

Dokumentation der Projektarbeit - Carrera Bahn

Florian Weber - 44907

28. August 2018

Inhaltsverzeichnis

1 Vorwort	4
1.1 Zustand der Carrera Bahn vor der Projektarbeit	5
1.2 Probleme der vorhandenen Carrera Bahn und Zielsetzung zur Projektarbeit	6
2 Funktion - Hardware	7
2.1 Sensoren	9
2.2 Human Maschine Interface	9
2.3 Arduino	10
2.4 RaspberryPi	10
2.5 Tiefsetzsteller	11
3 Software	12
3.1 RaspberryPi	12
3.2 Arduino	12
3.2.1 Interrupts	12
3.2.1.1 Timer	13
3.2.1.2 Pin Change Interrupt	13
3.2.1.2.1 Unterschied: Pin Change Interrupt - Poling mit echtzeitfähigem System - Poling ohne echtzeitfähigem System	15
3.2.1.3 Analog Digital Converter Interrupt	16
3.2.2 Regler	16
3.2.2.1 Bahnspannung	17
3.2.2.2 Bahngeschwindigkeit	17
3.2.3 Analog Digital Wandler	19
3.2.4 PWM Generierung Tiefsetzsteller	19
4 Bekannte Störungen	22
5 Bedienungsanleitung	23
5.1 Einschalten	23
5.2 Betrieb	23
5.2.1 Manuelles Fahren	23
5.2.2 Automatisiertes Fahren	24
5.2.2.1 Auto-Wait	24
5.2.2.2 Auto-Run	24

5.3	Ausschalten	25
6	Reflexion	26
7	Quellenverzeichnis	27
8	Anhang	28

Abbildungsverzeichnis

1.1	Überblick über die Carrera Bahn	5
2.1	Energiefluss einer Bahn	8
2.2	Signalflussbild der Steuerung	8
2.3	HMI mit Arduino, RaspberryPi und TSS	9
3.1	Interruptgesteuerter Programmablauf	13
3.2	Abtastung eines digitalen Signals	16
3.3	Regelkreis Bahnspannung	17
3.4	Regelkreis Bahngeschwindigkeit	18
3.5	Zustandsautomat des MUX im ADC	20
3.6	Zustandsautomat des Timers zur PWM Generierung, sowie resultierendes Ausgangssignal bei konstantem Dutycycle.	21
5.1	Zustandsautomat zum Wechsel der Betriebsmodi einer Bahn . .	24

Kapitel 1

Vorwort

Die Carrera Bahn ist in Abbildung 1.1 dargestellt. Sie ermöglicht es, zwei Autos gegeneinander antreten zu lassen. Dabei kann bei jedem Auto gewählt werden, ob es automatisch fährt oder durch einen Nutzer manuell gesteuert wird. Die Energieversorgung der jeweiligen Bahn kann unabhängig davon gewählt werden. Es ist dadurch zum Beispiel möglich, als Studierender ein Auto mit Netzstrom manuell, gegen ein automatisch fahrendes Auto, welches mit Solarstrom betrieben wird, zu steuern. Dies soll die Chancengleichheit der zwei Energieversorgungen demonstrieren.

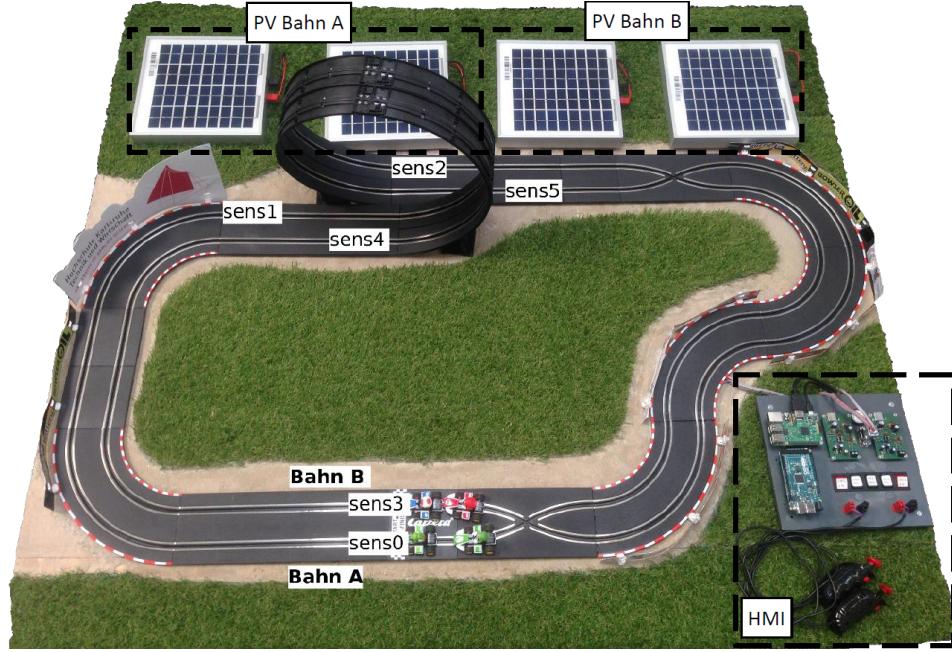


Abbildung 1.1: Überblick über die Carrera Bahn mit den Photovoltaik Modulen, der Steuerung mit dem Human Interface Device (HMI), sowie der Bahn mit den markierten Sensorpositionen.

1.1 Zustand der Carrera Bahn vor der Projektarbeit

Zu Beginn des Projekts ist die Carrera Bahn auf dem Stand, dass die Fahrzeuge schon konventionell sowie automatisiert betrieben werden können. Außerdem ist die Rundenzeitmessung und deren Visualisierung bereits realisiert. Bei der Energieversorgung kann zwischen Solarstrom und Netzversorgung gewählt werden. Die Solarstromversorgung wird pro Bahn durch 2 Solarpanels und einem Tiefsetzsteller mit konstantem Dutycycle hergestellt, die Netzstromversorgung mit einem 15V Schaltnetzteil.

Die manuelle Steuerung der Fahrzeuge durch den Nutzer ist realisiert, indem die klassischen Carrera Handregler, als variablen Vorwiderstand zu den Fahrzeugen eingesetzt werden. Im automatisierten Modus ist es möglich, mit je einem 5W Potentiometer die Geschwindigkeit des langsamen Streckenabschnittes einzustellen. In dem schnellen Steckenabschnitt (dem Looping) überbrückt ein Relais diesen Potentiometer und es liegt die volle Betriebsspannung, der jeweiligen Energieversorgung am Auto an.

1.2 Probleme der vorhandenen Carrera Bahn und Zielsetzung zur Projektarbeit

Der zuvor erwähnte Aufbau der vorhandenen Anlage vor der Projektarbeit weist einige Probleme auf:

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

Die oben genannten Probleme sollen im Rahmen der Projektarbeit durch ein neues Steuer-/Regelkonzept, sowie einen Mikrocontroller mit echtzeitfähiger Software gelöst werden.

Kapitel 2

Funktion - Hardware

Im Folgenden wird der grundlegende Aufbau der Carrera Bahn nach Abschluss der Projektarbeit 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 1.1 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 (Streckenabschnitt 2) wird die Spannung auf einen konstanten Wert geregelt. Die Grenze dafür ist durch die Position der Sensoren festgelegt und somit nicht variabel. Wie in Abbildung 2.1 zu sehen ist, ist der Tiefsetzsteller (TSS) immer zwischen die Energiequelle und das Auto geschaltet. Somit ist es möglich die Bahnspannung zu steuern.

In Abbildung 2.2 ist eine Übersicht über den Signalfluss der Carrera Bahn dargestellt. Auf der Bahn stellt der Arduino das zentrale Element dar. Dieser liest die Sensoren in den Schienen, sowie die Taster des Human Machine Interfaces (HMI) parallel ein und gibt die Signale der LEDs, welche den aktuellen Zustand des jeweils aktiven Modus signalisieren, parallel aus. Die analogen Größen (Ausgangsspannung TSS, Spannung über Strommessshunt TSS, Ausgangsspannung Spannungsteiler Handregler) sind an den Analog-Digital-Wandler (ADC) des Arduinos angeschlossen. Der Arduino generiert außerdem die zwei PWM Signale, die von den Tiefsetzstellern benötigt werden, über einen 8-bit Hardwaretimer. Dies ist genauer in Abschnitt 3.2.4 beschrieben. Um eine Visualisierung zu realisieren ist ein RaspberryPi per Universal Asynchronous Receiver Transmitter (UART) 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.

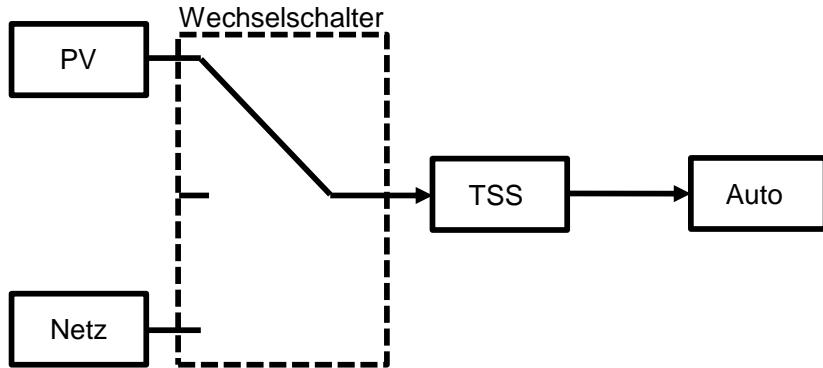


Abbildung 2.1: Energiefluss einer Bahn.

Durch einen Wechselschalter, kann der Nutzer zwischen den zwei Energieversorgungen Photovoltaik (PV) und Netz wählen, oder die Versorgung komplett ausschalten. Der Tiefsetzsteller (TSS) ist dabei immer zwischen Energiequelle und dem Auto auf der Bahn geschaltet.

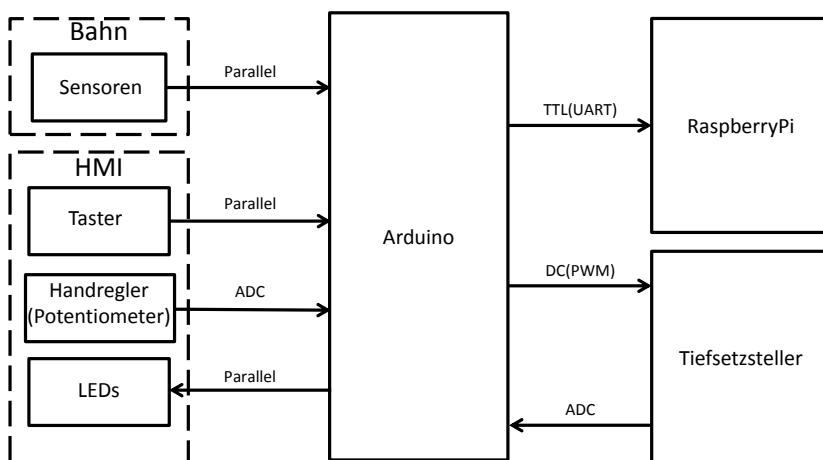


Abbildung 2.2: Signalflussbild der Steuerung.

Arduino liest Signale von Sensoren, sowie Tastern parallel aus und misst die Ausgangsspannung der Tiefsetzsteller (TSS) und Handregler. Auf Grundlage dieser Daten, generiert der Arduino die PWM Signale für die TSS und leitet die Sensordaten per Universal Asynchronous Receiver Transmitter (UART), an den RaspberryPi, zur Rundenzzeitmessung weiter.

2.1 Sensoren

Wie in Abbildung 1.1 zu sehen ist, gibt es pro Bahn 3 Sensoren. Diese sind als Gabellichtschranken ausgeführt. Sensor 0 befindet sich am Start von Bahn A, Sensor 1 vor dem Looping und Sensor 2 nach dem Looping.

Sensor 0 wird ausschließlich 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. Sensor 3 bis 5 stellen die selben Signale, analog zu Bahn A, von Bahn B bereit. 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 werden die Signale der Bahnsensoren durch Hardware Pin Change Interrupts (PCINT) des Arduino digitalisiert. Die Funktionsweise dieser PCINTs ist genauer im Abschnitt 3.2.1.2 erklärt

2.2 Human Maschine Interface

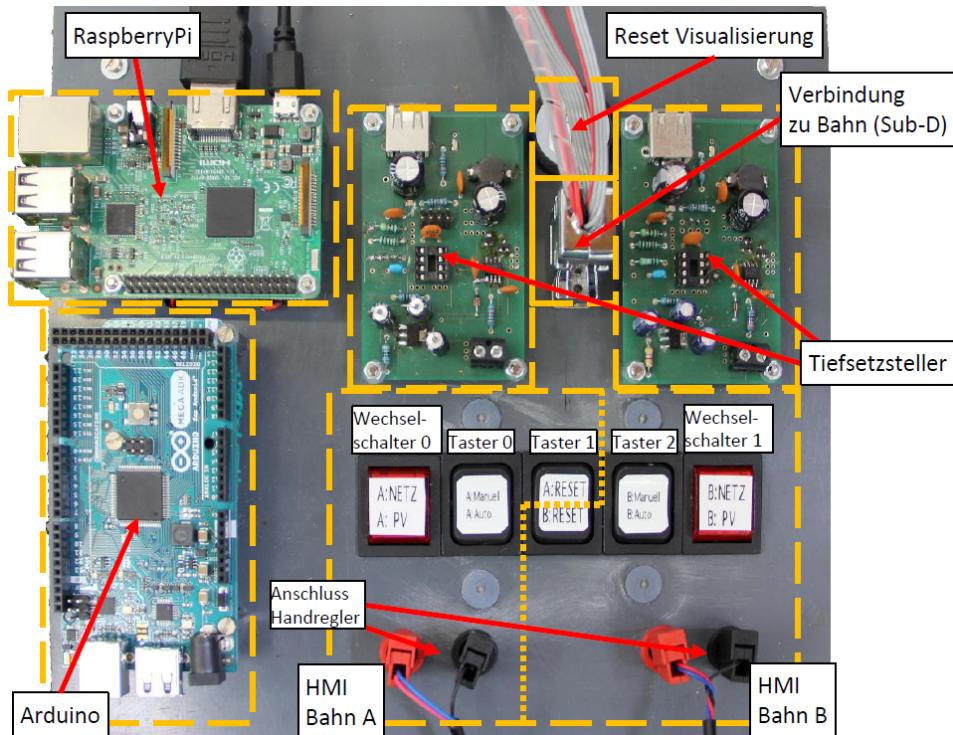


Abbildung 2.3: HMI mit Arduino, RaspberryPi und TSS

Wie in Abbildung 2.3 dargestellt, besteht das Human Maschine Interface (HMI) aus drei Wechseltaster mit Mittelstellung, sowie einem Wechselschalter mit Mit-

telstellung.

Der Wechselschalter („Wechselschalter 0“ für Bahn A und „Wechselschalter 1“ für Bahn B) ist zum Auswählen der Energieversorgung der jeweiligen Bahn. In der oberen Stellung wird die Bahn mit Netzenergie versorgt, in der unteren Stellung mit Solarenergie. Ist einer dieser Wechselschalter in Mittelstellung, wird die Bahn nicht versorgt und das entsprechende Auto steht, unabhängig des gewählten Modus.

Über die Wechseltaster 0 beziehungsweise Wechseltaster 2 lässt sich der Modus der zugehörigen Bahn auswählen. Dies erfolgt indem einer der Taster oben beziehungsweise unten betätigt wird. Der aktuell ausgewählte Modus wird durch die LEDs über beziehungsweise unter den Tastern signalisiert. Das LED oben signalisiert dabei den manuellen Modus, indem das Auto durch den Handregler gesteuert wird und ein LED unten, den Modus indem das Auto selbstständig fährt.

Mit Wechseltaster 1 lassen sich die Integratoren der Regler, der Bahngeschwindigkeit (Abbildung 3.4), welche ausschließlich im automatisierten Modus aktiv sind, auf ihren Startwert zurücksetzen. Dabei setzt ein oberer Tastendruck den Regler von Bahn A und ein unterer den Regler von Bahn B zurück. Diese Regler sind genauer in Abschnitt 3.2.2 beschrieben. Global existiert noch ein weiterer Taster um die Visualisierung (Rundenzzeitmessung) zurückzusetzen. Dieser ist in Abbildung 2.3 mit „Reset Visualisierung“ beschriftet und befindet sich zwischen den zwei Tiefsetzstellern oben links.

2.3 Arduino

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

Dieser ist mit einem ATmega2560 der Firma Microchip bestückt. Der ATmega2560 ist perfekt für die Anwendung geeignet, da er einen Analog-Digital-Wandler (ADC) zur Messung analoger Größen, Hardwaretimer mit PWM Funktionalität, sowie vier UARTs zur Kommunikation besitzt. Der Arduino kommt zum Einsatz, da er die Pins des Mikrocontrollers auf Buchsenleisten führt. So ist es unter anderem möglich, ohne größeren Aufwand, Drähte an die Pads der Buchsenleisten anzulöten. Außerdem enthält ein Arduino bereits alle zum Betrieb notwendigen Bauteile für den Mikrocontroller. Dies sind zum Beispiel die Abblockkondensatoren an der Versorgungsspannung, oder der Glättungskondensator an der Referenzspannung des ADCs. Des Weiteren enthält der Arduino einen USB-Seriell Wandler, der genutzt werden kann, um sich Daten parallel zum Prozess an ein Terminal auszugeben. Dies ist zum Beispiel sehr hilfreich, um das Programm zu debuggen, oder die Regelparameter einzustellen. Der Mikrocontroller ist mit 16MHz getaktet und wird direkt mit 5V aus dem Steckernetzteil des RaspberryPi versorgt.

2.4 RaspberryPi

Die Visualisierung ist durch ein RaspberryPi Model 3 B+ realisiert. Dieser empfängt per UART die codierten Signale der Sensoren, sowie des Tasters „Re-

set Visualisierung“ (siehe Abbildung 2.3). Die Visualisierung ist zu Beginn der Projektarbeit bereits vorhanden und in Form eines Python Skripts implementiert.

2.5 Tiefsetzsteller

Die Tiefsetzsteller (TSS) fungieren als Stellglieder der Spannungsregelungen der Bahnen. Dabei sind die Bahnen jeweils direkt an die Ausgänge der beiden TSS angeschlossen. Der Arduino generiert die zwei PWM Signal für die TSS. Die Ausgangsspannung der TSS wird vom Arduino gemessen und durch den in Abschnitt 3.2.2.1 auf ihre aktuellen Sollwerte geregelt. Die Ausgangsspannung eines TSS ist lediglich durch die Leerlaufspannung eines PV-Strangs ($21V_{max}$) nach oben hin begrenzt. Da diese größer sein kann wie die Referenzspannung des ADC ($V_{ref}=V_{CC} = 5V$), wird sie über einen Spannungsteiler auf $5V_{max}$ angepasst und auf einen Kanal des Analog-Digital-Wandlers des Mikrocontrollers geführt. Dies bewirkt die größtmögliche Aussteuerung des ADC ohne dass dieser übersteuert wird. Des Weiteren ist über ein Shunt eine Strommessung realisiert. Dieser Stromwert liegt zwar im Arduino digital vor, wird allerdings nicht weiter verarbeitet und ist lediglich für weiterführende Projekte gedacht.

Kapitel 3

Software

3.1 RaspberryPi

Bei der Software, die auf dem RaspberryPi ausgeführt wird, handelt es sich um ein Python Skript. Das Script bekommt die Signale der Sensoren, sowie des Tasters „Reset Visualisierung“, über die serielle Schnittstelle „ttyAMA0“, gesendet und führt eine Rundenzeitmessung je Bahn durch. Diese Zeitmessung wird auf dem Bildschirm angezeigt. Zum Rendern der Schriften sowie dem Hintergrundbild, ist die Open Source Bibliothek „PyGame“ in Verwendung. Das Bild ist separiert in einen Hintergrund und einem Block der den dynamischen Inhalt der Rundenzeitmessung enthält. Um die Framerate entgegen der Version aus einer vorherigen Arbeit zu steigern, wird lediglich der dynamische Inhalt framweise neu gerendert.

3.2 Arduino

Die Software des Arduinos ist nicht in der Arduino Entwicklungsumgebung geschrieben, da dies einen direkten Zugriff auf die Hardware erschwert und der Programmablauf bereits vorgegeben wäre. Stattdessen ist die Software in der integrierten Entwicklungsumgebung (IDE) „Atmelstudio 7.0“ entwickelt. Der Bootloader des Arduino wurde dazu entfernt. Dies ermöglicht nun einen direkten Zugriff auf die Prozessorregister und damit Konfiguration der einzelnen Komponenten des Mikrocontrollers, auf dem Arduino. Dies ist notwendig, um das Timing des Controllers exakt zu steuern und ist effizienter als für alle Hardwarekomponenten eine Bibliothek zu nutzen , wie es die Arduino Entwicklungsumgebung vorsehen würde.

3.2.1 Interrupts

Der Programmablauf ist interruptgesteuert und folgt damit dem Modell in Abbildung 3.1. Dies hat den Vorteil, dass das Timing nicht mehr von der Länge der MainLoop abhängt und Vorgänge wie zum Beispiel der Regelalgorithmus immer mit der selben Frequenz ausgeführt werden. In der Mainloop des Programms werden zeitunkritische Aufgaben erledigt, wie die LEDs der HMI zu

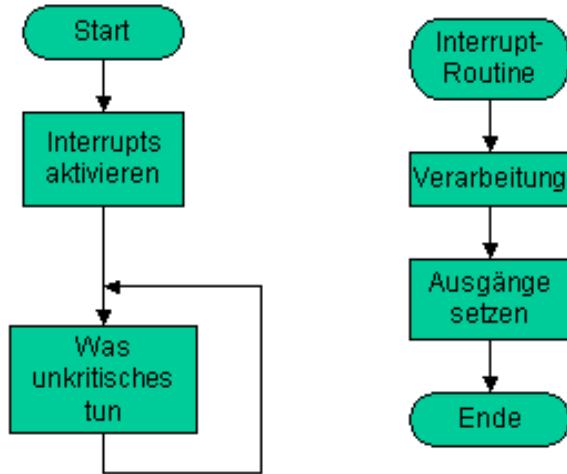


Abbildung 3.1: Interruptgesteuerter Programmablauf [scout1]

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 Hardwaredtimer zur Verfügung stehen, ist deren Funktionalität in einem Softwaredtimer nachgebildet. Dieser bezieht seinen Takt von dem Hardwaredtimer „Timer4“. Die Verwendung der Hard- sowie Softwaredtimer kann der Tabelle 3.1 entnommen werden.

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*.

Sensor 0 ist nach Tabelle 3.2 an PCINT16 angeschlossen. Aus dem Datenblatt

Tabelle 3.1: Belegung der Hard- sowie Softwaretimer

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 Softwaretimer
Softwaretimer	
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)

des Mikrocontrollers (ATmega2560) geht hervor, dass für die Pin Change Interrupts, drei Interruptvektoren vorgesehen sind. Dabei besteht eine Zuordnung von jeweils immer 8 PCINT Pins an einen Interruptvektor. Diese Zuordnung beginnt bei dem Pin PCINT0 und PCINT0vect. Durch das Überfahren des Autos über den Sensor, wird also das letzte Pin Change Interrupt, PCINT2_vect, zweimal ausgelöst. In der Interrupt Service Routine (ISR) muss nun unterschieden werden, durch welchen Pin, das Interrupt ausgelöst wurde. Die Lösung dieses Problems ist gegeben, indem der Signalpegel der PCINT Pins in der letzten ISR gesichert wurden. Nun wird das aktuelle Eingangsbyte des Ports mit dem zuvor gespeicherten bitweise exklusiv-oder verknüpft. Ist das Ergebnis der Verknüpfung nicht „0“, entspricht die Stelle der logischen „1“ im Byte, der Nummer des PCINT Pins gezählt, von dem ersten zugeordneten Interruptpin des Bytes. So ist das Ergebnis der exklusiv-oder Verknüpfung dieses Beispiels mit Sensor 0: 00000001 da sich das nullte Bit verändert hat. In der ISR muss nun weiter, zwischen den zwei Möglichkeiten, unterschieden werden:

- *High→Low* Übergang
- *Low→High* Übergang

Hat ein *High→Low* Übergang stattgefunden, wird ein Event erzeugt und je nach PCINT Pin unterschiedlich behandelt. Bei einem *Low→High* Übergang, kann das Ereignis verworfen werden.

Das Entprellen des Eingangs ist implementiert, indem nach dem Auslösen eines Events, die PCINT Funktion für den zugehörigen Pin deaktiviert wird, sodass dieser das Interrupt nicht mehr auslösen kann. Zusätzlich wird ein Softwaretimer gestartet, der die PCINT Funktion des Pins, nach dessen Ablauf wieder aktiviert. Mit diesem einfachen Prinzip wird sichergestellt, dass das Auto beim

Überfahren 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.

Tabelle 3.2: Belegung der Pin Change Interrupts

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

3.2.1.2.1 Unterschied: Pin Change Interrupt - Polling mit echtzeitfähigem System - Polling ohne echtzeitfähigem System wie in Abbildung 3.2 dargestellt, gibt es zwei Möglichkeiten, ein wert diskretes Signal mit den gültigen Zuständen High oder Low zu digitalisieren. Die erste Möglichkeit ist die zyklische Abfrage des Zustands. Dieses Verfahren wird in der Informatik Polling genannt. Bei Polling ist zu beachten, dass das Abtasttheorem eingehalten wird. Dieses besagt, es muss mit mindestens der doppelten Frequenz des abzutastenden Signals abgetastet werden, anderenfalls kommt es zu Aliasing und es ist nicht mehr möglich, das abgetastete Signal ohne Informationsverlust zu rekonstruieren. Da in einem nicht echtzeitfähigen Betriebssystem, wie zum Beispiel Raspian, nicht garantiert werden kann, dass die Signalabtastung lückenfrei, in äquidistanten Zeitabständen stattfindet, ist die Einhaltung des Abtasttheorems und damit das Vermeiden von Aliasingfehlern nicht gesichert. Ein solchen Fehler ist in Abbildung 3.2 mit der roten Datenreihe dargestellt. Hier ist einem anderen Prozess, innerhalb der zu detektierenden Signalfalte, Rechenzeit zugewiesen, sodass die Abtastung des Signals in dieser Zeit ausfällt. Genau dieses Verhalten führt zu dem in Abschnitt 1.1 beschrieben entsprechenden Problem, dass Sensoren nicht zuverlässig detektiert werden.

Eine Alternative zu Polling, stellt ein Abtasten der Signalfalten, durch eine Hardwareeinheit, wie sie in einem Atmega2560 integriert ist, dar. Diese Hardwareeinheit tastet nun in den Zeitabständen der Taktversorgung des Mikrocontrollers (in dieser Arbeit: 16MHz) das Signal ab, und löst bei jeder Änderung ein Hardwareinterrupt aus. Somit ist garantiert, dass die Flanken sicher erkannt

werden. In der vorliegenden Projektarbeit sind deshalb Pin Change Interrupts in Verwendung, zuvor war dies nicht der Fall.

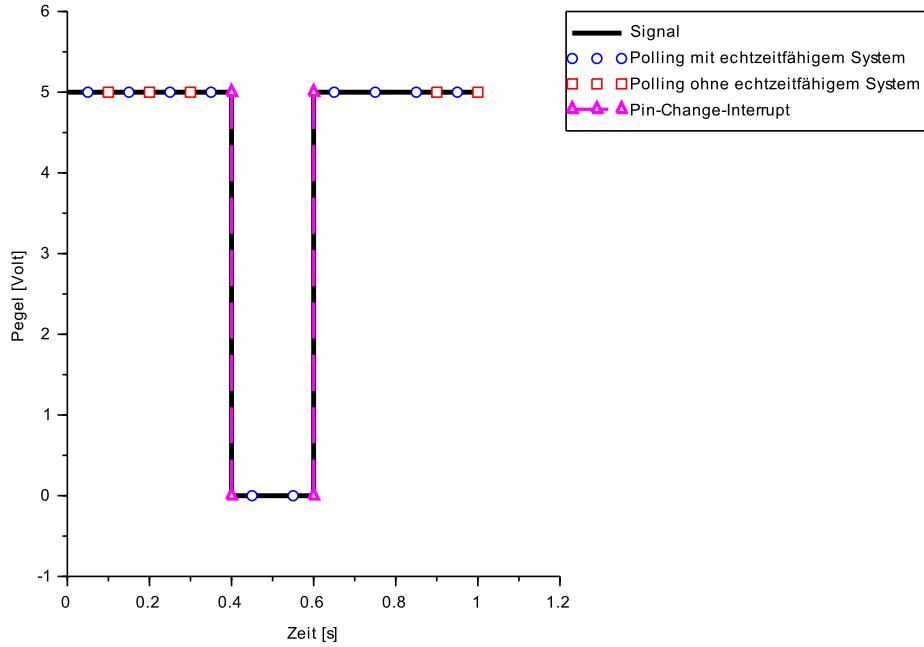


Abbildung 3.2: Abtastprozess eines analog vorliegenden wertdiskreten Signals durch Softwarepolling mit echtzeitfähigem System, Softwarepolling ohne echtzeitfähigem System sowie Hardware Pin Change Interrupts.

Die schwarze Linie stellt ein abzutastendes Beispieldesignal dar, wie es zum Beispiel von einem Bahnsensor zu erwarten wäre. Die blauen Kreise stellen die Stützstellen einer zyklischen Abtastung eines echtzeitfähigen Systems, mit der vierfachen Abtastrate wie die Grundfrequenz des Signals, dar und die roten Vierecke die Stützstellen bei einem nicht echtzeitfähigen System mit verletztem Abtasttheorem im Bereich der zu detektierenden Flanken.

3.2.1.3 Analog Digital Converter Interrupt

Zum Betrieb des Zustandsautomats (Abbildung 3.5) 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 Carrera Bahn hat zwei verschiedene Regelkreise. Der eine regelt die Bahnspannung (siehe Abschnitt 3.2.2.1) und ist in jedem Modus (manuell oder automatisiert) aktiv. Der andere Regelkreis (siehe Abschnitt 3.2.2.2) ist lediglich im

automatisierten Modus aktiv und regelt dort die Soll-Spannung auf den langsa-
men Streckenabschnitten (außerhalb des Loopings). Zusammen bilden die zwei
Regelkreise im automatisierten Modus eine Kaskadenregelung.

3.2.2.1 Bahnspannung

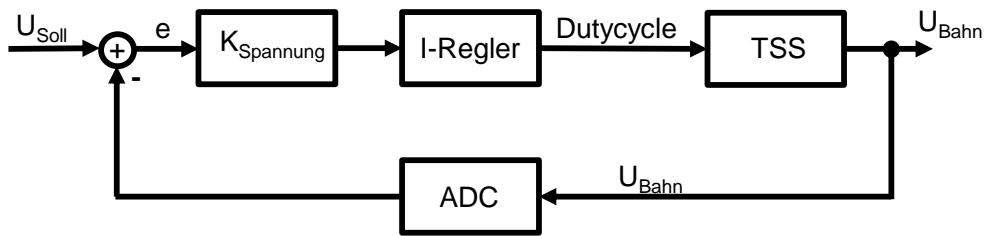


Abbildung 3.3: Regelkreis Bahnspannung.

Die Soll-Spannung wird je nach aktiven Modus der Bahn vorgegeben. Die Ist-Spannung der jeweiligen Bahn wird mit dem ADC gemessen und die Regelabweichung errechnet. Regelabweichung wird mit dem konstanten Faktor $K_{Spannung}$ verstärkt und auf den I-Regler gegeben. Dieser erzeugt Den Duty-
cycle der auf den TSS gegeben wird. TSS erzeugt Bahnspannung.

Um Schwankungen in der Energieversorgung auszugleichen, sowie das Steuern der Bahnspannung zu ermöglichen, ist eine Spannungsregelung nach Abbildung 3.3 implementiert. Die Ist-Spannung der jeweiligen Bahn wird zyklisch mittels des ADCs gemessen. Jedes mal wenn ein neuer Wert der Bahnspannung von Bahn-A vorliegt, wird die Funktion *i_control_voltage_a* (siehe Quellcode) aufgerufen. Die Funktion kann nun die Regeldifferenz „e“ bilden und diese mit dem Faktor „ $K_{Spannung}$ “ verstärken. Der I-Regler (einfacher digitaler Integrator mit der Differenzengleichung $y[k] = y[k - 1] + x[k]$) integriert diese verstärkte Regelabweichung auf und generiert damit den DutyCycle für den TSS, bei dem sich dann eine neue Ausgangsspannung (U_{Bahn}) entsprechend des neuen Duty-
cycle einstellt. Diese neue Spannung wird wieder gemessen und die Funktion wird erneut aufgerufen. Als Regler ist ein digitaler I-Regler in Verwendung, da Die Strecke (TSS) kein freier Integrator besitzt. Jedoch braucht ein Regel-
kreis mindestens ein freien Integrator um stationäre Genauigkeit aufzuweisen. Zusätzlich zu dem gerade eben beschriebenen Regelalgorithmus, ist die Aus-
gangsgröße des I-Reglers nach oben (maximal 200/255), sowie unten (minimal 10/255) begrenzt. Dies ist notwendig da der TSS jenseits eines DutyCycle von 200, bei einer weiteren Erhöhung, nicht mehr eine höhere Ausgangsspannung liefert und das Auto sich unter einem DutyCycle von 10 noch nicht bewegt. Diese Grenzen wurden empirisch ermittelt und beziehen sich auf die Quelle mit dem höchsten Innenwiderstand (PV).

3.2.2.2 Bahngeschwindigkeit

Da die Carrera Bahn mit verschiedenen Fahrzeugen betrieben werden kann und somit die Modellparameter nicht konstant sind (zum Beispiel Reibungs-

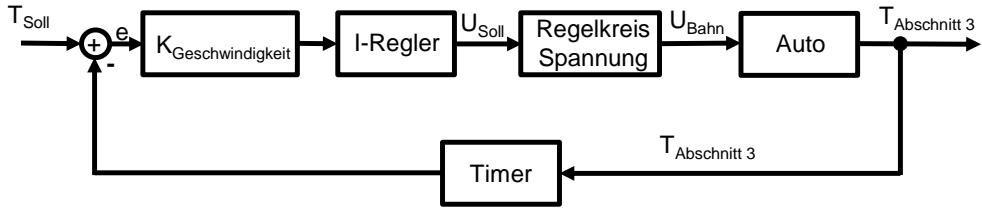


Abbildung 3.4: Regelkreis Bahngeschwindigkeit.

Die Sollzeit ist als konstanter Parameter vorgegeben. Die Zeit in Streckenabschnitt 3 (zwischen dem Ende des Loopings und der Startlinie) wird mit einem Softwaretimer gemessen und mit der Sollzeit verglichen. Die resultierende Regelabweichung wird mit dem konstanten Faktor $K_{Geschwindigkeit}$ verstrtzt und auf den I-Regler gegeben. Dieser erzeugt daraus die Soll-Spannung fr die jeweilige Bahn und gibt diese an den inneren Regelkreis (Regelkreis Spannung, Abbildung 3.3) weiter. Dieser generiert daraus die Ist-Spannung die am Auto anliegt. Daraus ergibt sich eine neue Zeit fr den Streckenabschnitt 3.

verluste im Fahrzeug, allgemein Motor), ist es notwendig die Soll-Spannung im automatisierten Modus auf den langsamen Streckenabschnitten anzupassen. Hierzu ist ein Regler nach Abbildung 3.4 implementiert. Dabei wird die Zeit gemessen, welche das Auto bentigt, um die Strecke zwischen dem Sensor nach dem Looping bis zum Sensor an der Startlinie zu fahren. Fr diese Zeit ist im Arduino eine konstante Soll-Zeit hinterlegt. Jedes mal wenn ein neuer Messwert fr die Zeit auf diesem Streckenabschnitt vorliegt, wird die Funktion *i_control_railspeed_a* (siehe Quellcode) aufgerufen und darin die Regeldifferenz „e“ bilden. Analog zu dem Regelkreis in Abschnitt 3.2.2.1, wird auch hier die Regeldifferenz gebildet und mit dem Faktor $K_{Geschwindigkeit}$ skaliert. Als Regler kommt ein I-Regler wie in Abschnitt 3.2.2.1 zum Einsatz. Die so erzeugte Stellgre U_{Soll} wird nun auf das Stellglied (innerer Regelkreis Bahnspannung aus Abschnitt 3.2.2.1) gegeben und es ergibt sich durch eine neue Bahnspannung auch eine neue Zeit fr den gemessenen Streckenabschnitt. In der nchsten Runde wiederholt sich der Kreis.

Um Fehler die durch kurzzeitige externe Strungen verursacht werden abzufangen hat man die Anderung von U_{Soll} begrenzt. Solche Strungen wren zum Beispiel wenn das Auto in dem Streckenabschnitt zwischen dem Looping und der Startlinie festgehalten wird oder herausfllt. Dies frt dazu, dass das Auto ein paar Runden mehr braucht, um seinen stationren Endwert in der Geschwindigkeit zu finden, allerdings macht es die Regelung insgesamt robuster. Der in diesem Abschnitt beschriebene Regelalgorithmus ist ausschlielich im automatisierten Modus einer Bahn aktiv. Durch ein Wechsel des Modus aus dem Modus Auto-Run (sie Abschnitt 5.2.2) in einen anderen Modus, bleibt der aktuelle Wert des Integrators erhalten, sodass das Auto beim nchsten mal, vorausgesetzt das Auto ist dasselbe, mit der richtigen Geschwindigkeit startet.

3.2.3 Analog Digital Wandler

Der Mikrocontroller des Arduinos (Atmega2560) hat bereits einen 10-bit Analog Digital Converter (ADC) auf dem Chip integriert, sodass auf einen externen Wandler verzichtet wird. Da der Wandler immer nur eine Messung durchführen kann, hat der Hersteller einen Multiplexer (MUX) integriert, um zwischen den einzelnen ADC Pins (Kanälen), des Mikrocontrollers durchzuschalten. Die Belegung der ADC Pins ist in Tabelle 8.3 enthalten.

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

Die Steuerung des MUX erfolgt über ein Zustandsautomat (siehe Abbildung 3.5), der immer dann ein Schritt weiter springt, wenn dieses Interrupt eintritt. In jedem Schritt des Automats, wird schematisch dasselbe durchgeführt:

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

3.2.4 PWM Generierung Tiefsetzsteller

Wie aus Abbildung 3.6b 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 Hardwartertimer 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 3.6a als Zustandsautomat dargestellt.

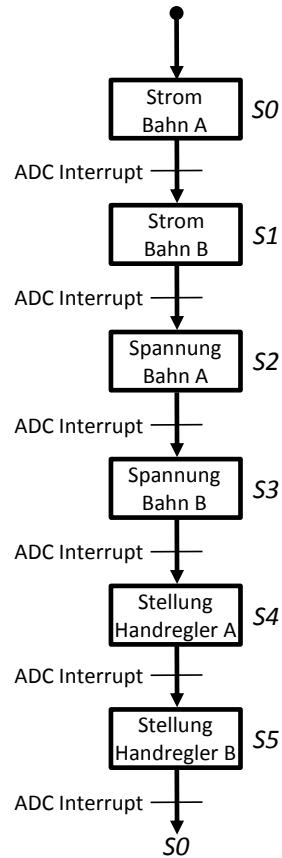
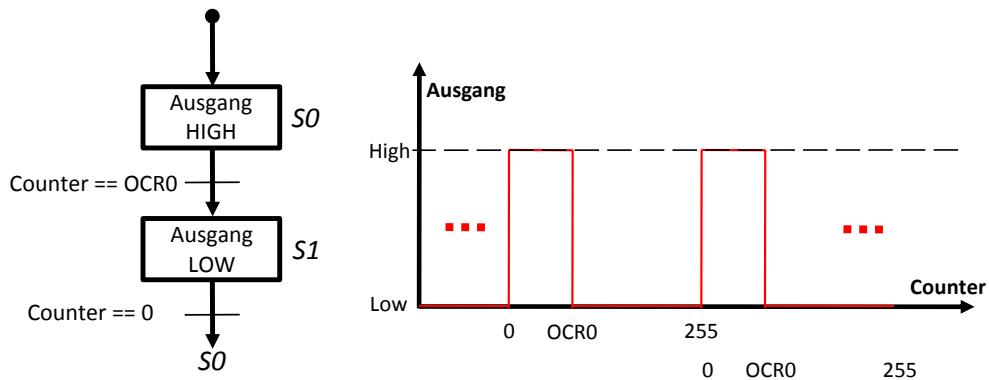


Abbildung 3.5: Zustandsautomat des MUX im ADC.

Die Zustände(S0 bis S1) beschreiben, welcher ADC Pin des Mikrocontrollers gerade durch den MUX mit dem ADC verbunden ist. Die Transitionsbedingung (links) beschreibt das Ereignis, mit dem man von einem Zustand in den Nächsten wechselt. Ein Punkt markiert den Schritt in den Initialzustand (S0) des Automats und ein Pfeil einen Sprung zu einem darunter definierten Zustand.



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

Abbildung 3.6: Zustandsautomat des Timers zur PWM Generierung, sowie resultierendes Ausgangssignal bei konstantem Duty cycle.

Kapitel 4

Bekannte Störungen

Gelegentlich fallen Autos im Looping herunter. Durch längere Tests und Analyse der Fehlerbilder, hat sich herausgestellt, dass dies kein Problem der Regelung selbst darstellt, sondern vielmehr der schlechten Qualität der zugekauften Carrera Bahn. Genauer ist die Stromschiene der Bahn im Looping ein wenig in der Fahrbahnoberfläche versenkt und der Übergang zwischen den zusammengesteckten Streckenstücken weist einen unstetigen Verlauf auf. Dadurch ergibt sich eine Diskontinuität des Stroms, durch den Antrieb des Fahrzeugs im Looping. Dies führt zu Geschwindigkeitsverlust, sowie letztendlich zum Absturz des Fahrzeugs. Werden nun, um die schlechte mechanische Beschaffenheit auszugleichen, die Strombürsten des Fahrzeugs weiter in Richtung Schiene gebogen, taucht die Führungsnase nicht mehr weit genug in die Bahn ein und das Fahrzeug fliegt aus der Kurve oder triggert die Sensoren nicht mehr verlässlich.

Kapitel 5

Bedienungsanleitung

5.1 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 Carrera Bahn betriebsbereit und im Modus manuelles Fahren (Abschnitt 5.2.1).

5.2 Betrieb

Die Carrera Bahn beherrscht 2 Modi.

- Manuelles Fahren
- Automatisiertes Fahren

Die Modi können unabhängig voneinander auf beiden Bahnen gewählt werden, sodass es zum Beispiel möglich ist, manuell gegen ein automatisiertes Fahrzeug anzutreten. Die Wahl eines dementsprechenden Modus (automatisiert oder manuell), 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 zwei Streckenabschnitte, sowie die Rundenzeit angezeigt. 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. Nach 30 Runden ist das Rennen vorbei und die Visualisierung wird statisch. Der Wechsel zwischen den Modi findet nach Abbildung 5.1 statt und ist in den jeweiligen Abschnitten genauer erklärt.

5.2.1 Manuelles Fahren

Im Modus Manuelles Fahren kann das Fahrzeug konventionell, über die originalen Carrera Handregler, gesteuert werden. Durch Betätigung des HMI-Tasters Auto, kann die Bahn in den Auto-Wait Zustand versetzt werden.

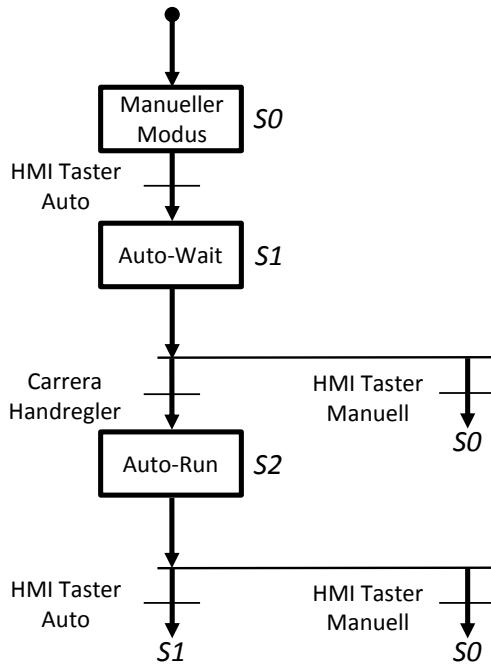


Abbildung 5.1: Zustandsautomat zum Wechsel der Betriebsmodi einer Bahn. Durch den „HMI Taster Auto“ der jeweiligen Bahn, kann der Nutzer immer in den „Auto-Wait“ Zustand (S_1) wechseln, in dem das Auto hält. Von diesem Zustand aus, kann der Nutzer die Bahn durch Betätigung des Handreglers, in den „Auto-Run“Zustand (S_2) versetzen und das Auto fährt anschließend selbstständig. Aus jedem Zustand kann der Nutzer mit dem „HMI Taster Manuell“, die Bahn in den manuellen Modus versetzen und das entsprechende Auto mit dem Handregler konventionell steuern.

5.2.2 Automatisiertes Fahren

Der automatisierte Modus ist aufgeschlüsselt in zwei Zustände:

- Auto-Wait
- Auto-Run

5.2.2.1 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.

5.2.2.2 Auto-Run

Im Auto-Run Zustand fährt das Auto selbstständig. Durch die Regelung der Geschwindigkeit des langsamen Streckenabschnitts, braucht das Auto einige Runden um die optimale Spannung, für diese Geschwindigkeit zu finden. Dies ist genauer in Abschnitt 3.2.2.2 beschrieben. Es ist darauf zu achten, dass wenn das Auto im Looping herunterfällt, es wieder an der Startlinie eingesetzt wird. Ist

dies nicht der Fall, ist weiterhin die erhöhte Geschwindigkeit aus dem Looping aktiv und das Auto verlässt in der nächsten Kurve erneut die Bahn.

5.3 Ausschalten

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

Kapitel 6

Reflexion

Die zu Beginn, in Abschnitt 1.2 definierten Probleme der vorhandenen Carrera Bahn, sind durch das neue Steuer-/Regelkonzept, sowie eine neue Hardware Architektur, fast alle beseitigt.

Da nun unabhängig der gewählten Energiequelle, die Bahnnspannung immer durch die Tiefsetzsteller erzeugt und geregelt wird, ist diese nicht mehr von der eigentlichen Quelle abhängig. Somit stellt nun auch die Solarenergieversorgung eine stabilisierte Spannungsquelle dar (siehe Abschnitt 3.2.2.1).

Der Temperaturdrift, des in der vorherigen Bahn eingesetzten variablen Widerstands, ist behoben, da dieser in der neuen Anlage, durch die Möglichkeit die Bahnnspannung durch den Dutycycle der Tiefsetzsteller anzupassen, nicht mehr benötigt wird. (siehe Abbildung 2.1)

Die Framerate der Visualisierung wurde angehoben, indem immer nur der dynamische Bildinhalt (die Rundenzeiten) neu gerendert wird (siehe Abschnitt 3.1). Dies reduziert die Rechenzeit pro Frame.

Die Probleme mit Aliasing beim Auslesen der Sensoren sind durch den Einsatz der „Pin Change Interrupts“ des Arduinos zum Digitalisieren der Sensorsignale vollständig behoben (siehe Abschnitt 3.2.1.2).

Ein bestehendes Problem der Carrera Bahn ist, das gelegentlich ein Auto im Looping herunterfällt, wenn dieses automatisch fährt. Die Ursache dessen wurde identifiziert (siehe Abschnitt 4), kann allerdings nicht mit vertretbarem Aufwand gelöst werden.

Insgesamt wird die Projektarbeit vom Autor als Erfolg angesehen.

Kapitel 7

Quellenverzeichnis

Scout1. (September 2004). Interrupt Programme.gif – Mikrocontroller.net.

Von Mikrocontroller.net: https://www.mikrocontroller.net/articles/Datei:Interrupt_Programme.gif abgerufen

Kapitel 8

Anhang

Tabelle 8.1: Pinbelegung der Sub-D Buchse auf dem HMI

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

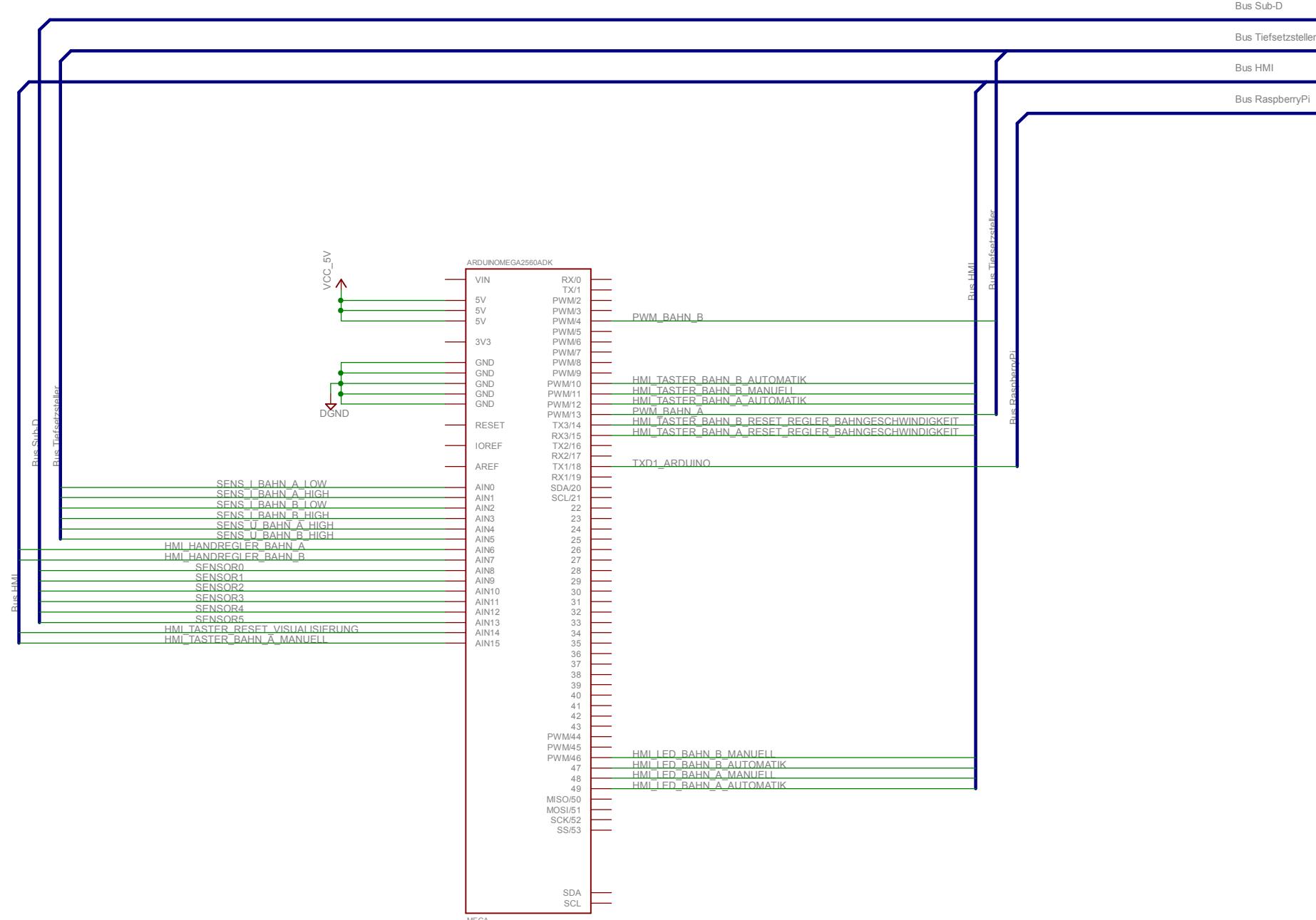
Tabelle 8.2: Pinbelegung GPIO RaspberryPi

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

Tabelle 8.3: Pinbelegung des Arduino

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		Serielle Verbindung zu RaspberryPi
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“

1 2 3 4 5 6 7 8



1 2 3 4 5 6 7 8

	CarreraBahn
	28.08.2018 14:54:37
	Sheet: 1 / 4
Florian Weber 44907	

1 2 3 4 5 6 7 8

Bus Sub-D

Bus Tiefsetzsteller

Bus Sub-D

Bus Tiefsetzsteller

Bus HMI

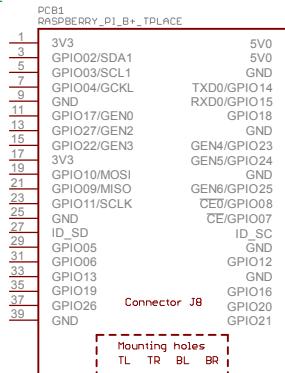
Bus HMI

Bus RaspberryPi

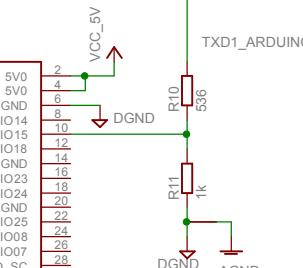
Bus Energievorsorgung

A

Raspberry Pi hat ein Steckernetzteil mit mikro-USB.
mit dem es VCC_5V und DGND bereitstellt



Spannungsausgang



5

6

7

8

Bus Energievorsorgung

Bus Energievorsorgung

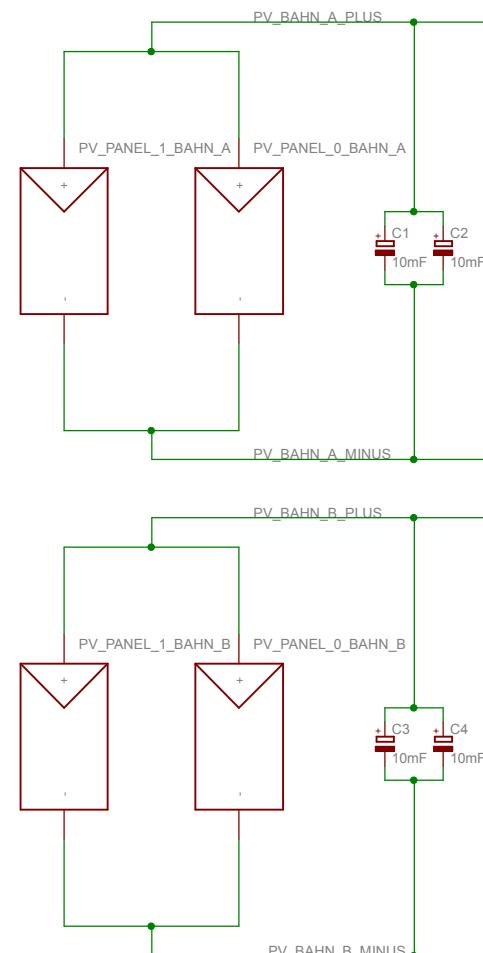
A

B

C

D

E



Florian Weber 44907	CarreraBahn
	28.08.2018 14:54:37
	Sheet: 2/4

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

Bus Sub-D

Bus Tiefsetzsteller

Bus Sub-D

Bus Tiefsetzsteller

Bus HMI

Bus Energieversorgung

WECHSELSCHALTER_NETZPV_BAHN_A

NETZ_BAHN_A_PLUS S0T
PV_BAHN_A_PLUS S0B
NETZ_BAHN_A_MINUS S1T
PV_BAHN_A_MINUS S1B

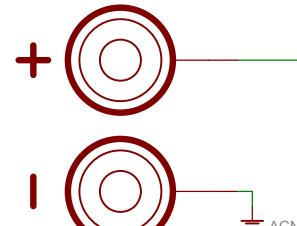
S0 POWER_IN_TIEFSETZSTELLER_BAHN_A_PLUS
S1 POWER_IN_TIEFSETZSTELLER_BAHN_A_MINUS

WECHSELSCHALTER_NETZPV_BAHN_B

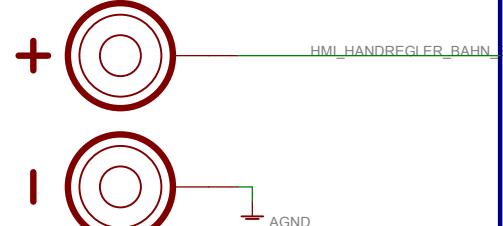
NETZ_BAHN_B_PLUS S0T
PV_BAHN_B_PLUS S0B
NETZ_BAHN_B_MINUS S1T
PV_BAHN_B_MINUS S1B

S0 POWER_IN_TIEFSETZSTELLER_BAHN_B_PLUS
S1 POWER_IN_TIEFSETZSTELLER_BAHN_B_MINUS

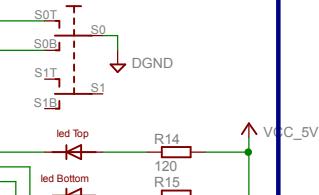
HANDEGLER_BAHN_A



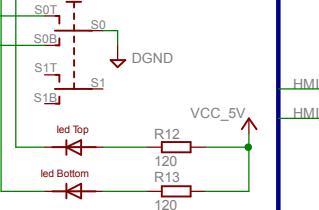
US2

HMI_TASTER_BAHN_A_MANUELL
HMI_TASTER_BAHN_A_AUTOMATIKHMI_LED_BAHN_A_MANUELL
HMI_LED_BAHN_A_AUTOMATIKHMI_LED_BAHN_B_MANUELL
HMI_LED_BAHN_B_AUTOMATIKHMI_TASTER_BAHN_B_MANUELL
HMI_TASTER_BAHN_B_AUTOMATIK

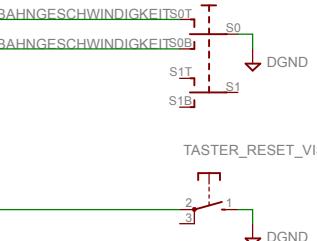
WECHSELTASTER_MODI_BAHN_A



WECHSELTASTER_MODI_BAHN_B



WECHSELTASTER_RESET_REGLER_BAHN_A_UND_B



HMI_TASTER_BAHN_A_MANUELL

HMI_TASTER_BAHN_A_AUTOMATIK

HMI_LED_BAHN_R_MANUELL

HMI_LED_BAHN_B_AUTOMATIK

HMI_TASTER_BAHN_A_RESET_REGLER_BAENGESCHWINDIGKEIT

HMI_TASTER_BAHN_B_RESET_REGLER_BAENGESCHWINDIGKEIT

HMI_TASTER_RESET_VISUALISIERUNG



Florian Weber 44907

CarreraBahn
28.08.2018 14:54:37
Sheet: 3/4

1 2 3 4 5 6 7 8

1 2 3 4 5 6 7 8

Bus Sub-D

Bus Tiefsetzsteller

A

B

C

D

E

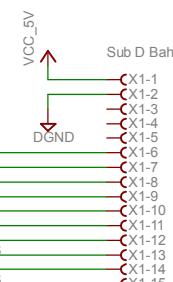
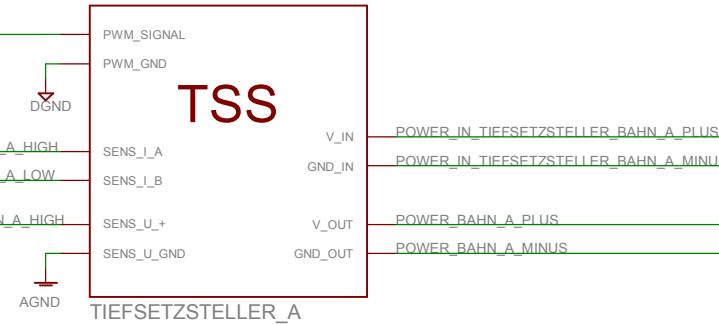
A

B

C

D

E



1 2 3 4 5 6 7 8

CarreraBahn
28.08.2018 14:54:37
Sheet: 4 / 4