Diplomarbeit:
Lastenroboter

Höhere Technische Bundeslehranstalt Graz Gösting
Schuljahr 2024/25

Diplomanden:		**Betreuer:**
Daniel Schauer	5AHEL	Prof. DI. Gernot Mörtl
Simon Spari	5AHEL	
Felix Hochegger	5AHEL	

Eidesstattliche Erklärung

Wir erklären an Eides statt, dass wir die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht haben.

Ort, am TT.MM.JJJJ

Daniel Schauer

Simon Spari

Felix Hochegger

Danksagung

An dieser Stelle möchten wir unseren aufrichtigen Dank aussprechen.

Ein besonderer Dank gilt Herrn Prof. DI Gernot Mörtl für seine wertvolle Unterstützung, seine fachliche Begleitung und seine konstruktiven Anregungen während der gesamten Arbeit. Seine Expertise und sein Engagement haben maßgeblich zum Gelingen dieser Diplomarbeit beigetragen.

Ebenso danken wir unseren Freunden, insbesondere Michael Johannes Anderhuber, für seine Unterstützung beim Schweißen des Gehäuses. Sein handwerkliches Geschick und seine Hilfe waren für die Umsetzung unseres Projekts von großem Wert.

Unser großer Dank gilt zudem unserem großzügigen Sponsor,"Vogl Baumarkt Rosental", für das Sponsoring des Metalls für das Gehäuse. Durch diese Unterstützung konnten wir unser Projekt in dieser Form verwirklichen.

Inhaltsverzeichnis

1 Einleitung

1.1 Kurzzusammenfassung

In dieser Diplomarbeit wird ein Lastenroboter entwickelt, der bis zu 25 Kilogramm transportieren kann. Der Roboter wird über eine Website gesteuert, die als Steuerungsplattform dient. Zusätzlich ist eine Kamera eingebaut, die den Transportbereich zeigt, sowie eine Waage, die das Gewicht der transportierten Last misst.

Ein Schwerpunkt der Arbeit lieg auf der mechanischen Konstruktion des Roboters, bei der ein stabiles Gehäuse aus Stahl gebaut wird, um Sicherheit und Stabilität zu gewährleisten. Außerdem wird eine eigene Platine entwickelt, die die verschiedenen Hardware-Komponenten, wie die Sensoren und die Motoren, steuert und miteinander verbindet.

Die Steuerung des Roboters erfolgt über eine Website, die es dem Benutzer ermöglicht, den Roboter zu bedienen und wichtige Daten wie Akkustand und Gewicht abzurufen. Ein besonderer Fokus liegt auch dabei auf der Übertragung des Kamerabildes auf die Web-Oberfläche sowie der Integration einer schwenkbaren Kamera, um eine flexible Sicht auf den Transportbereich zu gewährleisten. Der ESP32 sorgt dafür, dass die Befehle des Benutzers an den Roboter übermittelt werden.

Zusätzlich wird eine OnBoard-Software entwickelt, die es ermöglicht, die Sensoren auszulesen und die Motoren als auch die Kamera anzusteuern.

1.2 Abstract

This thesis develops a load robot that can carry up to 25 kilograms. The robot is controlled via a website, which serves as the control platform. Additionally, a camera is integrated to display the transport area, as well as a scale to measure the weight of the carried load.

A key focus of the work is on the mechanical design of the robot, where a sturdy steel housing is built to ensure safety and stability. Furthermore, a custom circuit board is developed to control and connect the various hardware components, such as sensors and motors.

The robot is controlled via a website, which allows the user to operate the robot and access important data such as battery level and weight. A particular focus is also placed on

transferring the camera feed to the web interface and integrating a swivel camera to ensure flexible viewing of the transport area. The ESP32 ensures that the user's commands are transmitted to the robot.

Additionally, onboard software is developed to read the sensors and control the motors and camera.

2 Projektmanagement

2.1 Projektteam

Betreuer: Prof. DI. Gernot Mörtl

Abbildung 1: Porträt Daniel Schauer

Daniel Schauer: Software-OnBoard

- Projektleiter
- Verbindung von Software und Hardware (ESP32 zu Sensoren)
- Kamera-Übertragung zur Web-Oberfläche
- Umsetzung einer schwenkbaren Kamera
- Dokumentation

Abbildung 2: Porträt Simon Spari

Simon Spari: Software-App

- Benutzeroberfläche (Web-Oberfläche) für Steuerung
- Übertragung der Steuerung/Befehle von Web-Oberfläche zu ESP32
- Kamera-Übertragung zur Web-Oberfläche
- Dokumentation

Abbildung 3: Porträt Felix Hochegger

Felix Hochegger: Hardware-Design und Mechanik

- Bau des Roboter Gehäuse
- Ansteuerung und Verbindung von Hardware (Ansteuerung und Berechnung der Motoren)
- Dokumentation

2.2 Projektstrukturplan

2.3 Meilensteine

Um unseren Fortschritt und unsere Zeiteinteilung besser im Überblick zu behalten, haben wir uns bestimmte Meilensteine für unser Projekt gesetzt. Diese sind sehr hilfreich, um das Projekt strukturiert umzusetzen, angefangen von der Projektplanung bis hin zum fertigen Prototyp.

Die folgenden Meilensteine haben wir uns bei der Projektplanung gesetzt:

Meilenstein	**Datum**
Grundlegendes Gehäuse	07.11.2024
Funktionsfähige Website	19.12.2024
Funktionsfähige steuerbare Motoren	16.01.2025
Funktionsfähiger Prototyp	06.03.2025

2.4 Kostenaufstellung

3 Antrieb

3.1 Motoren

3.2 Motorentreiber

3.3 Schaltungsaufbau

3.4 Code

4 Webserver

4.1 Grundlegende Ziele

In diesem Kapitel befassen wir uns mit der geplanten Website, die den Benutzern und Benutzerinnen die Steuerung des Lastenroboters ermöglichen soll. Zuallererst definieren wir die grundlegenden Funktionen, die die Website erfüllen soll. Sobald diese erfüllt sind, versuchen wir, das User Interface so einfach und benutzerfreundlich wie möglich zu gestalten. Ein weiterer Punkt ist die Darstellung der spezifischen Messwerte und Daten, damit diese am Webserver schnell und leicht zugänglich sind.

Für die Entwicklung der Website verwenden wir HTML, CSS und JavaScript, um alle funktionalen und optischen Anforderungen zu erfüllen.

Videoübertragung

Auf der Website soll eine Echtzeit Videoübertragung der ESP32-CAM angezeigt werden. Die Bildfrequenz und die Qualität der Videoübertragung sollte ausgeglichen sein, so dass in der Übertragung alle Objekte und ggf. Hindernisse frühzeitig erkennbar sind und noch Reaktionszeit zum Manövrieren besteht.

Steuerung

Auf der Website soll eine grafische Steuereinheit implementiert werden, mit der der Roboter gesteuert und navigiert werden kann. Diese soll dann die entsprechenden Steuerbefehle bzw. Richtungen an den ESP32 senden, wo sie dann in Steuerungsbefehle für die Motoren übersetzt werden.

Anzeigen von Daten

Auf der Website sollen bestimmte Messwerte und Daten, wie zum Beispiel Akkustand oder zurzeit aufliegende Last, die vom ESP32 durch Sensoren oder Messungen ausgewertet werden, übersichtlich und leicht zugänglich angezeigt werden.

4.2 Ideen und Entwürfe

4.3 Webserver

4.3.1 Webserver Setup

Der Webserver wird mithilfe der Bibliothek ESPAsyncWebServer auf einem ESP32-CAM Mikrocontroller eingerichtet. Diese Bibliothek ermöglicht einen nicht-blockierenden Betrieb, wodurch parallele Anfragen effizient verarbeitet werden können.[1]

Für die Netzwerkkonfiguration wurde ein eigener Access Point mit folgenden Parametern definiert:

- SSID (Service Set Identifiert) : “Carybot” – dient zur Identifikation des drahtlosen Netwerkes
- Passwort: “123456789“ – dient zum Schutz des Netwerkes
- Lokale IP-Adresse: 192.168.4.1 – dient zum Zugriff auf die Webserver-Oberfläche
- Gateway-Adresse: 192.168.4.1 - ESP32-CAM fungiert als Access Point
- Subnetzmaske: 255.255.255.0 - ermöglicht die Kommunikation zwischen Geräten im Bereich 192.168.4.x

Der Webserver wird auf Port 80 erstellt. Port 80 ist der Standardport für HTTP-Dienste.

In der Setup Funktion wird danach überprüft, ob der Access Point erfolgreich konfiguriert worden ist und ob der Access Point erfolgreich gestartet werden kann. Falls ein Fehler auftreten sollte, wird der Setup unterbrochen und die jeweilige Fehlermeldung in der seriellen Konsole ausgegeben. Wenn alles erfolgreich konfiguriert ist und starten kann, wird im Seriellen Monitor die IP-Adresse in der seriellen Konsole ausgegeben. Anschließend wird der Webserver gestartet.

[1] https://github.com/lacamera/ESPAsyncWebServer

4.3.2 SPIFFS Setup

SPIFFS (SPI Flash File System) ist ein leichtgewichtiges Dateisystem für Mikrocontroller mit SPI-Flash-Speicher. Es ermöglicht das Speichern und Verwalten von Dateien direkt im Flash-Speicher des Mikrocontrollers. SPIFFS wird in unserem Projekt benötigt, um statische Dateien für unseren Webserver (HTML-, CSS-, JavaScript Anwendungen) bereitzustellen.

In unserem Code wird zuallererst einmal überprüft, ob SPIFFS beim Start der ESP32-CAM richtig initialisiert werden kann. Falls es fehlschlägt, wird eine Fehlermeldung in der seriellen Konsole ausgegeben und das Programm gestoppt. Ansonsten werden die Webserver-Endpunkte über HTTP-GET-Routen definiert, über die unsere statischen Dateien aus SPIFFS an Clients gesendet werden.

"/" ist die Standardroute des Servers. Somit wird dpad.html als Startseite angezeigt, wenn ein Client sich verbindet.

"menu-icon.svg" und "Fernlicht.svg" werden als SVG-Bilder (Scalable Vector Graphics) an den Browser gesendet. Image/svg+xml sorgt dafür, dass der Browser die Dateien als SVG-Bilder erkennt.

"mystyles.css" wird mit text/css als CSS-Datei gesendet und dient zur Formatierung der Website.

"carybot.js" wird mit application/javascript als Javascript-Datei gesendet und verarbeitet die Eingaben von Clients auf der Website.

Wenn alle Dateien erfolgreich geladen sind, wird eine Nachricht in der seriellen Konsole ausgegeben.

Die readFile() Funktion wird benötigt, um die jeweiligen Dateien aus SPIFFS lesen zu können. Die Funktion liest eine Datei aus dem SPIFFS-Speicher und gibt den Inhalt als String zurück.

4.4 WebSocket Kommunikation

4.4.1 Kommunikation Setup

Für die Kommunikation zwischen dem Webserver und dem ESP32 wird die ArduinoJson und die ArduinoWebSockets Bibliothek benötigt. Die ArduinoJson Bibliothek wird für

die Umwandlung der JSON-Steuerbefehle benötigt. Die ArduinoWebSockets Bibliothek wird für die Kommunikation über das WebSocket Protokoll benötigt.

Für die Netzwerkkonfiguration als Client werden folgenden Parameter definiert:

- SSID: “Carybot“ – gleiche SSID wie ESP32-CAM
- Passwort: “123456789“ – gleiches Passwort wie ESP32-CAM
- Lokale IP-Adresse: 192.168.4.3
- Gateway-Adresse: 192.168.4.1 – Adresse des Access Points (ESP32-CAM)
- Subnetzmaske: 255.255.255.0 – ermöglicht Kommunikation zwischen Geräten im Bereich 192.168.4.x

Für die WebSocket-Kommunkation wurde der Port 8080 gewählt.

In der Setup Funktion des Programmes wird dann überprüft, ob die IP-Konfiguration erfolgreich abgeschlossen wurde. Ansonsten kommt es zu einer Fehlermeldung und das Setup wird abgebrochen. Danach wird versucht, sich mit dem WLAN-Netzwerk zu verbinden. Wenn sich der ESP32 erfolgreich mit dem WLAN verbunden hat, wird eine Nachricht und die IP-Adresse des ESP32 in der seriellen Konsole ausgegeben. Danach wird die WebSocket Konfiguration noch gestartet. Es wird definiert, dass die Funktion onWebSocketEvent aufgerufen wird, wenn Events über den WebSocket registriert werden. In der loop Funktion wird dann noch ständig überprüft, ob neue Events am WebSocket registriert werden.

4.4.2 Message Handling

In der Funktion onWebSocketEvent() werden die WebSocket-Ereignisse verarbeitet. Sie wird aufgerufen, wenn sich ein WebSocket-Client verbindent, eine Nachricht sendet oder die Verbindung sich trennt. Der Parameter num steht für die ID des Clients, der das Event ausgelöst hat. Der Parameter type gibt die Art des WebSocket-Event an. Der Parameter payload sind die empfangenen Daten. Der Parameter length steht für die Größe des payload-Arrays. In der Funktion werden die Events mit einem switch-case verarbeitet

Übersicht der WebSocket-Ereignisse:

Die Funktion handleWebSocketMessage() verarbeitet die WebSocket-Nachricht, die als JSON-Objekte gesendet werden. Die Parameter sind wieder die Client-ID, die empfangenen Nachricht und die Länge der empfangenen Nachricht.

Ereignistyp	Beschreibung	Verarbeitung
WStype_CONNECTED	Ein neuer Client verbindet sich.	Ausgabe der Client-ID & IP-Adresse in der Konsole
WStype_TEXT	Eine Textnachricht wird empfangen.	Übergabe an `handleWebSocketMessage()`
WStype_BIN	Binärdaten werden empfangen.	Nachricht in der Konsole (wird nicht verarbeitet)
WStype_DISCONNECTED	Ein Client trennt die Verbindung.	Meldung mit der Client-ID in der Konsole.

Tabelle 2: Übersicht der WebSocket-Ereignisse

Zuerst wird die empfangene Nachricht (payload) in einen String konvertiert. Diese wird dann in der seriellen Konsole ausgegeben. Danach wird ein JSON-Dokument mit max 200 Bytes erstellt. Die empfangene Nachricht wird dann mit deserialzeJson() geparst. Falls das Parsen erfolgreich war, wird die JSON-Nachricht verarbeitet.

Wenn die JSON-Nachricht den Namen `robot_direction` enthält, wird die Richtung mit der Funktion `stringToDirection()` in eine eigene Variable umgewandelt. Auch die mitgesendete Variable `speed` wird ebenfalls in eine eigene Variable gespeichert.

Wenn die JSON-Nachricht den Namen `camera_position` enthält, wird der Wert der Nachricht in die Variable `camera_pos` gespeichert und die Funktion `cam_turn()` aufgerufen.

Wenn die JSON-Nachricht den Namen `light_status` enthält, wird der Wert der Nachricht in die boolesche Variable `light_status` gespeichert. Dieser Variable wird dann in die numerische Variable `light_st` umgewandelt (1 = an, 0 = aus). Wenn `light_st` eine 1 ist, wird die Funktion `lights_on()` aufgerufen, ansonsten wird die Funktion `lights_off()` aufgerufen.

4.5 Kamera

4.5.1 Kamera Setup

Um die Kamera programmieren zu können, muss zunächst das richtige Board `AI Thinker ESP32-CAM` ausgewählt werden. Danach müssen die ESP32-Kamera-Treiber mit der Bibliothek `esp_camera.h` inkludiert werden. Dann muss das passende ESP32-CAM-Modell festgelegt werden. Da wir uns für das `Ai-Thinker` Modell entschieden haben, muss diese nun definiert werden. Falls ein anderes Modell genutzt wird, muss es entsprechend angepasst werden.

Als nächstes werden die GPIO-Pins der ESP32-CAM für die Kamera OV2640 konfiguriert. Diese Zuordnung ist spezifisch für das Ai-Thinker-Modell und muss für jedes Modell individuell angepasst werden.

Als nächstes muss die ESP32-CAM mit der ESP-IDF `esp_camera` Bibliothek konfiguriert und initialisiert werden. Als erstes muss mit `camera_config_t` eine Struktur definiert werden, mit der verschiedene Parameter und Eigenschaften wie GPIO-Pins, Bildgröße und Qualität festlegt werden können.

LEDC-Kanal und Timer werden für das Taktsignal benötigt, um die Kamera zu betreiben.

4.6 Videoübertragung

In unserem Projekt werden die Live-Bilder per WebSocket an den Webserver gesendet. Um dies umzusetzen, wird zuerst eine Webserver-Route benötigt. Dazu wird eine http-GET-Anfrage für die Hauptseite (“/“) definiert. Im Code wird ein JavaScript Skript benutzt, um eine Websocket-Verbindung herzustellen. Die IP-Adresse wird automatisch durch `windows.location.hostname` erkannt. Port 81 wird für das WebSocket-Streaming verwendet. Wenn die ESP32-CAM ein neues Bild als WebSocket-Nachricht versendet, wird es als JPEG-Blob gespeichert. Danach wird ein temporär URL-Link erstellt. Das Bild wird schlussendlich in einem `<img>`-Tag mit der id `stream` angezeigt.

Die WebSocket Verbindung wird die ganze Zeit überwacht. In der Konsole wird ausgegeben, wenn die Websocket-Verbindung aktiv ist. Falls die Verbindung abbrechen sollte, wird nach 5 Sekunden automatisch ein erneuter Verbindungsversuch gestartet. Zum Schluss wird der HTML-Code mit `HTTP-Status 200` (OK) an den Browser gesendet.

4.7 Website

4.7.1 Implementierung der Steuerung

4.7.2 Echtzeit-Videoanzeige

4.7.3 Anzeige von Sensordaten und Systemstatus

4.8 Herausforderungen und Optimierungen

4.8.1 Probleme bei der WebSocket Kommunikation

4.8.2 Latenz- und Performance Optimierungen

4.8.3 (Speicher- und Rechenleistungseinschränkungen des ESP32)

4.9 (Fazit und Ausblick)

4.9.1 (Mögliche Erweiterungen und Verbesserungen)

5 Gehäuse

5.1 Planung und Design

5.2 Realisierung

5.3 Materialliste

6 Platine

6.1 Grundschaltung

6.2 Circuit Board

6.3 Fertiger Prototyp

7 Kamera

7.1 Kamera im Überblick

7.2 Videoübertragung

7.3 Kameraschwenkung

7.3.1 Gehäuse

7.3.2 Servomotor

7.4 Code

8 Sensoren

8.1 Abstandsensor

8.2 Gewichtsmessung

8.2.1 Grundprinzip

8.2.2 Schaltungsaufbau

8.2.3 Code

9 Entwicklungstools

9.1 Autodesk Fushion

9.2 Eagle

9.3 VS-Code

Visual Studio Code (VS-Code) ist eine kostenlose IDE (integrated development environment) entwickelt von Microsoft. VS-Code funktioniert auch auf anderen Betriebssystemen wie zum Beispiel Windows, Linux oder macOS. VS-Code unterstützt einen Großteil der Programmiersprachen und kann durch Extentions mit vielen nützlichen Features und Sprachen immer wieder erweitert werden.[2]

9.3.1 Setup

Um ein Projekt in VS-Code erstellen zu können, müssen einige Schritte befolgt werden. Zuallererst muss die IDE von https://code.visualstudio.com/ für das jeweilig passende Betriebssystem heruntergeladen und installiert werden. Um Mikrocontroller wie ESPs oder Arduinos in VS-Code programmieren zu können, wird die PlatformIO IDE Extension benötigt. Um die PlatformIO IDE Extension in VS-Code zu installieren, drückt man einfach auf das Extensions Symbol oder drückt die Tastenkombination Ctrl+Shift+X um das Extensions Menü zu öffnen. Danach gibt man in der Suchleiste “PlatformIO IDE“ ein und wählt die Extension mit der Ameise als Icon. Dann drückt man auf “Install“ und wartet, bis die Extension fertig heruntergeladen ist. Nach der Installation sollte das PlatformIO Icon (Ameisenkopf) auf der linken Seite unter dem Extension Menü erscheinen.

[2] https://code.visualstudio.com/

Um nun ein neues Projekt zu erstellen, klickt man auf das PlatformIO Icon und wählt “+ New Project“.

Im Project Wizard wählt man nun den gewünschten Namen, das Board, das Framework als auch den Speicherungsort des Projektes. Bei unserem Projekt wählten wir das Board “Espressif ESP32 Dev Module“ und als Framework “Arduino“, da wir einen EPS32 zum Programmieren verwendeten.

9.3.2 Bibliotheken

Bibliotheken sind ein weiterer wichtiger Bestandteil für das Programmieren. Bibliotheken beinhalten bereits eine Sammlung von vorgefertigtem Code, der für bestimmte Aufgaben, wie zum Beispiel zum Auswerten eines Sensors, verwendet werden kann. Bibliotheken werden verwendet, um den Code zu minimieren und dadurch die Lesbarkeit sowie die Effizienz zu steigern. Um eine Bibliothek für PlatformIO in VS-Code zu installieren, muss das PIO Home Menü geöffnet werden. Darin befindet sich der Reiter „Libraries“. Wenn dieses geöffnet wird, erscheint eine Suchleiste, mit der die gewünschten Bibliotheken zum Projekt hinzugefügt werden können.

9.3.3 verwendete Bibliotheken

ESPAsyncWebServer

Die ESPAsnycWebServer Bibliothek ermöglicht es, Webanwendungen effizient und mit hoher Performance auf ESP8266- und ESP32 Mikrocontroller zu hosten. Der Asynchrone Betrieb verhindert Blockierungen und sorgt für eine flüssige Verarbeitung mehrere Anfragen gleichzeitig. Außerdem unterstützt die Bibliothek WebSockets, welche für die Echtzeitkommunikation zwischen Client und Server benötigt wurden. Die Bibliothek ist eine leistungsfähigere Alternative zur klassischen WebServer-Bibliothek, da sie ressourcenschonender und nicht blockieren arbeitet.[3]

ArduinoJSON

Die ArduinoJson Bibliothek ermöglicht die Verarbeitung von JSON-String in Objekte und umgekehrt. Sie ist speziell für Geräte mit begrenztem Speicher und Rechenleistung optimiert. Die Bibliothek wird benötigt, um die erhaltenen Steuerbefehle am ESp32 zu konvertieren und um sie anschließend weiterzuverarbeiten.[4]

[3] https://github.com/lacamera/ESPAsyncWebServer
[4] https://arduinojson.org/

HCSR04

arduinoWebSockets

Die WebSockets Bibliothek ermöglicht eine Kommunikation über das WebSocket Protokoll für Arduino-Boards, ESP8266 und ESP32. Die Bibliothek wird für die Echtzeit-Kommunikation zwischen dem Webserver und den ESP32 benötigt. Außerdem werden die Bilder der ESp32-CAM über einen WebSocket an den Webserver gesendet. Die Bibliothek ist eine großartige Ergänzung zur ESPAsyncWebServer Bibliothek.[5]

ESp32Servo

Adafruit_MCP23x17

HX711_ADC

9.4 LaTex

9.5 GitHub

10 Abbildungsverzeichnis

Abbildungsverzeichnis

11 Literaturverzeichnis

[5] https://github.com/Links2004/arduinoWebSockets

```
if (!SPIFFS.begin(true)) {
  Serial.println("Fehler beim Mounten von SPIFFS");
  return;
}

server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
  String dpad = readFile(SPIFFS, "/dpad.html");
  request->send(200, "text/html", dpad);
});

server.on("/menu-icon.svg", HTTP_GET, [](AsyncWebServerRequest *request) {
  String icon = readFile(SPIFFS, "/menu-icon.svg");
  request->send(200, "image/svg+xml", icon);
});

server.on("/Fernlicht.svg", HTTP_GET, [](AsyncWebServerRequest *request) {
  String fernlicht = readFile(SPIFFS, "/Fernlicht.svg");
    request->send(200, "image/svg+xml", fernlicht);
});

server.on("/mystyles.css", HTTP_GET, [](AsyncWebServerRequest *request) {
  String css = readFile(SPIFFS, "/mystyles.css");
  request->send(200, "text/css", css);
});

server.on("/carybot.js", HTTP_GET, [](AsyncWebServerRequest *request) {
  String js = readFile(SPIFFS, "/carybot.js");
  request->send(200, "application/javascript", js);
});

Serial.println("SPIFFS-Dateien erfolgreich geladen");
```

Abbildung 4: SPIFFS Initalisierung
Quelle: eigene Abbildung