

HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG  
UNIVERSITY OF APPLIED SCIENCES

# Realzeitsysteme

Realzeitnachweis

**Manuel Vögele, Simon Wörner und Siegfried  
Kienzle**

Konstanz, den 28. Juni 2017

# Inhaltsverzeichnis

<b>1</b>	<b>Entwurf Hardware</b>	<b>3</b>
1.1	Verwendete Sensoren . . . . .	3
1.2	Probleme . . . . .	3
<b>2</b>	<b>Zeitmessungen</b>	<b>5</b>
<b>3</b>	<b>Entwurf Software</b>	<b>8</b>
3.1	Design . . . . .	9
3.2	System . . . . .	10
3.2.1	OS . . . . .	10
3.2.2	Remote Login . . . . .	10
3.2.3	Netzwerkanbindung . . . . .	11
3.2.4	Probleme . . . . .	11
3.3	Programmiersprachen . . . . .	11
3.3.1	Rust/C++ Anbindung . . . . .	12
3.4	Threads . . . . .	12
3.4.1	Main-Thread . . . . .	12
3.4.2	PWM-Signal der Motoren . . . . .	12
3.4.3	Ultraschall-Sensor . . . . .	13
3.4.4	RFID . . . . .	13
3.5	Notaus-Button . . . . .	13
3.6	RFID-Sensor-Library . . . . .	13
<b>4</b>	<b>Realzeitnachweis</b>	<b>14</b>
4.1	Realzeitanforderungen . . . . .	14
4.1.1	Physikalische Eigenschaften . . . . .	14
4.1.2	Ultraschall . . . . .	14
4.1.3	Infrarot . . . . .	14
4.1.4	RFID . . . . .	15
4.1.5	Motor . . . . .	15
4.2	Reaktionszeit . . . . .	15
4.2.1	Main / Infrarot . . . . .	15
4.2.2	RFID . . . . .	15

4.2.3	Engine . . . . .	16
4.2.4	Ultrasonic Echo . . . . .	16
4.2.5	Ultrasonic Trigger . . . . .	16
4.2.6	Emergency Stop . . . . .	16
4.3	Blokierzeiten . . . . .	16
4.3.1	Atomic Integer / Bool . . . . .	16
4.4	Nachweis . . . . .	17
4.4.1	Prioritäten . . . . .	17
4.4.2	Temporalen Parameter . . . . .	17
4.4.3	Nachweis ohne Berücksichtigung der Ressourcen . . . . .	17
4.4.4	Nachweis mit Berücksichtigung der Ressourcen . . . . .	18

# Kapitel 1

## Entwurf Hardware

### 1.1 Verwendete Sensoren

In diesem Projekt werden alle vorgegebenen Sensoren wie Infrarot, RFID-Reader und Ultraschall-Sensoren verwendet.

### 1.2 Probleme

#### Ultraschall

Bei der Verwendung eines einzelnen Ultraschallsensors haben wir Probleme bei der Erkennung von Wänden festgestellt, wenn der Roboter nicht orthogonal auf die Wand zufährt. Das lässt sich durch den geringen Winkel der Ultraschallmessung und die hohe gemessene Entfernung, wenn fast parallel auf die Wand zugefahren wird, erklären.

Deshalb kommt bei unserem Projekt anstatt einem, ein zweiter Ultraschall-Sensor zum Einsatz. Diese sollen im Zusammenspiel, die Detektion der Wände merklich verbessern. Es wurde nicht, wie ursprünglich vorgesehen ein Sensor in der Mitte, sondern der eine Sensor am rechten und der andere am linken Rand befestigt. Diese Konstruktion deckt nun einen großen Bereich ab, allerdings entsteht in der Mitte dennoch ein minimaler toter Winkel, der nicht detektiert werden kann.

#### Hinterrad

Die Ausrichtung des Hinterrades verändert die Fahrtrichtung merklich, deshalb ist das Fahrverhalten bei weitem nicht deterministisch. Um das Fahrverhalten zu verbessern wurde das Hinterrad mittels eines Gummibandes fixiert.

## **Notaus**

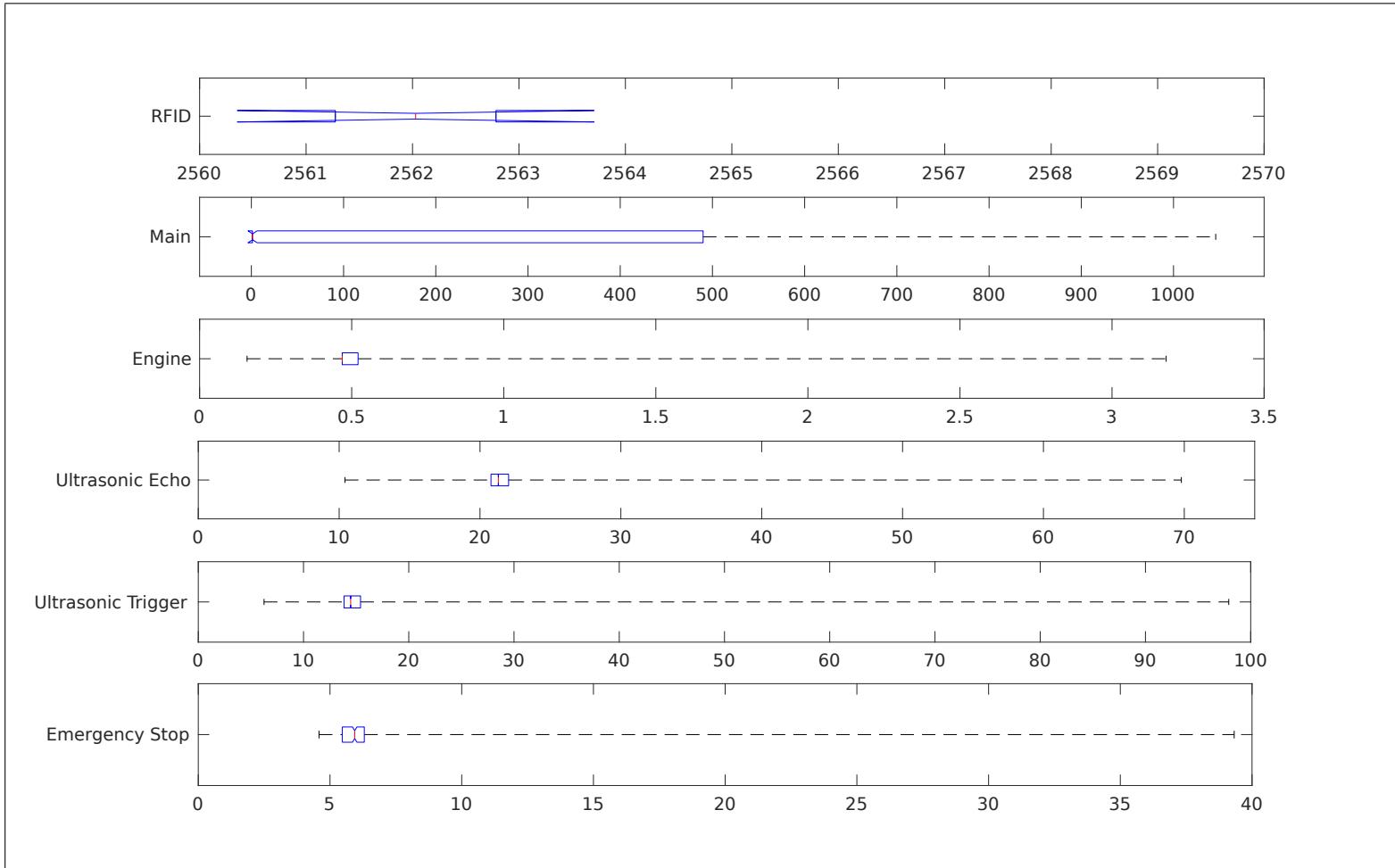
Der Notaus-Knopf führte häufig zu false positives aufgrund von Vibration während der Fahrt und statischer Aufladung der Stange, auf welcher der Knopf befestigt wurde.

Dieses Problem konnte gelöst werden durch kleben des Notaus an die Stange und Isolation der Kontakte mittels Klebeband.

# Kapitel 2

## Zeitmessungen

9

Abbildung 2.1: **ohne Last**

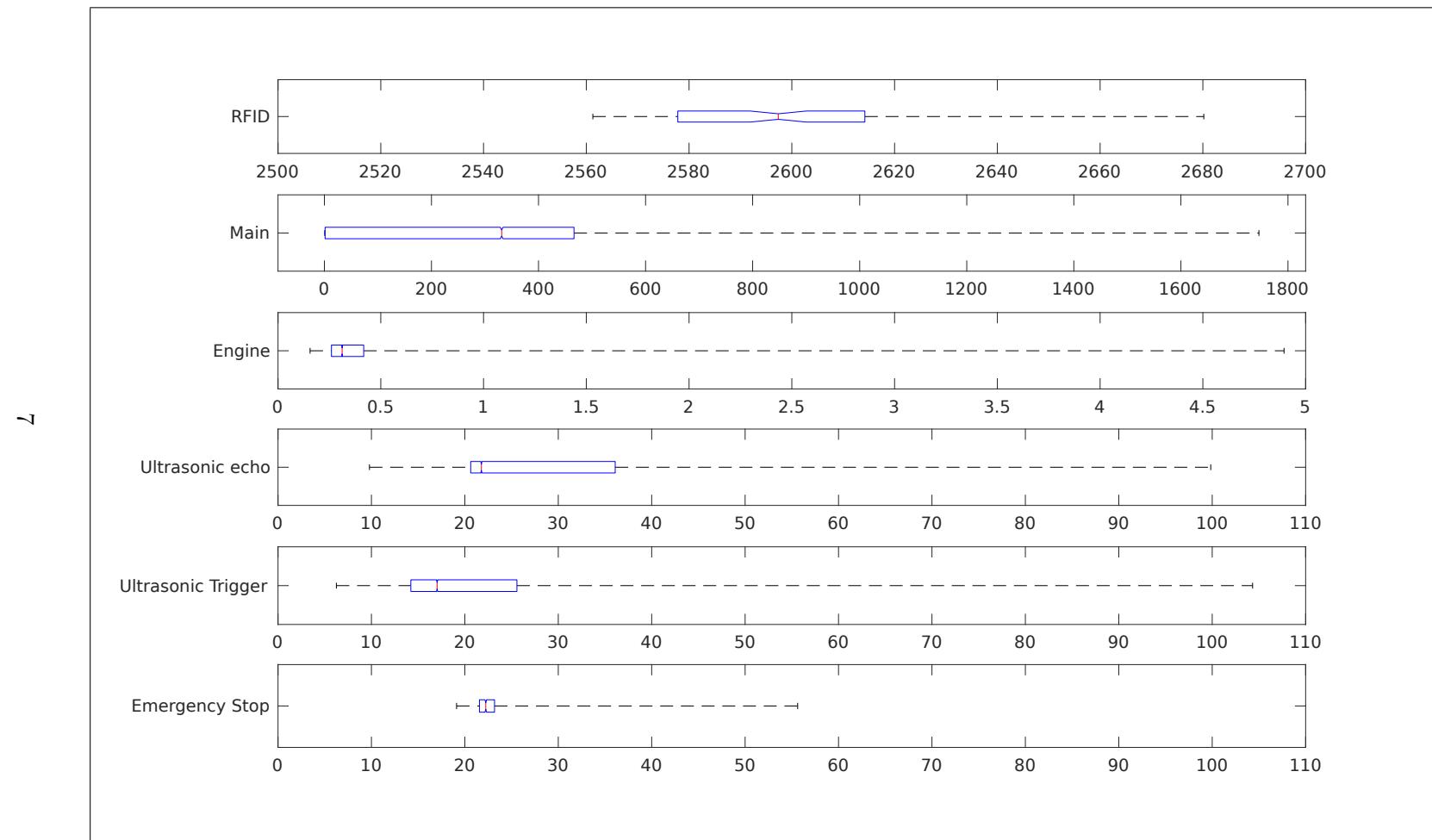


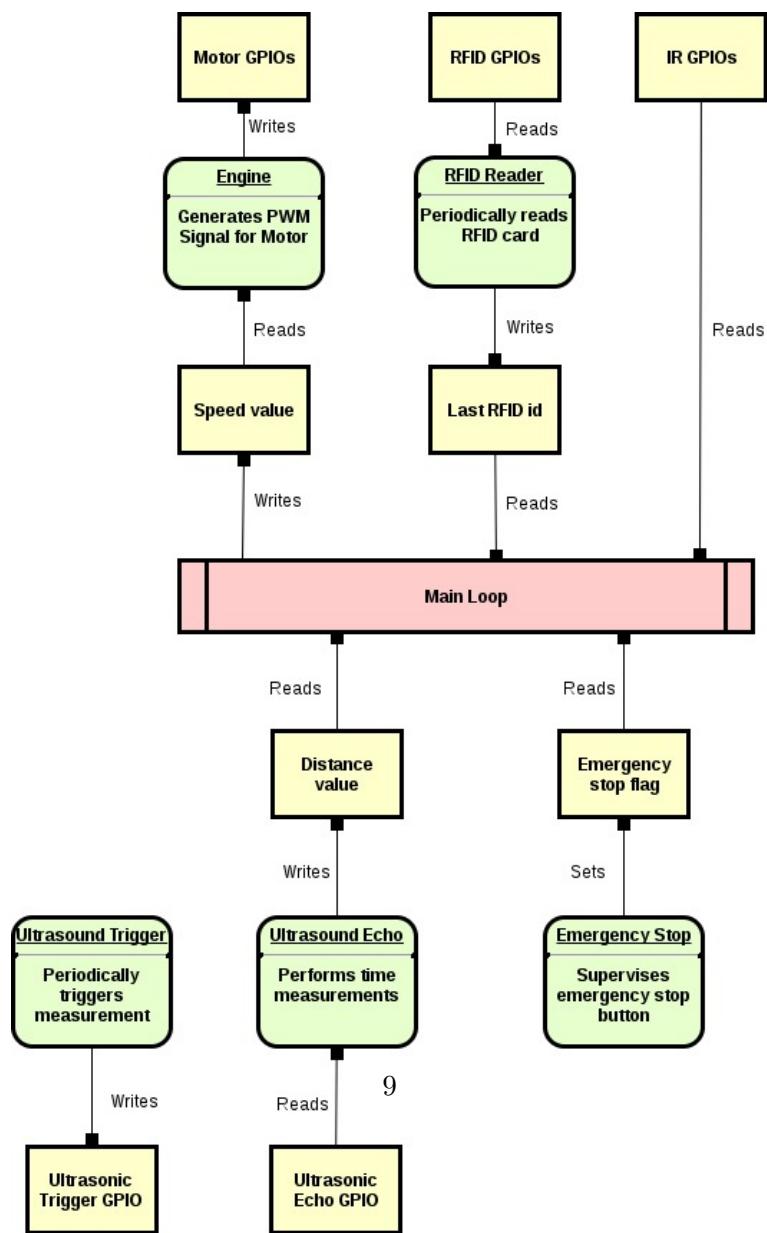
Abbildung 2.2: **mit Last**



# Kapitel 3

## Entwurf Software

### 3.1 Design



Die Software besteht aus einem Main-Loop, der das Herzstück der Anwendung bildet, sowie separaten Threads für das Auslesen der Sensoren, sowie die Steuerung der Motoren. Alle Komponenten der Software laufen im gleichen Prozess. Die Anwendung ist komplett im Userspace realisiert.

Im Main-Loop laufen die Daten aller Sensoren zusammen. Diese werden ausgewertet und eine adäquate Reaktion anhand aller relevanten Sensordaten errechnet. Anschließend wird das entsprechende Kommando an die Steuerthreads der Motoren weitergegeben.

Alle Sensoren — mit Ausnahme der Infrarot-Sensoren — verfügen über eigene Threads, die für das Auslesen und Aufbereiten der jeweiligen Sensordaten verantwortlich sind. Bei den Infrarot-Sensoren wurde auf diesen Thread verzichtet, da die Sensoren sehr schnell ausgelesen werden können und die Sensordaten keiner komplizierten Aufbereitung bedürfen. Daher ist der durch einen Thread entstehende Overhead hier nicht zu rechtfertigen. Der Zugriff auf die Sensoren erfolgt daher bei diesem Sensor auf einen direkten GPIO-Zugriff des Main-Loops.

Die Kommunikation zwischen den Threads erfolgt über einen geteilten Speicherbereich. In diesem wird nach einer erfolgreichen Messung der gemessene Wert abgelegt, sodass der Main-Loop bei Bedarf nur noch darauf zugreifen muss. Im Falle des Ultraschall-Sensors ist dieser Wert die Entfernung in Millimeter, im Falle des RFID-Readers die gelesene ID. Zur Steuerung der Motoren legt der Main-Loop einen Geschwindigkeitswert zwischen +100 und -100 in einem vorzeichenbehafteten Integerwert ab.

Im Kapitel 3.4 wird weiter auf die einzelnen Threads eingegangen.

## 3.2 System

### 3.2.1 OS

Das Projekt nutzt die Arch Linux ARM Distribution. Diese ist wiederum ein Fork von Arch Linux, welches nur für Desktop-Prozessoren ausgelegt ist.

Bei Arch Linux handelt es sich um eine Linux Distribution, die sich an Fortgeschrittene Linux Nutzer richtet und diesen das System selbst zusammen stellen lässt.

#### Anpassungen am Kernel

Da Arch Linux einen nicht realzeitfähigen Kernel nutzt, wurde dieser durch einen Kernelin der Version 4.9.30 mit Realtime-Patch ausgetauscht.

### 3.2.2 Remote Login

Für den Remote Login wurde *openssh* installiert und der *ssh*-Daemon aktiviert.

### 3.2.3 Netzwerkanbindung

#### Bluetooth

Um eine Drahtlose Verbindung unabhängig von der Verfügbarkeit eines WiFi AccessPoints zu gewährleisten haben wir ein Peer-to-Peer-Netzwerk über Bluetooth eingerichtet.

Hierfür wurde auf dem Raspberry Pi die Treiber für das Bluetooth Modul installiert und dieses aktiviert. Es wurden anschließend alle möglichen Clients einmalig verbunden und als vertrauenswürdig eingestuft.

Um Bluetooth für ein Netzwerk nutzen zu können muss ein *personal area network* (PAN) eingerichtet werden. Dies lässt sich automatisch nach dem Verbindungsauftakt durch *bt-pan*<sup>1</sup> erledigen.

Zusätzlich wurde ein *DHCP*-Server installiert und aktiviert um keine manuelle IP Konfiguration vornehmen zu müssen.

#### WLAN

Da es teilweise Verbindungs- und Durchsatzprobleme mit Bluetooth gab haben wir zusätzlich die WLAN-Verbindung mit dem Labor Router eingerichtet.

#### Ethernet

Da Bluetooth und auch WLAN teilweise Probleme hatte haben wir eine statische IP am Ethernet Port konfiguriert, um auch bei Problemen mit diesen den Raspberry Pi noch warten zu können.

### 3.2.4 Probleme

#### Zeit / NTP

Nach dem Starten des Raspberry Pi ist das Datum des Systems zurückgesetzt. Dies führt u. a. zu SSL-Fehlern da teilweise Zertifikate noch nicht gültig sind.

Erster Lösungsansatz hierfür war das Installieren und aktivieren des *NTP*-Daemons. Damit wurde das Problem zwar gelöst, allerdings nur nach etwas längerer Zeit da die Synchronisation nicht sofort stattfindet.

Wir haben zusätzlich ein Skript erstellt, was die Synchronisation manuell anstößt, falls wir sofort ein korrektes Datum benötigen.

## 3.3 Programmiersprachen

Bei unserer Software kommt sowohl die Programmiersprache C++ als auch Rust zum Einsatz. Die Wahl fiel auf diese beiden, da sie keine Garbage-

---

<sup>1</sup><https://github.com/mk-fg/fgtk/blob/master/bt-pan>

Collection besitzen, hardwarenah einsetzbar sind und sie leichter verwendbar sind als C.

### 3.3.1 Rust/C++ Anbindung

Unser Hauptroutine wurde in C++ entwickelt, deshalb musste eine Anbindung des Rust Codes nach C++ erstellt werden. Hierfür wurden in Rust C-Funktionen exportiert welche in C++ in Wrapper-Klassen gepackt wurden um die Vorteile der Objektorientierung beizubehalten zu können.

#### Probleme

Bei der Anbindung von Rust über C Funktionen stellte das Nutzen von komplexen Typen ein Problem dar, da die Rust Dokumentation darauf nur ungenügend eingeht. Nach längerer Recherche konnten wir einen Artikel der sich mit diesem Thema beschäftigt finden: *Complex types with Rust's FFI*<sup>2</sup>

## 3.4 Threads

Sämtliche Ansteuerungen für die Sensoren, bis auf die der Infrarot-Sensoren, sind im Projekt über Threads realisiert. Der Datenaustausch zwischen dem Main-Thread und den Ansteuerungs-Threads erfolgt über shared memory. Der Zugriff auf den gemeinsamen Speicherbereich wird mittels atomic-Variablen synchronisiert. Im Folgenden soll nun auf die Threads eingegangen werden.

### 3.4.1 Main-Thread

Main steht an vierter Position der Prioritäten und übernimmt neben der Auswertung der Sensordaten die Ansteuerung der Infrarot-Sensoren. Wenn an diesen eine Eins anliegt wurde ein schwarzer Bereich detektiert, ansonsten liefern diese null zurück.

### 3.4.2 PWM-Signal der Motoren

Für die beiden Motoren existiert jeweils ein Thread. Diese beiden Threads sind in der Priorität an zweiter Stelle. Das PWM-Signal schaltet in einem bestimmten Intervall die Motoren an und aus, um die Geschwindigkeit des Motors zu regulieren.

---

<sup>2</sup><https://medium.com/jim-fleming/complex-types-with-rust-s-ffi-315d14619479>

### **3.4.3 Ultraschall-Sensor**

Bei den beiden Ultraschall-Sensoren kommen für Echo und Trigger jeweils zwei Threads zum Einsatz. Der Trigger-Thread zieht alle  $20ms$  den Trigger-Output für  $20\mu s$  auf 1. Der Echo-Thread wartet entsprechend auf einen Interrupt und beginnt bei steigender Flanke mit der Zeitmessung. Kommt nun eine fallende Flanke, stoppt dieser Thread die Zeitmessung und berechnet dem entsprechend den Abstand. Die maximale Frequenz für den Trigger von  $20ms$  und die minimale Triggerlänge von  $20\mu s$  kommen bedingt durch die Hardware zustande. Der Ultraschall-Echo-Thread ist in der Prioritätenliste an dritter Stelle, der Ultraschall-Trigger-Thread an fünfter.

### **3.4.4 RFID**

Das Lesen beim RFID-Reader erfolgt periodisch und über einen eigenen Thread. Dabei wird in einem Intervall von  $50ms$  abgefragt, ob ein RFID-Chip vorhanden ist. Wenn eine Karte mit RFID-Chip vorhanden ist, wird die entsprechende UID gespeichert.

## **3.5 Notaus-Button**

Der Notaus-Button wird mittels atomic-Variable gesteuert. Ist diese durch drücken des Buttons gesetzt, führt der Main-Thread die entsprechenden Optionen durch und hält den Roboter an.

## **3.6 RFID-Sensor-Library**

Für die Ansteuerung des RFID-Sensors wurde C++-Quellcode vom Github-Respository <https://github.com/paguz/RPi-RFID> in eine eigene C++-Klasse eingebettet. Das Repository verwendet die bcm2835-Library.

# Kapitel 4

## Realzeitnachweis

### 4.1 Realzeitanforderungen

#### 4.1.1 Physikalische Eigenschaften

Um die Anforderungen zu bestimmen haben wir die physikalischen Eigenschaften unseres Roboters bestimmt.

Hierbei ist die Höchstgeschwindigkeit von Interesse: Laut Messungen fährt der Roboter  $2,01m$  in 5 Sekunden.

$$v = 2,01m/5s = 0,402m/s$$

Daraus ergibt sich eine Höchstgeschwindigkeit von  $0,402m/s$ .

#### 4.1.2 Ultraschall

Bei der Ultraschall Messung wird ab  $18cm$  gedreht, der Mindestabstand für das Wenden beträgt  $10cm$ :

$$t_{D_{max}-Ultraschall} = \frac{18cm - 10cm}{0,402m/s} = 199ms$$

Folglich liegt die Reaktionszeit für das Erkennen und Anhalten mit Ultraschall bei  $199ms$ . Hardwarebedingt lassen sich Entfernungsmessungen nur alle  $20ms$  durchführen:

$$t_{P_{min}-Ultraschall} = 20ms$$

#### 4.1.3 Infrarot

Die Breite der Linien beträgt  $1,5cm$ :

$$t_{D_{max}-Infrarot} = \frac{1,5cm}{0,402m/s} = 37,3ms$$

Daraus folgt eine maximale Reaktionszeit von  $37,3ms$  für das Erkennen der Linien. Aufgrund von physikalischen und hardwareseitigen Einschränkungen kann das Anhalten häufig nicht innerhalb Linie erfolgen. Die Gründe liegen bei der Massenträgheit, der schlechten Motoren und unzureichender Bodenhaftung.

Deshalb haben wir uns dazu entschieden ein Überfahren der Linie in Kauf zu nehmen, sofern diese zuverlässig erkannt wird und wir zurück fahren.

#### 4.1.4 RFID

Die minimale Länge eines RFID-Chips beträgt  $2,5cm$  und Länger der Messfläche beträgt ca.  $4cm$ . Folglich ist eine Messung möglich, während sich die  $4cm$ -lange Messfläche über dem Chip befindet.

$$t_{D_{max}-RFID} = \frac{4cm}{0,402m/s} = 99,5ms$$

#### 4.1.5 Motor

Für die Motoren haben wir eine Periode von  $500\mu s$  festgelegt.

## 4.2 Reaktionszeit

### 4.2.1 Main / Infrarot

$$\begin{aligned} t_{E_{max}} &= 1,4ms \\ t_{E_{max}} \leq t_{P_{min}} \leq t_{P_{max}} \leq t_{D_{max}} - t_{E_{max}} \\ 1,4ms \leq t_{P_{min}} \leq t_{P_{max}} &\leq 37,3ms - 1,4ms \\ 1,4ms \leq t_{P_{min}} \leq t_{P_{max}} &\leq 32,9ms \end{aligned}$$

Aus der maximalen Reaktionszeit und der maximalen Ausführungszeit ergibt sich eine Periode zwischen  $1,4ms$  und  $32,9ms$ . Wir haben uns für  $t_{P_{min}} = t_{P_{max}} = 10ms$  entschieden.

### 4.2.2 RFID

$$\begin{aligned} t_{E_{max}} &= 2,7ms \\ t_{E_{max}} \leq t_{P_{min}} \leq t_{P_{max}} \leq t_{D_{max}} - t_{P_{max-Main}} - t_{E_{max-RFID}} \\ 2,7ms \leq t_{P_{min}} \leq t_{P_{max}} &\leq 99,5ms - 32,9ms - 2,7ms \\ 2,7ms \leq t_{P_{min}} \leq t_{P_{max}} &\leq 63,9ms \end{aligned}$$

Wir haben uns für  $t_{P_{min}} = t_{P_{max}} = 50ms$  entschieden.

#### 4.2.3 Engine

$$t_{E_{max}} = 5\mu s$$

$$5\mu s \leq t_{P_{min}} \leq t_{P_{max}}$$

$$5\mu s \leq 500\mu s \leq 500\mu s \quad \square$$

#### 4.2.4 Ultrasonic Echo

$$t_{E_{max}} = 200\mu s$$

$$t_{E_{max}} \leq t_{P_{min}} \leq t_{P_{max}} \leq t_{D_{max}} - t_{P_{max-Main}} - t_{P_{max-Engine}} - t_{E_{max-U-Echo}}$$

$$200\mu s \leq t_{P_{min}} \leq t_{P_{max}} \leq 199ms - 32, 9ms - 0, 5ms - 0, 2ms$$

$$200\mu s \leq t_{P_{min}} \leq t_{P_{max}} \leq 165, 4ms$$

$$200\mu s \leq 20ms \leq 20ms \leq 165, 4ms \quad \square$$

#### 4.2.5 Ultrasonic Trigger

$$t_{E_{max}} = 88\mu s \approx 0, 1ms$$

$$t_{E_{max}} \leq t_{P_{min}} \leq t_{P_{max}} \leq t_{D_{max}} - t_{P_{max-Main}} - t_{P_{max-Engine}} - t_{E_{max-U-Trigger}}$$

$$200\mu s \leq t_{P_{min}} \leq t_{P_{max}} \leq 199ms - 32, 9ms - 0, 5ms - 0, 1ms$$

$$200\mu s \leq t_{P_{min}} \leq t_{P_{max}} \leq 165, 5ms$$

$$200\mu s \leq 20ms \leq 20ms \leq 165, 5ms \quad \square$$

#### 4.2.6 Emergency Stop

$$t_{E_{max}} = 1ms$$

### 4.3 Blokierzeiten

#### 4.3.1 Atomic Integer / Bool

$$t_B = 0, 26\mu s$$

## 4.4 Nachweis

### 4.4.1 Prioritäten

1. Emergency Stop
2. 2x Engine
3. 2x Ultrasonic Echo
4. Main
5. 2x Ultrasonic Trigger
6. RFID

### 4.4.2 Temporalen Parameter

- Main:  $(10.000\mu s; 1.400\mu s)$
- RFID:  $(50.000\mu s; 2.700\mu s)$
- 2x Engine:  $(500\mu s; 5\mu s)$
- 2x Ultrasonic Echo:  $(20.000\mu s; 200\mu s)$
- 2x Ultrasonic Trigger:  $(20.000\mu s; 88\mu s)$
- Emergency Stop:  $(-; 50\mu s; 1.000\mu s)$
- Atomic Integer / Bool:  $t_B = 0,26\mu s$

### 4.4.3 Nachweis ohne Berücksichtigung der Ressourcen

#### Hinreichender Schedulingtest

8 Tasks:  $u \approx 70\%$  Auslastungsgrenze

- Main:  $1.800\mu s / 10.000\mu s = 18\%$
- RFID:  $2.700\mu s / 50.000\mu s = 5,4\%$
- 2x Engine:  $5\mu s / 500\mu s = 0,1\%$
- 2x Ultrasonic Echo:  $100\mu s / 20.000\mu s \approx 0,1\%$
- 2x Ultrasonic Trigger:  $110\mu s / 20.000\mu s \approx 0,1\%$

$$14\% + 5,4\% + 6 * 0,1\% = 24\%$$

$$24\% < 70\% \quad \square$$

#### 4.4.4 Nachweis mit Berücksichtigung der Ressourcen Blockierzeiten

- Main:  $t_B = 7 * t_{B_{AtomicInteger/Bool}} = 1,82\mu s$
- RFID:  $t_B = 1 * t_{B_{AtomicInteger/Bool}} = 0,26\mu s$
- 2x Engine:  $t_B = 1 * t_{B_{AtomicInteger/Bool}} = 0,26\mu s$
- 2x Ultrasonic Echo:  $t_B = 1 * t_{B_{AtomicInteger/Bool}} = 0,26\mu s$
- 2x Ultrasonic Trigger:  $t_B = 1 * t_{B_{AtomicInteger/Bool}} = 0,26\mu s$

#### Hinreichender Schedulingtest

- Main:  $1.801,82\mu s / 10.000\mu s \approx 18,02\%$
- RFID:  $2.700,26\mu s / 50.000\mu s \approx 5,4\%$
- 2x Engine:  $5,26\mu s / 500\mu s \approx 0,11\%$
- 2x Ultrasonic Echo:  $100,26\mu s / 20.000\mu s \approx 0,1\%$
- 2x Ultrasonic Trigger:  $110,26\mu s / 20.000\mu s \approx 0,1\%$

$$14,02\% + 5,4\% + 2 * 0,11\% + 4 * 0,1\% = 24,03\%$$

$$24,03\% < 70\% \quad \square$$