Anforderungen Projektphase 1

Camp2Code

Florian Edenhofner und Robert Heise

Education4Industy GmbH

Zuletzt aktualisiert: 2024-06-22

Inhaltsverzeichnis

1. Überblick Projektziel

Das Ziel dieses Projektes ist die Entwicklung einer Software, die es dem Nutzer ermöglicht, ein Modellauto in verschiedenen Fahrmodi fahren zu lassen.

Dafür steht ein Modellauto zur Verfügung, das aus einem Chassis mit Lenkung und Antrieb besteht und mit einem Ultraschallsensor und einer Infrarotsensorleiste ausgestattet ist. Als zentrale Steuereinheit für alle Komponenten wird ein Raspberry Pi mit dem Betriebssystem Raspberry-Pi-OS verwendet. Die Implementierung der Software erfolgt mit der Programmiersprache Python.

Es werden verschiedene grundlegende Python-Klassen bereitgestellt, die den Zugriff auf die einzelnen Komponenten des Autos ermöglichen.

Das Ziel der ersten Woche besteht darin, die Bauteile und Sensoren des Autos anzusteuern bzw. auszulesen und erste einfache Fahrstrecken zu bewältigen. In der zweiten Woche soll das Auto in der Lage versetzt werden, eigenständig einer auf den Boden befindlichen Linie zu folgen. Darüber hinaus sollen die Fahrdaten des Autos (Informationen über den Status der Bauteile und Sensoren) während der Fahrt aufgezeichnet und visualisiert werden können.

Die finale Software soll dem Anwender ermöglichen, zwischen verschiedenen Fahrmodi zu wählen und diese zu starten. Sie wird in Form eines gut dokumentierten Quellcodes präsentiert. Eine kurze Anwenderdokumentation soll potenziellen Anwendern die Bedienung erleichtern bzw. ermöglichen.

2. Projektorganisation

2.1. Arbeitsweise

Das Projekt sollte als Teamarbeit in einer Gruppe von etwa fünf Personen durchgeführt werden. **Jedes Team soll gemeinsam eine Software bzw. einen Quellcode** entwickeln. Das Team ist für Planung, Entwurf, Entwicklung, Tests und Dokumentation verantwortlich.

In Anlehnung an eine agile Arbeitsweise soll vorrangig auf eine schlanke Projektplanung, die Definition von Teilaufgaben und die regelmäßige offene Kommunikation der Teammitglieder geachtet werden. Eine agile Methodologie (Scrum, Kanban) muss nicht strikt umgesetzt werden.

Teilaufgaben sollen als Arbeitspakete in kleinen Einheiten (User-Stories) beschrieben werden, die z.B. in einem Kanban- oder Scrum-Board organisiert werden können. So lassen sich Arbeitspaket innerhalb des Teams leicht dokumentieren und zuweisen.Es kann zusammen, ein Paaren oder einzeln an Teilaufgaben gearbeitet werden. Wichtig ist, dass die Arbeitsschritte aufeinander abgestimmt sind und agile Rituale wie z.B. ein “Daily Scrum” durchgeführt werden. (In einem Team können die Rollen des Product Owners und Scrum-Masters vergeben werden. Diese Teammitglieder sollten jedoch im Rahmen der Projektphase auch Teil des Entwicklerteam bleiben.)

2.2. Technische Voraussetzungen und Materialien

Folgende Schritte müssen zu Beginn des Projektes durchgeführt werden bzw. durchgeführt worden sein:

1. Raspberry Pi vorbereiten (OS installieren, Konfigurationen, Softwareinstallationen),
2. Notwendige Python-Files downloaden und sichten,
3. Funktionstests des Modellautos und gegebenenfalls Bugfixes,
4. Einrichten einer individuellen Arbeitsumgebung (z.B. Remoteverbindung testen).
5. Testen der Basisklassen

Die Basisklassen werden im File *basisklassen.py* zur Verfügung gestellt. Sie umfassen die Klassen *BackWheels*, *FrontWheels*, *Ultrasonic* und *Infrared*. Klären Sie die Bedeutung der einzelnen Argumente der Konstruktoren und den Zwecke der einzelnen Klassenmethoden. (Das File *basisklassen.py* beinhaltet weitere Klassen, welche von den Basisklassen verwendet werden. Mit diesen brauchen Sie sich nicht auseinandersetzen oder sie verstehen.)

Alle dafür notwendigen Material werden separat zur Verfügung gestellt.

2.3. Projektplanung

Die Anforderungen an die Software sind in diesem Dokument beschrieben. Folgende Aspekte sollten in der Planung und Teilplanung berücksichtigt werden. - Zeitplan - Aufgabenverteilung (siehe Arbeitsweise) - Aspekte der technischen Zusammenarbeit (Datenaustausch, Versionierung) - Modularisierung, Strukturierung und Dokumentation des Quellcodes - Verwendung von Klassen und Funktionen

2.3.1. Quellcodeverwaltung und Versionierung

Die zu entwickelnde Software soll während der Entwicklung versioniert werden. Die Versionierungsverwaltung mittels einer passenden Ordnerstruktur (z.B. über Dateinamen und Archivordner) oder unter Verwendung von Git. In diesem Zusammenhang muss in der Planung geklärt werden wie Daten in den Teams ausgetauscht werden.

2.3.2. Dokumentation

Während der Projektarbeit soll begleitend eine Dokumentation angefertigt werden. Das soll zum einen dabei helfen den Projektfortschritt besser nachvollziehen zu können. Andererseits soll damit auch eine Grundlage für den Projektabschluss vorbereitet werden, damit Schnittstellen, Bedienungsweisen, erreichte Ziele, aber auch Abweichungen vom Projektziel oder Einschränkungen dokumentiert werden.

3. Anforderungsbeschreibung

Im Folgenden werden die Anforderungen an die Software beschrieben.

3.1. Klassen

Das Projekt soll objekt-orientiert umgesetzt werden. Beschrieben Klassen können gegebenenfalls erweitert werden. Finales Ziel sind die Klassen *BaseCar* und *SensorCar*. Die Klasse *BaseCar* wird in der Projektphase 2 die Base für weitere KLassen bilden.

1. **Verwendung der Basisklassen:** Die Basisklassen werden im File *basisklassen.py* zur Verfügung gestellt. Sie umfassen die Klassen *BackWheels*, *FrontWheels*, *Ultrasonic* und *Infrared*. (Klären Sie die Bedeutung der einzelnen Argumente der Konstruktoren und den Zwecke der einzelnen Klassenmethoden. Das File *basisklassen.py* beinhaltet weitere Klassen, welche von den Basisklassen verwendet werden. Mit diesen brauchen Sie sich nicht auseinandersetzen oder sie verstehen.)
2. **Modularisierung:** Die Software ist derart zu strukturieren, dass alle notwendigen Klassen und/oder Funktionen in einem oder mehreren Modulen zusammengefasst werden, welche in dem eigentlich auszuführenden Hauptprogramm verwendet werden.
3. **Klasse - BaseCar:** Entwicklung und Testen einer Klasse *BaseCar* mittels der Basisklassen mit vorgegebenen Anforderungen. Die Klasse soll folgende Properties, Attribute und Methoden besitzen.

   Properties:

   - *steering_angle*: Setzen des Lenkwinkels und Rückgabe des aktuellen Lenkwinkels (Property mit Setter)
     - Der Lenkelwinkel soll durch den Setter Integers im Interval von 45 bis 135 Grad begrenzt werden. Dabei entspricht 90 Grad geradeaus. Falls ein zu hoher Wert gesetzt wird, soll der Stetter den maximalen Wert für die jeweilige Richtung verwenden.
   - *speed*: Setzen der Geschwindigkeit und Rückgabe der aktuellen Geschwindigkeit (Property mit Setter)

  - Der Wert für die Geschwindigkeit soll durch den Setter als Integer auf den Interval von -100 bis 100 begrenzt sein. Negative Werte führen zu einer Rückwärtsfahrt, der Wert Null zum Stillstand.
- *direction*: Rückgabe der aktuelle Fahrrichtung (1: vorwärts, 0: Stillstand, -1 Rückwärts) (Property ohne Setter)

Methoden:

- *drive()*: Methode zum Setzen von Geschwindigkeit und Lenkwinkel
  - Die Methode soll das flexible Setzen beider Parameter erlauben. Falls nur ein Parameter gesetzt wird, soll der momentan aktuelle Wert des fehlenden Parameters beibehalten werden.
  - Die Methode soll auf den Properties beruhen bzw. diese verwenden.
- *stop()*: Methode zum Anhalten des Autos. Sie setzt die Geschwindigkeit auf Null.

Prüfen Sie sorgsam, ob die Properties *steering_angle*, *speed* und *direction* immer die korrekten Werte liefern. Dies muss unabhängig von der Verwendungsgeschichte einer korrekten Instanz gewährleistet sein.

Die Klasse *BaseCar* soll mittels der folgenden Aufgaben getestet werden. Dabei ist in Instance der Klasse Basecar Gegenstand verschiedener Steueranweisungen.

- **Fahrmodus 1 - Vorwärts und Rückwärts**: Das Auto fährt mit langsamer Geschwindigkeit etwa 3 Sekunden geradeaus, stoppt für etwa 1 Sekunde und fährt dann etwa 3 Sekunden rückwärts.
- **Fahrmodus 2 - Kreisfahrt mit maximalem Lenkwinkel**: Das Auto fährt 1 Sekunde geradeaus, dann für 8 Sekunden mit maximalen Lenkwinkel im Uhrzeigersinn und stoppt. Dann soll das Auto diesen Fahrplan in umgekehrter Weise abfahren und an den Ausgangspunkt zurückkehren. Die Vorgehensweise soll für eine Fahrt im entgegengesetzten Uhrzeigersinn wiederholt werden.

4. **Klasse - SonicCar**: Eine Klasse *SonicCar* soll die Eigenschaften der Klasse *BaseCar* erben und zusätzlich den Zugriff auf den Ultraschallsensor ermöglichen. Dazu soll in *SonicCar* eine Methode *get_distance* implementiert werden, welche den Wert des Ultraschallsensor abfragt und zurückgibt. Gestalten Sie die Klasse derart, dass mittels einer Instanz von *SonicCar* folgende Fahrmodi mittels Methodenaufruf gestartet werden können.

   - **Fahrmodus 3 - Vorwärtsfahrt bis Hindernis**: Fahren bis ein Hindernis im Weg ist und dann stoppen.
   - **Fahrmodus 4 - Erkundungstour**: Das Auto soll bei freier Fahrt die Fahrrichtung, und optional auch die Geschwindigkeit, variieren. Im Falle eines Hindernisses soll das Auto die Fahrrichtung ändern und die Fahrt dann fortsetzen. Zur Änderung der

Fahrrichtung ist dabei ein maximaler Lenkwinkel einzuschlagen und rückwärts zu fahren. Als Ergebnis soll das Auto den hindernsfreien Raum “erkunden”. Die genaue Gestaltung obliegt Ihnen.

Während dieser Fahrten sollen Daten aufgezeichnet werden, sodass diese nach der Fahrt von der Instanz abgefragt werden und beispielsweise gespeichert oder anderweitig verarbeitet werden können. Diese Daten beschreiben den Status des Autos (Geschwindigkeit, Fahrtrichtung, Lenkwinkel) und die Daten des Ultraschallsensors. Die Aufzeichnung/Speicherung der Daten soll für jede Steueranwesiung geschehen und somit die Fahr bzw. deren Steuerung protokolieren.

5. **Visualiserung der Fahrdaten mit Dash**: Die aufgezeichneten Daten sollen mittels einer App visualisiert werden können. Dazu ist eine App mit *Ploty Dash* zu entwickeln.

- **Dash Stufe 1**: Eine erste Entwicklungsstufe der App soll eine passende Überschrift und KPIs (z.B. in Form von Karten) in der App angezeigt werden. Folgende KPIs sollen basierend auf den aufgezeichneten Fahrdaten ermittelt und angezeigt werden:
  - die maximale Geschwindigkeit,
  - die minimale Geschwindigkeit
  - die Durchschnittsgeschwindigkeit, sowie
  - die zurückgelegte Gesamtfahrstecke und
  - Gesamtfahrzeit.

  Dabei soll der gesetzt Werte der Geschwindigkeit wie eine echte Geschwindigkeit (mit Einheit) behandelt werden. Entsprechend ergibt sich die Strecke aus dem Produkt von Geschwindigkeit und Fahrzeit. Die Einheit der Geschwindigkeit kann vernachlässigt bzw. muss nicht ermittelt werden. Berücksichtigen Sie, dass die konkrete Berechung der KPIs aus der Art der Aufzeichnung der Fahrdaten ergeben kann.
- **Dash Stufe 2**: Die zeitliche Entwicklung ausgewählter Fahrdaten soll grafisch dargestellt werden.
- **Dash Stufe 3**: Die App soll um ein interaktives Dropdown-Menü erweitert werden. Damit sollen die jeweiligen Fahrdaten für die Darstellung der zeitlichen Entwicklung gewählt werden können.

6. **Klasse - SensorCar**: Die Klasse *SensorCar* soll ebenfalls von *BaseCar* erben und neben dem Zugriff auf den Ultraschallsensor *zusätzlich* den Zugriff auf die Infrarotsensoren erlauben. Mittels der Infrarotsensoren soll das Auto in die Lage versetzt werden eine Linie auf dem Boden zu erkennen und dieser zu folgen. Die Daten der Infrarotsensoren sollen ebenfalls aufgezeichnet werden.

Anmerkung: Die Sensitivität der Infrarotsensoren kann durch das blaue Potentiometer eingestellt werden. Dies kann zur erheblichen Verbesserung der Ergebnisse führen.

- **Fahrmodus 5 - Linienverfolgung** : Folgen einer etwas 1,5 bis 2 cm breiten Linie auf dem Boden. Das Auto soll stoppen, sobald das Auto das Ende der Linie erreicht hat. Als Test soll eine Linie genutzt werden, die sowohl eine Rechts- als auch eine Linkskurve macht. Die Kurvenradien sollen deutlich größer sein als der maximale Radius, den das Auto ohne ausgleichende Fahrmanöver fahren kann.
- **Fahrmodus 6 - Erweiterte Linienverfolgung**: Folgen einer Linie, die sowohl eine Rechts- als auch eine Linkskurve macht mit Kurvenradien kleiner als der maximale Lenkwinkel. Diese Linie *kann* eine eine geschlossen Linie sein, sodass das Auto mehrfache Runden abfahren kann.
- **Fahrmodus 7 - Erweiterte Linienverfolgung mit Hindernisserkennung (Optional)**: Kombination von Linienverfolgung per Infrarot-Sensor und Hinderniserkennung per Ultraschall-Sensor. Das Auto soll einer Linie folgen bis ein Hindernis erkannt wird und dann anhalten.

3.2. Nutzer Interface

Die Software soll dahin gehen erweitert bzw. zusammengefasst, dass dem Nutzer erlaubt wird, jeden der Fahrmodi mittels der Klasse SensorCar auszuwählen und zu starten. Dies kann über eine einfache Menüführung im Terminal oder optional durch eine Erweiterung der Funktionalität der Dash-App geschehen. Dieser Teil der Software soll als separater Quellcode die Klasse SensorCar verwenden.

Es soll eine kurze Anwendungsdokumentation für den Nutzer zur Verfügung gestellt werden.

3.3. Dokumentation

Die Dokumentation des erstellten Quellcodes soll als Minimalanforderung die Verwendung von *Doc-Strings* für Funktionen, Klassen und Methoden umfassen.

3.4. Präsentation der Software

Das fertige “Produkt” ist in einer abschließenden Präsentation vorzustellen. Die Präsentation sollte zum einen eine kurze an den potentiellen Anwender adressiert Demonstration beinhalten und zum anderen die Konzeption der Software (File, Klassen, Funktionen) erläutern.

A. Anhang

A.1. Links

A.2. Weiterführende Dokumente