PPS LEGO Minstorms 2007

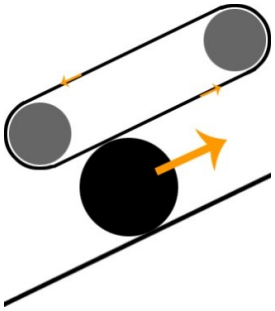
Roboball

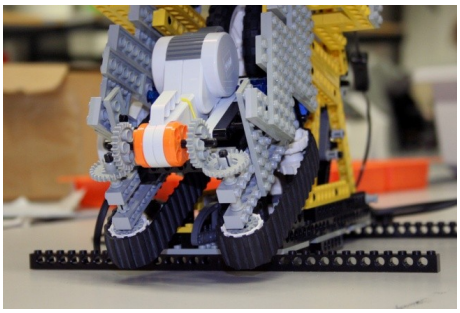
Hans-Peter Wii

Aufbau

Ballaufnahme

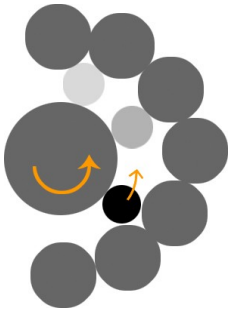
Der Ball kommt durch eine Art Trichter zu einem in der Mitte des Roboters angebrachten Aufnahmemechanismus, dieser befördert den Ball mit Hilfe einer Walze zum nächsten Teil, der den Ball zum Abschussmechanismus transportiert.

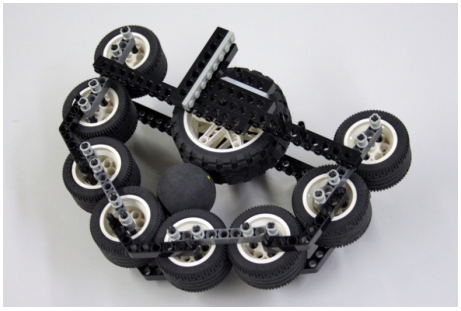




Balltransport

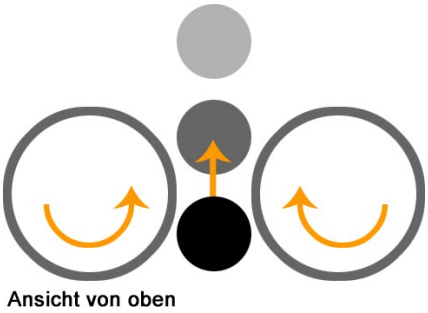
Um nach oben transportiert zu werden wird der Ball zwischen Räder, die in einem Kreis angeordnet sind, und einem drehenden Rad eingeklemmt, welches den Ball durch die Drehung und Reibung nach oben befördert. Hier befindet sich auch die Ballerkennung, sie besteht aus zwei auf den entgegengesetzten Seiten angebrachten Lichtsensoren von denen einer im aktiven und einer im inaktiven Modus geschaltet sind (d.h. bei einer ist die IR-Lampe eingeschaltet und der andere Sensor misst nur die Lichtstärke). Falls ein Ball durch diese Lichtschranke durchgeht nimmt die Lichtstärke beim passiven Sensor stark ab und weiss dass sich ein Ball dazwischen befindet.

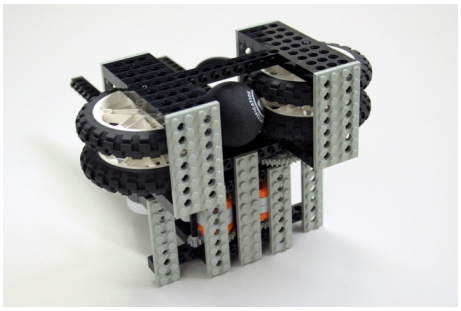




Abschuss

Der Abschussmechanismus besteht aus zwei schnell gegeneinander drehenden Rädern, die in einem Abstand platziert sind der etwas kleiner als der Durchmesser eines Balles ist (<40mm). Sobald der Ball zwischen diese zwei Räder kommt wird er so stark beschleunigt dass er über das Netz fliegt.





Orientierung

Der Roboter bewegt sich durch zwei unabhängige Motoren, die jeweils auf einer Seite festgebracht sind, und somit sehr wendig ist. Er orientiert sich mit Hilfe von drei Ultraschall-Sensoren, einer vorne und je einer auf den Seiten des Roboters, alle sind tiefer als das Netz, also unter 15cm. Diese Sensoren geben den jeweiligen Abstand zur Wand an, welche in dem Navigations-Programm verwendet werden um sich möglichst sinnvoll zu bewegen. Weiter befindet sich vorne nochmals ein Ultraschall-Sensor, jedoch höher als 15cm. Durch die Differenz der Abstände der beiden vorderen Sensoren wird dann festgestellt ob der Roboter auf das Netz ausgerichtet ist.

Ablauf

Der Roboter beginnt zufällig das Feld abzufahren und Bälle aufzunehmen. Falls ein Ball aufgenommen wurde und dieser die Lichtschranke passiert hat wird der Balltransport gestoppt und der Roboter beginnt sich an der Stelle zu drehen bis er erkennt dass sich das Netz vor ihm befindet. Sobald dies der Fall ist wird der Balltransport fortgesetzt und der Ball wird über das Netz geschossen. Wurde der Ball abgeschossen beginnt Hans-Peter wieder damit das Feld abzufahren und diesen Vorgang zu wiederholen.

Sourcecode Cruise V2 – Gruppe HansPeter Wii:

```
#include "NXCDefs.h"


#define BT_CONN 1

#define OUTBOX 5

#define INBOX 1


#define DD 30


sub BTCheck(int conn){

    if (!BluetoothStatus(conn)==NO_ERR){

       TextOut(5, LCD_LINE2, "Error", false);

       Wait(1000);

       Stop(true);

    }

}


int trigo(int _in_front, int _in_side) {

  int square_front = in_front * in_front;

  int square_side = in_side * in_side;

  int multi = in_front * _in_side;

  int add_ = square_front + square_side;

  int sqrt_ = Sqrt(add);

  int distance = abs(multi / sqrt);


  return _distance;

}


task main(){


 int US_1;
```

```
int US_2;
int US_3;
int US_4;
int distance_l;
int distance_r;

int ack = 0;
int in;
bool ball = false;
BTCheck(BT_CONN);

 SetSensorLowspeed(S1);
 SetSensorLowspeed(S2);
 SetSensorLowspeed(S3);
 SetSensorLowspeed(S4);

while(true){
 ClearScreen();

 US_1 = SensorUS(S1);
 US_2 = SensorUS(S2) + 20;
 US_3 = SensorUS(S3) + 20;
 US_4 = SensorUS(S4) + 20;

 distance_l = trigo(US_2, US_3);
 distance_r = trigo(US_2, US_4);


 if(ball == true){
  OnRev(OUT_A, 100);
  OnFwd(OUT_C, 100);
```

```
 if(abs(US_1 - US_2) >= DD){
  SendRemoteNumber(BT_CONN, OUTBOX, 0x0A);
  ball = false;
 }
}else{
 int brake = 0;
 if ((distance_l < 50) || (distance_r < 50) || (US_2 < 65)){
  if ((distance_l < 37) || (distance_r < 37) || (US_2 < 38)){
   if ((distance_l > distance_r) && (US_2 > 36)){
    OnRev(OUT_A, 100);
    OnFwd(OUT_C, 100);
   }
   if ((distance_l < distance_r) && (US_2 > 36)){
    OnRev(OUT_C, 100);
    OnFwd(OUT_A, 100);
   }
  }else{
   OnFwd(OUT_AC, 60);
  }
 }else{
  OnFwd(OUT_AC, 100);
 }
 int r = Random(10);

 NumOut(10,LCD_LINE1, US_2);
 NumOut(10,LCD_LINE2, US_3);
 NumOut(10,LCD_LINE3, distance_l);
 NumOut(10,LCD_LINE4, distance_r);
 NumOut(10,LCD_LINE5, r);
 NumOut(10,LCD_LINE6, US_1);
 if (r<9){
```

```
    Wait(80);
   }else{
    if ((distance_l < distance_r)){
     OnRev(OUT_C, 100);
     OnFwd(OUT_A, 100);
     Wait(200);
    }
    if ((distance_l > distance_r)){
     OnRev(OUT_A, 100);
     OnFwd(OUT_C, 100);
     Wait(200);
    }
   }
  }
  if(ReceiveRemoteNumber(INBOX, true, in) != STAT_MSG_EMPTY_MAILBOX){
   if(in == 0x0B){
    ball = true;
   }
   SendResponseNumber(OUTBOX, 0xFF);
  }
 }
}
```

Das Herumfahren basiert auf dem Konzept, dass der Roboter anhand der 3 Sensoren 2 Dreiecke bildet und den kürzesten Abstand zur Bande berechnet, sollte dieser einen Schwellenwert unterschreiten, so wird er automatisch wenden. Beim Wenden werden jedoch noch einige Faktoren mit einbezogen. (Abstand vorne, Abstand zur Seite)
Des weiteren passiert die Kommunikation mit dem zweiten NXT. Der Roboter fährt nur solange rum, bis ein Ball aufgenommen wurde. Er erhält dann einen Interrupt im regulären Programm und beginnt sich, nach der Seite auszurichten.

Sourcecode Shoot V2 – Gruppe HansPeter Wii:

```
#include "NXCDefs.h"


#define BT_CONN 0

#define OUTBOX 1

#define INBOX 5


#define BALL_DLIGHT 100


/* Motor Ports:

 *  A: Abschuss links

 *  B: Einzug

 *  C: Abschuss rechts

 *

 * Sensor Ports:

 *  1 & 2: Lichtsensoren

 */


sub BTCheck(int conn){

    if (!BluetoothStatus(conn)==NO_ERR){

       TextOut(5, LCD_LINE2, "Error", false);


       Wait(1000);

       Stop(true);

    }

}


task main(){

     int ack = 0;

     int in;

     BTCheck(0);
```

```
TextOut(5, LCD_LINE1, "Slave receiving", false);
SendResponseNumber(OUTBOX, 0xFF);

OnFwd(OUT_ABC, 100);
SetSensorType(IN_1, IN_TYPE_LIGHT_INACTIVE);
SetSensorType(IN_2, IN_TYPE_LIGHT_ACTIVE);
ResetSensor(IN_1);
ResetSensor(IN_2);

Wait(100);



int idleLight = SENSOR_1;
bool bEinzug = true;

while(true){
     if((idleLight - SENSOR_1) >= BALL_DLIGHT){
          Wait(250);
          Off(OUT_B);
          bEinzug = false;
          SendRemoteNumber(BT_CONN, OUTBOX, 0x0B);
     }
     if(ReceiveRemoteNumber(INBOX, true, in) != STAT_MSG_EMPTY_MAILBOX){
          if(in == 0x0A && !bEinzug){
                OnFwd(OUT_B, 100);
                bEinzug = true;
          }
          SendResponseNumber(OUTBOX, 0xFF);
     }
     Wait(10);
}
```

```
}
```

Der zweite NXT übernimmt die Kontrolle des Aufzugs. Sobald ein Ball aufgenommen wurde sendet er an den ersten NXT einen Wert über die Bluetoothschnittstelle. Dieser kann sich dann ausrichten und diesem NXT die Abschussfreigabe erteilen.

Ferdinand64

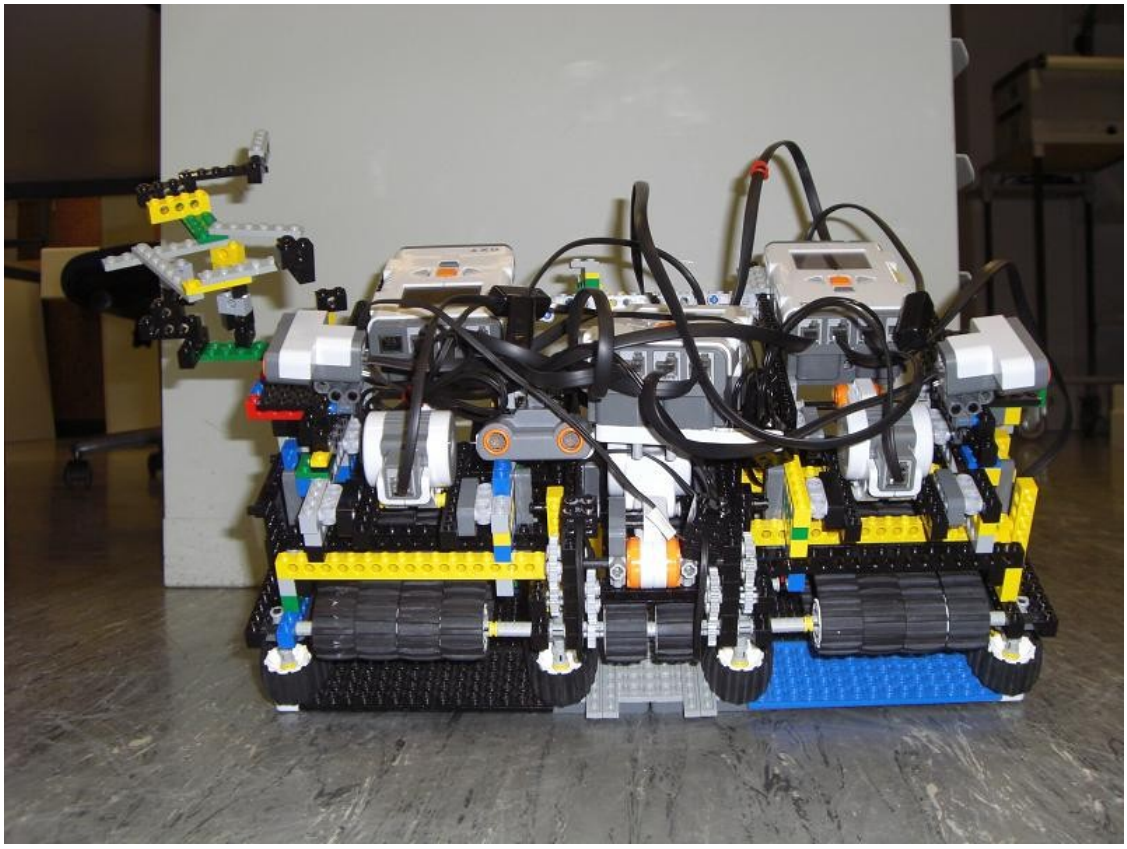


Bild 1: vorne

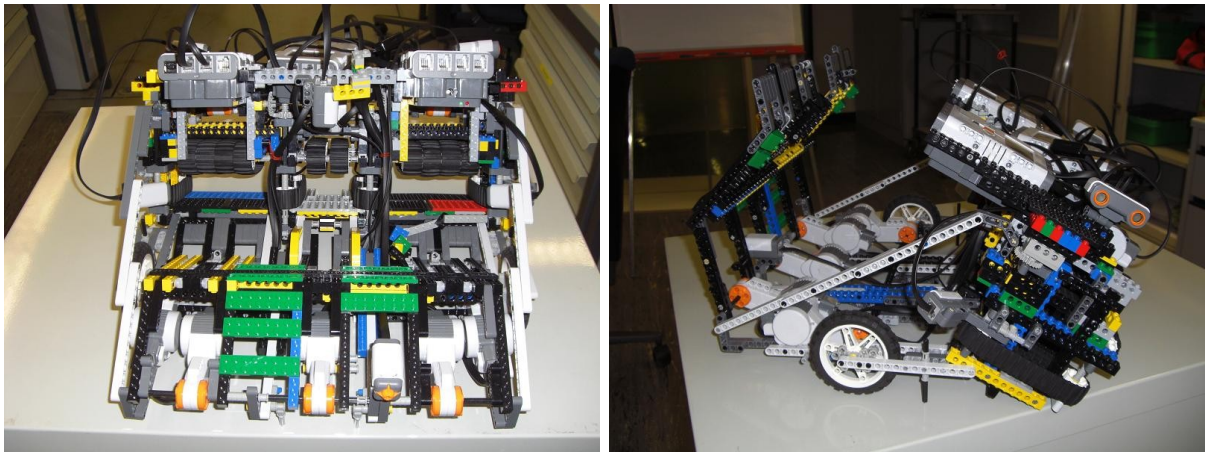


Bild 2: hinten - seite

Ferdinand64 ist ein Roboter auf 4 Rädern. Er besteht aus einer 40cm breiten vorderen Walze, einer gleich breiten hinteren Lade, 3 NXTs (Slave I, Slave II, Master) und aus 7 Sensoren:

-1 Touchsensor

-3 Lichtsensoren

-3 Ultraschallsensoren

Ferdinand64 teilt sich das 2m x 2m grosse Spielfeld in 40cm breite Reihen ein und fährt diese systematisch ab. Dabei muss er zu Beginn mit dem Rücken zum Netz und der rechten Seite zur Bande ausgerichtet werden. (**Bild 3: Vorwärts Routine**)

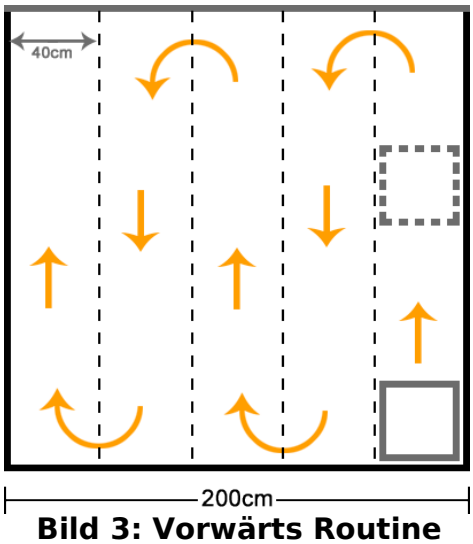


Bild 3: Vorwärts Routine

Er nimmt die Bälle auf mit der Walze im **Bild 4**.

Bei jeder 180° Drehung am Netz fährt er wieder leicht rückwärts zurück und entleert das Lager über das Netz hinweg (**siehe Bild 5: Lade**). Sollte der dazu als Auslöser verwendete Touchsensor nicht innerhalb von 3 Sek. ansprechen, wird das Lager trotzdem entleert, da davon ausgegangen wird, dass Ferdinand64 lediglich ein wenig schräge zum Netz steht und mit grösster Wahrscheinlichkeit die Bälle im

gegnerischen Feld landen.

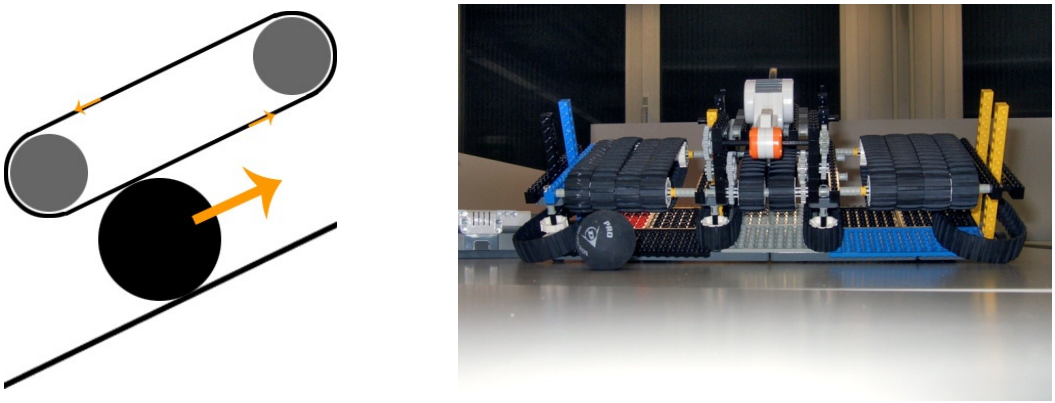


Bild 4: Walze

Sollte das Lager schon vorher mit der in den Spielregeln definierten maximalen Anzahl von Bällen gefüllt sein, meldet dies Ferdinand64 über Slave I per Bluetooth dem Master.

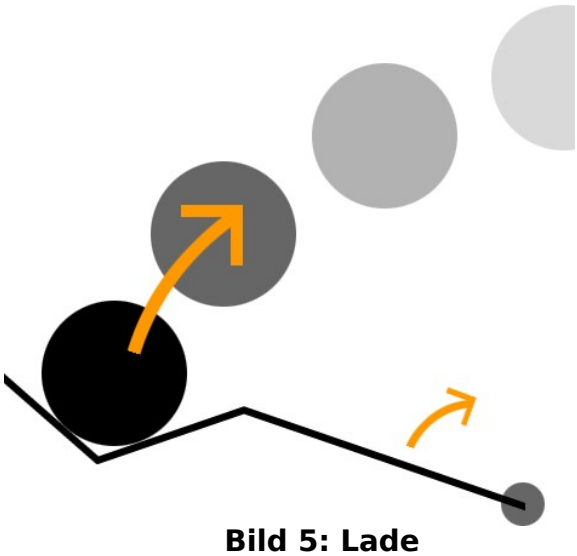


Bild 5: Lade

Gleichzeitig kehrt Slave I die Laufrichtug der Walze um, um zu verhindern, dass weitere Bälle aufgenommen und sich diese womöglich verkannten können. Da dieser Bereich aber bereits von Bällen befreit worden ist, ist es eher unwahrscheinlich, dass sonst noch Bälle im Wege liegen.

Eine Art Kollisionsdämpfer wurde dennoch im hinteren unteren Bereich des Roboters angelegt, sodass sich anstossende Bälle nicht verklemmen können.

Der Master fährt Ferdinand64 schliesslich mit dem Rücken zum Netz stehend zurück bis der dazu angebrachte Touchsensor anspricht oder 10 Sek. verstrichen sind.

Am Netz angekommen wird durch den Aufprall auf den Touchsensor der Befehl zur Entleerung des Lagers per Bluetooth ausgelöst. In der Folge wird das Lager mit den Bällen auf die gegenüberliegende Seite des Netzes entleert.

Sollte sich das Lager auf dem Wege zum Netz gefüllt haben, dreht der Roboter sich weg mit Rücken zum Netz, fährt rückwärts und entleert das Lager nach der üblichen Methode. Das Gleiche gilt für den Fall, dass sich das Lager erst nach der Kurve gefüllt haben soll.

In Reihe 5 kommt dann eine Routine zur Anwendung, die Ferdinand64 zurück zur Startposition fährt, das Lager leert und alle Werte der Variablen zurücksetzt (**siehe Bild 6: Rückkehr Routine**).

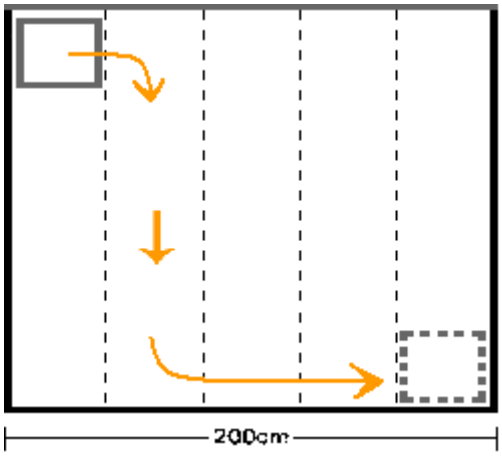


Bild 6: Rückkehr Routine

NXTs Beschreibung:

Master:
Für die Koordination zuständig.

OUT_A: Motor Links (statisch)
OUT_B: Motor rechts (korrigiert)
OUT_C: Lämpchen für Lichtschranke (Für die Sensoren ist eine Lichtquelle vorausgesetzt, damit die Bälle besser erkannt werden.)

IN_1: Ultraschallsensor vorne
IN_2: Ultraschallsensor rechts
IN_3: Ultraschallsensor links
IN_4: Touchsensor hinten

Slave I:
Zählt die Bälle und bedient die Walze.

OUT_AB: Walze
OUT_C: Walze (gegenläufig)

IN_1: Lichtsensor
IN_2: Lichtsensor
IN_3: Lichtsensor
IN_4: leer

Slave II:
Für den Abschuss zuständig.

OUT_ABC: Abschusssystem

Sourcecode Count Gruppe Ferdinand64:

```
#include "NXCDefs.h"
#define TRIGGER 20

#define BT_CONN 1
#define OUTBOX 1
#define INBOX 5

sub BTCheck(int conn){
  if (!BluetoothStatus(conn)==NO_ERR){
     TextOut(5,LCD_LINE2,"Error");
     Wait(1000);
     Stop(true);
  }
}

task main()
{
  //Initialisieren der Variablen
  int counter = 0;
  int ack, in;

  //Prüfen der Verbindung
  BTCheck(0);

  //Initialisieren der Sensoren
  SetSensorType(IN_1,SENSOR_TYPE_LIGHT_INACTIVE);
  SetSensorMode(IN_1,SENSOR_MODE_PERCENT);
  SetSensorType(IN_2,SENSOR_TYPE_LIGHT_INACTIVE);
  SetSensorMode(IN_2,SENSOR_MODE_PERCENT);
  SetSensorType(IN_3,SENSOR_TYPE_LIGHT_INACTIVE);
  SetSensorMode(IN_3,SENSOR_MODE_PERCENT);

  OnRev(OUT_AB, 100);
  OnFwd(OUT_C, 100);

  while (true){
    if (SENSOR_1 < TRIGGER){
      while(SENSOR_1 < TRIGGER){
       //PlayTone(330,400);
      }
      counter++;
      //TextOut(1,LCD_LINE1, anzahl);
    }
    if (SENSOR_2 < TRIGGER){
```

```
      while(SENSOR_2 < TRIGGER){
       //PlayTone(330,400);
      }
      counter++;
    }
    /*if (SENSOR_3 < TRIGGER){
      while(SENSOR_3 < TRIGGER){
       //layTone(330,400);
      }
      counter++;
    }*/
    if(counter == 5){
      Wait(1000);
      OnFwd(OUT_AB, 100);
      OnRev(OUT_C, 100);
      TextOut(0, LCD_LINE1, "Lager voll");

      until(ReceiveRemoteNumber(INBOX,true,ack) != STAT_MSG_EMPTY_MAILBOX
&& ack == 2){
        SendResponseNumber(OUTBOX,5);
      }
      //TextOut(0, LCD_LINE2, "Info gesendet");
      while(true){
       if(ReceiveRemoteNumber(INBOX,true,ack) != STAT_MSG_EMPTY_MAILBOX &&
ack == 3)  {
         TextOut(0, LCD_LINE1,  "Lager leer");
         SendResponseNumber(OUTBOX,4);
         counter = 0;
         OnRev(OUT_AB, 100);
         OnFwd(OUT_C, 100);
         break;
       }
      }
    }else{
        NumOut(0, LCD_LINE1, counter);
    }
  }
  }
```

Sourcecode Go Gruppe Ferdinand64:

```
#include "NXCDefs.h"

#define  MIN_WALL_DISTANCE 10      //cm
#define MAX_TURN_DISTANCE 20       //cm
#define RAND 5                  //cm Sicherheitsabstand
#define DREHRADIUS 23           //cm (20cm (halbe Roboterbreite) + 5cm (Radius)
#define DREHLATENZ 2500          //Zeit für Viertelsdrehunh
#define SHOOT_DISTANCE 160       //Abstand nach vorne, bei dem
Abschussposition erreicht
#define GRAD 1775

//Bluetooth
#define BT_WALZE 1
#define W_OUTBOX 5
#define W_INBOX 1

#define BT_SHOOT 2
#define S_OUTBOX 7
#define S_INBOX 3


  //Initialisieren der Variablen
  int row = 1;
  int baelle = 0;   //können durch angeschlossenen Sensor oder per Bluetooth
inkrementiert werden
  bool halten = 0;
  int direction = 1;
  int tuning = 0.0;

sub BTCheck(int conn){
 if (!BluetoothStatus(conn)==NO_ERR){
  TextOut(5,LCD_LINE2,"Error");
```

```
  Wait(1000);
  Stop(true);
 }
}
/*
Start: rechte Wand, am Netz
Abstandssensor: vorne mitte und rechts
*/
sub kurve();      //führt eine entsprechende Kurve aus (rechts/links)
sub get_target(); //prüft aktuelle Position und fährt in Abschussstellung
sub shoot();                    //löst Abschussmechanismus aus
sub tune();
sub check_tune();

task main ()
{
  //Initialisieren der Sensoren
  SetSensorLowspeed(IN_1);   //Abstand vorne
  SetSensorLowspeed(IN_2);  //Abstand rechts
  SetSensorLowspeed(IN_3);  //Abstand links
  SetSensorType(IN_4, SENSOR_TYPE_TOUCH);
  SetSensorMode(IN_4, SENSOR_MODE_BOOL);

  OnFwd(OUT_C, 100);

  int ack, in;
  BTCheck(BT_WALZE);
  BTCheck(BT_SHOOT);

  /*while(SensorUS(IN_1)<150){
    Off (OUT_AB);
    Wait(2000);
  }*/
  while(true){
```

```
    while(SensorUS(IN_1)>MIN_WALL_DISTANCE){
      if (ReceiveRemoteNumber(W_INBOX,true,in) != STAT_MSG_EMPTY_MAILBOX
&& in == 5) {
       SendRemoteNumber(BT_WALZE, W_OUTBOX,2);  //Bestätigung
       PlayTone(220,400);
       get_target();
       until(ReceiveRemoteNumber(W_INBOX,true,ack) !=
STAT_MSG_EMPTY_MAILBOX && ack == 4){
         SendRemoteNumber(BT_WALZE, W_OUTBOX,3);
       }
       in = 0;
      }
      tune();
      OnFwd(OUT_A, 80);  //Antrieb links
      OnFwd(OUT_B, tuning);  //Antrieb rechts
    }
    kurve();
  }
}
sub kurve(){
    if(row==5){
      while(SensorUS(IN_1)<(DREHRADIUS+RAND)){
        OnRev(OUT_A, 80);  //Antrieb links
        OnRev(OUT_B, 80);    //Antrieb rechts
      }
      Off(OUT_AB);

      RotateMotor(OUT_A, 80, GRAD); //180° Rechtskurve
      Off(OUT_AB);
      while(SensorUS(IN_1)>(DREHRADIUS+RAND)){    //row 4
        tuning = 50*10/SensorUS(IN_2);
        check_tune();
        OnFwd(OUT_A, 80);  //Antrieb links
        OnFwd(OUT_B, tuning);  //Antrieb rechts
```

```
    }
    Off(OUT_AB);
    RotateMotor(OUT_B, 80, GRAD/2); //90° Linkskurve
    Off(OUT_AB);
    while(SensorUS(IN_1)>(DREHRADIUS+RAND)){      //quer passieren
      tuning = SensorUS(IN_2)*10/10;
      check_tune();
      OnFwd(OUT_A, 80);    //Antrieb links
      OnFwd(OUT_B, tuning);   //Antrieb rechts
    }
    Off(OUT_AB);
    RotateMotor(OUT_B, 80, GRAD/2); //90° Linkskurve
    Off(OUT_AB);
    int timer = 0;
    while(!SENSOR_4 && timer < 300){
      timer++;
      OnRev(OUT_A, 80);    //Antrieb links
      OnRev(OUT_B, 80);      //Antrieb rechts
      Wait(10);
    }
    Off(OUT_AB);
    shoot();
  row = 1;
  direction = row%2;
}else{
  while(SensorUS(IN_1)<(DREHRADIUS+RAND)){
     OnRev(OUT_A, 80);    //Antrieb links
     OnRev(OUT_B, 80);      //Antrieb rechts
  }
  Off(OUT_AB);
  if(direction == 1){
    RotateMotor(OUT_B, 80, GRAD);
  }else{
    RotateMotor(OUT_A, 80, GRAD);
```

```
      }
      Off(OUT_AB);
      row++;
      direction = row%2;
      int timer = 0;
      if(direction == 1){
        while(!SENSOR_4 && timer < 300){
          timer++;
          OnRev(OUT_A, 80);    //Antrieb links
          OnRev(OUT_B, 80);      //Antrieb rechts
          Wait(10);
        }
        Off(OUT_AB);
        shoot();
      }
    }
}
sub get_target()
{
  if(direction == 0){   //noch nicht mit Rücken zum Netz
    while(SensorUS(IN_1)>(DREHRADIUS+RAND)){
      tune();
      OnFwd(OUT_A, 80);    //Antrieb links
      OnFwd(OUT_B, tuning);   //Antrieb rechts
    }
    kurve();
  }
  int timer = 0;
  while(!SENSOR_4 && timer < 1000){
      timer++;
      OnRev(OUT_A, 80);    //Antrieb links
      OnRev(OUT_B, 80);      //Antrieb rechts
      Wait(10);
  }
```

```
    Off(OUT_AB);
    TextOut(0, LCD_LINE1, "shoot();");
    shoot();
}
sub shoot(){
    int ack;
    SendRemoteNumber(BT_SHOOT, S_OUTBOX,1);
    while(true){
      if(ReceiveRemoteNumber(S_INBOX,true,ack) != STAT_MSG_EMPTY_MAILBOX &&
ack == 1)  {
        break;
      }
    }
}
sub tune(){
    if(row <=3 && direction == 1){
      tuning = ((row-1)*40+10)*10/SensorUS(IN_2); //ok
    }else if(row <=3 && direction == 0){
      tuning = (SensorUS(IN_3)*10/((row-1)*40+10)); //ok
    }else if(row > 3 && direction == 1){
      tuning = SensorUS(IN_3)*10/(200-(row*40)+10); //ok
    }else if(row > 3 && direction == 0){
      tuning = (200-row*40+10)*10/SensorUS(IN_2); //ok
    }
    TextOut(0, LCD_LINE1, "    ");
    NumOut(0, LCD_LINE1, tuning*8);
    check_tune();



}
sub check_tune(){
    if(!(tuning*(8)>100) && !(tuning*(8)<60)){
      tuning = tuning*(8);
    }else{
```

```
      if(tuning*(8)>100){
        tuning = 100;
      }else{
        tuning = 60;
      }
    }
}
```

Sourcecode Shoot Gruppe Ferdinand64:

```
#include "NXCDefs.h"

//Bluetooth
#define BT_SHOOT 2
#define S_OUTBOX 3
#define S_INBOX 7

sub BTCheck(int conn){
  if (!BluetoothStatus(conn)==NO_ERR){
    TextOut(5,LCD_LINE2,"Error");
    Wait(1000);
    Stop(true);
  }
}
task main(){
  int in;
  BTCheck(0);
  while(true){
    if (ReceiveRemoteNumber(S_INBOX,true,in) != STAT_MSG_EMPTY_MAILBOX &&
in == 1) {
      RotateMotor(OUT_ABC, 100, 66);
      Wait(3000);
      RotateMotor(OUT_ABC, -100, 66);
      Wait(500);
      SendResponseNumber(S_OUTBOX,1);
    }
    Wait(500);
  }
}
```

PPS Lego-Mindstorms

Volleyball Regelwerk für Lego-Mindstorms

Ideen und Konzept: Martin Stypinski <martisty@ee.ethz.ch>
Zelglistrasse 42
8122 Binz

Inhaltsverzeichnis

1 Allgemeine Regeln

- Die Mechanik des Roboters wird aus Lego aufgebaut. Zu Hilfe können jedoch auch andere Materialien verwendet werden.
- Für die Programmierung ist der NXT-Brick der Hauptbestand teil. Selbstgebautes Zubehör, darf ausschliesslich nur über den NXT angesteuert werden.
- Der Roboter darf die Spielarena oder umliegende Bereiche nicht zerstören, beschädigen oder verschieben.
- Der Roboter muss über über ein passives Grundverhalten verfügen. Sie dürfen Zuschauer, Hindernisse oder andere Roboter nicht anfahren (rammen), beschädigen oder gewollt behindern. Es sind somit auch keine Einrichtungen erlaubt, die dazu gedacht sind, den Gegner zu stören (Störsender, Blenden mit Scheinwerfern, Aufstellen von Hindernissen, Flüssigkeiten, Rauch u.s.w.)
- Der Roboter muss sich selbstständig und ohne Fernsteuerung in der Arena zurecht finden.
- Der Roboter muss über einen gut zugänglichen Not-Aus schalter verfügen (NXT nicht abdecken)
- Drohnen sind während dem Spiel erlaubt. Müssen jedoch nach dem Start abgesetzt werden.
- Die Standfläche beim Start beträgt: 400x400x400 [mm]
- Der Antrieb darf keinerlei Spuren hinterlassen.

2 Spielfeld

- Grösse: 2000x4000mm
- Banden: Das ganze Spielfeld ist mit weissen Banden abgegrenzt. die Höhe beträgt 300mm
- Netz: Das Netz ist rot. Die Netz höhe beträgt 150mm

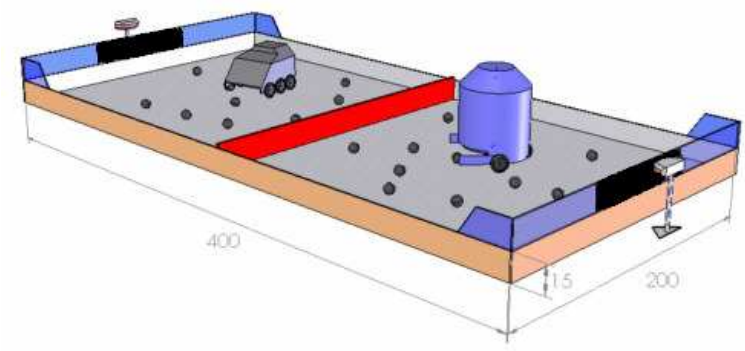


Abbildung 1: Das Spielfeld in 3D Ansicht

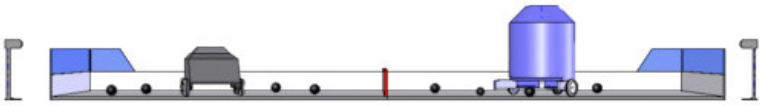


Abbildung 2: Das Spielfeld in top Ansicht

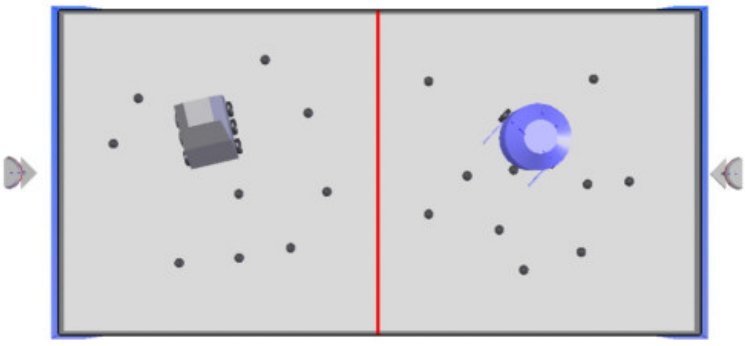


Abbildung 3: Das Spielfeld in front Ansicht

3 Spielverlauf

3.1 Spielanfang

In einem Spielfeld mit zwei Hälften und einem "Netzïn der Mitte treten jeweils zwei Roboter gegeneinander an. Sie müssen versuchen, möglichst viele Bälle aus der eigenen Hälfte in die des Gegners zu befördern. Eine Partie dauert 3 Minuten, danach müssen die Roboter stoppen oder gestoppt werden. Nun werden die Bälle in den beiden Hälften gezählt.

- Befördert ein Roboter einen Ball aus dem Spielfeld heraus, so wird dieser wieder in das Spiel gebracht. Und zwar in das Feld, aus dem er kam.
- Greift ein Spieler über das Netz, gibt es zwei Strafpunkte
- Fährt ein Roboter Wände oder das Netz um (mehr als ein leichter Kontakt!), gibt es zwei Strafpunkte

3.2 Wertung

- Punkte(SpielerA) = BälleSpielfeldB - BälleSpielfeldA - Strafpunkte
- Punkte(SpielerB) = BälleSpielfeldA - BälleSpielfeldB - Strafpunkte
- Wer mehr Punkte hat, gewinnt.

3.3 Der Roboter

- Die Roboter dürfen zu Beginn des Matches beliebig in der eigenen Hälfte platziert werden.
- Es ist wünschenswert, aber nicht notwendig, dass die Roboter das Match automatisch zu Spielende beenden.
- Die Roboter dürfen die Bälle aufnehmen, tragen, transportieren, werfen, rollen, pusten, saugen, kicken, heben etc.
- Die Roboter dürfen nicht mehr als 5 Bälle sammeln. Wenn der Roboter 5 eingesammelt hat, muss er diese zunächst in die gegnerische Hälfte bringen.
- Bälle gelten als gesammelt, wenn sie vom Boden entfernt werden. Es soll damit vermieden werden, dass ein Roboter Bälle sammelt und 3 sek. vor Schluss alle auf einmal zum Gegner rüberkippt.
- Die Roboter dürfen Bälle nicht mit solcher Energie abschiessen, dass Arena, Mitspieler oder gar Zuschauer gefährdet werden können!

3.4 Orientierung

Zur Orientierung dient die Bande, diese ist am Rand 30cm hoch und weiss, während in der Mitte (Netz) die Bande rot und nur 15cm tief ist.

3.5 Die Bälle

Als Spielbälle werden etwa 30 Sqash-Bälle d=40mm verwendet (oder nur so viele, wie eben da sein werden...). Diese können einfach beschafft werden. Die Bälle werden, nachdem die Roboter aufgestellt wurden, (vom Publikum) zufällig in der Arena verteilt.