

Aufgabe 5:

Problemstellung:

Der ASURO soll einer schwarzen Linie mit Hilfe eines PID-Reglers folgen.

Problemlösung:

Beschreibung der Steuerung:

Beim Initialisierungsvorgang werden die ADC-Werte (linker und rechter Sensor) der schwarzen Linie gespeichert. Ob sich der ASURO außerhalb der Strecke befindet, kann über diese Werte festgestellt werden. Wird das Verlassen der Strecke festgestellt, fährt der ASURO mit konstanter Geschwindigkeit rückwärts. Sobald die Sensoren die schwarze Linie erkennen, sprint der ASURO wieder in die PID-Regler Routine. Zusätzlich werden bei der Initialisierung die Werte (P, I und D) des PID-Reglers gesetzt. Der PID-Regler ist in eine Bibliothek ausgelagert. Der Mikrocontroller ruft in der while-Schleife die Funktion „lineFollower“ auf. Diese Funktion ermittelt und speichert die Werte der Sensoren einmal mit ausgeschalteter LED und mit eingeschalteter LED. Durch dieses Verfahren kann die Abhängigkeit der Sensoren vom Umgebungslicht kompensiert werden. Sind die Werte ermittelt, werden die Werte dem PID-Regler übergeben. Die Funktion pidExcute (siehe Abbildung 1) berechnet den jeweiligen PID-Wert zu den übergebenen Werten. Durch das vorher beschriebene Verfahren ist der should-Wert konstant 0.

```
double pidExecute(double should, double is, PIDState *state) {
    unsigned long now = Gettime();
    double timeChange = (double)(now - state->last);
    double error = should - is;
    double newErrorSum = state->sumError + (error * timeChange);
    if ((newErrorSum >= state->intMin) && (newErrorSum <= state->intMax))
        state->sumError = newErrorSum; // Prevent Integral Windup
    double dError = (error - state->lastError) / timeChange;
    double output = (state->kp * error) + (state->ki * state->sumError) + (state->kd * dError);
    state->lastError = error;
    state->last = now;
    if (output > state->outMax) {
        output = state->outMax;
    }
    if (output < state->outMin) {
        output = state->outMin;
    }
    return output;
}

void pidSet(PIDState *pid, double kp, double ki, double kd, double min, double max, double iMin, double iMax) {
    pid->kp = kp;
    pid->ki = ki;
    pid->kd = kd;
    pid->outMin = min;
    pid->outMax = max;
    pid->intMin = iMin;
    pid->intMax = iMax;
    pid->lastError = 0;
    pid->sumError = 0;
    pid->last = 0;
}
```

Abbildung 1: PID-Regler Quellcode

Anhand des Vorzeichens des PID-Wertes kann die nötige Bewegungsrichtung ermittelt werden. Der eigentlich PID-Wert wird zur Anpassung der Motorgeschwindigkeit verwendet. Dabei wird der berechnete PID-Wert auf beide Motoren jeweils zur Hälfte aufgeteilt. Das

bedeutet, dass zum Beispiel der Motor rechts um PID/2 schneller wird und Motor links um PID/2 langsamer. Nach der „lineFollower“ Funktion werden zwei Taster (K2 und K5) abgefragt. Diese Abfrage dient der Erkennung, ob der ASURO am Ende der Strecke angelangt ist. Der ASURO soll eine Drehung durchführen sobald beide Sensoren gedrückt werden.

Ablauf der Drehung:

- Motorgeschwindigkeit auf 0 setzen und bremsen
- Die aktuellen Werte der Sensoren speichern (schwarzer Bereich)
- Den linken Motor auf Drehrichtung vorwärts, Den rechten Motor auf Drehrichtung rückwärts
- Motorgeschwindigkeit: links: 112, rechts: 107 (Motoren laufen nicht gleich schnell)
- 200 ms Sekunden sleep um den schwarzen Bereich zu verlassen
- While-Schleife: Sensorwerte abfragen und bei Erkennung des schwarzen Bereichs anhalten
- While-Schleife verlassen

Die ursprüngliche while-Schleife wird nun wieder ausgeführt.

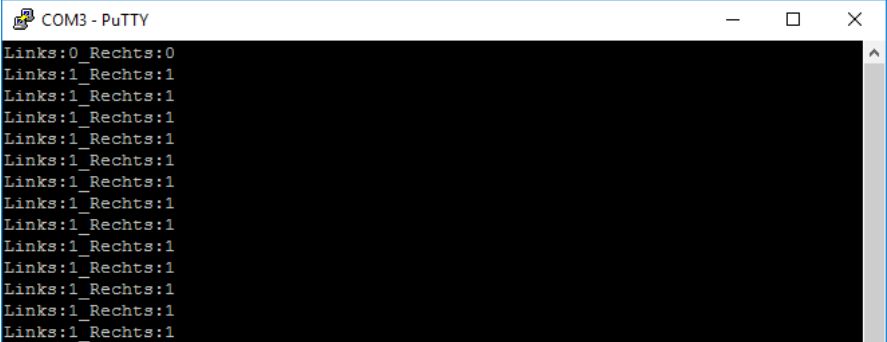
Quellcode: https://github.com/g40st/ASURO_Line_Follower

Ermittlung der Werte P, I und D:

Die Werte wurden per trial and error festgelegt.

Probleme:

Für die Drehung sollte die Funktion GoTurn(int distance, int degree, int speed) verwendet werden. Die Funktion verwendet ein Odometer um die Distanz beziehungsweise den Winkel zu berechnen. Dabei werden die Odometrie Sensoren im Interrupt-Betrieb verwendet. Zunächst sollte eine 180 Grad Drehung mit Hilfe der Funktion vollzogen werden. Allerdings endete dies in einer Endlosschleife. Nach einer genaueren Analyse stellte sich heraus, dass die Sensoren nicht in Ordnung waren. Anhand dieses [Beispielprogramms](#) wurden die Odometrie Sensoren getestet. Abbildung 2 zeigt, dass die Sensoren konstant die Werte 1 zurückgeben.



```

COM3 - PuTTY
Links:0 Rechts:0
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1
Links:1 Rechts:1

```

Abbildung 2: Terminalfenster Beispielprogramm

Ablaufdiagramme:

