

Dokumentation der Projektarbeit

Florian Weber - 44907

11. Juli 2018

Inhaltsverzeichnis

1 Vorwort	3
1.1 Probleme der vorhandenen Carrera Bahn und Zielsetzung zur Projektarbeit	3
2 Funktion - Hardware	5
2.1 Sensoren	7
2.2 Bedienschnittstelle	7
2.3 Arduino	8
2.4 RaspberryPi	8
2.5 Tiefsetzsteller	8
3 Software	9
3.1 RaspberryPi	9
3.2 Arduino	10
3.2.1 Interrupts	10
3.2.1.1 Timer	11
3.2.1.2 Pin Change Interrupt	11
3.2.1.3 Analog Digital Converter Interrupt	12
3.2.2 Regler	12
3.2.3 Analog Digital Wandler	14
3.2.4 PWM Generierung Tiefsetzsteller	14
4 Bedienungsanleitung	16
4.1 Einschalten	16
4.2 Betrieb	16
4.2.1 Manuelles Fahren	17
4.2.2 Automatisiertes Fahren	17
4.2.2.1 Auto-Wait	17
4.2.2.2 Auto-Run	17
4.3 Ausschalten	17
5 Zusammenfassung	18
6 Anhang	19
6.1 Pinbelegung	19

Abbildungsverzeichnis

2.1	Überblick über die Carrera-Bahn	6
2.2	Signalfluss	6
2.3	Bedienschnittstelle	7
3.1	Möglichkeiten zum Programmablauf	10
3.2	Kaskadenreglung Bahngeschwindigkeit	13
3.3	Zustandsautomat Analog Digital Converter Kanal	15
4.1	Zustandsautomat Betriebsmodi	16

Kapitel 1

Vorwort

Zu Beginn des Projekts war die Carrera Bahn auf dem Stand, dass die Fahrzeuge konventionell sowie automatisiert betrieben werden konnten. Außerdem war die Rundenzeitmessung und Visualisierung bereits realisiert. Bei der Energieversorgung konnte zwischen Solarstrom und Netzversorgung gewählt werden. Die Solarstromversorgung wurde pro Bahn durch 2 Solarpanels und einem Tiefsetzsteller mit konstantem Duty cycle hergestellt, die Netzstromversorgung mit einem 15V Schaltnetzteil.

Die manuelle Steuerung der Fahrzeuge durch den Nutzer wurde realisiert, indem die klassischen Carrera Handregler, als Variablen Vorwiderstand zu den Fahrzeugen eingesetzt wurden. Im automatisierten Modus konnte man mit einem Vorwiderstand die Geschwindigkeit der langsamten Streckenabschnitte einstellen. In den schnellen Steckenabschnitten wurde mit einem Relais dieser Widerstand gebrückt und es lag die volle Betriebsspannung der jeweiligen Energieversorgung an dem Auto an.

1.1 Probleme der vorhandenen Carrera Bahn und Zielsetzung zur Projektarbeit

Der im Vorwort grob erwähnte Aufbau der vorhandenen Anlage wies einige Probleme auf:

- 1. Solarstromversorgung und Netzversorgung sind nicht chancengleich ausgeführt. (Die Netzversorgung stellt eine stabilisierte Spannungsquelle dar, die Solarstromversorgung nicht)
- 2. Drift des Widerstandswerts des Potentiometer um den Arbeitspunkt im automatisierten Modus einzustellen (Temperaturdrift).
- 3. Framerate der Visualisierung ist zu gering, sodass diese immer die selbe Zahlensequenz nach dem Komma anzeigt.
- Manchmal wird durch einen Aliasingeffekt das Überfahren eines Sensors in der Bahn nicht erkannt und die Geschwindigkeit wird nicht umgeschalten. (Echtzeitfähigkeit des Betriebssystems Raspian nicht gegeben)

Die oben genannten Probleme sollten durch ein neues Steuer-/Regelkonzept gelöst werden.

Kapitel 2

Funktion - **Hardware**

Im Folgenden wird der grundlegende Aufbau der Carrera Bahn erklärt.

Diese Beschreibung ist immer nur für Bahn-A, da die Bahn-B analog dazu funktioniert. Dazu wird immer wieder auf die Abbildung 2.2, sowie auf die Abbildung 2.1 Bezug genommen.

Die Sensoren in den Schienen teilen die Bahn in 3 Streckenabschnitte auf:

- 1. Startlinie → Vor dem Looping
- 2. Vor dem Looping → Nach dem Looping
- 3. Nach dem Looping → Startlinie

Auf Abschnitt 1 und Abschnitt 3 wird im automatisierten Modus die mittlere Geschwindigkeit des Autos geregelt. Im Looping (Abschnitt 2) wird die Spannung auf einen konstanten Wert geregelt. Die Grenze dafür ist durch die Position der Sensoren festgelegt und somit nicht variabel. In Abbildung 2.2 bekommt man einen groben Überblick über den Signalfluss auf der Carrera-bahn. Hierbei stellt der Arduino das zentrale Element dar. Man kann in der Abbildung erkennen, dass der Arduino die Sensoren in den Schienen, sowie die Taster des Bediefelds parallel einliest. Die Analogen Größen sind an den Analog-Digital-Wandler des Arduinos angeschlossen. Dies sind die Signal der Potentiometer, sowie der Tiefsetzsteller (differenzielle Messung des Stroms über ein Shunt und Messung der Ausgangsspannung). Der Arduino generiert außerdem die zwei PWM-Signale, die von den Tiefsetzstellern benötigt werden. Um eine Visualisierung zu realisieren ist ein RaspberryPi per UART (Universal Asynchronous Receiver Transmitter) seriell angebunden. Wie die anderen Kommunikationsverbindungen, ist auch diese Verbindung nur unidirektional ausgeführt. Die genaue Implementierung der Schnittstellen ist in den jeweiligen Kapiteln genauer beschrieben.

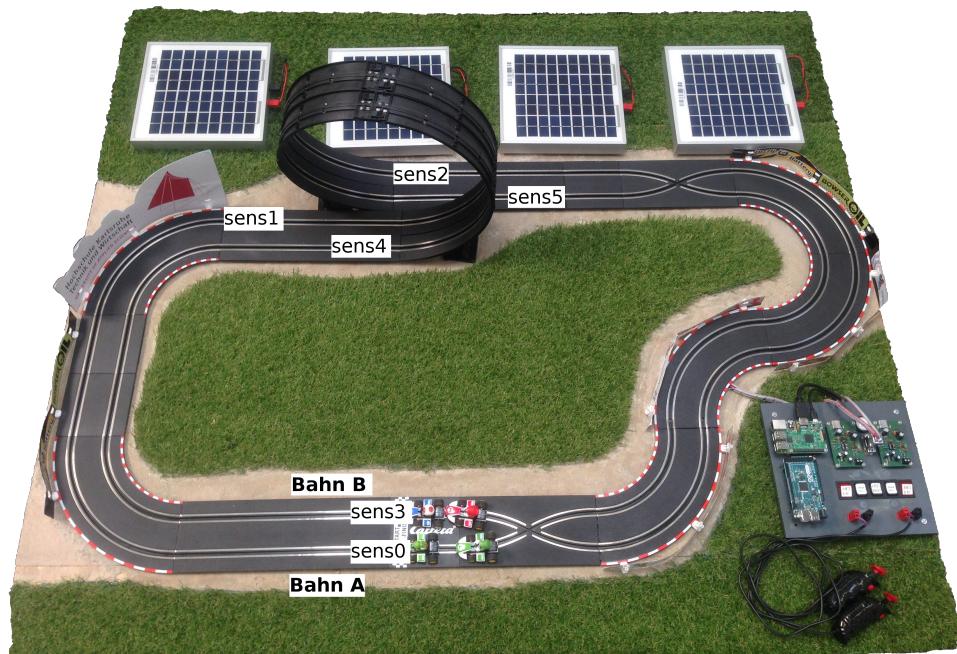


Abbildung 2.1: Überblick über die Carrera-Bahn

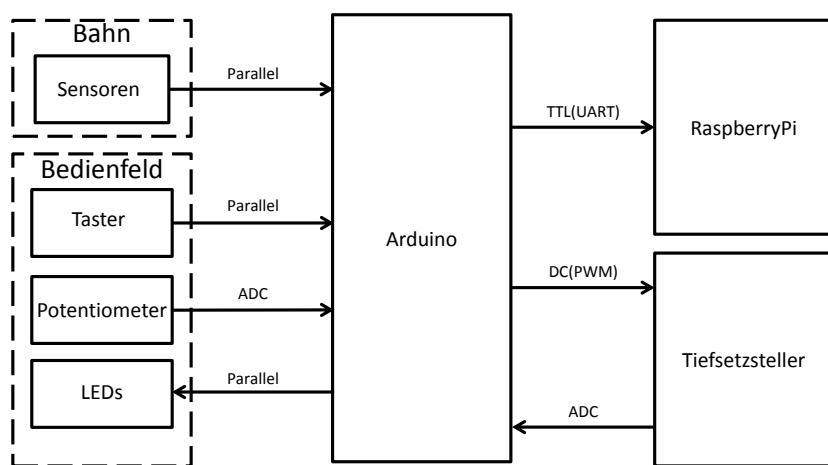


Abbildung 2.2: Signalfluss

2.1 Sensoren

Wie man in Abbildung 2.1 sehen kann, gibt es pro Bahn 3 Sensoren. Diese sind als Gabellichtschranke ausgeführt. Sensor 0 befindet sich am Start, Sensor 1 vor dem Looping und Sensor 2 Nach dem Looping.

Sensor 0 wird im Wesentlichen zur Zeitmessung genutzt. Sensor 1 wird genutzt um ein Signal zu generieren, so dass die Steuerung das Auto im Automatik-Modus für den Looping beschleunigen kann. Das Signal von Sensor 2 wird schließlich genutzt um nach dem Looping wieder die langsame Geschwindigkeit zu triggern.

Im manuellen Modus dienen die Sensoren nur zur Zeitmessung für die Visualisierung. Da die Flanke eines Sensors nur sehr kurz ist, lässt sich diese nicht per Polling ohne Aliasing Effekte digitalisieren. Stattdessen hat man die Hardware Pin Change Interrupts des Atmega2560 genutzt.

2.2 Bedienschnittstelle

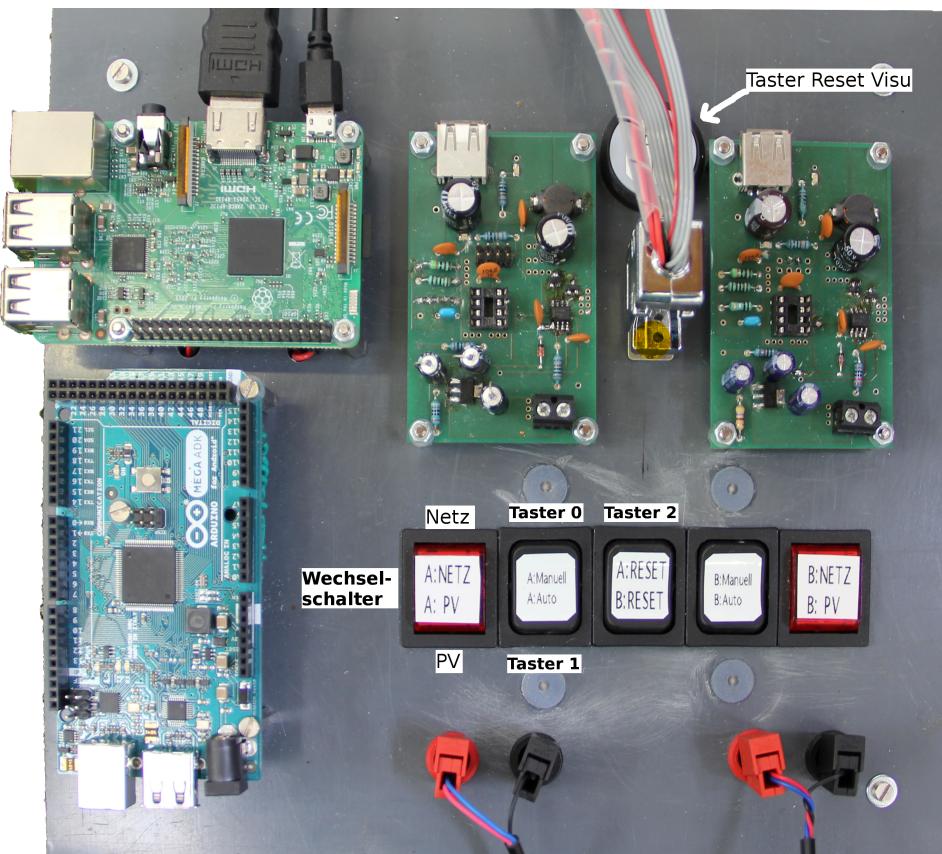


Abbildung 2.3: Bedienschnittstelle

Wie in Abbildung 2.3 dargestellt, besteht das HID (Human Interface Device)

aus 3 Tastern und einem Wechselschalter mit Mittelstellung je Bahn. Der Wechselschalter ist zum Auswählen der Energieversorgung der jeweiligen Bahn. Hierbei stehen die 2 Versorgungsmöglichkeiten Solar oder Netz zur Auswahl. Ist dieser Wechselschalter in Mittelstellung, wird die Bahn nicht versorgt und das Auto steht unabhängig des gewählten Modus. Über Taster 0 beziehungsweise Taster 1 lässt sich der Modus der Bahn auswählen. Taster 0 steht für den manuellen Modus Von Bahn A, Taster 1 für den automatisierten. Der aktuell ausgewählte Modus wird durch die LEDs über bzw unter den Tastern signalisiert. Mit Taster 2 lässt sich der Regler der Bahngeschwindigkeit, genauer Sollspannung außerhalb des Loopings, auf seinen Startwert zurücksetzen. Global existiert noch ein weiterer Taster um die Visualisierung (Rundenzeitmessung) zurückzusetzen.

2.3 Arduino

Beim Arduino handelt es sich um einen Arduino Mega 2560 ADK.

Diesen hat man gewählt, da es sich um eine preiswerte Platine handelt, welche bereits die wichtigsten Beschaltungen des Mikrocontrollers enthält. Dies sind zum Beispiel die Abblockkondensatoren an der Versorgungsspannung, aber auch einen USB-Seriell Wandler, den man nutzen kann, um sich Daten parallel zum Prozess an ein Terminal auszugeben. Letzteres lässt sich sehr gut für das Debugging nutzen.

2.4 RaspberryPi

Die Visualisierung ist durch ein RaspberryPi Model 3 B realisiert. Dieser empfängt per UART die codierten Signale der Sensoren, sowie des Reset Tasters der Visualisierung (siehe Abbildung 2.3). Die Visualisierung war zu Beginn der Projektarbeit bereits vorhanden und in Form eines Python Skripts implementiert.

2.5 Tiefsetzsteller

Die Tiefsetzsteller fungieren als Stellglieder der Spannungsregelungen der Schienen (innerer Regelkreis der Kaskadenregelung). Jeder bekommt sein PWM-Signal (Stellgröße) direkt vom Arduino. Des Weiteren ist über ein Shunt eine Strommessung realisiert. Der Wert liegt zwar im Arduino digital vor, wird allerdings nicht weiter verarbeitet und ist lediglich für weiterführende Projekte gedacht. Da die Ausgangsspannung des Tiefsetzstellers (lediglich begrenzt durch die Leerlaufspannung eines PV-Strangs) größer sein kann wie die Referenzspannung des ADC (interne Referenzspannung von 2.56V), wird diese über einen Spannungsteiler angepasst und auf einen Kanal des Analog-Digital-Wandlers des Mikrocontrollers geführt. Diese Spannung stellt die Regelgröße dar.

Kapitel 3

Software

3.1 RaspberryPi

Bei der Software die auf dem RaspberryPi ausgeführt wird, handelt es sich um ein Python Skript.

Dieses war zu Beginn der Projektarbeit bereits vorhanden und hat die Sensoren der Bahn parallel mit dem GPIO-Port eingelesen. Mit den Flanken der Sensoren **wurde** die Rundenzeit gemessen und die Bestzeit pro Bahn ermittelt.

Dieses Skript hat **man** größtenteils übernommen und lediglich in der Richtung abgeändert, dass die Trigger-Signale der Sensoren nun über die serielle Schnittstelle dem Pi mitgeteilt werden. Des Weiteren **hat man die** Framerate der Visualisierung angehoben indem man nicht immer die komplette **ildschirmseite** neu lädt, sondern lediglich die Bereiche, die dynamischen Inhalt repäsentieren. Als serielle Schnittstelle ist ttyAMA0 verwendet.

3.2 Arduino

Die Software des Arduinos ist nicht in der Arduino Entwicklungsumgebung geschrieben. Stattdessen hat man auf die IDE AtmelStudio 7 zurückgegriffen und den Mikrocontroller per ISP beschrieben. Der Bootloader des Arduino wurde dazu entfernt. Gründe dazu waren unter anderem die Wahl der Sprache C++, aber auf die Möglichkeit direkt auf die Hardware zuzugreifen (Timer, Hardware-interrupts...). Letzteres ist notwendig um das Timing des Controllers exakt zu steuern. Zu Beginn des Programms wird die ganze Peripherie konfiguriert und die globalen Variablen mit ihren default Werten geladen. Anschließend werden die Interrupts global freigegeben. Ab hier ist die Carrerabahn funktionsfähig.

3.2.1 Interrupts

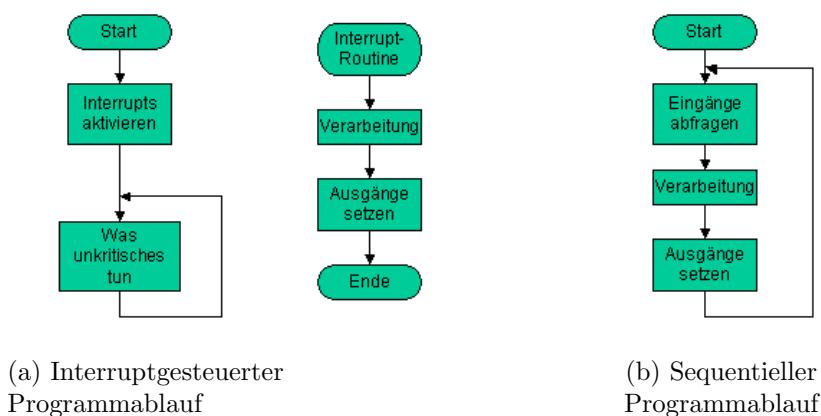


Abbildung 3.1: Möglichkeiten zum Programmablauf
Quelle: [Mikrocontroller.net](#)

Der Programmablauf ist größtenteils interruptgesteuert und folgt damit dem Modell in Abbildung 3.1a. Dies hat den Vorteil, dass das Timing nichtmehr von der Länge der MainLoop abhängt und Dinge wie zum Beispiel der Regelalgorithmus immer mit der selben Frequenz aufgerufen werden. In der Mainloop des Programms werden zeitunkritische Dinge erledigt, wie die LEDs der HMI zu aktualisieren. Interruptquellen des Programms sind:

- Timer (Abschnitt 3.2.1.1)
 1. Timer1, Compareregister B match
 2. Timer3, Compareregister A match
 3. Timer4, Compareregister A match
- Pin Change Interrupts (Abschnitt 3.2.1.2)
 4. PCINT0_vect
 5. PCINT1_vect

6. PCINT2_vect

- Analog Digital Converter

7. ADC_vect (Abschnitt 3.2.1.3)

3.2.1.1 Timer

Für **Die** Funktion der Steuerung werden viele Timer benötigt.

Da im Mikrocontroller allerdings nur begrenzt Timer zur Verfügung stehen, hat **man** deren Funktionalität in einem Software-Timer nachgebildet. Dieser bezieht **sein** Takt von einem Hardwretimer. Die Verwendung der Hard- sowie Softwretimer kann der Tabelle 3.1 entnommen werden.

Timer	Funktion
Hardwaretimer	
Timer0	Generierung der 2 PWM Kanäle für die Tiefsetzsteller
Timer1	Auslösen der Analog Digital Wandlung
Timer2	Ohne Verwendung
Timer3	Trigger für Spannungsregelung
Timer4	Trigger für Softwretimer
Softwretimer	
0..1	Zeitmessung Abschnitt 3 (Nach dem Looping → Startlinie)
2..7	Entprellen der Bahnsensoren
8..14	Entprellen der HID Taster
15..16	Ohne Verwendung
17	Trigger für Zeitmessung Resettaster Visualisierung (langer Tastendruck)

Tabelle 3.1: Belegung der Hard- sowie Softwretimer

3.2.1.2 Pin Change Interrupt

Wenn ein Auto auf, beispielsweise Sensor 0 (Bahn A, am Start), fährt, wird der Pegel kurze Zeit *Low* und nach Verlassen des Sensors wieder *High*.

Da Sensor 0 an PCINT16 angeschlossen ist und **16 im Bereich [16,23]** liegt, wird das letzte Pin Change Interrupt, PCINT2_vect, zweimal ausgelöst. In der ISR (Interrupt Service Routine) muss nun unterschieden werden, durch welchen Pin das Interrupt ausgelöst wurde. Dies hat **man** realisiert, indem man sich den Zustand des letzten Pin Change Interrupt gemerkt hat und diesen mit dem aktuellen Zustand, des jeweiligen Ports, durch ein Exklusivoder verknüpft. Außerdem gibt es die zwei Möglichkeiten:

- war es ein *High*→*Low* **Low** Übergang
- war es ein *Low*→*High* Übergang

Hat ein *High*→*Low* Übergang stattgefunden, wird ein Event erzeugt. Bei einem *Low*→*High* Übergang wird kein Event ausgelöst. Zum Entprellen des Eingangs wird direkt, nachdem das Event gehandelt wurde, der Eingang deaktiviert. Danach wird ein Software-Timer gestartet der den Eingang nach dessen Ablauf wieder aktiviert. Mit diesem einfachen Prinzip wird sichergestellt, dass das Auto beim Übefahren des Sensors, das dementsprechende Event nur einmal triggert. Analog zu dem Sensor 0, ist dies für jeden Sensor sowie Taster implementiert. Die Belegung der benutzten Pin Change Interrupts kann Tabelle 3.2 entnommen werden.

Zugehörigkeit	Signal	Bezeichnung
PCINT0_vect	PCINT4	Button: B-Automatik
	PCINT5	Button: B-Manuell
	PCINT6	Button: A-Automatik
PCINT1_vect	PCINT9	Button: A-Reset
	PCINT10	Button: B-Reset
PCINT2_vect	PCINT16	Sensor: 0
	PCINT17	Sensor: 1
	PCINT18	Sensor: 2
	PCINT19	Sensor: 3
	PCINT20	Sensor: 4
	PCINT21	Sensor: 5
	PCINT22	Button: Reset/Shutdown Pi
	PCINT23	Button: A-Manuell

Tabelle 3.2: Belegung der Pin Change Interrupts

3.2.1.3 Analog Digital Converter Interrupt

Zum Betrieb des Zustandsautomats (Abbildung 3.3) welcher den Multiplexer des ADC steuert ist ein Interrupt notwendig, welches ausgelöst wird, wenn der ADC eine Messung abgeschlossen hat. Dieser Automat ist genauer in Abschnitt 3.2.3 beschrieben.

3.2.2 Regler

Die Regelung für die Bahngeschwindigkeit ist als Kaskadenregelung (Abbildung 3.2) ausgeführt. Dem Führungsregler wird eine konstante Sollzeit für Streckenabschnitt 3 vorgegeben. Dieser gibt nun, im Automatikmodus, die Sollspannung für den Folgeregler vor. Im manuellen Modus entspricht die Sollspannung einer skalierten Größe des jeweiligen Handreglers. Der Folgeregler hat als Stellgröße den Duty-Cycle des zugehörigen Tiefsetzstellers und regelt die Spannung auf der Schiene auf den gegebenen Wert. Der Regelalgorithmus des Spannungsreglers wird in festen Zeitabständen, vorgegeben durch OCR3 (Overflow Compare Register Timer3), zyklisch aufgerufen. Der Regelalgorithmus des Reglers der Bahngeschwindigkeit wird immer dann aufgerufen, wenn ein neuer Zeitwert für

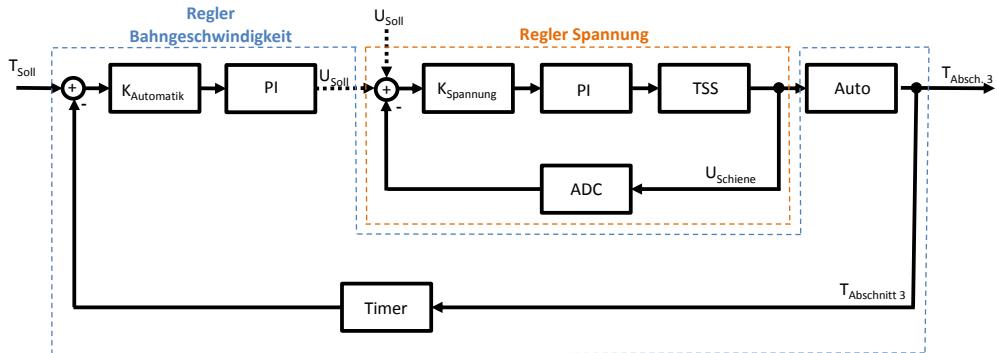


Abbildung 3.2: Kaskadenregelung Bahngeschwindigkeit

den Streckenabschnitt 3 vorliegt. Dieses Vorgehen ist durch die geringe Abtastrate des Führungsreglers sehr störanfällig (**Zum Beispiel wenn man das Auto in Streckenabschnitt 3 festgehalten wird**), allerdings ist es die einzige Möglichkeit, einen solche Regelung mit den vorhandenen Sensoren zu realisieren. Die Bedingung für eine Kaskadenregelung (inner Regelkreis mindestens 10mal schneller als der Äußere) ist sicher eingehalten.

Der Vorfaktor des Führungsreglers ist eher verhältnismäßig klein gewählt, um ein Überschwingen möglichst zu Verhindern, da dies bedeuten würde, dass das Auto die Strecke verlässt. **Die gerade beschriebene Regelstrategie ist als solche nur im Automatik-Modus aktiv.**

Im manuellen Modus wird die Sollspannung direkt durch den Handregler vorgegeben.

3.2.3 Analog Digital Wandler

Der Mikrocontroller des Arduinos (Atmel Atmega 2560) hat bereits einen 10 bit ADC (Analog Digital Converter) auf dem Chip integriert, sodass auf einen externen Wandler, verzichtet werden kann. Da der Wandler immer nur eine Messung durchführen kann, hat der Hersteller ein (MUX) Multiplexer integriert, um zwischen den einzelnen ADC Pins des Mikrocontroller durchzuschalten. Die Belegung der ADC Pins des Mikrocontrollers ist in Tabelle 6.3 enthalten.

Wenn die angestoßene Messung des ADC abgeschlossen ist, wird ein Interrupt ausgelöst.

Die Steuerung des Multiplexers erfolgt über ein Zustandsautomat (Abbildung 3.3), der immer dann ein Schritt weiter springt, wenn die letzte Messung des ADC abgeschlossen ist. In jedem Schritt des Automats, wird schematisch das **Selbe** durchgeführt:

- Letzter gemessener Wert in Variable speichern
- Nächste Messung vorbereiten (Multiplexer steuern und Messung anstoßen)

3.2.4 PWM Generierung Tiefsetzsteller

Wie aus Abbildung 3.4b hervorgeht, ist das PWM Signal direkt eine Funktion des aktuellen Zählerstands von Timer0, sowie dem Inhalt des jeweiligen Overflow Compare Registers OCR0A/B. Die PWM Generierung ist durch den Mikrocontroller in Hardware implementiert und muss lediglich aktiviert werden. Die Konfiguration dieser PWM Generierung ist in der Funktion initTimer implementiert und wird einmal zu Programmstart aufgerufen. Der Hardwarter-timer setzt nun den jeweiligen Pin beim Überlauf des 8-bit Timers (Wert 0) auf High-Pegel und beim Erreichen des Wertes im OCR0AB Registers (zum Beispiel bei DutyCycle = 50% OCR0A = 127) auf Low-Pegel. Dieses Verhalten des Hardwartertimers ist zur Veranschaulichung nochmal in Abbildung 3.4a als Zustandsautomat dargestellt.

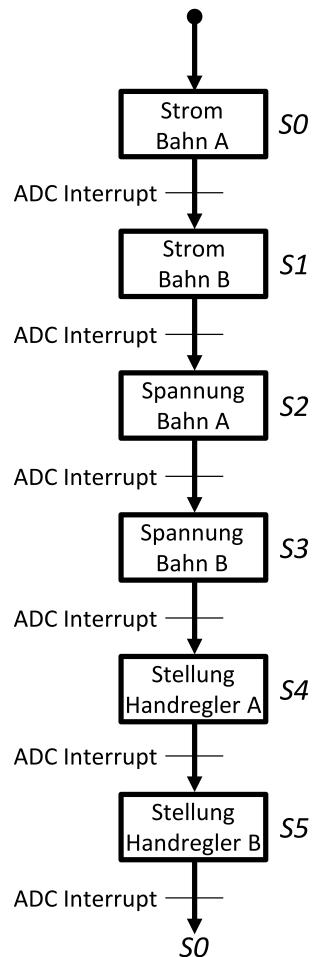
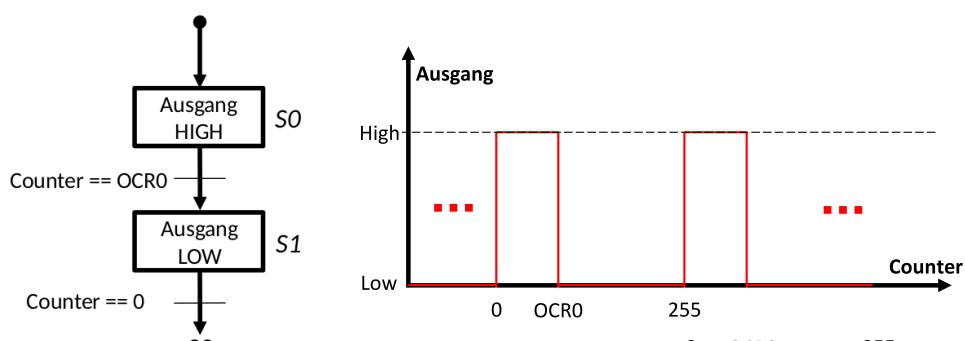


Abbildung 3.3: Zustandsautomat Analog Digital Converter Kanal



(a) Zustandsautomat PWM (b) Ausgang PWM-Pin als Funktion des Zählerstands
Generierung

Kapitel 4

Bedienungsanleitung

4.1 Einschalten

Die Anlage muss zum Einschalten lediglich mit der Netzversorgung verbunden werden. Sobald Die Anlage Spannung hat, bootet der RaspberryPi selbstständig und startet die Visualisierung. Ab diesem Punkt ist Die Carrerabahn betriebsbereit und im Modus manuelles Fahren (Abschnitt 4.2.1).

4.2 Betrieb

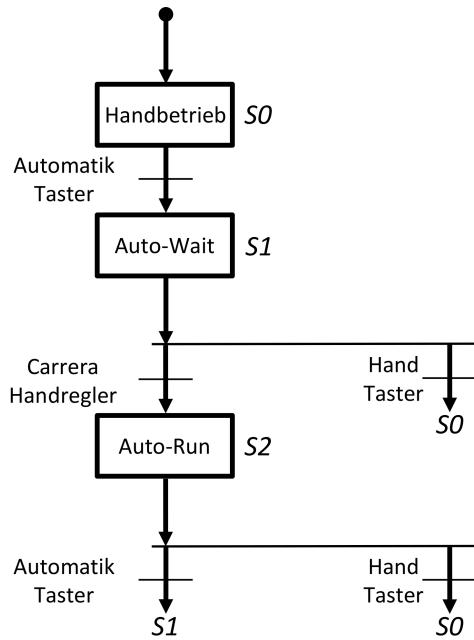


Abbildung 4.1: Zustandsautomat Betriebsmodi

Die Carrera-Bahn beherrscht 2 Modi.

- Manuelles Fahren

- Automatisiertes Fahren

Die Modi können unabhängig voneinander auf beiden Bahnen gewählt werden, sodass man zum Beispiel manuell gegen ein automatisiertes Fahrzeug antreten kann. Die Wahl eines dementsprechenden Modus wird durch eine Led über, beziehungsweise unter, des jeweiligen Tasters des gewählten Modus signalisiert. Außerdem kann die Energieversorgung jeder Bahn mit dem entsprechenden Wechselschalter ausgewählt werden. Unabhängig des gewählten Modus, wird in der Visualisierung die Zeit der 2 Abschnitte, sowie die Rundenzeit angezeigt. Nach 30 Runden ist das Rennen vorbei und die Visualisierung wird statisch. Der Wechsel zwischen den Modi findet nach Abbildung 4.1 statt und ist in den jeweiligen Abschnitten genauer erklärt.

4.2.1 Manuelles Fahren

Im Modus Manuelles Fahren kann das Fahrzeug konventionell über die orginalen Carrera Handregler gesteuert werden.

4.2.2 Automatisiertes Fahren

Der Automatisierte Modus ist aufgeschlüsselt in 2 Zustände:

- Auto-Wait
- Auto-Run

4.2.2.1 Auto-Wait

Immer wenn in den Modus des Automatisierten Fahrens gewechselt wird, startet die Steuerung in Auto-Wait und das Auto steht. Durch kurzes durchdrücken des zugehörigen Handreglers, wechselt das Auto in den Auto-Run Zustand.

4.2.2.2 Auto-Run

Im Auto-run Zustand fährt das Auto selbständig. Durch die Regelung der mittleren Geschwindigkeit des langsam Streckenabschnitts, braucht das Auto einige Runden um die optimale Spannung, für diese Geschwindigkeit zu finden. In der Visualisierung ist die Strecke von der Startlinie bis vor den Looping in Streckenabschnitt 1 zusammengefasst. Der Rest einer Runde (vor dem Looping bis zur Startlinie) bildet Streckenabschnitt 2.

4.3 Ausschalten

Um die Anlage ordnungsgemäß herunterzufahren, ist der Reset Taster am HMI (Abbildung 2.3 - Reset Visualisierung) mindestens 5 Sekunden gedrückt zu halten. Nun muss man warten bis der RaspberryPi komplett heruntergefahre ist. Dies ist daran zu erkennen, dass die grüne Led auf dem RaspberryPi nichtmehr blinkt und der Bildschirm in den Standby-Modus schaltet.

Kapitel 5

Zusammenfassung

Die Zu Beginn definierten Ziele der Projektarbeit wurden durch das neue Steuer-/Regelkonzept, sowie eine neue Hardware Architektur, fast alle erreicht. Lediglich die Robustheit der Anlage ist nicht vollsändig gegeben, da gelegentlich ein Auto im Looping herunterfällt. Durch längere Tests und Analyse der Fehlerbilder, hat sich herausgestellt, dass dies kein Problem der Automatisierung und Regelung selbst darstellt, sondern vielmehr der schlechten Qualität der zugekauften Carrerabahn. Man hat herausgefunden dass im Looping die Stromschiene auf der Bahn, ein wenig in der Oberfläche versenkt ist und der Übergang zwischen den zusammengesteckten Streckenstücken einen unstetigen Verlauf hat. Dadurch ergibt sich eine Diskontinuität des Stroms durch den Antrieb des Fahrzeugs im Looping, was zu Geschwindigkeitsverlust, sowie letztendlich zum Absturz des Fahrzeugs führt. Biegt man nun die Strombürsten des Fahrzeugs weiter in Richtung Schiene um die schlechte Mechanische Beschaffenheit auszugleichen, taucht die Führungsnahe nicht mehr weit genug in die Bahn ein und das Fahrzeug fliegt aus der Kurve oder triggert die Sensoren nichtmehr verlässlich. Ein weitere Randbedingung für die Realisierung war dass die Automatisierung unabhängig von der lange der Strecke funktionieren sollte. Aufgrund dieser Forderung musste die Funktionalität, das Auto auf der Geraden zu beschleunigen verworfen werden, da dies nun nichtmehr in Abhängigkeit einer konstanten Zeit realisiert werden kann. Als weiterführende Verbesserung könnte man zusätzliche Sensoren vor der Kurve zum abbremsen einbauen. Ein weiteren Vorschlag zur Weiterarbeit wäre die Energieversorgung des RaspberryPi, sowie des Arduino auch autark über die Solarpanels zu ermöglichen.

Kapitel 6

Anhang

6.1 Pinbelegung

Sub-D Pin	Funktion
1	VCC (5V)
2	GND
3	-
4	-
5	-
6	Sensor 0
7	Sensor 1
8	Sensor 2
9	Sensor 3
10	Sensor 4
11	Sensor 5
12	Schiene A - Plus
13	Schiene A - Minus
14	Schiene B - Plus
15	Schiene B - Minus

Tabelle 6.1: Pinbelegung der Sub-D Buchse auf dem HID

Raspberry Pin	Funktion
1	VCC MCU (3V3)
2	VCC Input (5V)
3→5	-
6	GND Input
7	-
8	-
9	GND Uart
10	RXD Uart
11→40	-

Tabelle 6.2: Pinbelegung GPIO RaspberryPi

Arduino Pin	Atmel Pin	Funktion
0		Ohne Funktion
1		Ohne Funktion
2		Ohne Funktion
3		Ohne Funktion
4	OC0B	PWM Tiefsetzsteller Schiene B
5		Ohne Funktion
6		Ohne Funktion
7		Ohne Funktion
8		Ohne Funktion
9		Ohne Funktion
10	PCINT4	Taster HID „Schiene B - Automatik“
11	PCINT5	Taster HID „Schiene B - Manuell“
12	PCINT6	Taster HID „Schiene A - Automatik“
13	OC0A	PWM Tiefsetzsteller Schiene A
14	PCINT10	Taster HID „Schiene B - Reset Regler Bahngeschwindigkeit“
15	PCINT9	Taster HID „Schiene A - Reset Regler Bahngeschwindigkeit“
16		Ohne Funktion
17		Ohne Funktion
18		Ohne Funktion
19→45		Ohne Funktion
46	PL3	HID LED „Schiene B - Manuell“
47	PL2	HID LED „Schiene B - Automatik“
48	PL1	HID LED „Schiene A - Manuell“
49	PL0	HID LED „Schiene A - Automatik“
A1→A0	ADC1→ADC0	Shunt Tiefsetzsteller Schiene A
A3→A2	ADC3→ADC2	Shunt Tiefsetzsteller Schiene B
A4	ADC4	Spannung Schiene A
A5	ADC5	Spannung Schiene B
A6	ADC6	Carrera Handregler Schiene A
A7	ADC7	Carrera Handregler Schiene B
A8	PCINT16	Sensor 0 (Schiene A - Startlinie)
A9	PCINT17	Sensor 1 (Schiene A - vor dem Looping)
A10	PCINT18	Sensor 2 (Schiene A - nach dem Looping)
A11	PCINT19	Sensor 3 (Schiene B - Startlinie)
A12	PCINT20	Sensor 4 (Schiene B - vor dem Looping)
A13	PCINT21	Sensor 5 (Schiene B - nach dem Looping)
A14	PCINT22	Taster HID „Reset Rundenzeit Visualisierung“
A15	PCINT23	Taster HID „Schiene A - Manuell“

Tabelle 6.3: Pinbelegung des Arduino