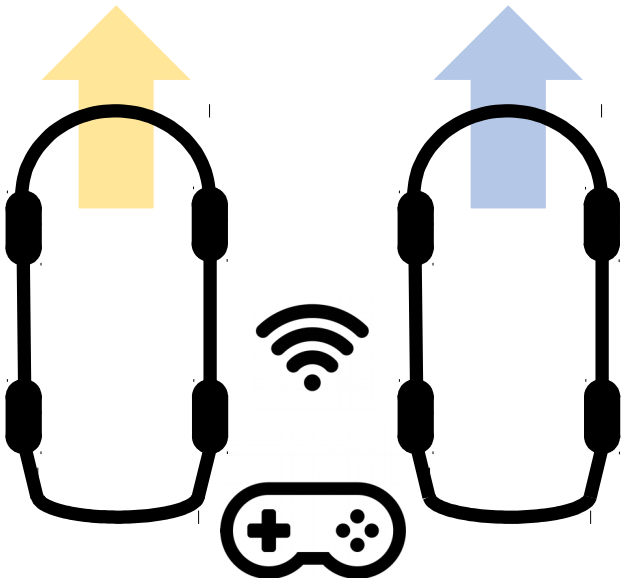
SYNC-PARTY

Automatisches synchrones Fahren mehrerer Modellautos



Inhalt

- Hardware
- Software
- Netzwerk
- Synchronisierung
- Kamera

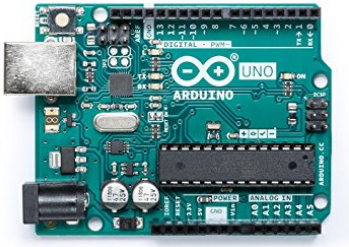
Hardware

Simon Hölzl

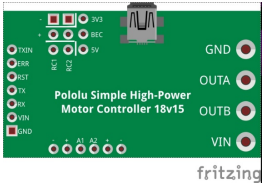
µController

- Raspberry Pi 3 Model B

- Arduino Uno Rev 3



- Pololu Simple High-Power Motor Controller 18v15



Controller

- Playstation DUALSHOCK4 Wireless Controller

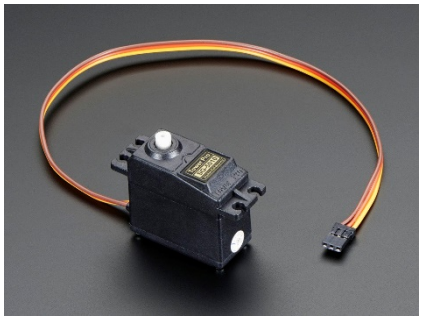
Sensoren

- Parallax INF PING))) Ultrasonic Distance Sensor

- Logitech Webcam C300

Aktoren

- TowerPro SG-5010 Servo

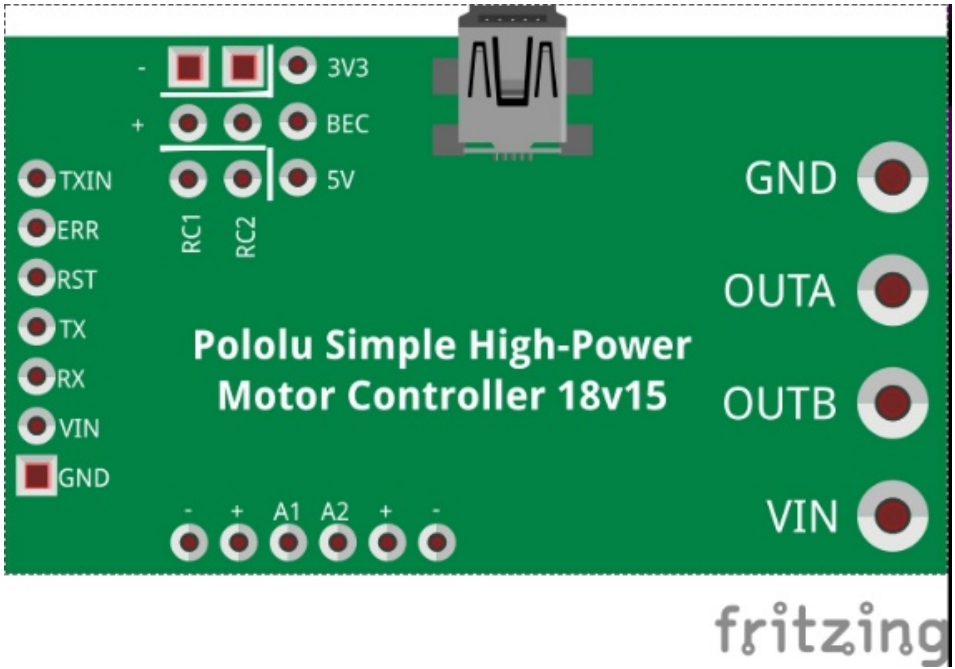


- Brushed DC Motor (Chassis)

Stromversorgung

- Power Bank
- Nickel-Metall-Hybrid Akku

Pololu Motor Controller 18v15

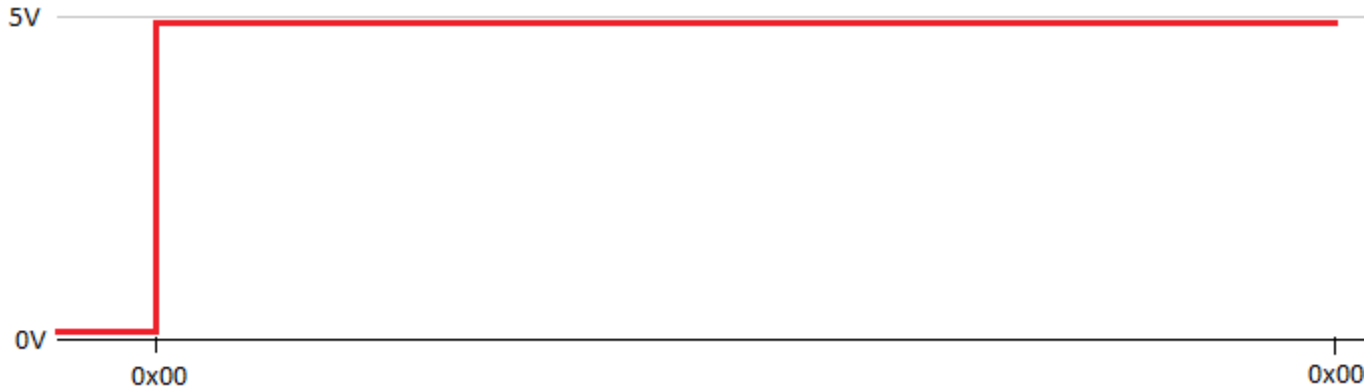


Pololu Motor Controller 18v15 - Kalibrierung

Motor Stop: High Anteil 18.75%

Pololu Motor Controller 18v15 - Kalibrierung

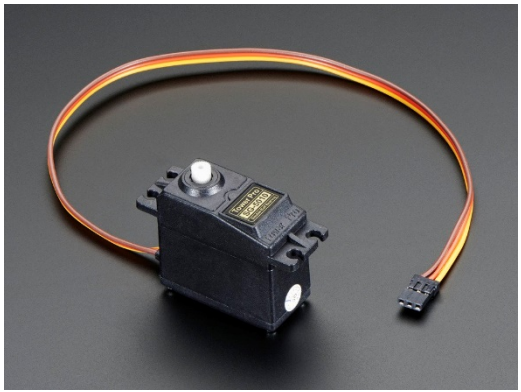
Motor Full Forwards: High Anteil ~100%



Pololu Motor Controller 18v15 - Kalibrierung

Motor Full Backwards: High Anteil 6,25%

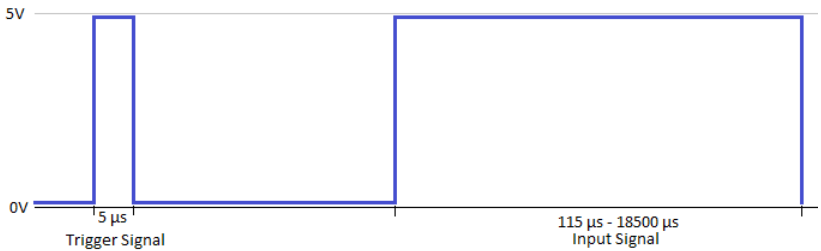
PowerPro SG-5010 Servo



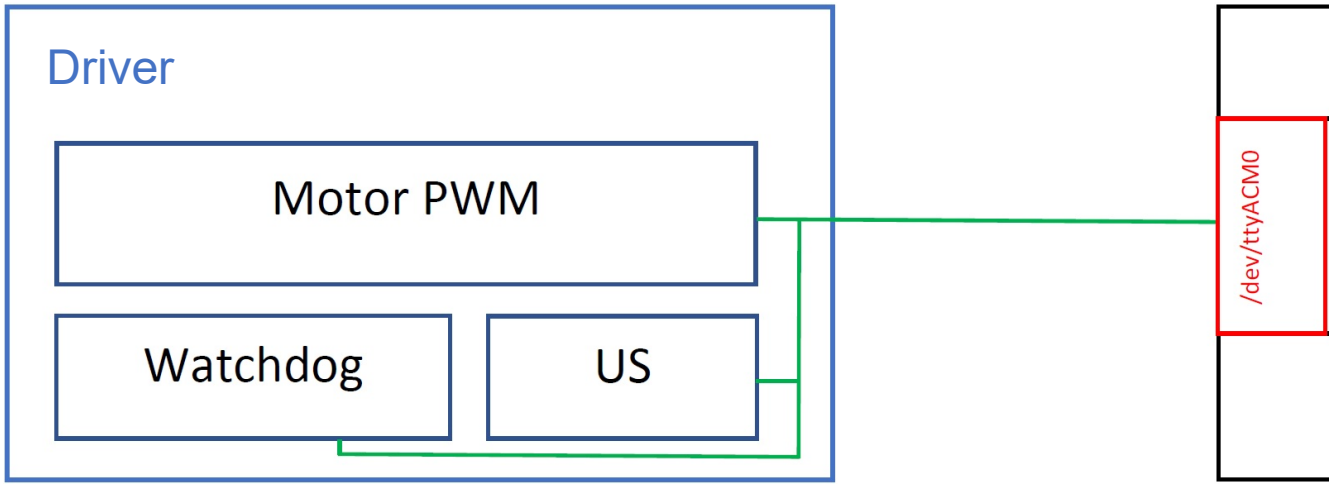
- Steuerung über PWM Signal
- Periodenlänge: 20 ms (50 Hz)
- Tastgrad (High-Anteil): ~1 ms (-90°) – ~2 ms (90°)
- Mit Reifen: 1,2 ms (-54°) – 1,8 ms (54°)

Parallax INF PING))) Ultrasonic Distance Sensor

- Reichweite: 2 cm – 3 m
- Elektrische Eigenschaften: 5V, 35mA
- Eine Leitung für Trigger Signal und Input Signal (SIG)



Arduino Uno Rev 3 - Software



Arduino Uno Rev 3 – PWM Signal

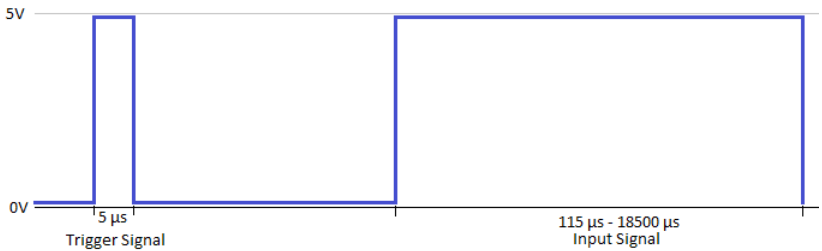
- Ausgabe am Pin 11
- Periodenlänge: 4 ms
- Timer 2, FastPWM, Clear Pin 11 on Compare Match
- Setzen der Compare Werte durch den Compare Interrupt

Arduino Uno Rev 3 - Ultraschall

- Ausgabe/Eingabe am Pin 7

- Erzeugung des Trigger Signals

```
// start
pinMode(pin, OUTPUT);
digitalWrite(pin, LOW);
delayMicroseconds(2);
digitalWrite(pin, HIGH);
delayMicroseconds(5);
digitalWrite(pin, LOW);
```



- Messen des Input Signals

```
// wait for echo
pinMode(pin, INPUT);
unsigned long dur = pulseIn(pin, HIGH, PING_TIMEOUT_MIS);
```

- Umrechnung der Zeit in Distanz

```
return dur*10 / 60;
```

Arduino Uno Rev 3 - Watchdog

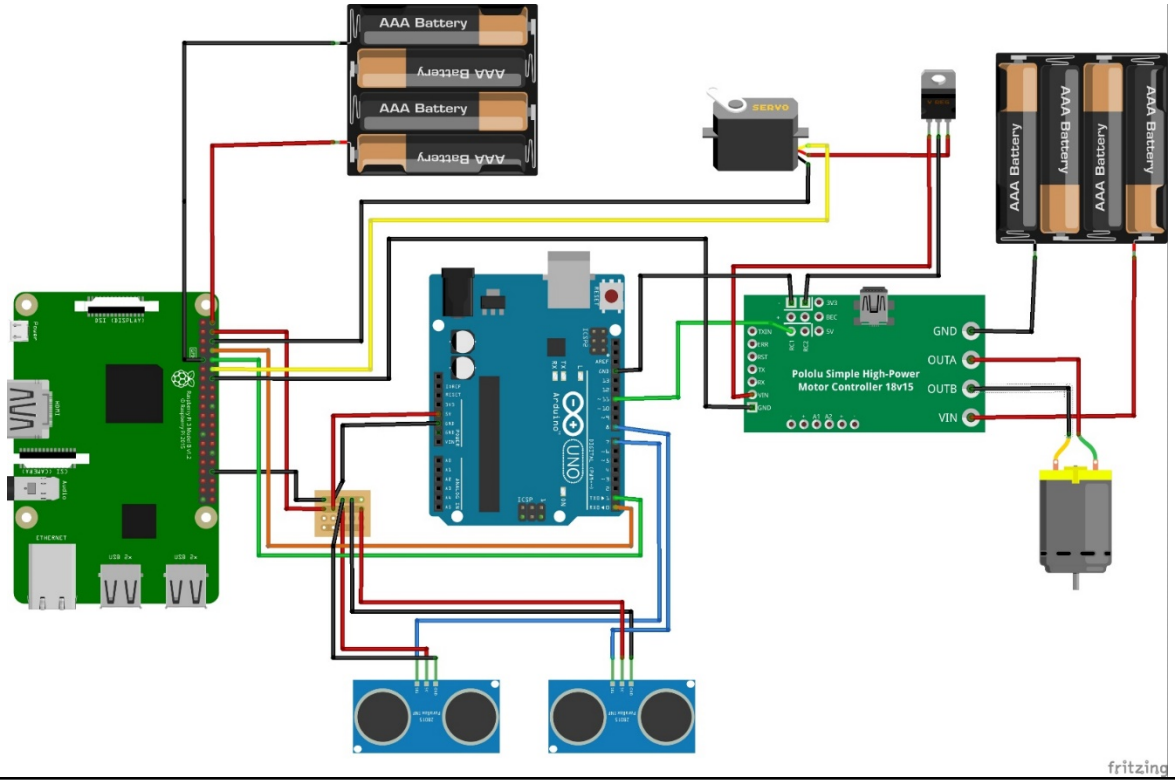
- Überprüfen, ob das Programm auf den Raspberry Pi noch läuft
- Periodisches Senden von Nachrichten über die serielle Schnittstelle
- Warten bis Antwort kommt
- Bei fehlender Antwort innerhalb 200 ms, Motor auf Stop schalten

Playstation DUALSHOCK4 Wireless Controller

- Verbindung über Bluetooth mit Master Raspberry Pi
- Steuerung der Geschwindigkeit über R2 (vorwärts) und L2 (rückwärts)

- Steuerung der Fahrtrichtung (links/rechts) über den linken Joystick

Schematischer Aufbau

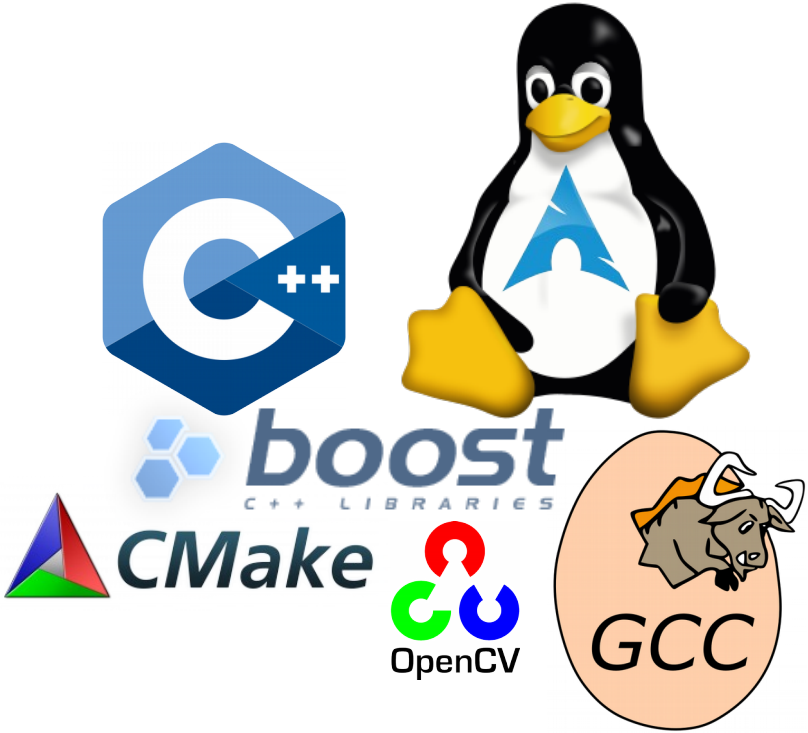


Software

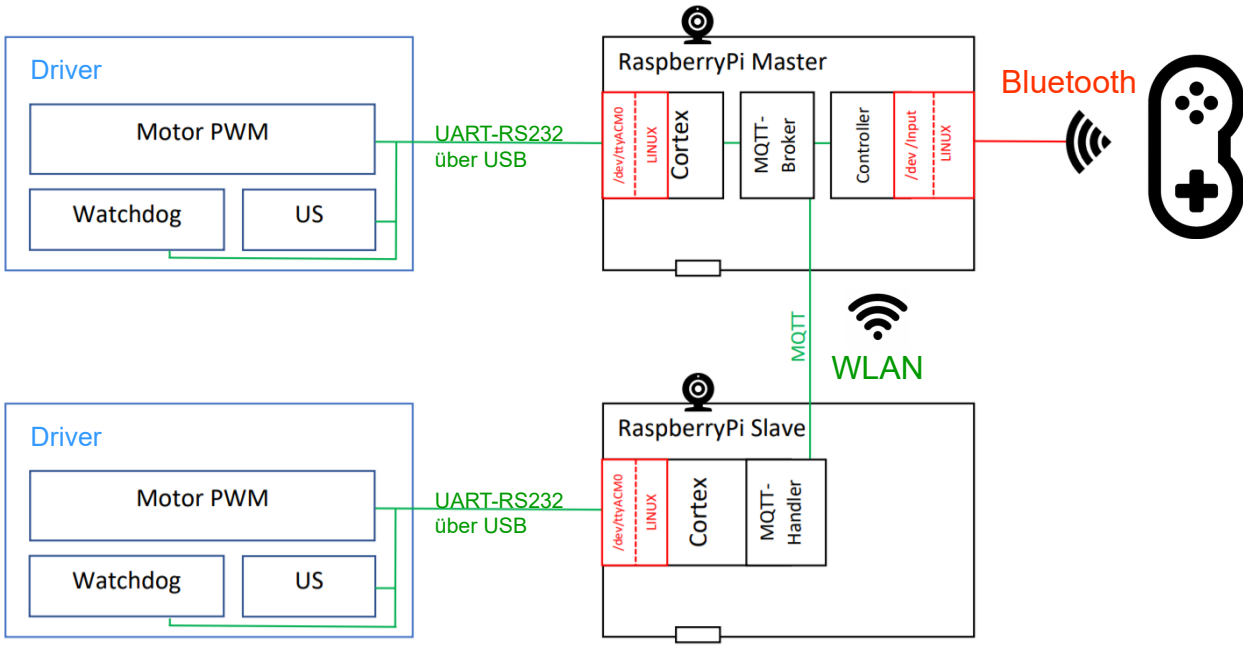
Andrej Utz

Auswahl

- OS: Linux 4.14
- Programmbasis
  - C++
  - Boost.Asio
  - mqtt_cpp
  - OpenCV
- Buildsystem
  - CMake
  - GCC

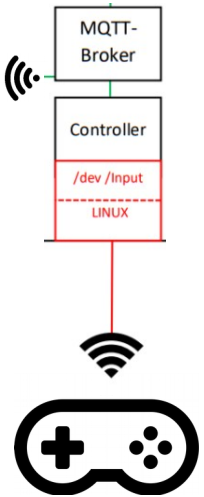


Übersicht

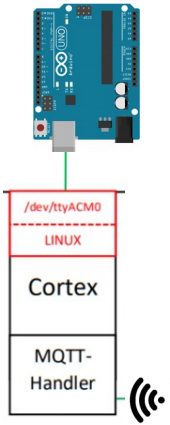

Driver
Motor PWM
Watchdog
US
UART-RS232
über USB
RaspberryPi Master
/dev/ttyACM0
LINUX
Cortex
MQTT-
Broker
Controller
/dev /input
LINUX
Bluetooth
MQTT
WLAN
Driver
Motor PWM
Watchdog
US
UART-RS232
über USB
RaspberryPi Slave
/dev/ttyACM0
LINUX
Cortex
MQTT-
Handler

Komponenten – Controller

- Läuft nur beim Master
- Warten auf Eingabe-Events vom Kernel
  - Lesen von `/dev/input/js0` – `fertig!`
    - Linux erleichtert Zugriff auf Eingaben enorm
    - Treiber für PS4 Controller bereits im Kernel
    - Bluetooth komplett abstrahiert
  - Trigger und Sticks am Controller werden Achsen genannt – 8 Achsen, 12 Knöpfe
  - Achsenwerte in Bereich von `[-32767;+32767]`
- Weiterleitung an MQTT-Broker
  - nur Wertänderungen!
  - Skalierung auf `[-64;+64]`: weniger Genauigkeit – weniger zum Senden

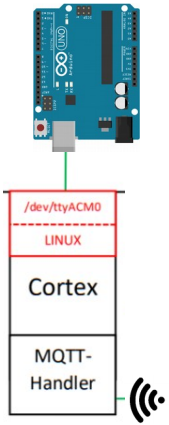


Komponenten – Cortex



- Fahrsteuerung – Synchronisierung, Lenkung, Beschleunigung
- Warten auf MQTT-Pakete
- Lenkung über RPi Hardware-PWM
  - sysfs-Interface in Linux: `/sys/class/pwm/pwm0/`
    - `/period` – Periodendauer, für Servos 20.000.000 ns
    - `/duty_cycle` – Tastgrad, [500.000; 2.500.000] ns
    - muss in `/boot/config.txt` aktiviert werden!
- Beschleunigung, Ultraschall, Akkustand übernimmt Driver
- Synchronisation
- Tracking mit Kameras
  - Linux hilft wieder: V4L2-Treiber abstrahieren die Kamera zu `/dev/video0`

Komponenten – Cortex



- Driver-Kommunikation mit einfachem Befehlsprotokoll
  - Anfrage: `[<Befehl><Parameter>] (4 Bytes)`
  - Antwort:
    - OK: `[<gleicher Befehl>,<Wert>]`
    - Error: `[MSB gesetzt + <gleicher Befehl><Wert>]`
  - Spiegeln der Befehle hilft festzustellen, was schon abgearbeitet wurde
  - Beispiel: Lese Ultraschall auf Pin 7 des Driver `[0x3,0x7]` → `[0x3, 0xF]` – 15 cm Distanz

Netzwerk

Adrian Leher

Physische Verbindung

- Wlan Verbindung

- Master als Access Point

- Slaves haben keinen Internetzugriff

Vorteil: Autos überall einsetzbar

hostapd

- Daemon für Access Points

- IEEE 802.11 AP Management

- Konfiguration mittels Textfile

hostapd.conf

```
ssid=Sync-Party
interface=wlan0
driver=nl80211
channel=1
ignore_broadcast_ssid=0
ap_isolate=0
hw_mode=g
wpa=3
wpa_passphrase=0987654321
wpa_key_mgmt=WPA-PSK
wpa-pairwise=TKIP CCMP
rsn_pairwise=CCMP
```

dnsmasq

- leichtgewichtiger DHCP Server

- Domain Namespace System

-> Kann Namen für lokales Netz stellen

- geringe Anforderungen an System Ressourcen

MQTT

- Message Queueing Telemetry Transport

- „Protokoll des IoT“

- basiert auf Publish und Subscribe Prinzip

- Entkopplung Sender, Empfänger

MQTT Vorteil

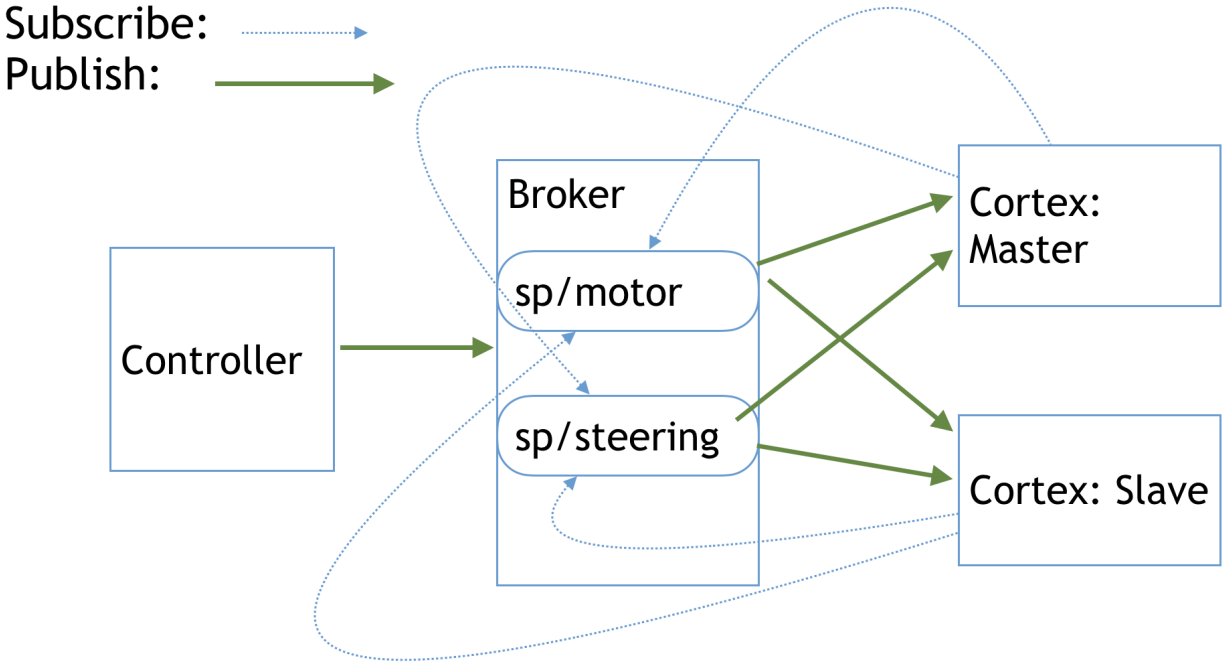
- leichtgewichtiges Protokoll

- gut wenn wenig Daten übertragen werden müssen

- geringer Ressourcen Verbrauch

- gut für one to many Verbindungen

Subscribe	Publish: sp/motor	Publish: sp
sp/#	Empfang	Empfang
sp/motor	Empfang	Kein Empfang
sp/steering	Kein Empfang	Kein Empfang
motor/sp	Kein Empfang	Kein Empfang


Subscribe:
Publish:
Broker
sp/motor
sp/steering
Controller
Cortex: Master
Cortex: Slave

Vision Swarm

- Verbinden mehrerer Autos mittels :
  sp/slave1 ... sp/slaveX

- Anbindung der Sensoren:

sp/slaveX/Ultraschall

sp/slaveX/Camera

Synchronisierung

Tran Luu Ngoc

Synchrones Abbiegen

Beispiel Linksabbiegen

Anpassung der Geschwindigkeit

Mit Hilfe von der Kamera

Anpassung der Distanz zueinander

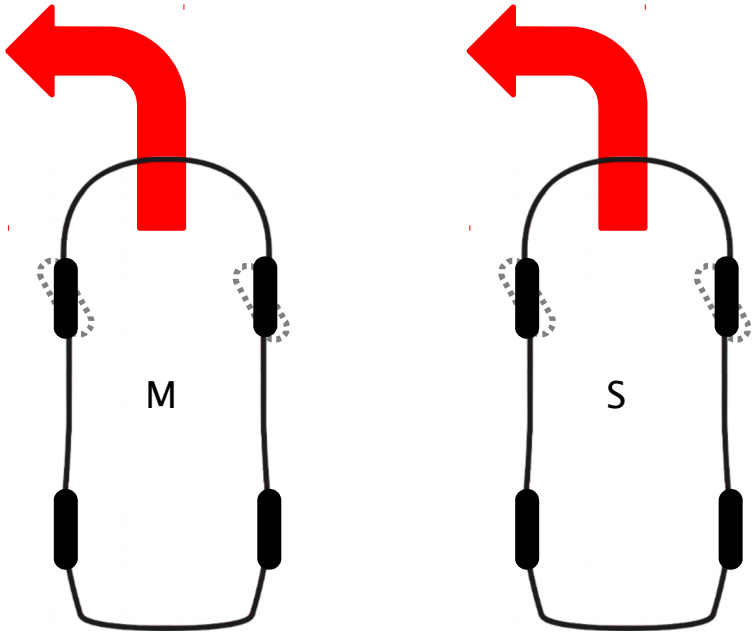
Mit Hilfe des Ultraschalls

Synchrones Abbiegen
Beispiel Linksabbiegen

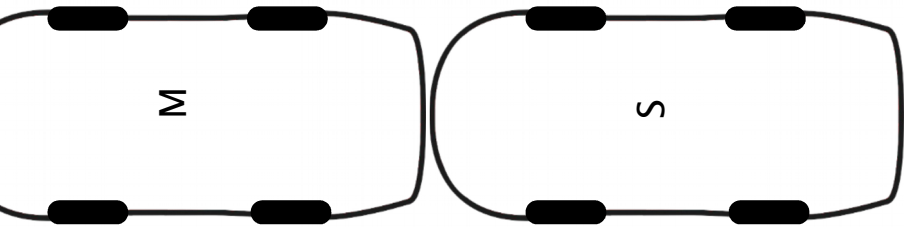
Anpassung der Geschwindigkeit
Mit Hilfe von der Kamera

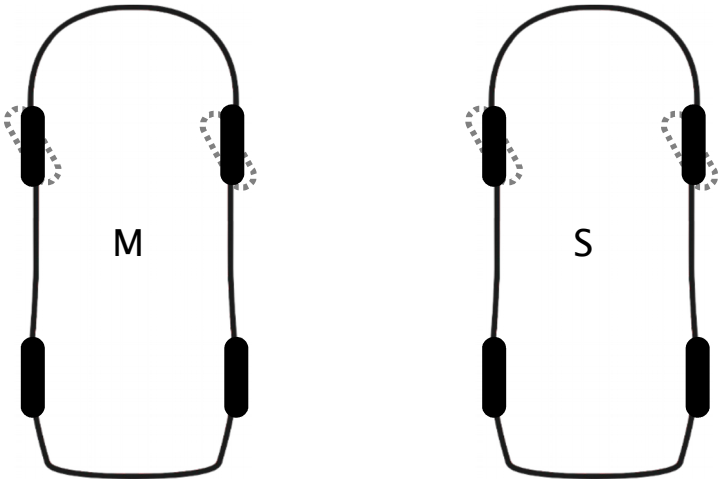
Anpassung der Distanz zueinander
Mit Hilfe des Ultraschalls

Synchrones Abbiegen

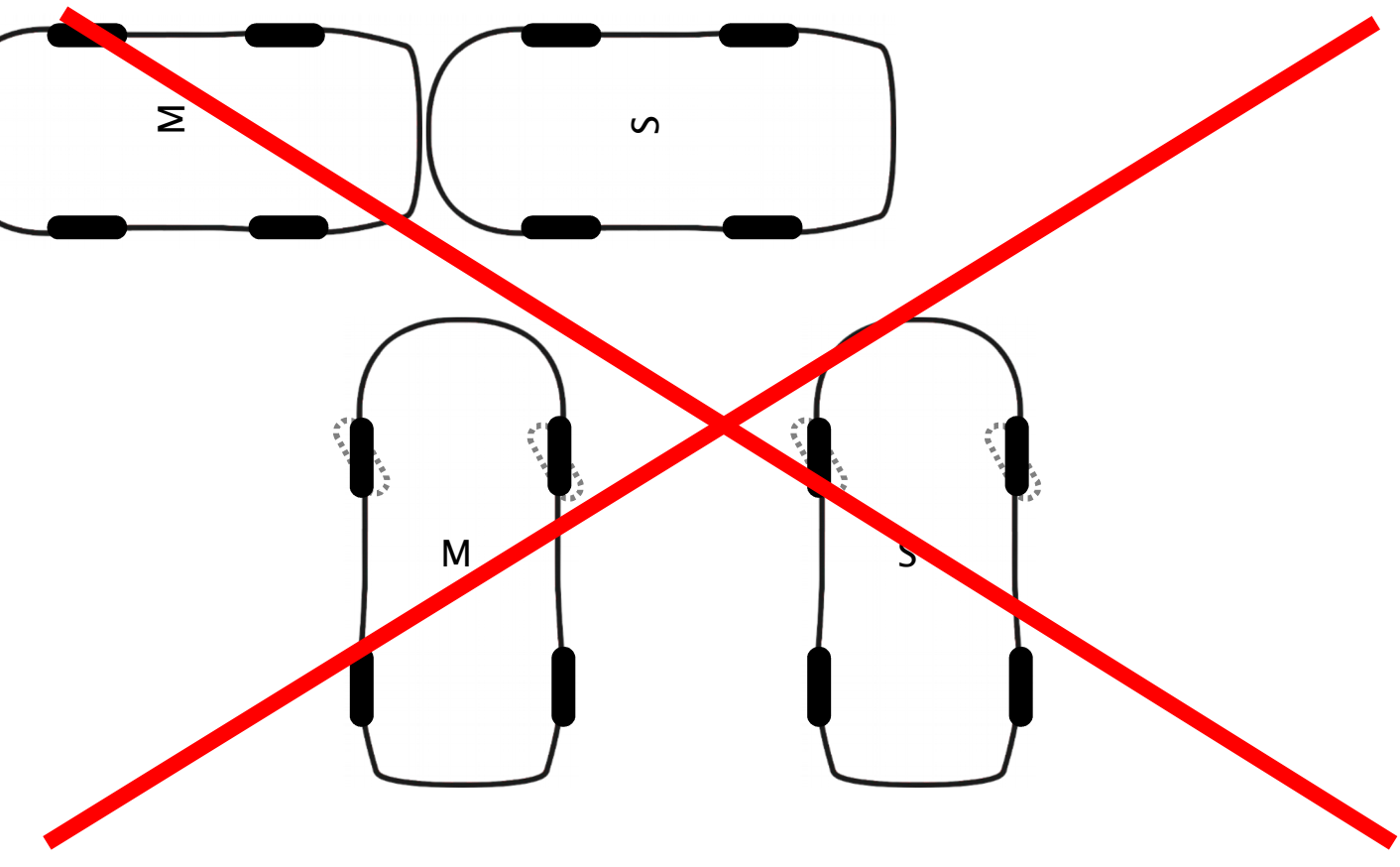


Synchronisierung

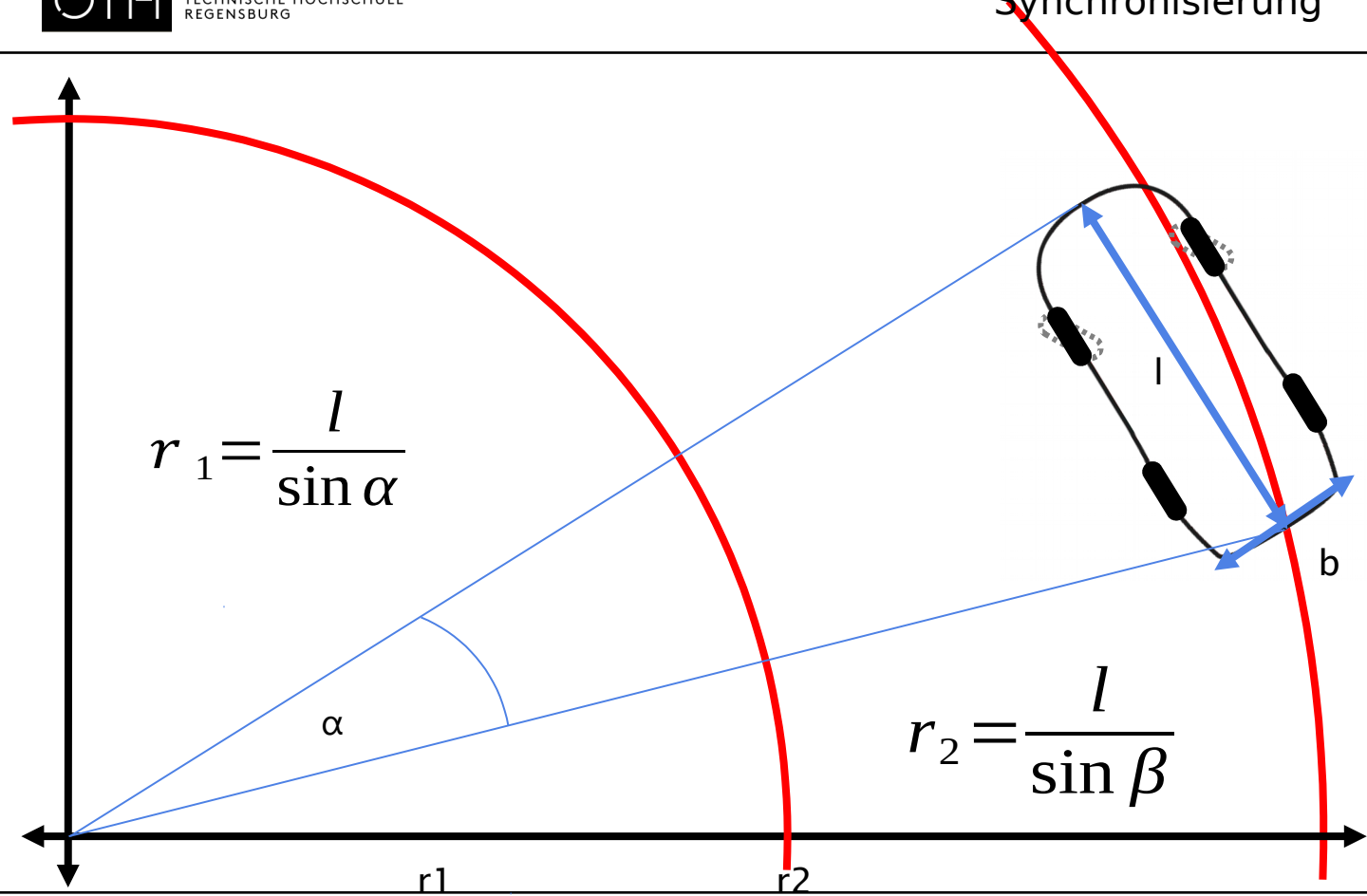




Synchronisierung



Synchronisierung



Die Geschwindigkeit berechnet sich durch das Verhältnis von den Geschwindigkeiten und dem Radius:

$$\frac{v_1}{v_2} = \frac{v_1}{v_2}$$

Ultraschall US misst den Abstand zwischen den Fahrzeugen. So lässt r2 alternativ berechnen:

$$r_2 = r_1 + US + b$$

Geschwindigkeit und Winkel:

$$v_2 = v_1 \frac{1 + (US + b) * \sin\alpha}{l}$$

$$\beta = \sin^{-1}\left(\frac{l}{\frac{l}{\sin\alpha} + US + b}\right)$$

Synchrones Abbiegen

Beispiel Linksabbiegen

Anpassung der Geschwindigkeit

Mit Hilfe von der Kamera

Anpassung der Distanz zueinander

Mit Hilfe des Ultraschalls

Synchrones Abbiegen

Beispiel Linksabbiegen

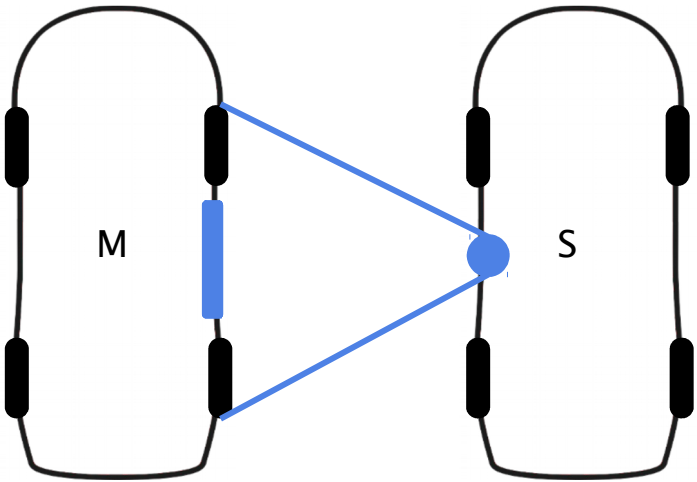
Anpassung der Geschwindigkeit

Mit Hilfe von der Kamera

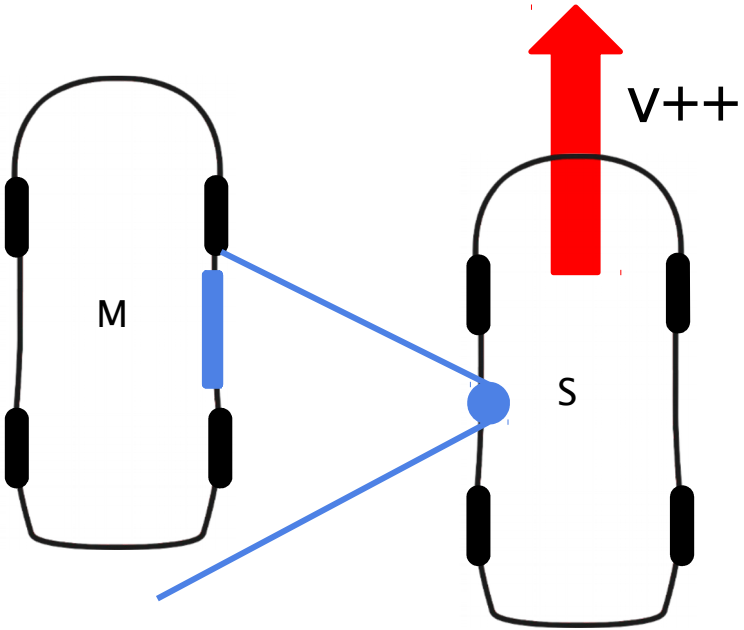
Anpassung der Distanz zueinander

Mit Hilfe des Ultraschalls

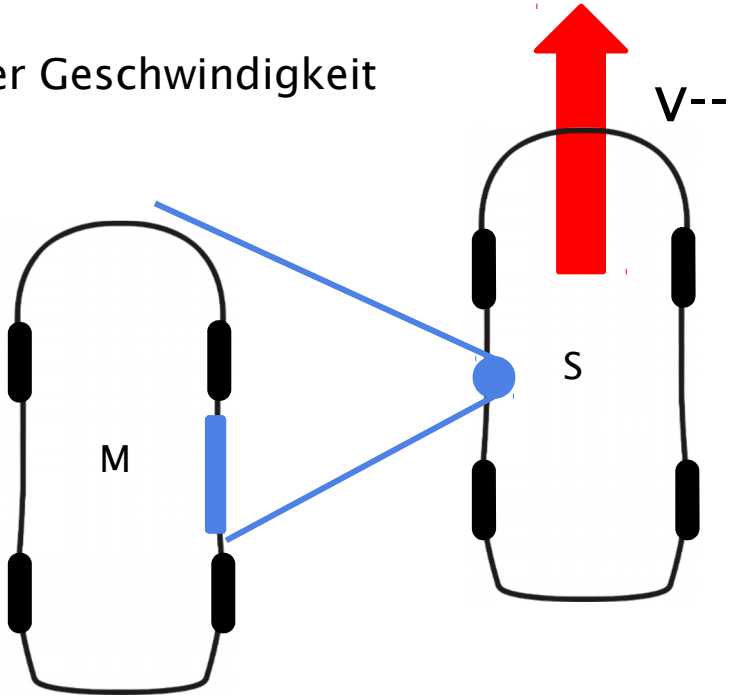
Anpassung der Geschwindigkeit



Synchronisierung



Anpassung der Geschwindigkeit



Synchrones Abbiegen

Beispiel Linksabbiegen

Anpassung der Geschwindigkeit

Mit Hilfe von der Kamera

Anpassung der Distanz zueinander

Mit Hilfe des Ultraschalls

Synchrones Abbiegen

Beispiel Linksabbiegen

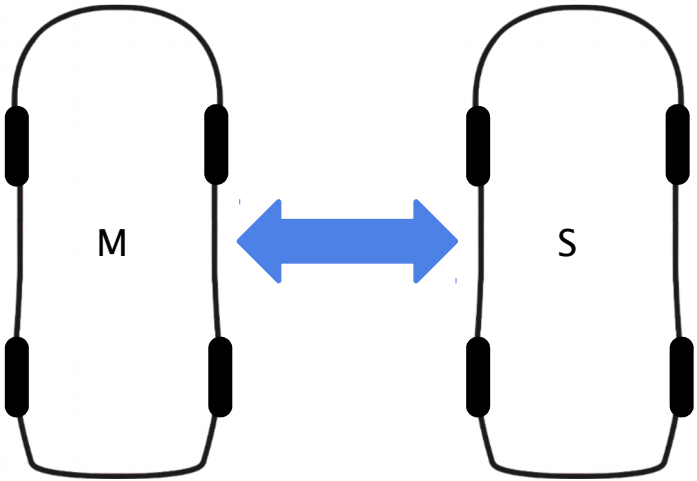
Anpassung der Geschwindigkeit

Mit Hilfe von der Kamera

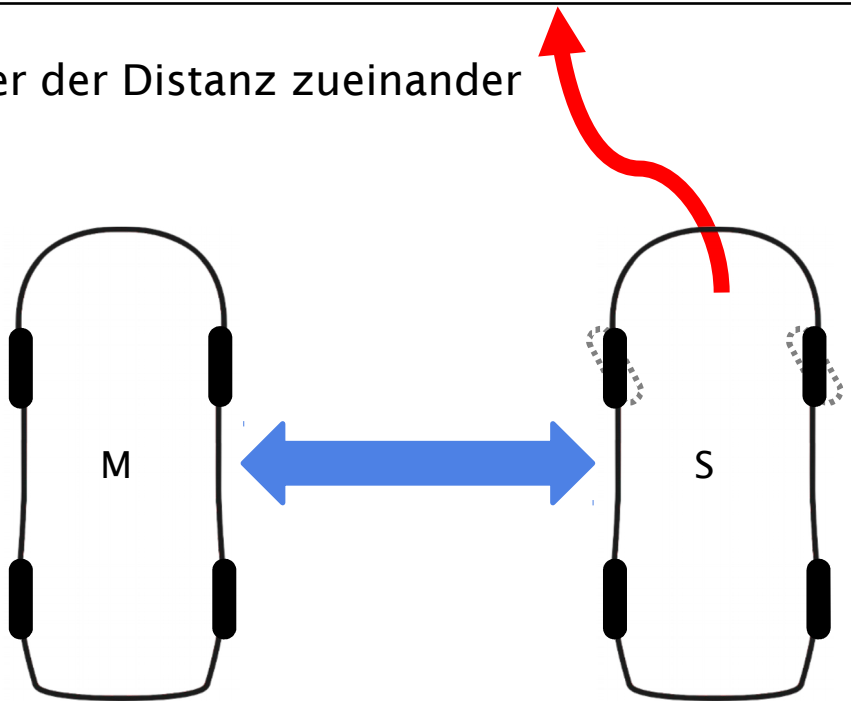
Anpassung der Distanz zueinander

Mit Hilfe des Ultraschalls

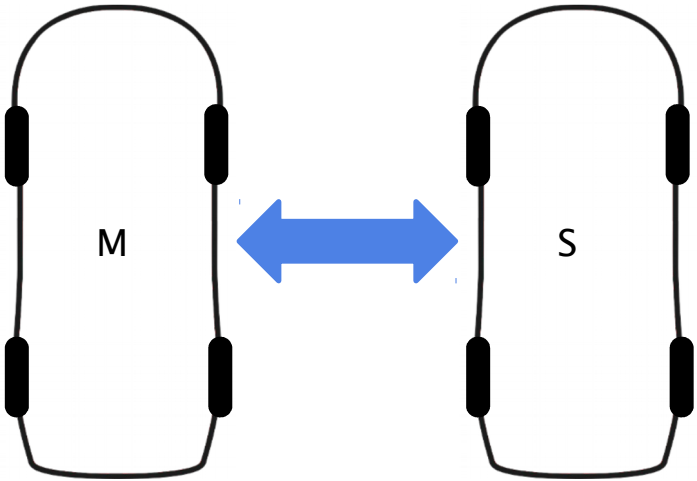
Anpassung der der Distanz zueinander



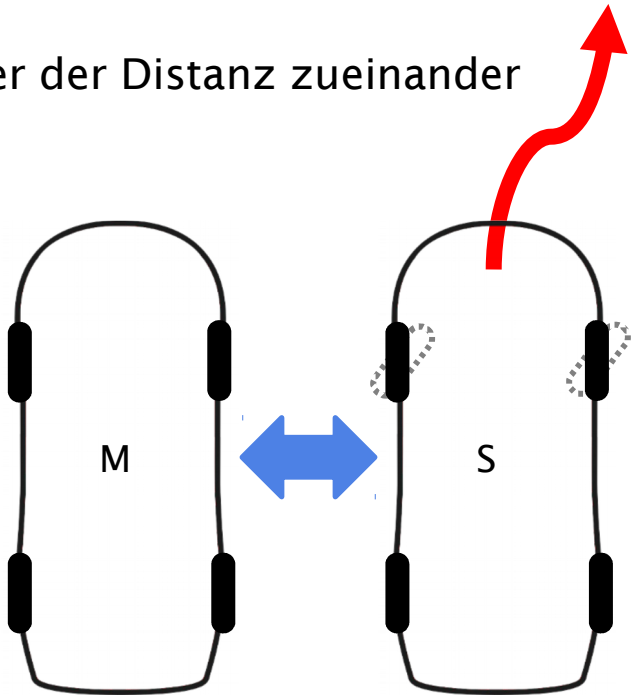
Anpassung der der Distanz zueinander



Anpassung der der Distanz zueinander



Anpassung der der Distanz zueinander



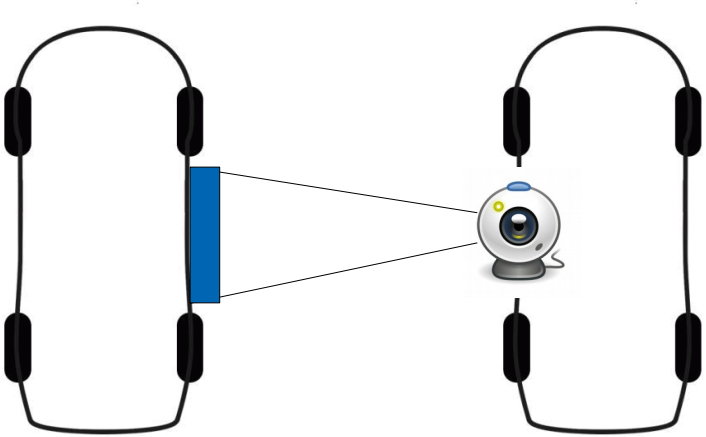
Kamera

Johannes Eichenseer

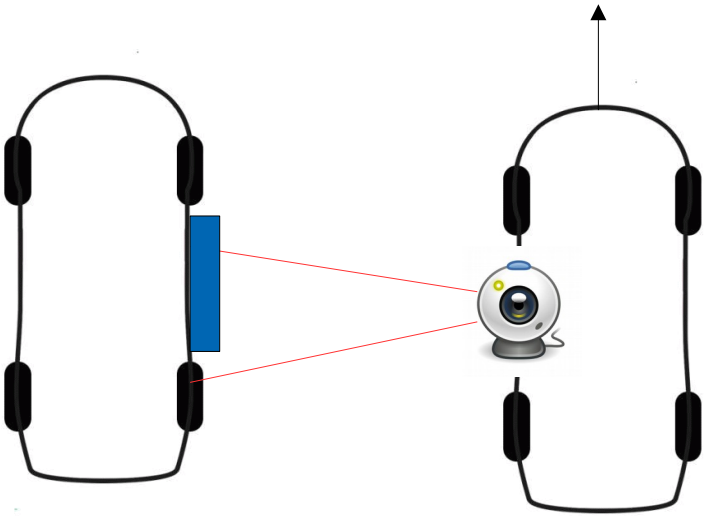
Kamera

- Logitech C300
  - Über USB an Raspberry Pi angeschlossen
  - Ansteuerung über OpenCV 3.4.1
  - 320x240 Auflösung bei 15/30 fps

Korrektur von Geschwindigkeitsdifferenzen

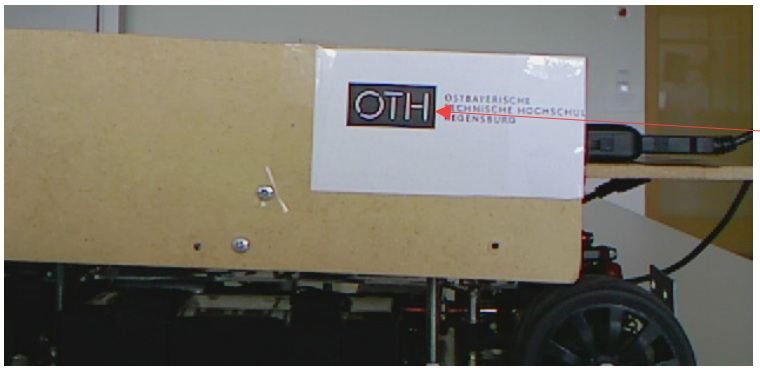


Kamera



Bilderkennung mit OpenCV

Fall 1: Bild weit entfernt



Kamerabild

Gespeichertes Muster

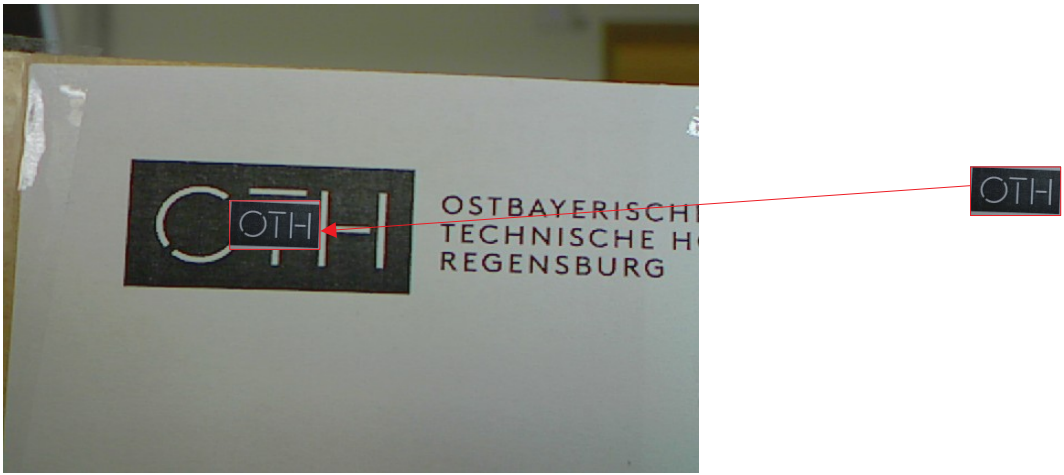
Bilderkennung mit OpenCV

Fall 1: Bild weit entfernt

Positive Erkennung!

Bilderkennung mit OpenCV

Fall 2: Bild sehr nahe



Keine Erkennung!

Bilderkennung mit OpenCV

Fall 2: Bild sehr nahe

- Bild wird in 20 Stufen von 100% - 20 % skaliert
- Bilderkennung wird auf jede Stufe angewendet
- Die beste Übereinstimmung wird gespeichert (Eckpunkte & Skalierungsfaktor)
- Die beste Übereinstimmung wird auf das Original rückskaliert und gespeichert

Beste Übereinstimmung

Bilderkennung mit OpenCV

Fall 2: Bild sehr nahe

Positive Erkennung!

Motion Tracking mit OpenCV

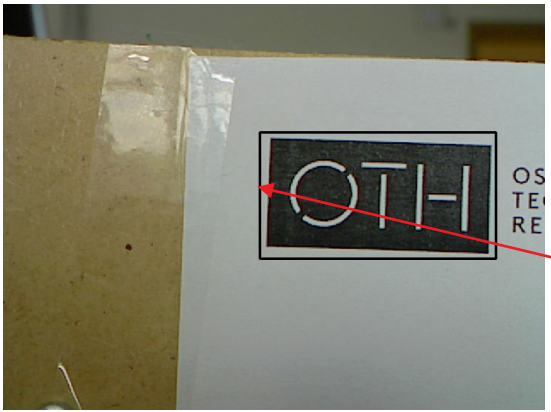
- Aus den Eckpunkten wird ein Mittelpunkt berechnet
- Nur die X-Koordinate ist relevant

Bildbreite:
192px – 52px = 140px

Mittelpunkt:
52px+140px/2 = 122px

Motion Tracking mit OpenCV

- Wiederhole Berechnung für jedes einzelne Bild

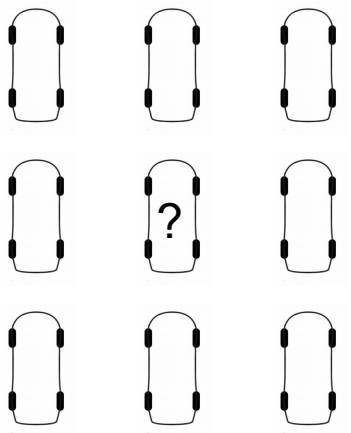


Bildbreite: 140px

Mittelpunkt:
150px+140px/2 = 220px

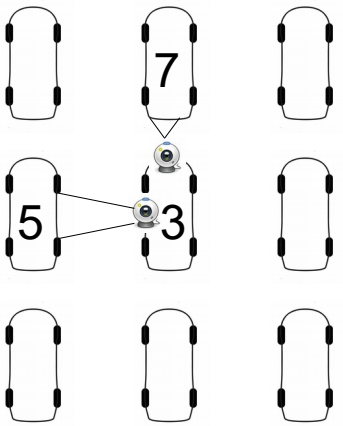
=> Auto muss etwas schneller fahren!

Positionsbestimmung mit mehreren Fahrzeugen



Positionsbestimmung mit mehreren Fahrzeugen

- Fahrzeuge erhalten individuelle ID
- ID wird als Barcode im Sichtbereich der Kamera montiert
- Jedes Fahrzeug gibt ID des vorderen und linken Fahrzeugs durch
- Vollständiges Grid kann erstellt werden



Fragen?

Vielen Dank
für eure Aufmerksamkeit!