

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor: 18. Informatika

Coopmaster - systém pro řízení a automatizaci kurníku

Jaroslav Němec

Hradec Králové 2025

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

**COOPMASTER - SYSTÉM PRO ŘÍZENÍ A
AUTOMATIZACI KURNÍKU**

**COOPMASTER - SYSTEM FOR COOP CONTROL
AND AUTOMATION**

AUTOR	Jaroslav Němec
ŠKOLA	Střední škola a vyšší odborná škola aplikované kybernetiky s.r.o.
KRAJ	Královéhradecký
ŠKOLITEL	Ing. David Podzimek
OBOR	18. Informatika

Hradec Králové 2025

Prohlášení

Prohlašuji, že svou práci na téma *Coopmaster - systém pro řízení a automatizaci kurníku* jsem vypracoval samostatně pod vedením pana Ing. Davida Podzimka a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Dále prohlašuji, že tištěná i elektronická verze práce SOČ jsou shodné a nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a změně některých zákonů (autorský zákon) v platném znění.

V Hradci Králové dne: _____

Jaroslav Němec

Poděkování

Rád bych vyjádřil svou hlubokou vděčnost svému školiteli, panu Ing. Davidu Podzimkovi. Jeho neúnavná podpora, cenné rady a mimořádná trpělivost pro mě byly neocenitelné během celé mé práce.

Za prvé jeho obětavá pomoc mi umožnila překonat řadu výzev, se kterými jsem se během studia setkal. Jeho schopnost najít efektivní řešení i pro komplikované problémy mi ukázala nové způsoby myšlení a pomohla mi posouvat se dál.

Za druhé jeho podnětné připomínky mě vždy motivovaly k tomu, abych zkoumal danou problematiku hlouběji a viděl věci z různých perspektiv. Díky jeho vedení jsem se naučil kriticky hodnotit své vlastní myšlenky a přístupy.

A konečně jeho trpělivost mi poskytla pocit jistoty a povzbuzení. I když jsem narazil na obtíže nebo neúspěchy, jeho klidné a podporující vedení mi pomohlo udržet si pozitivní postoj a pokračovat v práci.

Za všechny tyto věci jsem mu nesmírně vděčný a jeho příspěvek k mému vývoji si budu pamatovat po celý život.

Anotace

Tato práce se zaměřuje na návrh a realizaci pokročilého asistenčního systému Coopmaster pro automatizaci a zefektivnění chovu hospodářských zvířat, konkrétně kura domácího. Využívá moderních technologií, včetně zpracování obrazu pomocí neuronových sítí, mikroservisní architektury a open source softwaru Home Assistant. Díky nim zajišťuje vyšší úroveň bezpečnosti a minimalizuje nutnost fyzické přítomnosti chovatele. Propojení hardwaru a softwaru tvoří komplexní řešení od vývoje funkčních dvířek a digitálních vah až po intuitivní mobilní i webovou aplikaci s podporou vzdáleného přístupu. Systém v reálném čase monitoruje stav a aktivitu slepic, kontroluje hnízda a odhaluje případné vetřelce ve výběhu. Díky neustálému monitorování hmotnosti hnízd systém poskytuje chovateli aktuální přehled o snůšce vajec v kurníku. Součástí řešení je také rozsáhlý monitoring a tvorba statistik, které uživatelům poskytují přehled o dlouhodobých trendech, produkci vajec či stavu hejna. Projekt je nasazen a používán v reálném provozu. Přináší chovatelům vyšší komfort a zlepšuje blahobyt zvířat. Přestože je v některých oblastech prostor pro další optimalizace a zlepšení, představuje tento projekt významný krok vpřed ve vývoji moderních řídicích a automatizačních systémů pro chov hospodářských zvířat.

Klíčová slova

Programování, AI automatizace a zabezpečení, Mikroservisní architektura, Home Assistant, Vzdálený přístup, Chov hospodářských zvířat, Vlastní hardware

Annotation

This thesis focuses on the design and implementation of an advanced Coop-master assistance system to automate and streamline the management of livestock, specifically domestic chickens. It uses advanced technologies such as neural network image processing, microservice architecture and open source home assistant software. These provide a higher level of safety and minimise the need for the farmer's physical presence. The combination of hardware and software provides a comprehensive solution, from the development of functional doors and digital scales to an intuitive mobile and web application with remote access support. The system monitors the status and activity of the hens in real time, checks the nests and detects potential intruders in the run. By constantly monitoring the weight of the nests, the system provides the farmer with an up-to-date overview of the eggs laid in the coop. The solution also includes extensive monitoring and statistics generation to provide users with an overview of long-term trends, egg production or flock status. The project is being implemented and used in real operations. It is bringing more convenience to breeders and improving animal welfare. Although there is room for further optimisation and improvement in some areas, this project represents a significant step forward in the development of modern livestock management and automation systems.

Keywords

Programming, AI Automation & Security, Microservice Architecture, Home Assistant, Remote Access, Farm animals, Custom Hardware

Obsah

1	Úvod	10
2	Zadání projektu Coopmaster	12
3	Návrh architektury a volba technologií	14
3.1	Služby dle jednotlivých potřeb uživatele	15
3.2	Systém pro uživatelské rozhraní a automatizaci	16
3.3	Komunikace mezi moduly	17
3.4	Hardwarová zařízení	18
3.4.1	Snímání stavů hnízd	18
3.4.2	Ovládání dvířek, snímání teploty a vlhkosti	19
3.4.3	Snímání scén v kurníku a ve výběhu	19
3.5	Spolupráce modulů dle požadavků uživatele	20
3.5.1	Ovládání světla, data o teplotě a vlhkosti	20
3.5.2	Automatizace dvířek	21
3.5.3	Detekce vetřelců	22
3.5.4	Vizualizace stavů hnízd	23
3.6	Zpřístupnění aplikace z internetu	24
4	Systémové moduly	26
4.1	Camera Driver	26
4.2	Scale Driver	28
4.3	Room Driver	30

4.4	Health Checker	32
4.5	Room Assistant	34
4.6	Nest Watcher	36
4.7	Dog alarm	40
4.8	Chicken Watch Guard	43
4.9	Uživatelská aplikace	45
4.9.1	Požadavky na systém	45
4.9.2	Postup konfigurace systému Home Assistant	48
4.10	Digitální váha do hnízda	53
4.11	Řídící jednotka	57
4.12	Kamerový systém	60
4.13	Konstrukce dvířek	62
5	Učení Yolo modelů	65
5.1	Detekce objektů pomocí strojového učení	65
5.2	Segmentace objektů pomocí strojového učení	77
6	Tvorba programu pro Arduino	81
7	Testování systému Coopmaster	83
7.1	Integrační testy	83
7.2	Systémové a zátěžové testy	83
7.3	Uživatelské testování	84
8	Produkce	85
9	Ekonomická stránka projektu	91
10	Rozvržení práce do budoucna	93
10.1	Koncept identifikace nejlepší nosnice	93
10.2	Vylepšení konstrukce dvířek	99
10.3	Vylepšení funkcionality služby Room Assistant	101

11 Teoretické pojmy	102
11.1 Docker Engine	102
11.2 Docker Compose	102
11.3 Mikroservisní architektura	102
11.4 Jazyk Python	103
11.5 Wiring	103
11.6 C++	103
11.7 Táhlový motor	104
11.8 Github workflow	104
11.9 MQTT	104
11.10 Home Assistant	105
11.11 Arduino	105
11.12 IP kamera a RTSP	105
11.13 Flask	106
11.14 Yolo Ultralytics	106
11.15 Tenzometrický senzor	106
11.16 POE	107
11.17 Wifi extender	107
11.18 TailScale VPN	108
11.19 Postgres 15	108
11.20 Eclipse Mosquitto	108
11.21 Cloudflared	109
11.22 COCO formát	109
12 Závěr	110
Seznam zkratk	112
Použitá literatura	119
Seznam obrázků	121

1 Úvod

V dnešní době je populární využívat moderní technologie se zapojením umělé inteligence i v oblastech, kde by to člověk nečekal. Vyskytuje se všude kolem nás, objevuje se v různých aplikacích běžného života. Od rozpoznávání registračních značek automobilů, monitoringu pohybu zákazníků v obchodech nebo ve formě textových modelů, které fungují jako různí pomocníci nebo chatboti. Zároveň dnes ubývá lidí, kteří by chtěli trávit svůj čas chovem hospodářských zvířat, a proto je potřeba, aby něco zaujalo jejich místo. Nejsou, ale všichni takoví. Třeba já jsem člověk, kterému na zvířátkách záleží a chce, aby se měla dobře. Zároveň jsem i programátor, který rád zkoumá nové věci v oblasti informatiky a velice ho baví automatizování nejrůznějších úloh.

Všechna tato fakta mi pomohla rozhodnout jaký projekt chci realizovat. Pomohla vytvořit práci popisující využití moderních technologií při chovu hospodářských zvířat. Nápad na tuto práci vznikl díky mojí babičce, která chová doma slepice. Když odjede na dovolenou, stávám se já tím, kdo se o ně musí starat. Je třeba ráno i večer dojet slepice zkontrolovat a spočítat. Tato činnost je časově náročná kvůli cestování a kolikrát i zbytečná, protože většinou se nic neděje. A vzhledem k tomu, že jsem povoláním informatik, tuto práci věnuji tomu, jak jsem vyřešil automatizaci babiččina chovu kura domácího pomocí vlastního systému Coopmaster.

Moje práce se dělí na teoretickou a praktickou část. V teoretické části jsou popsány základní pojmy a principy, jež jsou důležité pro plné chápání práce.

V praktické části popisuji konkrétní implementaci a nasazení svého asistenčního systému pro chov hospodářských zvířat. Tato část zároveň může sloužit k motivaci a inspiraci čtenáře k tvorbě vlastního řešení. Čtenář využije znalosti, které nasbírám během čtení teoretické části, aby mu následně pomohly chápat praktickou část.

Řešení jsem realizoval pomocí mikroservisní architektury a jednotlivé služby jsou psány v jazyce Python. Jako grafické uživatelské rozhraní jsem použil open source software pro chytrou domácnost Home Assistant.

2 Zadání projektu Coopmaster

Moje dosavadní práce na školních projektech a zkušenosti s vývojem aplikací mě naučili, že před tím než se započne tvorba jakékoli aplikace, je třeba udělat důkladnou analýzu.

Je důležité mít představu, co má aplikace umět, jaké jsou na ni kladeny požadavky a také je dobré vědět, kdo bude uživatel.

Uživatel lépe řečeno dobře nadefinovaná uživatelská role nám pomáhá pochopit jaké má cíle a motivace. Ať už je to konkrétní člověk nebo fiktivní postava, vcítění se do jejího pohledu na věc je cenným zdrojem zpětné vazby.

Mým zákazníkem a uživatelem je moje babička. Je to člověk milující zvířátka, nepřítel stolního počítače, ale již několik let online na mobilním telefonu. Můžeme tedy říci, že v našem případě, děláme systém pro člověka znalého chovu hospodářských zvířat, ale méně zdatného v používání informačních technologií.

Teď se pojďme zamyslet jakou funkcionalitu je třeba do řešení implementovat. Postupem let, kdy babičce se slepičkami pomáhám, se mi v hlavě začal utvářet seznam. Její největší starost krom krmení je otevírání a zavírání dvířek, svícení v kurníku, sběr vajíček a starost o kuny, které chodí na vejce. Dále má pak strach ze psů, kteří se jí tam v minulosti dostali. Řeší počítání slepiček, aby zjistila jestli jsou všechny doma. Má problém i se slepicemi, které zanáší kdekoliv ve výběhu. A v neposlední řadě mívá i takový pocit, jako když člověk odejde z domu a neví jestli zamkl. V jejím případě by ráda věděla, jestli je všechno v pořádku.

V jedné větě bych mohl říct, že je třeba, aby aplikace umožňovala uživateli vzdálený dohled nad jeho chovem a pomáhala mu šetřit čas zautomatizováním každodenních činností.

Vytvořil jsem si podle zásad agilního vývoje [1] prioritizovaný seznam požadavků. Tento seznam jsem setřídil podle priorit, náročnosti a také svých možností a schopností. Na vrchol tohoto seznamu se dostaly úkoly, které pokrývají následující oblasti a poskytnou uživateli následující funkce

- Vzdálený přístup
- Pohledy z bezpečnostních kamer
- Automatizované ovládání světla a dvířek v kurníku
- Intuitivní vizualizaci stavů jednotlivých hnízd (sedící slepice, snesené vajíčko)
- Upozornění na detekování vetřelce ve výběhu
- Data o teplotě a vlhkosti

3 Návrh architektury a volba technologií

V předchozí kapitole jsme si pověděli o analýze požadavků na systém. Určili jsme si, kdo je uživatelem naší aplikace, co od naší aplikace očekává. V tomto bodě jsme schopni teoreticky navrhnout architekturu našeho systému a přesně popsat požadovanou funkcionalitu, jakou budou disponovat jeho jednotlivé části. Tato sekce se bude zabývat detailnějším rozbořem konkrétních požadavků a návrhem architektury systému Coopmaster.

3.1 Služby dle jednotlivých potřeb uživatele

Jako teoretický model na základě kterého budeme organizovat a třídit funkcionalitu, jsem zvolil mikroservisní architekturu (sekce 11.3). Tento způsob rozvržení zodpovědnosti částí systému jsem zvolil díky velké možnosti rozšíření, zapouzdření funkcionality a snadné úpravě jednotlivých služeb bez nutnosti kompletního restartu systému v případě znovunasazení.

Jako programovací jazyk pro tvorbu služeb je zvolen programovací jazyk Python (sekce 11.4). Python byl vybrán kvůli jeho univerzálnosti, snadné syntaxi a podpoře ze strany vývojářů a komunity.

Pro realizaci, konfiguraci a síťování mikroservisní architektury je použit Docker Engine společně s rozšířením Docker Compose.

Na základě předchozích rozhodnutí jsem tedy systém rozdělil do jednotlivých mikroslužeb tak, aby vznikla robustní a rozšiřitelná mikroservisní architektura a byly splněny všechny uživatelské požadavky, které jsme si určili (2).

Bylo tedy navrženo 8 služeb, které obsahují námi naprogramovanou funkcionalitu a dohromady společně s hardwarovými prvky tvoří systém Coopmaster.

- Camera Driver (4.1)
- Scale Driver (4.2)
- Room Driver (4.3)
- Health Checker (4.4)
- Room Assistant (4.5)
- Nest Watcher (4.6)
- Dog Alarm (4.7)
- Chicken Watch Guard (4.8)

3.2 Systém pro uživatelské rozhraní a automatizaci

Podstatnou otázkou bylo, zda se pouštět do tvorby kompletně vlastního systému pro uživatelské rozhraní a automatizaci. Po zvážení časové náročnosti, technologické náročnosti a komplexnosti takového systému, mi bylo jasné, že tudy cesta nepovede. Bylo tedy třeba najít open source systém, který již touto funkcionalitou disponuje. Nakonec jsem zvolil systém Home Assistant, protože jsem s ním již dříve pracoval a mám s ním dobré zkušenosti [2]. Home Assistant má na internetu také rozsáhlou podporu. Má bohaté diskuzní forum a širokou komunitu. Zároveň je možné nahlédnout do již hotových řešení pro inspiraci. Jak je nakonfigurováno uživatelské rozhraní v Home Assistantu, je popsáno v části 4.9.

3.3 Komunikace mezi moduly

Je třeba zajistit komunikaci mezi systémem Home Assistant a Coopmaster službami. Tuto úlohu musíme přijmout velice zodpovědně a navrhnout řešení, které půjde opět snadno rozšířit a modifikovat. Z toho důvodu nelze použít klasický HTTP, protože bychom komunikaci vázali na doménové jméno a port nebo ip adresu a port. Toto řešení má problém v tom, že pokud bychom potřebovali změnit, buď umístění částí služeb nebo port jedné ze služeb, bylo by třeba překonfigurovat i Home Assistanta. Další problém nastane, když potřebujeme z některé ze služeb poslat například notifikaci do Home Assistanta, je pro to potřeba, aby jednotlivé služby věděly, kde na síti Home Assistant běží, což je věc, která se může měnit, a museli bychom naopak přenastavovat jednotlivé služby. Jako řešení se nabízí použít messaging konkrétně třeba technologii MQTT (část 11.9). Tato technologie se běžně používá u IoT zařízení a pro naše použití je vynikajícím řešením.

3.4 Hardwarová zařízení

Tuto sekci je velmi důležité dobře a správně navrhnout, protože náš systém bude pracovat s daty z reálného světa. Tato data musejí získávat k tomu určená zařízení a na jejich přesnosti závisí efektivita a správnost chování zbytku systému. Musíme tedy navrhnout a realizovat několik zařízení, která společně budou plnit následující úlohy

- Snímání stavů hnízd
- Ovládání dvířek
- Snímání teploty
- Snímání vlhkosti
- Snímání interiéru kurníku
- Snímání výběhu

3.4.1 Snímání stavů hnízd

Všechny případy, které potřebujeme řešit v kontextu našeho hnízda, se dají rozpoznat na základě hmotnosti hnízda. Pokud je v hnízdě slepice, hnízdo bude hodně zatížené oproti počátečnímu prázdnému stavu. V případě, že bude hnízdo zatížené málo a nebo středně, může to pro nás ve většině případů spolehlivě znamenat, že se v hnízdě nacházejí vejce. Nestandardní situace jako neobvyklý nepořádek v hnízdě, případně výkal, řešit nebudeme, protože by to exponenciálně zvýšilo složitost řešení a nepřineslo by to o moc lepší data. Kurník je navíc v našem případě pravidelně udržován, takže šance na výskyt chybového faktoru v podobě nežádoucí zátěže je nízká. Jako nej-jednodušší prostředek pro splnění tohoto úkolu na základě předchozí analýzy se ukázala obyčejná digitální váha. Bude třeba vytvořit vlastní konstrukci vzhledem ke skutečnosti, že ji musíme zabudovat do již existujícího hnízda. Popis konkrétního zapojení a konstrukce je sepsán níže v sekci 4.10.

3.4.2 Ovládání dvířek, snímání teploty a vlhkosti

Pro ovládání dvířek a snímání povětrnostních podmínek je nejrozumnější vytvořit řídicí jednotku, která tyto požadavky obsáhne. Další částí jsou samotná elektrická dvířka. Na základě elektrického signálu budou otevřená nebo zavřená. Teplota a vlhkost jsou snímány senzory, jež jsou součástí řídicí jednotky. Řídicí jednotka je popsána v sekci 4.11 a konstrukce dvířek v sekci 4.13.

3.4.3 Snímání scén v kurníku a ve výběhu

Ke snímání scén v daných oblastech je třeba vybrat zařízení s dostatečnou odolností proti povětrnostním vlivům. Speciálně v kurníku je hodně agresivní prostředí kvůli vysoké vlhkosti. Pro dlouhodobý provoz je třeba vybrat kvalitní a odolnou techniku. Zařízení musí být připojeno přes Ethernet a musí být napájené ze sítě, nikoli pomocí baterie. Pro vývoj a testování jsem používal klasickou webovou kameru, ale pro produkční nasazení jsem na základě předešlé analýzy rozhodl o použití venkovní IP kamery. Kamera musí mít následující vlastnosti

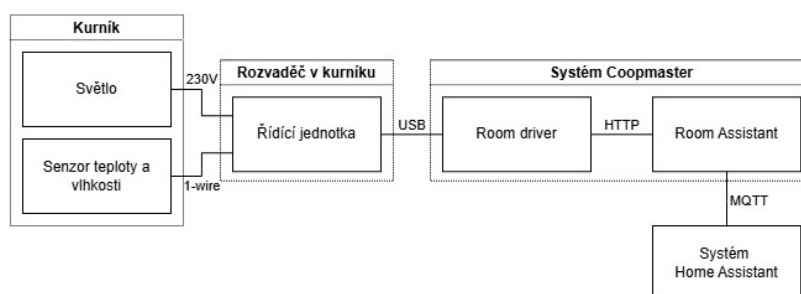
- Dostatečná odolnost proti vniknutí cizího tělesa a kapalin
- Dobrou kvalitou obrazu
- Velké zorné pole
- Jednoduchou instalaci
- Cenová dostupnost
- Kvalitní signál pro potřeby rozpoznávání

Podrobnosti ohledně výběru kamery se nachází v části 4.12

3.5 Spolupráce modulů dle požadavků uživatele

3.5.1 Ovládání světla, data o teplotě a vlhkosti

Každý tvor potřebuje k životu světlo a teplo a ani u slepiček to není jinak. Prodlužováním dne se jim prodlužuje čas, kdy se krmí a jsou aktivní. Zajištěný tepelný komfort přispívá jejich pohodě a následně větší produkci vajec. Tato část je tedy důležitá pro chovatele, aby měl na dálku přehled o podmínkách v kurníku, a to převážně o teplotě. Dalším z požadavků bylo dálkové ovládání a automatizace světla, čehož se tato sekce také dotýká. Graficky jsou použité moduly a jejich závislosti vyjádřeny v obrázku 3.1.



Obrázek 3.1: Diagram komunikace mezi relé a senzory Zdroj: [Autor]

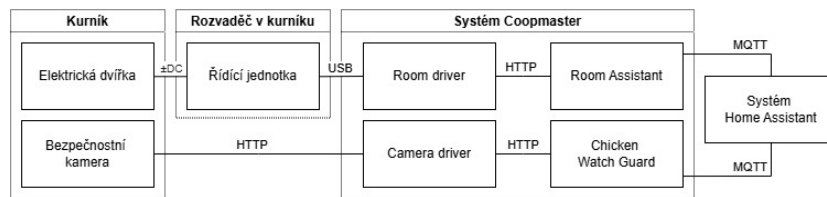
Hardwarový prvek, který dokáže ovládat světlo a zpřístupňuje systému informace o teplotě a vlhkosti, je řídicí jednotka. Ta je přes USB sběrnici propojena s Room Driverem, který je již součástí Docker Composu Coopmasteru. Room Driver je použit jako mezičlen pro komunikaci mezi hardwarovými prvky a výkonným kódem. Výkonný kód se nachází ve službě Room Assistant. Room Assistant je s Room Driverem propojen pomocí HTTP protokolu a dále komunikuje se systémem Home Assistant díky messaging protokolu MQTT. Home Assistant v této sekci poskytuje uživateli data o teplotě, vlhkosti a stavu světla. Uživatel pak může za pomoci Home Assistanta vysílat

požadavky na změnu stavu světla. Data o teplotě a vlhkosti Room Assistant aktualizuje každých 10 vteřin.

3.5.2 Automatizace dvířek

Jeden z klíčových požadavků, díky kterým vznikl tento projekt, je zautomatizování ranního otevírání a večerního zavírání dvířek v kurníku. V této části si tedy zjednodušeně popíšeme princip, jak je v systému Coopmaster řešené automatické ovládání dvířek. Graficky jsou závislosti mezi jednotlivými moduly vysvětleny v obrázku 3.2.

Dvířka v našem systému jsou kontrolována pomocí modulu řídicí jednotka. Module Room Driver zajišťuje pomocí USB sběrnice a HTTP protokolu komunikaci mezi řídicí jednotkou a Room Assistantem. Room Assistant je propojen se systémem Home Assistant pomocí MQTT protokolu. Dále potřebují získat data z kamerového systému konkrétně z kamery v kurníku. Komunikaci s kamerovým systémem zajišťuje přes HTTP protokol modul Camera Driver. Tento modul má vystavené REST API a poskytuje obrázky službám v systému Coopmaster. O obrázky si zde říká služba Chicken Watch Guard, která za pomoci neuronové sítě, konkrétně pak metody klasifikace [3], počítá slepice v kurníku a výsledný počet posílá opět pomocí MQTT do Home Assistanta. Chicken Watch Guard také předává co minutu do systému Home Assistant obrázek z kamery v kurníku. Home Assistant tedy pro tuto komponentu zajišťuje indikaci stavu dvířek, ovládací tlačítka pro dvířka, vizualizaci počtu slepic v kurníku a zobrazuje fotografii pořízenou v kurníku. Home Assistant je nakonfigurován tak, aby se od určitého časového okamžiku snažil zavřít dvířka pod podmínkou, že bude počet slepic, který identifikoval Chicken Watch Guard, stejný jako ten, jež jsme nastavili jako pevnou konstantu neboli aktuální počet slepic v kurníku.

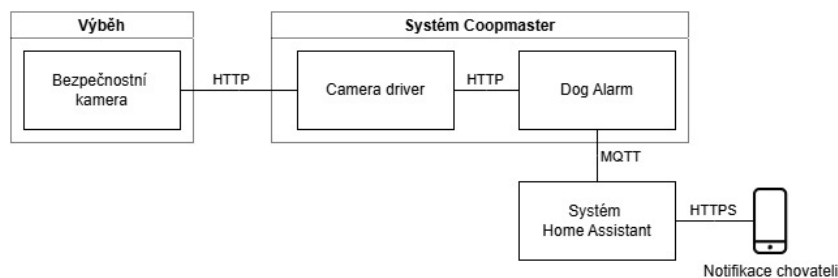


Obrázek 3.2: Diagram bezpečného zavření dvířek Zdroj: [Autor]

3.5.3 Detekce vetřelců

Pro tento úkol bude třeba využít obrázků z kamery ve výběhu a umělé inteligence, která v obraze rozpozná určené vetřelce. Princip komunikace je graficky vysvětlen v obrázku 3.3.

S kamerou ve výběhu komunikuje pomocí HTTP protokolu služba Camera Driver. Z REST aplikačního rozhraní služby si stahuje obrázky služba Dog Alert. Tato služba využívá neuronovou síť [4] k detekci vetřelce v obraze. Pokud je detekováno nějaké nebezpečí, služba pomocí MQTT protokolu informuje systém Home Assistant a předá mu aktuální fotografii, v níž byl vetřelec detekován jako důkaz. Home Assistant je nastaven tak, že při přijetí takové zprávy, odešle uživateli do aplikace v mobilním telefonu oznámení, aby ho upozornil na detekci hrozby. Dog Alarm dále pak každou minutu posílá do Home Assistanta aktuální snímek z kamery ve výběhu, aby měl uživatel stále přehled.

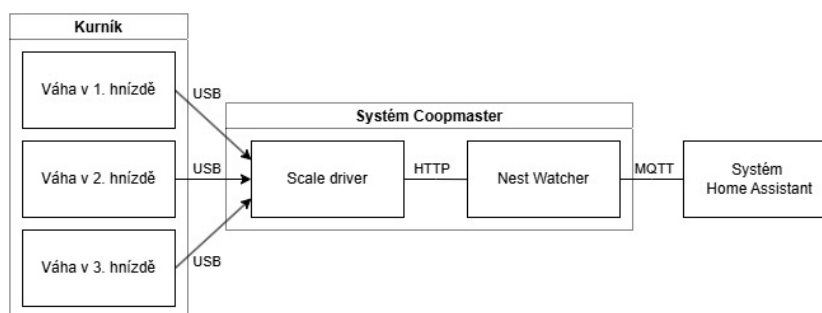


Obrázek 3.3: Diagram detekce vetřelců Zdroj: [Autor]

3.5.4 Vizualizace stavů hnízd

Abych mohl splnit tento požadavek, bude potřeba získat údaje o aktuální hmotnosti z jednotlivých vah ve hnízdech. Tato data je pak nutné předat do služby Nest Watcher. Nest Watcher následně data vyhodnotí a výsledky předá k vizualizaci do systému Home Assistant. Schématické vyjádření principu pro vizualizaci stavů hnízd je znázorněno na obrázku 3.4.

Pro komunikaci s modulem digitální váhy je použita služba Scale Driver, která s jednotlivými váhami komunikuje přes USB sběrnici. Data o váhách si ze Scale Driveru stahuje služba Nest Watcher. Tato služba ukládá v daném intervalu data z každé váhy do databáze. Služba po uplynutí časové konstanty analyzuje časovou řadu v databázi a vyhodnotí stav hnízda. Detailní popis algoritmu je v části věnované službě Nest Watcher (4.6). Po vyhodnocení stavů jsou data předávána pomocí MQTT protokolu do systému Home Assistant, který je díky na míru dělané komponentě zobrazí uživateli.



Obrázek 3.4: Diagram vizualizace stavů hnízd Zdroj: [Autor]

3.6 Zpřístupnění aplikace z internetu

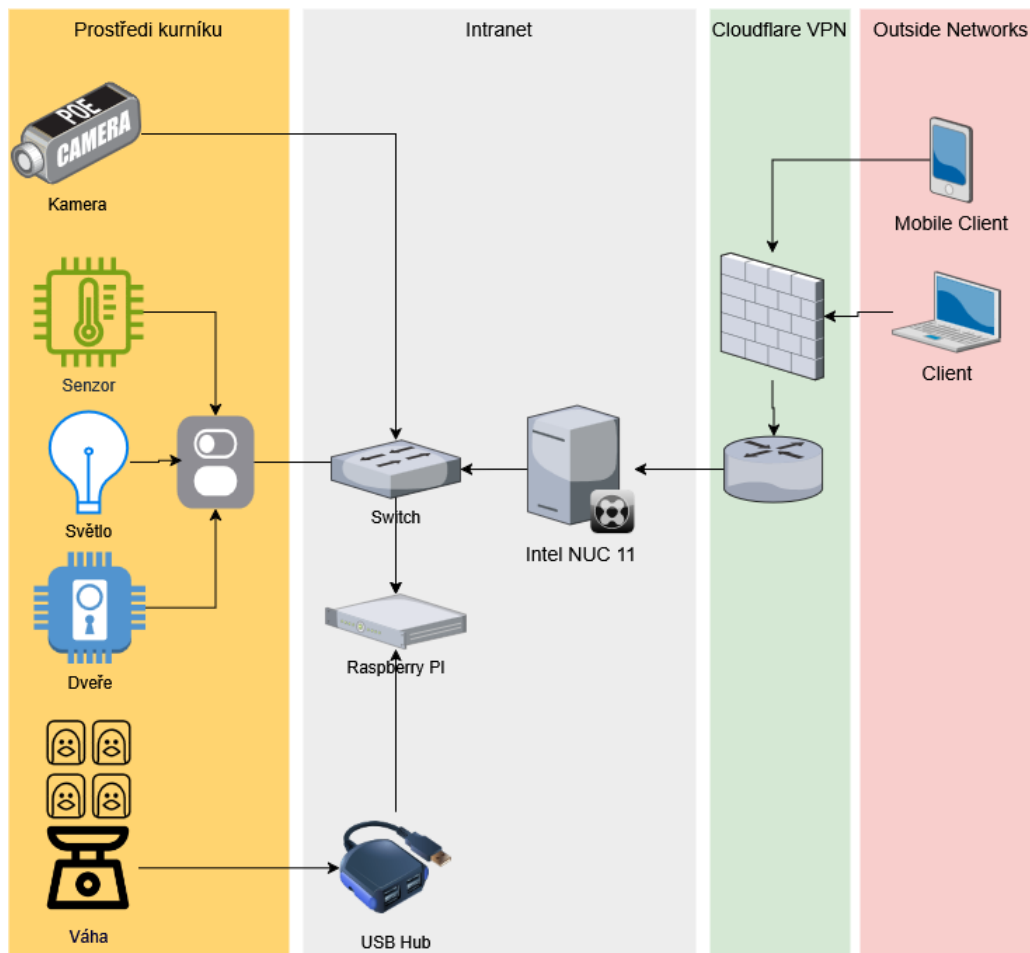
Aby systém mohl sloužit svému účelu, musí být vzdáleně přístupný. To znamená, že server, na kterém systém poběží, musí mít veřejnou IP adresu a v ideálním případě mít tuto adresu propojenou s doménovým názvem pro snadnější použití. Tato problematika se dá řešit několika způsoby.

Jeden ze způsobů je zřízení veřejné IP adresy pro svůj server. Toto řešení je, technicky i teoreticky náročné, protože v případě, se kterým pracujeme my, běží server na lokální síti a veřejná IP adresa tedy není přímo pro náš server, ale celou síť. To vytváří v určitých situacích velmi vážná bezpečnostní rizika, díky čemuž je tato metoda velice náročná na hardware a implementaci. Navíc poskytování veřejné IP adresy je placená služba poskytovatele internetového připojení.

Další způsob je nasazení části systému do některého z cloudových řešení jako je Microsoft Azure [5] nebo Amazon AWS [6]. Tato metoda by v podstatě vyhovovala našim požadavkům, ale pokud bychom se bavili o ceně, tak to rozhodně není levné řešení.

Metoda, kterou využíváme v našem řešení, je snadná, jednodušší na implementaci než zřizování veřejné IP adresy a levnější než využití cloudové služby. Tento způsob využívá takzvané tunely neboli šifrovaná spojení. Funguje na principu, při němž zařízení šifrovaně propojíte s veřejným serverem a tento server bude skrze sebe vystavovat službu, která běží na lokálním počítači například u nás doma. Výhoda je, že vzdálený server disponuje kvalitním a odborně nastaveným firewalem, který zajistí dobrou ochranu našeho lokálního serveru. Tuto službu poskytuje zdarma a bez omezení například společnost Cloudflare [7]. Tento způsob nejvíce odpovídá našim požadavkům, kterými byly cena a jednoduchost, společně s rychlostí nasazení. Pro lepší znázornění komunikace mezi moduly přikládám schéma 3.5.

Komunikační vrstva



Obrázek 3.5: Orientační topologie systému Zdroj: [Autor]

Poslední nutností, kterou ovšem je třeba zaplatit, je pronájem vlastního doménového jména. Doménu si lze pronajmout u doménových registrátorů jako jsou GoDaddy [8], Hostinger [9] nebo například Forpsi [10]. Ceny domén se odvíjí na základě jejího druhu a pohybují od 20 do 1500 korun za kus. My v našem řešení využíváme službu Forpsi, protože s ní mám již předchozí zkušenost.

Dále je tato část rozvedena v kapitole 8, kde je popsán konkrétní způsob nasazení do funkčního projektu.

4 Systémové moduly

4.1 Camera Driver

Camera Driver je služba zodpovědná za zprostředkování komunikace mezi IP kamerami a službami zpracovávajícími obraz (Dog Alarm, Chicken Watch Guard).

Popis algoritmu

Po startu služby naběhne REST API na portu 9001 vytvořené za pomoci Python frameworku Flask [11]. Toto API poskytuje endpoint pro získání aktuálního snímku z každé kamery, která je službě přiřazena v konfiguraci modulu. S kamerami pak služba komunikuje pomocí HTTP protokolu. Byla zvažována i možnost využití protokolu RTSP, ale jelikož moje řešení prozatím nemá vysoké nároky na živý přenos, ukázala se jako lepší volba využití REST API kamery, které umožňuje stačnou aktuálního snímku na požádání. Jakmile je přijat požadavek na získání aktuálního snímku z konkrétní kamery, služba za pomoci GET requestu stáhne obraz z restového API síťové kamery a vrátí ho ve Full HD rozlišení jako odpověď na HTTP dotaz. Endpoint kamery ze kterého má služba obrázky stahovat, získá z konfigurace předávané po-

mocí systémových proměnných. Kamery mají vlastní zabezpečení proti neoprávněnému přístupu k datům. V případě ověřování HTTP komunikace se používá metoda Digest Access Authentication [12], pomocí které je kameře předáváno uživatelské jméno a heslo.

4.2 Scale Driver

Scale driver zodpovídá za komunikaci mezi fyzickou váhou a službami, které využívají data o vážení.

Jako řídicí jednotka váhy je použito Arduino(11.11). Scale driver modul je připojen USB kabelem a komunikuje přes sériový port. Pomocí jednoduchého proprietárního protokolu a na přijatý dotaz poskytne jako odpověď aktuální hodnotu načtenou ze senzoru váhy. Scaler driver má vystavené REST rozhraní, které umožňuje komunikaci s ostatními komponentami systému.

Popis algoritmu

Nejprve naběhne hlavní vlákno, které se stará o poskytování REST API na portu 9004. Toto API obsahuje endpoint pro vrácení aktuální hmotnosti na váze pomocí požadavku s metodou GET. Následně je spuštěno nové vlákno, které má za úkol číst data z váhy a předávat je aplikačnímu rozhraní jako aktuálně naměřenou hmotnost na váze, jež se bude od teď poskytovat, pokud o ní někdo GET requestem požádá. Čtení informací z váhy se provádí v nekonečném cyklu. Před začátkem cyklu se do log souboru zapíše informace ohledně úspěšného spuštění nového vlákna. Poté proběhne otevření komunikace s váhou přes seriový port s využitím knihovny PySerial [13]. Po úspěšném otevření spojení se váze vždy po uplynutí časového intervalu 2 sekundy, odešle znak "w". Tento znak je ve firmwaru váhy veden jako příkaz, po jehož přijetí má váha vrátit aktuální naměřenou hodnotu. Časové zpoždění je implementováno kvůli tomu, že není třeba číst data tak často a zároveň by mohlo docházet k zahlcení váhy. Jakmile je celá odpověď od váhy přijata programem zpět, jsou data zvalidována a následně, pokud je to možné, převedena na datový typ Int. Datový typ Int je zvolen z důvodu, že váha posílá data v gramech a je zbytečné v tomto případě používat desetinná čísla, pro-

tože gramy poskytují dostatečné rozlišení pro naše účely. Pokud se nepodaří převést přečtenou hodnotu na číslo, je výstupní proměnná nastavena na -1, což zbytku systému naznačí, že váha není v pořádku. Na závěr se zalogují aktuálně získaná data a následuje další volání cyklu.

4.3 Room Driver

Room Driver zajišťuje komunikaci mezi službou Room Assistant a řídicí jednotkou v kurníku. Tato jednotka se stará o ovládání dveří, světla a poskytování dat o aktuální teplotě a vlhkosti. Ovládání nízkoúrovňových komponent a komunikaci s připojeným počítačem zajišťuje Arduino Nano. Arduino je připojeno pomocí USB sběrnice k počítači, který na základě domluvených pravidel jednoduchého komunikačního protokolu posílá znaky, na které Arduino příslušně reaguje dle programu. Tato služba má tedy za úkol přes sériové rozhraní posílat příkazy a načítat stavy řídicí jednotky na základě požadavků přijatých na REST API.

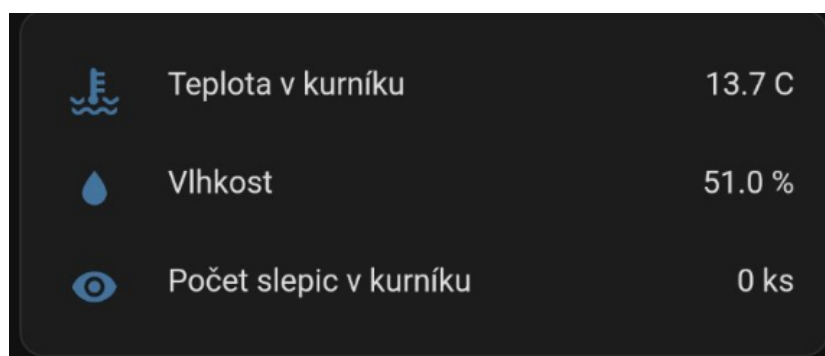
Popis algoritmu

Po startu služby se nejdříve vytvoří instance třídy `ArduinoReader`. Jedná se o vlastní třídu, při inicializaci vytvoří instanci sériového spojení k Arduino. Je zde implementován connection pool, který se stará o zdravé spojení po celou dobu běhu aplikace. Třída má jedinou metodu, a tou je `run_command`. Tato metoda přijímá jako parametr jeden znak, který při provolání metody pošle zkrze sériové připojení do Arduina a vrátí data typu `String`, kterými odpoví Arduino zpět. Jakmile se povede provést inicializaci sériového spojení je spuštěno REST API na portu 9005. Toto API poskytuje endpointy pro vyslání příkazů k zavření a otevření dvířek, zhasnutí a rozsvícení světla, informací o aktuálních stavech jednotky a informací o teplotě a vlhkosti v kurníku. Informace o aktuálním stavu jsou dobré pro případ, kdy se budou muset služby restartovat a bude třeba načíst reálný aktuální stav. V případě, kdy přijde požadavek na konkrétní endpoint, jako první věc se provede odeslání příslušného znaku jako příkaz Arduino. Následně se počká na jeho odpověď, pokud je požadavek typu POST. Odpověď je následně převedena do JSON objektů a vrácena v HTTP odpovědi s typem těla odpovědi JSON

(obrázek 4.1).

Příkazy ovládacího protokolu

- o = otevřít dvířka
- c = zavřít dvířka
- l = rozsvítit světlo
- d = zhasnout světlo
- j = vrátí zprávu ve formátu JSON s daty o teplotě a vlhkosti
- s = vrátí zprávu ve formátu JSON s daty o stavech jednotlivých ovládaných prvků (dveře, světlo)



The image shows a dark-themed interface with three rows of sensor data. Each row has a blue icon on the left, a text label in the middle, and a numerical value with a unit on the right. The icons are a thermometer, a water drop, and an eye.

	Teplota v kurníku	13.7 C
	Vlhkost	51.0 %
	Počet slepic v kurníku	0 ks

Obrázek 4.1: Vizualizace dat ze senzoru v systému Home Assistant
Zdroj: [Autor]

4.4 Health Checker

Health Checker je malá služba určená pro správce systému. Poskytuje informace o tom, zda všechny služby běží, aby správce nebyl nucen přihlašovat se vzdáleně na server a manuálně kontrolovat každou službu. Výpis stavů jednotlivých služeb je poskytován pomocí jednoduché webové stránky.

Popis algoritmu

Po nastartování služby je spuštěno REST API na portu 9000 pro komunikaci se správcem. Jediným endpointem služby je `<server:port>/status`. Pokud uživatel pomocí prohlížeče pošle GET požadavek na tuto službu, bude mu vrácen seznam služeb a jejich stavů převedený do formátu JSON (obrázek 4.3). Stavem aplikace je myšlen status kód a případné chybové hlášky, které služby po zavolání jejich metody `<server:port>/ping`.

Seznam služeb, které má Health Checker kontrolovat, je načítán z konfiguračního souboru při startu služby. Jednotlivé služby jsou v seznamu identifikovány na základě IP adresy a portu.

Algoritmus funguje na základě plánovače úloh neboli scheduleru, který jednou do minuty spustí kontrolu všech služeb a jejich stavy uloží do seznamu odkud jsou pak načítány při požadavku na API. Kontrola se provádí právě posláním HTTP požadavku na `<server:port>/ping` každé služby a následným uložením odpovědi.

Tento modul umí poskytovat základní informace o aktuálním stavu služeb systému Coopmaster. Pokud by se v budoucnu objevil požadavek na historická data, bylo by velmi jednoduché integrovat tuto komponentu s některou z open source platforem určenou pro analytiku a vizualizaci dat. Například Grafana [14] umožňuje uživatelům vytvářet interaktivní a dynamické panely (dashboards), které vizualizují různorodé časové řady dat získané z různých zdrojů (obrázek 4.2).



Obrázek 4.2: Možná vizualizace pomocí platformy Grafana Zdroj: [Autor]

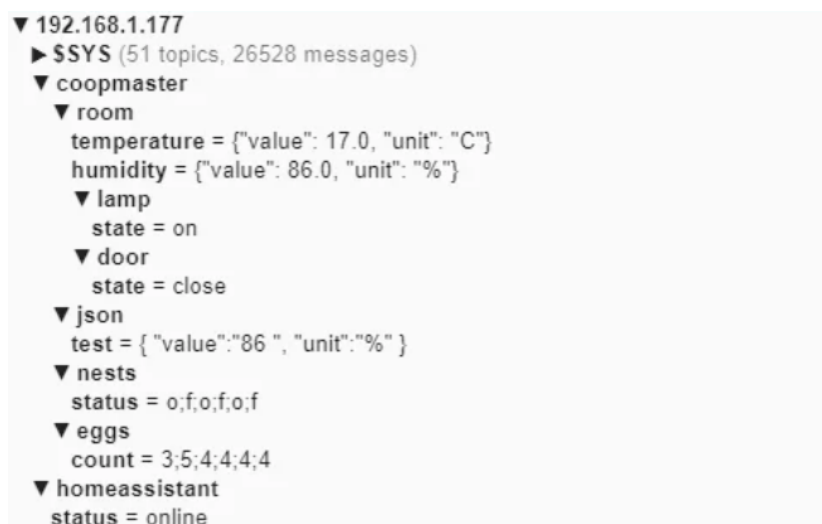
```
[
  "Service at 100.75.249.82:9001 - Status code: 200 - Camera Driver ready",
  "Service at 100.118.178.108:9003 - Status code: 200 - Chicken Watch Guard ready",
  "Service at 100.75.249.82:9004 - Status code: 200 - Scale Driver ready",
  "Service at 100.75.249.82:9005 - Status code: 200 - Room Driver ready",
  "Service at 100.118.178.108:9006 - Status code: 200 - Room Assistant ready",
  "Service at 100.118.178.108:9007 - Status code: 200 - Nest Watcher ready",
  "Service at 100.118.178.108:9008 - Status code: 200 - Dog Alarm ready"
]
```

Obrázek 4.3: Výstup ze služby Health Checker Zdroj: [Autor]

4.5 Room Assistant

Room Assistant zpřístupňuje komunikaci mezi systémem Home Assistant a službou Room Driver.

S Home Assistantem je komunikace realizována pomocí MQTT(11.9) a s Room driverem pomocí HTTP protokolu. Tato služba zpřístupňuje do systému Home Assistant data o teplotě a vlhkosti a Home Assistant přes ni ovládá světlo a dvířka. Při vývoji, ale i při provozu si můžeme zprávy prohlížet pomocí klientských aplikací (obrázek 4.4).



Obrázek 4.4: Prohlížeč zpráv uložených na MQTT serveru. Zdroj: [Autor]

Popis algoritmu

Po odstartování služby se stane hned několik věcí. Jako první naběhnou MQTT subscribery pro přijímání příkazů ze systému Home Assistant. Jeden přijímá příkazy pro dvířka, druhý pro světlo. Vytvoří se vlastně instance dvou tříd LampTimeChecker a DoorTimeChecker.

Tyto instance mají každá svou instanci třídy NestMQTTClient, která za-

obaluje základní funkcionalitu ohledně používání protokolu MQTT. NestMQTTClient zaobaluje metody pro publikování a odebírání zprávy, připojení a odpojení od MQTT brokeru, reakci na navázané spojení s MQTT brokerem a reakci na odebrání nové zprávy. LampTimeChecker odebírá topic coop-master/room/lamp/cmnd a zpracovává zprávu pokud je jejím obsahem "on" nebo "off". Na základě přijatého obsahu zprávy se zavolá statická metoda call_room_driver_command() třídy driver_client, aby odeslala příslušný příkaz do Room Driveru na endpoint, který je předáván jako parametr. DoorTimeChecker funguje stejně jako NestMQTTClient, ale odebírá topic coop-master/room/door/cmnd a reaguje pokud je obsah MQTT zprávy "open" nebo "close".

Po startu se rozběhne BackgroundScheduler, který má za úkol vždy po uplynutí 5 vteřin spustit nový běh akce pro aktualizování stavů z řídicí jednotky. Podmínkou pro spuštění nového běhu jobu je, že předchozí běh je dokončen. Tuto funkci má již v sobě implementovanou scheduler z knihovny APScheduler [15], který využívám. Akce pro aktualizaci obsahuje jednu metodu a to detect_hardware_state(). Tato metoda nejdříve stáhne data z Room Driveru. Následně načte z konfigurace topic pro stav dvířek, stav světla, teplotu a vlhkost. Dále jsou pak data z Room Driveru zveřejněna na téma daná konfigurací, odkud je přijímá systém Home Assistant a vizualizuje je.

4.6 Nest Watcher

Tato služba interpretuje stavy jednotlivých hnízd v kurníku pro Home Asistanta. Úkolem této služby je načítání a analýza dat z jednotlivých hnízd v kurníku. Každé hnízdo je reprezentováno jednou instancí služby Scale Driver. V aktuální situaci služba obsluhuje šest hnízd, přičemž se data z každého hnízda monitorují a vyhodnocují. Toto komplexní sledování umožňuje efektivní řízení kurníku, zajišťuje optimální péči o slepice a včasný sběr vajec. Přístup založený na datech zvyšuje naši schopnost udržovat zdravé slepice a maximalizovat jejich produktivitu.

Popis algoritmu

Data jsou načítána několikrát za minutu a ukládána do databáze s časovým údajem, kdy byl záznam vytvořen. Záznam v databázi obsahuje následující sloupce:

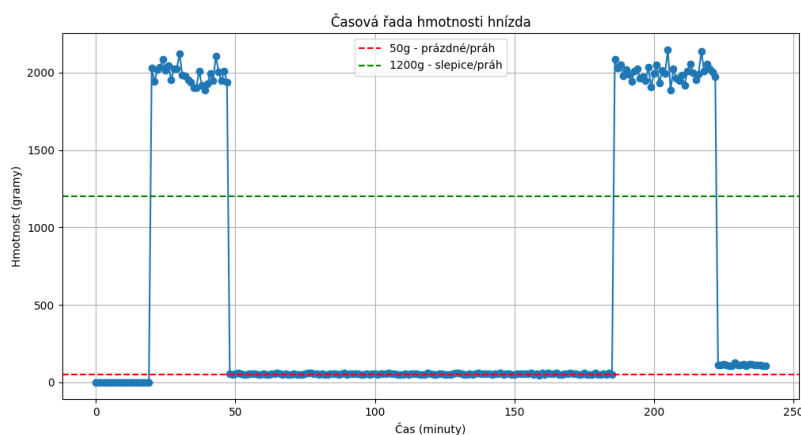
- id - jednoznačný identifikátor záznamu v databázi.
- nest_id - identifikační číslo hnízda
- weight - naměřená hmotnost v gramech pro dané hnízdo
- timestamp - čas vytvoření záznamu

Následně se jednou za minutu vyhodnotí průměrná hodnota během posledních několika vážení. Na základě vyhodnocené průměrné hmotnosti jsme schopni zjistit několik hlavních stavů hnízda:

- Hnízdo je prázdné - pokud průměrná hmotnost nepřevyšuje 50 gramů (vzhledem k možným chybám měření), vyhodnotíme, že hnízdo je prázdné. V hnízdě není ani slepice, ani vejce.

- V hnízdě se nacházejí vejce - jestliže se hodnota hmotnosti pohybuje mezi 50 a 1200 gramy, znamená to, že v hnízdě jsou pravděpodobně vejce. Hmotnost jednoho vejce je přibližně 50 gramů, takže počet vajec lze vypočítat vydělením celkové hmotnosti hmotností jednoho vejce.
- V hnízdě sedí slepice - pokud je průměrná hmotnost na váze více než 1200 gramů, vyhodnotí služba, že v hnízdě sedí slepice. Hmotnost slepice se pohybuje kolem 1200 gramů a více.

Pro lepší pochopení jsem přiložil grafickou reprezentaci dat z databáze (4.5). Znázorňuje vývoj hodnoty hmotnosti v čase pro konkrétní hnízdo. Na ose x je znázorněn čas a na ose y je hmotnost. Z grafu je možné odvodit následující příběh. Ve dvacáté minutě vlezla slepice s hmotností kolem 2000 gramu na váhu. Po 28 minutách ji opustila a zůstalo po ní snesené vejce s hmotností 55 gramu. Další slepice vstoupila do hnízda ve 186 minutě a opustila hnízdo za 37 minut. Následně pak váha ukazuje hmotnost kolem 115 gramu, což při znalosti hmotnosti jednoho vejce indikuje přítomnost 2 vajec.



Obrázek 4.5: Graf vývoje hmotnosti hnízda v čase Zdroj: [Autor]

Výsledná data pro jednotlivá hnízda služba pomocí MQTT protokolu předává do systému Home Assistant, aby je zpracoval a vizualizoval chovateli.

Popis možných vylepšení práce s váhou

Díky měřením hmotnosti můžeme odvodit i další vzorce, které mohou být využity k diagnostice závady na váze a nebo identifikaci toho, že některá ze slepic má nějaký problém. V případě, kdy by se data z váhy na základě času spojila společně se záběrem z kamery, dalo by se v tomto případě přesně určit, o kterou ze slepic se jedná.

Určit by slepice mohl chovatel díky tomu, že by mu byl předložen snímek, na němž je slepice v konkrétním hnízdě, které hlásí problém. Konkrétní slepici by časem mohla detekovat i neuronová síť.

Toto řešení je však velmi komplikované, protože vzhledem k vysoké podobnosti bude složité rozlišit například dvě podobně bílé slepice a chovateli jednoznačně sdělit, která slepice má problém. Na druhou stranu, pokud by se tento princip reidentifikace dovedl k dokonalosti, přinesl by revoluci monitoringu pomocí kamerových systémů. Dále se o implementaci vlastního reidentifikačního algoritmu hovoří v sekci 10.1.

Kromě výše zmíněných stavů by se v budoucnu z váhy dal identifikovat i následující seznam situací.

- Doba setrvání slepice v hnízdě - analýzou délky období, kdy je hmotnost hnízda nad 1200 gramů, lze zjistit, jak dlouho slepice v hnízdě zůstává. To může upozornit na možné zdravotní problémy, pokud setrvává v hnízdě déle než obvykle.
- Frekvence snášení u jednotlivých hnízd - porovnáním počtu snesených vajec v různých hnízdech lze zjistit, zda některá hnízda nejsou preferována více než jiná. To může vést k úpravám uspořádání kurníku nebo kontrolám hnízd, která jsou méně využívána.
- Poruchy senzorů - pokud hmotnost zůstává nezměněna po dlouhou dobu nebo vykazuje nerealistické hodnoty, může to indikovat technickou závadu. Pravidelná kontrola a kalibrace senzorů zajistí spolehlivost dat.

- Denní vzorce aktivity - analýzou hmotnostních dat během dne lze určit období nejvyšší aktivity slepic. To může být užitečné pro plánování krmení, čištění kurníku nebo jiných činností, které by mohly slepice rušit.

4.7 Dog alarm

Jednou z klíčových komponent systému Coopmaster je služba pro ochranu slepic před potenciálním nebezpečím ve výběhu, nesoucí název Dog Alarm. Za využití modelu Yolo11 služba detekuje vetřelce (psa) na fotkách z kamery ve výběhu a informuje pomocí MQTT protokolu chovatele do systému Home Assistant.

Obecný princip

Jako zdroj dat je použit kamerový systém ve výběhu, který je reprezentován službou Camera Driver. Z této služby pak Dog Alarm stahuje pomocí GET požadavků fotografie z kamery ve výběhu.

Pro detekci predátorů je využito modelu Yolo11. Model je velmi kvalitně vyučený a společnost Ultralytics [16] k němu nabízí rozsáhlou dokumentaci, což je také jeden z hlavních důvodů mé volby. Před nasazením jsem model ještě doučil o další fotografie psa pořízené z venkovní kamery, podobně jako je to v sekci 5.1, aby se model více přizpůsobil podmínkám vstupních fotografií. Jakmile model identifikuje přítomnost psa nebo jiného predátora, služba Dog Alarm okamžitě vygeneruje výstrahu. Tato výstraha je odeslána pomocí protokolu MQTT do systému Home Assistant. Spolu se zprávou je připojen specifický záběr, na kterém byl predátor detekován, aby mohl chovatel zhodnotit situaci sám.

Kromě výstražných zpráv, Dog Alarm pravidelně zasílá aktuální záběry z kamery do Home Assistantu, čímž umožňuje uživatelům mít průběžný přehled o situaci ve výběhu.

Popis algoritmu

Po startu služby se inicializuje REST API vytvořené pomocí Flask frameworku. Toto API poskytuje endpoint pro volání služby Health Checker s dotazem na stav služby. Následně je spuštěn scheduler, který má za úkol jednou za minutu spustit úkol s názvem `check_dog`. Jako poslední je spuštěno poskytování REST API na portu 9008.

Popis algoritmu pro detekci vetřelců

Tento úkol běží v pravidelných intervalech detekuje vetřelce v obraze a stará se o report informací. Prvním krokem je stažení aktuálního snímku z kamery ve výběhu. Další následuje načtení příslušného Yolo modelu dle konfigurace. Následně je provedena detekce objektů v obraze (ukázka výsledku je na obrázku 4.6), o čemž se dále rozepisují v sekci 5.1. Výsledný seznam detekovaných objektů se analyzuje a zjistí se, zda obsahuje kategorii pes. Na závěr se předávají aktuální výsledky na MQTT server. Pokud byl detekován pes posílá se zpráva na topic `coopmaster/dog/detected`. Dále posíláme aktuální obrázek na topic `coopmaster/dog/image/actual`. Tyto MQTT topics odebírá systém Home Assistant a dále s nimi pracuje, což je popsáno v sekci 4.9.



Obrázek 4.6: Pes detekován ve výběhu Zdroj: [Autor]

4.8 Chicken Watch Guard

Služba Chicken Watch Guard je dalším klíčovým prvkem systému Coopmaster. Poskytuje data o počtu slepic v kurníku a tato data předává do systému Home Assistant.

Obecný princip

Podobně jako u služby Dog Alarm, Chicken Watch Guard používá kamerový systém pro získávání aktuálních záběrů z kurníku. Tyto záběry jsou staženy ze služby Camera Driver 4.1. Získané obrázky se analyzují modelem Yolo11, který je upravený pro přesnější detekci v našich lokálních podmínkách. Výsledky detekce společně s aktuální fotografií jsou pomocí protokolu MQTT [17] odeslány do systému Home Assistant, který je využije při automatizaci dvířek a vizualizuje v uživatelském rozhraní.

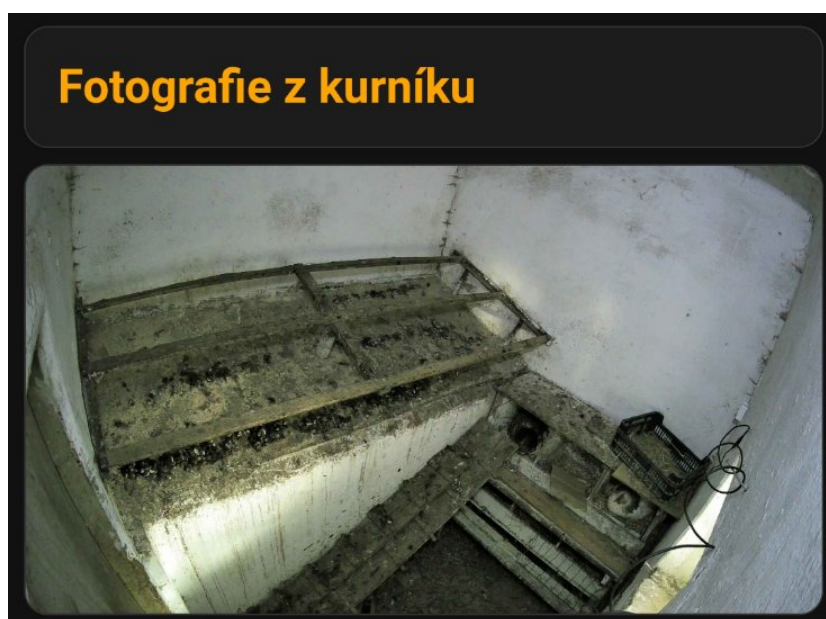
Popis algoritmu

Jako první je vytvořeno API definované pomocí Flask frameworku pro komunikaci se službou Health Checker. Následně je inicializován a spuštěn scheduler, který má za úkol jednou za minutu spustit job s názvem `check_chicken`. Závěrem spouštěcí metody se spustí propagace API na portu 9003, který je předáván jako proměnná prostředí při startu služby.

Popis algoritmu pro počítání slepic

Po spuštění se jako první provolá metoda `get_image`, která je zodpovědná za stažení aktuálního obrázku ze služby Camera Driver. Dále je inicializován detekční upravený model Yolo11 a provedena detekce objektů v obraze (sekce 5.1). Výsledkem je seznam objektů, které se podařilo identifikovat.

Hlavními daty, která obsahuje jedna položka v seznamu, jsou informace o detekované kategorii a oblast v obraze, kde přesně byl objekt nalezen. Počet slepic v kurníku je dán počtem, kolikrát se v seznamu vyskytuje kategorie slepice. Job končí publikováním výsledného počtu slepic na MQTT topic "coopmaster/chicken/count" a aktuální fotografie (obrázek 4.7) na MQTT topic "coopmaster/chicken/image/actual". Systém Home Assistant je MQTT subscriber zmíněných topics a reaguje na ně, jak je popsáno v sekci 4.9.



Obrázek 4.7: Obrázek přijatý z MQTT a vizualizovaný v dashboardu
Zdroj: [Autor]

4.9 Uživatelská aplikace

Na začátku této práce jsem si v kapitole 2 identifikoval požadavky, které by měl systém Coopmaster plnit. V této části se podíváme na functionality z pohledu uživatelského rozhraní.

Pro implementaci jsem, po prvních pokusech s vlastní aplikací, zvolil open source systém pro chytré domácnosti Home Assistant (11.10). Mezi jeho hlavní výhody patří robustnost, konfigurace, integrovatelnost a široká možnost vizualizace dat. Vytvořit takto robustní a kvalitní uživatelské rozhraní by za mě trvalo velmi dlouho, a to nepočítám čas na jeho odladění a případné opravy.

4.9.1 Požadavky na systém

Vzdálený přístup

Má Home Assistant podporu pro vzdálený přístup? Odpověď je ano. Home Assistant umožňuje vzdálený přístup, což uživatelům dovoluje řídit a konfigurovat jejich chytrou domácnost odkudkoli. Pokud se provozuje systém v cloudu, je to výrazně jednodušší. V mém případě Home Assistant běží na lokálním stroji a je přístupný pouze z lokální sítě. Tuto drobnost jsem vyřešil využitím privátního tunelu [18], který poskytuje například společnost Cloudflare [7].

Podrobně je výběr technologie pro vzdálený přístup popsán v sekci 3.6.

Při své práci jsem systém Home Assistant využíval již od začátku skrze webové rozhraní a mohu tedy potvrdit, že práce s ním je příjemná a vyhovující. Vzdáleným přístupem tedy systém disponuje a nemusíme ho tedy nijak složitě implementovat.

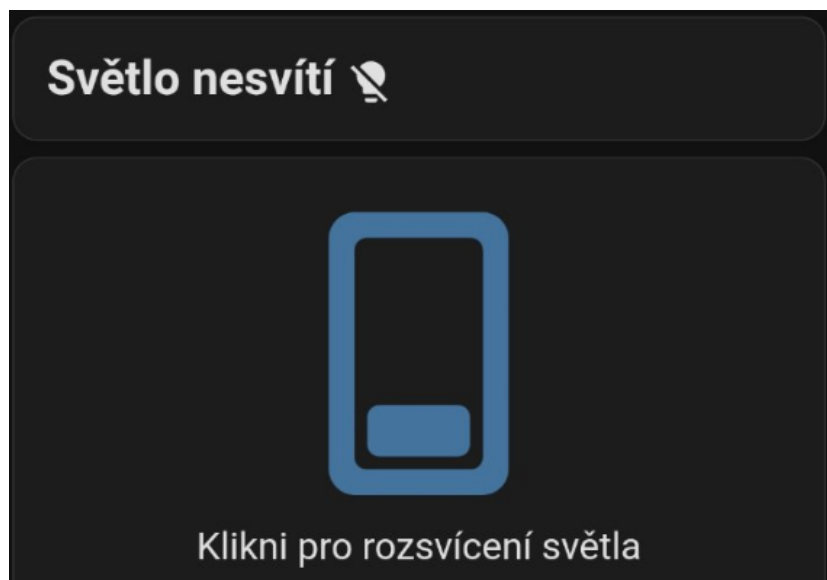
Pohledy z bezpečnostních kamer

Zobrazování obrázků z kamer je jednou ze základních komponent, kterou Home Assistant nabízí. Stačí si vybrat komponentu, která bude schopna zpracovat data z MQTT serveru. Já jsem si vybral komponentu Picture [19]. Použil jsem pro ni základní konfiguraci, jen jsem jí nastavil atribut title, abych si na dashboardu odlišil přehledová data z výběhu a kurníku.

Dálkové ovládání a automatizace světla a dvířek v kurníku

Tímto bodem rozumíme v první řadě ovládací komponenty/tlačítka [20], která bude využívat chovatel, aby manuálně vyslal příkaz k provedení nějaké akce. Za druhé se zde dostáváme k dalšímu skvělému využití systému Home Assistant, a to je automatizace. Využitím systému Home Assistant lze automatizovat právě reakce na hodnoty různých čidel a senzorů. Může se jednat o automatizaci od úplných základů jako je zapnutí topení, když je v bytě zima, až po celé inteligentní domácnosti ovládané hlasem. Takové domácnosti například na základě toho, kdo přišel domů, nastaví barvu světla, spustí jeho oblíbenou hudbu a uvaří kávu. Já ve svém projektu využiji automatizaci k tomu, aby zavírala dvířka na základě času a počtu slepic, který poskytuje služba Chicken Watch Guard 4.8 systému Coopmaster. Dále pak automatizace na základě času zapíná a vypíná světlo.

Pro splnění tohoto požadavku bude tedy třeba nakonfigurovat automatizaci a vytvořit ovládací tlačítka (obrázek 4.8).



Obrázek 4.8: Komponenta pro ovládání světla systému Home Assistant
Zdroj: [Autor]

Intuitivní vizualizace stavů jednotlivých hnízd (sedící slepice, v opačném případě počet vajec)

Tento bod je ze všech nejnáročnější. Home Assistant totiž nedisponuje žádnou pěknou komponentou přímo pro vizualizaci hnízd v kurníku a ani žádnou jinou, která by se dala po nějakých úpravách využít.

Z tohoto důvodu bude tedy třeba vytvořit vlastní komponentu pro systém Home Assistant, která půjde snadno parametrizovat dle požadavků kurníku a bude umět zpracovat data, která přijdou ze služby Nest Watcher systému Coopmaster.

Upozornění na detekování vetřelce ve výběhu

Zde se bude opět jednat o automatizace. Požadavkem je, aby v případě poplachu systém poslal notifikaci chovateli a zobrazil na dashboardu aktuální fotografii, kde byl vetřelec detekován.

Abychom splnili tento bod, budeme muset nastavit automatizaci, která zareaguje

na určitý stav nějaké MQTT entity a následně vytvoří notifikaci. Tato notifikace vyskočí i na chovatelově telefonu. Dále bude použita komponenta s názvem Conditional [21], která umožňuje na základě hodnoty měnit vizualizaci komponent na dashboardu. Při určitém stavu MQTT entity zobrazí v dashboardu upozornění na vetřelce ve výběhu a v opačném případě nezobrazí nic. Což je velmi šikovné právě pro různá upozornění.

Data o teplotě a vlhkosti

Tyto dva požadavky nebudou co do implementace nic složitého. Jedná se pouze o dvě hodnoty, které navíc ze služby Room Assistant chodí ve formátu JSON, takže není problém je zpracovat.

Bude tedy třeba jen nakonfigurovat MQTT entitu, kterou následně bude načítat komponenta Entities. Tato komponenta zobrazuje v seznamu pod sebou entity, jejichž seznam přebírá jako parametr. Pro zobrazení každé entity je možno upravit ikonu a zobrazovaný název, což přesně odpovídá našim požadavkům.

4.9.2 Postup konfigurace systému Home Assistant

Po rozvržení řešení jsem se začal seznamovat se systémem Home Assistant, s čímž mi hodně pomohla rozsáhlá a kvalitní dokumentace společně s diskusními fóry, která jsem hojně využíval.

MQTT entity

Konfiguraci jsem započal definováním všech potřebných MQTT entit [22, 23] v souboru configuration.yaml, do kterých systém ukládá data přijatá z MQTT brokeru a následně je můžeme využít v našem dashboardu. Používám tři typy MQTT entit.

- sensor - přijímá běžné hodnoty senzoru jako čísla a texty (maximální velikost je 255 bytů)

- image - umožňuje přijmout obrázek z MQTT brokeru
- binary_sensor - na základě nakonfigurovaných hodnot je jeho stav buď 0 nebo 1

Každá MQTT potřebuje nastavit jednoznačné jméno pro identifikaci v systému a topic, ze kterého bude odebírat zprávy.

Základní dashboard

Jakmile se nakonfiguruje entity, následuje tvorba samotného dashboardu. Za pomoci webového rozhraní jsem tedy přidal tlačítka, vizualizace obrázků a výpisy hodnot entit tak, jak bylo rozebráno dříve v návrhu. Po přidání komponent bylo třeba přiřadit odpovídající entity ke komponentám opět s využitím webového rozhraní.

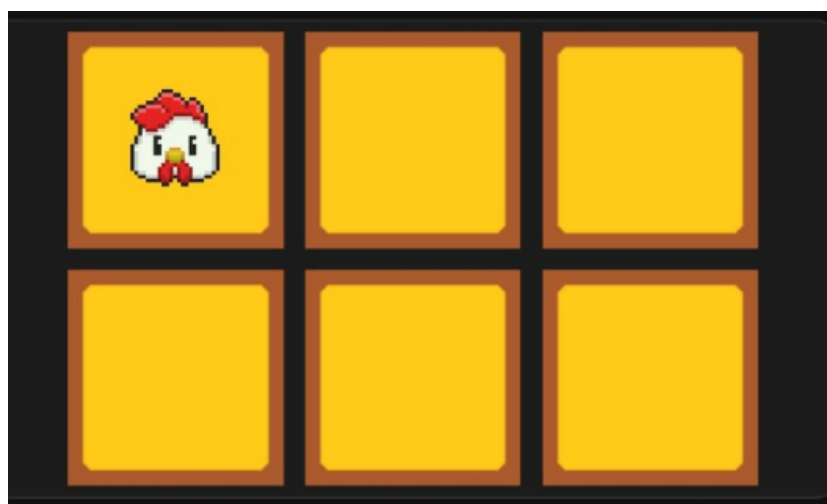
Automatizace

Díky intuitivnímu webovému rozhraní systému není problém přidat jakoukoli automatizaci. Automatizaci [24] si lze představit stejně jako klasickou podmínku v programování. Je zde sekce Spouštěč, která nastavuje hlavní důvod spuštění automatizace. Následuje volitelná sekce A Pokud. Tato sekce udává další omezení pro spuštění scriptu. Na závěr je zde sekce Pak Provést, která právě specifikuje akci nebo akce, které se mají uskutečnit při splnění podmínek. S těmito vědomostmi jsem opět nakonfiguroval dle návrhu potřebné automatizace pro světlo, dvířka a notifikace.

Tvorba vlastní Home Assistant komponenty

Zde se již dostáváme k opravdovému programování. Inspirací, jak komponentu vytvořit, mi byla vývojářská dokumentace [25]. Služba Nest Watcher poskytuje data o tom, zda je hnízdo obsazené a kolik je v každém hnízdě vajec. Tato data jsou ve formátu CSV, kdy každý prvek v poli znamená jedno hnízdo. Data se do komponenty dostávají z entit. Tyto entity se nastavují jako parametry během přidávání komponenty na dashboard. Data, která

dostaneme z entity, jsou v CSV formátu, takže se pomocí funkce split rozdělí podle separátoru, kterým je v tomto případě znak středníku. Skript pokračuje do další části, kde dochází ke zpracování naparsovaných dat. Jednoduchými dvěma for cykly se zajišťuje vykreslování nastaveného počtu řádků a sloupců v kurníku. Jedno hnízdo se vykresluje obarvený DIV a jeho child komponenta je DIV, který slouží k vizualizaci stavů hnízda. Graficky jsou DIVy nastýlovány tak, aby uživateli připomínaly hnízdo. Do child divu se následně vykreslí vpravo dole počet vajec, pokud je větší než 0, a zároveň se ve středu vykresluje ikonka slepice, která značí, že slepice sedí v hnízdě (obrázek 4.9). Komponenta také při každé aktualizaci dat loguje do konzole pro případ, kdy by se něco pokazilo.



Obrázek 4.9: Vlastní komponenta pro systém Home Assistant Zdroj: [Autor]

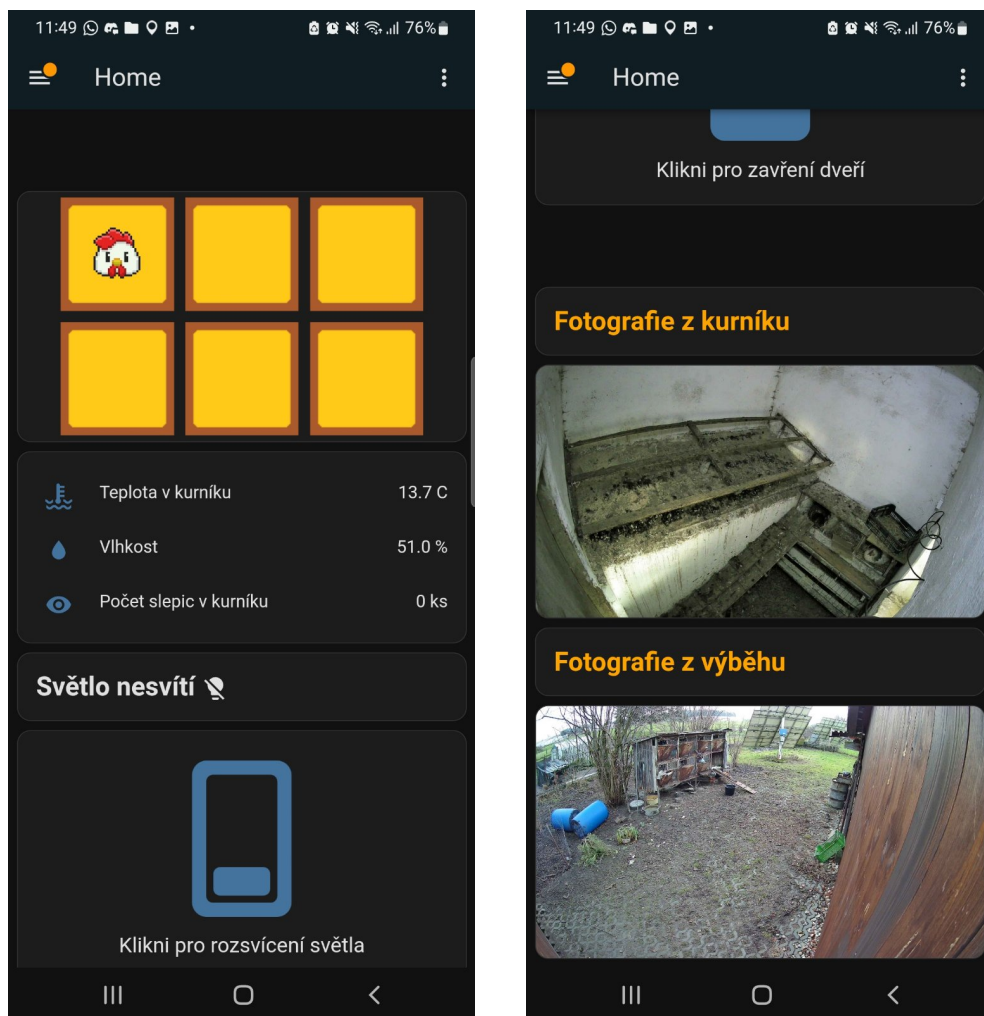
Maximální velikost hodnoty senzoru

Během tvorby komponenty pro hnízda, mě poněkud zaskočil fakt, že maximální velikost hodnoty state v entitě je 255 bytů. Z počátku jsem chtěl posílat data o hnízdech ve formátu JSON. Dlouho mi to procházelo, ale problém začal v situaci, kdy jsem serializoval celý seznam s objekty hnízd a chtěl ho poslat jako hodnotu senzoru. Nebylo jednoduché na to přijít, pro-

tože v tomto případě Home Assistant data prostě nenačítal a já nevěděl proč. Nakonec jsem se na diskusním fóru dočetl [26], že velikost hodnoty state je omezena. Byl jsem tedy donucen začít používat formát CSV a posílaná data omezit pouze na nejnutnější informaci o stavu. Pořád je zde problém s maximální velikostí v případě, že bych měl více než 128 hnízd, místo by mi i tak nevystačilo. Na kontrolu tolika hnízd bych ale potřeboval pravděpodobně o dost celkově robustnější infrastrukturu systému Coopmaster, takže to ponechávám ve formátu CSV.

Mobilní aplikace

Mobilní aplikace Home Assistant umožňuje uživatelům vzdáleně monitorovat a ovládat jejich inteligentní domácnost, poskytuje notifikace v reálném čase a nabízí přehledné uživatelské rozhraní pro snadnou správu zařízení a automatizací.



Obrázek 4.10: Mobilní aplikace Home Assistant Zdroj: [Autor]

4.10 Digitální váha do hnízda

Pro účely projektu bylo potřeba postavit digitální váhu. Tato váha má být umístěna v každém hnízdě v kurníku, kde slepice snáší vejce, a jejím úkolem je přes USB poskytovat aktuální data o tom, jakou hmotností je zatěžována podložka hnízda.

Popis algoritmu

Po připojení napájení k Arduinu, se nejdříve nastaví a připojí sériová komunikace rychlostí 9600 baudů pomocí USB. Následně je instancován objekt starající se o zprostředkování komunikace s převodníkem HX711 [27] a nastaví se jednotlivé piny dle fyzického zapojení převodníku. Dále je nastaven kalibrační faktor, což je konstanta používaná knihovnou HX711.h při převodu dat získaných z převodníku na gramy, s nimiž program již nadále pracuje a následně je posílá po sériovém připojení. Jako poslední krok v rámci přípravy programu je vynulování váhy pomocí metody tare(), která je jedna z metod HX711.h knihovny. Jakmile program dokončí fázi příprav, vstoupí do nekonečného cyklu. Tento cyklus pravidelně čte data přicházející po sériovém připojení. V případě, kdy jsou přijata data, program počítá se třemi scénáři. Buď je přijat znak "w", znak "t" nebo jakákoli jiná data. V případě znaku "w" (od slova weight) program reaguje odesláním aktuální naměřené hodnoty váhy po sériovém připojení jako reakce na příkaz. Přijatý znak "t" znamená tare neboli vynulovat váhu. Na základě toho se programu řekne, že aktuální hodnotu, kterou získal ze senzoru, má brát jako hmotnost 0 g.

Konstrukce váhy

Jako řídicí jednotka váhy je použito Arduino Nano [28], jež je zodpovědné za komunikaci s počítačem pomocí sériového připojení a načítání dat o hmot-

nosti z AD převodníku HX711 přes, který je Arduino připojeno k tenzometrickému [27] senzoru. Hmotnost je měřena tenzometrickým senzorem se jmenovitým zatížením 20 kg. Toto zapojení je nutné vzhledem ke konstrukci a vlastnostem senzoru. Tenzometrické senzory jsou analogové a jejich výstupem jsou velmi malé změny v napětí v rozmezí 0.15 až 1 mV. Tyto signály jsou navíc hodně slabé, proto je nutné použít konkrétně převodník HX711, který signál zesílí a zajistí větší rozlišení díky vysoké citlivosti. Složitější zapojení není jedinou nevýhodou. Značnou nevýhodou představuje také fakt, že už vlivem zatížení se profil senzoru mírně deformuje a na základě toho narůstá chyba měření v průběhu času. Tento problém se dá snadno řešit pravidelným nulováním v případech, kdy zrovna senzor zatížen není. Na konstrukci váhy je velmi zajímavý tvar a důvody samotné železné konstrukce. Protože má být váha instalována do již existujícího hnízda a slepice bude vlastně sedět rovnou na váze. Nakonec je jako platforma, na které slepice sedí, zvolena čtvercová miska o rozměrech odpovídající s rezervou rozměrům celého hnízda. Nemohla by to být například pouze rovná deska, protože slepice potřebuje pro pohodlné sezení podestýlku, kterou tvoří seno a v případě řešení s deskou by se seno dostalo mezi stěny hnízda a desku a váha by se tak nemohla volně pohybovat a zasekávala by se. Takto v misce je mnohem menší šance, že bude nějak větší množství sena vytlačeno a bude přidírat váhu (obrázek 4.11 a 4.12).



Obrázek 4.11: První prototyp váhy instalovaný v hnízdě Zdroj: [Autor]



Obrázek 4.12: Druhý prototyp váhy s logem Zdroj: [Autor]

Kalibrace

Po fyzickém sestavení váhy je třeba provést kalibraci. Kalibrace používaných senzorů je nutná, protože se díky ní eliminují rozdíly mezi senzory, překryjí se nedostatky konstrukce váhy a zajistí přesnost a korektnost měření pro konkrétní senzor. To znamená najít vhodnou hodnotu kalibračního faktoru, která se již za běhu váhy nebude měnit. Doporučená metoda, jak provádět kalibraci, je vzít si předmět, o němž víme přesně jeho hmotnost a ten použít jako vzorové závaží, na které chceme váhu kalibrovat. Konkrétně u mnou použitého senzoru do 20 kg je dobré volit závaží v řádech minimálně několika kil. Já jsem doma použil 5 balíčků polohrubé mouky. Hmotnost balení jsem si ověřil přesnou kuchyňskou váhou. Následujícím krokem je nalezení odpovídající hodnoty kalibračního faktoru pro náš senzor. Položil jsem známé závaží na váhu a zvyšováním nebo snižováním kalibračního faktoru, jsem sedoval, jestli se návratová hodnota na výstupu, při aktuálním faktoru přibližuje nebo vzdaluje k hodnotě hmotnosti závaží na váze.

Hardware

- Arduino Nano v3.0
- AD převodník HX711
- Tenzometrický senzor se jmenovitým zatížením do 20 kg
- Železná konstrukce váhy

Arduino knihovny

- Arduino.h - základní Arduino knihovna
- HX711.h - pro usnadnění komunikace s AD převodníkem HX711

4.11 Řídící jednotka

Řídící jednotka je vlastní sestavené zařízení, které zprostředkovává pomocí USB připojení data o teplotě a vlhkosti v kurníku. Další úlohou je ovládání dvířek a světla. Srdcem řídící jednotky je Arduino Nano. Tento programovatelný mikrokontroler slouží jako logický prvek mezi nízkoúrovňovými prvky (relé, senzory, táhlový motor) a počítačem. Na počítači zase běží služby ovládající nebo využívající data z těchto prvků.

Popis zapojení

Zapojení řídící jednotky je zdokumentováno na obrázku 4.13 a blokové schéma zapojení je na obrázku 4.14.

Základní komponenty řídící jednotky:

- mikrokontroler Arduino Nano v3.0
- senzor DHT11 pro teplotu a vlhkost
- relé s napětím cívky 5 V a spínacím proudem 10 A stejnosměrných

Pro snadnou práci s jednotlivými součástmi během vývoje, byly zvoleny moduly, které disponují svorkovnicemi a není tedy nutné k nim nic pájet. Připojení Arduina je realizováno pomocí terminal schieldu, který pro každý pin desky disponuje jednou svorkou pro připojení kabelu. Obdobně byl pro práci s relátkou vybrán modul, který již 4 rele obsahoval a pro snadné zapojení také svorkovnicí.

Nejkomplikovanější byl modul snímače, ke kterému bylo třeba vytvořit konektor, protože disponoval pouze výstupními piny. Jako instalační platformu jsem zvolil dřevotřískovou desku a na ni jsem připevnil všechny moduly pomocí krátkých vrutů do dřeva. Jednotlivé moduly byly následně propojeny vodiči o průřezu 0,15 mm² dle technické dokumentace dodávané ke každému

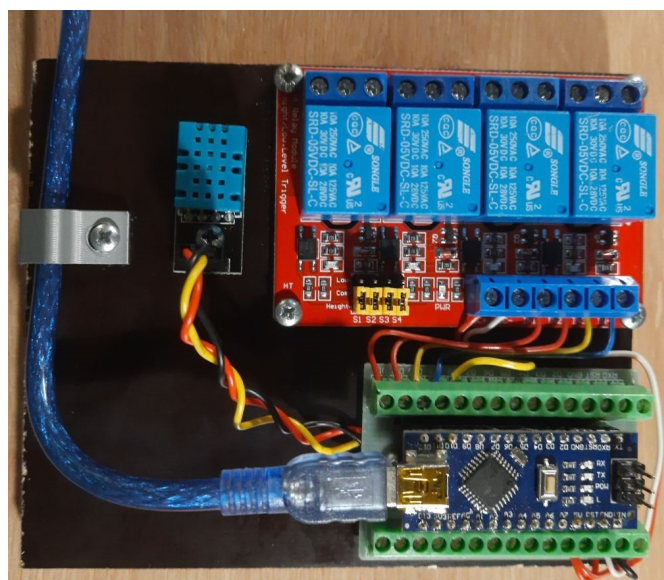
modulu. Hardwarová stránka byla tímto hotová.

Arduino a ostatní periferie se napájí skrze USB port Arduina, takže nebylo třeba řešit žádné přídavné napájení. Přídavné napájení by bylo třeba řešit například, kdybychom používali 10 relejových modulů současně, protože limit pro napájení skrze USB 2.0 je pouze 500 mA [29] a jedna cívka spotřebuje až 70 mA.

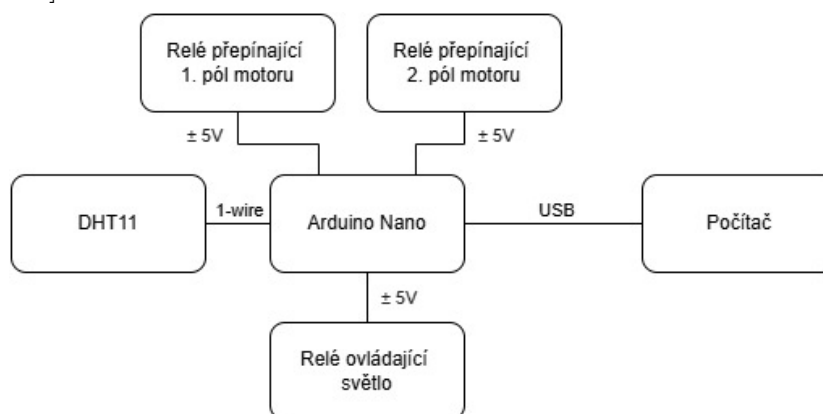
Následovala tedy poslední věc a to nahrát kód programu do samotné vývojové desky Arduino Nano. Zde je výborným pomocníkem vývojářský nástroj Arduino IDE [30], který celý složitý proces nahrávání vyřeší.

Popis algoritmu

V části programu Setup se jako první po startu programu nastaví příslušné módy na vstupní a výstupní piny. Následně se provede inicializace sériového připojení s rychlostí 9600 baudů. Poté se inicializuje připojení s čidlem DHT11 [31] pomocí knihovny DHT.h a na závěr jsou nastaveny výstupy dle aktuálních stavů řídicích neboli stavových proměnných pro dvířka a světlo. Dále program pokračuje do části Loop, což je metoda, která se volá pořád dokola jako nekonečný cyklus. Tato metoda řeší pravidelnou aktualizaci dat ze senzoru, dále čtení příkazů ze sériového připojení a posílání odpovědi na ně. Aktualizace senzorových dat se provádí jednou do vteřiny za pomoci metod readTemperature a readHumidity, které jsou součástí knihovny DHT.h. Zpracování příkazů probíhá zavoláním metody processCommand, která pokud je v zásobníku přijatých dat něco ke čtení, přečte první znak a ten následně zpracuje jako příkaz dle seznamu uvedeném v části 4.3.



Obrázek 4.13: Skutečné zapojení řídicí jednotky Zdroj: [Autor]



Obrázek 4.14: Blokové schéma řídicí jednotky Zdroj: [Autor]

4.12 Kamerový systém

Kamerový systém je jeden z hlavních prvků systému Coopmaster. Data z jeho kamer využívají skrze službu Camera Driver služby Dog Alarm a Chicken Watch Guard. Do systému jsou aktuálně instalovány dvě kamery. Jedna pro venkovní výběh, druhá pro pohled do kurníku. Po pečlivém výběru a volbě parametrů jsou vybrány kamery značky Hikvision typ Hilook IPC-B180HA-LU [32].

Výběr kamery

Klíčové vlastnosti pro výběr našich kamer byly

- Možnost venkovní instalace
- Připojení kabelem
- Volitelně oba způsoby napájení buď PoE nebo přímo ze sítě (záleží na způsobu instalace a umístění)
- Kvalitní noční vidění
- Velké zorné pole
- Vysoká kvalita obrazu (dostatek dat pro umělou inteligenci)

Snažil jsem se vybrat kameru, která bude relativně cenově dostupná a bude vyhovovat zmíněným parametrům. Musel jsem se kvůli tomu dovzdělat v mnoha ohledech, například detailně prozkoumat technologii PoE, digitálního videa, konstrukcí venkovních elektrických zařízení a mnoha dalších parametrů, které venkovní IP kamery mají. S výsledným výběrem jsem velmi spokojený. Kamera poskytuje kvalitní data pro vývoj tohoto systému a práce s ní není nijak komplikovaná, jak je detailněji popsáno v sekci o službě Camera Driver.

IP camera Hilook by Hikvision IPC-B180HA-LU

Klíčové parametry, které využíváme v projektu, jsou

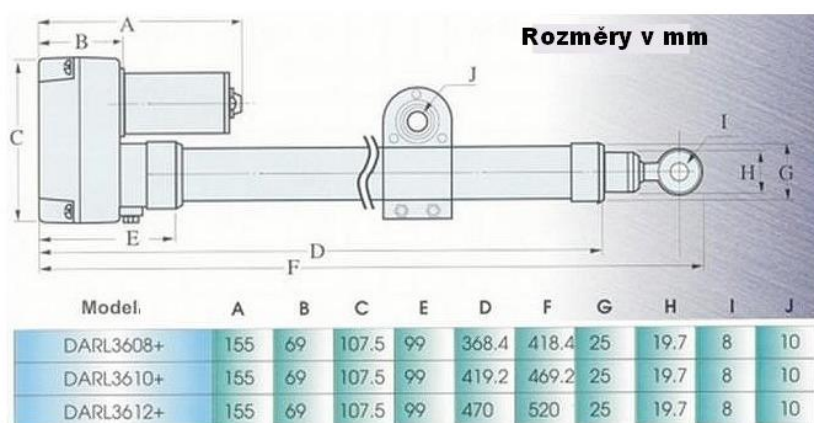
- Rozlišení 8 Mpx
- Přísvit pomoci bílou LED a IR LED s dosahem až 30 m
- Zajímavá technologie ColorVu pro snímání barev za horších světelných podmínek
- Technologie Motion Detection 2.0 pro detekci osob či vozidel
- Technologie WDR pro zmenšení rozdílů při různě silném osvětlení scény
- Napájení pomocí PoE nebo přes síťový adaptér na 12V 2A.
- REST API a HTTP nebo RSTP protokol pro poskytování obrázků, streamování videa a zvuku
- Dobře zpracované webové rozhraní pro konfiguraci

4.13 Konstrukce dvířek

Elektricky ovládaná dvířka systému Coopmaster plní hlavní požadavek, který na systém babička měla. Slouží k automatickému a bezpečnému otevírání vstupu do kurníku.

Aktuální verze dvířek je sestavena ze standardních hliníkových profilů a spojovacích komponent. Pro pohyb dvířek se pro tento prototyp jeví jako nejsnazší volba táhlový motor. Pracuje na napětí 12V stejnosměrných (popis v sekci 11.7). Je vybaven nastavitelnými dorazy, které umožňují nastavit rozmezí ve kterém je třeba, aby se výsuv pohyboval. Tělo motoru je připevněno k rámu dveří. Technický výkres motoru jsem připojil (obrazek 4.15). Pohyblivá část je volně připojena k vertikálně se pohybujícímu kovovému plátu, který slouží jako spouštěcí dveře.

První návrh počítal s pevným připojením mezi dveřmi a motorem, ale během testů se ukázalo, jak důležité je motor propojit s dvířky nějakým flexibilním spojem. Pokud by byl totiž plát připevněn pevně k motoru, mohla by se snadno stát z dvířek gilotina. Táhlový motor je schopen totiž vyvinout sílu až 200 kg (2000 N).



Obrázek 4.15: Rozměry táhlového motoru Zdroj: [ATOS spol. s r.o.]

Pokud by z nějakého důvodu zůstalo cokoliv v průchodu, mohlo by dojít ke zranění slepice nebo poškození dveřního mechanismu. Já jsem ve svém řešení použil silonový provázek.

Musel jsem se vypořádat s faktem, že motor má nedostatečný výsuv. Řešením byla instalace kladky, která zdvojnásobila zdvih dveří tak , aby slepice vůbec prolezla pod plátem.

Ačkoli má toto řešení řadu nevýhod, tak bylo realizováno jednoduše z toho důvodu, že jsem většinu součástí měl doma již z minulých projektů.

I kdyby se do budoucna vyřešily zmínované problémy, řešení by pořád nebylo produkční, protože hliníkové součásti společně s motorem jsou příliš drahé. Z ekonomického hlediska je jasné, že by si malý chovatel nechtěl do kurníku instalovat dvířka, která svou cenou převýší cenu vajec, které vyprodukuje za rok.

Nicméně projekt je aktuálně ve fázi vývoje, kdy řeším spíše softwarovou než hardwarovou stránku. Pro testovací účely je mechanismus aktuálních dvířek vyhovující. Výhodou hliníkových profilů je možnost snadných úprav a ladění konstrukce.

Na obrázku 4.16 můžete vidět aktuální podobu dvířek instalovaných u kurníku.



Obrázek 4.16: Instalovaný prototyp dvířek Zdroj: [Autor]

Myslím, si že pro finální instalaci budu muset zvážit úpravy konstrukce a pohonu. Hlinková konstrukce ze stavebnicového systému je drahá a věřím, že i samotný pohon půjde realizovat levněji.

5 Učení Yolo modelů

5.1 Detekce objektů pomocí strojového učení

Úvod do problematiky

V posledních letech došlo k významnému pokroku v oblasti počítačového vidění, což umožňuje efektivní a přesné rozpoznávání objektů v různých prostředích. Jednou z nejmodernějších knihoven, která si získala širokou popularitu, je YOLO11 [16] (You Only Look Once version 11). Tato knihovna přináší rychlé a přesné metody detekce objektů a je ideální pro implementaci v reálném čase. Má také předučené již 80 kategorií, které model umí v základní verzi detekovat. Tyto sekce mi můžeme využít a dodat do nich data z našeho prostředí, abychom je zpřesnili pro naše úlohy. Mezi těmito kategoriemi je také pes (dog) a pták (bird). V této sekci se zaměříme na to, jak využít YOLO11 ke specifickému úkolu: rozpoznávání slepic v kurníku. Automatizované rozpoznávání umožňuje sledování počtu slepic. Použití YOLO11 přináší následující výhody:

1. Rychlost a přesnost - YOLO11 je navrženo pro rychlou detekci objektů s vysokou mírou přesnosti, což je ideální pro aplikování v reálném čase na farmách.

2. Jednoduchost implementace - díky snadnému rozhraní a rozsáhlé dokumentaci je integrace YOLO11 do existujících systémů velmi přímočará.
3. Flexibilita - můžete trénovat model na vlastním datasetu slepic, což zajistí optimální rozpoznávání i ve specifických podmínkách vašeho kurníku.

V následujících sekcích si podrobně projdeme kroky, jak připravit dataset, trénovat vlastní model se zaměřením na rozpoznávání slepic a jak jej implementovat do našeho systému monitorování.

Obrázkový dataset

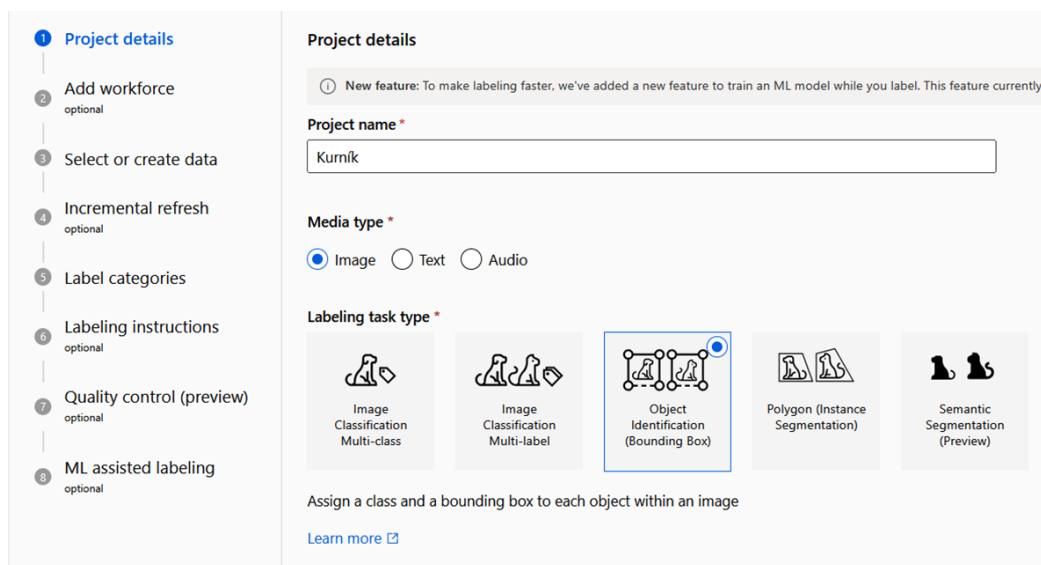
Obrázkový dataset je strukturovaná kolekce obrazových dat, která se používá pro potřeby strojového učení a počítačového vidění. Tyto datasety obsahují jednotlivé obrázky, které jsou často spojeny s dodatečnými informacemi nebo anotacemi, jež jsou potřebné pro trénink modelů. Níže jsou uvedeny hlavní aspekty, které charakterizují obrázkové datasety:

1. Obrázky - snímky, které mohou být v různých formátech (např. JPEG, PNG). Rozlišení a počet barev se může lišit v závislosti na účelu datasetu
2. Anotace - metadata obsahující následující elementy
 - Klasifikační štítky - identifikují, co se na obrázku nachází (např. „kočka“, „pes“, „slepice“)
 - Ohraničující boxy - polohu specifických objektů na obrázku

Datasety jsou klíčové pro vývoj a zlepšování modelů strojového učení, protože poskytují potřebné tréninkové a testovací údaje.

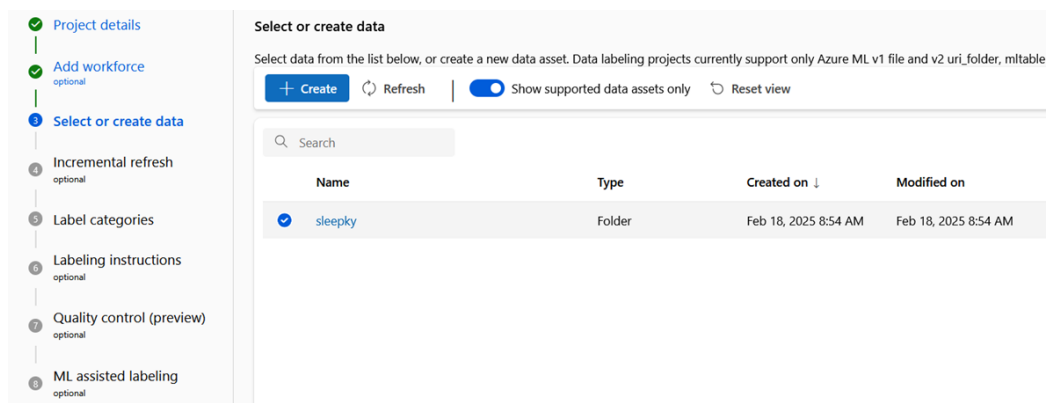
V kontextu využití datasetů pro strojové učení je důležité zajistit, aby datasety byly kvalitní, rozmanité a dostatečně rozsáhlé, což pomáhá modelům převážně se zobecněním a výkonem v různorodých situacích. Pro specifické nasazení v babiččině kurníku jsem připravil vlastní dataset, který jsem použil pro doučení

již existujícího základního modelu, který poskytuje Yolo11. Záběry jsem vyfotil mobilem v prostředí babiččina výběhu a kurníku. Existuje mnoho systémů pro tvorbu anotací (image labeling). Já jsem zvolil Azure AI - Azure Machine Learning studio [33]. Pro práci je velmi intuitivní a nechá se spustit v několika krocích. Nejdříve jsem pomocí průvodce vytvořil nový projekt jak je znázorněno na obrázku 5.1.



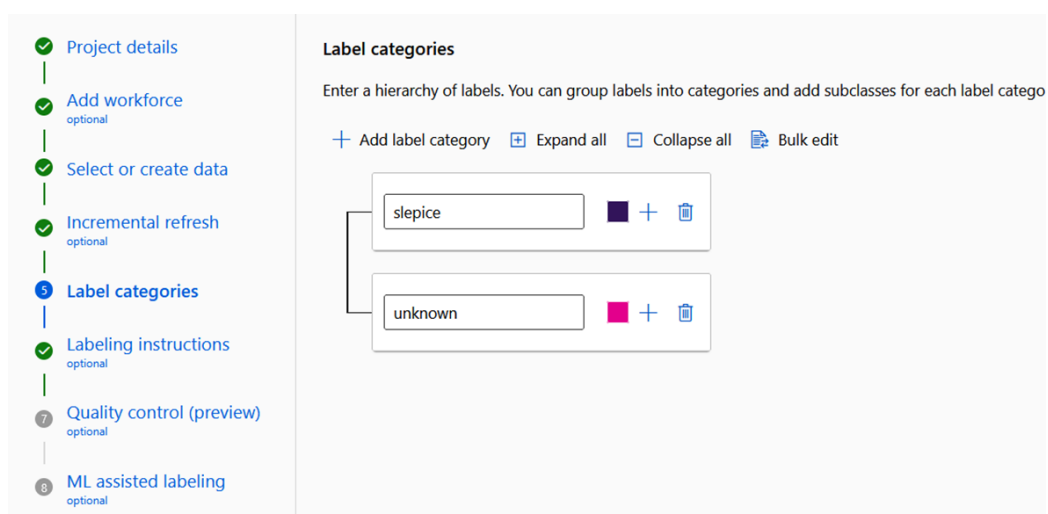
Obrázek 5.1: Vytvoření nového projektu v AML Zdroj: [Autor]

Dále jsem vybral zdroj obrázků, které bylo potřeba klasifikovat. Vzhledem k tomu, že se jedná o cloudovou službu bylo třeba obrázky slepiček na Azure naimportovat. Na obrázku 5.2 je snímek obrazovky z Azure Machine Learning studia, kde se zrovna dataset importuje.



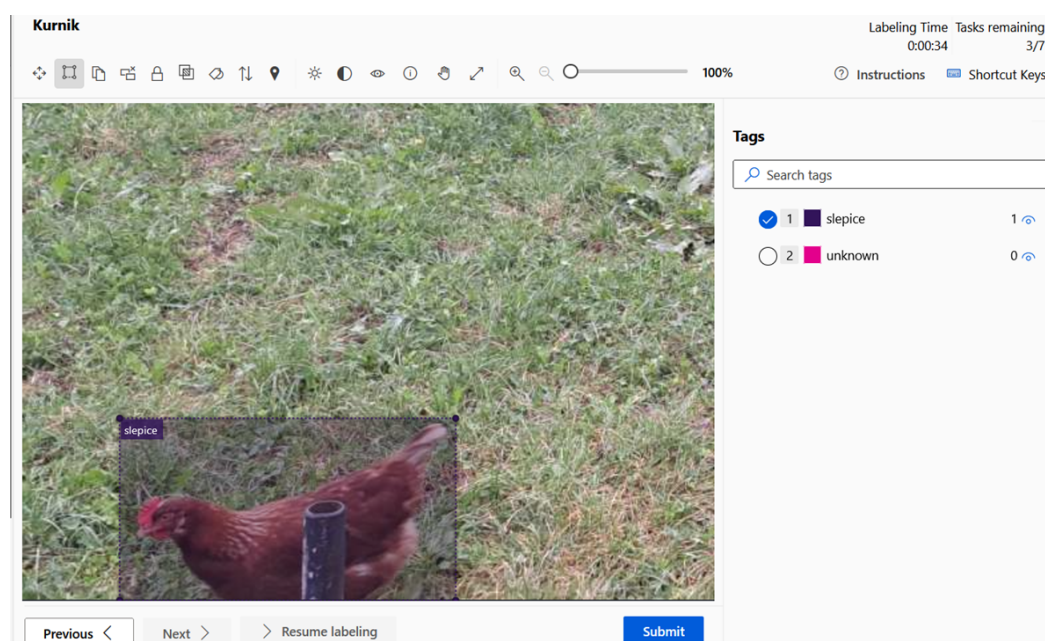
Obrázek 5.2: Tvorba datasetu v AML Zdroj: [Autor]

V dalším kroku jsem nadefinoval kategorie, které budu chtít v obraze identifikovat. V mém případě jsem zvolil label „slepice“. Definice kategorií je na obrázku 5.3.



Obrázek 5.3: Definice kategorií pro učení v AML Zdroj: [Autor]

V následujícím kroku následovala nejméně zábavná činnost. Bylo třeba projít jednotlivé snímky a ručně vybrat oblasti, kde já jako člověk vidím slepici. Oblast, kterou jsem označil se pak následně ukládá jako metadata k obrázku. Tento proces je znázorněn na obrázku 5.4.



Obrázek 5.4: Označování slepice pomocí AML Zdroj: [Autor]

Když se podařilo projít celý obrázkový dataset, mohl jsem vyexportovat „labeling“ data (obrázek 5.5). Jak jsem psal výše, mít kvalitní učící dataset je důležité, proto jsem exportoval pouze data, která se podařilo úspěšně označit. Do fotogalerie se mi dostalo i několik nejasných snímků, případně těch, kde slepice nebyla vůbec. Takové snímky jsem vyřadil. Anotace jsem exportoval ve formátu COCO [34] 11.22. Formát COCO (Common Objects in Context) je standardní formát pro anotaci obrázků používaný ve strojovém učení a počítačovém vidění. Jeho hlavním cílem je poskytnout strukturu pro uložení informací o objektech v obrázcích, což umožňuje efektivní trénink a hodnocení modelů detekce objektů, segmentace a dalších úloh počítačového vidění. Obrázek 5.1 ukazuje, jak například může vypadat nastavení pro export COCO formátu z AML.

Export data

×

Asset type

Labeled

Export format:

COCO file

Coordinate type:

Normalized

Export File Name

Export File Name

Include these additional details in the export output

☐ Labeler details (labeler name, approver name, and labeling time)

☐ Only approved

Obrázek 5.5: Export metadat k obrázkům z AML do COCO formátu
Zdroj: [Autor]

Soubor obsahuje několik klíčových sekcí: images, annotations, a categories. Zde je popis jednotlivých částí daného JSON souboru ve formátu COCO:

1. images: Tato sekce obsahuje informace o každém obrázku v datasetu
 - id - unikátní identifikátor obrázku (zde 1)
 - width a height - rozměry obrázku (zde 534 pixelů x 291 pixelů)
 - file_name: cesta k souboru obrázku včetně názvu souboru
 - coco_url, absolute_url - adresy, kde je obrázek uložen, ať už v cloudovém sandboxu (coco_url) nebo přes URL pro přímý přístup (absolute_url)
 - date_captured - čas a datum zachycení obrázku v UTC formátu
2. annotations - poznámky k jednotlivým objektům na obrázcích

- id - unikátní identifikátor anotace (zde 1)
- category_id - odkazuje na kategorii objektu
- image_id - ID obrázku, na který se tato anotace vztahuje
- area - relativní plocha objektu na obrázku (zde 0.016) Tato hodnota je často normalizována vzhledem k velikosti obrázku
- bbox - relativní souřadnice (bounding box)

3. categories - seznam kategorií, do kterých mohou objekty na obrázcích patřit

- id - unikátní identifikátor kategorie
- name - jméno nebo popis kategorie (zde "slepice" a "unknown")

Tento JSON tedy popisuje dataset obsahující obrázek s jedním označeným objektem, který patří do kategorie "slepice". COCO formát umožňuje uložení komplexních a strukturovaných dat o objektech na obrázcích, což je velmi užitečné pro trénování a testování modelů strojového učení. Příklad tohoto formátu metadat pro strojové učení je na obrázku 5.6.

```
{
  "images": [
    {
      "id": 1,
      "width": 534,
      "height": 291,
      "file_name": "UI/2025-02-18_075342 UTC/segmentace/sleepka-466.png",
      "coco_url": "Am1Datastore://workspaceblobstore/UI/2025-02-18_075342 UTC/segmentace/sleepka-466.png",
      "absolute_url": "https://kurnik3204699652.blob.core.windows.net/azureml-blobstore-55fc1b39-2e18-45fe-af76-9e1d9df7d80a/UI/2025-02-18_075342 UTC/segmentace/sleepka-466.png",
      "date_captured": "2025-02-18T08:00:02.9684730Z"
    }
  ],
  "annotations": [
    {
      "id": 1,
      "category_id": 1,
      "image_id": 1,
      "area": 0.016,
      "bbox": [
        0.5061702437590774,
        0.07131312514796402,
        0.0838699330811750,
        0.1898989775606553
      ]
    }
  ],
  "categories": [
    {
      "id": 1,
      "name": "slepice"
    },
    {
      "id": 2,
      "name": "unknown"
    }
  ]
}
```

Obrázek 5.6: Ukázka COCO metadat pro učení Zdroj: [Autor]

Doučování modelu Yolo11

V předchozí sekci jsem si popsali, jak vytvořit zdroje dat a nyní jsme připraveni využit existující model. Doučovat doma existující model je relativně složité, ale díky výborné dokumentaci na stránkách Ultralytics.com [16] jsem to zvládnul. Základem pro úspěšné a efektivní učení je hardware. Hlavně GPU, protože trénování modelů hlubokého učení je výpočetně náročné. Vypozoroval jsem, že největší vliv na učení má velikost paměti grafické karty. V mém případě byla použita Nvidia RTX 4070. O stavbě a následném učení neuronových sítí by se nechalo napsat desítky stránek, ale to není předmětem mé práce. Rád bych alespoň představil blokové schéma učícího procesu, který jsem použil.

Pro učení jsem použil script, který jsem si připravil. Volání jednotlivých kroků pro učení je znázorněno na screenshotu učící metody na obrázku 5.7

```
13
14 learn_config = SLEPICKY_LEARN_CONFIG
15
16 if __name__ == '__main__':
17
18     CleanWorkspace(learn_config).clean_workspace()
19     GroupCreator(learn_config).create_groups()
20     DatasetSplitter(learn_config).process()
21     TrainYolo(learn_config).learn()
22     YOLOTester(learn_config).process()
23
24
```

Obrázek 5.7: Kód pro učení nového modelu Zdroj: [Autor]

Krok 1 - CleanWorkspace

Tento krok je zodpovědný za přípravu pracovního prostředí před zahájením tréninkového procesu. Odstraňuji v něm archivaci stávajících dat a modelů, které mi ve workspace zůstaly z předchozích běhů a již nejsou potřebné. Čištění pomáhá zabránit kolizím s předchozími seancemi a udržet v něm

pořádek, což zajišťuje, že nové modely a datové soubory jsou aktuální a správně organizované.

Krok 2 - GroupCreator

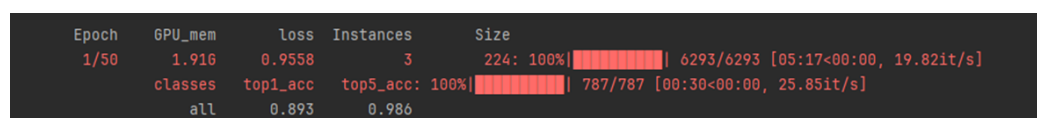
Tento krok se zaměřuje na organizaci dat do různých skupin pro trénování a testování modelu. Data mám rozdělena v několika různých adresářích a tento krok má na starost jejich správné složení. Stejná data používám pro výuku modelu na počítání slepic, detekce vetřelce a plánované reidentifikace slepic.

Krok 3 - DatasetSplitter

Proces rozdělení datasetu je klíčový pro oddělení trénovací, validační a testovací sady. Rozdělení může být prováděno v daných poměrech, například 70% trénovací data, 15% validační data a 15% testovací data. Správné rozdělení dat je důležité pro zajištění objektivního hodnocení modelu, aby se předešlo přeskolení. Tento krok také zajišťuje, že žádné datové soubory nejsou opomítnuty a že rozdělení je náhodné a reprezentativní.

Krok 4 - TrainYolo

Jedná se o hlavní krok, kde je model YOLO11 trénován na trénovacích datech. Proces zahrnuje několik iterací (epoch) přes trénovací data, během kterých model aktualizuje své váhy podle chybného odhadu. Na obrázku 5.8.



Epoch	GPU_mem	loss	Instances	Size
1/50	1.916	0.9558	3	224: 100%
classes	top1_acc	top5_acc: 100%		6293/6293 [05:17<00:00, 19.82it/s]
all	0.893	0.986		787/787 [00:30<00:00, 25.85it/s]

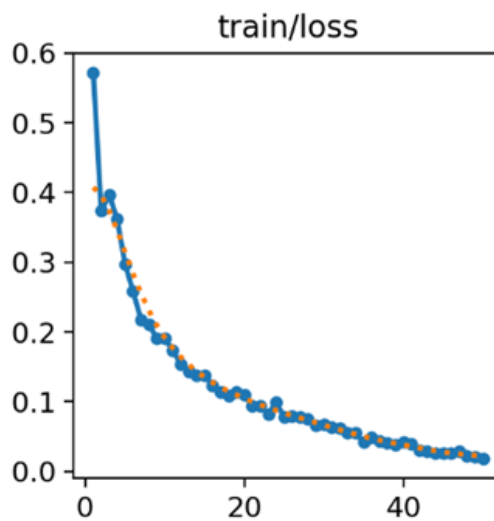
Obrázek 5.8: Proces trénování YOLO11 modelu Zdroj: [Autor]

Během trénování se sleduje výkonnost modelu na validačních datech, která zabraňují přeskolení. Přeskolení si můžeme představit například u modelu, který rozpoznává kačeny, kočky a slepice. I když všechny konfigurační parametry byly správně, systém v některých případech špatně identifikoval kočku.

Až zpětnou analýzou trénovacích dat bylo zjištěno, že došlo k přeškolení. Systém identifikoval kachnu nikoliv podle tvaru jejího těla, případně zabarvení, jako to udělá člověk, ale podle toho, zda na obrázku byla či nebyla tráva. To pak vedlo k tomu, že pokud byla kočka na trávě, byla označena za kachnu. Spustit trénovací smyčku je jednoduché. Bylo třeba předat jen několik základních parametrů.

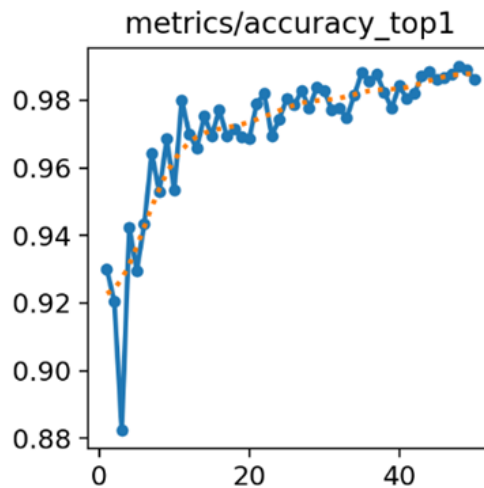
- Epochs: 50 - znamená, kolik poběží učících cyklů
- Batch size: 16 - specifikuje, kolik vláken poběží paralelně, je třeba dbát na velikost paměti grafické karty
- Pretrained: True - říká, že budeme rozšiřovat již naučený model
- Data path: c:/kurnik/klasifikace/kurnik_learn_temp/train - specifikuje cestu odkud se načítají data k učení

Krok 5 - YOLOTester: Po úspěšném tréninku modelu je nutné jeho testování na dříve oddělené testovací sadě. Tento krok zahrnuje vyhodnocení výkonnosti modelu pomocí metrik, jako je přesnost, citlivost. Testování nám poskytuje objektivní zpětnou vazbu o tom, co za objekty model umí ve snímcích rozpoznat. Důležité je předkládat pro testování snímky, které během tréninku neviděl. Na základě testovacích výsledků jsem pak následně prováděl úpravy učících parametrů, případně jsem dodával další data do učícího datasetu.



Obrázek 5.9: Graf loss funkce v závislosti na počtu etap Zdroj: [Autor]

Graf na obrázku 5.9, který se při učení automaticky generuje, nám umožňuje sledovat, jak dobře se model učí vzhledem k časové ose tréninku. Tento graf typicky zobrazuje ztrátovou funkci (loss) na svislé ose (Y) a počet trénovacích epoch nebo kroků na vodorovné ose (X).



Obrázek 5.10: Top-1 accuracy chart Zdroj: [Autor]

Graf na obrázku 5.10 se zaměřuje na metriku přesnosti, konkrétně na "top-1

accuracy", která měří, jak často je nejpravděpodobnější (nejvyšší hodnocená) předpověď modelu správná.

Výsledky detekce jsou demonstrovány na obrázku 5.11.

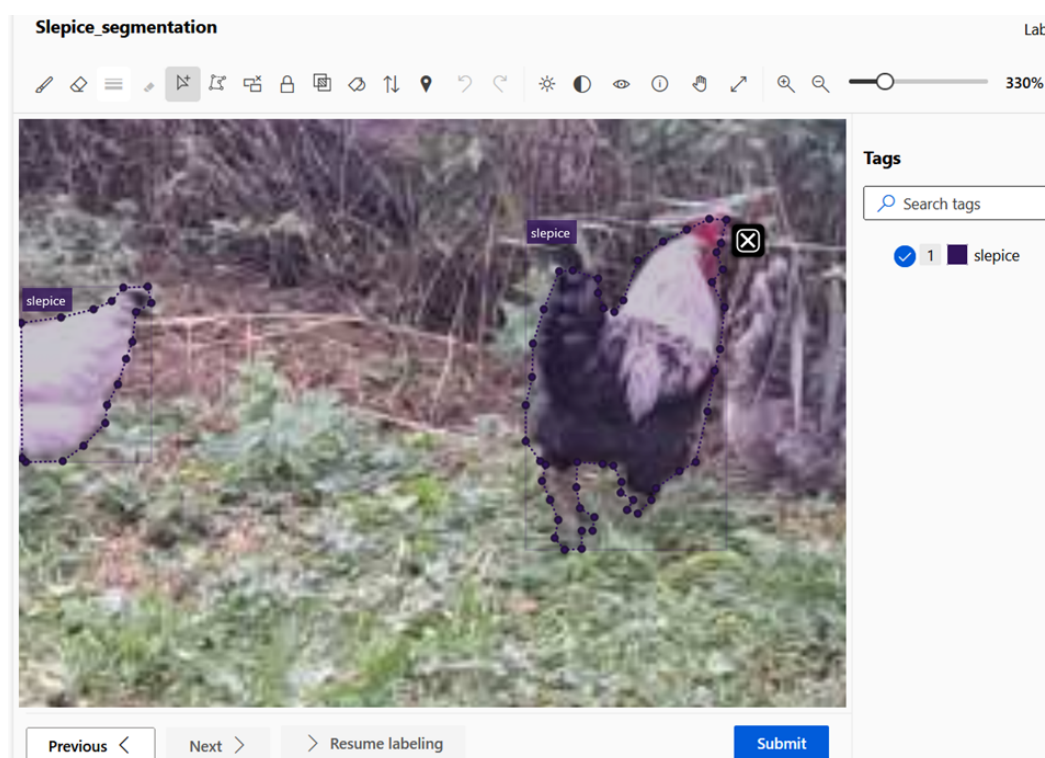


Obrázek 5.11: Model detekující slepice Zdroj: [Autor]

5.2 Segmentace objektů pomocí strojového učení

Image segmentation [35] je technika v počítačovém vidění, která má za cíl rozdělit obrázek na několik segmentů nebo oblastí pro jednodušší analýzu. Cílem image segmentation je identifikovat a rozlišit různé objekty nebo části obrázku, což může být užitečné pro různé aplikace, jako je rozpoznávání objektů, detekce hranic, klasifikace a další analytické úlohy. V mém případě jsem chtěl využít tuto techniku k tomu, abych identifikoval, která slepice snesla vajíčko, abych tak získal statistiku o tom, v jaké kondici slepice jsou a jak dobře snáší. Existuje několik možností, jak slepici identifikovat, já jsem se rozhodl pro rozpoznání slepice z obrazu.

Nejdříve jsem začal pracovat na získání a přípravě dat. Shromáždil jsem dataset snímků slepic z kurníku, který pokrývá různé úhly pohledu, osvětlení a pozice. Dataset jsem anotoval. V tomto případě byla tvorba anotačního souboru významně složitější, protože bylo třeba nejenom identifikovat, že slepice je na obraze, ale identifikovat její konkrétní tvar, případně jenom část těla. Znamenalo to tedy ručně projít jednotlivé snímky a myší vyklikat potřebné oblasti, což je znázorněno na obrázku 5.12.



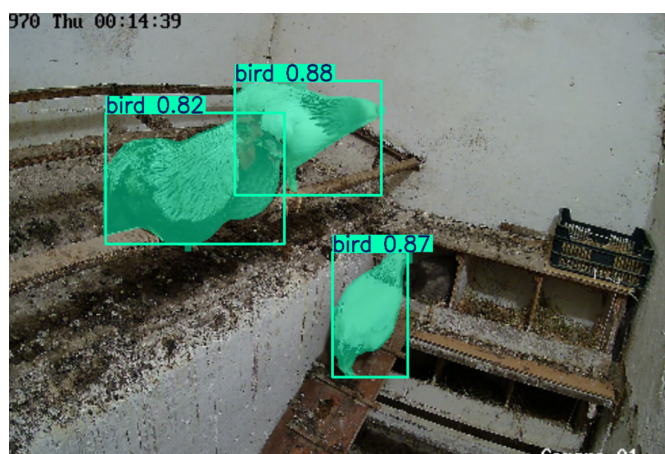
Obrázek 5.12: Proces anotace pro trénování segmentace Zdroj: [Autor]

Obecná zásada říká, že platí přímá úměra mezi množstvím trénovacích dat a kvalitou modelu. Toto jsem mohl vyřešit větším počtem obrázků v testovacím datasetu nebo použít techniku augmentace dat. To je postup, který jsem zvolil já. Na obrázky jsem aplikoval řadu transformací, které přispěly k různorodosti dat.

- Rotace - otáčení obrázků o náhodně vybrané úhly, aby se model stal invariantním vůči natočení objektů
- Škálování - změny velikosti obrázků buď zmenšením nebo zvětšením, aniž by se změnil poměr stran, což pomáhá modelu lépe zvládat objekty různých velikostí.
- Oříznutí (cropping) - vyříznutí náhodných částí obrázku.
- To pomáhá modelu naučit se, že objekty mohou být částečně oříznuty.

- Překlopení (flipping) - horizontální nebo vertikální zrcadlení obrázku pro zvýšení variability dat.
- Změna jasu, kontrastu a saturace - úprava těchto parametrů dělá model odolnějším vůči různým světelným podmínkám.
- Přidání šumu - přidání náhodného šumu do obrázků může pomoci modelu být méně citlivý na zašumění v datech.
- Posun (translation) - posunutí obrázku ve vodorovném nebo svislém směru, které pomáhá modelu rozpoznávat objekty při různých umístěních.
- Random Erasing - náhodné vymazání malých částí obrázku, které může pomoci vylepšit model proti částečnému zakrývání objektů.
- MixUp - směšování dvou obrázků a jejich odpovídajících anotací ke generování nových tréninkových vzorů.
- Mosaic Augmentation - kombinování čtyř různých obrázků do jednoho, což rozšiřuje kontext a zvyšuje variabilitu scén.

Následovalo trénování segmentačního modelu. Řešení jsem postavil na YOLO11 z oficiálního repozitáře. Díky knihovně z projektu Ultralytics je volání přímočaré. Na vstup jsem dal cestu k obrázkovému datasetu a anotačnímu souboru v COCO formátu [34]. Další parametry jsem pro výuku ponechal defaultní. Počet epoch jsem ponechal na 50. Již od 40. epochy model dosahoval přijatelné úrovně přesnosti při rozpoznávání a segmentaci slepic. Výsledek segmentace je vidět na obrázku 10.2.



Obrázek 5.13: Výsledek detekce slepic 1 Zdroj: [Autor]



Obrázek 5.14: Výsledek detekce slepic 2 Zdroj: [Autor]

6 Tvorba programu pro Arduino

Proces tvorby programu a jeho následného nahrání do jedné z vývojových desek Arduino zahrnuje několik kroků [36].

Já jsem pro psaní kódu použil vývojářské prostředí Arduino IDE. Tento nástroj umožňuje psát programy v lehce upraveném jazyce C++ (11.6) pro desky Arduino. Takto upravený jazyk se dnes moderně nazývá Arduino jazyk a vychází z dřívějšího projektu Wiring 11.5. Arduino IDE je velmi dobré prostředí pro začátečníky. Disponuje základními funkcemi jako zvýraznění syntaxe nebo automatické doplňování pro snadnější psaní. Prostředí zároveň zaobaluje celý proces kompilace a nahrávání programu.

Postup nahrání programu do Arduino desky

Jakmile máme dopsaný a uložený kód, je třeba ho ověřit pomocí funkce Verify (ověřit), což zkontroluje syntaktické chyby. Tato funkce následně provede kompilaci programu společně s jeho knihovnami do jednoho binárního souboru. Pokud je kód bez chyb a zkompileovaný, připojíme Arduino k počítači pomocí USB kabelu, který zajišťuje napájení i datový přenos. Po připojení systém rozpozná zařízení a přiřadí mu sériový port. V IDE je třeba vybrat správný typ desky a sériový port, aby program věděl, s jakým zařízením má komunikovat.

Následně nahrajeme program do Arduino desky funkcí Upload (nahrát). Arduino IDE komunikuje s bootloaderem na Arduino přes sériový port. Boot-

loader je malý program v paměti mikrokontroléru, který umožňuje nahrávat nový kód bez externího programátoru. Přijímá binární soubor a zapisuje jej do flash paměti mikrokontroléru.

Po úspěšném nahrání předá bootloader řízení novému programu, který se začne automaticky provádět. Program běží v nekonečné smyčce, dokud není zařízení resetováno nebo vypnuto. Pro ladění a interakci s programem v reálném čase nabízí Arduino IDE nástroj Serial Monitor, který umožňuje sledovat data z Arduina a posílat mu příkazy.

Celý proces je díky funkcím IDE a vestavěnému bootloaderu uživatelsky přívětivý a nevyžaduje pokročilé znalosti programování mikrokontrolérů. Umožňuje tak široké škále uživatelů snadno vytvářet programy pro platformu Arduino.

7 Testování systému Coopmaster

Testování [37] je klíčovým krokem k zajištění spolehlivosti, správnosti a bezpečnosti aplikací a systémů. Jako programátor jsem se zaměřil na několik úrovní testování, které mi zajistily, že všechny komponenty systému fungují podle očekávání. Na svůj projekt jsem ručně aplikoval integrační testy, systémové, zátěžové a na závěr uživatelské testy. Unit testy, které jsou jinak základem každého správně vyvíjeného softwaru, jsem aplikoval hned při vývoji funkcionality.

7.1 Integrační testy

Ručně jsem prováděl integrační testy. Procházel jsem je vždy po dokončení služby, abych zjistil zda všechna komunikační rozhraní fungují správně. K ověření funkčnosti Flask API rozhraní jsem používal program Postman [38]. Pro ověření komunikace skrze MQTT jsem použil program MQTT-Explorer [39].

7.2 Systémové a zátěžové testy

Dále jsem pouštěl opakovaně pouštěl systémové testy. Ukazovaly totiž funkčnost celého systému a během času ověřovali funkčnosti

v různých situacích. Díky těmto testům jsem například odhalil špatnou konfiguraci MQTT brokeru, který byl nastaven, aby ukládal všechny zprávy, které skrze něj projdou.

Další chyba kterou testy odhalily byla "Access denied při restartu". Chyba s restarty byla při nastavování linuxových práv k různým periferiím a souborům. Problém nastal v případě, kdy bylo potřeba udělit práva pro využití USB portu službě Scale Driver. Práva se totiž nastavila na konkrétní port, jenže po restartu systému se porty zpřeházely a služba k novému portu neměla přístup.

Hned od začátku vývoje aplikace jsem si byl vědom, že systém poběží neustále. Proto jsem kladl velký důraz na jeho robustnost. Od prvních momentů snažil, abych měl aplikaci permanentně nasazenou na testovacím prostředí.

7.3 Uživatelské testování

Uživatelské testování jsem prováděl kvůli tomu, abych zjistil, zda je grafické rozhraní v systému Home Assistant dostatečně srozumitelné a přehledné. Díky těmto testům jsem například opravil nelogičnost v ikonkách tlačítek nebo jsem změnil rozvržení komponent podle toho, v jakém pořadí je chce uživatel na první pohled vidět.

8 Produkce

V této kapitole si popíšeme, jaký je vlastně aktuální stav běžícího systému, a vysvětlíme si, co je potřeba pro přidání nového modulu.

Verzování a nasazení

Pro verzování zdrojových kódů svých služeb jsem se rozhodl využívat GitHub [40], což mi umožňuje efektivně zálohovat kód. V rámci GitHubu používám GitHub Workflow, které automatizuje proces vytváření Docker image [41], což zrychluje a automatizuje proces nasazení nové verze služby. Jakmile je Docker image vytvořen, nahrávám jej na platformu Docker Hub[42]. Na tuto platformu image ukládám a následně je vzdáleně načítám při nasazení konkrétní služby.

Infrastrukturní přípravné práce

Vlastní instalaci částí systému Coopmaster do kurníku a výběhu předcházely přípravné stavební práce. Do budovy bylo třeba přivést Ethernetové připojení, což nebyl vůbec snadný úkol. Dále pak bylo potřeba připravit a nainstalovat rozvaděč s řídicí elektronikou (obrázky 8.1 8.2). Musel jsem také na příslušných místech udělat průrazy zdí, v místech kde povedou kabely z rozvaděče ke kamerám a do kurníku. Samotné Ethernetové připojení jsem

vyřešil poněkud nestandardně. Nepřicházelo v úvahu, abych táhl metalický kabel od domácího routeru do chléva, kvůli přílišné náročnosti na stavební úpravy. Použil jsem tedy Wifi extender s Ethernet výstupem a elektroniku v kurníku jsem k internetu připojil přes něj.



Obrázek 8.1: Nainstalovaný rozvaděč ve chlévě Zdroj: [Autor]



Obrázek 8.2: Vnitřek rozvaděče Zdroj: [Autor]

Výsledná fyzická infrastruktura

Doma v místnosti se stabilním prostředím, které je vhodné pro výpočetní techniku, běží Intel NUC 11 [43] s operačním systémem Ubuntu 22.04 LTS. Tento počítač slouží jako hlavní server a zdroj výpočetního výkonu pro

neuronové sítě. Výpočetní výkon je poskytnut díky grafické kartě NVIDIA GeForce RTX 2060.

Dále pak máme v kurníku a ve výběhu periferie (dvířka, senzory a kamery), které potřebují propojit se systémem Coopmaster. Všechny tyto periferie jsou do systému připojeny skrze mikropočítač Raspberry Pi 5 [44] s operačním systémem Ubuntu 24.04 LTS. Výpočetní část je takto rozdělena do dvou částí kvůli agresivnímu prostředí v kurníku (vlhkost, teplotní výkyvy). Raspberry Pi 5 je mnohem levnější než Intel NUC 11 a v případě poškození zařízení vlivem prostředí, nevznikne tak velká škoda. Síťová komunikace je mezi těmito dvěma počítači realizována pomocí služby Tailscale [45], která mezi nimi vytváří mesh VPN. Zároveň této služby využívám já jako správce, abych měl vzdálený přístup k oběma zařízením.

Základní konfigurace a klíčové služby

Systém je rozvržen tak, že na počítači v kurníku běží služby, které zajišťují komunikaci mezi hardwarovými periferiemi a dalšími službami systému Coopmaster. Doma na počítači jsou služby, jež potřebují výpočetní výkon a řeší konkrétní body zadání. Na obou počítačích musí tedy být nainstalovaný Docker společně s nástrojem Docker Compose, aby bylo možné služby hromadně spouštět a konfigurovat. Parametry jednotlivých služeb se načítají v obou případech ze souboru `.env`, aby konfigurační soubor Docker Compose neztrácel na přehlednosti.

Základní služby, které jsou nutným minimem pro fungování systému, jsou

- Home Assistant (11.10)
- Postgres 15 (11.19)
- Eclipse Mosquitto (11.20)
- Health Checker (4.4)

- Cloudflared (11.21)

Tyto služby tvoří běhové prostředí, do kterého jsou již pouze zasazovány jednotlivé moduly systému Coopmaster.

Procedura přidání nového modulu

Nyní si ukážeme celý proces, jak přidat do systému Coopmaster nový modul. Ukázkou provedeme na modulu pro detekci vetřelců. Máme tedy dvě služby Camera Driver a Dog Alarm, tedy jedna služba pro poskytování dat a druhá pro výpočty.

Například tento modul využívá data z IP kamery ve výběhu, kterou je zde potřeba nainstalovat, nakonfigurovat a připojit na Ethernet.

Následně je třeba do Docker Compose konfigurace v kurníku přidat službu, která bude data z kamery předávat do útrob systému Coopmaster. Této službě je třeba nastavit parametry na základě nichž bude fungovat. V aktuální situaci je tato služba Camera Driver a vyžaduje IP adresu a přihlašovací údaje k IP kameře.

Jakmile připojíme novou periférii pomocí driver služby do systému, potřebujeme službu, která data zpracuje. Tato služba se musí přidat do Docker Compose konfigurace běžící doma na počítači Intel NUC 11 a opět jí musíme nastavit požadované parametry. Každá služba z výpočetní sekce vyžaduje minimálně přihlašovací údaje a IP adresu MQTT brokeru. V tomto případě se jedná o službu Dog Alarm, která se stará o detekci vetřelců na záběrech kamery. Jako parametry tato služba vyžaduje zmíněné údaje o MQTT brokeru a dále pak IP adresu a port služby, ze které má stahovat obrázky k analýze.

Služby z výpočetní části posílají výsledky své činnosti skrze MQTT broker

do systému Home Assistant, aby je zobrazil. Aby Home Assistant věděl, jak má s daty pracovat, je třeba ho nakonfigurovat pomocí souboru `configuration.yaml`. Do tohoto souboru se definují entity, které představují jednotlivé vstupní a výstupní prvky. Zde se jedná o několik MQTT entit. Každé definujeme minimálně názvem, topicem a datovým formátem.

Na závěr je třeba pomocí webového rozhraní systému Home Assistant přidat karty a automatizace, které budou vizualizovat a reagovat na data přijatá do entit. Pro tento modul je potřeba hned několik karet, a to kartu pro vizualizaci fotografií z kamery, kartu pro upozornění chovatele na hrozbu. Je třeba také přidat automatizaci, která odešle chovateli notifikaci o hrozícím nebezpečí.

Pro shrnutí můžeme říci, že jsme si popsali kompletní proces přípravy, přidání, konfigurace a nasazení nového modulu do ekosystému Coopmaster.

9 Ekonomická stránka projektu

Projekt Coopmaster je rozhodně revoluční řešení v chovatelské branži. Každopádně se rozhodně nehodí jeho využití jako odpověď na snížení ceny za provoz malých domácích chovů. Hlavním důvodem je už samotná pořizovací cena komponent, která je u mého prototypového řešení okolo 20 tisíc korun. Je tedy rozhodně zbytečné instalovat systém do malého domácího kurníku, když samotná pořizovací cena předčí cenu vajec vyprodukovaných za 3 roky, a to se ještě nebavím o licencích za software a energii potřebné pro běh systému.

Potenciál tohoto nápadu by se dle mého dal určitě zúročit ve větších podnicích, kde bude možno naplno využít schopností celého řešení. Například počítání by určitě pomohlo například na pastvách s ovci či kravami, kde jsou počty výrazně větší a je již složité a časově náročné ručně spočítat všechny kusy. Dále by se potenciál kamer a neuronových sítí dal využít k detekci nemocných nebo zraněných zvířat. Toto jsou faktory, které by rozhodně stálo za to vyzkoušet zlepšit. Například pokud by systém správně s předstihem detekoval nějakou nemoc na jednom jedinci ve stádě krav, mohlo by to pomoci dříve zareagovat a zachránit zbytek stáda před uhynutím. V takovém případě by se právě systém velice vyplatil, protože by vlastně zachránil firmu před krachem.

Dále je určitě vhodné zmínit možnost automatizace krmení a čištění sítel zvířat. To je věc, která se dá plně automatizovat a ušetřit tak čas a případně i pracovní sílu, která by jinak tyto činnosti konala.

Na závěr tedy můžeme říct, že myšlenka tohoto systému se nevyplatí pro malé chovatele, pokud to nejsou techničtí nadšenci jako já. Své uplatnění, ale najde u větších firem, kterým může v daných situacích i zachránit byznys.

10 Rozvržení práce do budoucna

V tomto projektu vidím potenciál jak z osobního, tak i profesního hlediska. Z osobního pohledu mě motivuje příležitost k sebevzdělávání, protože mě zajímá kompletní vedení projektu od počáteční myšlenky až k jeho realizaci. Rád bych se aktivně zapojil do všech fází – od návrhu přes vývoj po řešení různých problémů.

Z profesního hlediska mě přitahuje vývoj těchto technologií, protože věřím, že v budoucnu budou mít klíčový význam. To by mohlo zvýšit atraktivitu tohoto projektu, a časem by mohl přitáhnout pozornost investorů. Například by jej mohl chtít financovat některý zemědělec, pod podmínkou vývoje, implementace a podpory na jeho farmě, což by mi přineslo mnoho cenných zkušeností.

Na základě těchto úvah plánuji v projektu pokračovat a dále jej rozvíjet. Níže uvádím funkcionality, které mám v úmyslu implementovat po dokončení této práce.

10.1 Koncept identifikace nejlepší nosnice

Při chovu slepic by bylo velmi užitečné vědět, kolik vajec která slepice snáší, protože to umožňuje farmářům sledovat produktivitu jednotlivých slepic. Díky tomu může:

- Optimalizovat chov: Identifikací slepic s vysokou a nízkou snáškou může farmář rozhodnout o selekci nebo o zvláštní péči pro méně produktivní slepice.
- Zlepšit zdraví slepic: Nízká snáška může být indikátorem zdravotních problémů. Včasná identifikace umožňuje podniknout kroky k léčbě nebo prevenci nemocí.
- Efektivněji plánovat krmení a zdroje: Sledování výkonu pomáhá v rozhodování o výživě a péči, aby byla maximalizována produkce vajec při optimálních nákladech.
- Zvýšit ziskovost: Monitorováním snášky lze zlepšit celkovou produktivitu farmy, a tím i její ekonomickou efektivitu.

Obecně můžeme říci, že znalost počtu snesených vajec od každé slepice pomáhá v efektivním řízení chovu a zajišťuje lepší výsledky, jak po stránce produkční, tak ekonomické.

Abych toho dosáhl, potřebuji být schopen jednotlivé slepice identifikovat a tuto informaci propojit s momentem, kdy slepice snese v hnízdě vejce. V sekci 4.10, je popsán mechanismus, kdy pravidelnou analýzou časové řady, kterou poskytuje váha, získáme informaci o časovém údaji, kdy bylo vejce sneseno. Pak již jen stačí identifikovat, která z našich slepic byla v tu dobu v hnízdě.

Existuje několik variant, jak slepice identifikovat, já jsem se rozhodl využít k identifikaci obrazovou analýzu. Vždycky mě zajímalo na jakém principu funguje rozpoznávání lidí na základě kamerových záznamů, které používá policie případně sledování vozidla, které projíždí nočním městem na červenou skrze několik křižovatek.

Hledání zdrojů na internetu mě přivedlo ke spoustě článků, příspěvků, příkladů i vědeckých prací.

Nakonec jsem našel článek [46], který mě inspiroval. Z článku jsem pochopil základní princip.

Postupuji následujícím způsobem. Nejprve procházím videozáznam a identifikuji oblasti, kde se slepice nacházejí. Tyto oblasti následně vystřihnu a uložím do samostatných souborů. Poté procházím tyto menší obrázky a pomocí segmentace vystřihnu plochu, kde je slepice, zatímco ostatní části zůstanou bílé. Takto vyříznuté obrázky převádím na tenzory [47]. Převod na tenzor znamená, že dvourozměrný obraz (matice pixelů) převedu do vícerozměrné datové struktury, která dokáže reprezentovat různé vlastnosti obrazu a je vhodná pro strojové učení a další matematické zpracování. Tímto způsobem získám numerickou reprezentaci obrazu, se kterou mohu efektivně pracovat. Získané tenzory porovnávám se vzorovými vektory uloženými v databázi. Pokud je vektor shodný, znamená to, že jsem našel odpovídající slepici, a podařilo se mi ji tak identifikovat. Algoritmus tedy na počátku vezme jako vstup obrázek z kamery viz obrázek 10.1.



0_4_527.jpg

Obrázek 10.1: Jeden ze vstupů do algoritmu pro nosnice Zdroj: [Autor]

Dále je slepice z obrázku vystřižena (segmentována) podle kontury, kterou algoritmus vyhodnotil jako slepice viz obrázek 10.2.



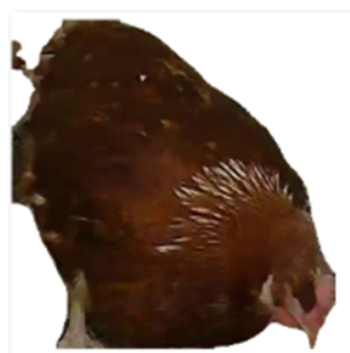
0_2_1151.jpg.png



0_2_1127.jpg.png



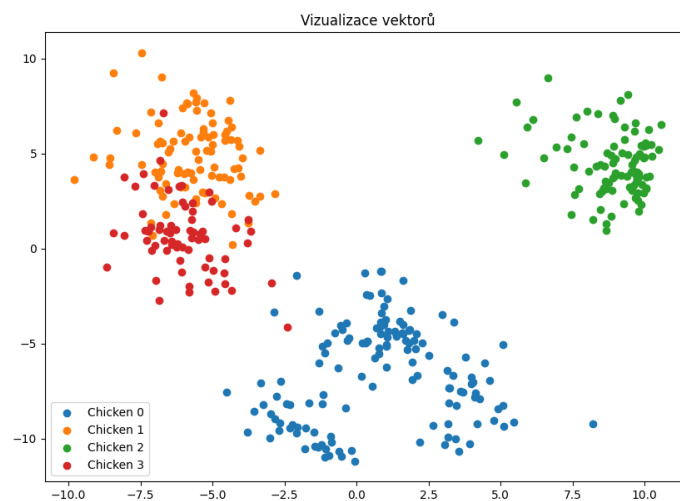
0_1_935.jpg.png



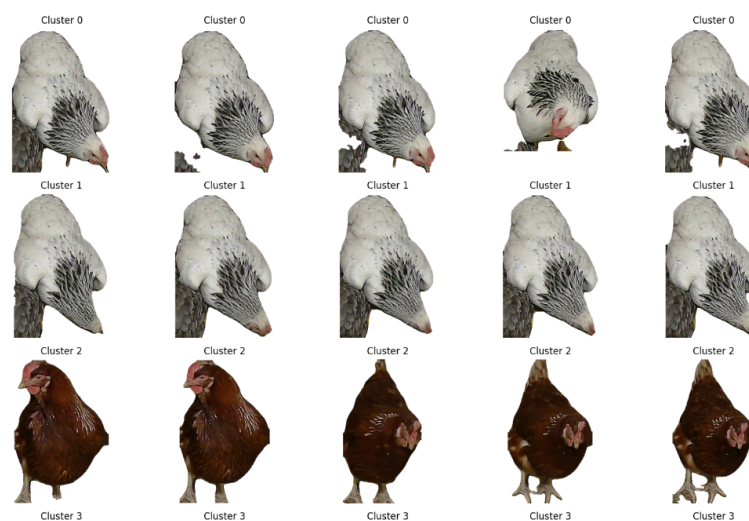
0_1_791.jpg.png

Obrázek 10.2: Segmentované slepice z fotek Zdroj: [Autor]

Na závěr je pro každý výřez vypočítán tenzor (vektor), na jehož základě jsou slepice roztrženy do jednotlivých skupin (obrázky 10.3 a 10.4).



Obrázek 10.3: Algoritmem rozdělené vektory Zdroj: [Autor]



Obrázek 10.4: Algoritmem rozdělené slepice Zdroj: [Autor]

Během této části projektu zaměřeného na třídění slepic pomocí umělé inteligence a počítačového vidění, jsem vyvinul základní procesy pro načítání a zpracování obrázků. Pomocí modelu ResNet50 [48] jsem z obrázků extraho-

val klíčové vlastnosti a pokusil se slepice zařadit do skupin pomocí metody K-means [49]. Bohužel výsledky zatím nesplnily očekávání.

Hlavním problémem je nedostatečná přesnost klasifikace. Model má potíže s rozpoznáním rozdílů mezi slepicemi, což vede k logicky nevhodným skupinám. Tím ukazuje, že model aktuálně neumí efektivně vyhodnotit důležité charakteristiky pro úspěšné třídění.

Je zřejmé, že projekt vyžaduje další práci. Potřebuji prozkoumat pokročilejší metody zpracování obrázků a shlukování, případně i techniky hlubokého učení speciálně zaměřené na rozpoznávání zvířat. Je možné, že bude nutné rozšířit dataset o větší různorodost obrázků a aplikovat techniky, které zlepší schopnost modelu třídit. Věřím, že s lepšími daty a vylepšeným modelem dosáhnu lepších výsledků.

Při testování na několika obrázcích se zdálo, že by princip mohl fungovat, ale v praxi byl výsledek nepřesný. Moje babička má dvě červené, dvě bílé a dvě černé slepice, plus jednoho bílého kohouta, což způsobuje, že jsou si slepice často velmi podobné, s nepatrnými rozdíly.

Aktuálně přemýšlím, jak tyto slepice jednoduše odlišit. Ačkoli pomalování slepic babička pravděpodobně nepovolí, možná by pomohlo zjednodušení úkolu.

Namísto soutěže o nejlepší nosnici bych mohl zaměřit úsilí na sestavení nejefektivnějšího závodního týmu, jako např. týmů bílých, červených nebo černých slepic.

10.2 Vylepšení konstrukce dvířek

Vylepšení konstrukce dvířek a realizace finálního řešení je jeden z plánů do budoucna pro tento projekt. Jak již bylo zmíněno v popisu současné konstrukce v sekci 4.13, aktuální konstrukce je příliš drahá na výrobu a také zbytečně robustní. Pro finální instalaci v následujících větvích popíšu návrh na realizaci produkční konstrukce.

Konstrukce by měla být levnější na výrobu i na materiál a zároveň stejně

bezpečná a výkonná jako ta stávající prototypová. Nový rám bude tvořen z železných U profilů použitých zároveň jako vodící lišty pro posuv dvířek. Posuvný plát bude z plastové desky.

Navíjecí mechanismus

Způsob ovládání neboli napájení se nemění takže není třeba dělat žádné úpravy na řídicí jednotce popsané detailněji v sekci 4.11. Pohyb bude dvířkům dodávat malý elektrický naviják vlastní konstrukce. Díky vlastnímu návrhu bude možno celý rám navijáku navrhnout pomocí CAD programu a následně ho celý vyrobit na 3D tiskárně. Tento způsob výroby šetří čas a je zajištěna přiměřeně stejná kvalita u všech součástí. Pohonnou jednotkou navijáku bude 12 V elektromotor model JGA25-370 na stejnosměrné napětí s vestavěnou převodovkou. Díky převodovce dokáže motor vyvinout kroutící moment až 130N na 1cm dlouhé páce, což je na pohyb dvířek dostačující. Vývodový hřídel z převodovky elektromotoru bude připevněn k navíjecímu bubnu o průměru okolo 15 mm. Na navíjecím bubnu bude připevněno i lanko, které se bude používat na vytahování dvířek nahoru. Pohyb dolů je zajištěn díky gravitaci, díky čemuž stačí povolit lanko a dvířka sama sjedou dolů. Jako pokračování v ose otáčení motoru a bubnu bude k bubnu připevněn malý šnekový hřídel s maticí. Matici je zamezeno otáčení opřením o rám navijáku, takže se při otáčení hřídele se bude matice posouvat. Tohoto pohybu bude využívat poslední součást navijáku a to systém dorazů, aby bylo možno nastavit maximální počet otáček bubnu a nedošlo tak k úplnému odvinutí lanka. Úplné odvinutí lanka by v případě, že by se motor pořád točil, znamenalo, že se lanko zase začne navíjet opačným směrem, což obrátí navíjecí logiku navijáku. Řídicí jednotka totiž počítá pouze s jedním směrem otáčení a nepočítá s náhlým přehozením směru otáček motoru pro navíjení a odvíjení.

10.3 Vylepšení funkcionality služby Room Assistant

Aktuální systém zavírání dvířek je nastaven na pevný čas, například na 18. hodinu, což představuje problém, protože nezohledňuje změny času východu a západu slunce během roku.

Plánované řešení by mělo umožnit nastavit otvírání a zavírání dveří podle aktuálních údajů o slunci. Pomocí API, jako je například <https://api.sunrise-sunset.org/>, získáme přesný čas východu a západu slunce pro naši farmu. Díky tomuto systému se dveře kurníku otevřou ve chvíli, kdy slunce vyjde, a zavřou se, když zapadne. Slepice tak mohou využít co nejvíce denního světla a večer budou chráněny před predátory a nepříznivým počasím.

V této implementaci také zajišťujeme kontrolu, zda jsou všechny slepice večer doma. Pokud ne, dveře se nezavřou. Tím se zajistí, že systém funguje podle aktuální situace.

Tento dynamický systém je mnohem lepší než starší statické nastavení, protože snižuje možnost lidské chyby, usnadňuje práci farmářům a zlepšuje péči o slepice.

11 Teoretické pojmy

11.1 Docker Engine

Jedna z technologií používaných pro kontejnerizaci je Docker Engine [50]. Používá se pro tvorbu, správu, orchestraci, verzování a nasazování jednotlivých kontejnerů.

11.2 Docker Compose

Docker compose [51] je nástroj pro Docker, který nám umožňuje tvořit komplikované ekosystémy jednotlivých kontejnerů, jež spolu v rámci něho mohou spolupracovat. Pomáhá nám například se síťováním nebo konfigurací kontejnerů.

11.3 Mikroservisní architektura

Mikroservisní architektura [52] je přístup k vývoji softwarových aplikací, kdy je celek rozdělen na malé, nezávislé služby nazývané mikroservisy. Každá služba běží samostatně, komunikuje s ostatními službami pomocí dobře definovaného API a je zaměřena na konkrétní funkcionalitu. Tento model umožňuje snadnější údržbu, škálovatelnost a flexibilitu systému.

11.4 Jazyk Python

Python [53] je vysokoúrovňový programovací jazyk známý svou jednoduchou a čitelnou syntaxí. Podporuje několik programovacích paradigmat, včetně objektově orientovaného, procedurálního a funkcionálního programování. Python je široce používán v různých oblastech, jako je webový vývoj, vědecké výpočty, umělá inteligence, strojové učení a datová analýza. Disponuje rozsáhlou standardní knihovnou a množstvím externích modulů, které usnadňují vývoj komplexních aplikací. Je multiplatformní, což znamená, že programy v Pythonu lze spouštět na různých operačních systémech.

11.5 Wiring

Wiring [54] je open source programovací jazyk a vývojové prostředí určené pro práci s mikrokontroléry. Jeho cílem je usnadnit programování elektronických zařízení a interaktivních projektů, zejména pro umělce, designéry a studenty. Syntaxe jazyka Wiring vychází z jazyka C++ a je navržena tak, aby byla snadno pochopitelná i pro začátečníky.

11.6 C++

C++ [55] je univerzální vysokoúrovňový programovací jazyk, který vychází z jazyka C a rozšiřuje jej o objektově orientované programování. Vyznačuje se vysokou rychlostí a efektivitou, což jej předurčuje k tvorbě výkonnostně náročných aplikací, jako jsou graficky náročné hry či systémy reálného času. Mezi jeho silné stránky patří mimo jiné možnost nízkoúrovňové práce s hardwarem a pamětí, což jej činí vhodným pro vestavěné systémy nebo vývoj ovladačů. Díky podpoře více programovacích paradigmat (procedurální, objektově orientované i generické) umožňuje psát kvalitní, udržitelný a zároveň výkonný kód.

11.7 Táhlový motor

Jedná se o lineární motor (actuator) [56], jehož součástí je elektromotor a k němu je připevněn šnekový převod. Šnek je pevně upevněn na šroubovici, která se stará o rozvod síly a točivého momentu po celé délce motoru. Převod z otáčivého pohybu na pohyb přímočarý je zajištěn maticí, která pevně drží na táhlu. Rozhodnutí, zda se táhlo zasouvat nebo vysouvat, je dáno polaritou napětí dodávaného elektromotoru, který se na základě toho otáčí buď po nebo proti směru hodinových ručiček.

11.8 Github workflow

GitHub Workflow je nástroj pro automatizaci procesů při vývoji softwaru na platformě GitHub. Umožňuje vytvářet a spravovat tzv. workflow pomocí souborů ve formátu YAML, které definují jednotlivé kroky. Tyto kroky nebo také akce, se mohou spouštět na základě různých událostí, jako je push kódu nebo vytvoření pull requestu. GitHub Workflow podporuje kontinuální integraci a nasazení (CI/CD), automatické testování, nasazení aplikací a další úlohy.

11.9 MQTT

MQTT (Message Queuing Telemetry Transport) [22] je lehký messagingový protokol pro publikování a odebírání zpráv, navržený pro komunikaci mezi zařízeními v sítích s omezenou kapacitou nebo vysokou latencí. Je často využíván v oblasti Internetu věcí (IoT) pro přenos dat mezi senzory, akčními členy a centrálními systémy. MQTT pracuje na principu architektury klient–server, kde klienti(publisheři) publikují zprávy na určité téma(topic) a broker tyto zprávy distribuuje odběratelům(subscriberům), kteří jsou na dané téma přihlášení.

11.10 Home Assistant

Home Assistant [57] je open source platforma pro automatizaci domácnosti, která umožňuje integraci různých IoT zařízení pomocí protokolů jako MQTT, Z-Wave a Zigbee. Umožňuje vytvářet pokročilé automatizace a skripty prostřednictvím YAML konfigurace nebo webového rozhraní, přičemž vše běží lokálně, což zajišťuje vysokou úroveň soukromí a bezpečnosti. Podporuje integraci s hlasovými asistenty jako Google Assistant a Amazon Alexa. Platforma je kompatibilní s různými zařízeními, včetně Raspberry Pi, a nabízí mobilní aplikace pro iOS a Android pro vzdálený přístup. Home Assistant je flexibilní, rozšiřitelný a široce používaný v aplikacích pro chytrou domácnost a domácí bezpečnostní systémy. Díky pravidelným aktualizacím a podpoře komunity je ideálním nástrojem pro uživatele, kteří chtějí mít plnou kontrolu nad svou chytrou domácností.

11.11 Arduino

Arduino je open source platforma pro prototypování elektroniky založená na snadno použitelném hardwaru a softwaru. Skládá se z mikroprocesorové desky a vývojového prostředí Arduino IDE [30], které využívá jazyk podobný C/C++. Arduino desky umožňují komunikaci s různými senzory a akčními členy, což usnadňuje tvorbu interaktivních projektů. Podporuje řadu rozšiřujících modulů, tzv. shieldů, které rozšiřují jeho funkčnost například o bezdrátovou komunikaci, ovládání motorů či připojení k internetu. Arduino usnadňuje rychlý vývoj a testování elektronických aplikací bez hlubokých znalostí elektroniky.

11.12 IP kamera a RTSP

IP kamera je digitální zařízení, které přenáší obraz a zvuk přes IP síť, jako je internet nebo lokální síť. To umožňuje vzdálený přístup k živému vysílání

nebo záznamům bez nutnosti speciálního kabelového připojení. Pro efektivní přenos multimediálních dat v reálném čase se často využívá protokol RTSP (Real Time Streaming Protocol).

11.13 Flask

Flask je lehký webový framework pro Python, který umožňuje rychlé a jednoduché vytváření webových aplikací. Patří mezi mikroframeworky, což znamená, že poskytuje pouze základní funkce potřebné pro webový vývoj, jako je směrování URL a zpracování HTTP požadavků. Díky své modularitě umožňuje vývojářům přidávat rozšíření a knihovny podle potřeby, například pro práci s databázemi, autentizaci či validaci. Flask využívá šablonovací systém Jinja2 pro vytváření dynamických HTML stránek a nástroj Werkzeug pro WSGI kompatibilitu.

11.14 Yolo Ultralytics

Ultralytics YOLO je pokročilý systém pro detekci objektů v reálném čase založený na hlubokém učení. Jedná se o implementaci architektury YOLO (You Only Look Once), kterou vyvinula společnost Ultralytics. Tento model využívá konvoluční neuronové sítě k analýze obrazových dat a současně identifikaci více objektů během jediného průchodu sítí. Díky své vysoké rychlosti a přesnosti je ideální pro aplikace náročné na čas, jako je autonomní řízení, bezpečnostní systémy nebo analýza videa. Ultralytics YOLO je dostupný jako open source software, což umožňuje jeho široké využití ve výzkumu i v průmyslových aplikacích.

11.15 Tenzometrický senzor

Tenzometrický senzor je zařízení sloužící k měření mechanického napětí nebo deformace v materiálu či konstrukci. Základním principem tenzometru je

využití změny elektrického odporu vodivého materiálu při jeho mechanickém zatížení. Nejčastěji se používají tenzometry s kovovou fólií nebo drátkem, který je pevně spojen s měřeným objektem. Když je objekt zatížen silou, dojde k jeho deformaci, což způsobí prodloužení nebo zkrácení tenzometru a tím i změnu jeho elektrického odporu. Tato změna je úměrná velikosti aplikovaného napětí a může být přesně změřena pomocí elektrických obvodů, jako je Wheatstoneův můstek. Tenzometrické senzory nacházejí uplatnění v oblastech jako je strojírenství, stavebnictví, letectví či při vývoji nových materiálů, kde je důležité sledovat mechanické vlastnosti a bezpečnost konstrukcí.

11.16 POE

POE je technologie umožňující přenos elektrické energie společně s daty prostřednictvím standardního Ethernetového kabelu. Tato technologie umožňuje napájet síťová zařízení, jako jsou IP kamery, bezdrátové přístupové body nebo VoIP telefony, bez potřeby samostatného napájecího zdroje. Technologie využívá nevyužité páry vodičů v kabelu nebo kombinuje napájení s datovými signály na stejných vodičích. Existují různé standardy PoE, jako IEEE 802.3af, 802.3at a 802.3bt, které definují maximální výkon a kompatibilitu zařízení. Použití PoE zjednodušuje instalaci, snižuje náklady na kabeláž a umožňuje flexibilnější umístění zařízení bez závislosti na elektrických zásuvkách.

11.17 Wifi extender

Wi-Fi extender, také známý jako repeater nebo zesilovač signálu, je zařízení určené k rozšíření dosahu bezdrátové sítě. Přijímá existující Wi-Fi signál z routeru a znovu ho vysílá do oblastí s nedostatečným pokrytím. Tím eliminuje "mrtvé zóny" v domácnosti nebo kanceláři, kde je signál slabý nebo žádný. Instalace je obvykle jednoduchá a nevyžaduje dodatečné kabely. Wi-Fi exten-

dery podporují různé standardy Wi-Fi, jako 802.11n nebo 802.11ac, a nabízejí přenosové rychlosti odpovídající těmto standardům. Pro správné využití je důležité umístit extender tam, kde ještě přijímá silný signál z routeru.

11.18 TailScale VPN

Tailscale VPN je moderní služba pro vytváření virtuálních privátních sítí, která využívá protokol WireGuard k zajištění bezpečné komunikace mezi zařízeními. Umožňuje snadné nastavení sítě bez složitých konfiguračních procesů tradičních VPN řešení. Služba vytváří šifrované peer-to-peer spojení mezi zařízeními na základě jejich identity, spravované prostřednictvím cloudu. To umožňuje uživatelům bezpečně přistupovat k interním sítím a službám odkudkoli na světě. Díky automatické správě síťových konfigurací a firewallu snižuje nároky na údržbu a zvyšuje celkovou bezpečnost. Tailscale je vhodný pro jednotlivce, týmy i organizace hledající efektivní a jednoduché VPN řešení pro propojení svých zařízení.

11.19 Postgres 15

PostgreSQL [58] je výkonný open-source relační databázový systém. Je navržen pro ukládání a správu strukturovaných dat a podporuje SQL (Structured Query Language) pro dotazování a manipulaci s daty. Často je používán pro webové aplikace, analytiku a další databázové aplikace vyžadující robustní a efektivní datové řešení.

11.20 Eclipse Mosquitto

Eclipse Mosquitto [59] je open-source broker pro protokol MQTT. Slouží k zprostředkování komunikace mezi zařízeními v prostředích internetu věcí (IoT). Mosquitto umožňuje klientům publikovat a odebírat zprávy na konkrétních kanálech (tzv. tématech), čímž umožňuje efektivní a škálovatelnou výměnu

dat mezi senzory, síťovými zařízeními a aplikacemi.

11.21 Cloudflared

Cloudflared je nástroj, který umožňuje propojení mezi lokální sítí a službami Cloudflare. Primárně se používá pro tunelování internetového provozu do lokální sítě (tzv. Cloudflare Tunnel). To zajišťuje bezpečné a šifrované připojení mezi Cloudflare a vaší sítí, což pomáhá chránit aplikace, servery a další zdroje před hrozbami z internetu.

11.22 COCO formát

COCO formát (Common Objects in Context) je standardní datový formát pro anotace obrázků, zejména v oblasti počítačového vidění. Umožňuje ukládat informace o třídách objektů, jejich bounding boxech, segmentacích a klíčových bodech v jednom JSON souboru. Díky jednotné struktuře usnadňuje trénování a vyhodnocování modelů pro detekci a rozpoznávání objektů. V praxi se velmi často využívá pro benchmarking a sdílení datasetů.

12 Závěr

Závěrem lze říci, že se mi podařilo úspěšně vyvinout systém, který výrazně usnadňuje hospodářům plnění každodenních úkolů v reálném čase. Díky mé aplikaci může hospodář kdykoli kontrolovat situaci v kurníku i ve výběhu, aniž by musel být fyzicky přítomen. Systém ho automaticky informuje o jakýchkoli nesrovnalostech, například pokud se ve výběhu objeví vetřelec, který by mohl představovat nebezpečí pro slepice. Úspěšně jsem také implementoval kontrolu počtu slepic v kurníku, což je užitečné například při automatizaci zavírání dvířek.

Dále systém poskytuje aktuální záběry z bezpečnostních kamer, což umožňuje hospodáři mít neustálý přehled o svém chovu. Implementoval jsem také část systému schopnou kontrolovat stavy jednotlivých hnízd a zjistit, zda v nich sedí slepice nebo kolik vajec je v hníždě. Tato data lze využít k tvorbě statistik, díky nimž může hospodář analyzovat různé aspekty svého chovu.

Na druhou stranu jsem narazil na některé výzvy. Nepodařilo se mi plně optimalizovat systém pro detekci vetřelců za zhoršených světelných podmínek, což může ovlivnit jeho spolehlivost v nočních hodinách. Také přesnost při rozpoznávání počtu vajec v hnízdech potřebuje další vylepšení, aby byla zajištěna maximální spolehlivost dat.

Do budoucna, až projekt zcela dokončím, plánuji jeho výslednou podobu publikovat v rámci komunity Home Assistant. Věřím, že díky zveřejnění tohoto řešení se jím budou moci inspirovat další nadšenci, kteří se zajímají o monitorování a automatizaci. Zároveň také doufám, že jim tato práce přinese

plno zajímavých informací a myšlenek, které budou ve svých projektech moci dále rozvíjet a vylepšovat.

Celkově můj systém představuje významný krok vpřed v automatizaci a monitorování chovu slepic, ačkoli stále existuje prostor pro další zdokonalení a optimalizaci některých funkcí.

Seznam zkratek

API Application Programming Interface 21

AWS Amazon Web Services 24

CSV Comma-Separated Values 49

Full HD Full High Definition 26

HTTP Hypertext Transfer Protocol 17

IDE Integrated Development Environment 81

IoT Internet of Things 17

IP Internet Protocol 24

JSON JavaScript Object Notation 30

MQTT Message Queuing Telemetry Transport 17

POE Power over Ethernet 107

REST Representational State Transfer 21

USB Universal Serial Bus 53

Použitá literatura

1. BECK, Kent; BEEDLE, Mike; BENNEKUM, Arie van; COCKBURN, Alistair; CUNNINGHAM, Ward; FOWLER, Martin; GRENNING, James; HIGHSMITH, Jim; HUNT, Andrew; JEFFRIES, Ron; KERN, Jon; MARICK, Brian; MARTIN, Robert C.; MELLOR, Steve; SCHWABER, Ken; SUTHERLAND, Jeff; THOMAS, Dave. *Manifest Agilního vývoje software* [online]. 2001 [vid. 2025-02-22]. Dostupné z: <https://agilemanifesto.org/iso/cs/manifesto.html>.
2. BAIJU, Nevin. *Home Assistant Home page* [online]. 2025 [vid. 2025-02-22]. Dostupné z: <https://www.home-assistant.io/>.
3. YOLO, Ultralytics. *Image Classification* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.ultralytics.com/glossary/image-classification>.
4. ETECHBLOGCZ. *Úvod do neuronových sítí* [online]. 2023 [vid. 2025-02-24]. Dostupné z: <https://etechblog.cz/uvod-do-neuronovych-siti-5-learning-resources/>.
5. MICROSOFT. *Microsoft Azure: Cloud Computing Services* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://azure.microsoft.com/cs-cz/>.
6. SERVICES, Amazon Web. *Cloud computing with AWS* [online]. 2024 [vid. 2025-02-24]. Dostupné z: <https://aws.amazon.com/what-is-aws/>.

7. CLOUDFLARE. *Cloudflare VPN* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.cloudflare.com/en-gb/zero-trust/solutions/vpn-replacement/>.
8. GODADDY. *GoDaddy: Domény, webové stránky, hosting a nástroje pro online marketing* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.godaddy.com/>.
9. HOSTINGER. *Hostinger: Vytvořte svou webovou stránku snadno a rychle* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.hostinger.com/>.
10. FORPSI. *Domeny Forpsi* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.forpsi.com/>.
11. PROJECTS, Pallets. *Flask Documentation* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://flask.palletsprojects.com/>.
12. IETF. *HTTP Digest Access Authentication* [online]. 2015 [vid. 2025-02-24]. Dostupné z: <https://datatracker.ietf.org/doc/html/rfc7616>.
13. LIECHTI, Chris. *pySerial* [online]. 2020 [vid. 2025-02-24]. Dostupné z: <https://pyserial.readthedocs.io/en/latest/pyserial.html>.
14. LABS, Grafana. *Grafana Yolo* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://grafana.com/>.
15. GRÖNHOLM, Alex. *APScheduler: User guide* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://apscheduler.readthedocs.io/en/3.x/userguide.html>.
16. INC., Ultralytics. *Ultralytics Yolo* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://docs.ultralytics.com/models/yolo11>.
17. MQTT.ORG. *MQTT: The Standard for IoT Messaging* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://mqtt.org/>.

18. CLOUDFLARE. *Cloudflare Tunnel* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://developers.cloudflare.com/cloudflare-one/connections/connect-networks/>.
19. ASSISTANT, Home. *Picture card* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.home-assistant.io/dashboards/picture/>.
20. ASSISTANT, Home. *Button card* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.home-assistant.io/dashboards/button/>.
21. ASSISTANT, Home. *Conditional card* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.home-assistant.io/dashboards/conditional/>.
22. ASSISTANT, Home. *MQTT* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.home-assistant.io/integrations/mqtt/>.
23. ASSISTANT, Home. *Entities and domains* [online]. 2025 [vid. 2025-02-24]. Dostupné z: https://www.home-assistant.io/docs/configuration/entities_domains/.
24. VYORÁLEK, Jiří. *Automatizace v Home Assistantovi* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://jirkovynavody.cz/homeassistant/automations/intro/>.
25. ASSISTANT, Home. *Custom card* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://developers.home-assistant.io/docs/frontend/custom-ui/custom-card/>.
26. COMMUNITY, Home Assistant. *Rest sensor: State max length is 255 characters* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://community.home-assistant.io/t/state-max-length-is-255-characters/31807>.
27. TINA. *Jak připojit zesilovač zatížení buněk k Arduino?* [online]. 2024 [vid. 2025-02-24]. Dostupné z: <https://www.fibossensor.com/cs/how-to-connect-load-cell-amp-to-arduino.html>.
28. ARDUINO. *Nano* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://docs.arduino.cc/hardware/nano/>.

29. GODIYAL, Kanishk. *How Much Power does USB Port Output? (USB 1.0, 2.0, 3.0)* [online]. 2024 [vid. 2025-02-24]. Dostupné z: <https://www.etechnophiles.com/usb-port-power-output/>.
30. ARDUINO. *Arduino IDE* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.arduino.cc/en/software>.
31. DFROBOT. *DHT11 Humidity & Temperature Sensor* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.digikey.com/htmldatasheets/production/1801487/0/0/1/386.html>.
32. A.S., Alza.cz. *Hilook by Hikvision IPC-B180HA-LU* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.alza.cz/hilook-by-hikvision-ipc-b180ha-lu-d12432819.htm>.
33. MICROSOFT. *Azure Machine Learning* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://azure.microsoft.com/en-us/products/machine-learning>.
34. MICROSOFT. *COCO Dataset Format* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://cocodataset.org/#overview>.
35. YOLO, Ultralytics. *Instance Segmentation* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.ultralytics.com/glossary/instance-segmentation>.
36. ROBERT FALTUS, Aleš Svoboda. *Tipy pro nahrávání programu do Arduina* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://navody.drateg.cz/technikuv-blog/nahravani-programu-do-arduina.html>.
37. HOMANN, Maria. *The Testing Pyramid* [online]. 2024 [vid. 2025-02-24]. Dostupné z: <https://www.leapwork.com/blog/testing-pyramid>.
38. GURU99. *Co je Postman?* [online]. 2024 [vid. 2025-02-24]. Dostupné z: <https://www.guru99.com/cs/postman-tutorial.html>.
39. NORDQUIST, Thomas. *MQTT Explorer* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://mqtt-explorer.com/>.

40. TECNODIGITAL. *GitHub, co to je a jak funguje* [online]. 2023 [vid. 2025-02-24]. Dostupné z: <https://informatecdigital.com/cs/github-co-to-je-a-jak-to-funguje/>.
41. DOCKER, Inc. *Docker Build GitHub Actions* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://docs.docker.com/build/ci/github-actions/>.
42. DOCKER. *Docker Hub* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://hub.docker.com/>.
43. A.S., Alza.cz. *Intel NUC 11 Enthusiast NUC11PHKi7C* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.alza.cz/intel-nuc-11-enthusiast-nuc11phki7c-d6735655.htm>.
44. FOUNDATION, Raspberry Pi. *Raspberry Pi 5 – Raspberry Pi* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.raspberrypi.com/products/raspberry-pi-5/>.
45. INC., Tailscale. *What is Tailscale?* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://tailscale.com/kb/1151/what-is-tailscale>.
46. BAIJU, Nevin. *Navigating the Maze: Person Re-Identification Across Multiple Cameras with Deep Metric Learning* [online]. 2024 [vid. 2025-02-22]. Dostupné z: https://medium.com/@nevinbaiju_77488/navigating-the-maze-person-re-identification-across-multiple-cameras-with-deep-metric-learning-1083bdd37616.
47. ŠLÉGR, Jan. *Tenzory a tenzorový počet* [online]. 2012 [vid. 2025-02-24]. Dostupné z: <https://lide.uhk.cz/prf/ucitel/slegrja1/tenzory.pdf>.
48. CONTRIBUTORS, PyTorch. *resnet50* [online]. 2017 [vid. 2025-02-24]. Dostupné z: <https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html>.

49. DEVELOPERS, scikit-learn. *2.3.2. K-means* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>.
50. INC., Docker. *Docker Engine* [online]. 2025 [vid. 2025-02-09]. Dostupné z: <https://docs.docker.com/engine/>.
51. INC., Docker. *Docker Compose* [online]. 2025 [vid. 2025-02-09]. Dostupné z: <https://docs.docker.com/compose/>.
52. JŮZA, Petr. *Jak nedělat mikroservisy* [online]. 2020 [vid. 2025-02-24]. Dostupné z: <https://medium.com/openwise-tech-blog/jak-ned%C4%9Blat-mikroservisy-d93e004b8a2e>.
53. SLÁMA, Zdeněk. *Co je Python a k čemu se dá použít* [online]. 2024 [vid. 2025-02-24]. Dostupné z: https://slamazdenek.cz/cz/python/co-je-python-a-k-cemu-se-da-pouzit?utm_source=chatgpt.com.
54. MALÝ, Martin. *Arduino: jak pro něj začít programovat* [online]. 2010 [vid. 2025-02-24]. Dostupné z: <https://www.root.cz/clanky/arduino-jak-pro-nej-zacit-programovat/>.
55. BARNEY, Nick. *C++* [online]. 2023 [vid. 2025-02-24]. Dostupné z: <https://www.techtarget.com/searchdatamanagement/definition/C>.
56. R.O., ATOS spol. s. *Táhlový motor HARL 3618+, délka výtahu 18"* [online]. 2025 [vid. 2025-02-24]. Dostupné z: https://shop.atoselektro.cz/tahlovy-motor-harl-3618-delka-vytahu-18-_d494489.html?srsId=AfmB0orzD3cSSfVw6jZfo1LYmbwfl9SUFHD6NF7Bf8IrBE2XfRWmCL4w.
57. VYPÍNAČE, Chytré. *Home Assistant / Alternativní firmware Tasmota / ZigBee / Z-Wave* [online]. 2025 [vid. 2025-02-22]. Dostupné z: https://www.chytrevypinace.cz/home-assistant/-/alternativni-firmware-tasmota/-/zigbee/-z-wave?utm_source=chatgpt.com.

58. GROUP, The PostgreSQL Global Development. *PostgreSQL 15.12 Documentation* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://www.postgresql.org/docs/15/>.
59. FOUNDATION, Eclipse. *Eclipse Mosquitto* [online]. 2025 [vid. 2025-02-24]. Dostupné z: <https://mosquitto.org/>.

Seznam obrázků

3.1	Diagram komunikace mezi relé a senzory	20
3.2	Diagram bezpečného zavření dvířek	22
3.3	Diagram detekce vetřelců	22
3.4	Diagram vizualizace stavů hnízd	23
3.5	Orientační topologie systému	25
4.1	Vizualizace dat ze senzoru v systému Home Assistant	31
4.2	Možná vizualizace pomocí platformy Grafana	33
4.3	Výstup ze služby Health Checker	33
4.4	Prohlížeč zpráv uložených na MQTT serveru.	34
4.5	Graf vývoje hmotnosti hnízda v čase	37
4.6	Pes detekován ve výběhu	42
4.7	Obrázek přijatý z MQTT a vizualizovaný v dashboardu	44
4.8	Komponenta pro ovládání světel systému Home Assistant	47
4.9	Vlastní komponenta pro systém Home Assistant	50
4.10	Mobilní aplikace Home Assistant	52
4.11	První prototyp váhy instalovaný v hnízdě	55
4.12	Druhý prototyp váhy s logem	55
4.13	Skutečné zapojení řídicí jednotky	59
4.14	Blokové schéma řídicí jednotky	59
4.15	Rozměry táhlového motoru	62
4.16	Instalovaný prototyp dvířek	64

5.1	Vytvoření nového projektu v AML	67
5.2	Tvorba datasetu v AML	68
5.3	Definice kategorií pro učení v AML	68
5.4	Označování slepic pomocí AML	69
5.5	Export metadat k obrázkům z AML do COCO formátu	70
5.6	Ukázka COCO metadat pro učení	71
5.7	Kód pro učení nového modelu	72
5.8	Proces trénování YOLO11 modelu	73
5.9	Graf loss funkce v závislosti na počtu etap	75
5.10	Top-1 accuracy chart	75
5.11	Model detekující slepice	76
5.12	Proces anotace pro trénování segmentace	78
5.13	Výsledek detekce slepic 1	80
5.14	Výsledek detekce slepic 2	80
8.1	Nainstalovaný rozvaděč ve chlévě	86
8.2	Vnitřek rozvaděče	87
10.1	Jeden ze vstupů do algoritmu pro nosnice	96
10.2	Segmentované slepice z fotek	97
10.3	Algoritmem rozdělené vektory	98
10.4	Algoritmem rozdělené slepice	98