Systemdokumentation

1 Einführung

1.1 Webseite:

Die Webseite spielt eine zentrale Rolle bei der Steuerung der Roboter. Sie bietet den Benutzern die Möglichkeit, ihre Roboter zu kontrollieren und verschiedene Einstellungen vorzunehmen.

Verwendete Technologien:

- Frontend: HTML, CSS, JavaScript, Typescript
- Backend: Node.js, Express, Websockets, MQTT
- Datenbank: sqlite

Ports:

- 3000 – Steurung mit Webseite
- 3001 – Übersicht der Roboter
- 8080 – Websocket Server

1.2 Input Software:

Die Input Software ermöglicht die Kommunikation zwischen den Eingabegeräten und dem Server, um die Roboter in Echtzeit zu steuern.

Verwendete Technologien:

- Programmiersprache: Java
- Frameworks und Bibliotheken: JInput, JSON-Simple,

2 Installation und Setup

2.1 Webseite:

GITHUB - https://github.com/2324-3bhif-teaching/Input

```
npm install
```

```
Websocket server starten: ts-node Website/Websocket/Websocket-server.ts
```

```
Übersicht der Roboter: ts-node Website/Websocket/app.ts
```

```
Steuerungs Webseite Server: ts-node Website/src/app.ts
```

DOCKER

Docker image herunterladen

```
docker pull docker.io/hasenschwandtnera/robotercontroller:latest
```

```
Das Dockerimage starten mit Docker Desktop.
```

2.2 Input Software:

GITHUB - https://github.com/2324-3bhif-teaching/Input

3 Entwicklerdokumentation

3.1 Steuerungs Webseite:

3.1.1 Einleitung

Diese Dokumentation beschreibt die Funktionsweise der Software zur Steuerung eines Roboters über eine Benutzeroberfläche. Die Software verwendet verschiedene Technologien und Schnittstellen, um die Kommunikation zwischen dem Benutzerinterface, dem Roboter und einem WebSocket-Server zu ermöglichen.

3.1.2 Komponentenübersicht

Benutzerinterface

Das Benutzerinterface ermöglicht es dem Benutzer, verschiedene Steuerbefehle an den Roboter zu senden. Es besteht aus Buttons und Eingabefeldern, die über Events mit der Steuerungslogik verbunden sind.

Roboterobjekt (`robot`)

Das Roboterobjekt speichert den aktuellen Zustand des Roboters, einschließlich seiner Position, Geschwindigkeit und Ausrichtung. Es dient als zentraler Datenpunkt für die Steuerungslogik und wird über JSON-Nachrichten an einen WebSocket-Server gesendet.

WebSocket-Kommunikation

Die WebSocket-Verbindung (`socket`) ermöglicht die Echtzeitkommunikation zwischen dem Benutzerinterface und einem externen Server, der die Steuerbefehle entgegennimmt und verarbeitet.

3.1.3 Steuerungslogik

Die Steuerungslogik definiert, wie die Benutzereingaben (Button-Events) verarbeitet werden, um die Bewegungen des Roboters zu steuern. Sie umfasst folgende Hauptkomponenten:

`startControlling()`

Diese Funktion initialisiert die Steuerung, indem sie Event-Listener für die Steuerungs-Buttons registriert. Sie erfasst die Benutzerinteraktionen und ruft entsprechende Handler-Funktionen auf.

- **Button-Events:** Die `startControlling()` Methode nutzt `document.querySelectorAll('.controll-button').forEach(button => {...})`, um jedem Steuerungs-Button Event-Listener hinzuzufügen. Wenn ein Button gedrückt wird (`mousedown`), wird `handleButtonPress(event)` aufgerufen, und wenn er losgelassen wird (`mouseup`), wird `handleButtonRelease(event)` aufgerufen.

`handleButtonPress(event: MouseEvent)`

Diese Funktion wird aufgerufen, wenn ein Steuerungs-Button gedrückt wird. Sie interpretiert den Button und aktualisiert den Roboterzustand entsprechend.

- **Button-Handling:** Basierend auf der ID des gedrückten Buttons wird eine Richtung aus einem vordefinierten Mapping (`directionMap`) ermittelt und an `handleInputs(command)` übergeben.
- **Nachrichtenversand:** Die aktuelle Roboterstatus wird in ein JSON-Format konvertiert und über die WebSocket-Verbindung an den Server gesendet.

`handleInputs(command: string)`

Diese Funktion aktualisiert den Roboterstatus basierend auf dem empfangenen Befehl.

- **Switch-Case:** Entsprechend des übergebenen Befehls (`command`) werden die Roboterattribute wie `front`, `back`, `left`, `right`, `speed` und `direction` aktualisiert.
- **Bewegungsrichtung:** Basierend auf den gesetzten Bewegungsrichtungen werden die Werte für `direction` entsprechend gesetzt.

`handleButtonRelease(event: MouseEvent)`

Diese Funktion wird aufgerufen, wenn ein Steuerungs-Button losgelassen wird. Sie setzt die Roboterbewegung auf Null und sendet den aktualisierten Zustand über die WebSocket-Verbindung.

- **Button-Handling:** Ähnlich wie `handleButtonPress`, jedoch mit einem Mapping für Stop-Befehle (`stopDirectionMap`), um die Bewegung zu stoppen.

3.1.4 Initialisierung und Konfiguration

Die Initialisierungsfunktionen richten das Benutzerinterface ein, verbinden sich mit dem WebSocket-Server und bereiten das Roboterobjekt für die Steuerung vor.

`initInput()`

Diese Funktion initialisiert das Eingabefeld für die Geräte-ID und verarbeitet die Benutzerinteraktionen.

- **Geräte-ID:** Die eingegebene Geräte-ID wird gespeichert und für die Identifikation des Roboters verwendet.
- **Event-Handling:** Event-Listener für den Join-Button und Toggle-Mode (vereinfacht/erweitert) werden hinzugefügt, um die Benutzerinteraktion zu steuern.

`initLogin()`

Diese Funktion verwaltet das Login-System und aktualisiert die Authentifizierungszustände der Benutzeroberfläche.

- **Login-Buttons:** Event-Listener für Login- und Logout-Buttons werden hinzugefügt, um Benutzer anzumelden oder abzumelden.
- **Authentifizierung:** Die Funktion verwendet `fetchRestEndpoint()` zum Abrufen des Authentifizierungsstatus und aktualisiert die Oberfläche entsprechend.

3.2 Steuerungs Webseite Server:

3.2.1 Übersicht

Diese Software ist eine Webserver-Anwendung, die mit Express.js erstellt wurde. Sie bietet Authentifizierung über Auth0 und verwaltet Renneinstellungen sowie Roboter-Eingabegerätekonfigurationen über verschiedene API-Endpunkte. Die Anwendung verwendet außerdem SQLite für Datenbankoperationen.

3.2.2 Abhängigkeiten

- `express`: Ein minimales und flexibles Node.js-Webanwendungs-Framework.
- `cors`: Middleware zur Aktivierung von Cross-Origin Resource Sharing.
- `sqlite` und `sqlite3`: Pakete zur Interaktion mit SQLite-Datenbanken.

3.2.3 Konfiguration

Auth0 Konfiguration

Die Anwendung verwendet Auth0 zur Benutzer-Authentifizierung. Die Konfigurationseinstellungen für Auth0 sind:

- `authRequired`: Auf `false` gesetzt, um die Authentifizierung für alle Routen optional zu machen.
- `auth0Logout`: Auf `true` gesetzt, um Auth0-Logout zu ermöglichen.
- `secret`: Ein langes, zufällig generiertes Zeichen, das in der Umgebungsvariable gespeichert ist.
- `baseURL`: Die Basis-URL der Anwendung.
- `clientID`: Die Client-ID von Auth0.
- `issuerBaseURL`: Die Basis-URL des Auth0-Emittenten.

3.2.4 Routen und Middleware

Middleware

- `express.static`: Dient zum Bereitstellen statischer Dateien aus dem `public`-Verzeichnis.
- `express.json`: Analysiert eingehende JSON-Anfragen.

Routen

1. **Authentifizierungsroute**
   - `GET /authenticated`
     - Überprüft, ob der Benutzer authentifiziert ist.
     - Wenn authentifiziert, wird die Benutzer-ID zurückgegeben.
     - Wenn nicht authentifiziert, wird ein `204`-Status mit `userId: null` zurückgegeben.
   - `GET /logout`
     - Loggt den Benutzer von Auth0 aus und leitet zur Basis-URL um.
2. **Rennmanagement-Routen**
   - `POST /api/raceManagement/setInputDeviceId`
     - Setzt die Eingabegeräte-ID für einen Roboter.
     - Anforderungskörper: `{ robotId: string, inputDeviceId: number | null }`
     - Antwort: Status `200` mit einer Erfolgsmeldung oder `404`, wenn der Roboter nicht gefunden wurde.
   - `GET /api/raceManagement/robots`
     - Gibt eine Liste aller Roboter zurück.
     - Antwort: Status `200` mit einem JSON-Array von Robotern.
   - `POST /api/raceManagement/robotID`
     - Findet einen Roboter anhand seiner Eingabegeräte-ID.
     - Anforderungskörper: `{ inputDeviceId: number | null }`
     - Antwort: Status `200` mit der Roboter-ID oder `204`, wenn kein Roboter gefunden wurde.
   - `GET /api/raceManagement/inputIdRobot`
     - Generiert und gibt eine neue Eingabegeräte-ID zurück.
     - Antwort: Status `200` mit einer neuen `inputDeviceId`.
3. **Einstellungsrouten**
   - `POST /api/settings/getSettings`
     - Ruft die aktuellen Einstellungen für einen Benutzer ab.
     - Anforderungskörper: `{ userID: string }`
     - Antwort: Status `200` mit den Einstellungen oder `404`, wenn nicht gefunden.
   - `POST /api/settings`
     - Speichert die Einstellungen für einen Benutzer.
     - Anforderungskörper: `{ userID: string, acceleration: number, maxSpeed: number, steering: number }`
     - Antwort: Status `200` mit einer Erfolgsmeldung oder `401`, wenn nicht autorisiert.

3.2.5 Datenbankoperationen

Datenbankverbindung

- **createDBConnection()**: Stellt eine Verbindung zur SQLite-Datenbank her und stellt sicher, dass Tabellen erstellt werden, falls sie nicht existieren.
- **ensureTablesCreated()**: Erstellt die `Settings`-Tabelle, falls sie nicht existiert.

Einstellungsverwaltung

- **saveSettings(id, acceleration, maxSpeed, steering)**: Speichert oder aktualisiert die Einstellungen für einen bestimmten Benutzer.
- **fetchCurrentSettings(id)**: Ruft die aktuellen Einstellungen für einen bestimmten Benutzer ab. Wenn keine Einstellungen gefunden werden, werden Standardwerte eingefügt.

3.3 Übersicht der Roboter Webseite:

3.3.1 Einleitung

Die vorliegende Dokumentation beschreibt eine JavaScript-basierte Anwendung zur Steuerung und Visualisierung von Robotern. Die Anwendung umfasst Funktionen zur Berechnung der kürzesten Rotation, Aktualisierung der Roboterliste sowie Kommunikation über REST und WebSocket.

3.3.2 Funktionen und Methoden

```
calculateShortestRotation(current: number, target: number): number
```

Diese Funktion berechnet die kürzeste Drehung von der aktuellen zur Zielrichtung in Grad.

- **Parameter:**
  - `current`: Aktuelle Richtung in Grad.
  - `target`: Zielrichtung in Grad.
- **Rückgabewert:**
  - Differenz zwischen `target` und `current`, wobei Berücksichtigung der kürzesten Drehung durchgeführt wird.

```
updateRobotList(robotId: string, targetDirection: number, speed: number): void
```

Aktualisiert die Richtung und Geschwindigkeit eines Roboters in der Liste und visualisiert dies im UI.

- **Parameter:**
  - `robotId`: Eindeutige ID des Roboters.
  - `targetDirection`: Neue Zielrichtung des Roboters.
  - `speed`: Neue Geschwindigkeit des Roboters.
- **Ablauf:**
  - Ermittelt den Index des Roboters mit der angegebenen `robotId`.
  - Berechnet die kürzeste Rotation zur `targetDirection`.
  - Aktualisiert die visuelle Darstellung der Roboter-Richtung im UI.
  - Ruft `updateRobotSpeed` auf, um die Geschwindigkeit anzuzeigen.

```
updateRobotSpeed(speed: number): void
```

Aktualisiert die Geschwindigkeitsanzeige eines Roboters im UI.

- **Parameter:**
  - `speed`: Neue Geschwindigkeit des Roboters.
- **Ablauf:**
  - Berechnet die prozentuale Geschwindigkeit im Verhältnis zu einer maximalen Geschwindigkeit von 260.
  - Aktualisiert die Höhe des Geschwindigkeitsbalkens im UI entsprechend.

```
fetchRestEndpoint(route: string, method: "GET" | "POST" | "PUT" | "DELETE", data?: object): Promise<any>
```

Führt eine REST-Anfrage an den angegebenen Endpunkt durch.

- **Parameter:**
  - `route`: Der Endpunkt für die REST-Anfrage.
  - `method`: HTTP-Methode für die Anfrage (GET, POST, PUT, DELETE).
  - `data` (optional): Daten, die bei POST oder PUT gesendet werden sollen.
- **Rückgabewert:**
  - Gibt ein Promise zurück, das die Antwort der Anfrage enthält.

```
loadRobotList(): Promise<void>
```

Lädt die Liste der Roboter von einem REST-Endpunkt und aktualisiert das UI entsprechend.

- **Ablauf:**
  - Sendet eine GET-Anfrage an den REST-Endpunkt, um die Liste der Roboter abzurufen.
  - Fügt jeden Roboter zur internen `robots`-Liste hinzu.
  - Erstellt für jeden Roboter einen Listeneintrag im UI mit einem Button zur Anzeige von Details.

Weitere Funktionen und Methoden

Es gibt weitere Hilfsfunktionen wie `showRobotDetails`, `closeModal` und Event-Listener, die die Benutzeroberfläche steuern und interaktive Elemente verwalten.

3.3.3 Verwendung von WebSocket

Die Anwendung nutzt WebSocket für die Echtzeitkommunikation mit den Robotern.

- **WebSocket-Verbindung:**
  - Verbindet sich mit einem WebSocket-Server (`ws://localhost:8080`) und sendet einen Authentifizierungstoken.
  - Empfängt Nachrichten über Änderungen im Zustand der Roboter und aktualisiert die Anzeige entsprechend.

3.4 Websocket Server:

3.4.1 Einführung

Diese Dokumentation beschreibt die Funktionalität eines WebSocket-Servers, der mit TypeScript implementiert wurde. Der Server ermöglicht die Kommunikation über WebSockets und erfordert eine Authentifizierung mittels eines geheimen Tokens für bestimmte Clients.

3.4.2 Funktionen

1. **Verbindungsaufbau**
   - Der Server lauscht auf Port 8080 und akzeptiert eingehende WebSocket-Verbindungen.
   - Bei erfolgreicher Verbindungsaufnahme wird eine Bestätigung auf der Konsole ausgegeben.
2. **Authentifizierung**
   - Clients müssen sich durch Übermittlung eines Tokens authentifizieren.
   - Der Server überprüft den erhaltenen Token und speichert die Verbindung des spezifischen Clients.
3. **Nachrichtenübermittlung**
   - Sobald ein Client authentifiziert ist, können Nachrichten empfangen und an den spezifischen Client weitergeleitet werden.
   - Nicht-authentifizierte Nachrichten oder Nachrichten von anderen Clients werden ignoriert.
4. **Verbindungsmanagement**
   - Der Server erkennt das Schließen einer Verbindung und aktualisiert gegebenenfalls die Referenz des spezifischen Clients.

Nutzung

Um den WebSocket-Server zu verwenden, müssen Clients eine Verbindung zum `ws://localhost:8080` herstellen. Die Kommunikation erfolgt über JSON-formatierte Nachrichten, wobei der Token für die Authentifizierung erforderlich ist.

Fehlerbehandlung

- Fehler beim Parsen von Nachrichten werden auf der Konsole protokolliert.
- Ungültige Tokens oder fehlgeschlagene Authentifizierungen werden ebenfalls behandelt und geloggt.

Beispiel

Ein Beispiel für die Verwendung des WebSocket-Servers ist die Übermittlung von Echtzeitdaten zwischen einem Client und dem Server, wobei der Server sicherstellt, dass nur authentifizierte Clients Zugriff auf bestimmte Daten haben.

3.4.3 Konfiguration

Der Server verwendet den Port 8080 für WebSocket-Verbindungen und einen geheimen Token (`SECRET_TOKEN`), der für die Authentifizierung benötigt wird.

3.5 Input Software:

3.5.1 Übersicht

Diese Dokumentation beschreibt die Funktionsweise und Struktur einer Java-basierten Steuerungssoftware für einen Roboter. Die Steuerung erfolgt über Tastatur und Gamepad. Die Hauptkomponenten der Software umfassen die folgenden Klassen und Konfigurationsdateien:

1. **AppModel.java**: Hauptklasse der Anwendung
2. **GamepadInput.java**: Klasse zur Handhabung der Gamepad-Eingaben
3. **KeyboardInput.java**: Klasse zur Handhabung der Tastatur-Eingaben
4. **Robot.java**: Klasse zur Steuerung des Roboters
5. **gamepad.json**: Konfigurationsdatei für Gamepad-Steuerung
6. **keyboard.json**: Konfigurationsdatei für Tastatur-Steuerung

3.5.2 Files

AppModel.java

Beschreibung

`AppModel` ist die Hauptklasse der Anwendung, die die Steuerlogik des Roboters enthält. Diese Klasse initialisiert die Steuerungskomponenten und orchestriert deren Interaktionen.

Funktionen

- **Initialisierung der Steuerung**: Lädt die Konfigurationen für Tastatur und Gamepad.
- **Verarbeitung der Eingaben**: Verarbeitet die Eingaben von Tastatur und Gamepad und steuert den Roboter entsprechend.

Wichtige Methoden

- `initializeControllers()`: Lädt die Konfigurationsdateien und initialisiert die Eingabe-Controller.
- `processInput()`: Verarbeitet die aktuellen Eingaben und aktualisiert den Zustand des Roboters.

GamepadInput.java

Beschreibung

`GamepadInput` ist eine Klasse, die die Eingaben eines Gamepads verarbeitet. Die Konfiguration wird aus der Datei `gamepad.json` geladen.

Funktionen

- **Verarbeitung der Gamepad-Eingaben**: Liest und interpretiert die Eingaben des Gamepads.
- **Übersetzung der Eingaben**: Übersetzt die Gamepad-Eingaben in Aktionen für den Roboter.

Wichtige Methoden

- `loadConfiguration(String filepath)`: Lädt die Gamepad-Konfiguration aus der angegebenen JSON-Datei.
- `getGamepadState()`: Gibt den aktuellen Zustand der Gamepad-Eingaben zurück.

JSON-Konfiguration (gamepad.json)

Die Datei `gamepad.json` enthält die Zuordnung der Gamepad-Tasten zu den Steuerungsaktionen. Beispielinhalt:

```
json
{
  "forward": "Taste 2",
  "backward": "Taste 0",
  "steer": "X-Achse"
}
```

Dieser JSON-Inhalt ordnet die Taste 2 dem Vorwärtsfahren, die Taste 0 dem Rückwärtsfahren und die X-Achse dem Lenken zu.

KeyboardInput.java

Beschreibung

`KeyboardInput` ist eine Klasse, die die Eingaben einer Tastatur verarbeitet. Die Konfiguration wird aus der Datei `keyboard.json` geladen. Diese Klasse ermöglicht die Speicherung und das Laden von Tastenbelegungen sowie die Kommunikation mit dem Roboter über WebSocket.

Funktionen

- **Verarbeitung der Tastatur-Eingaben**: Liest und interpretiert die Eingaben der Tastatur.
- **Speichern und Laden der Tastenbelegungen**: Ermöglicht das Speichern und Laden der Tastenbelegungen in einer JSON-Datei.
- **WebSocket-Kommunikation**: Stellt eine Verbindung zum Roboter her und sendet Steuerbefehle über WebSocket.

Wichtige Methoden

- `saveKeybinds()`: Speichert die aktuellen Tastenbelegungen in der JSON-Datei.
- `loadKeybinds()`: Lädt die Tastenbelegungen aus der JSON-Datei.
- `start(Scene scene)`: Startet die Überwachung der Tastatureingaben in der gegebenen Szene.
- `handleKeyPress(KeyCode keyCode)`: Handhabt das Drücken einer Taste.
- `handleKeyRelease(KeyCode keyCode)`: Handhabt das Loslassen einer Taste.
- `changeMovementState(String direction, boolean isPressed)`: Ändert den Bewegungszustand des Roboters.
- `createMessage(Robot robot)`: Erstellt eine JSON-Nachricht für den Roboter.
- `sendMessage(JSONObject message)`: Sendet eine JSON-Nachricht über WebSocket.
- `checkConnection()`: Überprüft die WebSocket-Verbindung.

JSON-Konfiguration (keyboard.json)

Die Datei `keyboard.json` enthält die Zuordnung der Tastaturtasten zu den Steuerungsaktionen. Beispielinhalt:

```
json
{
  "left": "A",
  "forward": "L",
  "backward": "F",
  "right": "H"
}
```

Dieser JSON-Inhalt ordnet die Taste A dem Linkslenken, die Taste L dem Vorwärtsfahren, die Taste F dem Rückwärtsfahren und die Taste H dem Rechtslenken zu.

Robot.java

Beschreibung

`Robot` ist die Klasse, die die eigentlichen Steuerungsbefehle für den Roboter implementiert. Sie enthält Methoden zur Steuerung der Bewegung und anderer Aktionen des Roboters.

Funktionen

- **Bewegungssteuerung**: Implementiert die Logik zur Bewegung des Roboters.
- **Zustandsverwaltung**: Verwalten des aktuellen Zustands des Roboters.

Wichtige Methoden

- `moveForward()`: Bewegt den Roboter vorwärts.
- `moveBackward()`: Bewegt den Roboter rückwärts.
- `turnLeft()`: Lässt den Roboter nach links drehen.
- `turnRight()`: Lässt den Roboter nach rechts drehen.

4 Benutzerhandbuch

Webseite:

Input Software: