

Dokumentation „DriveGeany“

25.07.2017

Bearbeiter	Hinweise
Jonas Heinke	

Inhalt

Aufgabe.....	1
1. Aufbau des Modellautos und Schnittstellen	2
2. Schnittstellen des Raspberry Pi.....	5
3. Schnittstellen des Nucleo	7
4. Verdrahtung und Anschlussbelegung	8
Wichtig:	8
5. Manuelle Richtungskontrolle.....	11
6. Pixy-Kamera als Objektsensor	11
7. Richtungskontrolle mit der Pixy-Kamera in Bezug zu einem Zielobjekt.	12
}//main	14
8. Geschwindigkeitskontrolle mit der Pixy-Kamera in Bezug zu einem Zielobjekt.	14
9. Fernwartung und Fernbedienung.....	17
Anlagen.....	21
Schaltplan:	21
Empfänger:	21
Fahrtenregler:	22

Aufgabe

Ziel des Projektes ist die Entwicklung eines ferngesteuerten und autonom fahrenden Modellautos. Die Fernsteuerung kann von einem entfernten Computer erfolgen. Für die autonom fahrende Funktion des Modellautos wird ein Zielobjekt (Target) definiert, dem hinterhergefahren wird. Bleibt das Zielobjekt stehen, so soll auch das Modellauto in einem bestimmten Abstand stehen bleiben. Wird kein Zielobjekt erkannt, so soll das Modellauto ebenfalls halten.

1. Aufbau des Modellautos und Schnittstellen

Das Modellauto besitzt folgenden Baugruppen (siehe Abbildung 1):

- Akkumulator 7,2 V
- Funkempfänger der Fernbedienung mit zwei belegten Kanälen
- Servomotor für die Lenkung
- Antriebsverstärker/Fahrtenregler für den Hauptmotor
- Hauptmotor als Antrieb



Abbildung 1: Gesamtaufbau des Modellautos

Die Ausgänge des Funkempfängers sind von Interesse, da diese Verbindung gekappt werden, um diese an den Nucleo und an der Raspberry anzuschließen. Die Abbildung 2 zeigt den Funkempfänger mit den relevanten Steckplätzen.

Der vordere Stecker (braun, rot, orange) führt zum Servomotor der Lenkung. Der gelbe Leiter ist für das PWM-Signal vorgesehen. Rot und braun sind für die Spannungsversorgung des Servomotors zuständig.

Der hintere Stecker (schwarz, rot, weiß) ist mit dem Fahrtenregler verbunden (siehe Abbildung 3). Die Leitungen Schwarz und Rot liefern die Versorgungsspannung ($U=6V$) sowohl für den Funkempfänger als auch für den Servomotor der Lenkung. Der weiße Leiter überträgt das PWM-Signal vom Funkempfänger zum Fahrtenregler.



Abbildung 2: Funkempfänger mit den jeweiligen Ausgängen



Abbildung 3: Antriebsverstärker

Leider konnte keine exakte Beschreibung zu diesen Schnittstellen gefunden werden.

Aus diesem Grunde waren Messungen erforderlich. Von Interesse sind die Periodendauer der PWM-Signale und die Pulslängen für typische Betriebszustände. Die Messungen erfolgten mit

einem Oszillographen während eines Testlaufes im Originalzustand des Modellautos. Exemplarisch wird mit der Abbildung 4 eine Aufzeichnung des Oszillographen dargestellt.

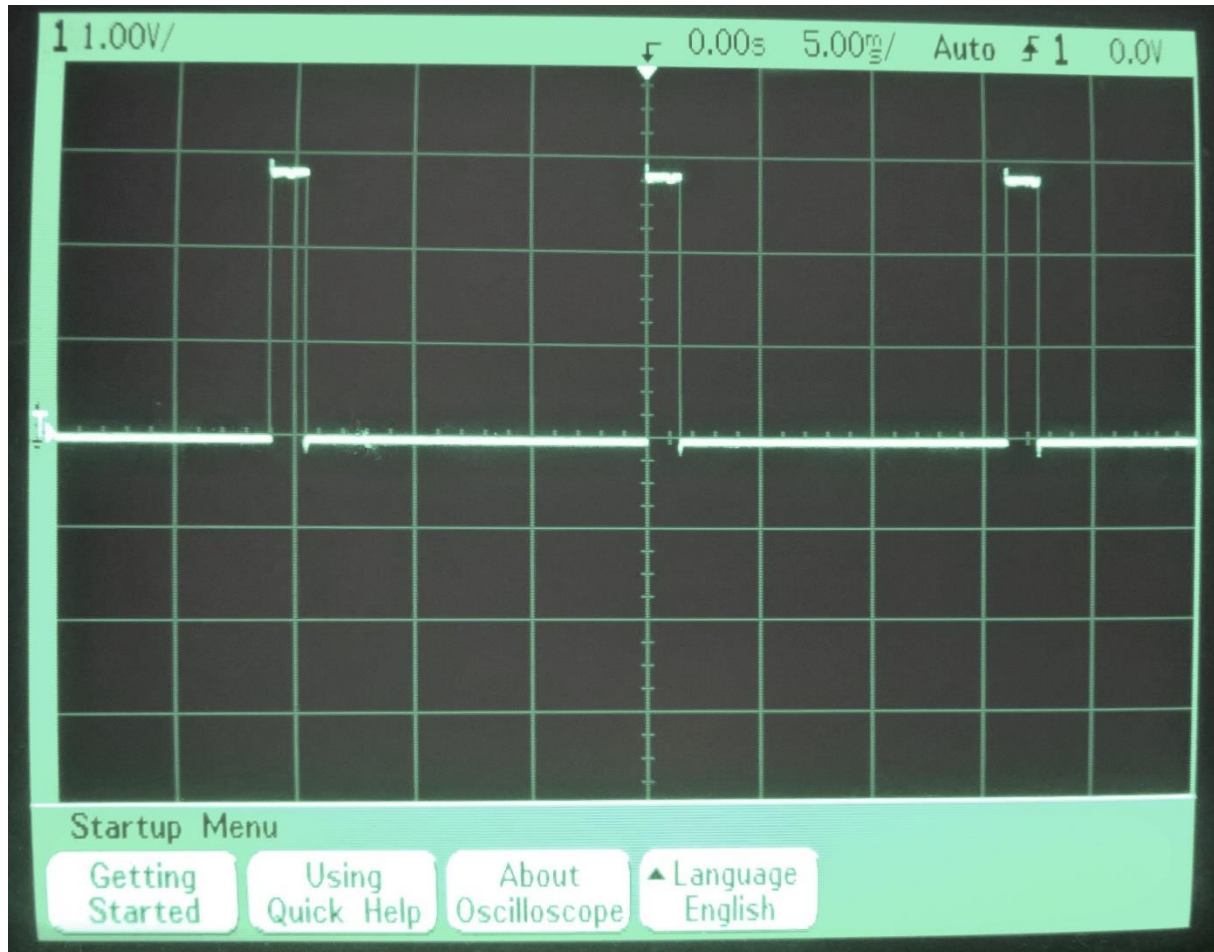


Abbildung 4: PWM-Signal

Die Tabelle 1 fasst die Messergebnisse zusammen.

Tabelle 1: PWM-Signale der Antriebe

Servomotor der Lenkung	
Periodendauer	16000 µs
Pulslänge: Vollausschlag links	1900 µs
Pulslänge: Mittelstellung	1500 µs
Pulslänge: Vollausschlag rechts	1200 µs
Fahrtenregler für Hauptmotor	
Periodendauer	16000 µs
Pulslänge: maximal rückwärts	1000 µs
Pulslänge: Halt	1500 µs
Pulslänge: maximal vorwärts	2000 µs

Auffällig sind die von 20000 µs abweichende Periodendauer und das asymmetrische PWM-Signal der Lenkung.

Mit diesen Erkenntnissen lassen sich die PWM-Ausgänge des Nucleo und des Raspberry konfigurieren.





















2. Schnittstellen des Raspberry Pi

Die Kommunikation des Raspberry Pi3 zum den Antrieb des Modellautos und zu einem Nucleo F303K08 erfolgt über den GPIO-Port (general purpose input/output). Die Darstellung aus dem Internet (Abbildung 5) [http://pi4j.com/pins/model-3b-rev1.html] wurde verwendet, um die Pins vorzustellen. Die fett dargestellte Nummer wird bei Verwendung der WiringPi-Bibliothek zur Programmierung der Ein- und der Ausgänge genutzt (siehe Tabelle 2).

Tabelle 2: Schnittstelle des Raspberry PI für des Modellauto

GPIO/WiringPi	Name	Aufgabe
LENKUNG – STEERING, << Ausgänge		
0	Bit0	Daten-Schnittstelle zum Nucleo mit dem bit0, bit1, bit2 und bit3. Übertragen wird ein 4-Bit-Wert zur Positionierung der Steuerung. Dieser wird vom Nucleo in ein entsprechendes PWM-Signal gewandelt.
2	Bit1	
3	Bit2	
4	Bit3	
5	bitCrt	Steuert die Datenübertragung zum Nukleo. Nur wenn dieses Bit gesetzt ist, werden die Steuer-Daten vom Nucleo übernommen.
HAUPTANTRIEB – MAINDRIVE , << Ausgänge		
26	motorPWM	Mit „motorPWM“ wird die Leistung des Hauptmotors gesteuert. Das geschieht über die Pulsweite (PWM - Pulsweitenmodulation).



Raspberry Pi 3 Model B (J8 Header)					
GPIO#	NAME			NAME	GPIO#
	3.3 VDC Power	1		2	5.0 VDC Power
8	GPIO 8 SDA1 (I2C)	3		4	5.0 VDC Power
9	GPIO 9 SCL1 (I2C)	5		6	Ground
7	GPIO 7 GPCLK0	7		8	GPIO 15 TxD (UART) 15
	Ground	9		10	GPIO 16 RxD (UART) 16
0	GPIO 0	11		12	GPIO 1 PCM_CLK/PWM0 1
2	GPIO 2	13		14	Ground
3	GPIO 3	15		16	GPIO 4 4
	3.3 VDC Power	17		18	GPIO 5 5
12	GPIO 12 MOSI (SPI)	19		20	Ground
13	GPIO 13 MISO (SPI)	21		22	GPIO 6 6
14	GPIO 14 SCLK (SPI)	23		24	GPIO 10 CE0 (SPI) 10
	Ground	25		26	GPIO 11 CE1 (SPI) 11
30	SDA0 (I2C ID EEPROM)	27		28	SCL0 (I2C ID EEPROM) 31
21	GPIO 21 GPCLK1	29		30	Ground
22	GPIO 22 GPCLK2	31		32	GPIO 26 PWM0 26
23	GPIO 23 PWM1	33		34	Ground
24	GPIO 24 PCM_FS/PWM1	35		36	GPIO 27 27
25	GPIO 25	37		38	GPIO 28 PCM_DIN 28
	Ground	39		40	GPIO 29 PCM_DOUT 29

Attention! The GPIO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

<http://www.pi4j.com>

Abbildung 5: GPIO-Port des Raspberry Pi3 [<http://pi4j.com/pins/model-3b-rev1.html>]

3. Schnittstellen des Nucleo

Der Nucleo dient zur Erzeugung eines störungsfreien PWM-Signals zur Steuerung des Modellautos. Dazu erhält der Nucleo vier Datenbit und ein Steuerbit vom Raspberry. Mit den vier Datenbits lassen sich Werte zwischen 0 und 15 darstellen. Mit dem Steuerbit wird garantiert, dass nur gültige Steuerwerte übernommen werden. Die Werte 0 und 15 stellen die maximalen Lenkausschläge dar. Der Mittenwert ist eigentlich 7,5 der sich allerdings nicht einstellen lässt. Folglich muss die 7 oder die 8 für die Geradeausfahrt festgelegt und bei der Montage des Autos entsprechend kalibriert werden.

Die Abbildung 6 zeigt die Pinbelegung und die Tabelle 3 erklärt die verwendeten Pins.

Tabelle 3: Pinbelegung des Nucleo

PIN	Name	Aufgabe
Daten für die Steuerung vom Raspberry, >> Eingänge		
PA0	Bit0	Datenbits für die Steuerung. Der Nucleo dient als Slave und wandelt die Bits in ein entsprechendes PWM-Signal um
PA1	Bit1	
PA3	Bit2	
PA4	Bit3	
PA7	crtBit	Steuerbit, wenn gesetzt, dann können die zugehörigen Datenbits übernommen werden.
Servo-PWM-Signal, << Ausgang		
PB2	steeringPwm	Liefert ein PWM-Signal mit einer Pulslänge von 1100 µs bis 1900 µs bei einer Periode von 16000 µs. Der Datenwert 0 entspricht der Pulslänge 1100µs und der Datenwert 15 entspricht der Pulslänge von 1900 µs. Bei 1100 µs fährt der Servo ganz nach rechts und bei 1900 µs ganz nach links. Die Umrechnung der Eingangswerte in das Ausgangs-PWM-Signal übernimmt der Nucleo.
Spannungsversorgung		
GND	Ground	Masse, bzw 0-Potential
VIN		Versorgungsspannung zwischen 7 bis 12 V Der Nucleo kann über diesen Pin direkt an die Spannungsquelle des Modellautos angeschlossen werden. Die meisten Modellautos werden mit 7,4 V betrieben.
5V	5V	Ein/Ausgangsspannung Hier kann der Nucleo mit 5V, alternativ zu VIN, versorgt werden. Es können aber auch 5V, zum Beispiel zur Versorgung des Servomotors, abgegriffen werden. Der maximal zulässige Ausgangsstrom ist bei Verwendung dieser Stromquelle zu beachten (300 mA).
+3V3	3,5V	Kann als Stromquelle dienen. Maximal zulässiger Strom ist zu beachten (500 mA).

Weiterführende Informationen sind der Originaldokumentation zu entnehmen [UM1956 User manual STM32 Nucleo-32 board] Neben dem Typ „NUCLEO-F303K8“ sind dort weitere Typen aufgeführt.

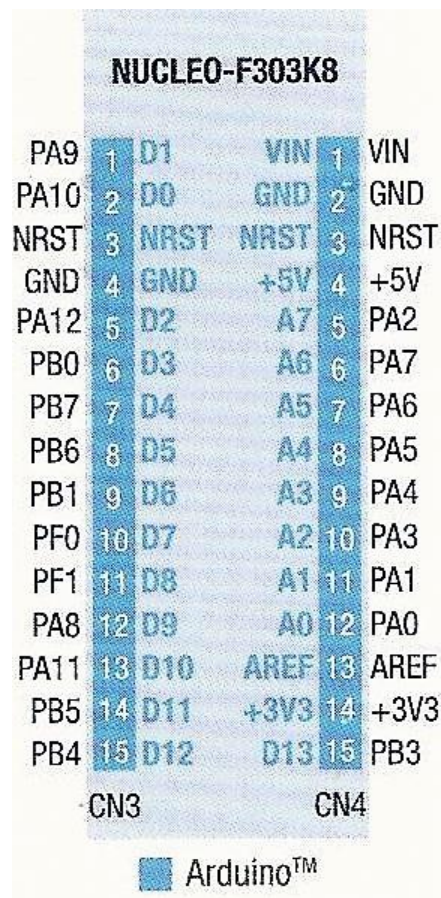


Abbildung 6: Pinbelegung des Nucleo F303K8
[\[https://developer.mbed.org/platforms/ST-Nucleo-F303K8/\]](https://developer.mbed.org/platforms/ST-Nucleo-F303K8/)

4. Verdrahtung und Anschlussbelegung

Die komplette Anschlussbelegung zeigt die folgende Darstellung (Abbildung 7).

Der Nucleo wird für den Fahrbetrieb über ein vollwertiges USB-Kabel mit dem Raspberry verbunden. Die Stromversorgung erfolgt ebenfalls über ein vollwertiges USB-Kabel mit dem Power Pack. Die Pinbelegung ist aus der Darstellung und den vorangegangenen Abschnitten ersichtlich.

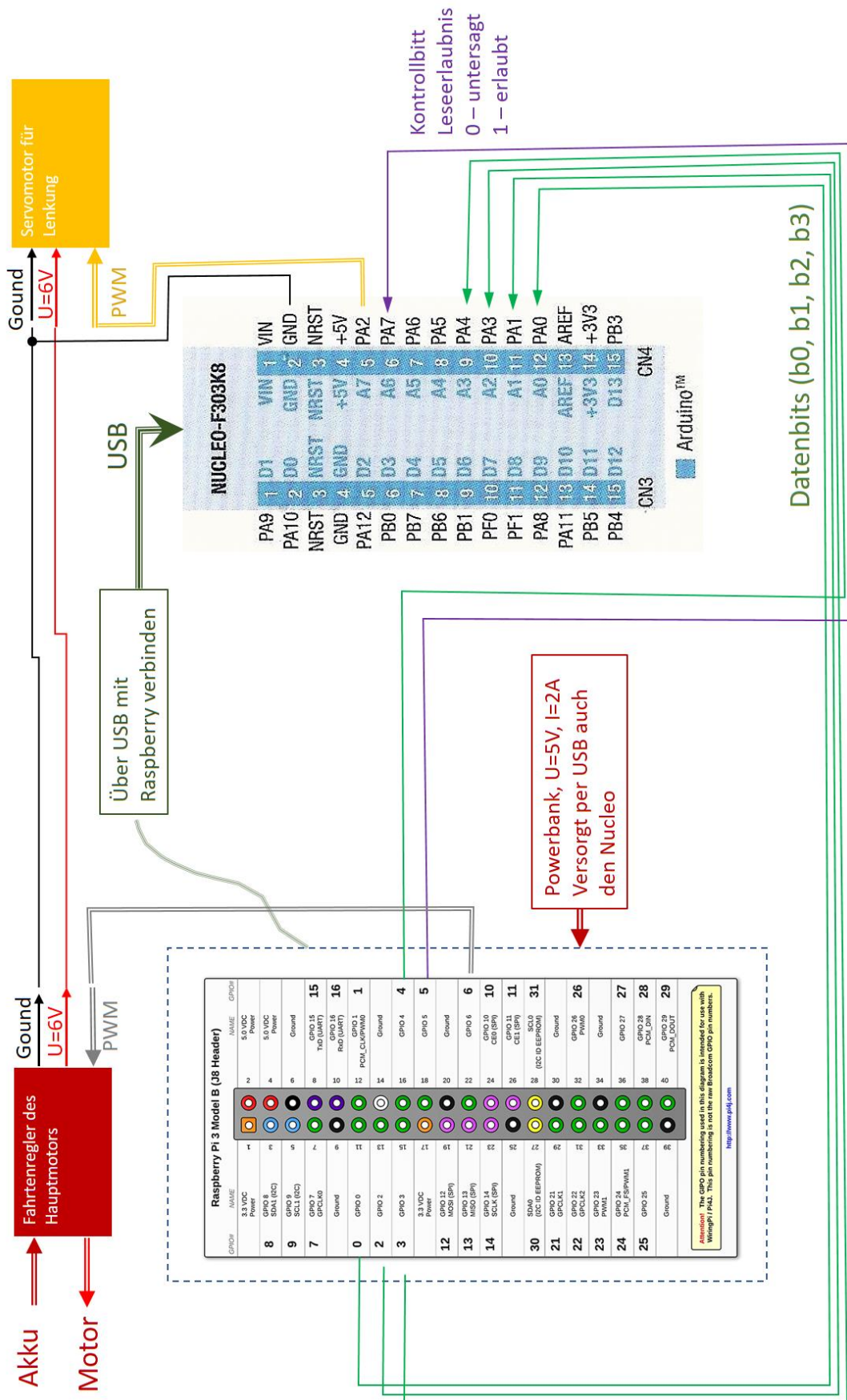
Zur Inbetriebnahme sind zuerst der Raspberry und der Nucleo in Betriebzunehmen und hochzufahren. Danach wird der Hauptschalter des Modellautos betätigt. Jetzt kann das Programm gestartet werden um die gewünschte Betriebsart auszuwählen.

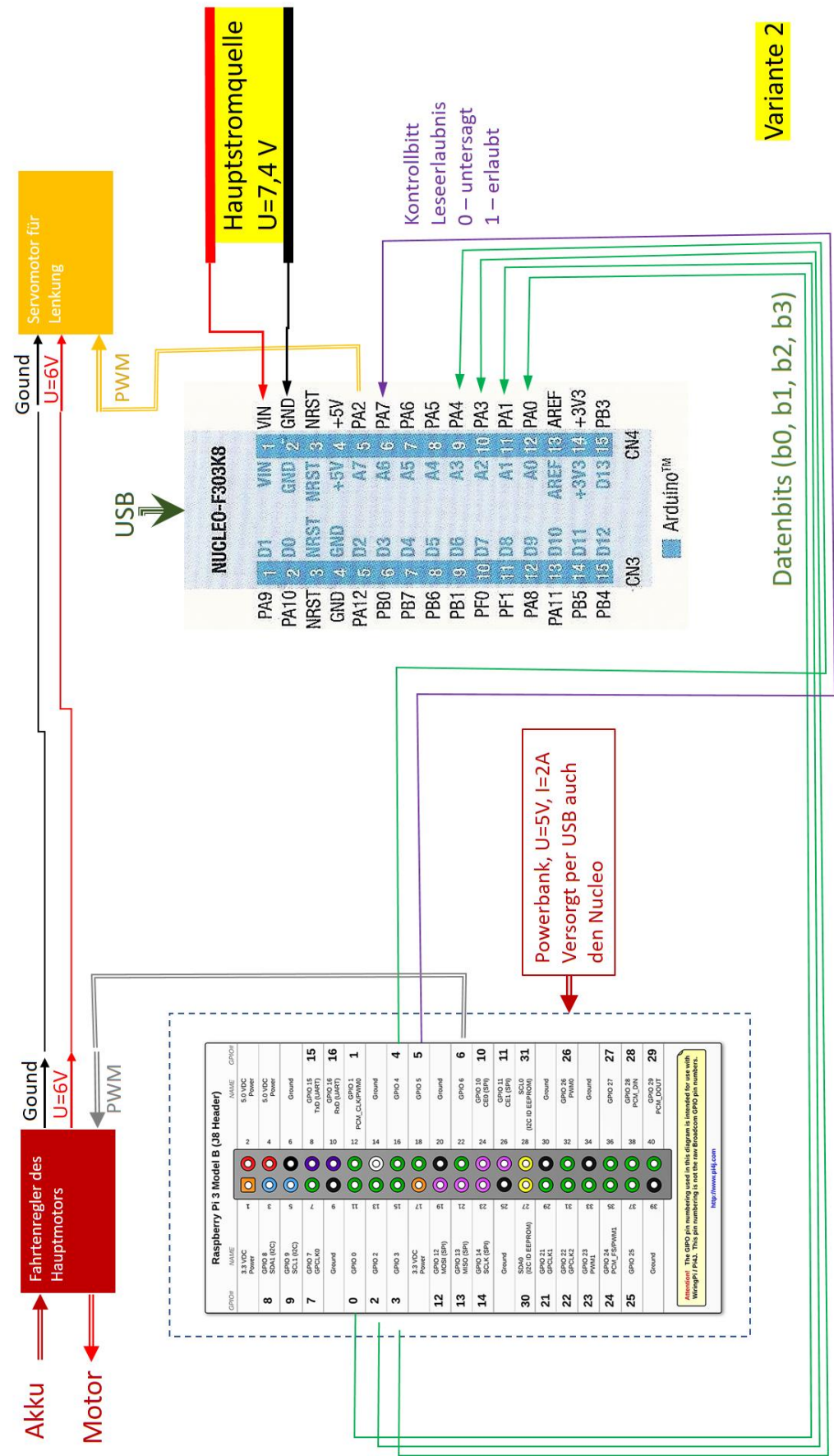
Wichtig:

Der Nucleo ist zwar klein, benötigt aber eine zuverlässige Spannungsversorgung. Die Spannungsversorgung über den Raspberry mittels USB-Kabel ist gegebenenfalls nicht ausreichend. Das zeigt sich durch eine unzuverlässige Steuerung.

Eine zuverlässige Energieversorgung des Nucleo kann durch den Anschluss der Pins VIN und GND direkt an die Hauptstromquelle $U=7,4\text{ V}$ des Modells erfolgen. Eine Masseverbindung zum $U=6\text{ V}$ Energiestrang ist dann nicht mehr erforderlich, gegebenenfalls auch nicht ratsam.

Um Missverständnisse zu vermeiden, wird der komplette Anschlussplan mit den hier genannten Änderungen in der Abbildung 8: Pinbelegung (Variante 2)“ dargestellt.





5. Manuelle Richtungskontrolle

Die Manuelle Richtungskontrolle erfolgt mit Hilfe der Cursortasten [Rechts, Links]. Bei jedem Tastendruck bewegt sich der Servomotor um 1/16 des Stellbereichs in die gewählte Richtung. Die Taste [Entf] dient zur schnellen Anwahl der Mittelstellung

Vorwärts und rückwärts werden ebenfalls per Cursortasten gewählt. Hier gibt es insgesamt nur 10 Schritte. Die Taste [Space] stoppt das Auto.

6. Pixy-Kamera als Objektsensor

Die hier verwendete Pixy-Kamera beherrscht eine Bildvorverarbeitung. Sie lässt sich auf Farb-Objekte anlernen, deren Positionswerte und Größe bereitgestellt werden. Eine eigene Klasse musste zu diesem Zweck erstellt werden. Die Methoden der Klasse „CamPixy“ initialisiert die Pixy-Kamera und liefert die Pixel-Koordinaten der angelernten Objekte. Das Anlernen der Objekte geschieht mit dem Originalprogramm „PixyMon“. Die Abbildung 9 zeigt eine Pixy-Momentaufnahme, mit den von der Pixy zur Verfügung gestellten Objektdaten. Der Koordinatenursprung des Bildes befindet sich in der linken oberen Ecke. Eine Signatur des Objektes wird als Index, mit 1 beginnend, ebenfalls mitgeliefert.

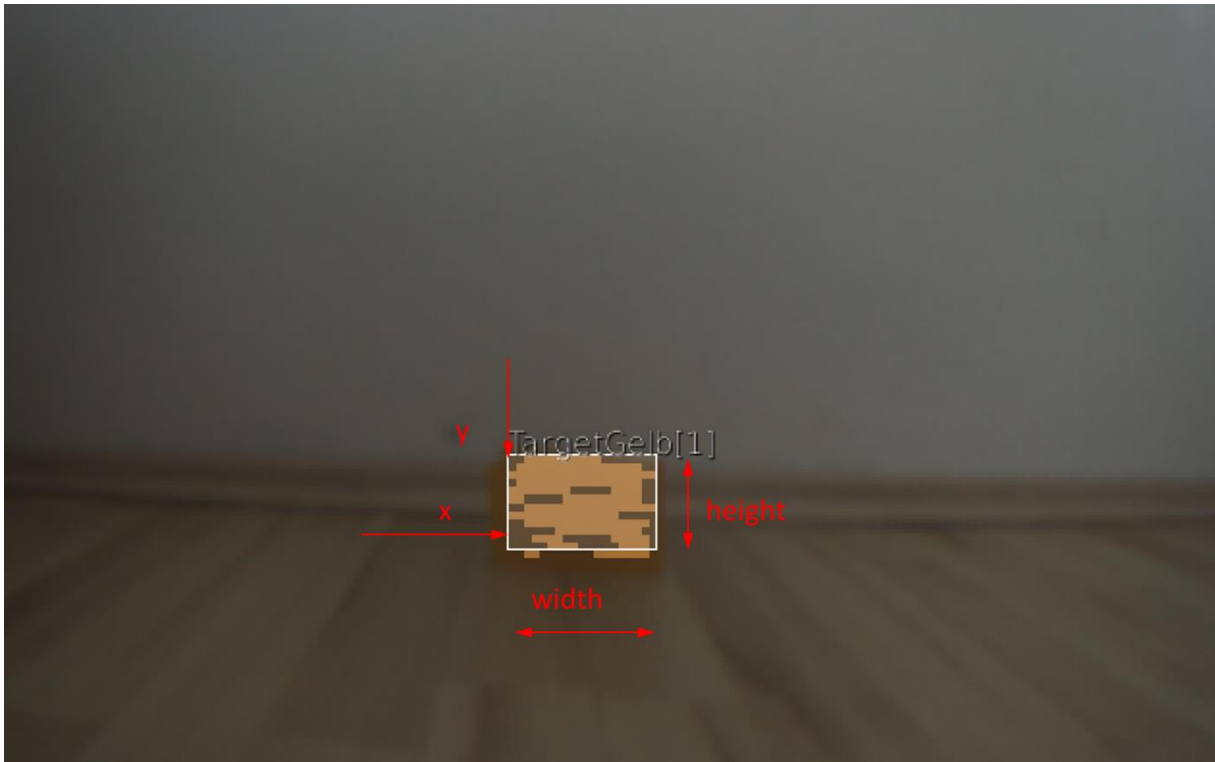


Abbildung 9: Bild der Pixy-Kamera

Die Objektdaten können in etwa wie im Programm 1 dargestellt, ausgelesen werden. Dazu werden statische Methoden der Klasse „CamPixe“ verwendet. Als Grundlage für diese Klasse diente das Programm „hello_pixy“ aus dem Internet [http://cmucam.org/projects/cmucam5/wiki/Hooking_up_Pixy_to_a_Raspberry_Pi]. Allerdings waren für diesen Zweck etliche Anpassungen erforderlich.

Programm 1: Pixy als Sensor

```
for (int index = 0; index != CamPixy::blocks_copied; ++index) {  
    printf(" \n komponent:  sig(index= %1d):%1d x:%4d y:%4d width:%4d height:%4d\n",  
        index,  
        CamPixy::blocks[index].signature,
```

```

    CamPixy::blocks[index].x,
    CamPixy::blocks[index].y,
    CamPixy::blocks[index].width,
    CamPixy::blocks[index].height);
}

```

7. Richtungskontrolle mit der Pixy-Kamera in Bezug zu einem Zielobjekt.

Dieser Abschnitt bezieht sich auf die Methode „colorTargetEasy(...)“ des Main-Programms „mainDrive“.

Die Methoden der Klasse „Bildanalyse“ berechnet näherungsweise die Positionsdaten des Zielobjektes in der realen Welt. Für die Richtung ist die Position des Objektes in Bezug zur Bildmitte von Interesse. Dazu muss folglich die Mitte des Zielobjektes ermittelt werden und im Anschluss die Abweichung zur Bildmitte. Negative Werte bedeuten: Fahre nach links. Positive Werte bedeuten: Fahre nach rechts. Der Differenzbetrag bestimmt die Stärke des Lenk ausschlages (siehe Abbildung 10).

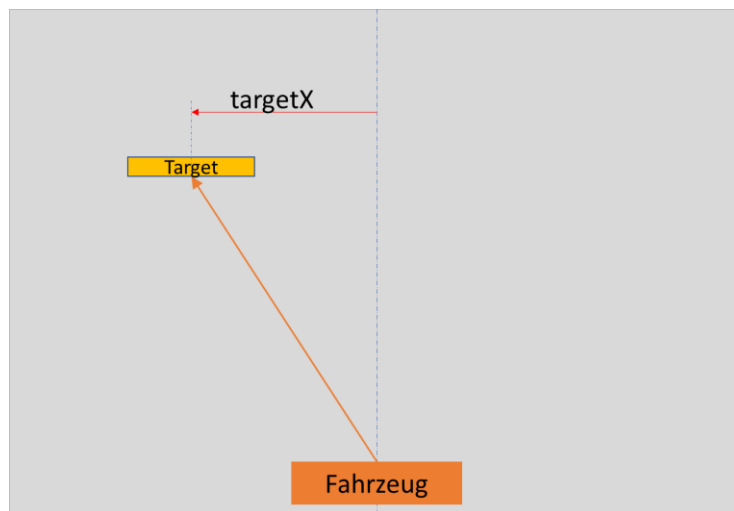


Abbildung 10: Lenkung

Zur besseren Kontrolle der Lenkbewegungen wird die Methoden „Regler::proportionalSym(...)“ verwendet. Mit dem Parameter „steeringFaktor“ lässt sich beispielsweise die Lenkbewegung dämpfen, um zu abrupte Lenkbewegungen zu vermeiden. Für optimale Einstellungen sind Versuche notwendig. Letztendlich soll ein Servomotor über den Nucleo angesteuert werden. Dazu sind die bisher vorliegenden Werte in den positiven Wertebereich von 0 bis 15 umzurechnen. Das sind im binären Zahlensystem genau 4 bit, die dann zum Nucleo übertragen werden. Die Abbildung 11 zeigt das Prinzip.

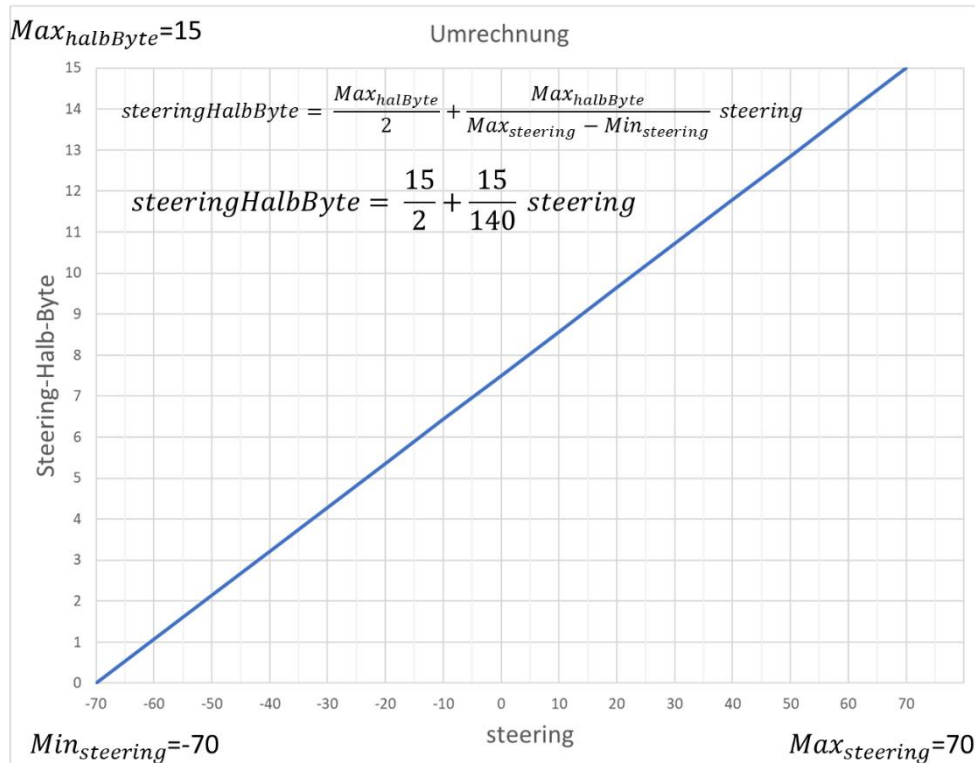


Abbildung 11: Umrechnung zum Lenken

Das Berechnete Halb-Byte wird zum Nucleo übertragen. Insgesamt ergeben sich daraus unterschiedliche Lenkpositionen. Allerdings besitzt der Wertebereich keine ganzzahlige Mitte, so, dass in eine Richtung ein Segment weniger zur Verfügung steht. Es sei noch Bemerkt, dass der Nucleo die 16 Einstellungen in Pulslängen von 1200 bis 1900 μ s umrechnet. Siehe Programm 2!

Wichtig ist die Einstellung „range“. Der Schwenkbereich eines Servomotors liegt in der Regel über 180°. Der Stellbereich der Lenkung ist in der Regel geringer. Im Programm, grün hinterlegt, lässt sich der Stellbereich über den Parameter „range“ beeinflussen. Leider ist keine Maßeinheit für diesen Wert bekannt. Der Wert „range=0.0004“ entspricht in diesem Fall einem Stellbereich von etwa 45°.

Programm 2: Nucleo Input-Schnittstelle vom Raspberry und PWM-Signal für Servoantrieb

```
#include "mbed.h"
//DEKLARATIONEN
PwmOut steeringPwm(PA_2); //Lenkung
//Test-LED
DigitalOut ledStatus(LED1);
//Bit-Code für Servoposition: 0..16 Positionen
DigitalIn bit0(PA_0); // A0
DigitalIn bit1(PA_1); // A1
DigitalIn bit2(PA_3); // A2
DigitalIn bit3(PA_4); // A3
// steuert gleichzeitige Servopositionsübernahme
DigitalIn readCrt(PA_7); // Lesen: 0- nicht erlaubt, 1- erlaubt
// Periodendauer
int periode=16000; // in microsekunden
//int periode=20000; //Standard-Servo
// Pulsdauer für die Lenkung
int steeringMitte=1500;
int steeringVollLinks=1900; //eigentlich 1900
int steeringVollRechts=1100; //eigentlich 1200
```



```

int main()
{
    //Main-Deklaration
    int halbByte=7; //daten
    int halbByteOld=7;
    int pulsweiteSteering=1500; //temporaer zur Berechnung
    printf("Wandelt ein Halbbyte (0 .. 15) in ein PWM-Signal um!\n");
    // Initialisierung
    steeringPwm.period_us(periode);
    ledStatus = 1;
    wait(2);
    // Pwm-Signale auf Mittelstellung
    steeringPwm.pulsewidth_us(steeringMitte);
    ledStatus=0;
    wait(3);

    //Umrechnungsgleichung  $Y=n + m * x$ 
    int n=steeringVollRechts;
    double m=(steeringVollLinks-steeringVollRechts)/15;

    while(1) // Big Cycle
    {
        ledStatus=1;
        halbByte=0;
        if (bit0) {halbByte +=1; }
        if (bit1) {halbByte +=2; }
        if (bit2) {halbByte +=4; }
        if (bit3) {halbByte +=8; }

        // LENKUNG .....
        if (readCrt) //gemeinsame Auswertung aller Eingänge
        {
            if (!(halbByte == halbByteOld))
            {
                pulsweiteSteering= (int) (n + halbByte * m);
                halbByteOld=halbByte;
                ledStatus=1;
                //wait(0.2);
                steeringPwm.pulsewidth_us(pulsweiteSteering);
            }
        }
    } //Big Cycle
} //main

```

8. Geschwindigkeitskontrolle mit der Pixy-Kamera in Bezug zu einem Zielobjekt.

Dieser Abschnitt bezieht sich auf die Methode „colorTargetEasy(...)“ des Main-Programms „mainDrive“.

Die Geschwindigkeit lässt sich leider nur indirekt über die Motorspannung beziehungsweise über die Motorleistung steuern. Während des Anfahrvorgangs wird eine höhere Motorleistung benötigt als während der Fahrt. Ferner sind die Straßenverhältnisse von Bedeutung. Folglich wird in Abhängigkeit von der Entfernung des Autos zum Zielobjekt die Motorleistung beeinflusst. Kurz vor dem Zielobjekt soll das Auto stehen bleiben. Gleiches gilt, wenn kein Zielobjekt erkannt wird. Es muss zunächst die Entfernung zum Zielobjekt abgeschätzt werden. Da nur eine Kamera zur Verfügung steht entfällt eine stereoskopische Entfernungsberechnung.

Zur Entfernungsabschätzung wird folgende Annahme getroffen. Auto und Zielobjekt befinden sich auf einer Ebene, beziehungsweise auf einer Straße. Die Kamera ist erhöht und gegebenenfalls geneigt angebracht. Wenn das Objekt sich entfernt, so verschiebt sich das

Objekt in Richtung der oberen Bildkante. Umgekehrt wandert es zur unteren Bildkante, wenn sich das Objekt nähert. Die y-Koordinate des Objektes sagt also etwas über die Objektentfernung aus. Die Objektunterkante wird detektiert, um die Entfernung abzuschätzen. Die optischen und geometrischen Zusammenhänge lassen sich aus der Abbildung 12 ableiten.

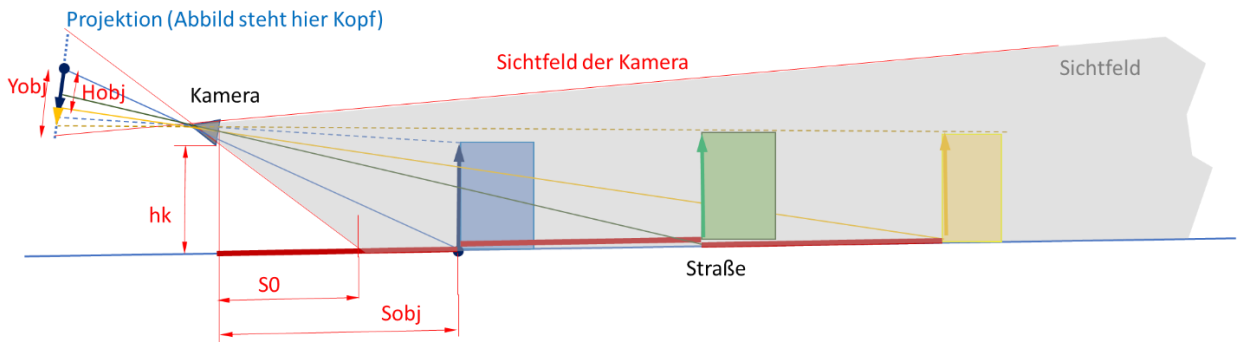


Abbildung 12: Projektion der Objekte

Ferner ist zu berücksichtigen, dass, je nach Einbausituation der Kamera, sehr nahe und sehr ferne Objekte nicht erkannt werden.

Ein vereinfachtes Berechnungsmodell geht davon aus, dass die optische Achse der Kamera parallel zur Fahrbahn ausgerichtet ist (siehe Abbildung 13).

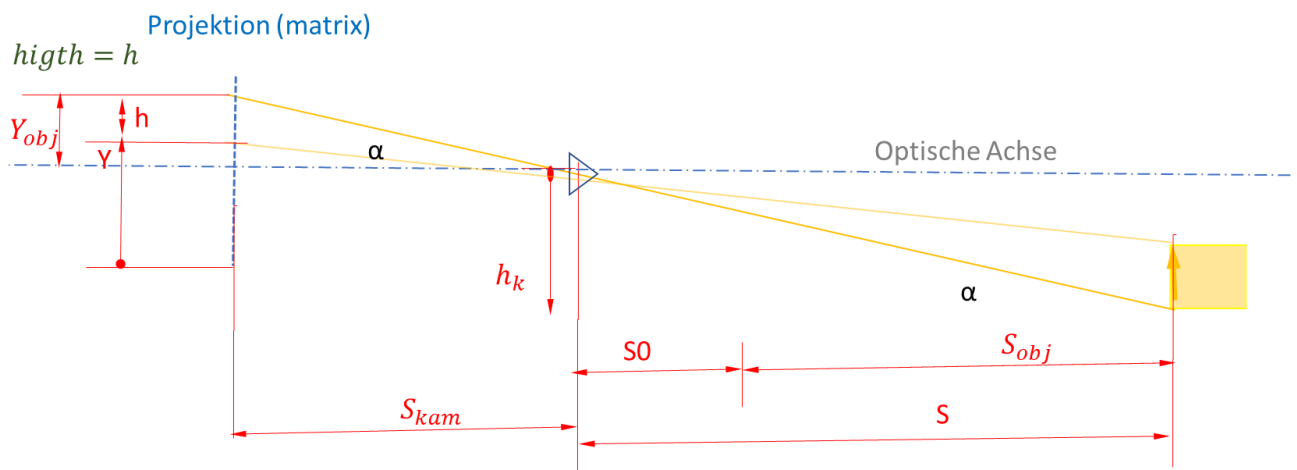


Abbildung 13: Horizontale Ausrichtung der Kamera

S_{kam} - Kamerakonstante in Pixel, kann experimentell ermittelt werden

Y_{max} - Kameraauflösung in y-Richtung in Pixel

Y_{obj} - Objektkante in Pixel

h_k - Optische Achse über Objektunterkante in cm

y - Kantenabbild der oberen Objektkante in Pixel

$y + h$ - Kantenabbild der unteren Objektkante in Pixel

S - Entfernung des Objektes in Bezug zur Kamera in cm

Die Berechnung der Entfernung erfolgt entsprechend der folgenden Gleichung:

$$S = \frac{h_k}{y_{obj}} S_{kam} = \frac{h_k}{y + h - \frac{y_{max}}{2}} S_{kam} \quad \text{für } (y + h) \neq \frac{y_{max}}{2}$$

Die Kamerakonstante S_{kam} muss experimentell ermittelt werden. Dazu ist für eine bekannte Entfernung S die Gleichung nach S_{kam} umzustellen.

Ein weitaus aufwendigeres Modell berücksichtigt die Neigung der Kamera. Dadurch reduziert sich der Totbereich unmittelbar vor der Kamera und sehr nahe Objekte können detektiert werden (siehe Abbildung 14).

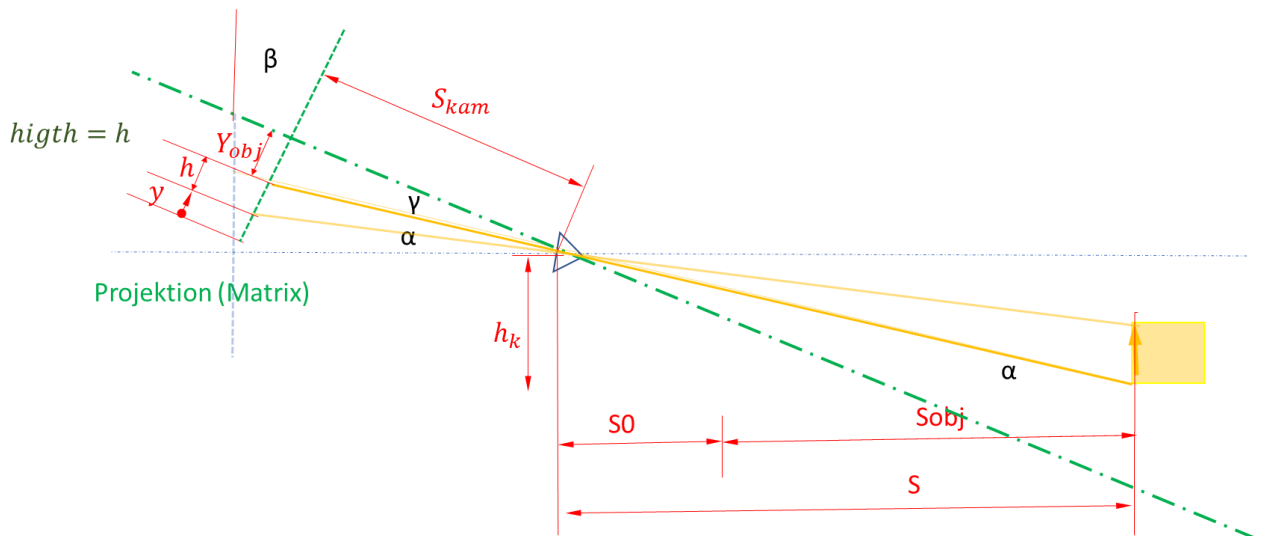


Abbildung 14: Geneigte Kamera

$\tan \beta$ - Neigung der Kamera

Die folgende Gleichung ist geeignet, um die Entfernung des Objektes bei geneigter Kamera zu berechnen.

$$S = h_k \left[\frac{S_{kam} + \left[y + h - \frac{y_{\max}}{2} \right] \tan \beta}{S_{kam} \tan \beta + y + h - \frac{y_{\max}}{2}} \right]$$

Die Methode „colorTargetEasy(..)“ verwendet gegenwärtig das einfachere Verfahren. In der Klasse „Bildanalyse“ sind allerdings mehrere Varianten als Methode hinterlegt. Innerhalb des Main-Programms wird zur Entfernungsmessung eine Methode der Klasse „Bildanalyse“ aufgerufen:

```
int targetY = Bildanalyse::targetEntfernungY_horizont(objektSig);
```

Die Steuerung der Geschwindigkeit bzw. der Motorleistung erfolgt über ein PWM-Signal, dass direkt vom Raspberry bereitgestellt wird. Es wird davon ausgegangen, dass die Qualität des PWM-Signals für die Fahrtenregelung ausreichend ist:

Das Programm 3 zeigt einen größeren Ausschnitt der Methode „colorTarget(...)“. Dort sind auch die Alternativen zu finden, die das Auto anhalten. Die Geschwindigkeit, eigentlich die Antriebsleistung, wird in Abhängigkeit vom Abstand zum Zielobjekt geregelt.

Programm 3: Ausschnitt aus der Methode „colorTargetEasy(...)“

```
/// --- GESCHWINDIGKEIT
//      Liefert die Entfernung
int targetY = Bildanalyse::targetEntfernungY_horizont(objektSig);
// Kein Ziel in Sicht: HALT
if (CamPixy::blocks_copied == 0) {
    cout << "Kein Ziel erkannt." << endl;
    pulsweiteMotor = motorHalt - 1;
    cout << " Warte, bis Ziel erscheint." << endl;
}
//Kurz vor dem Ziel: HALT
if (targetY < CamPixy::BILDKANTE_S0) {    //fahre d
    pulsweiteMotor = motorHalt - 1;
    printf("\n vor dem Ziel HALT! ");
}
// -- Fahrt mit angepasster Geschwindigkeit --
if (targetY >= CamPixy::BILDKANTE_S0) {    //fahre d
    pulsweiteMotor = Regler::objektAbstandToPwmPulsweite(targetY, minimalerAbstand,
                                                         abAbstandMaximalPower, motorHalt, motorVorwärtsMax);
    printf("\n GO");
}

softPwmWrite(motorPwm, pulsweiteMotor);
printf("\n Hauptantrieb, motorPulsweite: %3d\n", pulsweiteMotor);
```

9. Fernwartung und Fernbedienung

Der Raspberry Pi3 verfügt über mehrere Schnittstellen, die eine kontaktlose Fernbedienung zulassen. Dazu gehören Bluetooth, WLAN und Infrarot.

Die Fernsteuerung des Raspberry gelingtst mit VNC (Virtual Network Computing), ein plattformunabhängiges Verfahren zur Steuerung eines über das Netzwerk erreichbaren Computers. Auf dem Raspberry ist VNC verfügbar. Kontrolliere die Installation wie folgt:

Raspberry – Internet – VNC Viewer

Wen die Software installiert ist, so sind im Grunde keine Änderungen erforderlich.

Das Nachinstallieren der VNC-Software oder auch anderer Software ist über das Menü wie folgt möglich:

Raspberry – Einstellungen – Add / Remove Software

Der Raspberry fungiert als Server. Auf dem Steuerrechner, dem Client, zum Beispiel mit dem Betriebssystem Windows, ist in der Regel VNC zu installieren. Die erforderliche Software finden wir für das betreffende Betriebssystem über den Link:

<https://www.realvnc.com/en/download/viewer/>

Nach der Installation ist die Verbindung zu konfigurieren. Das Main-Menü der VNC-Software bietet das Anlegen einer neuen Verbindung an (siehe Abbildung 15).

DriveGeany

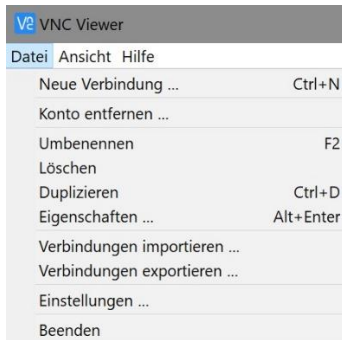


Abbildung 15: Main-Menü der VNC-Software

Die Verbindungseinstellungen sind in einem neuen Formular einzutragen. Dazu gehört die IP-Adresse des Raspberry und ein frei wählbarer Name. Anstelle der IP-Adresse kann alternativ der Hostname eingetragen werden (siehe Abbildung 16), zum Beispiel „raspberrypi“.

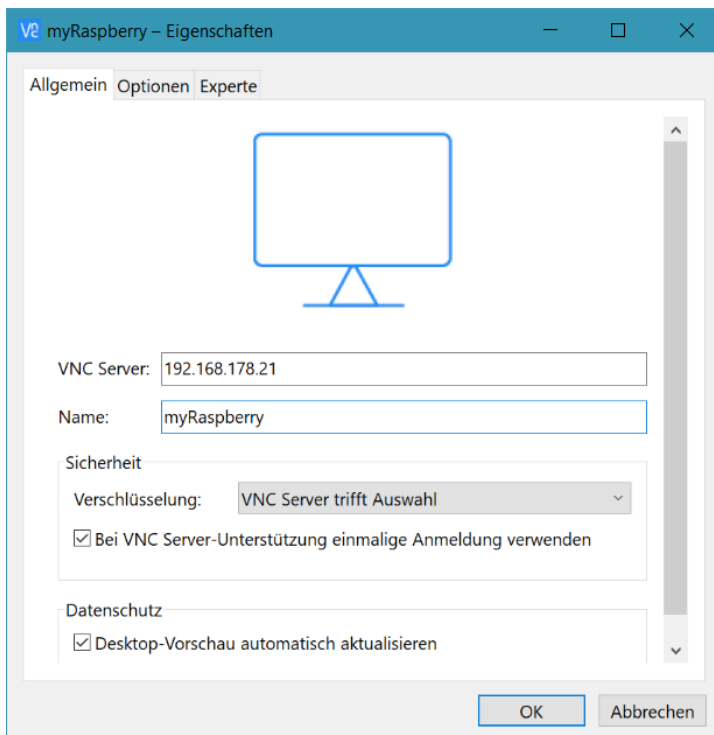


Abbildung 16: Verbindungseigenschaften

Ist die IP-Adresse unbekannt, so kann diese über das Terminalfenster mit „hostname -I“ ausgegeben werden.

```
pi@raspberrypi:~ $ hostname -I
192.168.178.31
pi@raspberrypi:~ $
```

Umfangreiche Informationen über die Konfiguration des Raspberry erhalten wir mit „ifconfig“. Die IP-Adresse der Rubrik „wlan0“ ist relevant. Zur besseren Orientierung sind der Befehl und die IP-Adresse in der folgenden Liste hellgrün dargestellt:

```
pi@raspberrypi:~ $ ifconfig
eth0      Link encap:Ethernet  Hardware Adresse b8:27:eb:b6:02:74
```



```
inet6-Adresse: fe80::dee2:1c3f:d7d4:84cb/64 Gültigkeitsbereich: Verbindung
UP BROADCAST MULTICAST MTU:1500 Metrik:1
RX packets:0 errors:0 dropped:0 overruns:0 frame:0
TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
Kollisionen:0 Sendewarteschlangenlänge:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

lo    Link encap:Lokale Schleife
      inet Adresse:127.0.0.1 Maske:255.0.0.0
      inet6-Adresse: ::1/128 Gültigkeitsbereich:Maschine
      UP LOOPBACK RUNNING MTU:65536 Metrik:1
      RX packets:577 errors:0 dropped:0 overruns:0 frame:0
      TX packets:577 errors:0 dropped:0 overruns:0 carrier:0
      Kollisionen:0 Sendewarteschlangenlänge:1
      RX bytes:46364 (45.2 KiB) TX bytes:46364 (45.2 KiB)

wlan0 Link encap:Ethernet Hardware Adresse b8:27:eb:e3:57:21
      inet Adresse:192.168.178.31 Bcast:192.168.178.255 Maske:255.255.255.0
      inet6-Adresse: fe80::6fb2:5636:ee78:4d1f/64 Gültigkeitsbereich:Verbindung
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metrik:1
      RX packets:26104 errors:0 dropped:0 overruns:0 frame:0
      TX packets:33650 errors:0 dropped:0 overruns:0 carrier:0
      Kollisionen:0 Sendewarteschlangenlänge:1000
      RX bytes:14166482 (13.5 MiB) TX bytes:27994683 (26.6 MiB)

pi@raspberrypi:~ $
```

Siehe auch: <https://www.elektronik-kompodium.de/sites/raspberry-pi/1906271.htm>

Sind die Verbindungsdaten korrekt so folgt die Frage nach dem Benutzer und dem Passwort auf dem Raspberry Pi (siehe Abbildung 17).

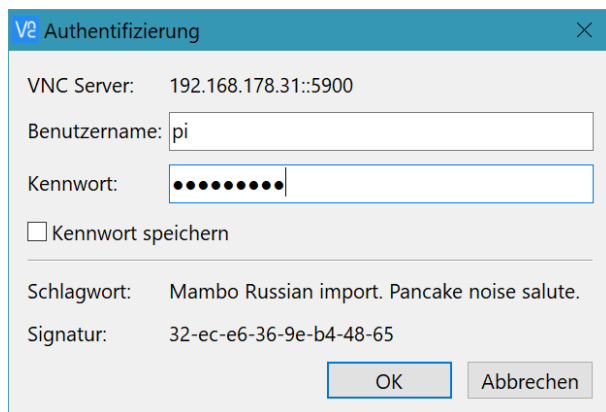


Abbildung 17: Anmeldung am Raspberry

Der Hostname und der Nutzernamen lassen sich auf dem Raspberry nachschlagen:

Raspberry – Raspberry-Pi-Konfiguration

Mit dem Passwort sieht das schon etwas anders aus. Sollten wir es vergessen haben, so muss ein neues gesetzt werden. Das Ändern des Passwortes kann über das Terminal gelingen:

```
pi@raspberrypi:~ $ sudo passwd pi
Geben Sie ein neues UNIX-Passwort ein:
Geben Sie das neue UNIX-Passwort erneut ein:
passwd: Passwort erfolgreich geändert
pi@raspberrypi:~ $
```

Eine Anleitung zum Zurücksetzen des Passwortes liefert auch der Link:

<https://www.pcwelt.de/ratgeber/Raspberry-Pi-Vergessenes-Passwort-zuruecksetzen-Sicherheit-9659550.html>

Nach erfolgreicher Verbindung haben wir den vollen Zugriff auf den Raspberry über einen entfernten Rechner. Nicht nur die Fernbedienung des Autos ist möglich sondern auch die Programmierung des Raspberry (siehe Abbildung 18).

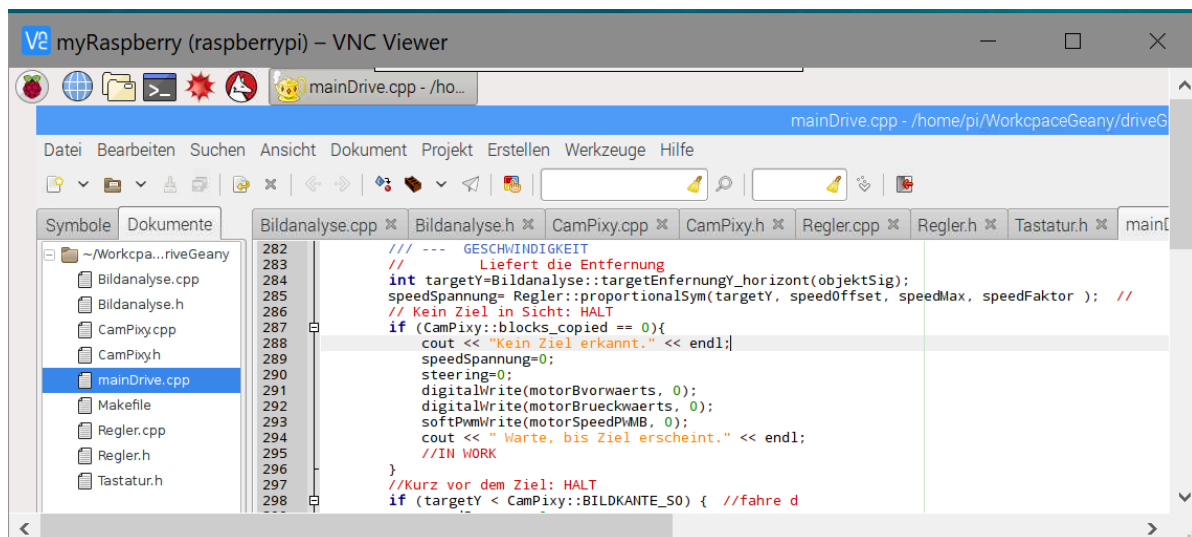


Abbildung 18: Raspberry Pi aus der Ferne

Anlagen

Schaltplan:

Electric Model / Elektro Modell

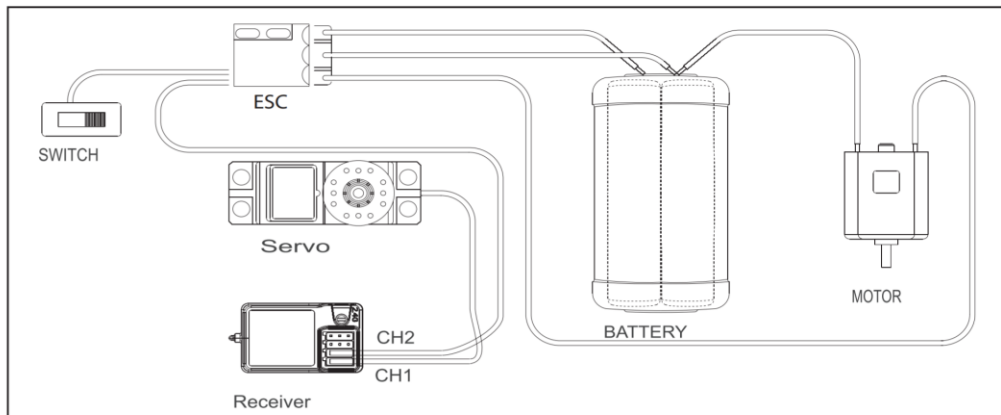


Abbildung 19: Schaltplan

[<https://cdn.billiger.com/dynimg/HFk2jxRZaE2iWIBZpPNAXDt4H9rycHjeBa9LTdXG2DAISvMs4TXAPKUf2y1aeFWuopzO8om4yzKYwt4MMoGW7k/ABSIMA-Buggy-AB1-2CH-RTR-AB12201-Bedienungsanleitung-393d20.pdf>]

Empfänger:

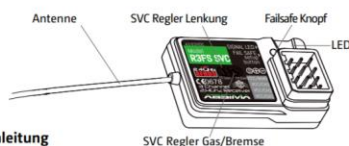
[http://www.absima.com/absima_uploads/Empf%C3%A4nger/3-Kanal-SVC/3%20Channel%20SVC%20RX%20german%20Kopie.pdf]



Einführung

R3FS SVC ist ein 3 Kanal Empfänger ausgestattet mit einem Kreisel. Zu den normalen Funktionen kommt das sogenannte Smart Vehicle Control (SVC) hinzu. Diese Funktion stellt sicher das Ihr Fahrzeug in der erwarteten Richtung fährt, egal ob auf holprigen / rutschigen Oberflächen oder in Kurven.

Empfänger Übersicht



Bedienungsanleitung

BIND Vorgang

1. Funkfernsteuerung einschalten. Überprüfen Sie das System in der Funke ob es auf AFHDS gestellt ist (für CR4T Fernsteuerungen).
2. Stellen Sie die Funke auf BIND Modus (für CR4T Fernsteuerungen).
3. Stellen Sie sicher das die Stromversorgung zum Empfänger nicht an ist.
4. Stecken Sie den BIND Stecker in den BIND/VCC Kanal ein. Dann schalten Sie den Regler oder eine andere Stromquelle ein. Die LED wird nun schnell rot blinken, das bedeutet das der Empfänger im BIND Modus ist.
5. Drücken und halten Sie nun den BIND Knopf an der Fernsteuerung und schalten diese ein. Die LED blinkt nun langsamer, der BIND Vorgang war erfolgreich und ist abgeschlossen.
6. Ziehen Sie das BIND Kabel wieder heraus und schalten Sie den Regler oder die Stromquelle aus. Dann stecken Sie den Regler oder die Stromquelle auf den BIND/VCC Kanal.
7. Überprüfen Sie die Servos ob alles funktioniert, wenn nicht dann wiederholen Sie den BIND Vorgang.

SVC Funktion

Diese Funktion hat 2 Verwendungen. Zum ersten verhindert die Funktion das Ausbrechen der Lenkung bei holprigen oder rutschigen Strecken. Zum zweiten wird die Geschwindigkeit reduziert während man in Kurven fährt um ein Ausbrechen zu verhindern und die Geschwindigkeit wird wieder erhöht wenn man aus der Kurve kommt.

Aktivieren/Deaktivieren der SVC Funktion

1. Regler oder Stromquelle am Empfänger ausschalten.
2. Stecken Sie das BIND Kabel in den Kanal 3 ein.
3. Halten Sie den Failsafe Knopf gedrückt und schalten Sie den Regler oder die Stromquelle ein. Die rote LED blinkt zweimal, SVC Funktion ist eingeschaltet.

Beachte:

Wenn die SVC Funktion aktiv ist, hat der Empfänger eine 2 Sekunden Startzeit. Während dieser Startzeit muss der Empfänger auf einer ebenen Fläche liegen.

R3FS SVC Anleitung

Reverse Funktion

Nach Einbau des Empfängers, drehen Sie das Fahrzeug um festzustellen ob die Räder sich in die richtige Richtung bewegen. Wenn Sie das Fahrzeug nach links drehen, müssen sich die Räder nach rechts drehen. Wenn Sie das Fahrzeug nach rechts drehen, müssen sich die Räder nach links drehen. Falls das nicht der Fall ist dann folgen Sie den folgenden Schritten:

1. Regler oder Stromquelle am Empfänger ausschalten.
2. Stecken Sie das BIND Kabel in den Kanal 3 ein.
3. Stecken Sie den Regler oder die Stromquelle am BIND/VCC Kanal ein. Die LED wird nun zweimal blinken und dann eine Pause haben. Das bedeutet das die Richtung geändert wurde.
4. Überprüfen Sie erneut die Reifen.

SVC Lenkung

SVC Lenkung korrigiert automatisch die Steuerung um das Fahrzeug wieder in seinen ursprünglichen Kurs zu bringen. Die Einstellung wird direkt am Empfänger über den ST.GAIN Knopf eingestellt. Der Korrekturwert liegt zwischen 0 - 100%.

SVC Gas

SVC Gas korrigiert das Gas während Kurvenfahrten ähnlich wie die Traktionskontrolle in echten Fahrzeugen. Sobald das Fahrzeug beginnt zu fahren verhindert das SVC das durchdrehen der Räder. Das bedeutet das die Räder auf rutschigen Untergrund weniger durchdrehen und das Sie eine schnellere Beschleunigung aus den Kurven haben.

Achtung: Wenn die SVC Funktion auf Kanal 1 und 2 aktiviert ist dann hat der Kanal 3 keine Funktion.

Failsafe Funktion

1. Stellen Sie sicher das die Funkfernsteuerung und der Empfänger gebunden sind und einwandfrei funktionieren.
 2. Stellen Sie den Gashebel und den Lenkhebel auf die gewünschte Position und drücken Sie dann den Failsafe Knopf. (Gashebel auf Bremsfunktion stellen)
- Die rote LED blinkt fünfmal, Failsafe Funktion ist nun aktiv. Wenn es zu einem Übertragungsabbruch kommt, stellt das Failsafe den Gashebel und die Lenkung auf die eingestellte Position.

Spezifikation

Kanäle	3
Frequenz	2.4055 - 2.475 GHz
Frequenz Band	140
RF Power	weniger als 20 dBm
2.4GHz System	AFHDS
Modell Typ	Auto/Boot
Code Typ	GFSK
Eingangsspannung	4.0 - 6.5V DC
Antenne Länge	26mm

Absima GmbH
Gibitzenhofstr. 127A / RG
90443 Nürnberg
Tel: 0911 / 65084130
www.absima.com

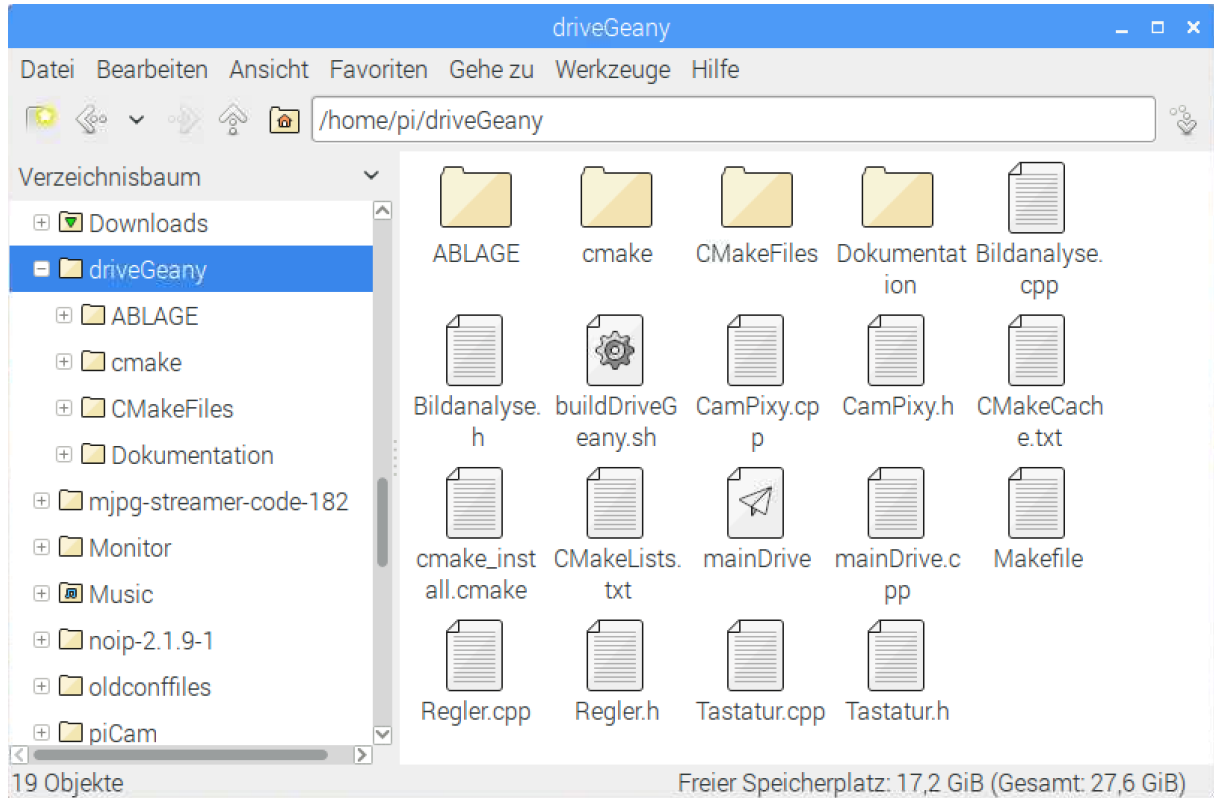
DriveGeany

Fahrtenregler:

<http://shop.absima.com/ABSIMA-TEAM-C-ERSATZTEILE/Absima-Ersatzteile/AB2-4-Buggy/Brushed-Regler-ECU1-70A-1-10-wasserdicht.htm?shop=absima&SessionId=&a=article&ProdNr=2100001&t=4187&c=13253&p=13253>

Programme

Programmstruktur



```
/** Bildanalyse
 *
 * Bearbeiter: Jonas Heinke (J.H)
 *
 * von 14.5.2017 bis: 28.6
 *
 * Analyse der Bilder und analyse der Bildparameter
 *
 * Ziel: Bessere Übersicht und bessere Wiederverwendbarkeit
 * */
// -- BIBLIOTHEKEN -----
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
```



```
#include <string.h>
#include "pixy.h"
#include <cmath>
//++j.h++
#include <iostream>

#include <wiringPi.h> // WiringPi-API einbinden
#include <cstdlib> // für system zum Aufruf externer Programme:
#include <fstream>
#include <termios.h>
#include <ctype.h>
#include <sys/select.h>

class Bildanalyse
{

public: //--> ggf private und GET-er verwenden

public:
    Bildanalyse();
    ~Bildanalyse();

public:
    static void help();
    static int targetXcenter(int objektSig);
    static int targetEnfernungY(int objektSig);
    static int targetEnfernungY_horizont( int objektSig);
    static int targetEnfernungY_geneigt(int y, int height);
};
```

```
#include "Bildanalyse.h"
#include "CamPixy.h"

Bildanalyse::Bildanalyse(){};
```

```
Bildanalyse::~Bildanalyse(){};

void printObjektKoordinatenX()
{
    //pixyPos(); //muss nicht sein, der jeweils letzte Datensatz wird gedruckt
    // Nur zur Kontrolle ein Ausgabe aller Koordnatenwerte und aller Obekte
    for(int index = 0; index != CamPixy::blocks_copied; ++index) {
        printf(" \n komponent: sig(index= %1d):%1d x:%4d y:%4d width:%4d height:%4d\n",
            index,
            CamPixy::blocks[index].signature,
            CamPixy::blocks[index].x,
            CamPixy::blocks[index].y,
            CamPixy::blocks[index].width,
            CamPixy::blocks[index].height);
    } //--//
}
//-----

/**
 * Beliebiger Hilfetext zur Abfrage durch den Entwickler oder Hinweise für den ANwender
 * */
void Bildanalyse::help()
{
    printf("\n Klasse Bildanalyse: Methoden zur Auwertung der Bilder und der
Bildparametern!\n\n");
};

/**
 * Liefert die Abweichung der Mitte eines Objektes zur Bildmitte
 * * in der x-Richtung
```

```

*/
int Bildanalyse::targetXcenter(int objektSig){
    CamPixy::pos();
    int index=objektSig -1;
    //T printObjektKoordinatenX();
    int xPos=CamPixy::blocks[index].x + (int) (CamPixy::blocks[index].width / 2);
    int xDelta=xPos - (int) (CamPixy::Xmax / 2);
    //T printf(" \n CamPixy::Xmax:  %4d  |  CamPixy::Xpos:  %4d  |  xDelta:  %4d\n",CamPixy::Xmax,xPos, xDelta);

    return xDelta; // int target
}

/**
 * STARK VEREINFACHTES ENFERNUNGSMODELL
 * Liefert die Entfernung zu einem spezifizierten Objektes
 * welches sich auf der Straße befindet.
 * Die Höhe der Kamera über Straße ist mit 10 cm festgelegt.
 * Verwendet wurde eine Geradengleichung. Das ist eine grobe Vereinfachung.
 * Vorteil: Parametrieren ist nicht erforderlich.
 * Gegebenenfalls ist eine Anpassung an das reale Modell erforderlich (Faktor 0.5).
 * */
int Bildanalyse::targetEntfernungY(int objektSig)
{
    CamPixy::pos();
    int index=objektSig -1;
    return CamPixy::BILDKANTE_S0 + (CamPixy::Ymax - CamPixy::blocks[index].y -
CamPixy::blocks[index].height) *0.5;
}

/**
 * VEREINFACHTES ENFERNUNGSMODELL
 * Liefert die Entfernung zu einem Objektes

```

```
* welches sich auf der Straße befindet.
* Randbedingung: Optische Achse verläuft exakt horizontal (parallel zur Straße)
* @param y          - y-Koordinate in Pixel der Objektoberkante in Pixel
* @param height     - Objektabbild (Höhe) in y-Richtung in Pixel (Objektunterkante= y + height)
* @return           - Entfernung des Objektes in mm (Objektunterkante) in Bezug zur Kamera
* */

int Bildanalyse::targetEntfernungY_horizont(int objektSig){
    CamPixy::pos();
    int index=objektSig -1;
    return (int) (CamPixy::Hcam * CamPixy::Scam / (CamPixy::blocks[index].y + CamPixy::blocks[index].height - CamPixy::Ymax/2));
}

/**
* Y-EBENEN-ENFERNUNGSMODELL
* Liefert die Entfernung zu einem Objektes
* welches sich auf der Straße befindet.
* Optische Achse kann auch um den Winkel Beta geneigt sein
* @param y          - y-Koordinate in Pixel der Objektoberkante in Pixel
* @param height     - Objektabbild (Höhe) in y-Richtung in Pixel (Objektunterkante= y + height)
* @return           - Entfernung des Objektes in mm (Objektunterkante) in Bezug zur Kamera
* */

int Bildanalyse::targetEntfernungY_geneigt(int y, int height){
    int yhmax=y+height - CamPixy::Ymax/2;
    double tanBeta= tan(CamPixy::Beta);
    return (int) (CamPixy::Hcam * (CamPixy::Scam + yhmax*tanBeta ) / (CamPixy::Scam *tanBeta + yhmax));
}
```

DriveGeany

```
/** Kamera Pixy
 * Bearbeiter: Jonas Heinke (J.H)
 * von 11.5.2017 bis:
 * Quellprogram der HELLO-PIXY zur Positionserkennung
 * modifiziert und in eine statische Klasse eingebaut
 * Ziel: Bessere Übersicht und bessere Wiederverwendbarkeit
 * */

// -- BIBLIOTHEKEN -----
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include "pixy.h"
//++j.h++
#include <iostream>
#include <wiringPi.h> // WiringPi-API einbinden
#include <cstdlib> // für system zum Aufruf externer Programme:
#include <fstream>
#include <termios.h>
#include <ctype.h>
#include <sys/select.h>

class CamPixy
{

#define BLOCK_BUFFER_SIZE 25
public:
// KONSTANTEN ZUM EINBAU DER Cam

/** @param BILDKANTE_S0 Abstand von der Kamera zum nahen Bildrand
static const int BILDKANTE_S0=15;
/** @param beta - Neigung der Kamera
```



```
static const int Beta=0;

/** @param yMax   - Maximale Pixelanzahl der Kamera in X-Richtung
static const int Xmax=320; //Matrixweite, in X

/** @param yMax   - Maximale Pixelanzahl der Kamera in Y-Richtung
static const int Ymax=200; //Matrixhöhe, in Y

/** @param Hcam   - Höhe der Kameraachse in mm in Richtung der zu
detektierenden Objektunterkante

static const int Hcam=15; //Anbauhöhe der Kamera

/** @param Scam   - Kamerakonstante in Pixel, Abbildungskonstante (kann
experimentell ermittelt werden)

static const int Scam=250; //Kamerakonstante, muß ermittelt werden


public: //--> ggf private und GET-er verwenden

static struct Block blocks[BLOCK_BUFFER_SIZE];
static int blocks_copied;
static int frame;


struct objectPosition{
    int entfernung;
    int orientierung;
};

static objectPosition objPos;


public:
    CamPixy();
    ~CamPixy();


public:
    static void help();
    // static void handle_SIGINT(int unused);
```

```
static int init();  
static int pos();  
static int pos(int objektSig);  
static void close();  
};
```

```
#include "CamPixy.h"  
  
struct Block CamPixy::blocks[BLOCK_BUFFER_SIZE];  
int CamPixy::blocks_copied;  
int CamPixy::frame=0;  
  
CamPixy::CamPixy(){};  
CamPixy::~~CamPixy(){};  
  
/**  
 * Beliebiger Hilfetext zur Abfrage durch den Entwickler oder Hinweise für den ANwender  
 * */  
void CamPixy::help()  
{  
    printf("\n Klasse CamPixy: Methoden zur Kamerasteuerung und zur  
Objektanalyse\n\n");  
};  
  
/**  
 * Initialisiert die Pixy-Kamera. Das geschieht nur einmal,  
 * zum Beispiel im Hauptprogramm.  
 * */  
int CamPixy::init()  
{  
  
// -- lokale Deklarationen ---  
int index;  
int pixy_init_status;
```

```
char    buf[128];

//-----Vorbereitung -----
// signal(SIGINT, handle_SIGINT); // Catch CTRL+C (SIGINT) signals //
printf("Drive Crt, libpixyusb Version: %s\n", __LIBPIXY_VERSION__);
pixy_init_status = pixy_init(); // Connect to Pixy //
//try
{
    if(!pixy_init_status == 0) // Was there an error initializing pixy? //
    {
        // Error initializing Pixy //
        printf("pixy_init(): ");
        pixy_error(pixy_init_status);
        return pixy_init_status;

    }
}
//catch (const std::exception& e){}

// Request Pixy firmware version //
{
    uint16_t major;
    uint16_t minor;
    uint16_t build;
    int    return_value;
    return_value = pixy_get_firmware_version(&major, &minor, &build);
    // Ausnahmebehandlungen

    if (return_value) {
        // Error //
        printf("Failed to retrieve Pixy firmware version. ");
        pixy_error(return_value);
        return return_value;
    }
}
```

```
} else {
    // Success //
    printf(" Pixy Firmware Version: %d.%d.%d\n", major, minor, build);
}

}

return 0; //frame

}

/**
 * Liefert alle Positionswerte der erkannten Objekte
 * */
int CamPixy::pos(){
    // Wait for new blocks to be available // DAS SIND DIE WICHTIGSTEN ZEILEN
    !!!!!!!!!!!!!!!!!!!!!!!

    while(!pixy_blocks_are_new());// && run_flag

    blocks_copied = pixy_get_blocks(BLOCK_BUFFER_SIZE, &blocks[0]);// Get blocks from
    Pixy

    if(blocks_copied < 0) {
        // Error: pixy_get_blocks //
        printf("pixy_get_blocks(): ");
        pixy_error(blocks_copied);
    }

    frame++;
return frame;
}

int CamPixy::pos(int objektSig){
    // Wait for new blocks to be available // DAS SIND DIE WICHTIGSTEN ZEILEN
    !!!!!!!!!!!!!!!!!!!!!!!

    while(!pixy_blocks_are_new());// && run_flag
```

```
    blocks_copied = pixy_get_blocks(BLOCK_BUFFER_SIZE, &blocks[0]); // Get blocks from Pixy
```

```
    //IN WORK: Entfernung und orientierung des Objektes berechnen
```

```
    if(blocks_copied < 0) {  
        // Error: pixy_get_blocks //  
        printf("pixy_get_blocks(): ");  
        pixy_error(blocks_copied);  
    }  
    frame++;  
    return frame;  
}
```

```
/**
```

```
 * Schließt die Kamera
```

```
 * */
```

```
void CamPixy::close(){  
    pixy_close();  
}
```

```
// Tastatur.h

//

/**
 * Regler.h
 * Regler für Steuerung und Verfahrensgeschwindigkeit
 * Bearbeiter: Jonas Heinke J.H
 * von: 26.04.2017 bis:
 * https://de.wikipedia.org/wiki/Regler#P-Regler\_.28P-Anteil.29
 */

// -- BIBLIOTHEKEN -----
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include "pixy.h"

//++j.h++

#include <iostream>

#include <wiringPi.h> // WiringPi-API einbinden
#include <cstdlib> // für system zum Aufruf externer Programme:
#include <fstream>
#include <termios.h>
#include <ctype.h>
#include <sys/select.h>
#include <termios.h>
#include <sys/types.h>
#include <signal.h>

class Regler
{
public:
// C++C+++++
static bool run_flag;
```

```
public:
    Regler();
    ~Regler();
private:
public:
    struct steeringByte
    {
        bool b0;
        bool b1;
        bool b2;
        bool b3;
    };

    static void help();
    static int proportionalSym(int eingang, int offset, int extrem);
    static int proportionalSym(int eingang, int offset, int extrem, double verstaerkung);

    static double toServoSpannung(int eingang, int eingangMax, double konstante);

    static int toServoPulsweite(int eingang);
    static double toServoPulsweite(int eingang, int max);
    static double toServoPulsweite(int eingang, int max, double grenzUnten, double
    grenzOben);

    static steeringByte toServoOutParallel(int eingang);
    static steeringByte toServoOutParallel(int eingang, int EingangSpannweite);
    //-----

};
```



```
#include "Regler.h"

#include <math.h>

Regler::Regler(){};
Regler::~~Regler(){};

/**
 * Hinweise für Entwickler und Anwender
 * */
void Regler::help()
{
    printf("\n Klasse Regler: Methoden zur Regeln");
};

//----- LOCAL -----
/**
 * define signum function
 * */
template<class T>
int signum(T val)
{
    if(val < T(0))
    {
        return T(-1);
    }
    else
    {
        return T(+1);
    }
}

//-----
```

```
/** *****  
 * symmetrischer Proportionalregler für Antrieb  
 *  
 *  
 * */  
int Regler::proportionalSym(int eingang, int offset, int extrem)  
{  
    int vz=signum(eingang);  
    double ausgang= vz*offset + eingang;  
    if (abs(ausgang) > extrem) ausgang= vz*extrem;  
    return (int) ausgang;  
}  
int Regler::proportionalSym(int eingang, int offset, int extrem, double verstaerkung)  
{  
    int vz=signum(eingang);  
    double ausgang= vz*offset + verstaerkung * eingang;  
    if (abs(ausgang) > extrem) ausgang= vz*extrem;  
    return (int) ausgang;  
}  
  
/**  
 * Verschiebt einen Eingangs-Ganzwert(positive, negativ) in einen natürlichen Ausgangswert  
(positiver Zahlenbereich) und kalibriert auf den  
 * erforderlichen Ausgangswert mit  
 * */  
double Regler::toServoSpannung(int eingang, int eingangMax, double konstante)  
{  
    double servoSpannung = (double) (eingang + eingangMax) * konstante;  
    //Verschiebung auf Anschluss und Empfindlichkeit  
    return servoSpannung;  
}
```

```
/**
 * Bestimmt die Pulsweite in Abhängigkeit von der Lenkbewegung
 *
 * */
int Regler::toServoPulsweite(int eingang)
{
    // int servoPulsweite = 1500 + (1000/200)* eingang;
    int servoPulsweite = 50 + (int) (0.5 * eingang);
    //Verschiebung auf Anschluss und Empfindlichkeit
    return servoPulsweite;
}

/**
 * Bestimmt die Pulsweite in Abhängigkeit von der Lenkbewegung
 *
 * */
double Regler::toServoPulsweite(int eingang, int max)
{
    double m=(double) max/200;
    double n=(double) max/2;

    double servoPulsweite= n + m * (double) eingang;

    return servoPulsweite;
}

/**
 * Bestimmt die Pulsweite in Abhängigkeit von der Lenkbewegung
 * Und berücksichtigt den mechanischen Stellbereich grenzUnten-->grenzOben
 * Verhindert Überbelastung des Servoantriebes
 *
 * */
double Regler::toServoPulsweite(int eingang, int max, double grenzUnten, double grenzOben)
{

```

```
double m=(double) max/200;
double n=(double) max/2;

double servoPulsweite= n + m * (double) eingang;
if (servoPulsweite<grenzUnten) servoPulsweite=grenzUnten;
if (servoPulsweite>grenzOben) servoPulsweite=grenzOben;
return servoPulsweite;
}

//Makro zum herauslösen eines Bits aus dem unsigned char
// unsigned char entspricht = Byte
#define getBit(x,y) ((x) & (1 << ((y)-1))) != 0

/**
 * Übergibt Steuerdaten an Nucleo (einfache eigene Parallelschnittstelle)
 * Spannweite der Eingangsdaten beträgt 200, von -100 bis + 100
 * */
Regler::steeringByte Regler::toServoOutParallel(int eingang){
    return Regler::toServoOutParallel(eingang, 200);
}

/**
 * Spannweite ist hier beliebig allerdings symmetrisch zum Ursprung.
 * */
Regler::steeringByte Regler::toServoOutParallel(int eingang, int EingangSpannweite)
{
    Regler::steeringByte steeringOut;
    int servoParallel= (int) ((15.0/2.0) + (eingang *15.0/EingangSpannweite) + 0.5);
    //T: Testausgabe
    printf("\n servoParallel: %4d", servoParallel );
    unsigned char parallel= (unsigned char) servoParallel;
    steeringOut.b0= getBit(parallel,1);
    steeringOut.b1= getBit(parallel,2);
}
```

DriveGeany

```
        steeringOut.b2= getBit(parallel,3);
        steeringOut.b3= getBit(parallel,4);
//T: Testausgabe
        printf("\n output: %2d %2d %2d %2d",steeringOut.b3 , steeringOut.b2, steeringOut.b1,
steeringOut.b0);
        return steeringOut;
}
```

```
// Tastatur.h
//
/**
 * Tastatur.h
 * Tastatursteuerung
 * Bearbeiter: Jonas Heinke J.H
 * von: 15.04.2017 bis:
 */
// -- BIBLIOTHEKEN -----
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include "pixy.h"
//++j.h++
#include <iostream>
#include <wiringPi.h> // WiringPi-API einbinden
#include <cstdlib> // für system zum Aufruf externer Programme:
#include <fstream>
#include <termios.h>
#include <ctype.h>
#include <sys/select.h>
#include <termios.h>
#include <sys/types.h>
#include <signal.h>

//
https://android.googlesource.com/platform/frameworks/native/+/master/include/android/keycodes.h
// http://www.thelinuxdaily.com/2010/05/grab-raw-keyboard-input-from-event-device-node-devinputevent/
```

```
class Tastatur
{

public:

// C++C+++++
    static bool run_flag;

public:
    Tastatur();
    ~Tastatur();

private:
    static int cbreak(int fd);

public:
    static void help();
    static void handle_SIGINT(int unused);
    static int getKeyCode(void);
    static int kbhit(void);
    static int getch(void);
};
```

```
#include "Tastatur.h"
```

```
Tastatur::Tastatur(){};
```

```
Tastatur::~~Tastatur(){};
```

```
/**
```

```
 * Hinweise für Entwickler und Anwender
```

```
 * */
```

```
void Tastatur::help()
```



```
{
    printf("\n Klasse Tastatur: Methoden zur Abfrage der Tastatur\n");
};
/**
 * Abfrage der Tastenkombination Strg+C
 * zur Steuerung der Schleife (zum Beenden)
 * (Dadurch kann die Schleife kontinuierlich laufen)
 * */
bool Tastatur::run_flag = true;
void Tastatur::handle_SIGINT(int unused)
{
    run_flag = false;
}

/** IN WORK
 * Liefert den Tastencode
 *
 * */
int Tastatur::getKeyCode(void){
    // blablabla
    return 32;
}

/**
 * C Hilfsfunktionen zur Tastaturabfrage ++++++
 * Stoppt ein Programm bis zur betätigung einer Taste
 *
 *      http://www.linuxquestions.org/questions/programming-9/pausing-the-screen-44573/#post220112
 * */
int Tastatur::kbhit(void)
{
    struct timeval tv;
    fd_set read_fd;
```

```
tv.tv_sec=0;
tv.tv_usec=0;
FD_ZERO(&read_fd);
FD_SET(0,&read_fd);
if(select(1, &read_fd, NULL, NULL, &tv) == -1) return 0;
if(FD_ISSET(0,&read_fd)) return 1;
return 0;
}

/**
 * Tastencode direkt abfragen ohne mit ENTER zu bestätigen
 * Endlich mal eine Quelle die brauchbar ist:
 *
 * http://openbook.rheinwerk-verlag.de/c\_von\_a\_bis\_z/016\_c\_ein\_ausgabe\_funktionen\_006.htm#mja821f1d0ab158bed792eccc5f30e3f84
 * */
static struct termios new_io;
static struct termios old_io;

/** Funktion schaltet das Terminal in den cbreak-Modus:
/* Kontrollflag ECHO und ICANON auf 0 setzen
/* Steuerzeichen: Leseoperation liefert 1 Byte VMIN=1 VTIME=1
* */
int Tastatur::cbreak(int fd) {
    /*Sichern unseres Terminals*/
    if((tcgetattr(fd, &old_io)) == -1)
        return -1;
    new_io = old_io;
    /* Wir verändern jetzt die Flags für den cbreak-Modus. */
    new_io.c_lflag = new_io.c_lflag & ~(ECHO|ICANON);
    new_io.c_cc[VMIN] = 1;
    new_io.c_cc[VTIME]= 0;

    /*Jetzt setzen wir den cbreak-Modus*/
```

```
    if((tcsetattr(fd, TCSAFLUSH, &new_io)) == -1)
        return -1;
    return 1;
}
/**
 * Liefert den ASCII-Code des Zeichens der gedrückten Taste.
 * Das Programm wird fortgesetzt ohne auf Return zu warten.
 * */
int Tastatur::getch(void)
{
    int c;
    if(cbreak(STDIN_FILENO) == -1) {
        printf("Fehler bei der Funktion cbreak ... \n");
        exit(EXIT_FAILURE);
    }
    c = getchar();
    /* alten Terminal-Modus wiederherstellen */
    tcsetattr(STDIN_FILENO, TCSANOW, &old_io);
    return c;
}
```

```
/**
 * mainDrive.cpp
 *
 * Bearbeiter: Jonas Heinke J.H
 * von: 22.04.2017 bis: 28.6.2017
 * Programmteile: manuelle Steuerung, Objektverfolgung
 *
 */

/*
 * HINWEISE
 * Compiliert mit Hilfe der "CMakeList.txt" zum Auffinden der Bibliotheken
 * --> Erstellen --> Make
 *
 */

// -- BIBLIOTHEKEN -----
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include "pixy.h"
//++j.h++
#include <iostream>
#include <wiringPi.h> // WiringPi-API einbinden
//Quelle: http://docs.biiicode.com/raspberrypi/examples/wiringpi.html
#include <softServo.h> //für PWM
// https://raspberrypi.stackexchange.com/questions/61043/raspberry-pi-software-driven-pwm-using-c
#include <softPwm.h> //Pulsweitenmodulation
#include <time.h>
```

```
#include <cstdlib> // für system zum Aufruf externer Programme:
#include <fstream>
#include <termios.h>
#include <ctype.h>
#include <sys/select.h>
#include <fcntl.h>

// my J.
#include "Tastatur.h"
#include "CamPixy.h"
#include "Bildanalyse.h"
#include "Regler.h"

/// --- GPIOs -----
// -----Motor, ungenutzt -----
/**
int motorA1=28;      //GPIO20
int motorA2=29;      //GPIO21
int motorPWMA=25;    //GPIO26 Mot 1 pwm
* */
//-----Hauptmotor -----
int motorBvorwaerts=22;    //GPIO6
int motorBrueckwaerts=23;    //GPIO13
int motorSpeedPWMB=26;    //GPIO12 Mot 2 pwm
//-----Lenkung über Raspberry-----
//int servoPWMP27=27 ;    //GPIO17 servo (3)
// -----Lenkung über Nucleo -----
/// Definiert eine eigene Paralelschnittstelle mit 4 Datenbits und einem Steuerbit
int servoNucleoB0= 0;
int servoNucleoB1= 1;
int servoNucleoB2= 2;
int servoNucleoB3= 3;
int servoNucleoBcrt= 6;
```

```

using namespace std;

/**
 * Kontrollausgabe
 * */
void printObjektKoordinaten()
{
    for(int index = 0; index != CamPixy::blocks_copied; ++index) {
        printf(" \n komponent: sig(index= %1d):%1d x:%4d y:%4d width:%4d height:%4d\n",
            index,
            CamPixy::blocks[index].signature,
            CamPixy::blocks[index].x,
            CamPixy::blocks[index].y,
            CamPixy::blocks[index].width,
            CamPixy::blocks[index].height);
    }
}

/**
 * Ausgabe der Zielposition in horizontaler und vertikaler Ebene
 * */
void printTarget(int targetX, int targetY)
{
    printf(" \n DX:%4d Pixel y:%4d cm \n",targetX,targetY);
}

/**
 * AUSGAENGE UND EINGÄNGE VEREINBAREN
 * WIRINGPi-Zuordnung
 * Wichtig: Hier wird das WiringPi Layout verwendet
 */
int GPIOinit(int max)
{

```

```

    // Starte die WiringPi-API (wichtig)
    if (wiringPiSetup() == -1) { cout<<"Setup wiring pi failed"; return 1;}

    // -- Motor

    /** ungenutzt

    pinMode(motorA1, OUTPUT);
    pinMode(motorA2, OUTPUT);
    pinMode(motorPWMA, OUTPUT);
    motorA1= 0; //Polung
    motorA2= 0;

    softPwmCreate(motorPWMA,0,100); // Spannweite des PWM Signals
    softPwmWrite(motorPWMA, 0);    // Initialisierung
    * */

    // --- Hauptmotor
    pinMode(motorBvorwaerts, OUTPUT);
    pinMode(motorBrueckwaerts, OUTPUT);
    pinMode(motorSpeedPWMB, OUTPUT);
    softPwmCreate(motorSpeedPWMB,0,max); //PWM-Kanal, Speed , von Max (hier in %)

    // --- Raspberry-Lenkung
    /// pinMode(servoPWMP27, OUTPUT);
    /// softPwmCreate(servoPWMP27,0,100); // Spannweite des PWM-Signals
    /// softPwmWrite(servoPWMP27, 0);    // Anfangswert ??

    // --- Nucleo-Lenkung - Positionsdaten und Steuerbit
    pinMode(servoNucleoB0, OUTPUT);
    pinMode(servoNucleoB1, OUTPUT);
    pinMode(servoNucleoB2, OUTPUT);
    pinMode(servoNucleoB3, OUTPUT);
    pinMode(servoNucleoBcrt, OUTPUT);
    return 1;
}

//----- MANUELLE DYNAMISCHE STEUERUNG -----
/// Steuerung
/// Jonas J.H

```



```

/// 17.4.2017
/// http://raspberrypiguide.de/howtos/raspberry-pi-gpio-how-to/
/// g++ gpio_Control.cpp -o gpio_Control -l wiringPi
int manuelDynamisch (){
    cout << "Raspberry manuell dynamisch !" << endl;
    // Anfangswerte
    char merkerSpeed='0';
    char merkerSteering='0';
    int speed=0;           //
    int speedOffset=40; //Anfahrwiderstand überwinden
    int speedMax=100;  // -100 <= v <= 100, Negative Werte für Rückwärts
    int steering=0;
    int steeringMax=8;  /// Spannweite / 2
    // int steeringRange=2*steeringMax;
    GPIOinit(speedMax);
    // Tastencode
    char key='x';
    int asciiCode=120;
    merkerSpeed='s';

    // Nichtabweisschleife
    do{
        cout << " \nEingabe: " << endl;
        cout << " [w],[Up]   : vorwärts   | [s],[Down]: zurück     | [x],[Space]: halt\n
[d],[Right]: nach rechts | [a],[Left]: nach links | [f],[Pos1] : mitte\n [e],[Ende]: zum Hauptmenü "
<< endl;

        asciiCode = Tastatur::getch();
        char character = (char) asciiCode;
        cout << "asciiCode: " << asciiCode << endl;
        cout << "character      : " << character << endl;
        cout << "speed          : " << speed << " %" << endl;
        cout << "steering speed      : " << steering << " %" << endl;

        // -- Lenkung ----

```

```

        if (asciiCode == 100 || asciiCode == 67) { // nach rechts d || right C
            //if (merkerSteering != 'r') steering=0;
            steering -=1;
            if (steering < -steeringMax) steering=-steeringMax;
            merkerSteering='r';
        }
        if (asciiCode == 97 || asciiCode == 68) { //lnach links a || left
            //if (merkerSteering != 'l') steering=0;
            steering +=1;
            if (steering > +steeringMax) steering=+steeringMax;
            merkerSteering='l';
        }

        cout << "speed      : " << speed << " %" << endl;

        /// Übergibt die Steuerdaten an den Nucleo.
        /// Der Nucleo definiert das PWM-Signale korrekt für den Servomotor der
Lenkung
        Regler::steeringByte
steeringOut=Regler::toServoOutParallel(steering,2*steeringMax); #####
        // Das Steuerbit "servoNucleoBcrt" sorgt dafür, dass zuerst alle Datenbits
zusammengestellt werden,
        // bevor diese wirksam werden.
        digitalWrite(servoNucleoBcrt,0);
        digitalWrite(servoNucleoB0,steeringOut.b0);
        digitalWrite(servoNucleoB1,steeringOut.b1);
        digitalWrite(servoNucleoB2,steeringOut.b2);
        digitalWrite(servoNucleoB3,steeringOut.b3);
        digitalWrite(servoNucleoBcrt,1);

        /// -- Fahrt ---
        if (asciiCode == 119 || asciiCode==65) { // beschleunigen w || Up
            if (merkerSpeed == 's') speed=speedOffset; //damit nicht der komplette
Intervall durchlaufen werden muss

```

```

        speed +=1;
        if (speed>speedMax) speed=speedMax;
        merkerSpeed='v';
    }
    if (asciiCode == 115 || asciiCode==66) {    //negativ beschleunigen s || Down
        if (merkerSpeed =='s') speed=-speedOffset;
        speed -=1;
        if (speed<-speedMax) speed=-speedMax;
        merkerSpeed = 'r';
    }
    if (speed>0){ //vorwärts
        digitalWrite(motorBvorwaerts, 1);
        digitalWrite(motorBrueckwaerts, 0);
        softPwmWrite(motorSpeedPWMB, speed);
    }
    if (speed<0){ //rückwärts
        digitalWrite(motorBvorwaerts, 0);
        digitalWrite(motorBrueckwaerts, 1);
        softPwmWrite(motorSpeedPWMB, abs(speed));
    }

    if (asciiCode == 120 || speed==0 ) { // Stop the care x
        digitalWrite(motorBvorwaerts, 0);
        digitalWrite(motorBrueckwaerts, 0);
        softPwmWrite(motorSpeedPWMB, 0);
        merkerSpeed='s';
    }
    if (asciiCode == 32) { //Bremsen
        digitalWrite(motorBvorwaerts, 0);
        digitalWrite(motorBrueckwaerts, 0);
        digitalWrite(motorSpeedPWMB, 1);
        softPwmWrite(motorSpeedPWMB, 0);
        speed=0; //falls space zum Bremsen
    }

```

```

        merkerSpeed='s';

    }

} while(asciiCode != 101 && asciiCode != 70) ; //e, F, Ende
/// -- Alles ausschalten --
// ----- Hauptmotor ---
digitalWrite(motorBvorwaerts, 0);
digitalWrite(motorBrueckwaerts, 0);
digitalWrite(motorSpeedPWMB, 0);
softPwmWrite(motorSpeedPWMB, 0);
softPwmStop(motorSpeedPWMB);
// ----- Lenkungs ---
/// softPwmWrite(servoPWMP27, 0);
/// softPwmStop(servoPWMP27);
return 6;
}

/**
 * autonome Steuerung: Objektverfolgung,
 * verfolge ein Ziel, z.B. Gelb
 * */
int colorTarget(int objektSig){
    GPIOinit(100);
    int steering=0;
    int steeringOffset=0;
    double steeringFaktor=0.8; // Dämpft die Lenkbewegung
    int steeringMax= 70; //Spannweite = 2 * SteeringMax
    double speedSpannung;
    int speedOffset=10;
    int speedMax=40;
    double speedFaktor=0.5;

```

```

//Verstärkung der Lenkausschläge
steering=0;
do{

    // RICHTUNG: Abweichung der Fahrtrichtung zum Objekt
    int targetX=Bildanalyse::targetXcenter(objektSig);

    ///Verbessert die Lenkregelung (Negativ "-targetX" für Rechts-Links-Tausch)
    double steering= Regler::proportionalSym(-targetX, steeringOffset=0,
steeringMax, steeringFaktor);

    /// Übergibt die Steuerdaten an den Nucleo.

    /// Der Nucleo definiert das PWM-Signale korrekt für den Servomotor der
Lenkung
    Regler::steeringByte steeringOut=Regler::toServoOutParallel(steering,
2*steeringMax);

    // Das Steuerbit sorgt dafür, dass zuerst alle Datenbits zusammengestellt
werden,

    // bevor diese wirksam werden.
    digitalWrite(servoNucleoBcrt,0);
    digitalWrite(servoNucleoB0,steeringOut.b0);
    digitalWrite(servoNucleoB1,steeringOut.b1);
    digitalWrite(servoNucleoB2,steeringOut.b2);
    digitalWrite(servoNucleoB3,steeringOut.b3);
    digitalWrite(servoNucleoBcrt,1);


    /// --- GESCHWINDIGKEIT
    // Liefert die Entfernung
    int targetY=Bildanalyse::targetEntfernungY_horizont(objektSig);
    speedSpannung= Regler::proportionalSym(targetY, speedOffset, speedMax,
speedFaktor ); //

    // Kein Ziel in Sicht: HALT
    if (CamPixy::blocks_copied == 0){
        cout << "Kein Ziel erkannt." << endl;
        speedSpannung=0;
        steering=0;
        digitalWrite(motorBvorwaerts, 0);
        digitalWrite(motorBrueckwaerts, 0);
    }
}

```

```

        softPwmWrite(motorSpeedPWMB, 0);
        cout << " Warte, bis Ziel erscheint." << endl;
        //IN WORK
    }
    //Kurz vor dem Ziel: HALT
    if (targetY < CamPixy::BILDKANTE_S0) { //fahre d
        speedSpannung=0;
        steering=0;
        digitalWrite(motorBvorwaerts, 0);
        digitalWrite(motorBrueckwaerts, 0);
        softPwmWrite(motorSpeedPWMB, 0);
        printf("\n ZURÜCK und suche Objekt");
    }
    // -- Fahrt mit angepasster Geschwindigkeit --
    if ( targetY > CamPixy::BILDKANTE_S0) { //fahre d
        digitalWrite(motorBvorwaerts, 1);
        digitalWrite(motorBrueckwaerts, 0);
        softPwmWrite(motorSpeedPWMB, speedSpannung);
        //printf("\n GO");
    }
    //printObjektKoordinaten();
    printTarget(targetX, targetY);
    // sleep(1); //! Pause: zur besseren Kontrolle
    // cout << " ..... " << endl;
    signal(SIGINT, Tastatur::handle_SIGINT); // Catch CTRL+C (SIGINT) signals
//
    } while(Tastatur::run_flag); //true/false
    printf("\n\nSchließen mit e\n");
    int asciiCode;
    do{
        asciiCode =Tastatur::getch(); //Weiter mit Tastendruck
    } while(asciiCode != 101); //warte solange bis Taste e
    //Alles ausschalten

```

```

digitalWrite(motorBvorwaerts, 0);

digitalWrite(motorBrueckwaerts, 0);

digitalWrite(motorSpeedPWMB, 0);

softPwmWrite(motorSpeedPWMB, 0);

softPwmStop(motorSpeedPWMB);

return 1;

}

/// ++++++
/// ++++++ M A I N ++++++
int main(int argc, char *argv[])
{
    Tastatur::help();
    CamPixy::help();
    Bildanalyse::help();

    CamPixy::init(); //Pyxi initialisieren
    CamPixy::pos(); //Koordinatenabfrage
    printObjektKoordinaten();// TESTAUSGABE -----
    char wahl='m';
    do{
        cout << " \n Steuerungsverfahren auswählen: " << endl;

        cout << " \n m manuelle Steuerung\n f Folge dem gelben Objekt \n z zwischen grüner und
roter Absperrung \n e Ende" << endl;

        cin >> wahl; // Eingabe muss mit ENTER bestätigt werden //
wahl=getKeyChar(); //das auch nicht

        switch (wahl){
            case 'm' : manuelDynamisch(); break;
            case 'f' : colorTarget(1); break; //Parameter ist das Farbobjekt sig 1
            case 'z' : /* in work*/ break;

        }

        cout << "....." << endl;

    }

    while (wahl!='e');

```



```
CamPixy::close();  
cout << " Ende des Programms!" << endl;  
return 0;  
}
```

Build

Programme zum Erstellen der Applikation

Starte buildDriveGeany.sh und es wird eine neue CMakeCache.txt erstellt. Neue Einstellungen erfordern diese Aktion. Wichtige Einstellungen stehen in der CMakeList.txt.

buildDriveGeany.sh

```
#!/bin/bash  
  
#TARGET_BUILD_FOLDER=./build  
# mkdir $TARGET_BUILD_FOLDER  
# mkdir $TARGET_BUILD_FOLDER/driveGeany  
# cd $TARGET_BUILD_FOLDER  
cmake ./CMakeLists.txt  
  
make
```

CMakeLists.txt

```
cmake_minimum_required (VERSION 2.8)  
project (mainDrive CXX)  
  
set (CMAKE_MODULE_PATH "${CMAKE_SOURCE_DIR}/cmake" )  
  
# Add sources here... #
```

```
add_executable (mainDrive mainDrive.cpp Tastatur.h Tastatur.cpp CamPixy.h CamPixy.cpp
Bildanalyse.h Bildanalyse.cpp Regler.h Regler.cpp)
```

```
# libpixyusb should always come before libboost and libusb #
```

```
target_link_libraries (mainDrive pixyusb)
```

```
target_link_libraries (mainDrive wiringPi)
```

```
find_package ( libpixyusb REQUIRED )
```

```
find_package ( libusb-1.0 REQUIRED )
```

```
find_package ( Boost 1.49 COMPONENTS thread system chrono REQUIRED)
```

```
target_link_libraries (mainDrive ${Boost_LIBRARIES})
```

```
target_link_libraries (mainDrive ${LIBUSB_1_LIBRARY})
```

```
file(STRINGS "cmake/VERSION" LIBPIXY_VERSION)
```

```
add_definitions(-D__LIBPIXY_VERSION__="${LIBPIXY_VERSION}")
```

```
include_directories (src
```

```
    include
```

```
    ../../common
```

```
    ${Boost_INCLUDE_DIR}
```

```
    ${LIBUSB_1_INCLUDE_DIRS})
```