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

Software Engineering

Übersicht der Arbeitsaufteilung

Contents

1 Einleitung

Im Rahmen des Kurses **Software Engineering** wurde die Aufgabe gestellt, einen Weed Detector zu entwickeln, der Unkraut in landwirtschaftlichen Flächen automatisch erkennen kann. Ziel dieses Projekts ist es, praxisnah die im Kurs erlernten Methoden zur Planung, Umsetzung und Dokumentation eines Softwareprojekts anzuwenden.

Das Projekt wird von Christof Renner und Manuel Friedl im Team durchgeführt. Bereits zu Projektbeginn war bekannt, dass Jonas Gassner nicht mehr teilnehmen wird, weshalb das Projekt von Anfang an ohne ihn geplant und umgesetzt wurde.

In diesem Dokument wird die Arbeitsaufteilung innerhalb des Teams klar und detailliert dargestellt, um Zuständigkeiten, individuelle Beiträge und die bearbeiteten Arbeitsschritte jedes Teammitglieds transparent aufzuzeigen. Dies schafft eine nachvollziehbare Grundlage für die Organisation im Team und ermöglicht eine faire und strukturierte Evaluierung der geleisteten Arbeit.

2 Arbeitsaufteilung

Beteiligte	**Manuel**	**Christof**
Model und Modeltraining	20%	80%
Docker-Container	–	100%
GUI	80%	20%
Roboter u. Navigation	100%	–
Refactor Codebase (MVC)	100%	–
Security-Scanning	–	100%
Tests	100%	–
Linting	50%	50%
Deployment (CI/CD-Pipelines)	–	100%
Dokumentation erstellen	50%	50%
Präsentation erstellen	50%	50%
Insgesamt:	**50%**	**50%**

3 Arbeitspunkte

3.1 Model und Modeltraining

Für die Unkrauterkennung wurde das YOLOv8-Modell verwendet, da es eine schnelle und präzise Objekterkennung ermöglicht. Nach der Annotation der Trainingsdaten wurde das Modell trainiert und anhand von mAP, Precision und Recall evaluiert. Das trainierte Modell wurde exportiert und für die Echtzeiterkennung auf der Zielhardware eingesetzt, wodurch eine zuverlässige Unkrauterkennung im Weed Detector erreicht wurde.

3.2 Docker-Container einrichten

Für den Weed Detector wurde ein Docker-Container eingerichtet, um eine einheitliche und portable Ausführungsumgebung bereitzustellen. Darin wurden alle benötigten Abhängigkeiten, das YOLOv8-Modell und die Laufzeitumgebung integriert, sodass das System unabhängig vom Host-System zuverlässig ausgeführt werden kann.

Der Einsatz von Docker erleichtert die Installation, Wartung und den Transfer des Projekts auf andere Geräte und ermöglicht eine konsistente Entwicklungs- und Testumgebung.

3.3 Graphical User Interface

Für den Weed Detector wurde ein Graphical User Interface (GUI) mit **Tkinter** in Python entwickelt, um die Bedienung des Systems intuitiv und effizient zu gestalten. Die Benutzeroberfläche verfügt über drei Buttons:

- **Bilddateien auswählen:** zum Hochladen und Analysieren gespeicherter Bilder
- **Kamera starten:** zur Live-Erkennung von Unkraut mit der integrierten- oder angeschlossenen Kamera
- **Roboter starten:** zum Aktivieren des Roboters für die automatische Unkrauterkennung und -bearbeitung im Feld/Beet

Die Ergebnisse der Unkrauterkennung werden dabei direkt im GUI angezeigt, indem erkannte Pflanzen und Unkraut in Echtzeit visualisiert und farblich markiert werden. Dies ermöglicht eine einfache Überwachung des Systems, während gleichzeitig alle notwendigen Steuerungsfunktionen zentral zugänglich bleiben.

Durch den Einsatz von Tkinter konnte eine leichtgewichtige und plattformunabhängige Lösung realisiert werden, die sich nahtlos in den Arbeitsablauf mit dem Weed Detector integriert.

3.4 Roboter und Navigation

Für den Weed Detector wurde ein simulierter Roboter entwickelt, der in einer vordefinierten Strecke fährt, um den praktischen Einsatz eines autonomen Unkrautbekämpfungssystems zu demonstrieren. Der Roboter bewegt sich dabei automatisch innerhalb der festgelegten Route und nutzt das integrierte Kamerabild, um in Echtzeit Unkraut zu erkennen.

Wird während der Fahrt Unkraut detektiert, steuert der Roboter einen simulierten Roboterarm an, an dem eine Spritze mit Unkrautbekämpfungsmittel befestigt ist. Diese „spritzt“ gezielt auf die erkannten Unkräuter, um eine punktgenaue Bekämpfung zu zeigen und den Einsatz von Herbiziden möglichst effizient darzustellen.

Die Navigation des Roboters, die Erkennung und die gezielte Ausführung der Sprühvorgänge laufen automatisiert ab und werden über das GUI gestartet und überwacht. So kann die komplette Prozesskette — von der Erkennung bis zur Bekämpfung — in einer sicheren Demo-Umgebung getestet und veranschaulicht werden.

3.5 Refactor Codebase (MVC)

m Verlauf des Projekts wurde die Codebase des Weed Detectors nach dem MVC-Designpattern (Model-View-Controller) refaktoriert, um Struktur und Wartbarkeit des Codes zu verbessern.

- **Model:** Beinhaltet die Logik zur Unkrauterkennung mit YOLOv8 sowie die Datenverarbeitung und Ergebnisverwaltung.
- **View:** Wird durch das mit Tkinter erstellte GUI repräsentiert, das die Benutzerinteraktion und Visualisierung der Erkennungsergebnisse übernimmt.
- **Controller:** Verknüpft die GUI mit der Erkennungs- und Steuerungslogik und steuert die Abläufe wie Bildauswahl, Kamerasteuerung und Roboterstart.

Durch diese Trennung wurden die Zuständigkeiten im Code klar abgegrenzt, wodurch Änderungen an der Oberfläche, der Erkennung oder der Steuerung unabhängig

voneinander durchgeführt werden können. Zudem erleichtert die Struktur die Erweiterung um neue Funktionen und verbessert die Übersichtlichkeit für künftige Arbeiten am Weed Detector.

3.6 Security-Scanning

Zur Verbesserung der Codequalität und Sicherheit wurde für den Weed Detector **Bandit** eingesetzt, ein Tool zur statischen Analyse von Python-Code auf Sicherheitslücken.

Bandit überprüfte die Codebase automatisiert auf potenzielle Schwachstellen wie unsichere Funktionen, fehlerhafte Verwendung von Bibliotheken oder mögliche Angriffspunkte im Umgang mit Dateipfaden und Benutzereingaben.

Durch den Einsatz von Bandit konnten Sicherheitsrisiken frühzeitig erkannt und behoben werden, wodurch die Stabilität und Sicherheit des Weed Detectors für den praktischen Einsatz erhöht wurde.

3.7 Unit-, Integration und Systemtests

Zur Sicherstellung der Funktionalität und Stabilität des Weed Detectors wurden Unit-, Integrations- und Systemtests durchgeführt.

- **Unit-Tests:** Einzelne Funktionen, wie Bildvorverarbeitung, Ergebnisinterpretation und Kamerazugriff, wurden isoliert getestet, um sicherzustellen, dass sie erwartungsgemäß arbeiten.
- **Integrationstests:** Die Zusammenarbeit zwischen YOLOv8-Modell, GUI und Steuerungslogik wurde getestet, um sicherzustellen, dass Daten korrekt übergeben und verarbeitet werden.
- **Systemtest:** Der gesamte Ablauf vom Starten des Weed Detectors über die Erkennung bis hin zur (simulierten) Unkrautbekämpfung mit dem Roboter wurde in einer realitätsnahen Umgebung geprüft.

Durch diese Tests konnte eine hohe Zuverlässigkeit und Nachvollziehbarkeit des Gesamtsystems erreicht und sichergestellt werden, dass Änderungen an einzelnen Modulen keine unerwarteten Fehler im Gesamtsystem verursachen.

3.8 Linting

Zur Verbesserung der Codequalität und Lesbarkeit wurde im Projekt **Pylint** eingesetzt. Pylint überprüfte den gesamten Python-Code auf **Syntaxfehler, Stilrichtlinien und potenzielle Programmierfehler** und gab Hinweise zur Optimierung der Struktur und Namenskonventionen.

Durch die Nutzung von Pylint konnte der Code **vereinheitlicht, unnötige Komplexität reduziert und die Wartbarkeit erhöht** werden. Zudem wurde sichergestellt, dass der Weed Detector den **PEP8-Standards** entspricht, was die langfristige Erweiterung und Übergabe an andere Entwickler erleichtert.

3.9 Deployment (CI/CD-Pipelines)

Für den Weed Detector wurde eine **Continuous Integration/Continuous Deployment (CI/CD) Pipeline** eingerichtet, um eine automatisierte und konsistente Auslieferung des Projekts zu ermöglichen.

Die Pipeline führt **automatisch Linting, Security-Scanning mit Bandit sowie Unit- und Integrationstests** bei jedem Push in das Repository durch, um Fehler frühzeitig zu erkennen. Nach erfolgreichem Durchlauf wird automatisch ein **Docker-Image erstellt und für den Einsatz auf der Zielhardware bereitgestellt**, wodurch die manuelle Einrichtung entfällt und die Deployment-Zeit verkürzt wird.

Durch den Einsatz von CI/CD konnte der **Entwicklungs- und Release-Prozess effizienter, zuverlässiger und reproduzierbar gestaltet werden**, wodurch der Weed Detector stabil und einsatzbereit gehalten wird.

3.10 Dokumentation erstellen

Zur Sicherstellung der **Nachvollziehbarkeit und Wartbarkeit** des Weed Detectors wurde eine umfassende Dokumentation erstellt. Diese umfasst die Projektstruktur, verwendete Technologien, Installations- und Nutzungshinweise, sowie die Beschreibung aller Arbeitspakete und deren Umsetzung.

Neben der technischen Dokumentation wurden auch **Code-Kommentare und Docstrings** hinzugefügt, um Funktionen und Abläufe im Code klar zu erläutern. Dies erleichtert zukünftigen Entwicklern den Einstieg und ermöglicht eine schnelle Erweiterung oder Anpassung des Systems.

Die Dokumentation dient zudem als Grundlage für Präsentationen und die Projekteinreichung im Software Engineering Kurs, wodurch die erarbeiteten Ergebnisse strukturiert und verständlich aufbereitet zur Verfügung stehen.

3.11 Präsentation erstellen

Zum Abschluss des Projekts wurde eine Präsentation erstellt, um den Weed Detector und die erarbeiteten Ergebnisse im Rahmen des Software Engineering Kurses vorzustellen.

Die Präsentation enthält eine übersichtliche Darstellung der Projektziele, des technischen Aufbaus, der verwendeten Technologien und der wichtigsten Arbeitsschritte. Zudem werden die Funktionalität des Weed Detectors und die Ergebnisse der Unkrauterkennung anhand von Screenshots und Codeausschnitten anschaulich präsentiert.
Durch die strukturierte Aufbereitung wird der Entwicklungsprozess nachvollziehbar erklärt, und der Mehrwert des Systems für den Anwendungsfall prägnant und verständlich vermittelt.