Politechnika
Śląska

PROJEKT INŻYNIERSKI

System wizyjny dla robota mobilnego

Łukasz GRABARSKI
Nr albumu: 300434

Kierunek: Automatyka i Robotyka
Specjalność: Technologie Informacyjne

PROWADZĄCY PRACĘ
dr inż. Krzysztof Jaskot
KATEDRA Automatyki i Robotyki
Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2024

Tytuł pracy

System wizyjny dla robota mobilnego

Streszczenie

(Streszczenie pracy – odpowiednie pole w systemie APD powinno zawierać kopię tego streszczenia.)

Słowa kluczowe

RaspberryPi, Python, OpenCV, YOLO, PyTorch

Thesis title

Vision system for a mobile robot

Abstract

(Thesis abstract – to be copied into an appropriate field during an electronic submission – in English.)

Key words

RaspberryPi, Python, OpenCV, YOLO, PyTorch

Spis treści

Rozdział 1

Wstęp

Rozwój technologii komputerowej oraz metod sztucznej inteligencji w ostatnich dekadach znacząco zmienił podejście do projektowania i implementacji systemów wizyjnych. Szczególnie istotne w tym kontekście stały się algorytmy głębokiego uczenia, które umożliwiły precyzyjne rozpoznawanie i analizę obrazów w czasie rzeczywistym. W połączeniu z dostępnością wydajnych i ekonomicznych platform obliczeniowych, takich jak Raspberry Pi, rozwiązania te znajdują szerokie zastosowanie w robotyce mobilnej. Niniejsza praca koncentruje się na opracowaniu systemu wizyjnego dla robota mobilnego z wykorzystaniem algorytmów YOLOv7, biblioteki PyTorch oraz OpenCV. System ma umożliwić detekcję i śledzenie obiektów w czasie rzeczywistym, co stanowi kluczowy element autonomicznego działania robota w dynamicznym środowisku.

1.1 Wprowadzenie w problem

Robotyka mobilna jest jedną z najszybciej rozwijających się dziedzin technologii, znajdującą zastosowanie zarówno w przemyśle, jak i w gospodarstwach domowych. Dzięki zastosowaniu systemów wizyjnych, roboty mobilne zyskują zdolność interakcji z otoczeniem, co zwiększa zakres ich funkcjonalności. Systemy wizyjne umożliwiają identyfikację obiektów, analizę ich ruchu, a także podejmowanie decyzji w czasie rzeczywistym, co otwiera drogę do autonomicznego działania robotów w dynamicznych środowiskach.

Rozwój technologii takich jak głębokie uczenie (ang. *Deep Learning*) oraz dostępność platform sprzętowych o dużej mocy obliczeniowej, poczynając od Raspberry Pi aż po NVIDIA Jetson, pozwalają na implementację złożonych algorytmów przetwarzania obrazu i sterowania robotem. W niniejszej pracy wykorzystano otwartoźródłową bibliotekę programistyczną PyTorch oraz algorytm YOLO (*You Only Look Once*) do realizacji zadań związanych z detekcją i śledzeniem obiektów.

1.2 Osadzenie problemu w dziedzinie

Identyfikacja i śledzenie obiektów w czasie rzeczywistym jest jednym z kluczowych wyzwań w robotyce mobilnej. Dzięki zastosowaniu metod przetwarzania obrazu, roboty mogą nie tylko „widzieć" otoczenie, ale również analizować je i realizować przypisane im funkcje. Problem ten jest szczególnie istotny w systemach autonomicznych, które muszą działać w zróżnicowanym środowisku i dostosowywać się do dynamicznych i nieprzewidywalnych zmian.

Wyzwaniem w realizacji takich systemów jest optymalizacja algorytmów wizyjnych pod kątem wydajności i dokładności, dostosowując je do ograniczeń sprzętowych platform takich jak Raspberry Pi. Integracja z systemami sterowania i nawigacji robotów mobilnych wymaga również opracowania odpowiednich rozwiązań programistycznych oraz architektonicznych.

1.3 Cel pracy

Celem pracy jest opracowanie systemu wizyjnego dla robota mobilnego, który umożliwi identyfikację różnego rodzaju obiektów, śledzenie ich ruchu oraz podążanie za nimi. W szczególności praca koncentruje się na:

- Implementacji algorytmu YOLOv7 w oparciu o bibliotekę PyTorch na platformie Raspberry Pi 4B.
- Optymalizacji systemu wizyjnego do pracy w czasie rzeczywistym w różnych warunkach środowiskowych.
- Integracji systemu wizyjnego z układem sterowania robota mobilnego.
- Analizie wydajności systemu oraz ocenie jego funkcjonalności w warunkach rzeczywistych.

1.4 Zakres pracy

Realizacja projektu obejmuje następujące etapy:

- **Analiza literatury** – przegląd istniejących rozwiązań w zakresie detekcji i śledzenia obiektów oraz systemów wizyjnych.
- **Projekt systemu** – zaprojektowanie architektury systemu wizyjnego, uwzględniając specyfikę robota mobilnego oraz ograniczenia sprzętowe platformy Raspberry Pi.
- **Implementacja** – realizacja algorytmów detekcji i śledzenia obiektów z wykorzystaniem YOLOv7, PyTorch oraz OpenCV.
- **Integracja** – połączenie systemu wizyjnego z modułem sterowania robota.
- **Testy i walidacja** – przeprowadzenie testów w rzeczywistych warunkach operacyjnych oraz analiza wyników działania systemu.

1.5 Struktura pracy

Praca została podzielona na następujące rozdziały:

- Rozdział pierwszy – **Wstęp**: zawiera wprowadzenie w problem, cel pracy, zakres oraz strukturę dokumentu.
- Rozdział drugi – **Analiza tematu**: przedstawia aktualny stan wiedzy, analizę literatury oraz istniejących rozwiązań w dziedzinie systemów wizyjnych.
- Rozdział trzeci – **Wymagania i narzędzia**: opisuje wymagania funkcjonalne systemu oraz narzędzia i technologie użyte w pracy.
- Rozdział czwarty – **Projekt systemu wizyjnego**: przedstawia architekturę systemu, zastosowane algorytmy oraz szczegóły implementacji.
- Rozdział piąty – **Weryfikacja i walidacja**: opisuje metodologię testowania, uzyskane wyniki oraz ich analizę.
- Rozdział szósty – **Podsumowanie i wnioski**: zawiera podsumowanie wykonanej pracy oraz propozycje dalszych badań.

1.6 Wkład własny autora

W ramach niniejszej pracy autor:

- Przygotował zestaw danych treningowych zawierający obrazy obiektu w różnych warunkach oświetleniowych.
- Przeprowadził trening sieci neuronowej YOLOv7 na platformie Google Colab.
- Samodzielnie zaimplementował system wizyjny z wykorzystaniem YOLOv7.
- Zaprojektował robota mobilnego oraz moduł sterowania.
- Zintegrował system wizyjny z robotem mobilnym.
- Przeprowadził testy systemu w różnych warunkach środowiskowych oraz dokonał analizy wyników.

Rozdział 2

[Analiza tematu]

2.1 Sformułowanie problemu

Współczesna robotyka mobilna stawia liczne wyzwania związane z autonomicznym działaniem robotów w dynamicznych środowiskach. Jednym z kluczowych aspektów jest detekcja i śledzenie obiektów w czasie rzeczywistym, co umożliwia robotowi interakcję z otoczeniem oraz podejmowanie decyzji w oparciu o dane wizualne. Problem ten komplikuje się w warunkach zmiennego oświetlenia, ograniczonej mocy obliczeniowej oraz konieczności integracji algorytmów wizji komputerowej z systemami sterowania.

Systemy wizyjne pełnią istotną rolę w realizacji autonomii robota, umożliwiając:

- wykrywanie przeszkód i nawigację w środowisku,
- identyfikację i śledzenie określonych obiektów,
- analizę otoczenia w czasie rzeczywistym.

W niniejszej pracy problematyka ta jest analizowana w kontekście systemów wizyjnych opartych na algorytmach głębokiego uczenia, takich jak YOLOv7, zaimplementowanych na platformie Raspberry Pi.

2.2 Stan wiedzy i osadzenie w kontekście aktualnych badań

W ostatnich latach w dziedzinie robotyki mobilnej i systemów wizyjnych zaobserwowano dynamiczny rozwój technik opartych na głębokim uczeniu. Metody te, w szczególności algorytmy detekcji obiektów, znalazły zastosowanie w projektach przemysłowych i badaniach naukowych.

2.2.1 Metody przetwarzania obrazu

Klasyczne podejścia, takie jak filtry Sobela czy segmentacja obrazów, miały ograniczoną skuteczność w dynamicznych środowiskach. Wprowadzenie algorytmów głębokiego uczenia, takich jak YOLO (*You Only Look Once*), znacząco poprawiło możliwości detekcji obiektów w czasie rzeczywistym.

2.2.2 Algorytmy detekcji obiektów

Współczesne metody detekcji obiektów można podzielić na:

- **Algorytmy jednoetapowe (ang. single-stage)**: YOLO, SSD, które łączą detekcję i klasyfikację w jednym przebiegu, oferując wysoką wydajność w czasie rzeczywistym.
- **Algorytmy dwuetapowe (ang. two-stage)**: Faster R-CNN, które cechuje wyższa dokładność, ale kosztem wydajności.

YOLO wyróżnia się szybkością działania oraz zdolnością do pracy na urządzeniach z ograniczoną mocą obliczeniową, takich jak Raspberry Pi.

2.2.3 Zastosowania w robotyce mobilnej

Systemy wizyjne oparte na YOLO są wykorzystywane w projektach przemysłowych i naukowych. Przykłady zastosowań obejmują:

- nawigację autonomiczną w robotach AGV (Automated Guided Vehicles),
- wykrywanie przeszkód i analiza ścieżki w pojazdach autonomicznych,
- systemy inspekcji w przemyśle.

2.3 Studia literaturowe i znane rozwiązania

W literaturze naukowej można znaleźć liczne prace dotyczące implementacji systemów wizyjnych w robotyce mobilnej. W niniejszym rozdziale przytoczono przykłady wybranych badań.

$$y = \frac{\partial x}{\partial t} \tag{2.1}$$

jak i pojedyncze symbole x i y składa się w trybie matematycznym.

Rozdział 3

Wymagania i narzędzia

3.1 Narzędzia głębokiego uczenia

3.1.1 Przygotowanie danych wzorcowych

Do poprawnego wytrenowania sieci neuronowej, konieczne jest przygotowanie zestawu danych wzorcowych. W przypadku algorytmów głębokiego uczenia kluczowe znaczenie ma jakość i zróżnicowanie danych treningowych, ponieważ sieci neuronowe uczą się rozpoznawania wzorców na podstawie dostarczonych przykładów.

Dane wzorcowe muszą spełniać pewne istotne wymagania, aby zapewnić odpowiednią skuteczność modelu:

- **Różnorodność scenariuszy**: Obrazy powinny przedstawiać obiekt w różnych warunkach oświetleniowych (np. światło dzienne, sztuczne oświetlenie, cienie), co pozwala modelowi uogólniać na nowe środowiska.

- **Zmienne pozycje i orientacje**: Dane powinny zawierać obiekt w różnych orientacjach, kątach oraz pozycjach względem kamery, aby uniknąć problemu nadmiernego dopasowania do specyficznego układu.

- **Różne odległości od kamery**: Obiekty powinny być widoczne w kadrze zarówno z bliska, jak i z daleka, co ułatwia modelowi detekcję niezależnie od dystansu.

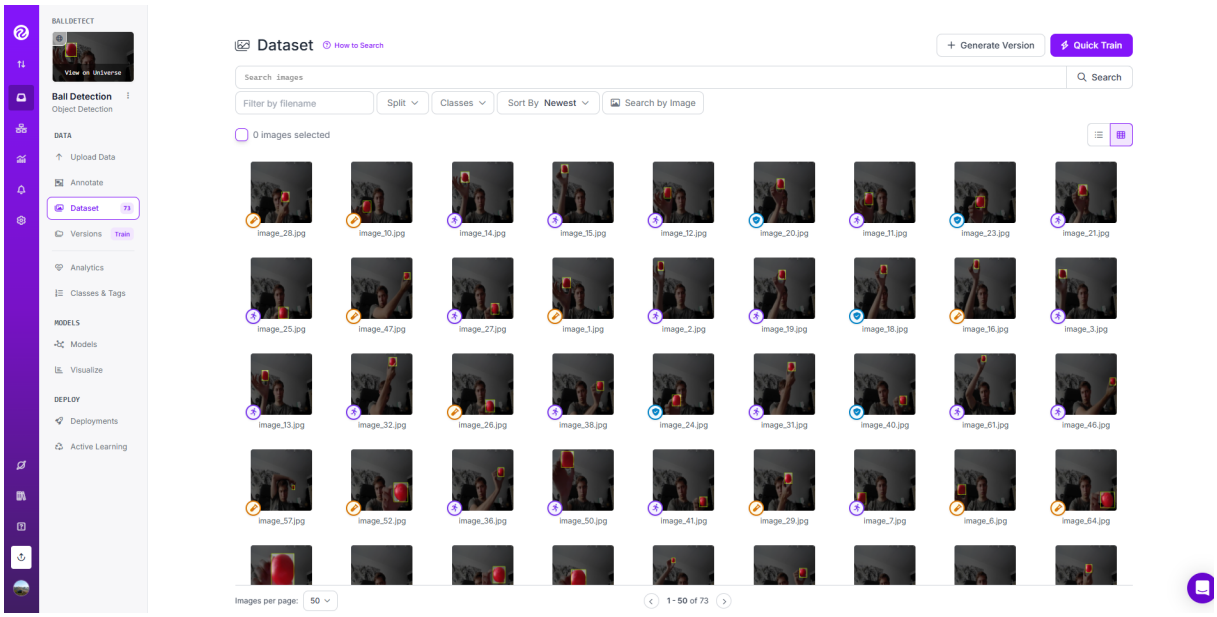
W praktyce przygotowanie takiego zbioru danych wymaga użycia narzędzi do annotacji, które pozwalają na oznaczenie obiektów w obrazach za pomocą ramek ograniczających(ang. *bounding boxes*). Do tego celu często wykorzystuje się narzędzia do annotacji zdjęć, takie jak **Roboflow**, **LabelImg**, **CVAT** czy **Makesense.ai**, które wspomagają proces zarządzania danymi, ich augmentacji (np. obrót, zmiana jasności, rozmycie) oraz podziału na zbiory treningowe, walidacyjne i testowe.

3.1.2 Platforma Roboflow

Aby zrealizować założenia pracy, do przygotowania zbioru uczącego wykorzystano platformę Roboflow. Jest to narzędzie online, które umożliwia zarządzanie danymi, ich etykietowanie oraz konwersję między różnymi formatami. W kontekście tej pracy wykorzystano obrazy przedstawiające czerwoną piłkę w realistycznym środowisku.

Proces przygotowania danych w Roboflow składał się z następujących etapów:

1. **Zbieranie danych** - zgromadzono zestaw około 100 zdjęć przedstawiających czerwoną piłkę w różnych warunkach oświetleniowych i na różnych tłach.
2. **Annotacja** - każde zdjęcie zostało oznaczone poprzez narysowanie ramki ograniczającej wokół piłki. Roboflow udostępnia intuicyjny interfejs do tego procesu, co znacznie przyspiesza pracę.
3. **Augmentacja danych** - aby zwiększyć różnorodność zbioru treningowego, zastosowano następujące techniki augmentacji:
   - Zmiana jasności
   - Rozmycie
   - Zróżnicowanie otoczenia
4. **Podział danych** - zbiór został podzielony w proporcjach:
   - 70% - zbiór treningowy
   - 20% - zbiór walidacyjny
   - 10% - zbiór testowy
5. **Eksport** - przygotowane dane zostały wyeksportowane w formacie YOLO v7, który jest kompatybilny z wybranym modelem detekcji obiektów.



Rysunek 3.1: Widok strony głównej witryny Roboflow.

3.2 PyTorch

PyTorch to jedna z bardziej popularnych bibliotek uczenia maszynowego, opracowanej przez Meta AI i obecnie wspieraną przez PyTorch Foundation. Biblioteka została zaprojektowana z myślą o intuicyjności, wydajności i elastyczności.

Główne cechy PyTorch

- **Dynamiczne grafy obliczeniowe**: Dzięki dynamicznemu budowaniu grafów obliczeniowych, PyTorch pozwala na elastyczne definiowanie i modyfikowanie modeli w trakcie ich działania [**bib:paszke2019pytorch**].
- **Obliczenia tensorowe z akceleracją GPU**: Biblioteka zapewnia wsparcie dla obliczeń tensorowych z wykorzystaniem GPU, co umożliwia szybkie przetwarzanie dużych zbiorów danych [**paszke2019pytorch**].
- **Wsparcie dla głębokich sieci neuronowych**: Moduł `torch.nn` ułatwia definiowanie i trenowanie złożonych modeli głębokiego uczenia.

Zastosowania PyTorch PyTorch jest szeroko wykorzystywany zarówno w badaniach naukowych, jak i w przemyśle. Do jego głównych zastosowań należą:

- **Wizja komputerowa**: PyTorch jest wykorzystywany do trenowania modeli detekcji obiektów, takich jak YOLOv7, oraz innych architektur CNN.
- **Przetwarzanie języka naturalnego (NLP)**: Biblioteki takie jak Hugging Face Transformers, używane do przetwarzania języka, opierają się na PyTorch.

- **Autonomiczne systemy**: PyTorch znajduje zastosowanie w systemach autonomicznych, takich jak systemy nawigacji dla robotów mobilnych.

Społeczność i rozwój PyTorch posiada jedną z najbardziej aktywnych społeczności w obszarze uczenia maszynowego. Dzięki otwartemu modelowi rozwoju, społeczność regularnie wprowadza nowe funkcje, poprawki błędów oraz dedykowane narzędzia, takie jak PyTorch Lightning, które ułatwiają proces trenowania modeli.

Podsumowanie Dzięki intuicyjności, wszechstronności i wysokiej wydajności, PyTorch stał się jednym z najważniejszych narzędzi w dziedzinie głębokiego uczenia. W tej pracy PyTorch został wykorzystany jako podstawowa biblioteka do implementacji algorytmu YOLO oraz zarządzania procesami trenowania modeli.

3.2.1 Algorytm YOLO

Algorytm YOLO (*You Only Look Once*) został zaprojektowany jako zintegrowany system detekcji obiektów w czasie rzeczywistym, przekształcając problem detekcji w zadanie regresji. YOLO stosuje jedną sieć neuronową do analizy całego obrazu i równoczesnego przewidywania współrzędnych ramek ograniczających oraz klas obiektów [**redmon2016yolo**].

Założenia projektowe Model YOLO dzieli obraz na siatkę $S \times S$, gdzie każda komórka siatki przewiduje:

- B ramek ograniczających (*bounding boxes*) wraz z ich współrzędnymi (x, y, w, h),
- prawdopodobieństwo obecności obiektu w każdej ramce,
- prawdopodobieństwo klasowe dla każdego obiektu.

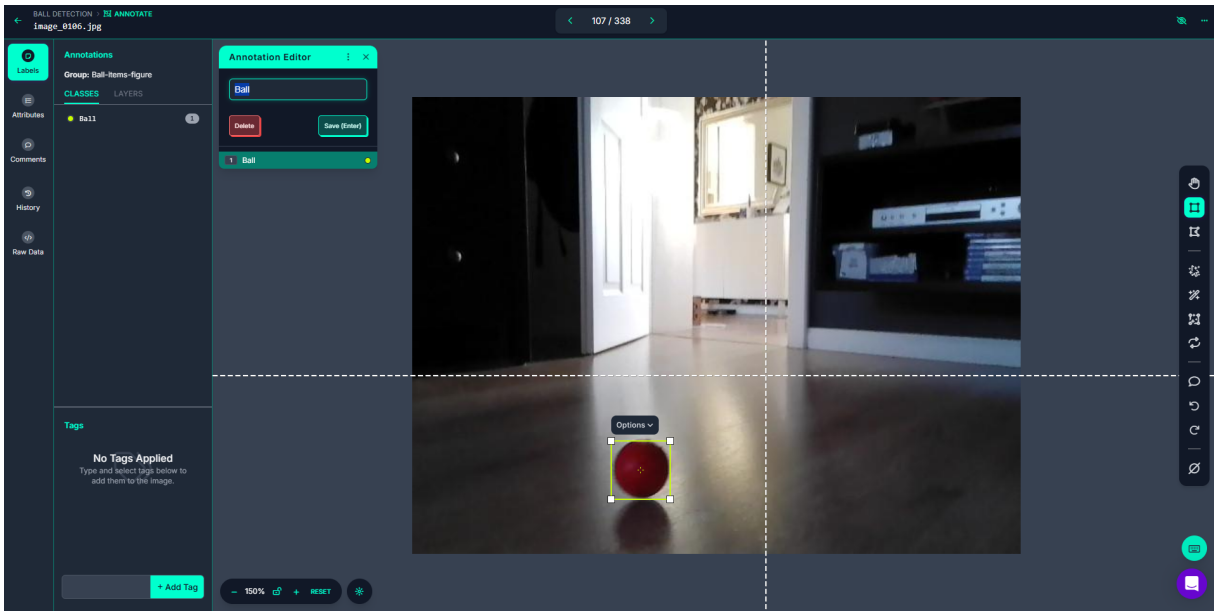
Łączne przewidywania są zapisywane jako tensor o wymiarze $S \times S \times (B \cdot 5 + C)$, gdzie C to liczba klas [1].

Architektura sieci YOLO Sieć YOLO opiera się na warstwach konwolucyjnych, w których kluczowe znaczenie ma redukcja wymiarów za pomocą warstw 1×1 i 3×3. W wersji YOLOv7 wprowadzono szereg usprawnień, takich jak:

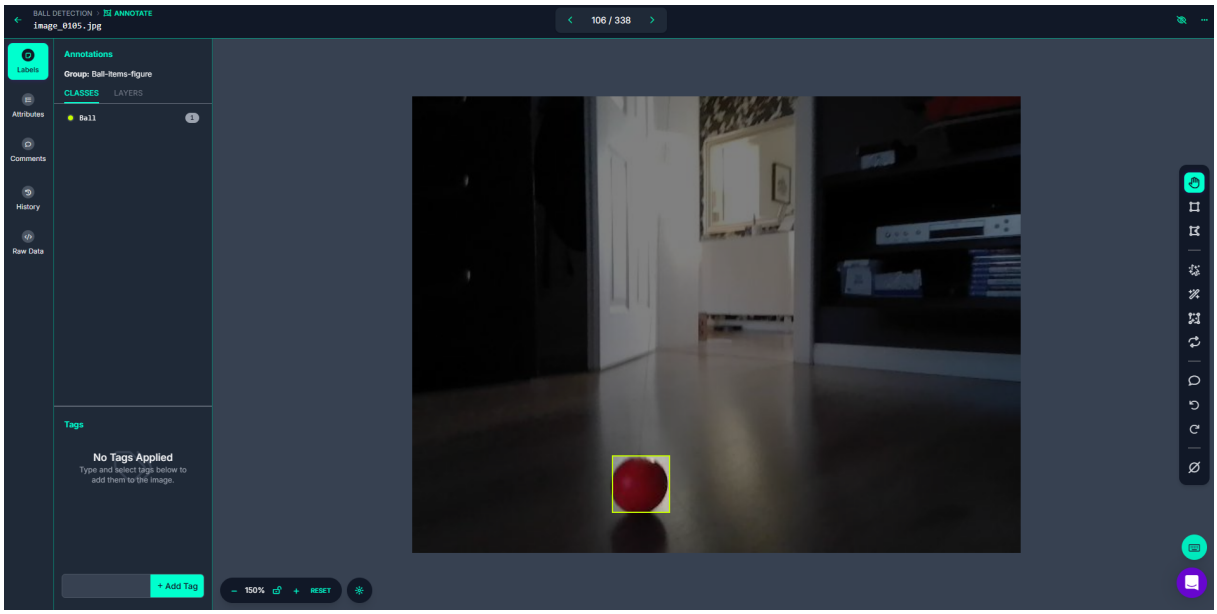
- użycie modułów re-parametryzowanych (*re-parameterized convolution*) dla poprawy propagacji gradientu,
- lepsza agregacja cech w warstwach za pomocą strategii E-ELAN (*Extended Efficient Layer Aggregation Networks*),
- optymalizacja procesu uczenia poprzez dynamiczne przypisywanie etykiet (*coarse-to-fine label assignment*) [2].

W ramach tej pracy YOLO zostało wykorzystane do detekcji obiektów w systemie wizyjnym robota mobilnego. Model YOLOv7, dzięki swojej optymalnej architekturze i wydajności, umożliwił realizację systemu wizyjnego z zachowaniem wymagań ograniczonej mocy obliczeniowej platformy Raspberry Pi.

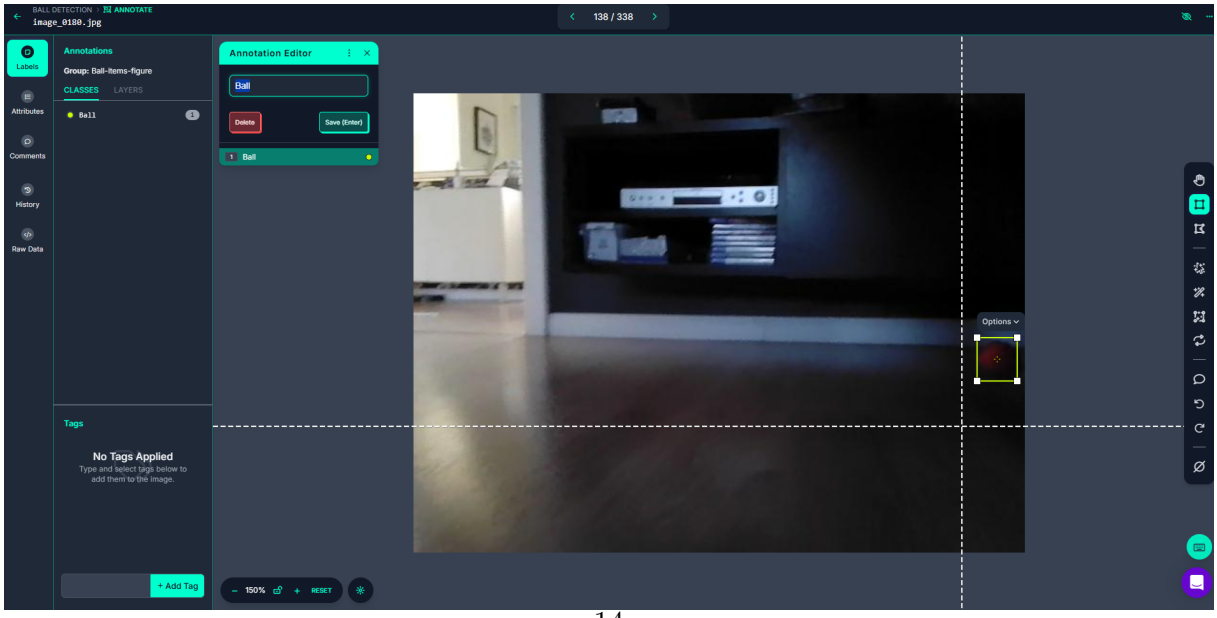
3.3 specyfikacja techniczna



(a) Etykietowanie obiektów w Roboflow.



(b) Obiekt po zetykietowaniu



(c) Etykietowanie w słabym oświetleniu

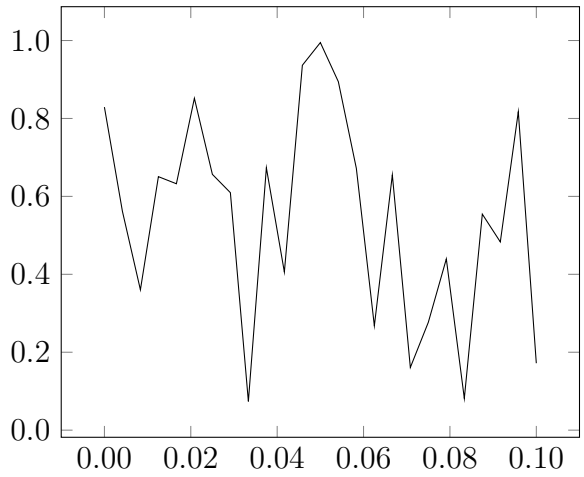
Rysunek 3.2: Proces użycia witryny Roboflow.

Rozdział 4

[Właściwy dla kierunku – np. Specyfikacja zewnętrzna]

Jeśli „Specyfikacja zewnętrzna”:

- wymagania sprzętowe i programowe
- sposób instalacji
- sposób aktywacji
- kategorie użytkowników
- sposób obsługi
- administracja systemem
- kwestie bezpieczeństwa
- przykład działania
- scenariusze korzystania z systemu (ilustrowane zrzutami z ekranu lub generowanymi dokumentami)



Rysunek 4.1: Podpis rysunku po rysunkiem.

Rozdział 5

[Właściwy dla kierunku – np. Specyfikacja wewnętrzna]

Jeśli „Specyfikacja wewnętrzna”:

- przedstawienie idei
- architektura systemu
- opis struktur danych (i organizacji baz danych)
- komponenty, moduły, biblioteki, przegląd ważniejszych klas (jeśli występują)
- przegląd ważniejszych algorytmów (jeśli występują)
- szczegóły implementacji wybranych fragmentów, zastosowane wzorce projektowe
- diagramy UML

Krótka wstawka kodu w linii tekstu jest możliwa, np. `int a;` (biblioteka `listings`). Dłuższe fragmenty lepiej jest umieszczać jako rysunek, np. kod na rys 5.1, a naprawdę długie fragmenty – w załączniku.

```
1  class test : public basic
2  {
3      public:
4        test (int a);
5        friend std::ostream operator<<(std::ostream & s,
6                                       const test & t);
7      protected:
8        int _a;
9
10 };
```

Rysunek 5.1: Pseudokod w `listings`.

Rozdział 6

Weryfikacja i walidacja

- sposób testowania w ramach pracy (np. odniesienie do modelu V)
- organizacja eksperymentów
- przypadki testowe zakres testowania (pełny/niepełny)
- wykryte i usunięte błędy
- opcjonalnie wyniki badań eksperymentalnych

Tabela 6.1: Nagłówek tabeli jest nad tabelą.

	metoda						
			alg. 3			alg. 4, $\gamma = 2$	
ζ	alg. 1	alg. 2	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 3$	$\beta = 0.1$	$\beta = -0.1$
0	8.3250	1.45305	7.5791	14.8517	20.0028	1.16396	1.1365
5	0.6111	2.27126	6.9952	13.8560	18.6064	1.18659	1.1630
10	11.6126	2.69218	6.2520	12.5202	16.8278	1.23180	1.2045
15	0.5665	2.95046	5.7753	11.4588	15.4837	1.25131	1.2614
20	15.8728	3.07225	5.3071	10.3935	13.8738	1.25307	1.2217
25	0.9791	3.19034	5.4575	9.9533	13.0721	1.27104	1.2640
30	2.0228	3.27474	5.7461	9.7164	12.2637	1.33404	1.3209
35	13.4210	3.36086	6.6735	10.0442	12.0270	1.35385	1.3059
40	13.2226	3.36420	7.7248	10.4495	12.0379	1.34919	1.2768
45	12.8445	3.47436	8.5539	10.8552	12.2773	1.42303	1.4362
50	12.9245	3.58228	9.2702	11.2183	12.3990	1.40922	1.3724

Rozdział 7

Podsumowanie i wnioski

- uzyskane wyniki w świetle postawionych celów i zdefiniowanych wyżej wymagań
- kierunki ewentualnych danych prac (rozbudowa funkcjonalna . . .)
- problemy napotkane w trakcie pracy

Bibliografia

[1] Joseph Redmon, Santosh Divvala, Ross Girshick i Ali Farhadi. „You Only Look Once: Unified, Real-Time Object Detection". W: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 2016 (2016). DOI: 10.1109/CVPR.2016.91, s. 779–788. URL: https://arxiv.org/abs/1506.02640.

[2] Chien-Yao Wang, Alexey Bochkovskiy i Hong-Yuan Mark Liao. „YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". W: *arXiv preprint arXiv:2207.02696* (2022). Available online: https://arxiv.org/abs/2207.02696. URL: https://arxiv.org/abs/2207.02696.

Dodatki

Spis skrótów i symboli

DNA kwas deoksyrybonukleinowy (ang. *deoxyribonucleic acid*)

MVC model – widok – kontroler (ang. *model–view–controller*)

N liczebność zbioru danych

μ stopnień przyleżności do zbioru

$\mathbb{E}$ zbiór krawędzi grafu

$\mathcal{L}$ transformata Laplace'a

Źródła

Jeżeli w pracy konieczne jest umieszczenie długich fragmentów kodu źródłowego, należy je przenieść w to miejsce.

```
1 if (_nClusters < 1)
2     throw std::string ("unknown number of clusters");
3 if (_nIterations < 1 and _epsilon < 0)
4     throw std::string ("You should set a maximal number of
          iteration or minimal difference — epsilon.");
5 if (_nIterations > 0 and _epsilon > 0)
6     throw std::string ("Both number of iterations and minimal
          epsilon set — you should set either number of iterations
          or minimal epsilon.");
```

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,
- dane testowe,
- film pokazujący działanie opracowanego oprogramowania lub zaprojektowanego i wykonanego urządzenia,
- itp.

Spis rysunków

Spis tabel