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

Projekt "WeedDetector" - Eigenanteil Manuel Friedl

Vorgelegt von:

Manuel Friedl (12306626)

Prüfungsleitung:

Prof. Dr. Holger Jehle

Datum: 05. Juli 2025

Contents

1 Einleitung

In diesem Abschnitt wird mein persönlicher Beitrag zum Projekt **"Weed Detector"** im Rahmen des Software Engineering Kurses dargestellt. Ziel ist es, die von mir übernommenen Aufgaben sowie meinen Anteil an der Planung, Umsetzung und Optimierung des Projekts transparent zu machen.

Dabei werden sowohl die **technischen Aufgaben**, die ich im Verlauf des Projekts übernommen habe, als auch **meine Beiträge zur Zusammenarbeit im Team** beschrieben. So wird nachvollziehbar, welche Bereiche des Weed Detectors maßgeblich von mir entwickelt oder betreut wurden und wie mein Beitrag zur erfolgreichen Fertigstellung des Projekts beigetragen hat.

2 Implementierung des Graphical User Interfaces

Ein wesentlicher Teil meines Beitrags war die Entwicklung und Implementierung der grafischen Benutzeroberfläche (GUI) für den Weed Detector unter Verwendung von Tkinter in Python.

Bei der Umsetzung habe ich das MVC-Designpattern (Model-View-Controller) angewendet, um die Struktur und Wartbarkeit des Codes zu verbessern. Die View umfasst die Tkinter-Oberfläche mit den Steuerelementen und der Visualisierung der Unkrauterkennung. Die Controller-Logik verbindet die Benutzerinteraktionen mit der Modelllogik zur Bildanalyse und Steuerung der Demo-Funktionen, während das Model die Ergebnisverarbeitung und Steuerung der Detektion beinhaltet.

2.1 Zentrale Aufgaben der GUI

Die **GUI des Weed Detectors** übernimmt folgende zentrale Aufgaben:

- **Bilddateien auswählen:** Ermöglicht das Laden von gespeicherten Bildern zur Analyse durch das YOLOv8-Modell.
- **Live-Kamera starten:** Aktiviert die integrierte oder angeschlossene Kamera zur Echtzeiterkennung von Unkraut.
- **Roboter-Demo starten:** Startet die simulierte Roboterbewegung auf einer vordefinierten Strecke zur Demonstration des automatisierten Erkennungs- und Bekämpfungsprozesses.
- **Visualisierung der Ergebnisse:** Zeigt die Erkennungsergebnisse direkt in der GUI an, indem erkannte Unkräuter markiert und hervorgehoben werden.
- **Einfache Bedienung und Steuerung:** Bietet eine benutzerfreundliche Schnittstelle zur Steuerung des Weed Detectors ohne manuelle Eingriffe im Code.
- Status- und Fortschrittsanzeige: Informiert den Benutzer über den aktuellen Status von Analyse und Roboterbetrieb, um den Ablauf nachvollziehbar zu gestalten.

Diese Aufgaben ermöglichen eine effiziente und intuitive Nutzung des Weed Detectors, sodass alle Funktionen zentral zugänglich und die Ergebnisse der Unkrauterkennung leicht überprüfbar sind.

2.2 Prozess der GUI

Nach der Anforderungsanalyse begann die Entwicklung der GUI zunächst mit einem einfachen Fenster und zwei Buttons (Bild hochladen, Live-Kamera starten) als ersten Prototypen, um eine funktionale Basis zu schaffen und die Kernfunktionen frühzeitig testen

zu können.

Im nächsten Schritt wurde die GUI vollständig refactored, um eine anspruchsvollere, optisch ansprechendere und nahtlose Benutzererfahrung zu ermöglichen. Dies umfasste die Anpassung des Layouts, eine verbesserte Strukturierung der Elemente sowie die Integration eines einheitlichen Designs.

Anschließend wurde die GUI nach dem MVC-Designpattern umgebaut, um die Struktur und Wartbarkeit zu verbessern. Hierbei wurden Call-Back-Funktionen eingesetzt, um Benutzerinteraktionen effizient mit der Verarbeitungslogik und Ergebnisanzeige zu verbinden.

Die Ergebnisse der Unkrauterkennung werden nun sowohl visuell im Kamerafenster als auch in einem Result Panel angezeigt, während Aktionen des simulierten Roboters in einem Roboter Status Panel ausgegeben werden, sodass der Nutzer den gesamten Prozess nachvollziehen kann.

Zum Abschluss wurden Tests geschrieben und Optimierungen durchgeführt, um die Stabilität, Performance und Benutzerfreundlichkeit der GUI sicherzustellen.

2.2.1 Callbacks in der GUI

In der Tkinter-GUI werden Callbacks verwendet, um Benutzeraktionen mit Funktionen zu verknüpfen. Ein Callback ist dabei eine Funktion, die automatisch ausgeführt wird, sobald der Benutzer eine bestimmte Aktion ausführt, wie z.B. das Klicken eines Buttons.

Listing 2.1: Callback-Definition

```
1     self.on_select_image = None # Callback for image selection
2     self.on_detect = None # Callback for detection
3     self.on_start_robot = None  # Callback for robot start
4     self.on_stop_robot = None  # Callback for robot stop
5     self.on_camera_frame = None # Callback for camera frame processing
```

Listing 2.2: Beispiel Callback-Verwendung

```
1     ...
2     if file_path and callable(self.on_select_image):
3             self.on_select_image(file_path)
```

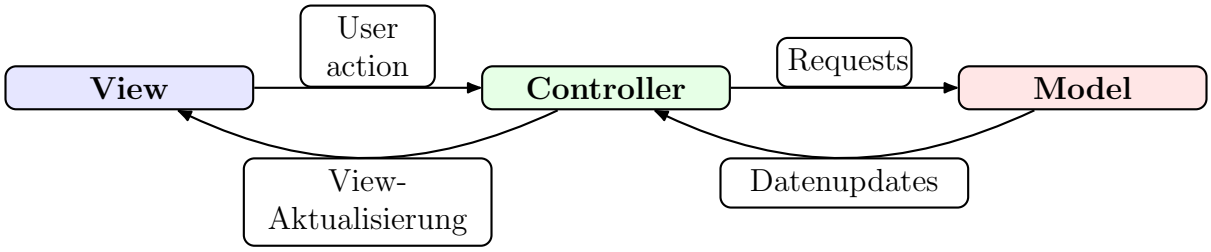

Figure 2.1: MVC-Architektur der WeedDetector-Anwendung

3 Roboter und Navigation

Ein weiterer zentraler Beitrag von mir war die Implementierung des *"Demo"*-Roboters, welcher den realen Einsatz eines Unkrauterkennungsroboters simuliert. Ziel war es, die Prozesskette von der Erkennung bis zur gezielten Bekämpfung in einer sicheren Testumgebung realistisch darzustellen.

Hierfür wurde eine **Robot**-Klasse entwickelt, die folgende Kernfunktionen umfasst:

- **start_robot:** Startet den Roboter, aktiviert die Kamera und initiiert den Fahrprozess in einem eigenen Thread, sodass die GUI reaktiv bleibt.
- **stop_robot:** Stoppt den Roboter und beendet den Fahrprozess sicher.
- **drive_loop:** Führt den Fahrvorgang in einer Schleife aus, liest kontinuierlich Bilder von der Kamera ein, führt die Unkrauterkennung mit YOLOv8 durch und zeigt die Ergebnisse live in der GUI an. Wird Unkraut erkannt, wird der Bekämpfungsprozess ausgelöst.
- **eliminate_weeds:** Simuliert die Bewegung des Roboterarms zu den erkannten Unkrautpositionen und die gezielte Eliminierung mit einer Spritze.

Die Implementierung verwendet Threads, um den Fahrvorgang unabhängig von der GUI auszuführen, wodurch eine flüssige Bedienung ohne Blockierungen möglich ist. Über ein Roboter-Status-Panel in der GUI werden alle Aktionen (z. B. *"Robot is driving forward“, "Weed detected“, "Weed eliminated“*) transparent protokolliert, sodass der Nutzer den Prozess nachvollziehen kann.

Durch die Entwicklung dieses DemoRoboters wurde eine realitätsnahe Simulation des autonomen Betriebs des Weed Detectors realisiert, die in der Lehre und für Präsentationen anschaulich die Funktionalität und das Zusammenspiel von Erkennung und gezielter Unkrautbekämpfung demonstriert.

Listing 3.1: Auszug Robot-Klasse

```
1  class Robot:
2      """Controller for the automation robot that can be controlled via
           the GUI."""
3      def __init__(self, gui, model):
4          self.gui = gui
5          self.model = model
6          self.is_running = False
7          self.thread = None
8
9      def start_robot(self):
10         """Start the robot."""
11         self.is_running = True
```

```
        self.gui.toggle_camera()
        self.gui.log_robot_action("Robot started")
        self.thread = threading.Thread(target=self.drive_loop, daemon=
            True)
        self.thread.start()
        ...
```

4 Unit-, Integration und Systemtests

4.1 Unit-Testing

m Rahmen des Projekts habe ich **Unit-Tests** entwickelt und durchgeführt, um sicherzustellen, dass die einzelnen Funktionen des Weed Detectors korrekt und stabil arbeiten.

Dabei wurden gezielt isolierte Funktionen getestet, darunter:

- Bildvorverarbeitung und Bildladefunktionen
- Ergebnisinterpretation und Auswertung der YOLOv8-Outputs
- Callbacks innerhalb der GUI (z.B. Button-Handler)
- Methoden innerhalb der Robot-Klasse wie *eliminate_weeds* und Zustandswechsel bei Start und Stopp

Für die Tests wurde das **unittest-Framework** von Python verwendet, wodurch automatisierte Testläufe möglich wurden. Die Tests wurden so gestaltet, dass auch fehlerhafte Eingaben und Grenzfälle überprüft wurden, um die Robustheit des Systems zu erhöhen.

Durch das Unit-Testing konnte sichergestellt werden, dass Fehler frühzeitig erkannt und behoben wurden, wodurch die Stabilität und Qualität des Projekts nachhaltig verbessert wurde.

4.2 Integration-Testing

Für den Weed Detector wurden **Integrationstests** durchgeführt, um sicherzustellen, dass GUI, YOLOv8-Modell und Robot-Klasse fehlerfrei zusammenarbeiten. Dabei wurde geprüft, ob Bildaufnahme, Unkrauterkennung und Ergebnisanzeige korrekt ineinandergreifen und Benutzeraktionen zuverlässig verarbeitet werden.

Die Tests stellten sicher, dass alle Module nahtlos funktionieren und der Weed Detector stabil im Gesamtsystem läuft.

4.3 Systemtest

Es wurde ein Systemtest durchgeführt, bei dem der Weed Detector vom Start über die Unkrauterkennung bis zur simulierten Bekämpfung mit dem DemoRoboter im Gesamtablauf getestet wurde.

Dabei wurde überprüft, ob alle Komponenten zusammen stabil arbeiten und der gesamte Prozess zuverlässig abläuft.