2007

Otto-von-Guericke
Universität Magdeburg

Florian Tanke, Nico
Gebauer, Stefan Kirst

[SOFTWAREPRAKTIKUM TEAMROBOTIK]

Programmierung von Steuerungsalgorithmen für mobile Roboter (LEGO-NXT)

Inhaltsverzeichnis

Motivation

Wir haben uns für das Softwarepraktikum „Teamrobotik“ entschieden, da wir so das erste mal selber ein Autonomes System programmieren konnten und dabei auch noch ein bisschen „Spielspaß“ für uns drin war. Ein weiterer Vorteil war es, dass man hier nicht nur theoretische Arbeit vollrichten musste, sondern sich auch mal ein bisschen praktische betätigen konnte indem man sich eine sinnvolle und funktionale Konstruktion für den Roboter ausdachte.

Die Aufgabe

Kollisionsfreie Navigation durch einen Parcours von variabel positionierbaren Hindernissen. Dabei soll ein Signal gebendes Objekt identifiziert, angesteuert und aus dem Parcours zu einem Zielpunkt befördert werden. Je nach verbleibender Projektzeit sollte versucht werden, dass diejenigen Roboter, die den Ball nicht in Besitz bringen konnten, dem Ball besitzenden Roboter den Zugang zum Zielpunkt zu versperren.

Zielsetzung

Eine wirkungsvolle Programmierung der Sensoren und Motoren einschließlich einer geschickten Verwaltung und Nutzung der Ressourcen CPU, Speicher und Kommunikation soll helfen, zukünftig ein originelles Anspielverhalten zu realisieren.

Die Teilaufgaben

Teilaufgabe 1

Unser erster Roboter

Zur Annäherung an die gegebene Hardware und Software haben wir uns erstmal ein Roboter nach der mitgelieferten Anleitung gebaut.

Mit diesem ersten Roboter konnten wir nun die ersten Tests durchführen. Zunächst haben wir die Sensoren nur direkt über den NXT ausprobiert (Try Me- Modus). Dabei haben wir festgestellt, dass sich die Sensoren so verhalten haben wie erwartet. Interessant für spätere Arbeiten fanden wir, dass man die Drehbewegungen der Motoren nachverfolgen kann.

Die Programmierumgebungen

Nachdem wir uns nun mit der Hardware auseinandergesetzt haben, stellte sich für uns die Frage welche Programmierumgebung nutzen wir?

Zur Auswahl standen:

- NXC mit der BricxCC- Umgebung
- Lego- Mindstorm- Software
- Lejos NXJ
- ROBOLAB

Die Lego- Mindstorm- Umgebung

Hier war uns schnell klar, dass diese graphische Umgebung nicht für wissenschaftliche Verwendung geeignet ist. Sie ist zwar sehr einfach zu bedienen, aber ist nicht umfangreich genug um komplexere Aufgaben zu lösen.

Vorteile	Nachteile
–Einfache Bedienung durch Drag & Drop –Leichtes Erstellen von Programmen, da vorgefertigte Module vorhanden sind –Keine Programmierkenntnisse nötig	–relativ unflexibel –Durch die Module nur bedingt zur Realisierung von eigenen Ideen geeignet

Robolab

Diese Umgebung wird zwar im Internet als eine sehr gute und umfangreiche Umgebung diskutiert, jedoch ist hier der Nachteil, dass diese eine kommerzielle Umgebung ist.

Lejos

Lejos stellt eine Java- Umgebung zur Programmierung des NXT-Bausteines dar, wobei hier die Nachteile sind, dass sich die aktuelle Version noch in der Alpha- Phase befindet und das sich die Java- Einbindung relativ schwierig gestaltet.

BricxCC

Das Bricx Command Center ist eine C-ähnliche Möglichkeit zur Programmierung des NXT-Bausteins, wir haben uns hier schnell einarbeiten können und uns sofort wohlgefühlt. Die Möglichkeit der Code-unabhängigen Ansteuerung des Bausteins für Funktionalitätstests und das einfache aber mächtige Auftreten der Umgebung überzeugten. Es bieten sich hier alle Möglichkeiten der Implementierung und Ausnutzung der Fähigkeiten des Roboters.

Somit viel unsere Wahl auf diese Umgebung

Teilaufgabe 2

Was sind Sensorschnittstellen? In welcher Form liegen Messdaten vor? Welchen Typus sind sie?

Qualitätsbestimmung: Logging von Sensordaten und deren Speicherung für die Ermittlung von Kennlinien.

Stellt die Akkuleistung eine kritische Leistungsgröße für die Kommunikationsqualität oder Sensoren dar?

Akkuleistung

Eine schwächer werdende Akkuleistung hat zur Folge, dass die Motoren nicht mehr die Angestrebte Leistung erbringen, Sensoren falsche bzw. ungenügende Messwerte liefern und die Bluetooth-Verbindung zusammenbricht.

Bluetooth

Die Bluetooth-Einheit des NXT-Bausteins ist als kabellose Kommunikationsvariante durchaus geeignet. Wir haben hier eine Reichweite von ca. 25 Metern feststellen können, jedoch hängt diese auch stark von der Umgebung ab. Bei größeren Entfernungen bricht die Verbindung zusammen bzw. ist nicht mehr herstellbar. Desweiteren ist es möglich über Bluetooth mit anderen Robotern zu kommunizieren.

Die Sensoren und ihre Messwerte

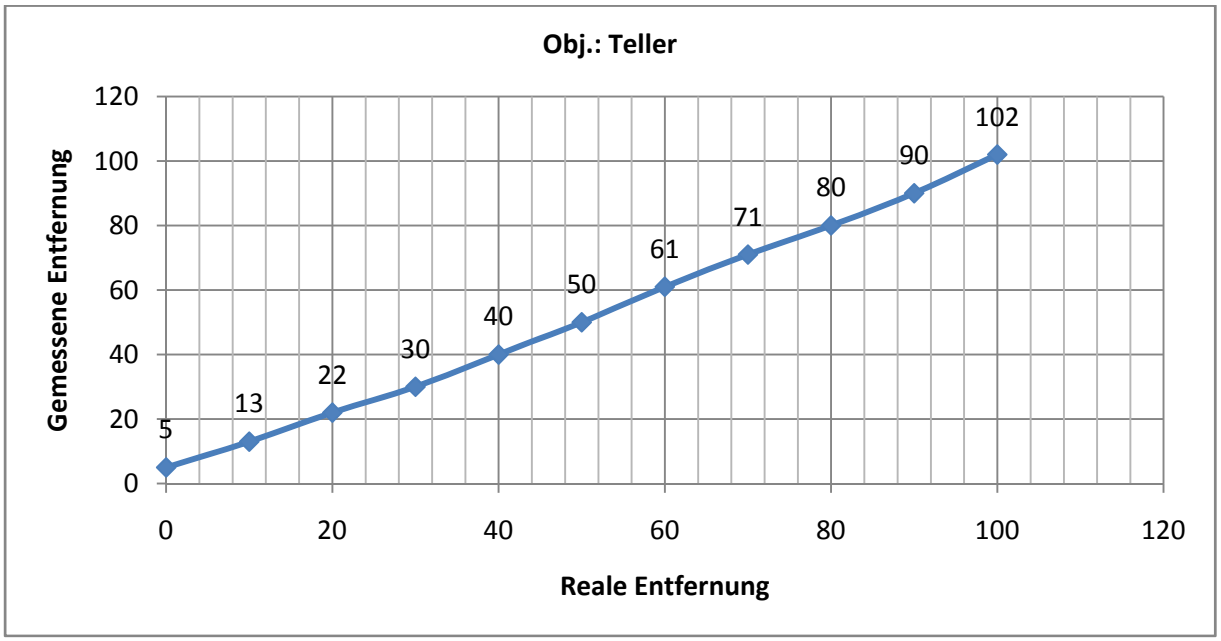
Welche Messwerte liefern welche Komponenten; in welchen Größenordnungen? Feststellung der Messwertgüte unter versch. Bedingungen.

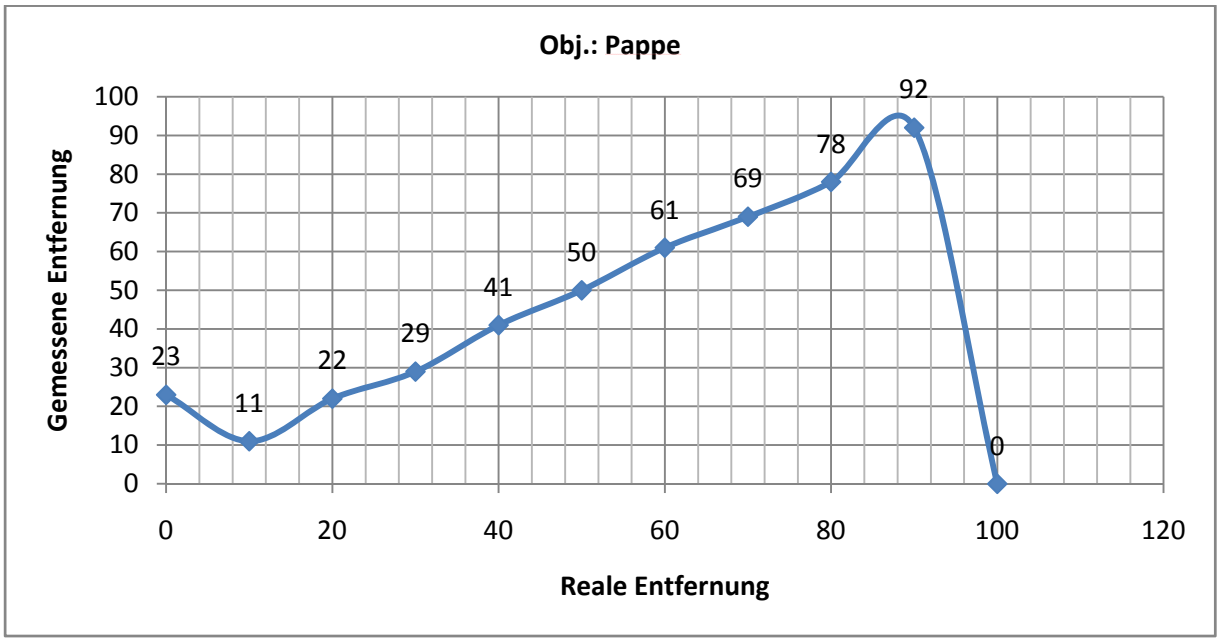
Vorgehensweise: Für Licht-, Ultraschall- und Akustiksensor wird ein Bandmaß auf den Boden gelegt und dann ein Objekt in diskreten Abständen vom zu testenden Sensor fortbewegt.

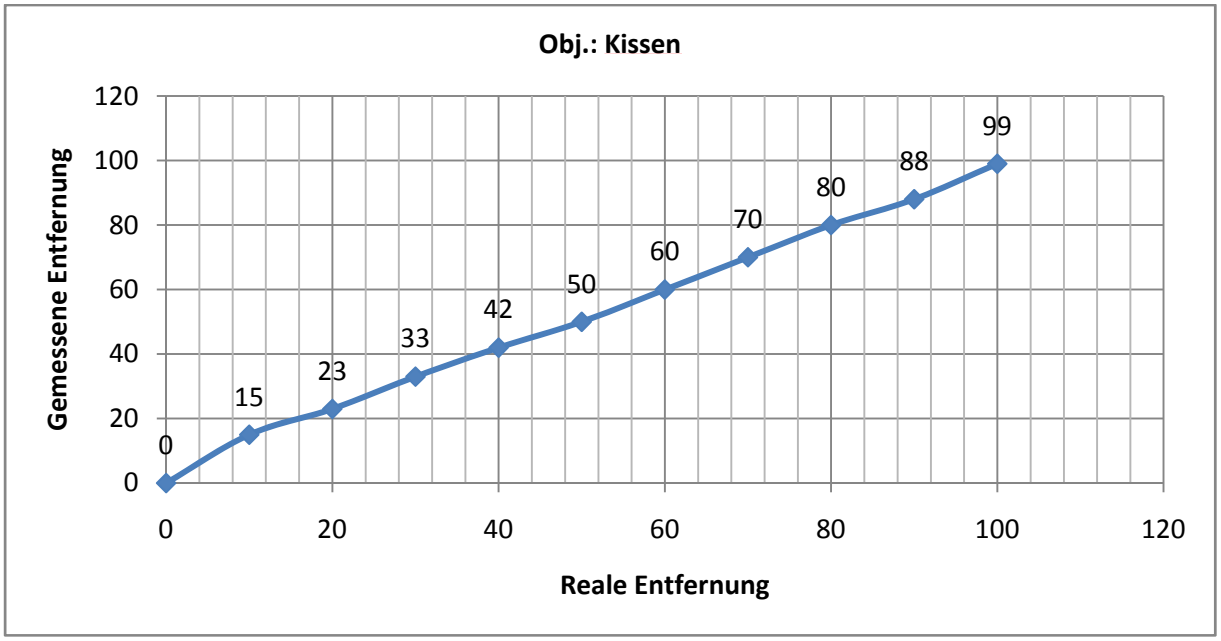
Die verschiedenen Sensoren und ihre Eigenschaften:

Sensor	Einheit	Verwendung
Ultraschallsensor	Entfernung in cm und inch	Entfernungsbestimmung
Lichtsensor	Helligkeit in Prozent	Suche nach Lichtquellen
Soundsensor	Lautstärke in dB und dBA	Steuerung per Geräusche
Tastsensor	Wert 1 oder 0	Erkennung physischer Kontakte
Motoren	Rotationen im Gradmaß	Erfassung der Umdrehung eines Motors

Der Ultraschallsensor

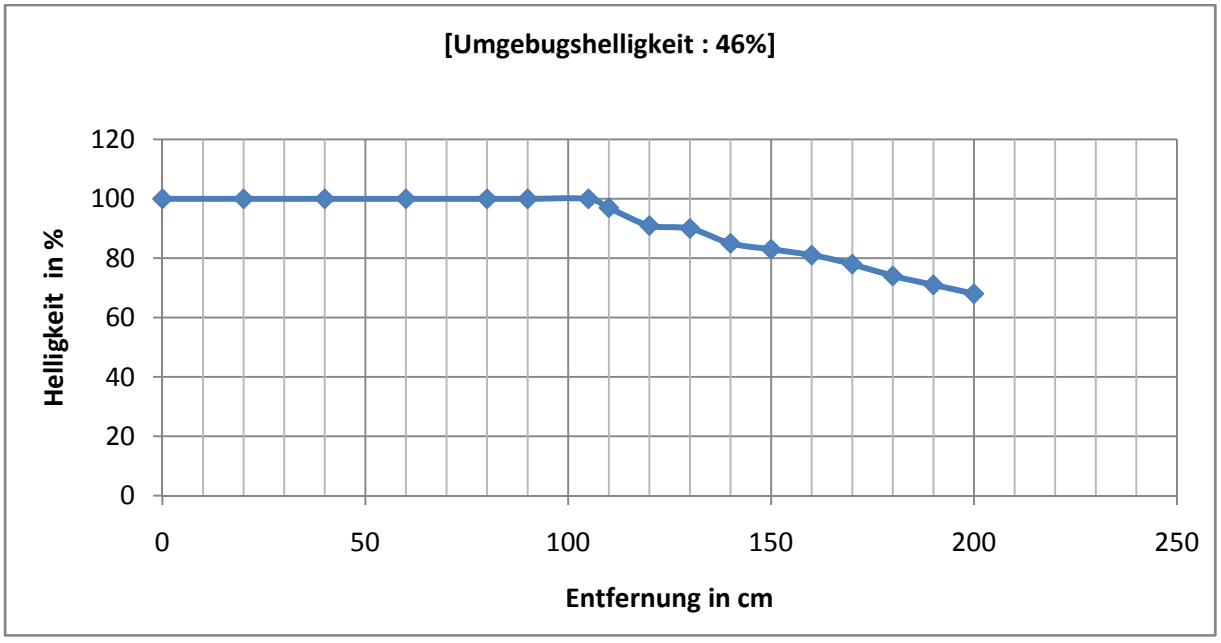






Durch die Messungen die wir mit dem Ultraschallsensor gemacht haben, können wir abschließend sagen, dass der Sensor mit einer Abweichung von ca. ± 3cm relative genau arbeitet und das die Reichweite ca. 150 cm beträgt, wobei diese vom Material abhängig ist.

Der Lichtsensor

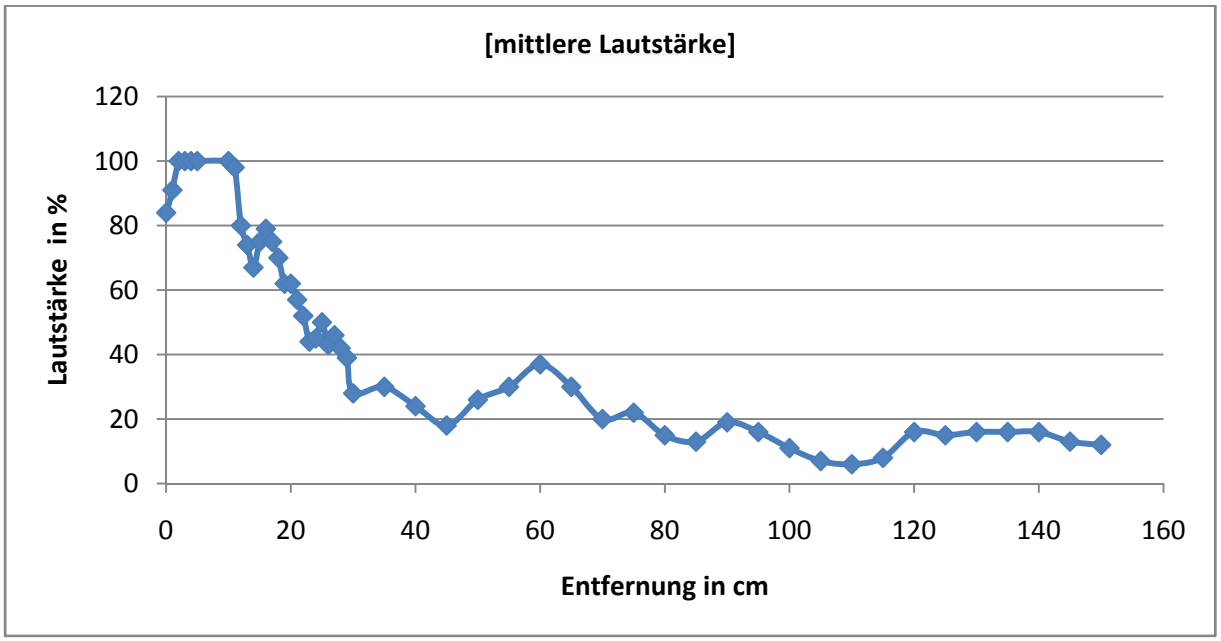


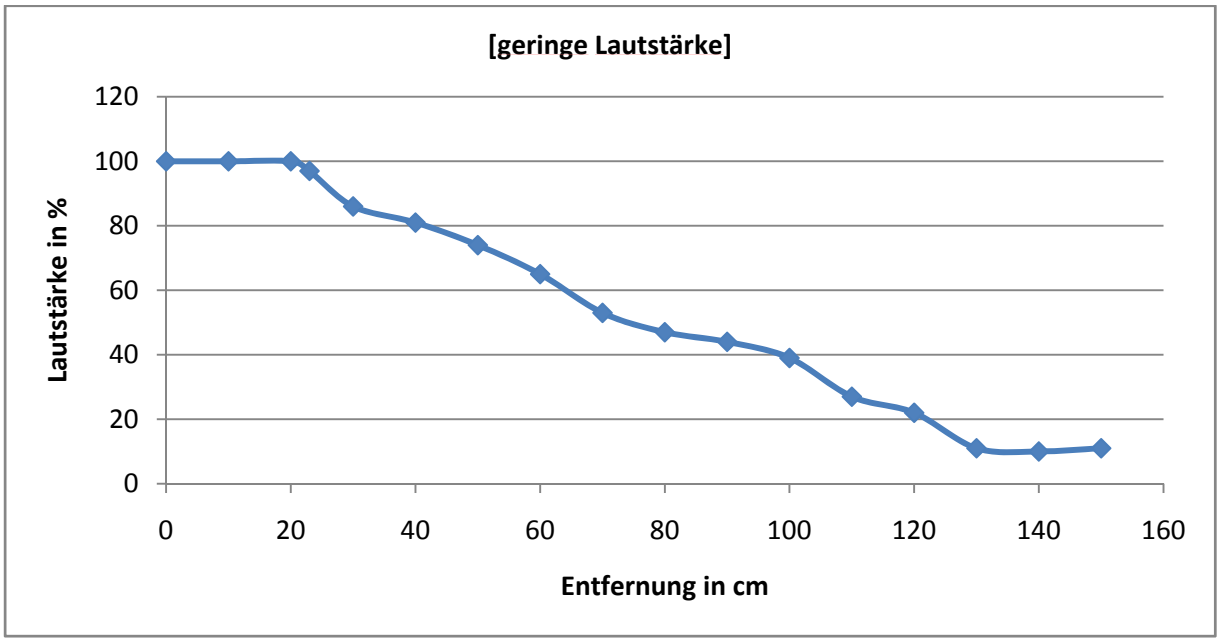
Der Lichtsensor hat zwei Operationsmodi, Ambientlight und Reflectedlight. Mit dem Modi Ambientlight misst man die Umgebungshelligkeit und mit dem Modi Reflectedlight können Farben gemessen werden.

Der Nachteil des Sensors ist, dass er in einem hellen Raum stetig 100% misst und somit der Gebrauch in einer solchen Umgebung keinen Nutzen bringt.

Bei dunkler und mittlerer Raumhelligkeit sind die Ergebnisse allerdings gut.

Der Soundsensor





Beim Soundsensor waren über den gesamten Messbereich starke Schwankungen zu beobachten, was eine genaues Messen schwierig machte. Bei Lautstärkemessungen einer lauten Quelle liefert der Sensor stets volle 100%.

Der Tastsensor

Der Tastsensor ist ein rein analoger Sensor und dient zu Kollisionserkennung. Um ihn auszulösen ist ein Gewicht von 34 Gramm nötig, was einer Kraft von 0,34 Newton entspricht.

Die Motoren

Beim Test der Motoren ist uns aufgefallen, dass (komischerweise) kein Leistungsabfall bei höherem Gewicht festzustellen ist.

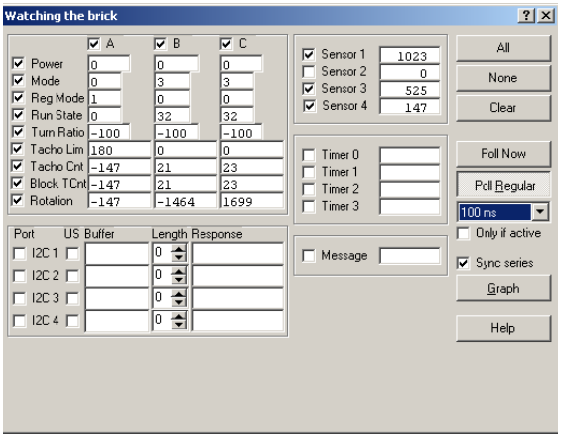
Unsere Messungen auf einer Strecke von 1,5 Metern:

Gewicht in g	Zeit in s
Eigengewicht	8,3
250	8,2
500	8,0

Wir haben desweiterem festgestellt das mit schwächer werdendem Akku die Motorleistung abnimmt und uns ist auch aufgefallen, dass die Motoren unter Volllast mehr Stromverbrauchen.

Teilaufgabe 3

Ein Teil dieser Aufgabe war es Messdaten vom NXT Roboter an eine Logging- Station (Laptop) zu senden, dort zu speichern und zu visualisieren. Dies konnte erstmal ganz einfach mit dem Tool „Watch the Brick“ von BricxCC realisiert werden. Hier kann man alle Werte von den Sensoren und Motoren ablesen.



Der zweite Teil dieser Aufgabe war es den IR-Ball zu detektieren. Dies kann man ganz einfach mit dem Ambientlightmodus realisieren. Im Modus Refelectedlight haben wir dies nicht ausprobiert und können daher nichts weiter dazu sagen!
Desweiteren sollten wir, sowie es uns möglich war, den Roboter per Bluetooth steuern was sich auch recht leicht realisieren lies. Man musste nur den passenden Bluetooth-Treiber installieren damit der PC den NXT finden kann und dann konnte man sich via der BricxCC- Umgebung mit dem NXC verbinden um ihn so über das Steuerkreuz zu bedienen.

Lösung des Hauptszenarios

Konstruktion und Besonderheiten

Zu Beginn stand für uns die Konstruktion im Mittelpunkt. Wie muss unsere Roboter aussehen um die gegebenen Aufgaben und Teilaufgaben erfüllen zu können. Für uns war sofort klar, es muss ein Kettenfahrzeug sein. Abgesehen von der Funktionalität der Ketten ist das Aussehen auch deutlich beeindruckender als das eines normalen mit Rädern betriebenen Fahrzeugs. Die Ketten bieten den einmaligen Vorteil, eine 180° Drehung auf der Stelle zu vollziehen, ohne weitere konstruktive Maßnahmen am Roboter, wie zum Beispiel eine Lenkvorrichtung, vorzunehmen. So ermöglichen gegeneinander laufende Kette eine nahezu perfekte Punktwendung.
Die Zweite Frage die sich uns stellte war die Positionierung der einzelnen benötigten Sensoren. So spielt die Lage des Ultraschallsensors zum Distanzmessen sowie die des Lichtsensors eine sehr wichtige Rolle für den weiteren Bau des Roboters. Verdecken einzelne Teile wie Kabel oder etwa der Greifarm den Einzugsbereich des Sensors und schränken somit seine Funktionalität ein? Wichtige Fragen die bei der Konstruktion zu beachten waren.
Der letzte gravierende Punkt der Konstruktion des Roboters war die Art des Greifmechanismusses. Soll es eine Art Zange sein die von beiden Seiten den Ball umschließt, ein einseitiger Greifarm der den Ball in sich aufnimmt oder etwa eine Kombination aus mehreren Techniken? Wir haben uns für eine von oben schließbare Greifer-Zangen-Kombination entschieden. So befindet sich an der Front des Roboters eine breite trichterförmige Zange die den Ball in sich aufnimmt und zum Mittelpunkt des Roboters führt, während sich abschließend von oben ein Greifer absenkt um den Ball zu fixieren.

Als die Grundlegenden Details der Konstruktion geklärt waren, ging es nun darum die verschiedenen Ideen und Techniken für den Bau umzusetzen und auszuprobieren. So durchlief unser Roboter mehrere einzelne Bauphasen die sich mehr oder weniger von der vorherigen stark unterschieden. Sämtliches Getüftel, Ausprobieren und Gebastel führten dann letztendlich zu der fertigen Konstruktion mit dem Namen „Frank the Tank“.

„Frank the Tank“ oder auch kurz FtT verfügt über verschiedene Technologien:
- → 2 Motoren die den Kettenantrieb realisieren
- → 1 Motor der die Steuerung des Greifarmes übernimmt
- → an den Greifarm montierte Sensoren die somit Schwenkbar sind und einige Vorteile bringen

Eine Besonderheit von FtT ist der schon eben angesprochene Greifarm mit den Sensoren. Während der Testphase sind wir auf mehrere Probleme gestoßen, die explizit die Sensoren betreffen. Von der Ungenauigkeit und der Beeinflussbarkeit durch die Umwelt mal abgesehen, war ein weiteres großes Problem die Verstauung des Balls und die damit folgende Irritation der Sensoren. Befanden sich die Sensoren in der Mitte von FtT, so war der Lichtsensor bei eingefangenem Ball durch dessen Licht irritiert und nicht mehr in der Lage eine sekundäre Lichtquelle ausfindig zu machen. Das Selbe galt für den Ultraschallsensor, der in diesem fallen den Ball direkt vor sich hatte und keine weiteren Distanzen mehr messen konnte. Um dieses Problem zu lösen, ließen wir uns einen wie wir meinen „genialen“ Trick einfallen. Wenn der Greifarm sich zum Fangen des Balls um 180° dreht, dann bedeutet dies, dass jedes an ihm befindliche Gerät auch um denselben Wert verändert wird. Für

Sensoren bedeutet dies, dass Sie in die Entgegengesetzte Fahrtrichtung zeigen, was wiederum heißt, dass der gefangene Ball kein Hindernis mehr für die Sensortechnik darstellt, und somit eine fehlerfreie Weiterverwendung der Sensoren möglich ist.

Programmiertechnische Vorgehensweise

Nachdem die Konstruktion nun umgesetzt war und ein fertiger „Frank the Tank“ vor uns stand, hieß es nun der ganzen Maschine Leben einzuhauchen, Sie zu Programmieren. Aber was müsste FtT in der Lage sein von selbst zu leisten um die gegebenen Aufgaben zu erfüllen? Und vor allem, was ist überhaupt möglich zu realisieren? Al s wir uns diese Fragen stellen stießen wir auf folgende für uns relevante Antworten. Frank muss:

- ➔ Eine Lichtquelle selbstständig ausfindig machen können
- ➔ Lichtquellenposition fixieren
- ➔ Auf auftretende Hindernisse reagieren (ausweichen)
- ➔ Den Ball einfangen
- ➔ Den Ball zu einer anderen Lichtquelle transportieren

Diese und weitere Teilaufgaben lösten wir in einzelnen Tasks.

Der erste und zugleich auch wichtigste Task dem wir uns widmeten, ist der Task „search_light“.

```
task search_light(){
     PlayTone(500,100); Wait(100); PlayTone(750,100); Wait(100); PlayTone(250,100);  Wait(100); PlayTone(500,200); Wait(200);
     bool move = true;
     maxLight  = 0;
     int turn  = 0;
     int temp  = 0;
     int lichtschwelle = 200 ;
     int rotate = MotorRotationCount(OUT_C);
     int fullturn=3000;
     if(contact==false){fullturn=2200;}
     int circle= rotate + fullturn;
     while(move){
                 OnFwd(OUT_C,65);
                 OnRev(OUT_B,65);
                 Wait(150);
                 temp = Sensor(IN_4);
                 rotate = MotorRotationCount(OUT_C);                    //falls des nich geht rotate aufsummieren
                 if ( maxLight < temp && temp > lichtschwelle){
                    maxLight = temp;
                    turn = rotate;
                    PlayTone (1000, 100);
                 }
                 if ( rotate > circle ){ move = false; }
     }
     if(maxLight>0){                                    //müsste wert der allgemeinen raumumgebung sein
          RotateMotorEx(OUT_BC,60,rotate-turn,-100,true);
          search_light_sub();
          program = 2;
     }
     else {
          program = 5;
     }
     access=true;
}
```

FtT dreht sich ca. 360° um die eigene Achse und sondiert die Umgebung nach Helligkeit. Dabei merkt er sich die Position mit der höchsten Lichtintensität und speichert den dazugehörigen Motorrotationswert. Ist die Aufnahme der Messwerte abgeschlossen und es wurde eine Lichtquelle gefunden, richtig sich FtT selbstständig zu dieser aus. Nun wird die „search_light_sub“ Methode aufgerufen, die FtT noch einmal exakter auf die Lichtquelle ausrichtet.

```
// Sucht den Ort mit der grössten Lichtintensitaet im Sichtfeld von FtT und richtet ihn dahingehend aus

sub search_light_sub(){
     bool move=true;
     int maxLight_sub=0;
     int count = 0;
     int rotate = MotorRotationCount(OUT_B);
     int fullturn=500;
     int add=0;
     if(contact==false){fullturn=300; add = 40}
     int circle= rotate + fullturn;
     int temp = Sensor(IN_4);
     int a,b,c;
     a=rotate;
     RotateMotorEx(OUT_BC,65,300,100,true);
     while(move){
                 OnFwd(OUT_B,50);
                 OnRev(OUT_C,50);
                 Wait(150);
                 temp = Sensor(IN_4);
                 rotate = MotorRotationCount(OUT_B);                    //falls des nich geht rotate aufsummieren
                 if ( temp > maxLight_sub){
                    maxLight_sub = temp;
                    count = rotate;
                    PlayTone (1000, 100);
                 }
                 if ( rotate > circle ){ move = false; }
     }
     RotateMotorEx(OUT_BC,65,rotate-count+add,100,true);
}
```

Hierzu dreht sich FtT um 30° nach links, um dann in einer 60° Drehung seine Umgebung nach der höchsten Lichtintensität zu untersuchen. Anschließend richtet er sich zu dieser, exakter als in der „search_light“, aus.

Sollte der Fall eingetreten sein, dass es keine Position gibt, die eine gewisse Mindesthelligkeit (die des LED Balles bzw. der Lampe) aufweist, wechselt FtT in den sogenannten „Explorer Mode“.

```
// Falls kein Licht gefunden wurde, steuert FtT etwas wahllos durch die Gegend  um eine neue position für eine lichtbeschtimmtung zu erhalten
task explorer_mode(){
     int maxDistance=0;
     int temp=SensorUS(IN_1);
     int count;
     int rotate = MotorRotationCount(OUT_B)*sgn;
     int circle= rotate + 3000;
     bool move = true;
     while(move){
                 OnFwd(OUT_B,65*sgn);
                 OnRev(OUT_C,65*sgn);
                 Wait(150);
                 temp = SensorUS(IN_1);
                 rotate = MotorRotationCount(OUT_B)*sgn;                //falls des nich geht rotate aufsummieren
                 if ( temp > maxDistance && temp < 100){                //falls falsche ergebnisse wegen nicht reflektierender begrenzung max.
                    maxDistance = temp;                                 //zu messende distanz festlegen
                    count = rotate;
                    PlayTone (1000, 100);
                 }
                 if ( rotate > circle ){ move = false; }
     }
     RotateMotorEx(OUT_BC,65,rotate-count,100*sgn,true);                // wildes rumfahren klaeren
     RotateMotor(OUT_BC,90,40*(maxDistance-30)*sgn);

     program = 1;
     access = true;
}
```

Hierbei wird die Umgebung per Ultraschallsensor abgesucht und es wird die Richtung gespeichert in der der Ultraschallsensor die größte Entfernung gemessen hat, allerdings darf der Wert nicht über 100cm sein, da sonst die Fehlerrate aufgrund von falsch oder nicht reflektierenden Seitenflächen oder Hindernissen zu groß wäre. In die gewählte Richtung fährt Frank in der Länge des gemessenen Entfernungswertes(-20 [Größe von Frank]). Anschließend wird wieder zur search_light() gewechselt.

Wurde eine Lichtquelle gefunden, wechselt FtT in den „move_to_light“ Task.

```
// Bewegt FtT so lange in Richtung des Lichtest bis a) Die Lichtintensität nach laesst, oder b) die Entfernung zu gering wird.
task move_to_light(){       //
     PlayTone(700,100); Wait(100); PlayTone(50,100); Wait(100); PlayTone(300,100);  Wait(100); PlayTone(100,200); Wait(200);
     int sensSonic;
     int sensLight;
     string temp;
     int check;
     bool lamp=false;
     int distance=19;
     if(contact){
     SetSensorType (IN_3, IN_TYPE_REFLECTION);
     }
     else {
     lamp=true;
     distance = 22;
     }

     do
     {
           OnFwd(OUT_BC,60*sgn);
           Wait(150);
           sensLight = Sensor(IN_4);
           sensSonic = SensorUS(IN_1);
           check = Sensor(IN_3)
           if(contact){
                    if(check>600){
                       RotateMotor(OUT_A,80,-180);
                       contact=false;
                    }
           }
           if(lamp){
                    if(sensLight>950){
                       lamp=false;
                    }
           }
```

```
     while( (contact || lamp) && (sensSonic > distance) && ( (maxLight-50) < sensLight ))

     OnFwd(OUT_BC,0);
     if ( sensSonic < distance){
              program = 3;  // conquer_obstacle
     }
     if ( /*contact ||*/ sensLight < (maxLight-50)){
              program = 1;  // search_light
     }
     if (contact==false){
        program = job;
        if ( sensLight < (maxLight-50)){
              program = 1;  // search_light
           }
     }            // catch_the_ball   oder wenn shcon gefangen bei lampe stehenbleiben

     SetSensorType (IN_3, IN_TYPE_SWITCH);
     access=true;
}
```

Zu Beginn dieser wird ein weiterer Lichtsensor aktiv, der sich in der Front von FtT befindet, Blickrichtung schräg auf den Boden. Die einzige Aufgabe die dieser Sensor übernimmt, ist die Kontrollen ob sich der LED-Ball im Greifer befindet. Sollte dies der Fall sein, wird der Greifer sofort ausgelöst, und FtT wechselt wieder in den „search_light“ Modus um sein neues Ziel, die zweite Lichtquelle, zu finden. Lässt die Lichtintensität während der Fahrt nach, wird wieder in den „search_light“ Task gewechselt. Sollte sich ein Hindernis auftun welches ein direktes weiterfahren von FtT verhindert, wird der „conquer_object“ Task ausgeführt.

Sollte der Ball in der „move_to_light“ erfolgreich gefangen worden sein, so stellt die „caught_the_ball“ die bewegungsrelevanten Variable für FtT um (werden invertiert, da FtT nun rückwärts mit umgedrehtem Greifer weiter agiert).

Trifft FtT während der Fahrt auf ein Hindernis das es ihm unmöglich macht sein Ziel direkt anzusteuern, wechselst FtT in den „conquer_obstacle“ Task.

```
task conquer_obstacle(){
     PlayTone(100,100); Wait(100); PlayTone(300,100); Wait(100); PlayTone(500,100);  Wait(100); PlayTone(700,200); Wait(200);
     int maxDistance = SensorUS(IN_1);
     int actDistance = 0;
     int count       = 0;
     int i           = 1;
     int j           = -1;
     int temp        = 0;
     bool move = true;
     RotateMotor(OUT_BC,100,-500*sgn);
     RotateMotorEx(OUT_BC,65,700,100,true);


     int rotate = MotorRotationCount(OUT_B);
     int circle= rotate + 1400;
     while(move){
                 OnFwd(OUT_B,65);
                 OnRev(OUT_C,65);
                 Wait(150);
                 temp = SensorUS(IN_1);
                 rotate = MotorRotationCount(OUT_B);                    //falls des nich geht rotate aufsummieren
                 if ( temp > maxDistance){
                    maxDistance = temp;
                    count = rotate;
                    PlayTone (1000, 100);
                 }
                 if ( rotate > circle ){ move = false; }
     }
     RotateMotorEx(OUT_BC,65,rotate-count,100,true);
     RotateMotor(OUT_BC,100,(maxDistance/2)*sgn);

     program = 1;
     access=true;
}
```

Zu Beginn bewegt sich FtT erstmal ein gewisses Stück zurück, um sich ein wenig vom Hindernis zu distanzieren und sich ein wenig mehr Spielraum zu schaffen. Nun dreht er sich zuerst 90° nach links und misst die einzelnenn Entfernungen. Der größte Wert wird hierbei samt Position gespeichert. Anschließend wird dieselbe Prozedur mit einer 90° nach rechts Drehung vollführt. Sind beide Messungen abgeschlossen, dreht sic FtT automatisch in die Richtung, die die größte Entfernung ergab. Anschließend wird wieder nach der hellsten Lichtquelle mit Hilfe des „search_light“ Tasks gesucht.