Institut für Technik der Informationsverarbeitung, KIT

Praktikum Software Engineering (PSE) – SoSe 2019

Prof. Dr.-Ing. E. Sax, Dipl.-Ing. Lukas Erlinghagen, Martin Haas M.Sc.

Spezifikation zum selbstfahrenden „TivSeg“

Zielbeschreibung

- Das übergeordnete Ziel ist die Objekterkennung und -Verfolgung mit einem TivSeg
- Der TivSeg soll selbständig einem voranfahrenden Fahrzeug (mit Marker) folgen können (Platooning)
- Im Ausgangszustand soll zunächst der Marker gesucht werden
- Der Abstand zum erkannten Marker soll konstant eingehalten werden
- Hindernisse im Fahrweg sollen erkannt und berücksichtigt werden
- Zustands- und Debug-Informationen sollen kontinuierlich ausgeben und protokolliert werden
- Konfigurationsparameter und Fahrbefehle sollen während des Betriebs eingegeben / verändert werden können

Anforderungen

- Die Bild- und Abstandsinformationen zur Marker- und Hinderniserkennung werden über eine RealSense-Kamera bereitgestellt. Zur Objekterkennung existiert ein vorhandenes Modul für die Bildauswertung („Region Growing“ Algorithmus). Die Schnittstelle dieser Vorverarbeitung sowie zur Abfrage der Abstandsinformationen ist unter „Schnittstellenspezifikation (partiell) Input-Interface von der Kamera“ angegeben.
- Wird kein Marker erkannt, soll z.B. durch eine einfache Drehung nach einem Marker gesucht und dieser aufgefunden werden.
- Anhand der in der Bildauswertung gewonnenen Informationen sollen Entscheidungen für die Steuerung des TivSeg getriggert werden können. Dabei soll eine Infrastruktur geschaffen werden, die es erlaubt, dass verschiedene Komponenten angemeldet werden können, die dann die entsprechenden Aktionen (auch nebenläufig) auslösen (publish-subscribe pattern):
    - Markersuche
    - Verfolgungsmodus
    - Hindernisvermeidung
- Abstand zum vorausfahrenden Fahrzeug und Lenkwinkel sollen mit Hilfe von PID-Reglern konstant gehalten werden. Ein entsprechendes Modul für die Motoransteuerung ist vorhanden und soll eingebunden werden. Die Schnittstelle dieses Moduls ist unter „Schnittstellenspezifikation (partiell) Motor-Regler“ angegeben
- Im ersten Entwicklungsschritt soll statt der automatischen Abstandsregelung ein Steuerungsmodul mit derselben Schnittstelle eingesetzt werden. Dieses soll die Fahrbefehle (Geschwindigkeit, Lenkung) über ein Terminal entgegennehmen.
- Die u.a. Konfigurationsparameter und Fahrbefehle (siehe „Befehlsspezifikation Fahrbefehle und Konfiguration“) sollen über das Terminal während des Betriebs eingeben werden können. Dabei sollen sich geänderte Parameter direkt auf das laufende System auswirken.
- Alle Eingabebefehle sollen auch per Skript ausführbar sein. Ein Skript besteht aus einer Abfolge der unten definierten Befehle.
- Die Zustands- und Debug-Informationen von allen Teilkomponenten des Systems sollen sowohl auf die Konsole ausgegeben werden können, als auch in eine Log-Datei geschrieben werden. Dabei sollen globale Zeitstempel mit ausgegeben werden und der Detailgrad des Loggings einstellbar sein. Die Grundfunktionalität ist dabei bereits vorgegeben (Klasse „Diagnostics“)
- Auf Modulebene sollen sämtliche Komponenten testbar sein. Hierfür sollen Unit-Tests fortlaufend während des Entwicklungsprozesses angewandt werden.

- Vor der Implementierung auf der Zielhardware soll die eigentliche Fahrfunktion mittels Simulation in CarMaker getestet werden. Dies muss beim Software-Entwurf berücksichtigt werden, indem darauf geachtet wird, dass wenig Kopplung zwischen den Komponenten der Fahrfunktion und dem restlichen System existiert.

Schnittstellenspezifikation (partiell) Input-Interface von der Kamera

- **class SensorManager**
  - `bool` **`runOnce`**`();`

    Soll periodisch aufgerufen werden um ein neues Bild zu holen und zu verarbeiten. Der Rückgabgewert gibt an, ob die Operation erfolgreich war.

  - `std::vector<MarkerInfo>&` **`get_marker_list`**`();`

    Gibt eine Liste aus erkannten Markern im zuletzt verarbeiteten Bild zurück. Eine `MarkerInfo` enthält dabei eine `BoundingBox` für einen erkannten Marker sowie einen Wert [0; 100], welcher die Zuverlässigkeit der Erkennung wiedergibt. „100“ steht dabei für die höchste Zuverlässigkeit.

  - `double` **`get_depth`**`(int x, int y);`

    Gibt die Tiefeninformation eines Pixels (x|y) im zuletzt verarbeiteten Bild in Metern zurück

Schnittstellenspezifikation (partiell) Motor-Regler

- **class DriveController**
  - `void` **`updateController`**`(int speed, int steering);`

    Setzt neue Werte für die Geschwindigkeit und die Lenkung als Eingabe an den Regler. Liegt der letzte `update()` Aufruf mehr als 500ms zurück, schaltet der Regler die Motoren auf stopp (Sicherheitsabschaltung). Der Wertebereich beträgt jeweils +/-100, wobei keine direkte Korrelation zwischen Wert und einer (Winkel-)Geschwindigkeit vorliegt. -100 als steering-Wert bedeutet, dass das linke Rad stehenbleibt und das rechte Rad mit der vorgegebenen Geschwindigkeit dreht.

  - `void` **`updateDirect`**`(int speed, int steering);`

    Setzt neue Werte für die Geschwindigkeit und die Lenkung ohne Regler für den Testbetrieb im aufgebockten Modus. Wird nur für die erste Testphase (statt `update()`) benötigt. Werte wie bei updateController

  - `void` **`updateDirectMotor`**`(int speedLeft, int speedRight);`

    Setzt neue Werte für die Motoren ohne Regler für den Testbetrieb im aufgebockten Modus. Wird nur für die erste Testphase (statt `updateController()`) benötigt.

  - `void` **`step`**`();`

    Diese Methode muss unabhängig von der restlichen Anwendung alle 10ms aufgerufen werden.

Befehlsspezifikation Fahrbefehle und Konfiguration

- **set -l <*value*>**

  Setzt den Grad der Logging Informationen <*value*>, welche von der Anwendung ausgegeben werden. 0 = Error, 1 = Warning, 2 = Info, 3 = Debug, 4 = Verbose

- **set -t <*value*>**

  Setzt die maximale Zeit <*value*>, die der TivSeg ohne weitere Eingabe fahren darf. Nach dieser Zeit muss ein Nothalt erfolgen.

- **set -d <*value*>**

  Setzt den angepeilten Abstand <*value*> zum Marker der automatischen Steuerung.

- **drive <*time*> <*velocity*> <*steering*>**

  Fahre eine Zeit <*time*> [0 – max] mit Geschwindigkeit <*velocity*> [-100; 100] und einer Lenkung <*steering*> [-100; 100]. Die Lenkung hat einen default Wert von 0, was einer geraden Ausrichtung entspricht.

- **script <*file*>**

  Führe ein Skript aus <*file*> aus. Das Skript soll alle vorgestellten Befehle enthalten können. Jeder Befehl blockiert bis er vollständig ausgeführt wurde.

- **STRG + c**

  Bricht die aktuelle Befehlsausführung sofort ab und kehrt in das Terminal zurück. (Hinweis: Funktion signal())

- **search**

  Sucht einen Marker in der Umgebung.

- **mode <*mode*>**

  Setzt den Fahrmodus <*mode*> [1:automatic; 2:manual]

  Alle angegebenen Befehle sollen eine entsprechende Fehlermeldung ausgeben, sofern ihre Aktion nicht ausgeführt werden konnte.

Aufgaben im Praktikum

1. Übersetzen der Anforderungen aus dieser Spezifikation in Use-Case und Sequenzdiagramme
2. Ableiten eines Softwaredesigns aus der Spezifikation und den Diagrammen (Klassendiagramm)
3. Design-Review mit dem Betreuer
4. Implementierung von Software und Testfällen
5. Fahrfunktion testen in CarMaker
6. Komponenten-Tests (Unit-Tests) durchführen (mit Testaufbau am Platz)
7. Implementierungs-Review mit dem Betreuer
8. Integrations-Tests durchführen (mit Ziel-Hardware im aufgebockten Modus)
9. Abnahme-Tests durchführen (mit Ziel-Hardware im autonomen Modus)

Liefergegenstände

1. Systemdesign in Enterprise Architect (Use Cases, Sequenz-Diagramme, Klassen-Diagramm)
2. Test-Projekt in CarMaker
3. Software-Projekt (über KIT-Gitlab)
4. Begleitdokumentation (z.B. per Doxygen mit im Softwareprojekt)