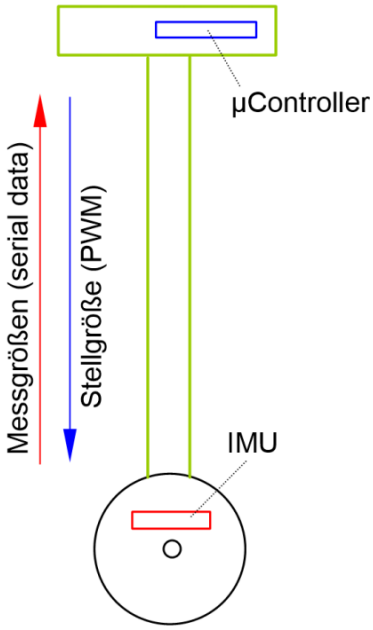
Schnittstelle

Segway-IMU ←→Controller



Aufgabe des Controllers

Die Regelung zum Balancieren des Segways erfolgt im µController. Dieser erhält dazu von der IMU (Inertial Measurement Unit) des Segways alle notwendigen Sensordaten, berechnet daraus die notwendige Spannung für die Motoren und erzeugt entsprechende PWM-Signale.
Diese Berechnung wird im Controller *periodisch* ausgeführt, als praktikable Periodendauer wird 10…15ms vorgeschlagen.

Aufgabe der Segway-IMU

Die IMU erfüllt folgende Aufgaben:

- Steuern des Kalibriervorgangs nach dem Einschalten
- Erfassen der Sensordaten (insbesondere Neigungswinkel, Radgeschwindigkeit und Weg der beiden Räder)
- Überwachen der Batterie-Spannung und des mittleren Stromverbrauchs
- Überwachung des HANDSHAKE-Signals

Startvorgang

Nach dem Einschalten der Versorgungsspannung müssen sich die Sensoren kalibrieren können. Dazu wird der Segway in senkrechter Position so gehalten, dass neben der ROTEN LED zusätzlich die ORANGE LED dauerhaft zum Leuchten kommt. Nach ca. 1 Sekunde sind die Sensoren kalibriert und die ROTE und die ORANGE LED verlöschen, die GRÜNE LED beginnt zu leuchten.
Mit dem Verlöschen wird das RESET-Signal von der Segway-IMU für ca. 0,5ms auf LOW geschaltet. Jetzt kann der Controller mit der Regelung des Segways beginnen. Nach dem RESET-Signal reagieren die Motoren auf entsprechende Steuersignale vom Controller.

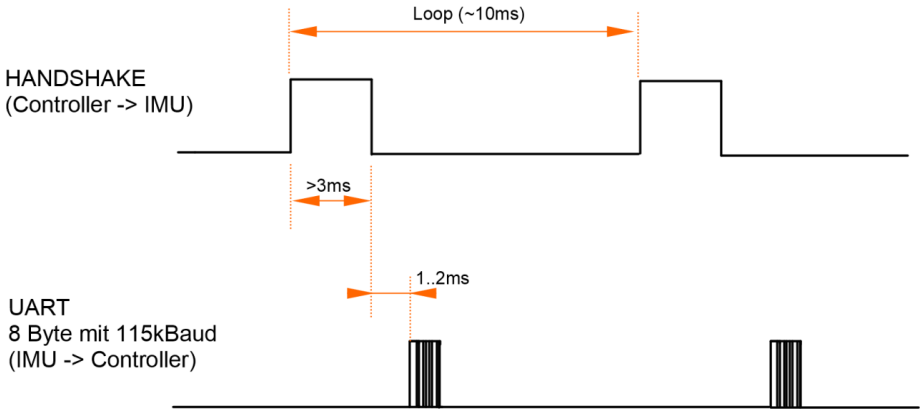
Signale von der Segway-IMU zum Controller

- RESET. OpenDrain, intern über Widerstand 100k an +3,3V gelegt
  nach erfolgreicher Kalibrierung der Sensoren wird RESET über ca. 0,5ms auf LOW geschaltet
- UART. Serielle Daten von den Sensoren (TTL, 3.3V). Nach der fallenden Flanke des HANDSHAKE-Signals werden 8 BYTE Daten mit 115200 Baud gesendet

Signale vom Controller zur Segway-IMU

- HANDSHAKE . Nachdem auf HANDSHAKE ein **LOW-HIGH-LOW** -Pulse von wenigstens 3ms gelegt wird, reagiert die Segway-IMU mit dem Senden der aktuellen Messdaten über die UART. (Erfolgt über 40ms kein HANDSHAKE, schaltet die IMU den Segway in Shutdown.)
- PWM_A . Pulsweiten-moduliertes Signal zum Ansteuern des rechten Motors. Die Pulsweite entspricht der angelegten Motorspannung, (0…100% entspricht 0V…12V).
- DIR_A . Drehrichtung des rechten Motors, HIGH entspricht vorwärts, LOW rückwärts
- PWM_B .Pulsweitenmoduliertes Signal für den linken Motor.
- DIR_B . Drehrichtung des linken Motors, HIGH entspricht vorwärts, LOW rückwärts.

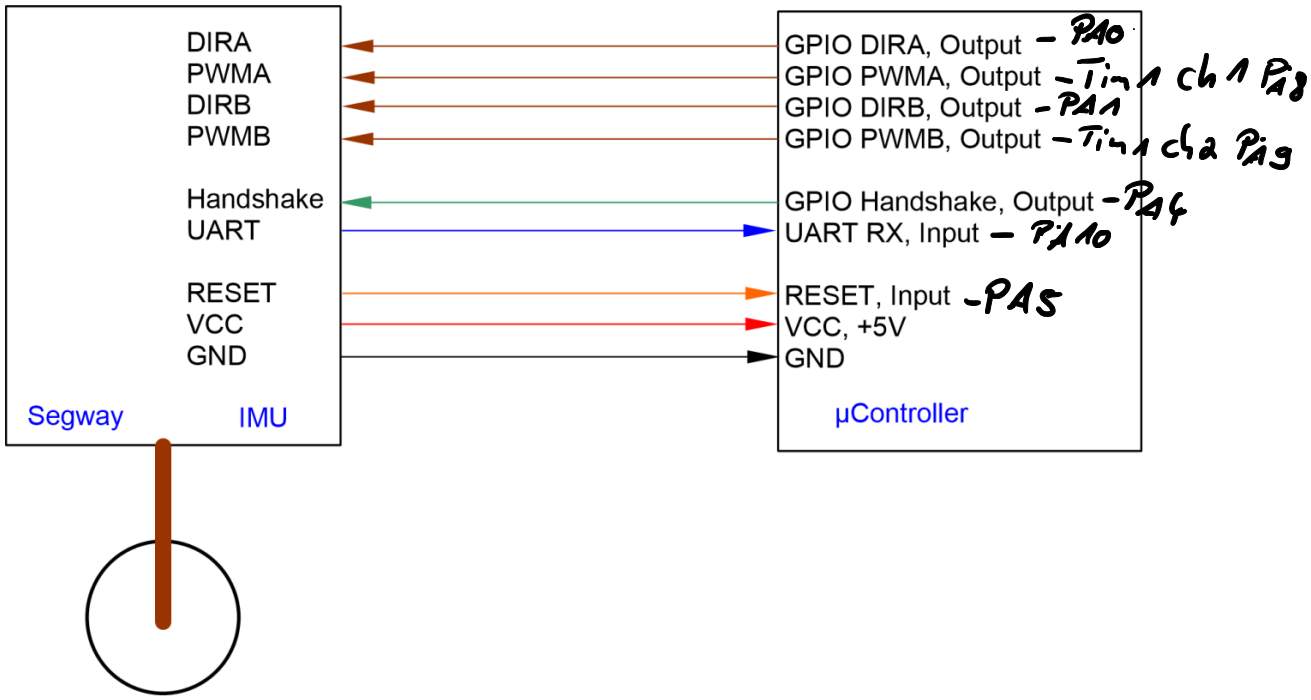
Für das Balancieren des Segways können beide Motoren parallel angesteuert werden, d.h. beide Motoren bekommen die gleichen PWM- und DIR-Signale.



UART-Datenformat UART[0]...UART[7]

Winkel Phi [°]:	**Phi**=(UART[0]+256* UART[1]-32768)/1024;
zurückgelegter Weg s_r [m], rechter Motor:	**Sr**=(UART[2]+256* UART[3]-32768)*0.00452;
Geschwindigkeit v [m/sek] der Räder (rechts):	**Vist**=(UART[4]+256* UART[5]-32768)/10000;
zurückgelegter Weg s_l [m], linker Motor:	**Sl**=(UART[6]+256* UART[7]-32768)*0.00452;

Anschluss eines µControllers an den Segway



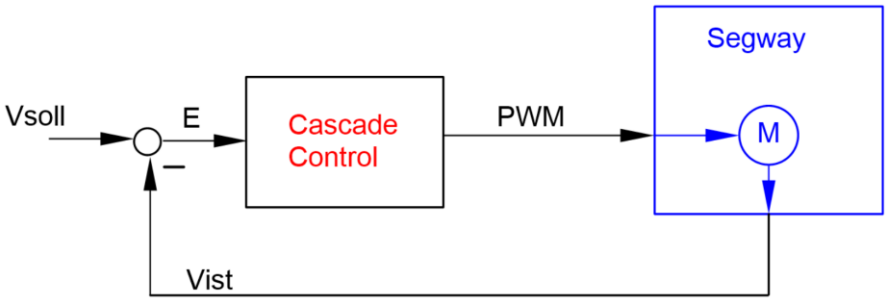
Programmiervorschlag

Die Regelschleife muss periodisch abgearbeitet werden. Dazu kann ein hochlaufender TIMER eingesetzt werden, der regelmäßig abgefragt wird und nach Ablauf der Periodendauer (z.B. 10 ms) wieder zurückgesetzt wird. Alternativ kann ein Interrupt benutzt werden.

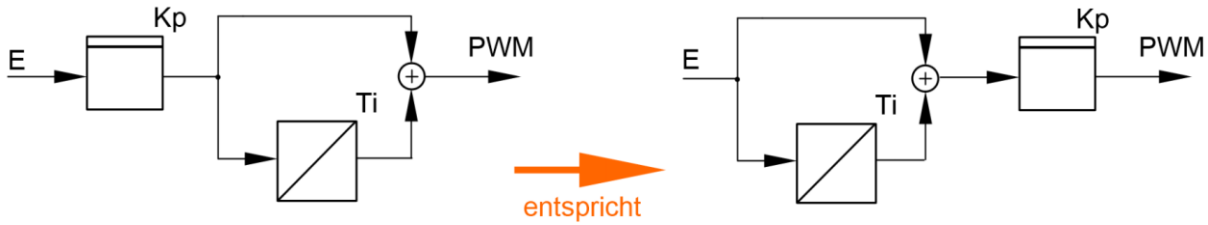
INIT Initialisierung UART
Initialisierung TIMER
Initialisierung PORTs
Deklaration von Variablen und Konstanten
LOOP TIMER auf 0 setzen
PORT Handshake auf HIGH setzen
TIMER abfragen, warten, bis TIMER 3 ms anzeigt
PORT Handshake auf LOW setzen
Abfrage UART: warten, bis Daten angekommen sind
Berechnen der Stellgröße, Ausgabe von PWMs und DIRs
TIMER abfragen, warten, bis TIMER 10ms anzeigt
LOOP wiederholen

Motoransteuerung mit Kaskaden-Regler

Zur Vereinfachung der Modellbeschreibung wurde vorausgesetzt, dass die Motoren des Segways sehr kräftig sind und unabhängig von wirkenden Drehmomenten die gewünschte Drehzahl annehmen. Um dieses Verhalten zu erreichen, wird ein **Kaskaden-Regler** eingesetzt. Dieser steuert den Motor so an, dass die Differenz **E** aus gewünschter Drehzahl (hier mit **Vsoll** bezeichnet) und tatsächlicher Drehzahl (**Vist**) gegen Null geht.



Der Kaskadenregler kann als PI-Regler ausgeführt werden und besitzt folgende Struktur:



Für die Programmierung der Regelschleife lässt sich das I-Glied mit dem periodischen Aufsummieren der Regelabweichung E in einem Integrationsregister **Ireg** abbilden:

E = Vsoll – Vist;
Ireg = Ireg + E;
PWM = (E + (Ireg*(Ta/Ti)))*Kp;

Ta: Periodendauer der Regelschleife, z.B. 10ms
Ti: Integrationskonstante, beim Segway Verzögerung des Motors: Ti=100ms

Kp: Verstärkung des Kaskadenreglers: Kp=(n/0.216)*(256/12)
(bei PWM-Auflösung von 8 Bit)
n: Beschleunigungsfaktor des Kaskadenreglers, hier n=2

Ausgabe an den Segway:
if (PWM>=0) {Pin_DIRA=HIGH; Pin_DIRB=HIGH;} else {Pin_DIRA=LOW; Pin_DIRB=LOW;};
if PWM<0){PWM=abs(PWM);};
if (PWM>255) {PWM=255;};
Pin_PWMA=PWM;
Pin_PWMB=PWM;

Beispielcode für Arduino Mega (nur Umsetzung der Motor-Regelung)

```
const byte DIRA_Pin = 2;
const byte DIRB_Pin = 3;
const byte PWMA_Pin = 4; //uses TIMER4
const byte PWMB_Pin = 5; //uses TIMER3
const byte HANDSHAKE = 6;
const byte RESET = 7;
const byte TestPin= 10;
const float Ta=0.01; //Periodendauer
const float Ti=0.1; //Integrationszeitkonstante (=Motorzeitkonstante)
const float Kp=(2/0.216)*(256/12); //198

float UART[8]; //UART[0]...UART[7] = 8 BYTE
float PHI, SR, SL, Vist;

//Variablen zur Drehzahlregelung
float Vsoll=1.0, E, Ireg=0.0, U; //hier wird die Sollgeschw. festgelegt
byte PWM;

void setup() {
pinMode(DIRA_Pin, OUTPUT);
pinMode(DIRB_Pin, OUTPUT);
pinMode(PWMA_Pin, OUTPUT);
pinMode(PWMB_Pin, OUTPUT);
pinMode(HANDSHAKE, OUTPUT);
pinMode(RESET, INPUT);
pinMode(TestPin, OUTPUT);
Serial.begin(115200);
Serial1.begin(115200);

//Timer5 konfigurieren
TCCR5A = 0;// set entire TCCR5A register to 0
TCCR5B = B0000010;// Prescaler 8, TIM5 Tp=0.5us
TCCR5C = 0;// set entire TCCR5C register to 0

//digitalen Ausgänge preset
digitalWrite(DIRA_Pin, LOW);
digitalWrite(DIRB_Pin, LOW);
digitalWrite(PWMA_Pin, LOW);
digitalWrite(PWMB_Pin, LOW);
digitalWrite(HANDSHAKE, LOW);

//RESET --|___|--- abwarten
while(digitalRead(RESET)==HIGH); //wartet, bis RESET auf LOW wechselt
while(digitalRead(RESET)==LOW); //wartet, bis RESET wieder HIGH ist
```

```
}

void loop() { //Schleife wird periodisch alle 10ms ausgeführt

while(TCNT5 < 20000); //wartet, bis Timer 10ms erreicht hat
TCNT5 = 0; //Timer neu starten

//Handshake auslösen
digitalWrite(HANDSHAKE, HIGH);
while(TCNT5 < 6000);
digitalWrite(HANDSHAKE, LOW);

//UART auslesen
for (byte i=0; i<8; i=i+1) {UART[i]=Serial1.read();}
Vist=(UART[4]+256*UART[5]-32768)/10000; //Geschwindigkeit des rechten
Rads[m/sek]

E=Vsoll-Vist;
Ireg=Ireg+E;
U=(E+(Ireg*(Ta/Ti)))*Kp;

if (U>=0) {digitalWrite(DIRA_Pin,HIGH); digitalWrite(DIRB_Pin,HIGH);}
else {digitalWrite(DIRA_Pin,LOW); digitalWrite(DIRB_Pin,LOW);};

U=abs(U);
if (U>255){U=255;};
PWM=byte(U);
analogWrite(PWMA_Pin, PWM);
analogWrite(PWMB_Pin, PWM);
}
```