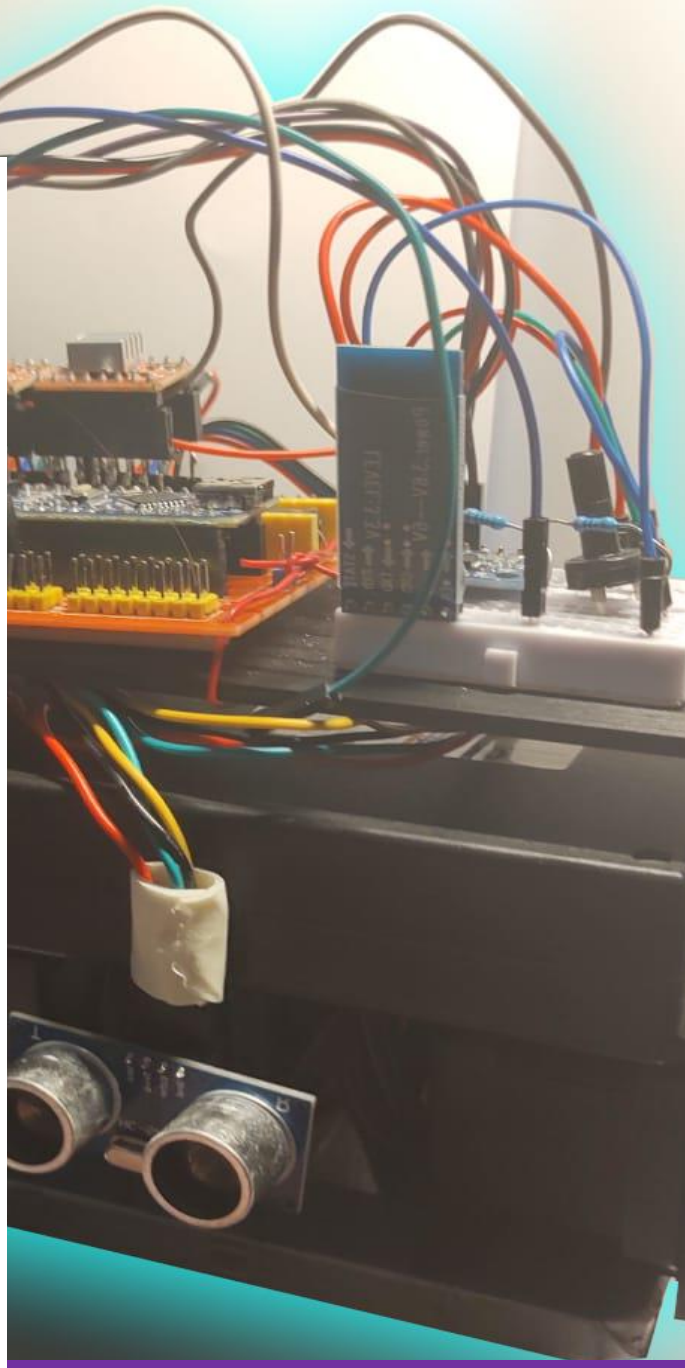
Die Software von Segways und Raketen



2021

OTH-Regensburg
Verfasst von: Strohmaier Lukas

PID-Control eines inversen Pendels

Am Beispiel eines selbst-balancierenden Roboters

Während man im Alltag wohl eher selten versuchen wird, ein inverses Pendel zu balancieren, findet das Konzept viele Anwendungen von Segways bis hin zu den aufrecht-landenden Raketen der Firma SpaceX. Als inverses Pendel wird ein Pendel mit einem Schwerpunkt oberhalb seiner Achse, welches sich in seinem höchsten Punkt in einer instabilen Ruhelage befindet bezeichnet. Solche Vorrichtungen benötigen diverse Antriebsmöglichkeiten, um sich selbst, trotz Störungen dieses Systems, in ihrer Ruhelage zu halten. Zur Veranschaulichung kann man sich einen Wagen vorstellen an dessen Oberseite ein Stab mit einer Masse am oberen Ende lose befestigt ist. Alternativ kann man sich einen zweirädrigen Roboter vorstellen, dessen Aufgabe es ist sich aufrecht zu halten. Im Zuge des „Praktikum Messtechnik 2“ an der OTH-Regensburg durfte ich mit meinem Team solch einen Roboter konstruieren. Neben der Hauptaufgabe sich aufrecht zu erhalten, kann unser Projekt Hindernisse erkennen und sich fortbewegen.

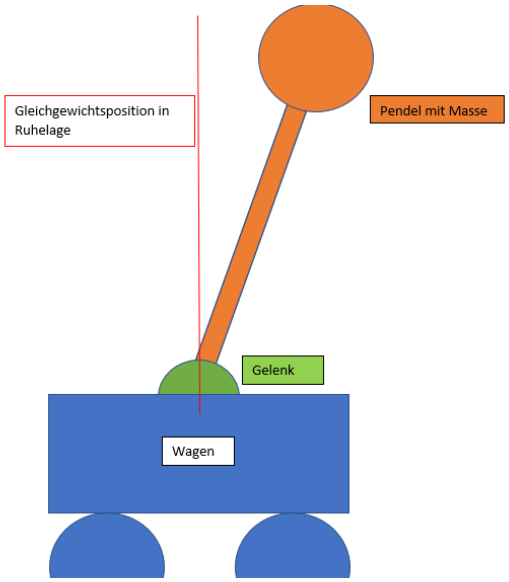


Abbildung 1: Modell eines inversen Pendels

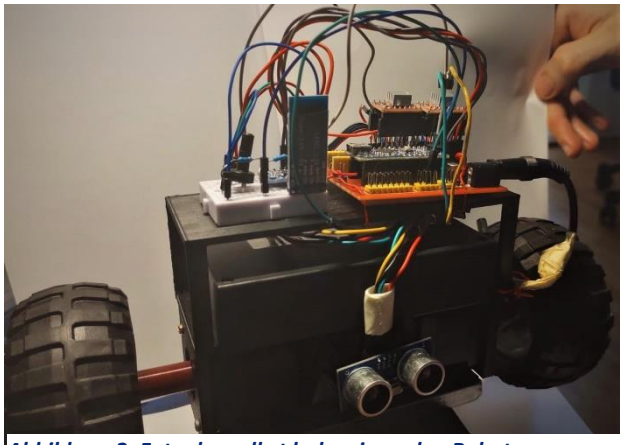


Abbildung 2: Foto des selbst balancierenden Roboters

Im Folgenden wird die Software hinter diesem Projekt repräsentativ für die Steuerung eines inversen Pendels erklärt.

Da dieses Projekt auf zwei Arduino Nanos basiert, wurde als Programmierungsumgebung die Arduino eigene IDE mit den jeweiligen Standard-Bibliotheken gewählt. Als Programmiersprache wurde C gewählt. Außerdem wurde eine MPU6050-Bibliothek verwendet, um die Kommunikation mit dem verbauten Gyroskop zu ermöglichen. Programme welche aus dieser IDE stammen haben grundsätzlich einen universell anwendbaren Aufbau aus setup() und loop(). Die setup-Funktion, welche als Einstiegspunkt des Programmes festgelegt ist, führt den darin enthaltenen Code ein einziges Mal zu Beginn aus. Am Ende dieser Funktion springt das Programm in den loop. Die loop-Funktion kann ebenfalls als while(true) {} angesehen werden, also eine Schleife welche endlos wiederholt wird, bis es zu einer externen Unterbrechung kommt.

Das Konzept des Programms lässt sich knapp erklären. Alle fünf Millisekunden wird der Winkel ausgelesen und verrechnet. Dieser Wert wird dann genutzt, um die Drehzahl und Richtung der Motoren zu kontrollieren. Außerdem lässt sich der Roboter bewegen, wenn er über die UART Schnittstelle bestimmte Bytes empfängt. Diese Steuerung wird aber nur ausgeführt, falls der zweite Arduino mit den HCSR-04 Ultraschall Modulen kein Hindernis detektiert. Anschließend wird noch detaillierter auf diesen Prozess eingegangen.

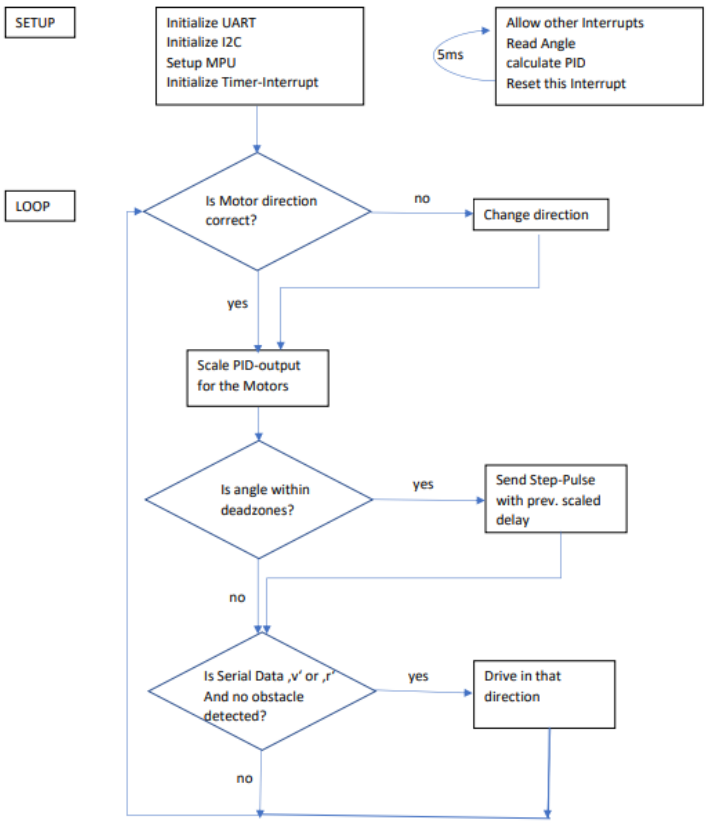


Abbildung 3: Flussdiagramm der Software (größer im Anhang)

In der hier vorgestellten Ausführung werden nach diversen Includes, Defines und globalen Variablen im Setup zunächst die GPIO- Pins (general purpose input/output), welche direkt in diesem Programm angesteuert werden auf OUTPUT gestellt. Dies geschieht unter Verwendung der, von der IDE bereitgestellten, Funktion pinMode(). Standardmäßig sind diese bereits so kalibriert, man sollte sich aber nicht darauf verlassen. Anschließend wird die UART Schnittstelle des Arduinos mit einer Baud-Rate von 9600 gestartet um die Kommunikation mit dem HC-05 Bluetooth Modul zu ermöglichen. Danach werden durch

die, von der MPU6050 Bibliothek, zur Verfügung gestellten Funktionen die I^2C Schnittstelle initialisiert und das Gyroskop kalibriert. Außerdem wird ein Startwert für den Winkel vom Gyroskop gelesen.

```
pinMode(pinStep_1,OUTPUT);
pinMode(pinStep_2,OUTPUT);
pinMode(pinDirection_1,OUTPUT);
pinMode(pinDirection_2,OUTPUT);
pinMode(pinObst_1,INPUT);
pinMode(pinObst_2,INPUT);

Serial.begin(9600);

while(!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))
{
  delay(500);
}
mpu.calibrateGyro();
mpu.setThreshold(3);

readMPU();
```

Für die Drehrichtungen der Motoren wird ebenfalls ein Wert festgelegt, da ohne diesen Startwert die Abfrage zur Richtungsänderung nicht immer wie gewollt reagieren würde.

```
PORTD=PORTD | 0x08;
```

Diese Schreibweise kann etwas verwirrend wirken, ist aber eigentlich relativ simpel. Das PORTD Register korrespondiert zu den GPIO-Pins D0 bis D7. Unsere Richtungssteuernden Pins sind D2 und D3 und die Step-Pins, welche einen Puls an die A4988-Motor-Treiber Module senden, um die Schrittmotoren einen Schritt zu bewegen, sind D5 und D6. Da diese alle durch das PORTD-Register angesteuert werden, ist es aus Laufzeitgründen vor allem bei Mehrfachänderungen zu empfehlen sich an diese Schreibweise zu gewöhnen.
Zu guter Letzt wird ein Timer-Interrupt initialisiert, um so periodisch alle fünf Millisekunden neue Messwerte auslesen zu können.

PortD	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Belegung		Step Pin 2	Step Pin 1		Direction Pin 2	Direction Pin 1		
PORTD = PORTD | 0x08 (0x08 = 00001000)								
Nach Ausführung	0	0	0	0	1	0	0	0

Abbildung 4: Einfache Visualisierung des PORTD Registers

Nach alledem springt das Programm in die Loop Funktion. Hier wird zunächst geprüft, ob die Richtung der Motoren bereits richtig ist. Dazu wird abgefragt, ob im PORTD Register bereits die richtigen Bits für den derzeitigen Überhangwinkel gesetzt sind. Ist dies nicht der Fall so werden die jeweils Richtigen, also Bit drei oder vier gesetzt. Hier ist zu bedenken, dass niemals beide Bits gleich gesetzt sein dürfen, da die Motoren gegenüberliegend montiert sind.

```
if((angle<0) && ((PORTD & 0x0C)==0x08))
{
  PORTD = PORTD & ~0x0C;
  PORTD =  PORTD | 0x04;
}
else if((angle>0) && ((PORTD & 0x0C)==0x04))
{
  PORTD = PORTD & ~0x0C;
  PORTD = PORTD | 0x08;
}
```

Anschließend wird der Rückgabewert, welcher in der Interrupt-Routine aufgerufenen PID-Funktion indirekt proportional skaliert, um so auf das passende Delay für die Step-Impulse zu kommen. Außerdem wird dieser Wert nochmal auf den passenden Wertebereich eingeschränkt, um bei extremen Ausschlägen des Gyroskops weder die Schrittmotoren, noch die jeweiligen A4988 Schrittmotor-Treiber Module zu beschädigen.

```
motDelay=scal(abs(PID_out),0,30*Kp,2900,800);
motDelay=constrain(motDelay,800,2900);
```

Um die Motoren nun passend anzusteuern wird überprüft, ob der derzeitige Winkel zwischen ±1 und ±45 Grad liegt. Diese Deadzones wurden gewählt um unnötiges Zittern um den Nullpunkt zu vermeiden und um den Roboter im Falle eines Sturzes nicht gegen die nächste Wand fahren zu lassen. Ist diese Bedingung erfüllt, so wird ein Step-Impuls an beide Motoren geschickt und anschließend die zuvor errechnete Verzögerung durchgeführt.

```
if (((angle>1) && (angle<45)) || ((angle<-1) && (angle>-45)))
{
  PORTD =PORTD | 0x60;
  delayMicroseconds(2);
  PORTD =PORTD & ~0x60;
  delayMicroseconds(motDelay);
}
```

Das Nächste Code-Segment dient zur Steuerung. Als erstes wird ein Byte aus dem Seriellen Puffer ausgelesen. Ist dieses Byte ein ‚v' für vorwärts oder ein ‚r' für rückwärts wird die jeweilige Richtung eingestellt und es wird eine bestimmte Anzahl an Step-Impulsen gesendet, um das Gefährt in diese Richtung zu Bewegen. Die Richtungs- und Bewegungsansteuerung erfolgt nach dem gleichen Muster wie zuvor.

```
recByte=Serial.read();
if(recByte=='v' || recByte=='r')
{
  if(recByte=='v' && !digitalRead(pinObst_1))
  {
```

Die für all das verwendeten Funktionen wurden bisher als gegeben angesehen. Auf diese und die Interrupt-Routine wird hier nochmal eingegangen.

Zunächst wird die Initialisierung des Timer-Interrupts betrachtet. Um Fehler zu vermeiden, werden während der Einrichtung andere Interrupts deaktiviert. In den TCCR Registern wird der Prescaler konfiguriert. Im OCR Register wird der Vergleichswert festgelegt und das TIMSK Register aktiviert den Interrupt.

```
void initTimer()
{
  cli();

  TCCR1A =0;
  TCCR1B=0;
  OCR1A=9999;
  TCCR1B |= 0b00001010;
  TIMSK1 |= 0b00000010;

  sei();
}
```

Um das MPU6050 Modul auszulesen, wird eine modifizierte Version einer, von der MPU6050 Bibliothek zur Verfügung gestellten, Funktion verwendet. Da diese Funktion periodisch alle fünf Millisekunden aufgerufen wird, wird dieser Wert als timeStep festgelegt. Außerdem wird für dieses Projekt nur eine Dimension des Gyroskops benötigt.

```
void readMPU()
{
  Vector readings = mpu.readNormalizeGyro();
  angle = angle + (double)readings.YAxis*timeStep;
}
```

Um den PID Wert des Winkels zu berechnen, wurde eine rudimentäre Funktion geschrieben. Diese speichert immer den letzten Wert und berechnet daraus den (P)proportionalen, (I)integralen und (D)differenziellen Teil.

```
void calcPid(double inpAngle)
{
  PID_P=setpoint-inpAngle;                    //calculate P
  PID_I += PID_P*5;                           //calculate I
  PID_D = (PID_P - PID_P_old)/5;              //calculate D

  PID_out= Kp*PID_P + Ki*PID_I + Kd*PID_D;    //calculate PID

  PID_P_old=PID_P;                            //safe PID for next iteration
}
```

Die scal-Funktion ist eine modifizierte Version der Arduino-Standardfunktion map(). Sie wird verwendet, um Werte eines Wertebereichs auf einen anderen zu skalieren. Sie wurde bearbeitet, um mit double Werten zu arbeiten.

```
double scal(double x, long in_min, long in_max, long out_min, long out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Die Interrupt Service Routine wird wie bereits erwähnt alle fünf Millisekunden aufgerufen. Ihre Aufgabe ist es so periodisch den aktuellen Winkel des Roboters auszulesen und auf diesen dann die PID-Funktion anzuwenden.

```
ISR(TIMER1_COMPA_vect)
{
  sei();                                          //allow other intterupts...
  TIMSK1 = TIMSK1 & ~0x02;                        //...but disable this one

  readMPU();
  calcPid(angle);

  TCNT1=0;                                        //reset timer value
  TIMSK1 = TIMSK1 | 0x02;                         //reactivate timer1 interrupt
}
```

Der Roboter hat zusätzlich einen zweiten Arduino Nano verbaut, welcher die Ultraschall Sensoren kontrolliert. Dieser Prozess wurde ausgelagert, da er verhältnismäßig sehr zeitintensiv ist und die Ausbalancierung des Roboters aufhalten würde. Hier wird lediglich ein Puls auf den Trigger des Sensor-Moduls gesendet und durch die Dauer des empfangenen Signals über die Schallgeschwindigkeit die Distanz bestimmt. Solange er Hindernisse erkennt, welche weniger als 20 Zentimeter entfernt sind, werden die jeweiligen Leiterbahnen zum Steuernden Arduino auf HIGH gesetzt, um ihm zu signalisieren, dass in diese Richtung keine Fortbewegung möglich ist.

Falls noch genaueres Interesse besteht, können die Artikel zum Aufbau, einer genaueren Beschreibung der Sensorik und eine detaillierte Beschreibung der gestengesteuerten Fernbedienung, sowie der komplette Sourcecode und der Link zur MPU6050-Bibliothek hier gefunden werden:

https://github.com/Stromi1011/SelfBalancingRobot_Woita

Anhang: Flussdiagramm