Dokumentation zum AMS-Projekt „Flying Knipser“

Gruppe: 2

Mitglieder: Christian Heile,
Sebastian Ali Nasrollahkhan Shahrestanaki

Betreuer: Ingo Boersch

Aufgabe: Erstellung eines wettbewerbsfähigen LEGO-Roboters unter Benutzung des AKSEN-Boards, der das vorgegebene Labyrinth bewältigen kann.

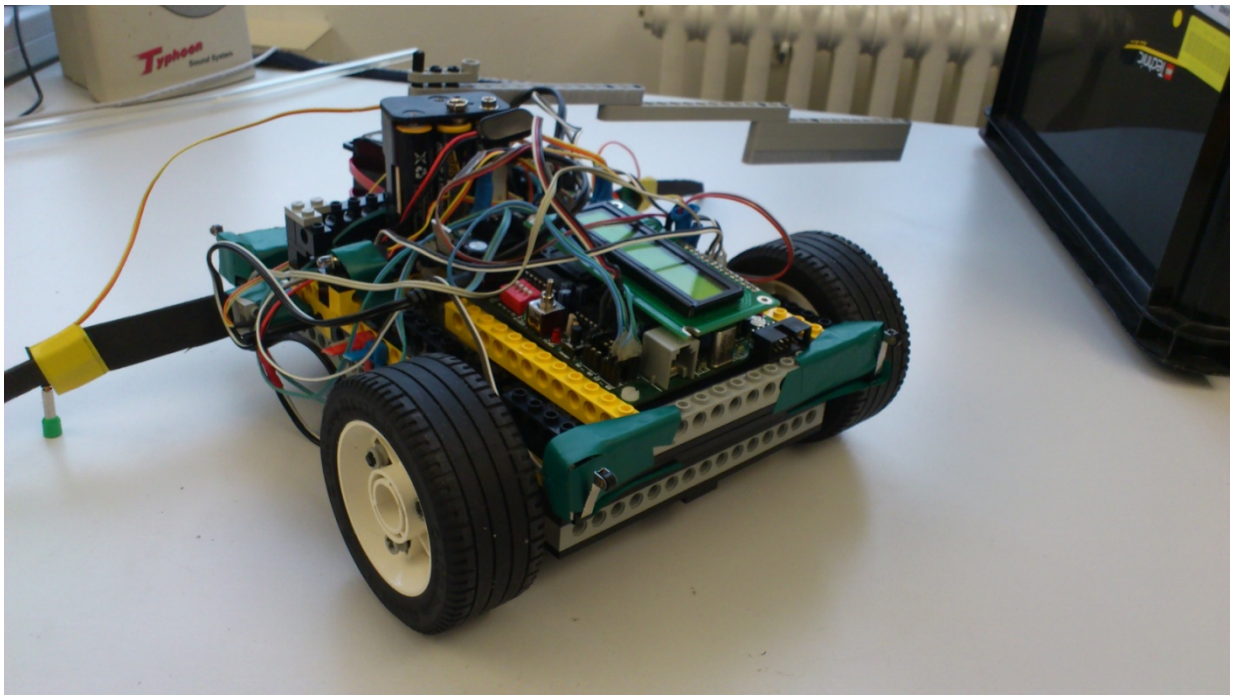

Abbildung 1: Fertiger Roboter „Flying Knipser“

1 Unser Lösungsweg

Nach der Vorstellung der Aufgabe haben wir uns dafür entschieden, den Roboter den kürzesten Weg durch das Labyrinth fahren zu lassen. Das bedeutet, wir haben uns von der Idee der Linienverfolgung distanziert und versucht, mithilfe der Odometrie den Parkour zuverlässig zu bewältigen.

Bei dem Bau unseres ersten Roboters wussten wir noch nicht, wo die Stärken und Schwächen des jeweils anderen lagen, so dass wir gemeinsam anfingen zu bauen.

Der erste Roboter sollte zunächst nur die Strategie des kürzesten Weges verfolgen. Zur Befahrung der Kurven entschieden wir uns für zwei Motoren, jeweils mit einem Getriebe verbunden. Der Gedanke war, eine möglichst gute Lenkung einzubauen, um die Strategie zu verwirklichen. Des weiteren war uns bewusst, dass unser Roboter das Element der Geschwindigkeit inne haben sollte. Aus diesem Grund entschieden wir uns bei unserem ersten Roboter für eine Übersetzung von 1:75.

Zur Getriebeübersetzung ist folgendes zu sagen. Uns standen vier verschiedene Zahnräder zur Verfügung. Eins mit acht Zähnen, eins mit 16, ein weiteres mit 24 und eins mit 40 Zähnen. Auf den Motor wurde ein Zahnrad mit acht Zähnen aufgesetzt. Verband man dieses nun mit einem Rad mit 24 Zahnrädern, erhielten wir eine Übersetzung von 1:3. Band man daran ein weiteres Zahnrad, so wurde das Verhältnis stets größer. Je kleiner das Verhältnis(1:150) ist, desto langsamer und kräftiger wurde der Roboter. Eine großes Verhältnis (1:60) spricht für einen schnellen Roboter mit weniger Antriebskraft.

Nachdem uns dieser Zusammenhang bewusst wurde, trafen wir weitere Designentscheidungen. Zum einen sollte die Übersetzung vom Motor auf die Räder klein sein, um eine möglichst hohe Geschwindigkeit zu erreichen, und zum anderen musste aufgrund des Kraftverlusts unser Roboter sehr leicht sein. Die Vorgabe des leichten Roboters setzten wir durch Verwendung leichter Legosteine um. Wir verzichteten auf Platten, unnötige Anbauten und größere Steine und verbauten stattdessen leichte und kleine Bausteine.

Zur Wegfindung bauten wir im ersten Versuch Optokoppler an den Roboter, um die schwarzen Linien, welche quer im Labyrinth liegen, zu finden.

Nachdem das erste Modell fertig war, wurde uns bei einigen Tests bewusst, wie unzuverlässig dieser Roboter war. Zum einen gab es Probleme mit der Linienerkennung und zum anderen konnte unser Roboter nicht geradeaus fahren. Das Problem waren die Getriebe. Jedes hatte unterschiedliche Reibung aufgrund des Aufbaus.

Aufgrund diese Feststellung bauten wir einen zweiten Roboter, welcher dieses mal allerdings mit nur einem Motor ausgestattet wurde. Wir behielten die Getriebeübersetzung von 1:75 bei. Die Lenkung realisierten wir mithilfe eines Rades am hinteren Teil des Roboters, welches durch einen Servo-Motor gesteuert wurde. Der Roboter war sehr klein, kompakt und schnell.
Auch bei diesem Roboter traten Probleme auf. Nach einem Gespräch mit unserem Betreuer Ingo Boersch bauten wir die Optokoppler ab und versuchten nun, den tatsächlichen Odometrieweg einzuschlagen.
Zunächst dachten wir an eine weiße Plastikscheibe, präpariert mit schwarzen Streifen, die mithilfe einer Lichtschranke gezählt werden konnten. Die Scheiben allerdings gaben uns nur vier Ticks je Radumdrehung, was zu wenig war.
Nach diesem Rückschlag bauten wir Lichtschranken direkt an den Zahnrädern des Getriebes an. Indem wir neben einem der Zahnräder einen Vorsatz bauten, konnten wir mithilfe von Sender und Empfänger infraroten Lichts eine Lichtschranke bauen, die wesentlich zuverlässiger die Ticks je

Radumdrehung zählte. Mittels der Encoder-Funktion des AKSEN-Boards zählten wir diese. Trotz dieses Anbaus war der Roboter noch unzuverlässig. Den Roboter lenkten wir mit einem kleinen Rad an einem Servomotor hinten am Roboter. Leider Jedoch funktionierte die Lenkung nicht, da wir aus Unerfahrenheit, beide Räder auf eine Achse legten ohne ein Differenzial dazwischen, sodass ein Rad bei der Kurvenfahrt immer hinterhergezogen wurde, wodurch eine Kurvenfahrt nie identisch war

Zu diesem Zeitpunkt befanden wir uns eine Woche vor Weihnachten und waren mittlerweile sehr frustriert über unsere bisherigen Ergebnisse.

Da wir davon überzeugt waren, dass dieser Roboter auch nicht die Lösung ist, entschlossen wir uns zum Bau eines dritten Roboters, der allerdings auf Linienerkennung beruhen sollte.
Diese Idee wurde jedoch auch schnell verworfen, allerdings nicht der Roboter.

Der „Flying-Knipser“ besaß bereits zwei Motoren und ein stabiles Gehäuse. Durch die zwei Motoren wollten wir das geradeaus Fahren gewährleisten, da wir nun die Möglichkeit hatten, ohne große Abweichungen zu lenken. Christian, der sich als wahrer Autobauingenieur erwies, baute je ein Getriebe mit einer Übersetzung von 1:81 ein. Um Lichtschranken an den Zahnrädern anzubringen, konstruierten wir dieses mal keinen Vorsatz an einem der Zahnräder, sondern ließen über mehrere Zahnräder die Umdrehung nach hinten laufen, dort wo genug Platz war, um Lichtschranken anzubauen und maßen dort über ein Legorad, welches in gleichem Abstand sechs Löcher hat.

Durch die Lichtschranke konnten wir nun Ticks zählen, die jeweils von den Motoren abgeleitet wurden. Mittels eines Vergleichs der Werte und einer Ausgleichsfunktion war nun unser erster Roboter gebaut, der tatsächlich zuverlässig geradeaus fahren konnte.

Da dieser Roboter wesentlich breiter als die letzten zwei war, fand sich genug Platz, um das AKSEN-Board sowie einen Servo-Motor einzubauen. Der Servo-Motor wurde später zur Drehung des Arms verwendet.

Nach einigen Tests im Labyrinth stellten sich weitere Probleme ein. Zunächst gab es noch zu große Abweichungen in den Kurven. Zum einem bauten wir eine Funktion ein, die ebenfalls anhand der Ticks die Dauer des Abbiegens festlegte. Zum anderen mussten die Reifen erst warm sein, bevor der Roboter das Labyrinth reibungslos durchfahren konnte.

Als letzten Schritt mussten wir den Roboter noch wettbewerbsfähig machen. Dazu mussten wir noch vier Dinge umsetzten. Einen Arm zum Abwurf der blauen Kugeln, das Energieproblem lösen, aufgrund dessen unser Roboter am Ende des Parkours langsamer wurde, trotz unserer kürzesten-Strecke-Strategie alle Wegpunkte einfahren und die Abweichungen, die bis zum Ende des Weges aufgetreten waren, wieder auszugleichen, um auf einem sicheren Weg an den Kugeln vorbeizufahren.

Der Arm war das leichteste Problem und doch stellte es sich als sehr trickreich heraus. Ein Arm war bereits gebaut. Er bestand aus den langen, abgerundeten Legosteinen und war stark genug, alle Kugeln zu bewegen. Allerdings stellte sich im weiteren Verlauf heraus, dass es sehr schwer war, mit einem so massiven Arm im Labyrinth zu manövrieren. Durch geschicktes Drehen des Arms im Quellcode konnten wir diese Schwierigkeit jedoch lösen.

Das Energieproblem war uns zunächst ein Rätsel. Zu diesem Zeitpunkt hatten wir das Bremsen vor den Kurven so implementiert, dass der Roboter kurz stoppt und dann lenkt. Das Stoppen funktionierte mittels einer kurzen Richtungsänderung der Motoren. Das heißt für 50 Millisekunden

drehte sich der Motor in die entgegengesetzte Richtung, um ein sauberes Stoppen zu ermöglichen. Nach einigen Versuchen fand Christian heraus, dass der Roboter durch diese Art von Stopp sehr viel Energie verbrauchte. Wir ließen die Bremsfunktion unverändert, bauten allerdings eine kurze Wartezeit ein, in der der Motor nichts tat, bevor er wieder einen Richtungswechsel vornahm, um weiterzufahren.

Vor dem Renntag bauten wir mit Schaumstoffstreifen an beiden Seiten des Roboters eine Art Aufsatz, an denen wir die lichtempfindlichen Komponenten anbauten, sodass der Roboter erst bei einem Lichtsignal startete. Da wir durch unsere Strategie nicht alle Wegpunkte mitnahmen, kam uns die Idee, diese angebauten Schaumstoffstreifen zu verlängern. Dadurch war es dem Roboter möglich, trotz des schnellen Wegs alle Wegpunkte zu erreichen. Die Regel besagte, dass der Roboter den Luftraum über den Wegpunkten berühren müsse. Durch diesen Anbau gelang uns das. Dadurch kommt auch das Flying in „Flying-Knipser“, da die Schaumstoffstreifen aussehen wie Flügel.

Das letzte Problem und vermutlich auch das komplexeste war die Ausrichtung, bevor wir auf die letzte Gerade fuhren, um die blauen Kugeln herunter zu werfen. Ingo Boersch brachte uns dann auf die Idee, mithilfe der Wände eine lokale Positionierung vorzunehmen. Aus diesem Grund bauten wir sowohl vorne als auch hinten je zwei Bumper an den Roboter. Diese Bumper oder auch Schalter liefern den Wert 1, wenn sie offen (unbetätigt) sind, andernfalls den Wert 0 (betätigt). Die Strategie für diesen Abschnitt war nun, den Roboter gegen die Wand fahren zu lassen. Dadurch wurden die Bumber aktiviert und wir fuhren rückwärts, drehten uns und fuhren noch einmal gegen die Wand hinter uns. Da gewährleistet war, dass unser Roboter geradeaus fährt und er nun gerade stand, konnte der Roboter vorwärts fahren und die Kugeln bewegen, ohne gegen sie zu fahren.
Aufgrund dieser Lösung erhielt der Roboter den zweiten Teil seines Namens „Knipser“.

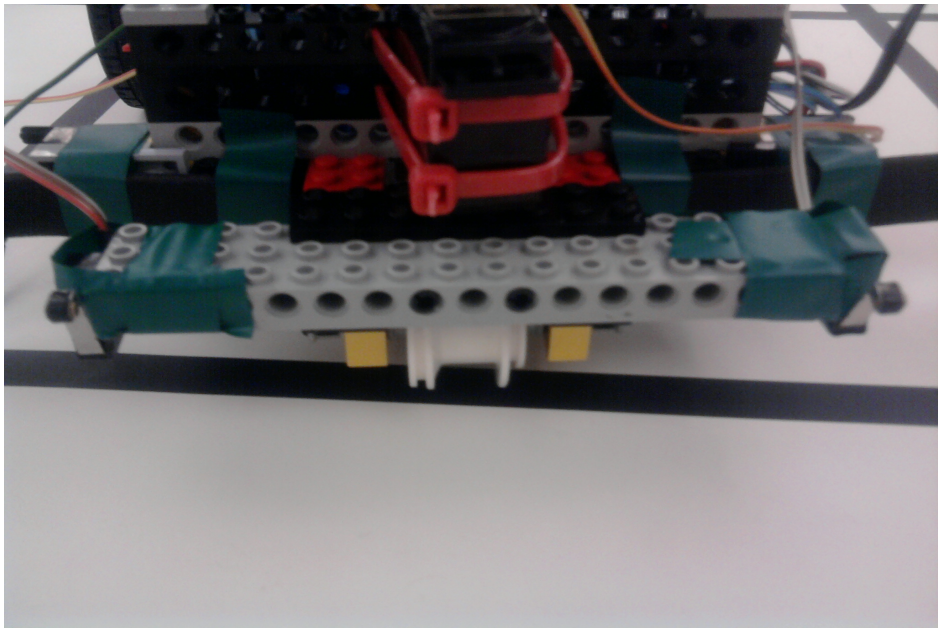

Abbildung 2: Bumper hinten am Roboter

Somit hatten wir einen fertigen Roboter, der zuverlässig durch das Labyrinth fahren konnte und dabei den schnellsten Weg nahm. Zeit verlor er jedoch durch das Bremsen vor den Kurven sowie das Ausrichten am Ende des Parkours.

2 Vorstellung des Roboters

Der „Flying-Knipser“ verfügt über zwei Getriebe mit jeweils einer Übersetzung von 1:81. An jedem Getriebe läuft ein Motor. Die Motoren sind in der Mitte des Roboters angebracht. Der Aufbau des Getriebes verläuft nach vorne zu den Rädern, nach hinten sind extra Zahnräder angebaut, an die eine Lichtschranke angebaut ist.

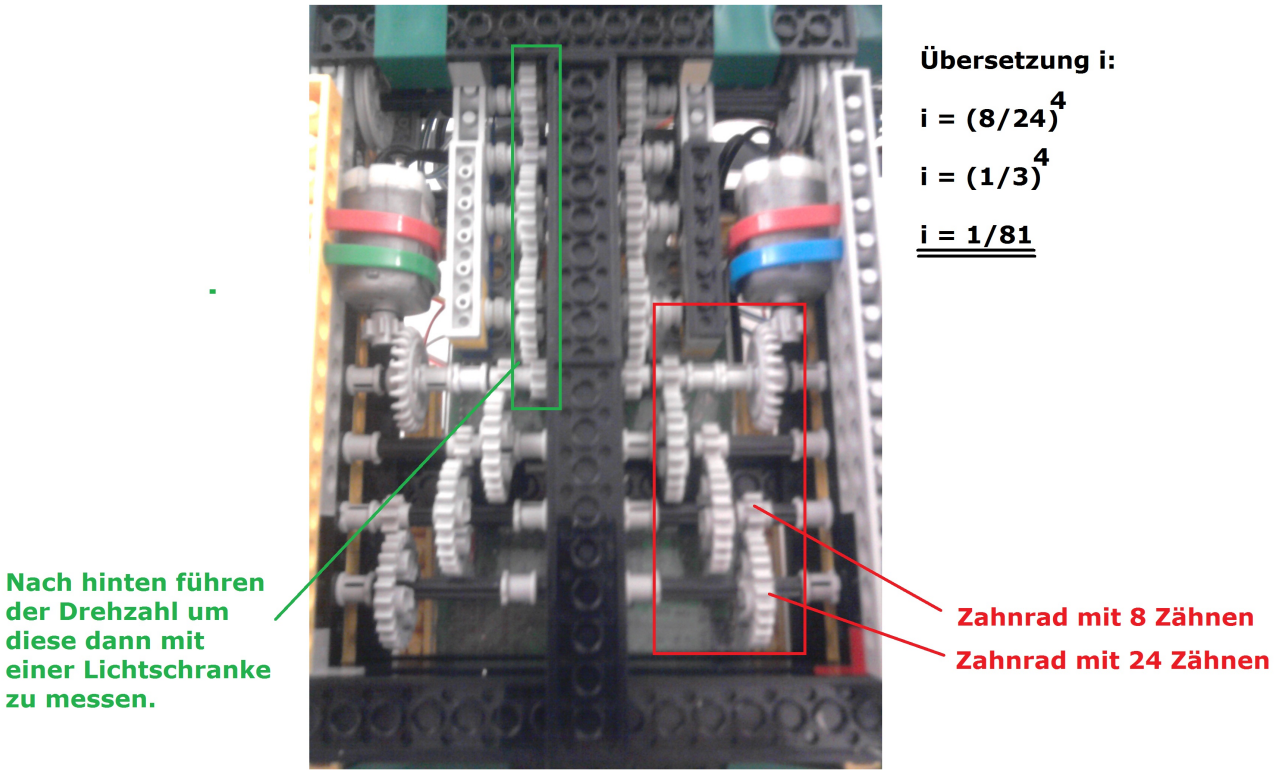

Abbildung 3: Getriebe des Roboters

Die Lichtschranke besteht aus einem Sender, welcher infrarotes Licht aussendet und einem entsprechenden Empfänger. An beiden Getrieben ist eine solche Lichtschranke angebracht. Die Empfänger sind jeweils an die Encodereingänge des AKSEN-Boards angeschlossen. Die Lichtschranken zählen so die Anzahl der Umdrehungen des Zahnrades, an dem sie angebracht sind. Im Quellcode sind Variablen angelegt, welche dem Encoderspeicher zugeordnet sind, um das Aufzählen zu gewährleisten:

```
volatile xdata at 0x1EFE unsigned char encoder00;
volatile xdata at 0x1EFF unsigned char encoder01;
volatile xdata at 0x1EFC unsigned char encoder10;
volatile xdata at 0x1EFC unsigned char encoder11;
```

Über die Funktion `getEncoder(unsigned int nummer)` kann die aktuelle Anzahl an vergangenen Ticks abgefragt werden:

```
unsigned int getEncoder(unsigned int nummer)
{
  unsigned int wert = 0;

  switch(nummer)
  {
  case 0:
          wert = encoder01;
```

```
            wert <<= 8;
            wert += encoder00;
            break;
    case 1:
            wert = encoder11;
            wert <<= 8;
            wert += encoder10;
            break;
    }

    return wert;
}
```

Durch den Parameter nummer wird angegeben, welche Lichtschranke abgefragt werden soll. Mithilfe der Odometrie kann auch das Geradefahren durchgeführt werden:

```
void fahreGerade(int laenge, unsigned int richtung)
{
  int differenz = 0;
  resetEncoder();
  motor_richtung(0,richtung);
  motor_richtung(1,richtung);
  motor_pwm(0,10);
  motor_pwm(1,10);

  while(getEncoder(0) < laenge || getEncoder(1) < laenge)
  {
          differenz = getEncoder(0) - getEncoder(1);
          //links zu schnell
          if(differenz > 1)
          {
                  motor_pwm(0,5);
                  motor_pwm(1,10);
                  led(0,1);
                  led(1,0);
          }
          //rechts zu schnell
          else if(differenz < -1)
          {
                  motor_pwm(0,10);
                  motor_pwm(1,5);
                  led(0,0);
                  led(1,1);
          }
          else
          {
                  motor_pwm(0,10);
                  motor_pwm(1,6);
                  led(0,0);
                  led(1,0);
          }
  }
  bremsen(50);
}
```

Mithilfe dieser Ausgleichsfunktion kann der Roboter zuverlässig geradeaus fahren.

Zur lokalen Positionierung des Roboters am Ende des Labyrinths werden Bumper verwendet. Da der Roboter an der Vorder- und Hinterseite flach ist, kann er durch schnelles Fahren gegen die Wand sich selbst parallel dazu ausrichten. Über die Bumper wird festgestellt, ob der Roboter tatsächlich parallel zur Wand ist oder nicht. Erst wenn beide Bumper betätigt sind, steht der Roboter korrekt. Diese Ausstattung des Roboters ist besonders gut gelungen. Der Roboter richtet sich jedesmal

korrekt aus und alle Abweichungen durch die Kurven, die bis dahin eintreten, werden eliminiert.
Mithilfe der Funktion `void fahreGegenWand(int richtung)` wird getestet, ob die Bumper aktiviert sind:

```
while((digital_in(0) == 1 && digital_in(1) == 1) &&
      (digital_in(6) == 1 && digital_in(7) == 1))
```

Der Arm des Roboters ist im hinteren Teil an einem Servo-Motor angebracht. Er ist robust und stabil, hat dadurch aber auch den Nachteil, im Parkour bei der Lenkung hinderlich zu sein.

Die lichtempfindlichen Sensoren sind an Schaumstoffstreifen angebracht, die links und rechts am Roboter kleben. Dadurch kann der Roboter sowohl auf das Startsignal reagieren, als auch die Wegpunkte einsammeln. Durch eine Verlängerung der Schaumstoffstreifen reicht der Roboter bis in den Luftraum dieser Punkte, wird aber nicht an Wänden aufgehalten.

Der Roboter verfolgt auf beiden Startseiten die gleiche Strecke jeweils spiegelverkehrt:

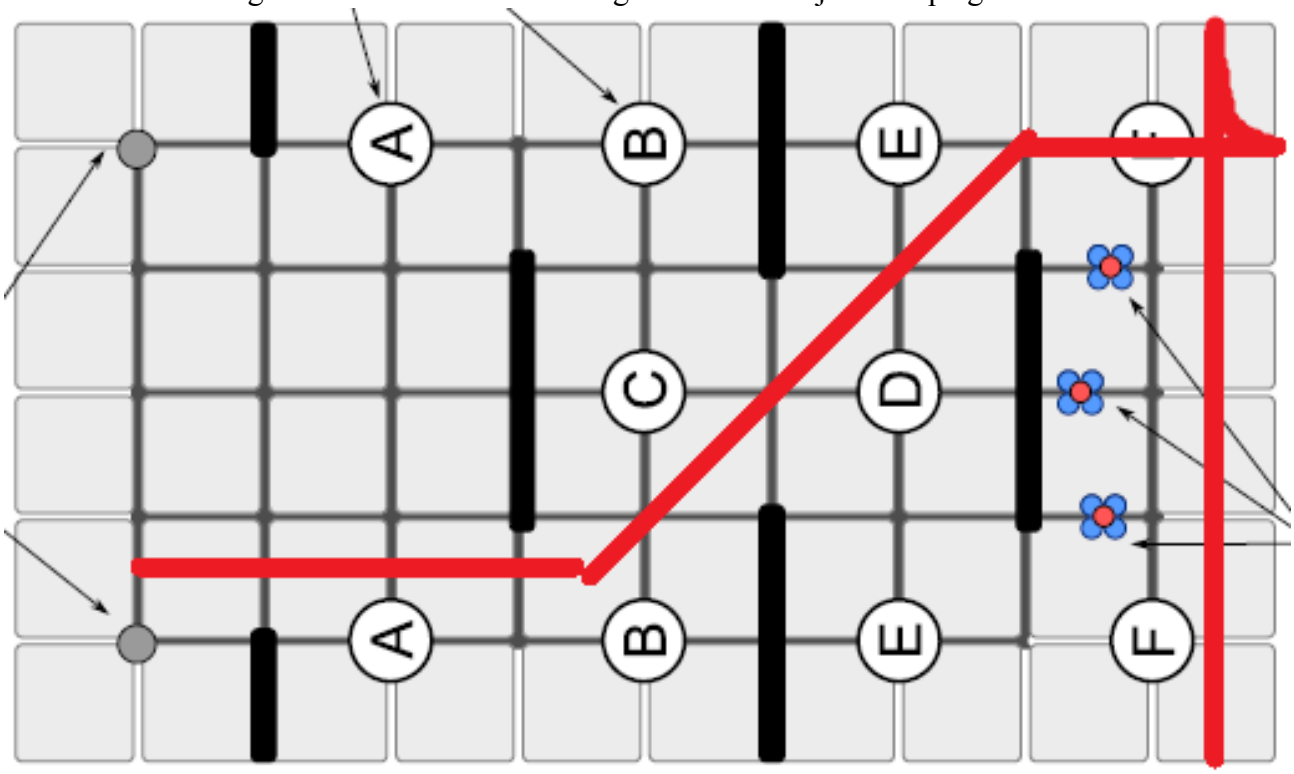

Abbildung 4: Der Weg des “Flying Knipser”

Mithilfe des Dipschalters null kann vor dem Rennen angegeben werden, von welcher Position der Roboter startet.

3 Hardware

Im Roboter sind ein AKSEN-Board, vier Bumper, zwei Motoren, ein Servo-Motor, zwei infrarote Sender und entsprechende Empfänger verbaut.

Der Antrieb läuft über zwei Getriebe, die mithilfe von Lichtschranken im Quellcode synchronisiert werden.
Trotz seiner Breite kann sich der Roboter auch noch im Labyrinth schnell bewegen. Da zwei Motoren für den Antrieb sorgen, kann sich der Roboter im besten Fall auf dem Mittelpunkt der

Vorderachse drehen. Wird allerdings nur ein Rad gedreht ohne das andere, so fährt der Roboter eine präzise Kurve. Präzise daher, da er ausschließlich auf Grundlage der Lichtschranke bei gleichen Einstellungen in acht von zehn Fällen einen nahezu identischen Winkel einschlägt mit maximal 3 Grad Abweichung.

Die Lichtschranken liefern ihr Signal an das AKSEN-Board, welches die Informationen sofort weiter verarbeitet. Es findet keine Vorverarbeitung statt.

Besonders hervorzuheben sind die Schalter oder Bumper, welche zur Ausrichtung des Roboters vorne und hinten angebracht sind. Mittels der Übertragung an das Board kann am Ende des Labyrinths eine nahezu exakte Positionsbestimmung vorgenommen werden.

4 Software

Auf dem AKSEN-Board wurden folgende Funktionen implementiert:

```
unsigned int getEncoder(unsigned int nummer)
void resetEncoder()
void bremsen(unsigned int dauer, unsigned int richtung)
void fahreGerade(int laenge, unsigned int richtung)
void fahreGegenWand(int richtung)
void drehen(unsigned int dauer, unsigned int richtung)
```

Sowie die Hauptfunktion

```
void AksenMain(void)
```

Über die Funktion `getEncoder(unsigned int nummer)` kann jeweils die Lichtschranke am rechten oder am linken Rad angesprochen werden. In der Funktion wird der Wert aus den dafür vorgesehenen Speicheradressen

```
volatile xdata at 0x1EFE unsigned char encoder00;
volatile xdata at 0x1EFF unsigned char encoder01;
volatile xdata at 0x1EFC unsigned char encoder10;
volatile xdata at 0x1EFD unsigned char encoder11;
```

angesprochen und korrekt zusammengefügt. `Encoder00` und `Encoder01` bilden den Speicherbereich für den Encodereingang 0, der die Ticks des linken Getriebes ausgibt und `Encoder10` und `Encoder11` bilden den Speicherbereich für den Encodereingang 1, der die Ticks des rechten Getriebes misst. Mithilfe eines Shift-Befehls wird aus den zwei Encoderwerten ein 16-Bit Wert ermittelt, der die vergangenen Ticks speichert.

```
unsigned int getEncoder(unsigned int nummer)
{
      unsigned int wert = 0;

      switch(nummer)
      {
      case 0:
            wert = encoder01;
            wert <<= 8;
            wert += encoder00;
            break;
      case 1:
            wert = encoder11;
            wert <<= 8;
            wert += encoder10;
```

```
            break;
      }
      return wert;
}
```

An Kurven beziehungsweise nach Kurven muss der Zähler wieder zurückgesetzt werden. Mittels der Funktion `void resetEncoder()` werden die bis dahin gezählten Ticks wieder zurückgesetzt.

```
void resetEncoder()
{
      encoder00 = 0;
      encoder01 = 0;
      encoder10 = 0;
      encoder11 = 0;
}
```

Das Fahren wird durch die Funktionen `void fahreGerade(int laenge, unsigned int richtung)` und `void drehen(unsigned int dauer, unsigned int richtung)` umgesetzt.
Der Funktion `void fahreGerade(int laenge, unsigned int richtung)` wird die Anzahl der zu fahrenden Ticks sowie die Richtung mitgegeben. Über die Differenzbildung der Encoderwerte des linken und rechten Getriebes wird ermittelt, ob eine Seite langsamer fährt als die andere. In diesem Fall findet ein Ausgleich statt, in dem der Roboter einen Motor verlangsamt.

```
void fahreGerade(int laenge, unsigned int richtung)
{
      int differenz = 0;
      resetEncoder();
      motor_richtung(0,richtung);
      motor_richtung(1,richtung);
      motor_pwm(0,10);
      motor_pwm(1,10);

      while(getEncoder(0) < laenge || getEncoder(1) < laenge)
      {
            //Differenzbildung
            differenz = getEncoder(0) - getEncoder(1);
            //links zu schnell
            if(differenz > 1)
            {
                  motor_pwm(0,4);
                  motor_pwm(1,10);
                  led(0,1);
                  led(1,0);
            }
            //rechts zu schnell
            else if(differenz < -1)
            {
                  motor_pwm(0,10);
                  motor_pwm(1,4);
                  led(0,0);
                  led(1,1);
            }
            //normal
            else
            {
                  motor_pwm(0,10);
                  motor_pwm(1,5);
                  led(0,0);
```

```
                        led(1,0);
                }
        }
        bremsen(75, richtung);
}
```

Das Ansprechen der led's ist eine Testfunktion um zu prüfen, wie oft und auf welcher Seite ausgeglichen wird.
Die Funktion `void bremsen(unsigned int dauer, unsigned int richtung)` am Ende des Fahrens bremst den Roboter. Die Motoren drehen in der Zeitspanne `dauer` entgegen der ursprünglichen Richtung, dann wird der Motor für 750ms abgestellt da sonst bei schnellem Wechsel von vor- und rückwärtsfahren die Versorgungsspannung zusammenbricht. ruht der Roboter
.

```
void bremsen(unsigned int dauer,unsigned int richtung)
{
        switch(richtung)
        {
        case 0:
                motor_richtung(0,1);
                motor_richtung(1,1);
                break;
        case 1:
                motor_richtung(0,0);
                motor_richtung(1,0);
                break;
        }
        sleep(dauer);
        motor_pwm(0,0);
        motor_pwm(1,0);
        sleep(750);
}
```

Über die Funktion `void drehen(unsigned int dauer, unsigned int richtung)` kann der Roboter sich in eine Richtung drehen. Der Parameter dauer gibt an, wieviele Ticks an dem entsprechenden Getriebe vergangen sein müssen, bis der Roboter wieder zum Stillstand kommt.

```
void drehen(unsigned int dauer, unsigned int richtung)
{
        motor_richtung(0,0);
        motor_richtung(1,0);
        resetEncoder();

        motor_pwm(richtung,10);
        while(getEncoder(richtung) < dauer);

        motor_pwm(richtung,0);
}
```

Für das Ende des Labyrinths ist die Funktion `void fahreGegenWand(int richtung)` implementiert. Diese Funktion ist ähnlich der Funktion `void fahreGerade(int laenge, unsigned int richtung)`, hat allerdings als Abbruchbedingung die Aktivierung der Bumper. Auch in dieser Funktion ist das Ausgleichssystem verankert.

```
void fahreGegenWand(int richtung)
{
        int differenz = 0;
        resetEncoder();
        motor_richtung(0,richtung);
```

```
        motor_richtung(1,richtung);
        motor_pwm(0,10);
        motor_pwm(1,10);

        while((digital_in(0) == 1 && digital_in(1) == 1) &&
              (digital_in(6) == 1 && digital_in(7) == 1))
        {
              //Differenzbildung
              differenz = getEncoder(0) - getEncoder(1);
              //links zu schnell
              if(differenz > 1)
              {
                    motor_pwm(0,5);
                    motor_pwm(1,10);
                    led(0,1);
                    led(1,0);
              }
              //rechts zu schnell
              else if(differenz < -1)
              {
                    motor_pwm(0,10);
                    motor_pwm(1,5);
                    led(0,0);
                    led(1,1);
              }
              //normal
              else
              {
                    motor_pwm(0,10);
                    motor_pwm(1,6);
                    led(0,0);
                    led(1,0);
              }
        }
        sleep(150);
        bremsen(50,0);
  }
```

Der Code hat während des Wettbewerbs keine Schwächen gezeigt. Allerdings ist der Sieg durch eine zu lange Wartezeit an den Kurven verloren gegangen. Im letzten Rennen gab es einen Ausfall, der allerdings der Hardware zuzuschreiben ist. Es ist allerdings unklar, wo genau der Fehler lag.
Die Wartezeit an den Kurven kann noch verbessert und die Mainloop um zusätzliche Wege erweitert werden, um auf gegnerische Strategien besser zu reagieren.

5 Vorschläge für das Projekt

Für uns war das Projekt ein voller Erfolg. Trotz einiger Rückschläge beim Bau eines fähigen Roboters hatten wir Spaß an der Arbeit und einen großen Lernerfolg, welcher mit dem ersten Platz auf der Informania gekrönt wurde. Die Gruppendynamik war ebenfalls sehr erfrischend. Es gab durchaus eine Wettbewerbsstimmung unter uns, allerdings haben sich alle untereinander geholfen und bei Problemen auch mal bei vermeindlichen Gegnern mit Hand angelegt.

Desweiteren hätten wir uns gewünscht, dass es ein Studiumbegleitendes Projekt in der Richtung gäbe. Die Idee dahinter wäre eine Arbeit mit dem AKSEN-Board an verschiedenen Projekten, die durchaus auch vom Studenten selbst gewählt werden könnten.

6 Anhang

Quellcode

```
//Autoren: Christian Heile, Sebastian Ali Nasrollahkhan
Shahrestanaki
//Konstrukteur: Christian Heile

#include <stub.h>

unsigned int getEncoder(unsigned int nummer);
void resetEncoder();
void bremsen(unsigned int dauer, unsigned int richtung);
void fahreGerade(int laenge, unsigned int richtung);
void fahreGegenWand(int richtung);
void drehen(unsigned int dauer, unsigned int richtung);

volatile xdata at 0x1EFE unsigned char encoder00;
volatile xdata at 0x1EFF unsigned char encoder01;
volatile xdata at 0x1EFC unsigned char encoder10;
volatile xdata at 0x1EFD unsigned char encoder11;

//Hauptprogramm zur Durchquerung des Labyrinths
//Strategie des schnellsten Wegs.
//Mithilfe von Odometrie und einem Differenzial (2 Motoren) wird
der Weg mit
//geringfügiger Abweichung befahren.
//Ziel ist es die Optimallinie ohne nach außen "guckende" Sensoren
zu befahren.
void AksenMain(void)
{

     while(analog(7) >  15 && analog(6) > 15);
     resetEncoder();
     servo_arc(2, 90);

     //Start von rechts
     if(dip_pin(0) == 1)
     {
          fahreGerade(175,0); //Erste Gerade
          drehen(33,1);       //Erste Kurve
          fahreGerade(300,0); //Lange Gerade
          drehen(35,0);       //Richtung Wand drehen
          fahreGegenWand(0);  //Drauf zu fahren
          fahreGerade(58,1);  //Rückwärts fahren
          drehen(32, 0);      //Aktive Lokalisierung
          servo_arc(2,100);   //mit Hilfe der Bumber
          drehen(32, 0);      //Ausrichten des Armes
          servo_arc(2, 130);  //und Ausrichtung zu den Kugeln
          fahreGegenWand(1);
          fahreGerade(250,0); //Kugeln umschmeißen
          fahreGegenWand(0);
     }
```

```
     //Start von links
     else
     {

          fahreGerade(175,0); //Erste Gerade
          drehen(29,0);       //Erste Kurve
          fahreGerade(290,0); //Lange Gerade
          drehen(30,1);       //Richtung Wand drehen
          fahreGegenWand(0);  //Drauf zu fahren
          fahreGerade(58,1);  //Rückwärts fahren
          drehen(20, 1);      //Aktive Lokalisierung
          servo_arc(2,30);    //mit Hilfe der Bumber
          drehen(32, 1);      //Ausrichten des Armes
          servo_arc(2, 10);   //und Ausrichtung zu den Kugeln
          fahreGegenWand(1);
          fahreGerade(250,0); //Kugeln umschmeißen
          fahreGegenWand(0);


     }
     motor_pwm(0,0);
     motor_pwm(1,0);
     while(1);
}

//Prozedur zum Einlesen der Encoder-Werte
unsigned int getEncoder(unsigned int nummer)
{
     unsigned int wert = 0;

     switch(nummer)
     {
     case 0:
          wert = encoder01;
          wert <<= 8;
          wert += encoder00;
          break;
     case 1:
          wert = encoder11;
          wert <<= 8;
          wert += encoder10;
          break;
     }

     return wert;
}

//Prozedur setzt Encoder-Werte zurück
void resetEncoder()
{
     encoder00 = 0;
     encoder01 = 0;
```

```
     encoder10 = 0;
     encoder11 = 0;
}

//Prozedur zum geradeaus fahren
//ueber die Differenz der Umdrehung des linken und rechten Rades
wird ermittelt,
//ob der Roboter gerade fährt und eventuell wird gegengesteuert
//laenge: Anzahl der zu fahrenden Ticks
//richtung: 0 == vorwaerts; 1 == rueckwaerts
void fahreGerade(int laenge, unsigned int richtung)
{
     int differenz = 0;
     resetEncoder();
     motor_richtung(0,richtung);
     motor_richtung(1,richtung);
     motor_pwm(0,10);
     motor_pwm(1,10);

     while(getEncoder(0) < laenge || getEncoder(1) < laenge)
     {
          //Differenzbildung
          differenz = getEncoder(0) - getEncoder(1);
          //links zu schnell
          if(differenz > 1)
          {
               motor_pwm(0,4);
               motor_pwm(1,10);
               led(0,1);
               led(1,0);
          }
          //rechts zu schnell
          else if(differenz < -1)
          {
               motor_pwm(0,10);
               motor_pwm(1,4);
               led(0,0);
               led(1,1);
          }
          //normal
          else
          {
               motor_pwm(0,10);
               motor_pwm(1,5);
               led(0,0);
               led(1,0);
          }
     }
     bremsen(75, richtung);
}


//Prozedur zum bremsen
```

```
//dauer: Anzahl der Ticks
//richtung: 1 == rechter Motor; 0 == linker Motor
void bremsen(unsigned int dauer,unsigned int richtung)
{
     switch(richtung)
     {
     case 0:
          motor_richtung(0,1);
          motor_richtung(1,1);
          break;
     case 1:
          motor_richtung(0,0);
          motor_richtung(1,0);
          break;
     }
     sleep(dauer);
     motor_pwm(0,0);
     motor_pwm(1,0);
     sleep(750);
}


//Prozedur zum links drehen
//richtung: 1 == Linksdrehung; 0 == Rechtsdrehung
void drehen(unsigned int dauer, unsigned int richtung)
{
     motor_richtung(0,0);
     motor_richtung(1,0);
     resetEncoder();

     motor_pwm(richtung,10);
     while(getEncoder(richtung) < dauer);

     motor_pwm(richtung,0);
}

//Prozedur zum Fahren gegen eine Wand, bis die Bumper ausgeloest
wurden
//richtung: 0 == vorwaerts; 1 == rueckwaerts
void fahreGegenWand(int richtung)
{
     int differenz = 0;
     resetEncoder();
     motor_richtung(0,richtung);
     motor_richtung(1,richtung);
     motor_pwm(0,10);
     motor_pwm(1,10);

     while((digital_in(0) == 1 && digital_in(1) == 1) &&
(digital_in(6) == 1 && digital_in(7) == 1))
     {
          //Differenzbildung
          differenz = getEncoder(0) - getEncoder(1);
```

```
        //links zu schnell
        if(differenz > 1)
        {
             motor_pwm(0,5);
             motor_pwm(1,10);
             led(0,1);
             led(1,0);
        }
        //rechts zu schnell
        else if(differenz < -1)
        {
             motor_pwm(0,10);
             motor_pwm(1,5);
             led(0,0);
             led(1,1);
        }
        //normal
        else
        {
             motor_pwm(0,10);
             motor_pwm(1,6);
             led(0,0);
             led(1,0);
        }
    }
    sleep(150);
    bremsen(50,0);
}
```

```
//Autoren: Christian Heile, Sebastian Ali Nasrollahkhan Shahrestanaki
//Konstrukteur: Christian Heile

#include <stub.h>

unsigned int getEncoder(unsigned int nummer);
void resetEncoder();
void bremsen(unsigned int dauer, unsigned int richtung);
void fahreGerade(int laenge, unsigned int richtung);
void fahreGegenWand(int richtung);
void drehen(unsigned int dauer, unsigned int richtung);

volatile xdata at 0x1EFE unsigned char encoder00;
volatile xdata at 0x1EFF unsigned char encoder01;
volatile xdata at 0x1EFC unsigned char encoder10;
volatile xdata at 0x1EFD unsigned char encoder11;

//Hauptprogramm zur Durchquerung des Labyrinths
//Strategie des schnellsten Wegs.
//Mithilfe von Odometrie und einem Differenzial (2 Motoren) wird der Weg mit
//geringfügiger Abweichung befahren.
//Ziel ist es die Optimallinie ohne nach außen "guckende" Sensoren zu befahren.
void AksenMain(void)
{

    while(analog(7) >  15 && analog(6) > 15);
    resetEncoder();
    servo_arc(2, 90);

    //Start von rechts
    if(dip_pin(0) == 1)
    {
        //Erste Gerade
        fahreGerade(175,0);
        //Erste Kurve
        drehen(33,1);
        //Lange Gerade
        fahreGerade(300,0);
        //Richtung Wand drehen
        drehen(35,0);
        //Drauf zu fahren
        fahreGegenWand(0);
        //Rückwärts fahren
        fahreGerade(58,1);
        //Drehung zu den Kugeln
        drehen(32, 0);
        servo_arc(2,100);
        drehen(32, 0);
        servo_arc(2, 130);
        fahreGegenWand(1);
        fahreGerade(250,0);
        //Arm ausfahren
        //Kugeln umschmeißen
        fahreGegenWand(0);


    }

    //Start von links
    else
    {
        //Erste Gerade
        fahreGerade(175,0);
        //Erste Kurve
        drehen(29,0);
        //Lange Gerade
        fahreGerade(290,0);
        //Richtung Wand drehen
        drehen(30,1);
        //Drauf zu fahren
        fahreGegenWand(0);
        //Rückwärts fahren
        fahreGerade(58,1);
        //Drehung zu den Kugeln
```

```
        drehen(20, 1);
        servo_arc(2,30);
        drehen(32, 1);
        servo_arc(2, 10);
        fahreGegenWand(1);
        fahreGerade(250,0);
        //Arm ausfahren
        //Kugeln umschmeißen
        fahreGegenWand(0);


    }
    motor_pwm(0,0);
    motor_pwm(1,0);
    while(1);
}

//Prozedur zum Einlesen der Encoder-Werte
unsigned int getEncoder(unsigned int nummer)
{
    unsigned int wert = 0;

    switch(nummer)
    {
    case 0:
        wert = encoder01;
        wert <<= 8;
        wert += encoder00;
        break;
    case 1:
        wert = encoder11;
        wert <<= 8;
        wert += encoder10;
        break;
    }

    return wert;
}

//Prozedur setzt Encoder-Werte zurück
void resetEncoder()
{
    encoder00 = 0;
    encoder01 = 0;
    encoder10 = 0;
    encoder11 = 0;
}

//Prozedur zum geradeaus fahren
//ueber die Differenz der Umdrehung des linken und rechten Rades wird ermittelt,
//ob der Roboter gerade fährt und eventuell wird gegengesteuert
//laenge: Anzahl der zu fahrenden Ticks
//richtung: 0 == vorwaerts; 1 == rueckwaerts
void fahreGerade(int laenge, unsigned int richtung)
{
    int differenz = 0;
    resetEncoder();
    motor_richtung(0,richtung);
    motor_richtung(1,richtung);
    motor_pwm(0,10);
    motor_pwm(1,10);

    while(getEncoder(0) < laenge || getEncoder(1) < laenge)
    {
        //Differenzbildung
        differenz = getEncoder(0) - getEncoder(1);
        //links zu schnell
        if(differenz > 1)
        {
            motor_pwm(0,4);
            motor_pwm(1,10);
            led(0,1);
            led(1,0);
        }
```

```
        //rechts zu schnell
        else if(differenz < -1)
        {
            motor_pwm(0,10);
            motor_pwm(1,4);
            led(0,0);
            led(1,1);
        }
        //normal
        else
        {
            motor_pwm(0,10);
            motor_pwm(1,5);
            led(0,0);
            led(1,0);
        }
    }
    bremsen(75, richtung);
}


//Prozedur zum bremsen
//dauer: Anzahl der Ticks
//richtung: 1 == rechter Motor; 0 == linker Motor
void bremsen(unsigned int dauer,unsigned int richtung)
{
    switch(richtung)
    {
    case 0:
        motor_richtung(0,1);
        motor_richtung(1,1);
        break;
    case 1:
        motor_richtung(0,0);
        motor_richtung(1,0);
        break;
    }
    sleep(dauer);
    motor_pwm(0,0);
    motor_pwm(1,0);
    sleep(750);
}


//Prozedur zum links drehen
//richtung: 1 == Linksdrehung; 0 == Rechtsdrehung
void drehen(unsigned int dauer, unsigned int richtung)
{
    motor_richtung(0,0);
    motor_richtung(1,0);
    resetEncoder();

    motor_pwm(richtung,10);
    while(getEncoder(richtung) < dauer);

    motor_pwm(richtung,0);
}

//Prozedur zum Fahren gegen eine Wand, bis die Bumper ausgeloest wurden
//richtung: 0 == vorwaerts; 1 == rueckwaerts
void fahreGegenWand(int richtung)
{
    int differenz = 0;
    resetEncoder();
    motor_richtung(0,richtung);
    motor_richtung(1,richtung);
    motor_pwm(0,10);
    motor_pwm(1,10);

    while((digital_in(0) == 1 && digital_in(1) == 1) && (digital_in(6) == 1 && digital_in ↙
    (7) == 1))
    {
        //Differenzbildung
        differenz = getEncoder(0) - getEncoder(1);
```

```
        //links zu schnell
        if(differenz > 1)
        {
            motor_pwm(0,5);
            motor_pwm(1,10);
            led(0,1);
            led(1,0);
        }
        //rechts zu schnell
        else if(differenz < -1)
        {
            motor_pwm(0,10);
            motor_pwm(1,5);
            led(0,0);
            led(1,1);
        }
        //normal
        else
        {
            motor_pwm(0,10);
            motor_pwm(1,6);
            led(0,0);
            led(1,0);
        }
    }
    sleep(150);
    bremsen(50,0);
}
```