

Technische Hochschule Deggendorf
Faculty Applied Information Technology
Studiengang Bachelor of Cyber-Security

Projektsimplementierung Reflexion zum Projekt ”WeedDetector”

Vorgelegt von:

Christof Renner
Matrikelnummer: 22301943
Manuel Friedl
Matrikelnummer: 1236626

Prüfungsleitung:

Prof. Dr. Holger Jehle

Am: 11. Mai 2025

Contents

1	Controller-Implementierung	3
2	Model & Modeltraining	4
3	Security Scanning & Linting	5
3.1	Security Scanning mit Bandit	5
3.2	Codequalität mit Pylint	5
4	Docker & Deployment	7
4.1	Containerisierung	7
4.2	Automatisiertes Deployment	7
4.3	Dependency-Management	8
5	Fazit	9

1 Controller-Implementierung

Die WeedDetector-Anwendung folgt dem Model-View-Controller (MVC) Entwurfsmuster, wobei der `WeedDetectorController` als zentraler Koordinator zwischen GUI und Erkennungsmodell fungiert.

Die Hauptaufgaben des Controllers umfassen:

- Ereignisbasierte Kommunikation zwischen GUI und Modell
- Verarbeitung von Benutzeraktionen (Bildauswahl, Erkennung, Robotersteuerung)
- Implementierung robuster Fehlerbehandlungsmechanismen

Die Kommunikation erfolgt über Callback-Funktionen, die eine lose Kopplung der Komponenten ermöglichen:

Listing 1.1: Callback-Registrierung

```
1 # GUI-Callbacks werden im Konstruktor registriert
2 self.gui.on_select_image = self.handle_select_image
3 self.gui.on_detect = self.handle_detect
4 self.gui.on_start_robot = self.handle_start_robot
5 self.gui.on_stop_robot = self.handle_stop_robot
```

2 Model & Modeltraining

Der WeedDetector nutzt ein YOLO-Modell für die Unkrauterkenennung aufgrund seiner exzellenten Balance aus Geschwindigkeit und Genauigkeit. Die Entscheidung basiert auf einem umfassenden Vergleich verschiedener Objekterkennungsalgorithmen.

Die `WeedDetectorModel`-Klasse implementiert:

- Automatische Modellsuche mit Fallback-Mechanismus
- Konfigurierbare Erkennungsparameter (Konfidenz, IoU)
- Methoden für Training, Inferenz und Ergebnisvisualisierung

Der Trainingsprozess nutzt die Ultralytics-YOLO-API und einen umfangreichen Datensatz mit über 13.000 annotierten Bildern:

Listing 2.1: Training

```
1 def train(self, train_data_path, epochs=50):  
2     """Train the YOLO model with provided training data."""  
3     self.model.train(data=train_data_path, epochs=epochs)
```

Die Trainingsdaten wurden mit verschiedenen Augmentierungstechniken aufbereitet:

- Horizontale Spiegelung (50% Wahrscheinlichkeit)
- Zufällige Rotation und Bildausschnitte
- Belichtungsanpassungen

3 Security Scanning & Linting

3.1 Security Scanning mit Bandit

Für kontinuierliche Sicherheitsüberprüfungen wird Bandit eingesetzt, ein auf Python spezialisiertes Security-Scanning-Tool. Die Integration erfolgt über GitHub Actions:

Listing 3.1: Security Scan Workflow

```
1 name: Security Scan
2 on:
3   push:
4     branches: [ main ]
5   pull_request:
6     branches: [ main ]
7 jobs:
8   bandit:
9     name: Bandit Scan
10    runs-on: ubuntu-latest
11    steps:
12      - uses: actions/checkout@v3
13      - uses: actions/setup-python@v4
14        with: { python-version: '3.x' }
15      - run: pip install bandit
16      - run: bandit -r . --skip trojansource
```

Bandit analysiert den Code auf Sicherheitsprobleme wie:

- Injektionsanfälligkeiten
- Hartcodierte Credentials
- Unsichere Funktionsaufrufe

3.2 Codequalität mit Pylint

Zur Sicherstellung konsistenter Codequalität wird Pylint als statisches Analysetool eingesetzt:

Listing 3.2: Pylint Workflow

```
1 name: Pylint
2 on: [push]
3 jobs:
4   build:
5     runs-on: ubuntu-latest
6     strategy:
7       matrix: { python-version: ["3.13"] }
8     steps:
9       - uses: actions/checkout@v4
10      - uses: actions/setup-python@v5
```

```
11     with: { python-version: ${{ matrix.python-version }} }
12     - run: pip install pylint
13     - run: pylint $(git ls-files '*.py')
```

Pylint prüft PEP 8 Konformität, Dokumentationsqualität, Komplexitätsmetriken und potenzielle Bugs.

4 Docker & Deployment

4.1 Containerisierung

Die Anwendung wird in Docker containerisiert, was eine konsistente Laufzeitumgebung sicherstellt. Die Dockerfile-Qualität wird mit Hadolint überprüft:

Listing 4.1: Docker Lint

```
1 name: Docker Build and Test
2 on:
3   push:
4     paths: [ 'Dockerfile' ]
5   pull_request:
6     paths: [ 'Dockerfile' ]
7 jobs:
8   lint-dockerfile:
9     runs-on: ubuntu-latest
10    steps:
11      - uses: actions/checkout@v4
12      - uses: hadolint/hadolint-action@v3.1.0
13      with: { dockerfile: Dockerfile }
```

Der Container umfasst die Python-Laufzeitumgebung, YOLO- und OpenCV-Bibliotheken sowie die Anwendungskomponenten.

4.2 Automatisiertes Deployment

Das Deployment erfolgt automatisiert über GitHub Actions mit Integration in Docker Hub:

Listing 4.2: Build and Publish

```
1 name: Build and Publish Docker image
2 on:
3   push:
4     branches: [ main ]
5     paths: [ 'Dockerfile' ]
6 jobs:
7   build-and-push:
8     runs-on: ubuntu-latest
9     steps:
10      - uses: actions/checkout@v4
11      - uses: docker/setup-buildx-action@v3
12      - uses: docker/login-action@v3
13      with:
14        username: ${ secrets.DOCKERHUB_USERNAME }
15        password: ${ secrets.DOCKERHUB_TOKEN }
16      - uses: docker/build-push-action@v6
17      with:
```

```
18     context: .
19     push: true
20     tags: crnnr/weeddetector:latest
```

4.3 Dependency-Management

Dependabot übernimmt die automatische Aktualisierung von Abhängigkeiten für Python-Pakete, Docker-Images und GitHub Actions.

Das Projekt unterstützt verschiedene Deploymentstrategien:

- Lokales Deployment für Entwicklung
- Container-basiertes Deployment für Produktion
- Versionierte Releases mit automatischen Updates

5 Fazit

Die Implementierung des WeedDetector-Projekts demonstriert die erfolgreiche Integration moderner Softwareentwicklungspraktiken:

- MVC-Architektur für strukturierte Komponentenorganisation
- YOLO-basierte Objekterkennung für präzise Unkrautidentifikation
- Automatisierte Sicherheits- und Codequalitätsprüfungen
- Containerisierung für plattformübergreifende Kompatibilität
- CI/CD-Pipeline für kontinuierliche Integration und Deployment

Diese Kombination ermöglicht eine robuste, wartbare und sichere Anwendung für die Unkrauterkennung mit minimalen Ressourcenanforderungen und hoher Erkennungsgenauigkeit.