Lastenheft der Projektphase I

Robert Heise und Florian Edenhofner –

Education4Industry GmbH

UNIVERSITY
4 INDUSTRY

Zuletzt aktualisiert: 17.03.2022

Inhaltsverzeichnis

1. Projektauftrag

Titel: Camp 2 Code Projektphase I

Thema: Raspberry Pi Car – ein kleines autonomes Auto

Zeitraum: 21.03.2022 – 01.04.2022

1.1. Beschreibung des Projektziels

Der im Bootcamp verwendete Raspberry Pi soll nun in ein Modellauto integriert werden. Dazu steht ein Bausatz zur Verfügung, welcher ein lenk- und fahrbares Chassis sowie einen Ultraschallsensor und einen Helligkeitssensor umfasst. Die Implementierung der Software erfolgt mit Python. Es werden mehrere Basisklassen zu Verfügung gestellt, die den Zugriff auf die Motoren und Sensoren erlauben. Ziel der ersten Woche ist es die Bauteile und Sensoren des Autos ansteuern bzw. auslesen zu können und erste Fahrparcours zu erledigen. Das Ziel der zweiten Woche ist es einer Linie zu folge. Zudem sollen die Fahrtdaten des Autos (Informationen über Bauteile und Sensoren während der Fahrt) aufgezeichnet und visualisiert werden.

2. Vorgehen

1. Planung

- Projektzieldefinition und -abgrenzung (schriftlich festhalten)
- Projektzeitplan (Kanban-Board)
- Teamzusammensetzung (siehe Arbeitsweise)

2. Aufbau des Modellautos (Eine Anleitung zum Aufbau des Modellautos wird gestellt.)
3. Versionierungsverwaltung mittels einer passenden Ordnerstruktur (z.B. über Dateinamen und Archivordner) oder unter Verwendung von Git.
4. Notwendige Python-Files downloaden. Die entsprechenden Links zu den unterschiedlichen Repositorys werden gesondert zur Verfügung gestellt.
5. Datenübertragung auf den Raspberry Pi testen.
6. Funktionstests und gegebenenfalls Bugfixes
7. Testen der Basisklassen. Die Basisklassen umfassen die Klassen Back_Wheels, Front_Wheels, Ultrasonic und Infrared.

8. **Klasse - BaseCar:** Entwicklung und Testen einer Klasse BaseCar mittels der Basisklassen mit vorgegebenen Anforderungen. Die Klasse soll folgende Attribute (mit entsprechenden Gettern und Settern) und Methoden haben:

- *steering_angle*: Zugriff auf den Lenkwinkel
- *drive(int, int)*: Fahren mit übergebener Geschwindigkeit und Fahrrichtung
- *stop()*: Anhalten des Autos
- *speed*: Zugriff auf die Geschwindigkeit
- *direction*: Zugriff auf die Fahrrichtung (1: vorwärts, 0: Stillstand, -1 Rückwärts)

  Die Klasse BaseCar soll mittels folgenden Aufgaben getestet werden.

- **Fahrparcours 1 - Vorwärts und Rückwärts**: Das Auto fährt mit langsamer Geschwindigkeit 3 Sekunden geradeaus, stoppt für 1 Sekunde und fährt 3 Sekunden rückwärts.
- **Fahrparcours 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.

9. **Klasse - SonicCar**: Eine Klasse SonicCar soll die Eigenschaften der Klasse BaseCar erben und zusätzlich den Zugriff auf den Ultraschall-Sensor ermöglichen. Die Fahrdaten umfassen die Geschwindigkeit, die Fahrtrichtung, den Lenkwinkel und die Daten des Ultraschall-Sensors. Dazu soll Folgendes entwickelt und getestet werden.

- **Fahrparcours 3 - Vorwärtsfahrt bis Hindernis**: Fahren bis ein Hindernis im Weg ist und dann stoppen. Während dieser Fahrt sollen die Fahrdaten (Geschwindigkeit, Lenkwinkel, Fahrrichtung, Sensordaten) aufgezeichnet werden.
- **Fahrparcours 4 - Erkundungstour**: Das Auto soll geradeaus fahren und im Falle eines Hindernisses die Fahrrichtung ändern und die Fahrt fortsetzen. Zur Änderung der Fahrrichtung soll ein maximaler Lenkwinkel eingeschlagen und rückwärts gefahren werden. Optional können sowohl die Geschwindigkeit als auch die Fahrtrichtung bei freier Fahrt variiert werden. Zusätzlich sollen die Fahrdaten aufgezeichent werden.

10. **Visualiserung der Fahrdaten mit Dash**: Die aufgezeichneten Sensordaten und Fahrdaten sollen mittels einer Dash-App visualisiert werden. Dazu soll eine Dash-App entwickelt werden.

- **Dash Stufe 1**: Eine Dash App soll erstellt werden. Dazu soll eine passende Überschrift und KPIs (in Form von Karten) in einer Dash App angezeigt werden. Es sollen folgende KPIs basierend auf den aufgezeichneten Fahrdaten erstellt werden: Maximale

Geschwindigkeit, minimale Geschwindigkeit und Durchschnittsgeschwindigkeit, zurückgelegte Gesamtfahrstecke, Gesamtfahrzeit. Die zurückgelegte Fahrstrecke soll aus den aufgezeichneten Fahrdaten ermittelt werden.

- **Dash Stufe 2**: Die zeitliche Entwicklung einer ausgewählten Fahrdaten soll grafisch dargestellt werden.
- **Dash Stufe 3**: Die Dash App soll um ein interaktives Dropdown Menü mit verschiedenen Fahrdaten erweitert werden. Damit sollen die jeweiligen Fahrdaten für die Darstellung der zeitlichen Entwicklung dargestellt werden.

11. **Klasse - SensorCar**: Die Klasse SensorCar soll zusätzlich den Zugriff auf die Infrarot-Sensoren ermöglichen. Mittels dieser Sensoren soll das Auto in die Lage versetzt werden eine Linie auf dem Boden zu erkennen. Die Daten der Infrarotsenoren sollen ebenfalls aufgezeichet werden. **Anmerkung**: Die Sensitivität der Infrarotsensoren kann durch das blaue Potentiometer eingestellt werden. Dies kann zur erheblichen Verbesserung der Ergebnisse führen.

    - **Fahrparcours 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.
    - **Fahrparcours 6 - Erweiterte Linienverfolgung**: Folgen eine Linie, die sowohl eine Rechts- als auch eine Linkskurve macht mit Kurvenradien kleiner als der maximale Lenkwinkel.

12. **Fahrparcours 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.
13. Vollständige Dokumentation des Vorgehens und des erstellten Codes
14. Präsentation der Ergebnisse

3. Projektaufbau

Die Teams haben die Möglichkeit die Softwarearchitektur flexibel selbst festzulegen. Es sollen dafür auf Basisklassen für die Motorsteuerung und Sensorabfragen genutzt werden.

3.1. Arbeitsweise

Das Projekt sollte als Teamarbeit in einer Gruppe von fünf Personen unter Anwendung einer agilen Methode durchgeführt werden. Arbeitspakete sollten als kleine Einheiten (User-Stories) beschrieben werden, die in einem Kanban- oder Scrum-Board organisiert sind. So lassen sich Aufgaben innerhalb des Teams leicht zuweisen. Das Team wählt seine Arbeitsweise selbst: Es kann zusammen, zu Pairs oder allein gearbeitet werden. Wichtig ist, dass die Arbeitsschritte aufeinander abgestimmt sind und agile Rituale wie das “Daily Scrum” durchgeführt werden.

Es werden Projektteams zu je ca. 5 Personen gebildet. In jedem Team wird die Rolle des Scrum-Masters vergeben. Das Team ist für alle Arbeiten am Produkt verantwortlich: Analyse, Entwurf, Entwicklung, Tests und Dokumentation.

3.2. Dokumentation

Während der Projektarbeit soll begleitend eine Dokumentation angefertigt werden. Das soll zum einen dabei Helfen das gelernte festzuhalten und besser nachvollziehen zu können. Andererseits soll damit auch eine Grundlage für den Projektabschluss (siehe 14. und 15.) vorbereitet werden, damit Schnittstellen, Bedienungsweisen, erreichte Ziele, aber auch Fehler oder Einschränkungen dokumentiert werden.

3.3. Vorstellung der Ergebnisse

Am letzten Tag der Projektphase soll jedes Team das Ergebnis kurz präsentieren. Dabei soll auch auf die grundlegende Struktur der erstellten Software eingegangen werden.

A. Anhang

A.1. Links

A.2. Weiterführende Dokumente

A.3. Session-Info

```
sessionInfo()
## R version 4.1.3 (2022-03-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=German_Germany.1252  LC_CTYPE=German_Germany.1252    LC_MONETARY=German_Germany.1252
## [4] LC_NUMERIC=C                    LC_TIME=German_Germany.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] forcats_0.5.1   stringr_1.4.0   purrr_0.3.4     tibble_3.1.4    ggplot2_3.3.5   tidyverse_1.3.1 ret
##  [8] bookdown_0.24   tidyr_1.1.4     dplyr_1.0.7     formatR_1.11    readr_2.0.2     knitr_1.36
##
## loaded via a namespace (and not attached):
##  [1] tinytex_0.34     tidyselect_1.1.1 xfun_0.26        haven_2.4.3      lattice_0.20-45  colorspace_2.0
##  [7] vctrs_0.3.8      generics_0.1.0   htmltools_0.5.2  yaml_2.2.1       utf8_1.2.2       rlang_0.4.11
## [13] pillar_1.6.3     withr_2.4.2      glue_1.6.2       DBI_1.1.1        dbplyr_2.1.1     readxl_1.3.1
## [19] modelr_0.1.8     lifecycle_1.0.1  cellranger_1.1.0 munsell_0.5.0    gtable_0.3.0     rvest_1.0.1
## [25] evaluate_0.14    tzdb_0.1.2       fastmap_1.1.0    fansi_0.5.0      broom_0.7.9      Rcpp_1.0.7
## [31] backports_1.2.1  scales_1.1.1     jsonlite_1.7.2   fs_1.5.0         hms_1.1.1        png_0.1-7
## [37] digest_0.6.28    stringi_1.7.4    rprojroot_2.0.2  grid_4.1.3       cli_3.2.0        tools_4.1.3
## [43] magrittr_2.0.1   crayon_1.4.1     pkgconfig_2.0.3  ellipsis_0.3.2   Matrix_1.4-0     xml2_1.3.2
## [49] reprex_2.0.1     lubridate_1.7.10 assertthat_0.2.1 rmarkdown_2.11.3 httr_1.4.2       rstudioapi_0.1
## [55] R6_2.5.1         compiler_4.1.3
```