcB

[9] JANÁKOVÁ, Ilona. Microsoft PowerPoint - 01_Uvod a Motivace. In: MACHINE VISION GROUP. *Machine Vision Group* [online]. 2023 [cit. 2024-03-17]. Dostupné z: http://vision.uamt.feec.vutbr.cz/POV/lectures/01_Uvod%20a%20motivace.pdf

[10] NAGAO, Makoto a Takashi MATSUYAMA. *A Structural Analysis of Complex Aerial Photographs: Advanced applicationsin pattern recognition*. III. Series. New York: Plenum Press, 1980. ISBN 0-306-40571-7.

[11] HøYE, Toke, Johanna ÄRJE, Kim BJERGE et al. Deep learning and computer vision will transform entomology. *Proceedings of the National Academy of Sciences* [online]. 2021, **118**(2) [cit. 2021-05-13]. ISSN 0027-8424. Dostupné z: doi:10.1073/pnas.2002545117

[12] YAN, Xinchen, Jimei YANG, Ersin YUMER, Yijie GUO a Honglak LEE. *Perspective Transformer Nets: Learning Single-View 3D Object Reconstruction without 3D Supervision: Learning Single-View 3D Object Reconstruction without 3D Supervision*. 2016.

[13] RAYSPER, Yuri. File:Axonometric projection.svg. In: *Wikimedia Commons: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2024-05-23]. Dostupné z: https://commons.wikimedia.org/wiki/File:Axonometric_projection.svg

[14] ANAND, Sheila a Priya LOGANATHAN. *A Guide for Machine Vision in Quality Control*. 2019. ISBN 9781003002826. Dostupné z: doi:10.1201/9781003002826

[15] GITHUB, INC. Tesseract-ocr · GitHub. In: GITHUB, INC. *GitHub: Let's build from here · GitHub* [online]. 2024 [cit. 2024-05-23]. Dostupné z: https://github.com/tesseract-ocr

[16] ULTRALYTICS, INC. *Home - ultralytics YOLOv8 Docs* [online]. 2024 [cit. 2024-05-23]. Dostupné z: https://docs.ultralytics.com

[17] Willus.com's Blog. In: *Willus.com Home Page* [online]. 1998, 25-Dec-2023 [cit. 2024-05-23]. Dostupné z: https://willus.com/blog.shtml?tesseract_accuracy

[18] REUL, Christian, Dennis CHRIST, Alexander HARTELT et al. OCR4all—An Open-Source Tool Providing a (Semi-)Automatic OCR Workflow for Historical Printings. *Applied Sciences* [online]. 2019, **9**(22) [cit. 2024-05-23]. ISSN 2076-3417. Dostupné z: doi:10.3390/app9224853

[19] BATRA, Pulkit, Nimish PHALNIKAR, Deepesh KURMI, Jitendra TEMBHURNE, Parul SAHARE a Tausif DIWAN. *OCR-MRD: Performance Analysis of Different Optical Character Recognition Engines for Medical Report Digitization: Performance Analysis of Different Optical Character Recognition Engines for Medical Report Digitization*. 2023. Dostupné z: doi:10.21203/rs.3.rs-2513255/v1

[20] HUKKERI, Geeta, R GOUDAR, Prashant JANAGOND a Pooja PATIL. Machine Learning in OCR Technology: Performance Analysis of Different OCR Methods for Slide-to-Text Conversion in Lecture Videos: Performance Analysis of Different OCR Methods for Slide-to-Text Conversion in Lecture Videos. *International Journal of Advanced Computer Science and Applications*. 2022, **13**. Dostupné z: doi:10.14569/IJACSA.2022.0130839

[21] RAMDHANI, Tosan, Indra BUDI a Betty PURWANDARI. Optical Character Recognition Engines Performance Comparison in Information Extraction. *International Journal of Advanced Computer Science and*

Applications. 2021, **12**, 2021. Dostupné z: doi:10.14569/IJACSA.2021.0120814

[22] JOHN, Anand a Dr. MEVA. A Comparative Study of Various Object Detection Algorithms and Performance Analysis. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING*. 2020, **8**, 158-163. Dostupné z: doi:10.26438/ijcse/v8i10.158163

[23] PATEL, Hrishitva. *A Comprehensive Study on Object Detection Techniques in Unconstrained Environments*. 2023.

[24] RAHAMAN, Md. *The Current Trends of Object Detection Algorithms: A Review: A Review*. 2023. Dostupné z: doi:10.13140/RG.2.2.19067.49442

[25] STACK EXCHANGE INC. Python - what's the mean of "reshape(-1,1,2)" - Stack Overflow. In: STACK EXCHANGE INC. *Stack Overflow - Where Developers Learn, Share, & Build Careers* [online]. 2009 [cit. 2024-05-23]. Dostupné z: https://stackoverflow.com/questions/49788723/whats-the-mean-of-reshape-1-1-2

[26] BRABLC, František. Letiště Medlánky - YouTube. In: YOUTUBE. *YouTube* [online]. 2012-2024 [cit. 2024-05-23]. Dostupné z: https://www.youtube.com/playlist?list=PLjpmUWN6gS9FuO5TIKFjg7m07XMzt1Cp7

[27] MICROSOFT CORPORATION. Download Visual Studio Code - Mac, Linux, Windows. In: MICROSOFT CORPORATION. *Visual Studio Code - Code Editing. Redefined* [online]. 2015 [cit. 2024-05-24]. Dostupné z: https://code.visualstudio.com/download

[28] MICROSOFT CORPORATION. Get Started Tutorial for Python in Visual Studio. In: MICROSOFT CORPORATION. *Visual Studio Code - Code Editing. Redefined* [online]. 2015 [cit. 2024-05-24]. Dostupné z: https://code.visualstudio.com/docs/python/python-tutorial

[29] W3SCHOOLS. Python PIP. In: W3SCHOOLS. *W3Schools Online Web Tutorials* [online]. 1999 [cit. 2024-05-24]. Dostupné z: https://www.w3schools.com/python/python_pip.asp

[30] PYTHON PACKAGING AUTHORITY. Installing Packages - Python Packaging User Guide. In: *Welcome to Python.org* [online]. 2001 [cit. 2024-05-24]. Dostupné z: https://packaging.python.org/en/latest/tutorials/installing-packages/

[31] PYTHON SOFTWARE FOUNDATION. Pytesseract · PyPI. In: PYTHON SOFTWARE FOUNDATION. *PyPI · The Python Package Index* [online]. 2003 [cit. 2024-05-24]. Dostupné z: https://pypi.org/project/pytesseract/

[32] YOUTUBE. Extract Text From Images in Python (OCR) - YouTube. In: YOUTUBE. *YouTube* [online]. 2005 [cit. 2024-05-24]. Dostupné z: https://www.youtube.com/watch?v=PY_N1XdFp4w&ab_channel=NeuralNine

10. Seznam příloh

PŘÍLOHA 1 složka script