Dokumentation

OpenPEARL

Semesterprojekt WS17/18

Referent	:	Prof. Dr. Rainer Müller
Korreferent	:	-
Vorgelegt am	:	18.12.2017
Vorgelegt von	:	Kevin Hertfelder Stefan Kienzler Patrick Kroner Daniel Petrusic Daniel Schlageter

Inhaltsverzeichnis

1 Einleitung

Das Semesterprojekt OpenPEARL im Wintersemester 2017/18 beschäftigt sich mit der Anwendung von OpenPEARL insbesondere im Zusammenhang mit Sensorik.

1.1 OpenPEARL

PEARL steht für *Process and Experiment Automation Realtime Language*, eine höhere Programmiersprache, die speziell dafür entworfen wurde, Multitaskingaufgaben bei der Steuerung technischer Prozesse zu steuern. Die Sprache wurde um 1975 am IRT Institut der Leibniz Universität in Hannover entwickelt und 1998 vom Deutschen Institut für Normung in der DIN66253-2 standardisiert (http://pearl90.de).
OpenPEARL ist eine quelloffene Implementierung eines Compilers und einer Laufzeitumgebung für PEARL. OpenPEARL befindet sich aktuell noch in der Entwicklung, ist aber weit genug fortgeschritten, um erste Beispielprojekte damit umzusetzen. In dieser Dokumentation werden deshalb auch in OpenPEARL gefundene Fehler festgehalten.

1.2 Projektziel

Ziel des Projektes ist es, das OpenPEARL-System bei der Ansteuerung verschiedenartiger Peripheriegeräte zu testen. Die verschiedenen Sensoren und Aktoren sollen auf ihre Nutzbarkeit mit OpenPEARL allgemein und in einem größeren OpenPEARL-Projekt im Speziellen überprüft werden. Als Plattform wird ein Raspberry Pi 3 genutzt.
Als Gesamtsystem soll ein autonomes Modellauto entstehen, das, mit ebenjenen Sensoren ausgestattet, nach Möglichkeit einer Linie folgen und auf Abruf weitere Manöver ausführen kann.

1.3 Über dieses Dokument

Diese Dokumentation beschreibt die Planung, Struktur und die Ergebnisse des Projektes. Sie beschreibt nicht das vollständige Vorgehen oder alle genutzten

Informationen, sondern beschreibt die Ergebnisse unter Erwähnung wichtiger Ereignisse und mit Verweisen auf genutzte Quellen.
Der Programmcode sowohl für einzelne Module als auch für das Gesamtsystem ist in einem öffentlichen Git-Repository abgelegt, auf das hier und an geeigneter Stelle verwiesen wird. Der Programmcode ist dokumentiert und nicht Teil dieses Dokumentes, stattdessen werden in diesem Dokument die zugrundeliegenden Ideen erläutert, die notwendig sind, um den dokumentierten Code vollständig zu verstehen.

2 Projektverlauf

2.1 Organisation

Das Projektteam besteht aus den fünf Mitgliedern Kevin Hertfelder, Stefan Kienzler, Patrick Kroner, Daniel Petrusic und Daniel Schlageter. Betreut wird das Projekt durch Herrn Prof. Dr. Rainer Müller. In wöchentlichen Projekttreffen wird er aktuelle Status festgehalten und die Aufgaben für die nächste Woche, gemäß angepasster Planung, festgelegt. Die Protokolle dieser Sitzungen können im Repository eingesehen werden.

2.2 Struktur

Das Projekt gliedert sich, entsprechend des oben beschriebenen Ziels, in mehrere Teilprojekte:

- **Projektstart**
  Einführung in das Projekt und Festlegung der Organisation. Grundlegende Abstimmung über Ziel und Umfang.
- **Einführung OpenPEARL und Philosophenproblem**
  Zur Einarbeitung in die Programmiersprache OpenPEARL implementiert das Projektteam jeweils das Philosophenproblem.
- **Einrichtung der Raspberry Pis**
  Für die weitere Arbeit am Projekt werden zwei Rasperberry Pi 3 eingerichtet, um mit OpenPEARL und NFS zu arbeiten. Der Zugriff auf die Rechner soll mittels *ssh* möglich sein.
- **Inbetriebnahme und Test der Hardwarekomponente**
  Die einzelnen Komponenten (Sensoren und Aktoren) werden als Teilprojekte separat am Raspberry Pi in Betrieb genommen und getestet.
- **Entwurf und Druck des Modells**
  Das Modell für den mechanischen Aufbau wird entwickelt und mit dem 3D-Drucker ausgedruckt und getestet.

- **Detailplanung des Gesamtsystems**
  Die genaue Funktionalität und der Hardwareaufbau des Gesamtsystems werden festgelegt und dokumentiert.

- **Zusammensetzung Gesamtsystems**
  Nachdem alle Komponenten erfolgreich in Betrieb genommmen und getestet wurden, wird das Gesamtsystem im Sinne des Ziels zusammengebaut. Daraufhin wird die Funktionalität des Gesamtsystems programmiert. Im Sinne eines inkrementell iterativen Vorgehens erfolgen Anforderungsanalyse, Implementierung und Tests bis das System die Anforderungen erfüllt.

- **Projektabschluss**
  Zum Abschluss des Projekts erfolgt die Fertigstellung der Dokumentation und Abnahme des Ergebnisses.

- **Projektvorstellung**
  Das Ergebnis des Projektes wird im Rahmen der Projektpräsentationen vorgestellt.

2.3 Ablauf

Tabelle 1 detailliert die grobe Zeitplanung für den Projektablauf. Änderungen sind vorbehalten.

Zeitraum	**Teilprojekt / Aufgabe**
09.10.17 – 16.10.17	Projektstart
16.10.17 – 23.10.17	Einführung in OpenPEARL und Philosophenproblem
23.10.17 – 30.10.17	Einrichtung der Raspberry Pis
30.10.17 – 11.12.17	Inbetriebnahme und Test der Hardwarekomponenten
30.10.17 – 18.12.17	Entwurf und Druck des Modells
11.12.17 – 15.01.18	Zusammensetzung des Gesamtsystems
15.01.18 – 22.01.18	Projektabschluss
26.01.18	Projektpräsentation

Tabelle 1: Zeitplan

3 Einführung in OpenPEARL

Um mit der Sprache OpenPEARL und ihren Konstrukten insbesondere zur Nebenläufigkeit vertraut zu werden, wurde als einführendes Beispiel das Philosophenproblem gelöst. Eine allgemeine Einführung in die Sprache PEARL ist nicht Teil dieser Dokumentation, stattdessen wird auf den Sprachreport verwiesen.

3.1 Das Philosophenproblem

Das Philosophenproblem ist ein bekanntes Synchronisationsproblem. Fünf Philosophen sitzen um einen runden Tisch, jeder der Philosophen hat einen Teller mit Spaghetti vor sich. Um diese zu essen, benötigt jeder Philosoph zwei Gabeln, es sind jedoch nur fünf Gabeln, jeweils eine zwischen zwei Philosophen, vorhanden. Es können also nicht alle Philosophen gleichzeitig essen.
Die Philosophen werden als Prozesse oder Threads betrachtet, die gleichzeitig auf mehrere Ressourcen, die Gabeln, zugreifen möchten.

3.2 Umsetzung

Die Gabeln werden als Semaphoren realisiert. Diese bieten nur einer bestimmten Anzahl Threads bzw. im Kontext von OpenPEARL *Tasks* die Möglichkeit der gleichzeitigen Belegung. Beim Philosophenproblem kann jede Gabel, also jede Semaphore, nur einmal belegt werden.
Zum Essen benötigt ein Philosoph zwei Gabeln, also auch zwei Semaphoren. Erfolgt der Zugriff auf die Semaphoren nicht-atomar, das heißt voneinander getrennt und nacheinander, so kann es zur Verklemmung kommen: Jeder Philosoph hält eine Gabel und wartet auf die Freigabe der zweiten — die Philosophen verhungern.
OpenPEARL bietet zur einfachen Lösung des Problems die Möglichkeit, mehrere Semaphoren in einer atomaren Aktion gleichzeitig aufzunehmen. Dies ist nur dann erfolgreich, wenn alle Semaphoren belegbar sind, und verhindert eine Verklemmung.

3.3 Codebeispiel

Im Folgenden sind die relevanten Codeausschnitte einer Lösung des Philosophenproblems in OpenPEARL aufgeführt.

3.3.1 Definition der Semaphoren

```
! Semaphoren ("Gabeln"), die maximal ein mal (gleichzeitig) belegt werden koennen
DCL (g1, g2, g3, g4, g5) SEMA PRESET(1,1,1,1,1);
```

3.3.2 Aktivität eines Philosophen als Task

```
philosopher1: TASK;
        REPEAT;
                ! Gleichzeitige Anfrage fuer die Belegung der Semaphoren
                REQUEST g1, g2;
                PUT 'Philosopher 1 starts eating...' TO termout BY A, SKIP;
                AFTER timeone RESUME;
                PUT 'Philosopher 1 stops eating...' TO termout BY A, SKIP;
                ! Gleichzeitige Freigabe der Semaphoren
                RELEASE g1, g2;
        END;
END;
```

Komplette Lösungen können im Ordner *Code/Philosophenproblem* des Repositories eingesehen werden.

4 Einrichtung

Die Einrichtung der Entwicklungs- und Zielumgebung, einer OpenPEARL-Installation auf einem Raspberry Pi 3, erfolgte nach der Anleitung im OpenPEARL Wiki und wird deshalb an dieser Stelle nicht weiter thematisiert.

5 Funktionsbeschreibung

In diesem Kapitel wird die Funktionalität des Gesamtsystems, an der sich dann auch das Vorgehen bei der Inbetriebnahme der einzelnen Komponenten orientiert, kurz beschrieben.

5.1 Grundfunktionalität

Das OpenPEARL SensorCar ist ein mit zwei Motoren und unterschiedlichen Sensoren ausgestattetes Modellauto. Die Sensoren sollen verschiedene Signale aufnehmen, verarbeiten und anzeigen, die Schrittmotoren sollen das Fahren in verschiedenen Geschwindigkeiten und Richtungen ermöglichen. Es werden folgende Komponenten eingesetzt:

- **Raspberry Pi mit OpenPEARL**: Kernkomponente des SensorCar ist ein Raspberry Pi, auf dem ein OpenPEARL Programm läuft, das alle angeschlossenen Sensoren und Aktoren verwaltet und steuert.
- **Schrittmotor**: Zwei Schrittmotoren steuern jeweils zwei über ein Gummi verbundene Räder (ein Motor für eine Seite). Dadurch lässt sich das Auto mit unterschiedlichen Geschwindigkeiten sowohl vorwärts wie auch rückwärts bewegen und Kurven mit beliebigen Radius fahren.
- **Lichtrechen**: Ein Lichtrechen, bestehend aus mehreren binären Helligkeitssensoren, erfasst den Untergrund in zwei Helligkeitsstufen.
- **Modellauto**: Alle Komponenten werden auf einem selbstgedruckten Modellauto aufgebracht.
- **Farbsensor**: Ein Farbsensor kann farbige Punkte auf dem Boden erfassen und verschiedene Aktionen einleiten.
- **Beleuchtung**:

  Weiße LEDs: Zwei weiße LEDs dienen als Frontscheinwerfer.

  Rote LEDs: Zwei rote LEDs dienen als Rückleuchten.

  Gelbe LEDs: Vier gelbe LEDs dienen als Blinker.

5.2 Erweiterte Funktionalität

Zusätzlich zum eingabegesteuerten Fahren soll das SensorCar nach Möglichkeit selbstständig einer schwarzen Linie folgen und verschiedene Aktionen (Anhalten, Geradeausfahren) auf Abruf – durch Erkennung von Farbpunkten auf der Linie – durchführen können.

5.3 Kommunikation

Aktuelle Informationen über das SensorCar können über einen Browser mittel einer *http* Anfrage abgefragt werden. Dazu läuft in OpenPEARL ein Webserver, der die Informationen abfragt und ausgibt.
Eingehende Kommunikation erfolgt über ssh.

6 Komponenten

6.1 Raspberry Pi 3

Als Entwicklungs- und Zielplattform wird ein Raspberry Pi 3 genutzt, auf dem OpenPEARL installiert ist. Die Sensoren und Aktoren sollen wie in Tabelle 2 am Raspberry Pi angeschlossen werden:

Gerät	**BCM**	**Versorgung**
Lichtrechen	6 – 13	3,5 / 5V
Motortreiber	16 – 19, 20 – 23	extern
Farbsensor	2, 3 (I^2C)	3,3 / 5V
Gyroskop	2, 3 (I^2C)	5V
Kompass	2, 3 (I^2C)	5V
LEDs	24 –	5V

Tabelle 2: Anschlussplan Raspberry Pi

Der zugrundeliegende Belegungsplan ist im Anhang 11.1 zu finden.

6.2 Schrittmotor

Die Schrittmotoren werden jeweils über einen Motortreiber und vier GPIO-Pins des Raspberry Pi angesteuert. Energie erhält der Motor durch ein an den Treiber angeschlossenes Netzteil mit 12V Gleichstrom. Durch paarweise Alternieren der Bits an den vier GPIO-Pins wird der Schrittmotor um jeweils einen Schritt bewegt. Es ergibt sich das Bitmuster in Tabelle 3.

Der Schrittmotor kann nun bewegt werden, indem die Bitmuster nacheinander, unterbrochen durch eine Wartezeit, an den Schrittmotor geschickt werden. Die Geschwindigkeit des Schrittmotors kann durch die Wartezeit, die Richtung

1010
1001
0101
0110

Tabelle 3: Bitmuster

durch die Reihenfolge der Signale geändert werden.

Zusätzlich ist zu beachten, dass die Belegung mehrerer Pins des Raspberry Pi abwärts erfolgt, d.h. `RPiDigitalOut(19, 4)` belegt die Pins `16, 17, 18, 19`. Das Speichern von `BIT(4)` Variablen in einem Array funktioniert in OpenPEARL aktuell noch nicht; aufgrund der wenigen Zustände wurde auf einen Workaround verzichtet.

In diesem Schritt wurde als Modul eine Prozedur entwickelt, die einen spezifizierten Motor um eine spezifizierte Schrittzahl in spezifizierte Richtung bewegt. Die Synchronisation zur gleichzeitigen Ansteuerung beider Motoren erfolgte erst im Gesamtsystem. Das komplette Schrittmotormodul kann im Ordner *Code/Komponenten/Schrittmotor* des Repositories eingesehen werden.

6.3 Lichtrechen

Der Lichtrechen ist essenzieller Bestandteil des Autos und ist dafür zuständig die Position des Autos relativ zum vorgegebenen Pfad zu ermitteln. Der Lichtrechen QTR-8A der Marke Pololu besteht aus 8 Reflektionssensoren, die mit infrarot LED's die Reflexivität des Untergrundes auf einer Linie von ca. 7cm ermitteln. In unseren Anwendungsfall soll der Sensor ein schwatzes Klebeband auf einem hellen Untergrund erkennen und diese Information zur Steuerung des Autos weiterleiten.

Die Verbindung zum System geschieht über acht separate Datenpins, die jeweils konstant ein einzelnes Bit aussendet das die Ausgabe eines einzelnen Sensors binär repräsentiert. Des Weiteren gibt es noch einen Abschluss für die Stromzufuhr für 5 und 3.3V, einen Pin für die Erdung und einen Pin mit dem die Infrarot-LED's an oder abgeschaltet werden können. In unserem Aufbau nutzen wir die 5V Stromversorgung und lassen die LED-Steuerungspin unangeschlossen, in welchem Fall die Sensoren dauerhaft eigeschaltet bleiben.

Im Anschlussplan ist vorgesehen, dass der Lichtrechen über die Datenpins 6 -13 am Raspberry Pi ausgelesen wird.

Bei der Auswertung der Datenpins werden den Pins von rechts nach links Gewichtungen von 4 bis -4 gegeben und jeder Pin der wenig Reflexion misst wird seiner Wertung nach aufsummiert. Diese Summe Wird dann durch die Anzahl des Summanden geteilt und ergibt somit einen Durchschnittswert für die Position des Leitstreifens. Dieses Verfahren schwächt auch fehlerhafte Werte einzelner Sensoren ab.

6.4 Farbsensor

Der Farbsensor TCS 34725 Der Marke Adafruit besitzt einen Sensor, der sowohl einen RGB Farbwert als auch die gesamt Helligkeit ermitteln lässt. Um die Farben Best möglichst zu erkennen besitzt der Sensor eine sehr helle LED mit weißem Licht. In unserem Projekt soll der Farbsensor Farbige Signal-Punkte auf der Strecke erkennen die denen das Auto beispielsweise das Ende der Stecke erkennen soll.

Verbunden wir der Sensor im Gegensatz zum Lichtrechen erfolgt die Datenverbindung zum Farbsensor über einen I2C-Bus. Des Weiteren Besitzt der Sensor ebenfalls einen Stromverbindung für 5 und 3.3V, eine Erdung und Einen Pin zur Steuerung der LED.

Die Datenpins 2 und 3 am Raspberry Pi sind standartmäßig für diesen Bus vorkonfiguriert. Die Unterstützung von I2C auf dem Raspberry Pi muss aber erst konfiguriert werden. Dazu muss der Befehl “raspi-config“ ausgeführt werden und dort unter “Advanced Options“ Die Unterstützung in Kernel eingeschaltet werden. Zusätzlich musste in der Datei “/boot/config.txt“ noch die Zeile “ dtparam=i2c1=on“ angefügt werden.

Um den Farbsensor in Pearl benutzen zu können musste dafür noch ein Treiben in C++ geschrieben werden. Dieser umfasst die Dateien “TCS34725.cc“, “TCS34725.h“ und “TCS34725.xml“ dieser Treiber wird in eine Headerfile von OpenPEARL integriert werden und stellt alle Methoden für die Nutzung des Sensors als DATION in OpenPEARL zur Verfügung.

Bei der Auswertung der Rohdaten vom Sensor konvertieren wir die Daten in Standard RGB Werte mit einer Spanne von 0 bis 255.

6.5 TCP/IP Socket

Über das Socket werden die Daten des SensorCars auf einer Webseite angezeigt. Hierbei bekommt das Socket einen html request vom Browser, an den er dann den HTML-Code schickt. Es werden die Geschwindigkeit des rechten sowie des linken Motors, den Status der Blinker und der Farbsensors angezeigt. Es können außerdem Informationen zum Projekt und die Dokumentation aufgerufen werden.

Die Webseite kann im gleichen Netz über <ipadresse>/index.html aufgerufen werden. Damit die Daten immer aktuell sind, wird die Seite alle zwei Sekunden aktualisiert.

OpenPEARL hat von sich aus keine Möglichkeit ein Socket aufzusetzen. Es musste daher zuerst implementiert werden. Hierzu wurde eine C++ Datei

mit dazugehöriger XML Datei geschrieben und dem System hinzugefügt. Die Dateien wurden im Makefile und im `PearlIncludes.h` registriert.

Um ein Socket aufzusetzen muss man im Systemteil von OpenPEARL TcpIpServer + den Port über den man kommunizieren will angeben. Im Problemteil kann man dann über `DATION INOUT` deklarieren. Um nun eine Anfrage erhalten zu können muss man das Socket über `OPEN` öffnen. Die Anfrage kann man dann mit `GET` abfragen und bearbeiten. Mit `PUT` schickt man Daten zurück. Zum Schluss muss man das Socket mit `CLOSE` wieder schließen. Wenn dies in einer Dauerschleife läuft kann jede Anfrage bearbeitet werden.

7 Gesamtsystem

Das SensorCar als Gesamtsystem setzt sich aus dem im Kapitel 6 beschriebenen Komponenten zusammen. Bisher wurde bei der Programmierung der Komponenten oft von Modulen gesprochen, es ist aber zu bemerken, dass die Nutzung des PEARL-Konstrukts `MODULE` aufgrund fehlender Implementierung in OpenPEARL nicht genutzt werden konnte. Im Folgenden wird zuerst die Architektur und dann die Bedienung des SensorCar erläutert. Der vollständige, dokumentierte Code kann auch hierzu im Git-Repository abgerufen werden.

7.1 Programmübersicht und Modi

Nach dem Starten des Programms wird ein Hauptmenü angezeigt, dass die Auswahl zwischen zwei Modi erlaubt: Der **Demonstrationsmodus** erlaubt es, das Fahrzeug über ssh zu steuern, d.h. es mit einer bestimmten Geschwindigkeit vor- oder rückwärts fahren zu lassen. Im **Parcourmodus** fährt das Auto selbstständig einer Linie nach und reagiert auf farbige Punkte auf der Strecke. In beiden Modi kann der aktuelle Ablauf jederzeit unterbrochen werden. Im Parcourmodus erfolgt zudem die Ausgabe der Sensorwerte über ein Webinterface.

7.2 Architektur

7.2.1 Tasks

Wie bereits erwähnt besteht das System nur aus einem `MODULE`, die einzelnen Module sind als `TASK` realisiert. Die Tasks lassen sich in drei Kategorien unterteilen:

- **Gesamtsteuerung** / **Menü**: Der Task zur Gesamtsteuerung enthält das Menü und startet und beendet alle anderen Tasks.
  Dies ist: `main`.
- **Steuerung**: Die steuernden Tasks lesen globale Variablen der Sensorik ein, berechnen das gewünschte Verhalten und schreiben in die globalen Variablen zur Steuerung der Aktorik.
  Dies sind: `parcour, demo, light`.

- **Ausführung**: Die ausführenden Tasks lesen Sensoren aus und schreiben die Ergebnisse in globale Variablen bzw. lesen steuernde globale Variablen aus und steuern die Aktoren.
  Dies sind: `blinker`, `light`, `driveleft`, `driveright`, `readlr`, `readfs`, `webinterface`.

Die Task `light` ist dabei ein Spezialfall, da sie sowohl die `blinker` Task verwaltet als auch die Scheinwerfer direkt ansteuert.

Im Folgenden ist die Funktionalität der einzelnen Tasks nochmals genauer erläutert:

- **`main`**: Die Task `main` ist zuständig für den Gesamtablauf. Sie ruft die Prozeduren `init` und `term` bei Beginn und Ende des Programms auf und stellt während der Laufzeit das Menü dar und behandelt Nutzereingaben.
- **`demo`**: Die Task `demo` implementiert die Funktionalität im Demonstrationsmodus. Unter Nutzung der Prozeduren `straight` und `stop_motors` ermöglicht sie das Fahren nach Eingabeparametern.
- **`parcour`**: Die Task `parcour` implementiert die Funktionalität des Parcourmodus. Unter Nutzung der Prozeduren `straight` und `stopmotors` ermöglicht sie das Fahren oder Halten nach erkannten Farben.
- **`light`**: Die Task `light` steuert die Scheinwerfer des Fahrzeugs und setzt Signale für die Blinker.
- **`blinker`**: Die Task `blinker` steuert die Blinker.
- **`driveleft`**: Die Task `driveleft` steuert mittels der Prozedur `step` den linken Schrittmotor an. Die Parameter für den Motor werden aus einer Geschwindigkeitsvariablen errechnet.
- **`driveright`**: Die Task `driveright` steuert mittels der Prozedur `step` den rechten Schrittmotor an. Die Parameter für den Motor werden aus einer Geschwindigkeitsvariablen errechnet.
- **`readlr`**: Die Task `readlr` liest das Signal des Lichtrechens aus und gibt es weiter.
- **`readlr`**: Die Task `readfs` liest das Signal des Farbsensors aus und gibt es weiter.
- **`webinterface`**: Die Task `webinterface` steuert das Webinterface.

7.2.2 Kommunikation

Die Kommunikation der Tasks untereinander erfolgt über globale Variablen. Der Zugriff erfolgt mittels `BOLT`, einem Konstrukt, dass das Schreiber-Leser-Problem effizient löst. So können mehrere lesende Zugriffe oder ein schreibender Zugriff gleichzeitig erfolgen.

7.2.3 Synchronisation / Ablaufsteuerung

Die Synchronisation bzw. Ablaufsteuerung der Tasks wird mittels Semaphoren umgesetzt. Für die Synchronisation von steuernden und ausführenden Tasks wurde das Erzeuger-Verbraucher-Muster mit Semaphoren umgesetzt, sodass z.B. neue Sensorwerte erst nach der Verarbeitung der letzten eingelesen werden. Dieses Muster wird ebenfalls eingesetzt um die Motoren zu synchronisieren, also das Beenden des Fahrvorgangs beider Motoren abzuwarten.
Bei der Umschaltung vom Demo- in den Parcourmodus wird die jeweilige steuernde Task terminiert. Dazu wird der Task über eine globale Variable mitgeteilt, dass sie sich beenden soll. Daraufhin wird der aktuelle Zyklus beendet, die Task gibt eine Semaphore frei und geht in den Zustand `SUSPEND`, sodass sie sicher terminiert werden kann. Dadurch ist sowohl die Konsistenz als auch der Zustand der Task gesichert, und sie kann problemlos neu gestartet werden.

8 Mechanischer Aufbau

8.1 Rahmenbedingungen

Aufgrund des Vorhandenseins eines 3D-Druckers an der Hochschule und der damit verbundenen Flexibilität, haben wir uns entschlossen, die Bodenplatte und die Halterungen für die einzelnen Komponenten zu drucken. Allerdings müssen auch hierbei einige Dinge beachtet werden. Dadurch, dass der Drucker die Modelle schichtenweise von unten nach oben aufbaut, können beispielsweise Löcher in Wänden nur umständlich oder überhaupt nicht realisiert werden. Theoretisch kann dies durch ein Kippen des zu druckenden Körpers oder das Mitdrucken von Hilfselementen umgangen werden. In der Praxis und vor allem dann, wenn solche Probleme auf mehreren Seiten beziehungsweise an sehr filigranen Stellen eines Modells auftreten, nutzen auch diese Methoden nichts mehr.
Zudem kann das Innenleben der gedruckten Teile entweder durch eine Art Wabenstruktur oder komplett gefüllt aufgebaut sein. Durch den Aufbau in Form einer Wabenstruktur wird zum einen ein geringerer Verbrauch von Druckmaterialen und zum anderen eine wesentlich geringere Druckzeit erzielt. Vor allem der Zeitfaktor ist dabei nicht zu unterschätzen, da selbst der Druck von einfachen und mittelgroßen Teilen in der Regel mehrere Stunden in Anspruch nimmt und ein gefüllter Druck die benötigte Zeit nicht selten verdoppelt. Wenn man jedoch beabsichtigt die Komponenten zu verschrauben, oder ein sehr hohe Stabilität vorsieht, sollte man auf diese Druckart verzichten und einen gefüllten Druck vorziehen.
Um in dem Fall, dass ein Teil unseres Modells beschädigt wird, nicht alles erneut drucken zu müssen, ist das Gesamtmodell modular aufgebaut. Jedes Modul weißt entweder den weiblichen oder männlichen Teil einer Steckverbindung auf.
Außerdem ist der Druckbereich eines 3D-Druckers beschränkt. Bei dem von uns genutzten 3D-Drucker sind wir vor allem auf einer der horizonzalen Achsen eingeschränkt. Der hier zur Verfügung stehende Druckbereich von maximal 21,4cm hat somit die Länge unseres Modells bestimmt. Bis auf wenige Ausnahmen werden von uns alle Bauteile mit einer Wabenstruktur im Inneren gedruckt.

8.2 Motorenführung

Da wir uns dazu entschlossen haben, jeweils beide Räder auf einer Seite über einen O-Ring miteinander zu verbinden, muss es eine Möglichkeit geben, diese ein- und nachspannen zu können. Aus diesem Grund haben wir eine Art Schiene modelliert, in der ein Motor über jeweils eine Schraube mit zwei Muttern an der Vorder- sowie Hinterseite befestigt werden kann. Eine Motorenführung weißt im Inneren eine effektive Länge von 68mm auf und erlaubt es uns somit je nach Größe der verwendeten Muttern die Motoren mit einer Länge von 40mm um ca. 15mm zu verstellen. Auf der hier gezeigten Skizze ist ein Schraubendurchmesser von 5mm vorgesehen. Dieser kann jedoch beliebig angepasst werden, wenn man die entsprechenden Maße der Motorenführung abändert. Durch eine Wandstärke von 2mm wird eine ausreichende Stabilität erreicht. Über den weiblichen Teil einer Steckverbindung am unteren Ende werden die Motorenführungen seitlich mit der weiter unten beschriebenen Bodenplatte verbunden. (Siehe Abbildung 1)

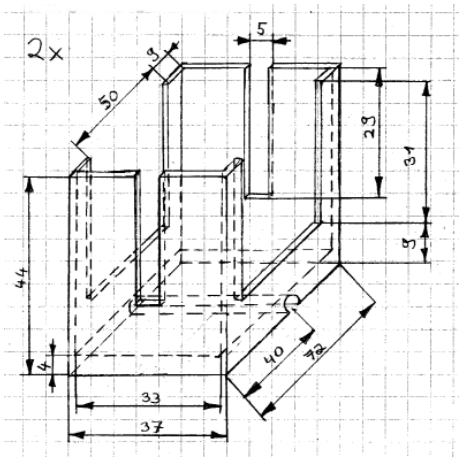

Abbildung 1: Motorenführung

8.3 Motorenfassung

Die beiden Motoren werden jeweils in einer Halterung eingefasst, welche später in den Motorenführungen beweglich gelagert sein werden, um die O-Ringe ein- und nachspannen zu können. Der Primärgrund für die Verwendung solcher Motorenfassungen ist jedoch der, dass die zu den Motoren gehörenden Treiberstufen über ein weiteres Bauteil an den Oberseiten der Motorenfassungen befestigt werden können. Auf diese Weise kann der für die Treiberstufen benötigte Platz auf der Bodenplatte eingespart werden. Zudem ist die Länge der Kabel zwischen Motor und Treiberstufe somit konstant. Um die Kabel, die seitlich aus den Motoren kommen zu berücksichtigen,

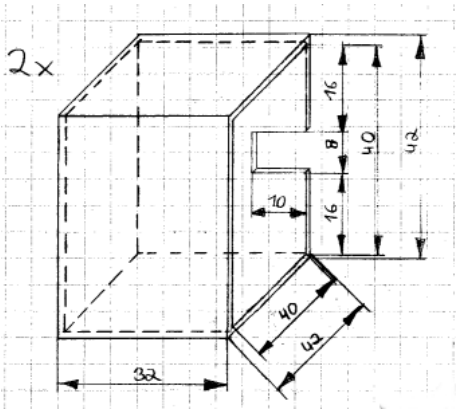

Abbildung 2: Motorenfassung

befindet sich an einer Seite der Motorenfassungen eine entsprechende Aussparung. (Siehe Abbildung 2)

8.4 Treiberstufenhalteung

Die Treiberstufenhalterung ist im Grunde lediglich eine quadratische Grundplatte mit einer Erhöhungen an jeder Ecke. Diese Erhöhungen dienen dazu, die Treiberstufen mit jeweils vier Schrauben an den Treiberstufenhalterungen befestigen zu können. Durch den Einsatz von Schrauben kommt hier ein Druck mit Wabenstruktur nicht in Frage. Deshalb werden die Treiberstufenhalterungen gefüllt gedruckt. Sie werden mit den Oberseiten der Motorenfassungen verklebt. (Siehe Abbildung 3)

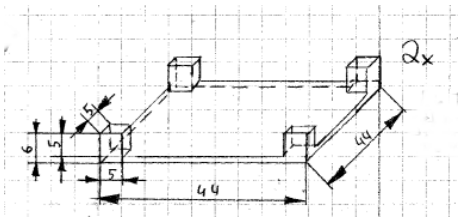

Abbildung 3: Treiberstufenhalterung

8.5 Vorderachsenhalterung

Um die vorderen Räder an unserem Modell befestigen zu können, ist eine entsprechende Halterung notwendig. Die beiden Vorderräder sind auf voneinander getrennten Achsen befestigt. Diese wiederum sind in jeweils einem Fischertechnik-Baustein frei drehbar gelagert. Um ein seitliches herausrutschen der Achsen zu verhindern werden an den Achsen befestigte Fischertechnik-Klemmen verwendet. Damit die Vorderräder auf dem selben Niveau wie die durch die relativ großen Schrittmotoren sehr hoch gelegenen Hinterräder sind, sind die Vorderachsenhalterungen 34mm hoch. Am oberen Ende befindet sich eine für die bereits zuvor genannten Fischertechnik-Bausteine vorgesehene Öffnung. Um eine ausreichende Stabilität gewährleisten zu können, ist auch hier eine Wandstärke von 2mm vorgesehen. Am unteren Ende befindet sich der weibliche Teil

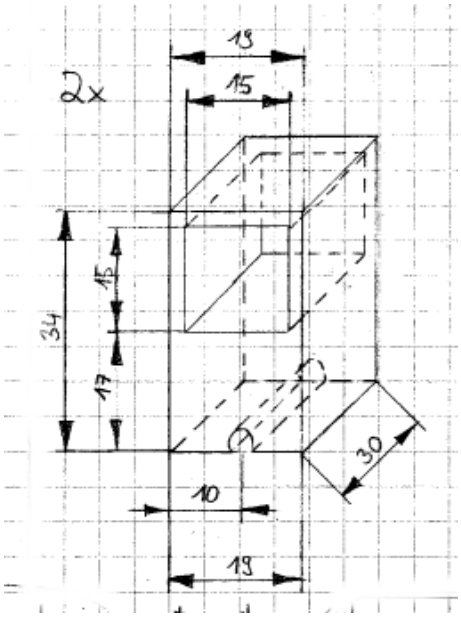

Abbildung 4: Vorderachsenhalterung

einer Steckverbindung, um die Vorderachsenhalterungen mit den auf der Bodenplatte befindlichen Gegenstücken zu verbinden. (Siehe Abbildung 4)

8.6 Bodenplatte

Um ein funktionierendes Gesamtsystem zu erhalten, ist eine Bodenplatte von nöten. Auf ihr werden schließlich alle anderen Komponenten befestigt. Um keine Stabilität zu verlieren, wird sie aus einem einzigen Teil gedruckt. Die daraus resultierenden Nachteile, wie zum Beispiel die eingeschränkten Möglichkeiten durch den begrenzten Druckbereich des 3D-Druckers, mussten wir daher in Kauf nehmen.
Die Bodenplatte hat unter anderem aus Stabilitätsgründen eine Dicke von 5mm. Außerdem werden die anderen Bauteile über Steckverbindungen in abgesenkten Bereichen der Bodenplatte seitlich eingeschoben. Eine Breite von 105mm ergibt sich aus der breitesten Komponente, die auf der Bodenplatte befestigt werden muss, dem Akku. Die Länge von 214mm wurde durch den maximalen Druckbereich des von uns verwendeten 3D-Druckers bestimmt.
Jeweils vorne (auf der zugehörogen Skizze: unten) und hinten auf der Bodenplatte werden die Platinen mit den LEDs befestigt. Direkt hinter der vorderen LED-Platine finden sich jeweils links und rechts die Stellen, an denen die Vorderachsenhalterungen eingeschoben werden können. Dazwischen ist ein Durchbruch um die Kabel der vorderen LEDs nach unten durchführen zu können.
Im Bereich dahinter findet der für die Stromversorgung benötigte Akku seinen Platz. Durch die vier Durchbrücke können zwei Kabelbinder oder ähnliches gezogen werden, um den Akku an der Bodenplatte zu befestigen. Dahinter befindet sich eine oben geöffnete Box mit einem langen Durchbruch im inneren. Dort können die Kabel der Sensoren, welche an der Unterseite der Bodenplatte befestigt werden, und die Kabel der vorderen LEDs wieder nach oben geführt werden. So können sie einfach an dem direkt dahinter befestigten Raspberry Pi angeschlossen werden. Zu lange Kabel können dann in dieser Box verstaut werden.
Der RaspberryPi wird später an separat gedruckten Stelzen festgeschraubt, welche mit der Bodenplatte verklebt werden. Sie werden separat gedruckt, da auch diese Teile aufgrund der Tatsache, dass hier Schrauben zum Einsatz kommen, im Gegensatz zur Bodenplatte gefüllt gedruckt.
Die beiden darauffolgenden seitlichen Absenkungen mit den männlichen Teilen der Steckverbindungen dienen zur Befestigung der beiden Motorenführungen. Dazwischen ist wieder eine nach oben geöffnete Box, die zur Verkabelung

dient. Diese ist zusätzlich teilweise nach vorne und komplett nach hinten geöffnet. So können auch bei der Verwendung eines Deckels Kabel nach vorne und hinten durchgeführt werden. Auch zum Anbringen der LED-Platine am hinteren Ende der Bodenplatte ist die Öffnung an der Hinterseite der Box aus Platzmangel notwendig. Am hintersten Bereich der Bodenplatte wird wie bereits oben beschrieben die Platine mit den Rücklichtern befestigt. (Siehe Abbildung 5)

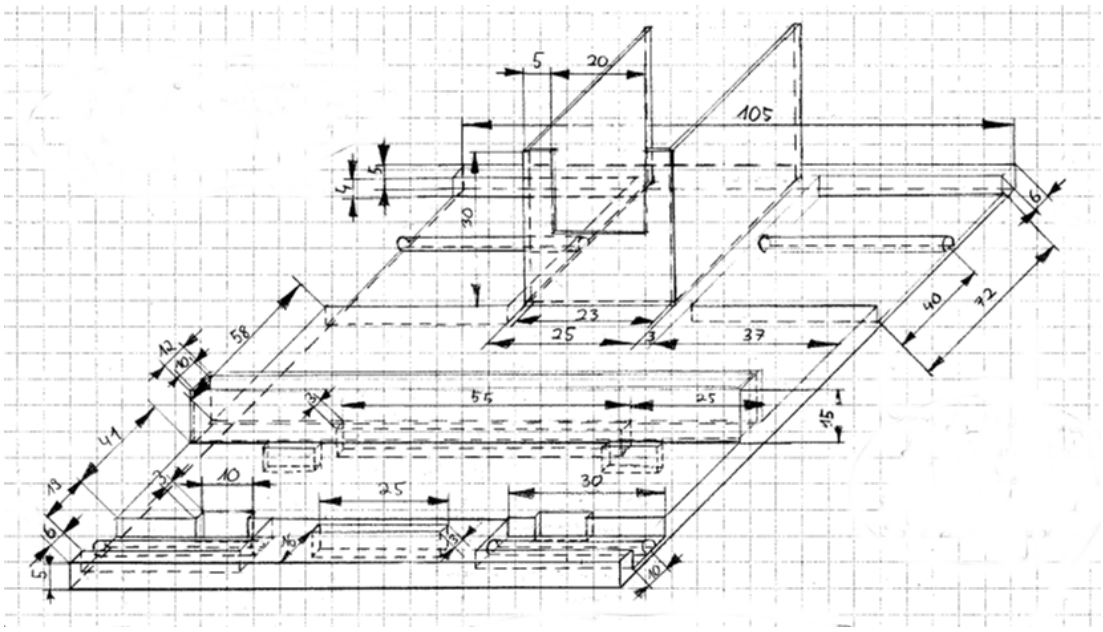

Abbildung 5: Bodenplatte

8.7 Konzeptionelle Änderungen im Projektverlauf

Während dem Verlauf unseres Projektes mussten wir aus unterschiedlichen Gründen in manchen Aspekten von unserem Grundkonzept für den mechanischen Aufbau abweichen.
Zum einen wurde auf die zuvor geplante Modularität der gedruckten Bauteile verzichtet. Das System mit den Steckverbindungen hat grundsätzlich funktioniert. Da es beim Druck jedoch zu Ungenauigkeiten kam, und es zeitlich keinen Sinn gemacht hätte, länger auf ein für weitere Entwicklungsschritte brauchbares Modell zu warten, haben wir die einzelnen Bauteile verklebt.
Eine weitere, größere Änderung an unserem Konzept ist, dass die Räder der beiden Fahrzeugseiten nicht miteinander verbunden sind. Diese Entscheidung wurde getroffen, da die zum verbinden der Räder genutzten O-Ringe nach sehr kurzer Fahrzeit von den Räder gerutscht sind. Das lag daran, dass die Räder nicht perfekt auf den Elektromotoren montiert werden konnten und daher geringfügige Unwucht hatten. Die Führung auf den Rädern reichte daher nicht aus, um die O-Ringe zu sichern.

Um die Kabel für die Stromversorgung für den Raspberry Pi anschließen zu können musste zumindest ein Teil des vorderen Verkabelungskastens entfernt werden. Hier würde es für einen neuen Druck Sinn machen, ihn auf der rechten Seite um wenige Zentimeter zu kürzen.
Bis auf diese Abweichungen, konnten wir alle Teile wie geplant einsetzen.

9 Compilerfehler

Hauptziel des Projekts ist das Testen von OpenPEARL in Anwendung. Tabelle 4 gibt eine Übersicht über die gefundenen Fehler im OpenPEARL Compiler.

Fehler	**Beschreibung**	**Kontext**
Belegung Semaphoren	Die atomare Belegung mehrere Semaphoren ist nicht möglich	Philosophen-problem
INIT BIT(4)	Führende 0 wird bei init von BIT-Arrays nicht berücksichtigt	Schrittmotor
TRY	TRY Anweisung gibt falschen Datentyp zurück	Gesamtsystem / Synchronisa-tion
RET	Return ohne Argumente ist nicht möglich	Gesamtsystem / Menü

Tabelle 4: Gefundene Fehler im OpenPEARL Compiler

10 Fazit

Hauptziel des Projektes war der Einsatz von OpenPEARL in einem größeren System mit meheren Sensoren und Aktoren, um den Zustand von OpenPEARL und die Eignung der Sensoren festzustellen.
Die Aussage, dass OpenPEARL weit genug entwickelt ist, um es in ersten Projekten einzusetzen, konnten wir bestätigen, tatsächlich ist es uns gelungen ein Modellauto zu entwickeln, das in den meisten Fällen autonom einer Linie folgen konnte. Auch haben wir einige Fehler im OpenPEARL Compiler gefunden und konnten so zur Weiterentwicklung bzw. Verbesserung des Compilers beitragen.
Bezüglich der Sensorik mussten wir in einigen Fällen feststellen, dass der Einsatz in einem Echtzeitsystem oftmals schwieriger ist als gedacht, zumal die Qualität der Daten nicht immer optimal ist. Grundsätzlich war OpenPEARL jedoch gut geeignet, um viele verschiedenartige Geräte anzusteuern und in einem System einheitlich zu verwenden.
Das entstandene SensorCar war bei der Projektpräsentation im Allgemeinen funktionsfähig, aufgrund fehlender Tests gab es allerdings auch noch einige Probleme.
Da das Projektziel, OpenPEARL einzusetzen und zu testen klar erreicht wurde, können wir von einem Erfolg des Projektes sprechen.

11 Anhang

11.1 Belegungsplan Raspberry Pi 3