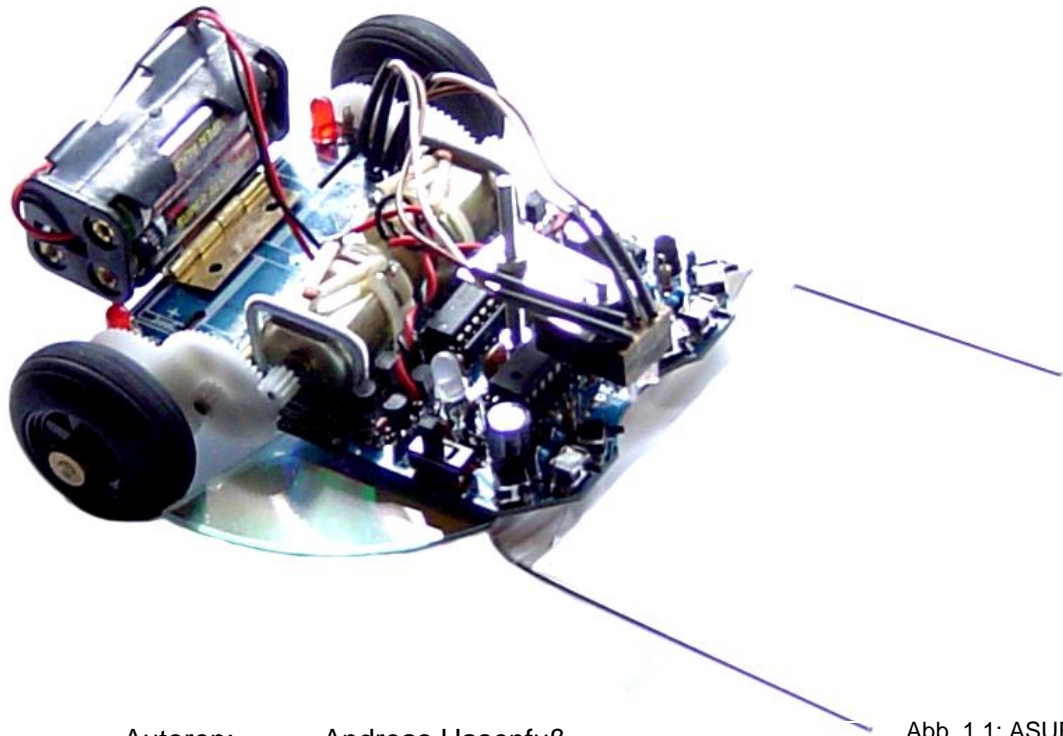


SIA 2007 - Wissenschaftliche Dokumentation



Autoren: Andreas Hasenfuß
 Paul Vorbach
 Dominik Walter

Abb. 1.1: ASURO

Datum: 20.06.2007 – 01.07.2007

Schule: Ganerben-Gymnasium Künzelsau

Inhaltsverzeichnis:

1	Einführung	5
2	Elektronikbauteile	5
2.1	Der Widerstand	5
2.2	Dioden	6
2.2.1	Die Diode	6
2.2.2	Die Zehnerdiode	6
2.2.3	Die Leuchtdiode	7
2.2.4	Die Zweifarbig Diode	7
2.3	Die Transistoren	7
2.3.1	Der Transistor	7
2.3.2	Der Fototransistor	8
2.4	Der Kondensator	9
2.5	Der Schwingquarz	9
2.6	Der Atmel Atmega8	10
3	Sensorik	12
3.1	Kollisionssensoren	12
3.2	Optische Sensoren	13
3.2.1	Linienverfolgung	13
3.2.2	Odometrie	14
3.2.3	Die Infraroteinheit	14
4	Grundlagen der C- und C++-Programmierung	15
4.1	Aufbau eines C-Programms	15
4.2	Variablen	16
4.3	Funktionen	17
4.4	Abfragen und Schleifen	18
4.4.1	if-Abfrage	18
4.4.2	while-Schleife	19
4.4.3	for-Schleife	19
4.4.4	switch-Abfrage	20

5	Projekt Abschlusspräsentation.....	22
5.1	Beschreibung des Programms.....	22
5.2	Modifizierung	23
5.2.1	Der „Greifer“	23
5.2.2	Der Abstandhalter	24
5.2.3	Der „Abweiser“	24
5.2.4	Der multifunktionale Schubladenknäuf	24
5.2.5	Befestigung der Motoren	24
5.2.6	Das Scharnier	25
5.2.7	Odometriesensor.....	25
5.2.8	Die Taschenlampe	25
5.3	Das Programm	26
5.3.1	Das Programm der ersten beiden ASUROs	26
5.3.2	Das Programm des dritten ASUROs	28
6	RFID – eine zukünftige Einsatzmöglichkeit.....	30
6.1	Funktion.....	30
6.2	Verwendung der RFID beim ASURO	30
7	Anhang.....	32
7.1	Quelltext-Legende	32
7.2	Quelltext	33
7.2.1	ASURO eins und zwei	33
7.2.2	ASURO drei	35
7.3	Schaltplan.....	39
8	Quellenverzeichnis	40
8.1	Literatur	40
8.2	Internet	40
8.3	Bildquellen.....	40


1 Einführung

ASURO ist die Abkürzung für „**A**nother **S**imple and **U**nique **R**obot from **O**berpfaffenhofen“, welcher am Deutschen Zentrum für Luft- und Raumfahrt entwickelt wurde. Dabei wurde versucht, die Kosten für ASURO so gering wie möglich zu halten, um einen Roboter für Einsteiger anzubieten.

ASURO zeichnet sich wie auch andere Roboter dadurch aus, dass er mit seinen Motoren und Sensoren verschiedene Aufgaben selbstständig erledigen kann.

2 Elektronikbauteile

2.1 Der Widerstand

Schaltzeichen: 

Wenn Elektronen durch ein Metallraumgitter wandern, stoßen sie mit dem zurück gebliebenen Atomrest oder anderen Elektronen zusammen. Je nach Raumgitter des Materials werden die Elektronen mehr oder weniger gebremst. Der elektrische Widerstand ist also die Kraft, die Elektronen abbremst.

Anschaulich formuliert bedeutet ein Widerstand eine Engstelle für den Stromdurchfluss, bei welchem die Elektronen behindert werden. Jedoch muss bei einem Widerstand berücksichtigt werden, dass die Maximalbelastbarkeit nicht überschritten wird, denn sonst könnte nicht nur der Widerstand, sondern auch weitere Bauteile beschädigt werden.



Abb. 2.1: Widerstände

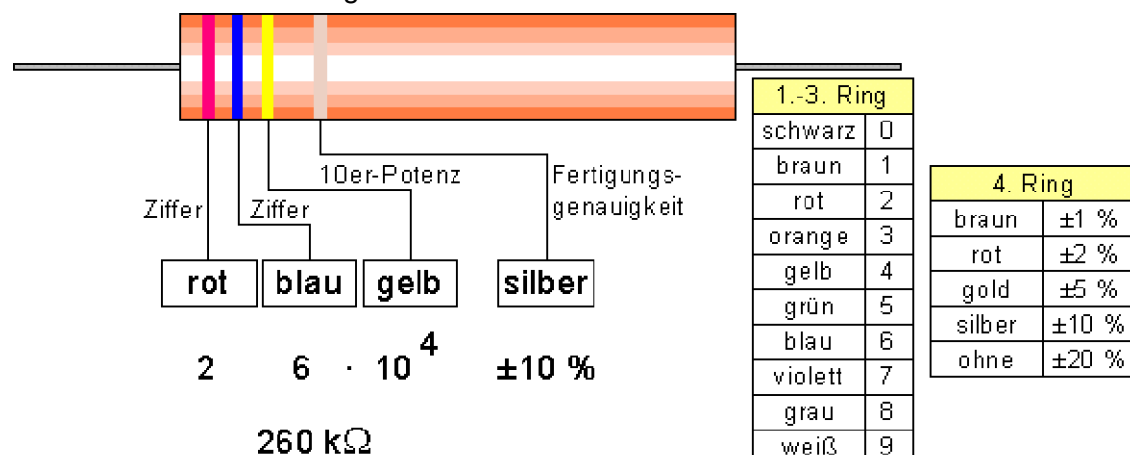
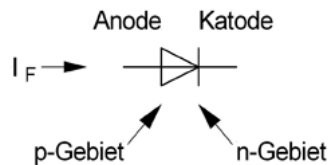


Abb. 2.2: Farbringe bei Widerständen

2.2 Dioden

2.2.1 Die Diode

Schaltzeichen:



Eine Diode besteht aus zwei Elektroden, der Anode und der Kathode. Sie lässt nur dann einen Stromfluss zu, wenn die Anode positiver als die Kathode ist. Legt man also zum Beispiel eine sinusförmige Wechselspannung mit wechselnder Polarität an, so lässt die Diode nur im positiven Bereich der Halbwellen Strom durch.

Eine Diode lässt also den Strom nur in eine Richtung durch, nämlich in die des Pfeils, also von Anode zur Kathode. Ist jedoch in Sperrrichtung eine zu hohe Spannung angelegt, so dass der Strom wieder fließt, dann wird die Diode normalerweise zerstört.

Eine Diode ist mit einem Ventil vergleichbar, welches sich in eine Richtung öffnet, in die Andere hingegen sperrt.



Abb. 2.3: Diode

2.2.2 Die Zehnerdiode

Schaltzeichen:

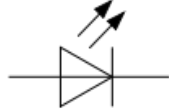


Die Zehnerdiode ist ein weiterer Diodentyp. Sie arbeitet in Durchlassrichtung wie eine normale Diode, jedoch wird sie in Sperrrichtung ab einem bestimmten Spannungswert niederohmig, dass heißt sie lässt den Strom durch.

Durch diese Eigenschaft wird die Zehnerdiode zur Stabilisierung von Spannungen eingesetzt.

2.2.3 Die Leuchtdiode

Schaltzeichen:



Die LED (**L**ight **E**mitting **D**iode) oder Leuchtdiode ist ein weiterer Diodentyp. Der Funktion und der Aufbau ist annähernd gleich wie bei einer normalen Diode, jedoch kann die Leuchtdiode Licht abstrahlen, wenn die sie in Durchlassrichtung betrieben wird.

Die vordere und die hinteren LEDs von ASURO werden verwendet, um konstante Lichtverhältnisse für die Fototransistoren zu schaffen, denn über die Odometriesensoren bei unterschiedlichen Lichtverhältnissen eine Strecke zu fahren, würde sonst überhaupt nicht gehen, und auch die Linienverfolgung würde misslingen. Andere hingegen werden als Betriebs- und Zustandsanzeige eingesetzt.

2.2.4 Die Zweifarbig Diode

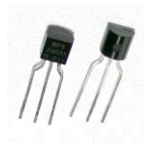
Sie funktioniert im Prinzip wie eine normale LED¹. Hier werden zwei LEDs in einem Gehäuse untergebracht. Je nach Wunsch Leuchtet die eine oder die andere LED. Will man nun eine dritte Farbe, so werden beide LEDs angesteuert und die Farben überlagern sich zu einer dritten Farbe.

Mit der Zweifarbigen Diode kann man also Platz sparend drei Farben erzeugen.²

2.3 Die Transistoren

2.3.1 Der Transistor

Schaltzeichen:



Ein Transistor ist ein Halbleiterbauelement, welches zum Schalten oder Verstärken von elektrischen Strömen eingesetzt wird. ASURO hat mehrere bipolare Transistoren, welche durch einen fließenden Strom angesteuert werden können.

Ein Transistor hat drei Anschlüsse, welchen zu verschieden dotierten Leitern führen. Beim n-p-n-

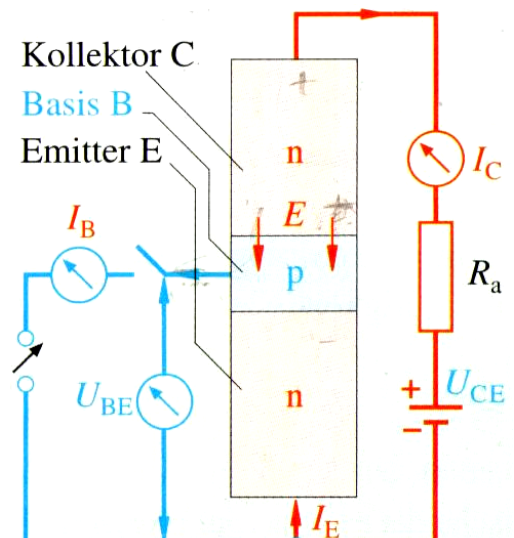


Abb. 2.4: Schema eines Transistors

¹ Siehe auch unter 2.2.3 Die Leuchtdiode

² siehe 7.3 Schaltplan

Transistor ist die untere n-Schicht der *Emitter*, die obere der *Kollektor* und die mittlere Schicht die *Basis*. Zwischen Emitter und Kollektor liegt die Spannung U_{CE} an, jedoch kann der Strom nicht fließen, da sich zwischen Kollektor und Basis eine Sperrschicht ausbildet (der Bereich Basis-Kollektor wirkt wie eine Diode, die in Sperrrichtung betrieben wird). Erhöht man nun die Spannung U_{BE} zwischen der Basis und dem Emitter auf etwa 0,7V, so fließt ein so genannter Steuerstrom. Dieser schwächt die Sperrschicht und Elektronen können durch die p-Schicht hindurchfließen, da die elektrischen Feldkräfte zwischen Emitter und Kollektor nun ausreichen, um die Elektronen durch die p-Schicht zu „ziehen“.

2.3.2 Der Fototransistor

Schaltzeichen:



Der Fototransistor ist mit einer kleinen Solarzelle vergleichbar. Er dient beim ASURO zum Unterscheiden von Helligkeiten, in unserem Fall *Schwarz und Weiß*. Hierbei muss man wissen, dass ein schwarzer Gegenstand Licht absorbiert, ein weißer Licht komplett reflektiert. Dieses reflektierte Licht ersetzt nun die an der Basis anliegende Spannung eines Transistors. Je mehr Licht also reflektiert wird, desto mehr Strom kann zwischen n-p-n Schicht fließen. Jedoch muss die oberste Schicht sehr dünn sein, da das Licht nicht in ihr aufgehalten werden soll, denn es muss bis zur Sperrschicht gelangen.

So kann der ASURO aufgrund der Fototransistoren zum Beispiel einer Linie folgen, oder am Abgrund stehen bleiben. Zudem kann über die Odometrie die Geschwindigkeit unseres Roboters gesteuert werden. Denn durch Zählen der Hell-Dunkel-Übergänge an den angebrachten Aufklebern, welche abwechselnd schwarze, und weiße Streifen besitzen, kann der Microcontroller bei entsprechender Programmierung die Geschwindigkeit errechnen.

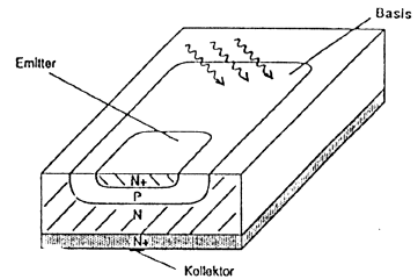


Abb. 2.5: Fototransistor

2.4 Der Kondensator

Schaltzeichen:



Der Kondensator ist ein Bauteil, das die Fähigkeit hat, elektrische Ladung, und die damit verbundene Energie zu speichern. Ein Kondensator besteht aus zwei leitenden Schichten, den Elektroden, welche eine nicht leitende Schicht, das Dielektrikum umgeben.

Schließt man einen Kondensator an eine Spannungsquelle an, so entsteht für kurze Zeit ein elektrischer Strom, welcher eine Elektrode negativ, die andere positiv lädt. Entfernt man nun die Spannungsquelle, so bleibt diese elektrische Ladung erhalten. Die Spannung eines Kondensators sinkt erst wieder, wenn man dem Kondensator Ladung entnimmt.

Ein Kondensator hat die Kapazität C , welche also Produkt von ϵ_0 (eine Konstante), ϵ_r (Permittivitätszahl) und dem Flächeninhalt A geteilt durch den Durchmesser d der Platten, hervorgeht.

$$C = \epsilon_0 \epsilon_r \cdot \frac{A}{d}$$

Wird ein Kondensator über einen Widerstand *ge-* bzw. *entladen*, so sieht der Spannungsverlauf $U_C(t)$ wie folgt aus:

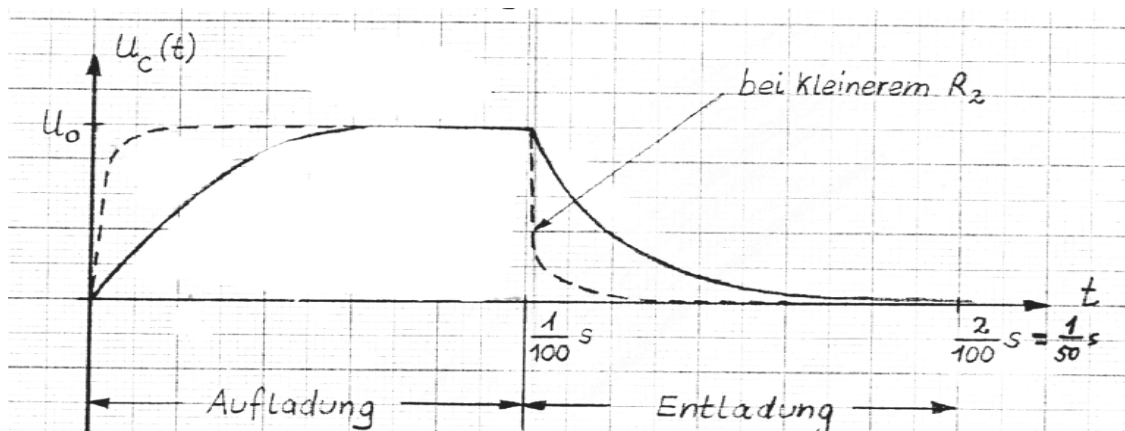
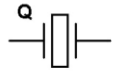


Abb. 2.6: Spannungsverlauf des Auf- und Entladevorgangs

Ein Kondensator wird benötigt, um eine wellige Gleichspannung zu glätten, um also konstante Bedingungen für die Bauteile zu schaffen. Des Weiteren werden die beiden Pole eines Motors über einen Kondensator verbunden, da beim Ausschalten des ASUROs in den im Motor verbauten Spulen durch Selbstinduktion Spannung entsteht, welche andere Bauteile beschädigen könnte.

2.5 Der Schwingquarz



Schaltzeichen:

Schwingquarze werden als Taktgeber für Sendeanlagen, Quarzuhren, oder im Fall des ASURO, im Prozessor eingesetzt, also überall, wo ein genauer Frequenzgang oder ein genauer Zeitgeber benötigt wird.

Ein Schwingquarz ist ein aus einem Quarzkristall herausgeschnittenes Plättchen, welches beidseitig leitend beschichtet und kontaktiert ist. Es wird frei (nur von den Anschlüssen gehalten) in einem luftdichten Gehäuse montiert. Beim Anlegen einer Wechselspannung wird das Plättchen nun zu Schwingungen angeregt und wirkt wie ein elektrischer Resonanzkreis.

Beim ASURO ist ein Quarz deshalb wichtig, da es als Taktgeber für den Microcontroller verwendet wird, denn dieser könnte ohne einen Quarz keine Informationen verarbeiten. Der Quarz von ASURO schwingt mit einer Frequenz von 8 MHz (8.000.000 Hz).

2.6 Der Atmel Atmega8

Der Atmel Atmega8 gehört zu den Mikrocontrollern. Ein Mikrocontroller ist eigentlich ein Mikroprozessor, der im Unterschied zu PC-Prozessoren auch einen Speicher und digitale oder analoge Ein- und Ausgänge auf einem Mikrochip vereinigt. Ein digitales Signal besteht nur aus zwei verschiedenen Spannungswerten, nämlich einem hohen und einem niedrigen, die in der Regel als 0 (niedrig) und 1 (hoch) ausgedrückt werden. Bei einem analogen Signal kann die Spannung jeden beliebigen Wert zwischen dem Minimum und dem Maximum annehmen, es ist also jede kleine Änderung wichtig. Mikrocontroller besitzen außerdem A/D-Wandler, mit denen analoge in digitale Signale umgewandelt werden. Das ist nötig, weil die Logikschaltungen des Mikroprozessors nur digitale Signale verarbeiten können.



Abb. 2.7: Atmel Atmega8

Mikrocontroller werden in erster Linie an der Bit-Zahl des verwendeten Datenbusses unterschieden. Ein Datenbus ist eine Folge von Nullen und Einsen, wobei jeweils eine Null oder eine Eins ein Bit(kleinste logische Einheit) ist. Diese Bit-Zahl gibt die Länge der Daten an, die der Mikrocontroller in einem Takt verarbeiten kann. Der Atmel Atmega8 ist ein 8-Bit-Mikrocontroller und kann demnach 8 Bit, also ein Byte, pro Takt verarbeiten. Das heißt der Atmel Atmega8 kann in einem Takt nur Zahlen von 0 bis 255 verarbeiten (z.B. Motorenansteuerung; für die Geschwindigkeit der Motoren sind beim Programmieren nur Werte von 0 bis 255 möglich). Der Mikroprozessor kann mit Hilfe von den in ihm verbauten Logikschaltungen Bits verarbeiten. Um aber – wie in unserem Fall – 8 Bits gleichzeitig zu verarbeiten, braucht der Mikroprozessor jede Logikschaltung auch achtmal. Sollen beim Atmel Atmega8 Zahlen, die größer als 255 sind, verarbeitet werden, so geschieht dies in mehreren Takten hintereinander, es dauert also länger.

Mikrocontroller brauchen einen Taktgeber, der die Arbeitsgeschwindigkeit vorgibt. Die Atmega Reihe von Atmel besitzt einen integrierten Taktgeber, der jedoch auf dem ASURO durch einen Externen ersetzt worden ist. Der Quarz auf dem ASURO schwingt mit einer Frequenz von 8 MHz, da aber der Atmega8 beide Flanken des Taktes nutzen kann (Flanken sind die steigenden und fallenden Abschnitte einer Schwingung, wenn man sie aufmalen würde), schafft er pro Schwingung zwei Takte, dass heißt er arbeitet mit einer tatsächlichen Taktfrequenz von 16 MHz.

Der Atmega8 ist das Herzstück des ASUROs, von hier aus werden die Sensoren, LEDs, Antriebe etc. angesteuert.

Außerdem geschieht das Verarbeiten der ausgelesenen

Werte ebenfalls durch den Mikrocontroller. Der Mikrocontroller stellt auch den nötigen Speicher für das Programm, das der ASURO ausführen soll, bereit und ebenso den Arbeitsspeicher, den der Mikroprozessor zum Abarbeiten der Befehle braucht.

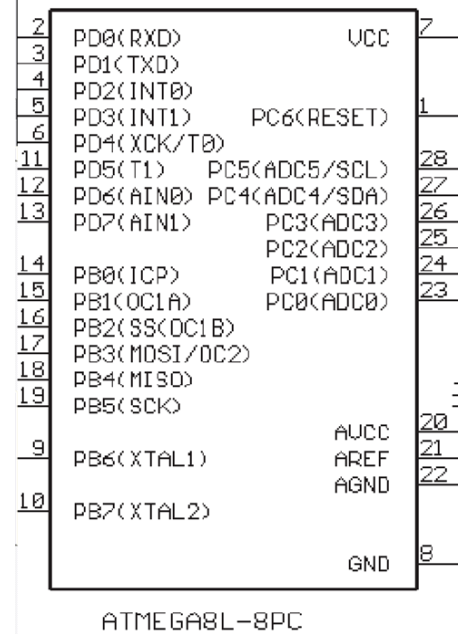


Abb. 2.8: Schaltplan vom Atmel Atmega8

3 Sensorik

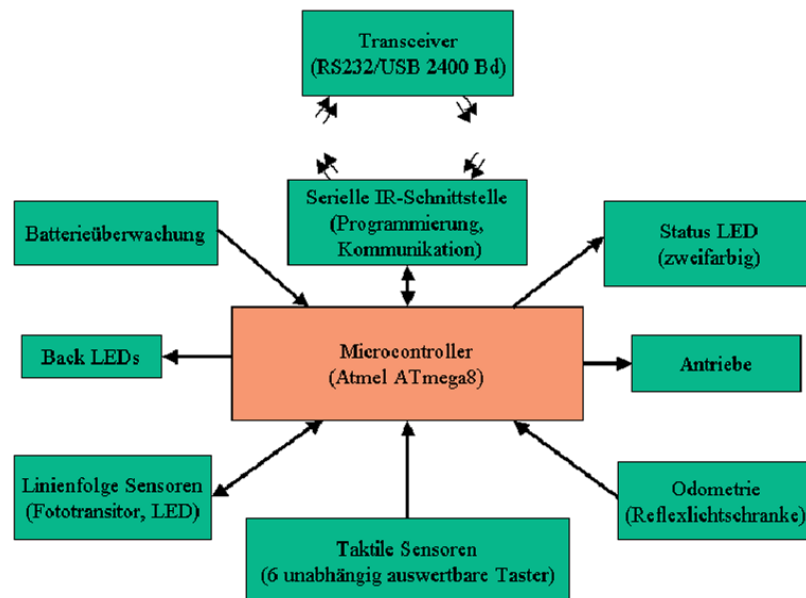


Abb. 3.1: Übersicht über Sensorik und LEDs

3.1 Kollisionssensoren

Wenn ein Kollisionssensor gedrückt wird, dann schließt er einen Stromkreis. Die Kollisionssensoren sind also Schalter, die, solange sie gedrückt sind, Strom leiten. Wenn sie nicht gedrückt sind, ist der Stromkreis unterbrochen. Damit am Mikrocontroller Pins gespart werden, teilen sich alle Kollisionssensoren 2 Pins. (Wäre jeder einzeln angeschlossen, wären alleine von den Kollisionssensoren schon 12 der 28 Pins belegt.) Damit die einzelnen Sensoren unterschieden werden können, wird jedem Sensor ein anderer Widerstand in Reihe geschaltet. Dem Mikrocontroller ist es daher möglich, anhand der verschiedenen Spannungswerte, zu ermitteln, welcher Sensor gerade gedrückt wird. Am Sensor K2 sind zwei Widerstände (mit jeweils 2 k Ω [Kiloohm]) in Reihe geschaltet, diese wirken jedoch wie ein Widerstand mit 4 k Ω . Im Mikrocontroller werden die verschiedenen Spannungswerte durch einen A/D-Wandler (Analog-Digital-Wandler) in ein digitales Signal umgewandelt, dabei nehmen die Sensoren folgende digitalen Werte an:



Abb. 3.2:
Taster

K1: 00100000 (32)
K2: 00010000 (16)
K3: 00001000 (8)
K4: 00000100 (4)
K5: 00000010 (2)
K6: 00000001 (1)

Im Quelltext können die Sensoren mit dem Befehl `PollSwitch()` abgefragt werden. Es gibt die Möglichkeit abzufragen, ob irgendein Sensor gedrückt ist: (`PollSwitch()`>0), oder die Möglichkeit gezielt einen einzelnen Sensor abzufragen: z. B. (`PollSwitch()==4`).

Vor der Klammer um „`PollSwitch()`“ muss im Quelltext noch ein Operator, z.B. eine Schleife oder eine if-Abfrage, stehen.

Die Kollisionssensoren teilen dem ASURO mit, dass er auf ein Hindernis getroffen ist. Durch die entsprechende Programmierung kann der ASURO dann das Hindernis umfahren, bei unserem Projekt beendet das Drücken der Sensoren K2 oder K4 die Linienverfolgung worauf dann die Erkennung des Gegenstandes folgt, auf den der ASURO aufgefahren ist.

3.2 Optische Sensoren

3.2.1 Linienverfolgung

Über die vorderen Fototransistoren kann ASURO einer Linie folgen. Eine in Mitten der beiden Fototransistoren angebrachte LED beleuchtet den Untergrund. Das ausgesendete Licht wird dann von den Fototransistoren wieder aufgenommen und ausgewertet.³ Bei einem dunklen Untergrund wird nun erheblich weniger Licht reflektiert, als bei einem Hellen. Entsprechend ändert sich der Spannungswert, welcher dann vom Prozessor ausgewertet wird.

Ein heller Untergrund ergibt große, ein schwarzer kleine Werte. Ist nun der Wert am linken Fototransistor kleiner als der des rechten, dann bedeutet dies, dass der ASURO von der Linie links abkommt, also muss dann die Motorgeschwindigkeit des rechten Motors erhöht werden und umgekehrt.

Eine weitere Fähigkeit, die man dem ASURO aufgrund der vorderen Fototransistoren beibringen kann, ist, dass er an einem Abgrund, also zum Beispiel einer Tischkante, anhält. Ein Abgrund bedeutet kleine Werte, da beide Fototransistoren schwarz erkennen. Somit kann ASURO gesagt werden, dass er anhalten soll, wenn beide Werte klein sind.⁴



Abb. 3.3: Linienverfolgung

³ Siehe auch unter 2.3.2 *Der Fototransistor*

⁴ Siehe auch unter 5.3 *Das Programm*

3.2.2 Odometrie

Die Odometrie funktioniert ähnlich, wie die Linienverfolgung. Anders als bei der Linienverfolgung ist jedoch, dass das Licht, welches vom einen Transistor ausgesendet wird, nicht sichtbar ist. Die Erkennung der Helligkeit erfolgt wieder über Fototransistoren. Leider ist die Odometrieinheit äußerst vom umgebungslicht abhängig.

3.2.3 Die Infraroteinheit

Über die Infraroteinheit kann ASURO mit dem PC kommunizieren. Mit ihr werden Programme auf den ASURO „geflasht“ und es können während des Programms über das Hyperterminal des PCs neue Informationen an den ASURO gesendet werden, aber auch Informationen vom ASURO empfangen werden. Wie der Name bereits vermuten lässt, werden die Daten durch infrarotes Licht, das für den Menschen unsichtbar ist, übertragen.

Die Infrarotkommunikation benötigt 2 Komponenten: Erstens den Transceiver und zweitens die Infraroteinheit des ASURO. Jede der Komponenten enthält eine Infrarotdiode (beim ASURO D10; beim Transceiver D5)⁵ zum Senden der Daten während des Programms und zum „Flashen“ des ASUROs. Der zweite Teil einer Infraroteinheit ist eine "Integrierte Schaltung", kurz IC. Diese Schaltung, die im Schaltplan mit IC2 bezeichnet ist, empfängt die von der Diode des Gegenübers gesendeten Daten und bereitet sie selbständig auf, bevor er sie weiter an den Mikrocontroller des ASURO, oder den IC1 des Transceivers schickt. Um die Daten zu übertragen müssen beide Infraroteinheiten Sichtkontakt zur jeweils anderen haben. Am besten stehen sie sich genau spiegelverkehrt gegenüber, die optimale Entfernung zur Übertragung von Daten beträgt dabei 5 bis 10 cm. Über die Infrarotschnittstelle ist es zum Beispiel auch möglich, den ASURO vom Hyperterminal aus, also vom PC, fernzusteuern.



Abb. 3.4: Infraroteinheit



Abb. 3.5: Infrarot-Transceiver

⁵ siehe 7.3 Schaltplan

4 Grundlagen der C- und C++-Programmierung

Dieser Abschnitt soll einen kurzen Einblick in die Programmierung mit C geben, welcher nötig ist, um das Programm später zu verstehen.

4.1 Aufbau eines C-Programms

Ein C-Programm setzt sich üblicherweise aus einem Header-Abschnitt, den deklarierten Funktionen und der Haupt-Funktion `main()` zusammen.

Im Header-Teil werden zusätzliche Programmbibliotheken in das Programm integriert, welche dann im eigentlichen Programm dann verwendet werden können. Zusätzlich können noch Konstanten deklariert werden, welche im gesamten Programm verwendet werden können und deren Wert sich niemals ändert. Alle Angaben im Header-Abschnitt werden hinter ein Raute-Symbol (#) gestellt, welches den Header-Abschnitt definiert.

Ein möglicher Header-Abschnitt kann z. B. so aussehen:

```
#include "asuro.h"
#define KONSTANTE 100
```

In diesem Beispiel würde zum einen die Programmbibliothek `asuro.h` eingebunden werden, sowie eine Konstante namens `KONSTANTE` definiert werden, welche den Wert `100` hat.

Im darauf folgenden Abschnitt können nun Variablen oder Funktionen dem gesamten Programm zur Verfügung gestellt werden. Wie diese im Einzelnen aussehen, wird später beschrieben.

Danach folgt der Abschnitt des Programms – die Funktion `main()` – welche später ausgeführt wird. In dieser Funktion wird der gesamte Ablauf des Programms beschrieben.

Die `main()`-Funktion sieht in der Regel so aus:

```
int main(void)
{
    Init();

    return 0;
}
```


4.2 Variablen

Die Verwendung von Variablen nimmt einen großen Teil in der Programmierung mit C ein. Variablen bestehen aus einem Variablentyp, dem Variablennamen und einem Wert.

Um eine Variable verwenden zu können, muss man sie zunächst einmal deklarieren. Dies kann in jedem Teil des Programms geschehen, allerdings ist die Variable nur für den Teil des Programms verfügbar, welcher ihrer Deklaration nachfolgt. Erfolgt die Deklaration einer Variable innerhalb der geschweiften Klammern einer Funktion, so ist diese auch nur für diese Funktion verfügbar.

Eine Variable wird nach folgendem Schema deklariert:

```
variablentyp Variablenname = Wert;
```

Die Übergabe eines Wertes bei der Deklaration ist optional. Standardmäßig sind die bereitgestellten Speicherplätze leer, in C und anderen Programmiersprachen hat dieser Zustand einen eigenen Namen – NULL.

Einige wichtige Variablentypen sind z. B.:

der BOOLEscher Wert (**bool** [boolean]),

welcher ausschließlich die Werte „wahr“ (**TRUE**) oder „falsch“ (**FALSE**) annehmen kann.

Ganzzahlen (**char** [character] / **int** [integer]),

welche je nach Typ verschieden hohe Werte annehmen können. **char** deckt z. B. die Werte von 0 bis 255 ab, während **int** hingegen alle Werte zwischen 0 und 4.294.967.295 abdeckt. Zusätzlich lässt sich noch definieren, ob eine Ganzzahlvariable ein Vorzeichen besitzen kann. Dies geschieht über den vorausgehenden Zusatz **signed** bzw. **unsigned**. **unsigned** halbiert den Wertebereich einer Ganzzahlvariable in positiver Richtung und ermöglicht Werte im gleichen negativen Bereich. Somit hat **unsigned char** beispielsweise einen Wertebereich von -128 bis 127.

Gleitkommazahlen (**float**),

welche Dezimalzahlen von $\pm 1,5 \cdot 10^{-45}$ bis $\pm 3,4 \cdot 10^{38}$ bis auf sieben Stellen genau speichern können.

Nachdem eine Variable deklariert wurde, kann mit dem Befehl

```
Variablenname = Wert;
```

der Variable ein neuer Wert zugewiesen werden.

Beispiel:

```
unsigned int Anzahl = 0;  
Anzahl = 25;
```


Hier wird eine Variable Anzahl1 bereitgestellt, die am Anfang den Wert 0 besitzt. Danach wird ihr der Wert 25 zugewiesen. Schreibt man im folgenden Programmabschnitt in einem Rechenausdruck oder ähnlichem nun das Wort Anzahl1, ohne dass vorher ein neuer Wert zugewiesen wurde, so wird es durch seinen Wert (25) ersetzt.

4.3 Funktionen

In der Programmiersprache C gibt es die Möglichkeit, Funktionen aufzustellen oder zu benutzen. Eine Funktion wird – wie auch in der Mathematik – über ihren Namen und einem zugewiesenen Argument aufgerufen. Mehrere Argumente werden durch Kommata getrennt.

Der Befehl

```
Funktion(Argument1, Argument2);
```

führt die Funktion aus und liefert ein Ergebnis zurück.

So könnte z. B. der Funktionsaufruf

```
Multiplizieren(4, 6);
```

als Ergebnis 24 zurückliefern, wenn die Funktion vorher entsprechend deklariert wurde.

Funktionen werden im Allgemeinen wie folgt deklariert:

```
ausgabetyp Funktionsname(typ Argument1, typ Argument2)
{
    // Ablauf innerhalb der Funktion
}
```

Der Abschnitt innerhalb der geschweiften Klammern wird beim Aufruf der Funktion abgehandelt.

Damit die Funktion Multiplizieren() funktionieren kann, wird vorher deklariert, was sie macht:

```

int Multiplizieren(int Faktor1, int Faktor2)
{
    int Ergebnis;      //Bereitstellen einer Variable „Ergebnis“

    Ergebnis = Faktor1 * Faktor2;    //Multiplizieren der Faktoren

    return Ergebnis;    //Ausgabe von „Ergebnis“
}

```

4.4 Abfragen und Schleifen

4.4.1 if-Abfrage

Eine if-Abfrage dient dem gezielten Ausführen eines Programmabschnitts anhand einer oder mehrerer Bedingungen.

Ihr Aufbau erinnert an den einer Funktion:

```

if(Bedingung)
{
    // Ablauf innerhalb der if-Abfrage
}
else if(Bedingung2)
{
    // Ablauf innerhalb des else if-Teils
}
else
{
    // Ablauf innerhalb des else-Teils
}

```

Bedingung können z. B. Vergleiche oder aber auch Funktionsaufrufe sein.

Der Ablauf sollte an folgendem Beispiel klar werden:

```

if(Multiplizieren(Faktor1, Faktor2) > 0)
{
    printf('Das Produkt ist größer als null.');
```

```

}
else if(Multiplizieren(Faktor1, Faktor2) < 0)
{
    printf('Das Produkt ist kleiner als null.');
```

```

}
else
{
    printf('Das Produkt ist null.');
```

```

}
```

Zunächst wird abgefragt, ob das Ergebnis der Multiplikation größer als null ist. Wenn dies zutrifft, wird mit der Funktion `printf()`, die in der standardmäßigen C-Bibliothek vorhanden ist, die Zeichenkette 'Das Produkt ist größer als null.' Ausgegeben. Der restliche Abfragenteil wird übersprungen.

Wenn dies aber nicht zutrifft, wird zunächst nach demselben Schema abgefragt, ob das Ergebnis kleiner als null ist und eine entsprechende Zeichenkette ausgegeben.

Ist dies aber wieder nicht der Fall, so wird die Zeichenkette 'Das Produkt ist null.' ausgegeben und die if-Abfrage ist abgeschlossen.

4.4.2 while-Schleife

Die while-Schleife stellt ebenfalls wie die if-Abfrage Bedingungen:

```

while(Bedingung)
{
    // Ablauf innerhalb der while-Schleife
}
```

Wie in der if-Abfrage wird die Bedingung überprüft; der Abschnitt innerhalb der while-Schleife wird solange wiederholt, bis die Bedingung nicht mehr erfüllt ist.

4.4.3 for-Schleife

Die for-Schleife dient in der Regel dazu, einen Vorgang eine bestimmte Anzahl von Malen durchzuführen. Sie kann allerdings auch zu viel mehr verwendet werden, was in dieser Dokumentation nicht genauer ausgeführt wird.

Die for-Schleife ist nach dem immer gleichen Schema aufgebaut:

```

for(Zählvariable = Anfangswert; Bedingung; Funktion)
{
    // Ablauf innerhalb der for-Schleife
}

```

Zunächst wird eine Zählvariable gesetzt, welche eine bestimmte Bedingung erfüllen muss. Die angegebene Funktion wird nun solange aufgeführt, bis die Bedingung erfüllt ist. Nach jeder Überprüfung der Bedingung wird der gesamte Ablauf, der innerhalb der for-Schleife beschrieben ist, ausgeführt.

Dies kann z. B. so aussehen:

```

int i;
for(i=0; i<=20; i++)
{
    printf(i.". Durchgang\n\r");
}

```

Mit der Funktion `i++` wird so lange `1` zur Variable `i` addiert, bis sie den Wert `20` hat. Entsprechend oft wird der Ablauf innerhalb der for-Schleife ausgeführt. Es entsteht also eine durchnummerierte Liste mit zwanzig Zeilen.

4.4.4 switch-Abfrage

Da in unserem Programm auch eine switch-Abfrage auftaucht, wird diese hier kurz erklärt, obwohl sie von keiner so großen Bedeutung ist und auch durch mehrere if-Abfragen ersetzt werden kann.

Die switch-Abfrage ist anders als die anderen Abfragen und Schleifen ausgebaut:

```
switch(Variable){  
    case Fall1:  
        // Ablauf im ersten Fall  
        break;  
    case Fall2:  
        // Ablauf im zweiten Fall  
        break;  
    .  
    .  
    .  
}
```

Die switch-Abfrage überprüft, ob die Variable mit einem der angegebenen Fälle übereinstimmt und führt die entsprechenden Programmabschnitte durch, die zwischen `case:` und `break;` stehen.

Mit dieser kurzen Einführung in die Programmiersprache C sollte es nun einfacher sein, die Programme unserer ASUROS zu verstehen.

5 Projekt Abschlusspräsentation

5.1 Beschreibung des Programms

In unserem Programm sollen von einem ASURO zwei verschiedenfarbige Gegenstände erkannt werden und anhand der Farbe auf zwei Zwischenlagerplätze verteilt werden. Dort sollen die Gegenstände von zwei weiteren ASUROs abgeholt und zu den Endlagerplätzen, zwei schwarzen Stofftaschen, gebracht werden.

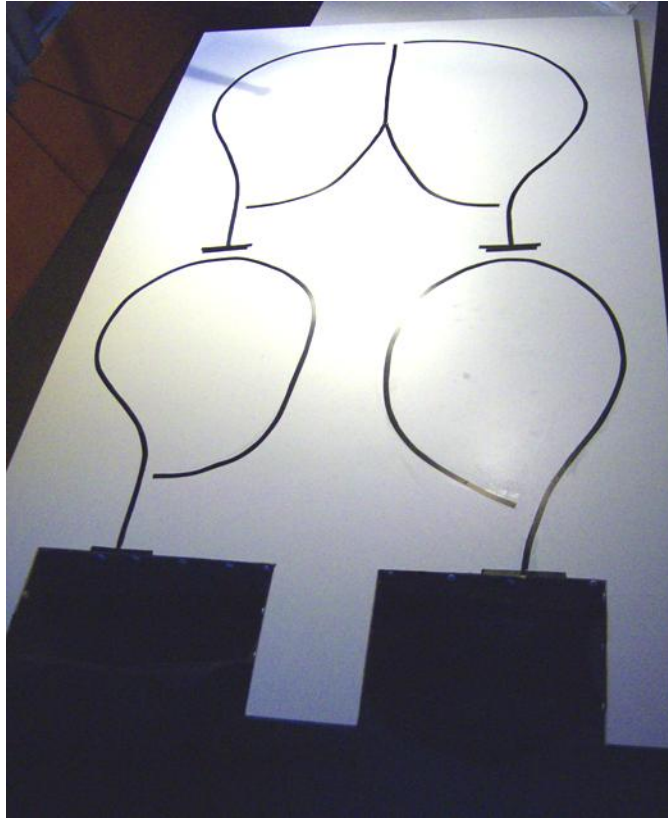


Abb. 5.1: Platte mit Parcours

5.2 Modifizierung

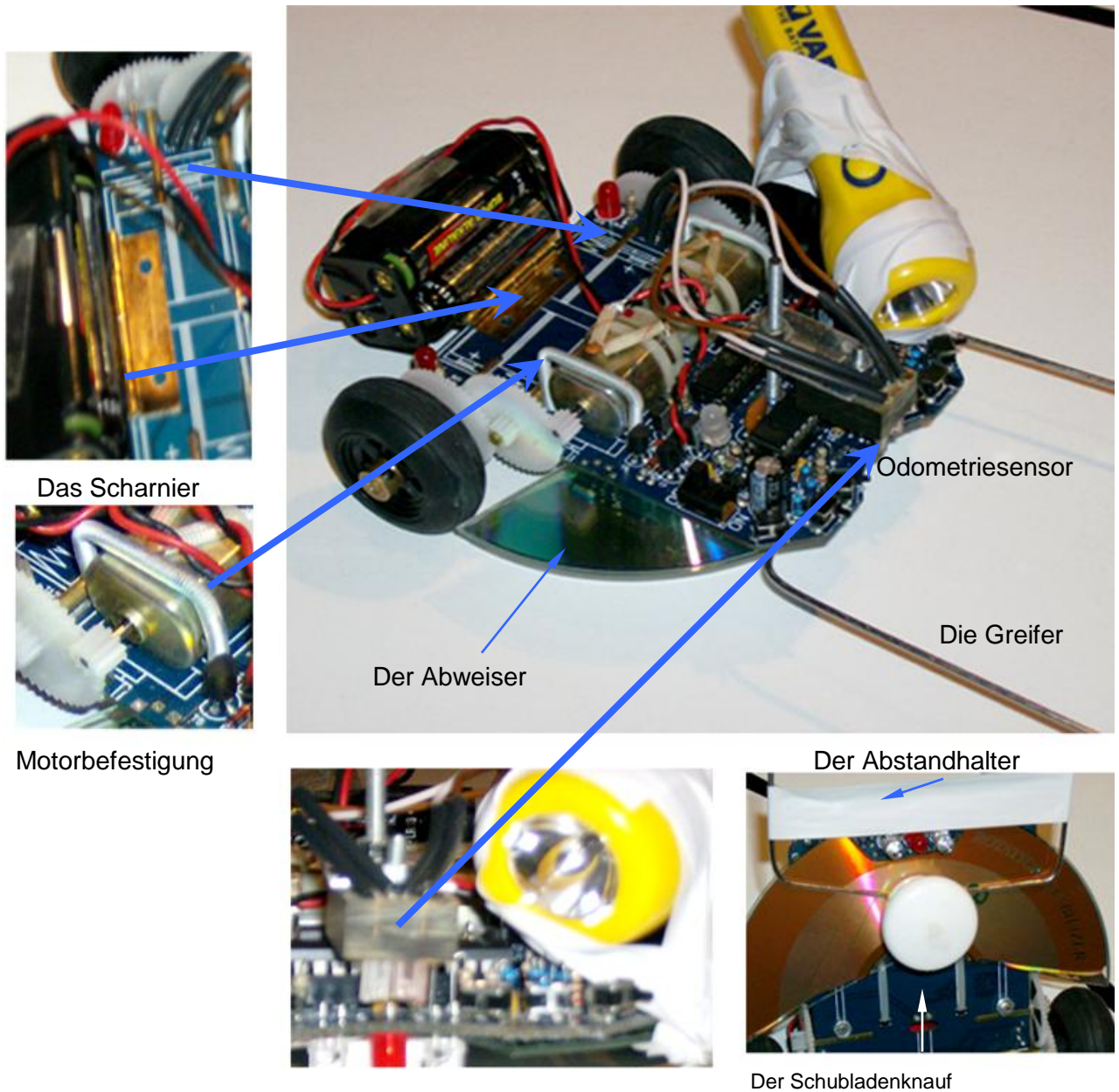


Abb. 5.2: Modifizierungen

5.2.1 Der „Greifer“

Um die beiden Gegenstände von einem Ort zum anderen zu transportieren, benötigen wir eine spezielle Vorrichtung, welche ein seitliches Abrutschen des Gegenstandes vor dem ASURO verhindert. Dazu verwendeten wir einen gebogenen Draht, der an der Unterseite des ASURO angebracht wurde. Da die beiden ASUROs, welche den Gegenstand vom Zwischenlagerplatz zur Tasche transportieren, den Gegenstand in einer Kurve aufnehmen

müssen, darf der Draht an der Innenseite der Kurve nicht zu lang sein, weil sonst der Gegenstand an der Außenseite des Greifers abgleiten würde.

5.2.2 Der Abstandhalter

Da ASURO jedoch dem schwarzen Gegenstand gefolgt ist, da er diesen als schwarze Linie registrierte, und somit vom eigentlichen Weg abkam, mussten wir einen Abstandhalter anbringen, welcher sicherstellte, dass der aufgenommene Gegenstand weit genug von den Fototransistoren weg ist. Jedoch mussten wir bedenken, dass wir mit unserem Abstandhalter die Taster nicht außer Gefecht setzten, denn sonst hätte ASURO keinen Gegenstand mehr erkennen können, da er diese über die Taster registriert. So lag die Kunst darin, den Abstandhalter so anzubringen, dass er weder die Taster, noch die vorderen Fototransistoren behindert.

Außerdem war der Abstandhalter nötig, um den Einfluss des vom Gegenstand geworfenen Schattens auf die Fototransistoren zu minimieren. Der Schatten könnte nämlich auf nur einen der beiden Transistoren fallen und so die Linienverfolgung behindern, oder sogar unmöglich machen.

5.2.3 Der „Abweiser“

Da der ASURO mit seinen Rädern an Gegenständen hängen bleiben kann, wollten wir einen Abweiser montieren, an dem herumliegende Gegenstände abgleiten können, anstatt am Rad zu verkanten. Wir wählten als Abweiser eine CD, weil sie einerseits den nahezu idealen Durchmesser hat, andererseits aber auch die eventuelle Überbrückung der Lötkontakte durch den Greifer verhindert. Durch die verspiegelte Oberseite wertet die CD den ASURO außerdem optisch auf.

5.2.4 Der multifunktionale Schubladenknauf

Zur Befestigung des Abweisers und des Greifers mussten wir etwas finden, das außerdem noch an der Unterseite leicht gebogen war und ähnliche Gleiteigenschaften wie der vorgesehene Tischtennisball aufwies. Des Weiteren durfte die Befestigung nicht zu hoch sein, da sonst die Taster an der Vorderseite nicht mehr ideal gedrückt werden können. Diese Eigenschaften vereinigte ein Schubladenknauf, der durch eine Schraube an der Platine des ASUROs befestigt wurde. Die Schraube konnte nach geringfügigem Aufbohren des bereits vorhandenen Loches zwischen IC und Mikrocontroller dort angebracht werden.

5.2.5 Befestigung der Motoren

Um eine gleich bleibende Kraftübertragung von den Motoren auf die Räder zu gewährleisten, ist es notwendig, die Motoren entsprechend zu fixieren.

Die Kabelbinder ermöglichten keinen ausreichenden Halt auf der Platine, weshalb wir den Kabelbinder auf der Außenseite durch einen metallischen Gewindestab ersetzten. Dazu haben wir den Gewindestab gebogen und an den Löchern, die für die Kabelbinder vorgesehen waren, mit Muttern befestigt.

5.2.6 Das Scharnier

Der Batteriewechsel am ASURO wird durch die Befestigung des Batteriegehäuses durch einen Kabelbinder wesentlich erschwert. Deshalb haben wir die Kabelbinder durch ein Scharnier ersetzt, das wir mit Sekundenkleber an dem Batteriegehäuse sowie der Platine festklebten. Neben der verbesserten Funktionalität wirkt das Scharnier außerdem optisch aufwertend.

5.2.7 Odometriesensor

Für die Erkennung der Gegenstände war ein weiterer optischer Sensor an der Vorderseite des ASUROs notwendig. Nachdem die beiden Odometriesensoren beim Auslöten zerstört wurden, mussten wir aus dem Conrad-Katalog neue bestellen. Glücklicherweise kamen die neuen Sensoren sehr schnell, sodass wir mit dem Umbau früh beginnen konnten.

Zuerst wurden die Sensoren links hinten durch 10 cm lange, dünne Kabel ersetzt, welche durch Schrumpfschläuche isoliert wurden. Danach haben wir aus einem Stück Holz eine Vorrichtung gebaut, in welcher die Sensoren nun Platz fanden. Nach dem Anbringen unserer Vorrichtung an den ASURO, haben wir die vorher angelöteten Kabel an den nun vorne sitzenden Odometriesensor gelötet und wieder isoliert.

Leider war ein Metallpin auf der Platine so beschädigt, dass der Kontakt eines Kabels nicht mehr auf der unteren Platine hergestellt werden konnte. So blieben uns nun zwei Möglichkeiten:

- Wir löten das Kabel auf den nächsten Metallpin, der auf der entsprechenden Leiterbahn sitzt.
- Wir befestigen alle Kabel auf der anderen Seite, da sich dort ein weiterer Odometriesensor befindet.

Wir haben uns für die zweite Möglichkeit entschieden, da uns das Risiko, einen kleinen Fehler zu begehen, bei der ersten Möglichkeit als zu hoch erschien. So haben wir die Kabel umgelötet und der nun vorne sitzende Odometriesensor funktioniert.

5.2.8 Die Taschenlampe

Da die Odometriesensoren eine gewisse Lichtstärke benötigen, um korrekt zu funktionieren, war die Anbringung einer zusätzlichen Lichtquelle, einer Taschenlampe, notwendig. Sie macht den Odometriesensor nahezu unabhängig von Lichtmangel. Angebracht wurde die Taschenlampe durch Festkleben mit Klebeband.

5.3 Das Programm

Technisch erfüllt der ASURO nun alle Voraussetzungen, die er für die Erkennung und den Transport der verschiedenen Gegenstände benötigt. Damit unsere ASUROs aber machen, was wir mit ihnen vorhaben, mussten wir sie noch entsprechend programmieren, was sich nicht als einfach erweisen sollte. Das Programm für die ersten beiden ASUROs, welche den Gegenstand nur vom Zwischenlagerplatz zu den Taschen befördern sollten, um danach nach einer Kurve

rückwärts den Vorgang zu wiederholen. Damit der ASURO den Gegenstand transportieren kann, benötigen wir ein Programm, mit dem der Roboter Linien verfolgen kann. Dies soll er so lange machen, bis er an einen Abgrund kommt – also die beiden Fototransistoren „schwarz sehen“. Wenn dies der Fall ist, soll der ASURO zurück fahren und gleichzeitig eine Kurve nach links oder rechts – je nach der Seite, auf der er unterwegs ist – fahren. Danach erkennen die beiden Fototransistoren auf der Unterseite des ASURO keinen Abgrund mehr und sollen wieder der Linie folgen, wie sie es vorher schon getan haben.

Ist ein Abgrund vorhanden?			
Ja	Nein		
Halte an.	Links heller als rechts?	Abgrund vorhanden?	Andernfalls:
Führe eine Kurve rückwärts durch.	Führe eine Rechtskurve durch	Halte an. Setze 'abgrund' auf 'wahr'.	Führe eine Linkskurve durch.

Abb. 5.3: Struktogramm

5.3.1 Das Programm der ersten beiden ASUROs ⁶

Das Programm der ersten beiden ASUROs setzt sich aus zwei ineinander verschachtelten if-Abfragen zusammen, die in einer while-Schleife endlos wiederholt werden. In einer untergeordneten Schleife wird zusätzlich abgefragt, ob der ASURO sich links oder rechts neben der Linie befindet oder ob er über einem Abgrund steht.

5.3.1.1 Header-Abschnitt

Das Programm beginnt mit dem Einbinden der ASURO-Bibliothek über den Befehl `#include "asuro.h"`. Um dem ASURO ein Halten an der Haltelinie und am Abgrund zu ermöglichen, wird noch ein Haltewert mit `#define HALT 80` definiert. Anhand von diesem Wert kann später

⁶ siehe im Anhang 7.2.1 ASURO eins und zwei

überprüft werden, ob sich die Fototransistoren gerade über einem Abgrund oder über der Haltelinie befinden.

5.3.1.2 Das Hauptprogramm

Nun folgt die `main()`-Funktion in der zunächst einmal über den `Init()`-Befehl die Bauteile auf dem ASURO angesprochen werden. Für die Ermittlung der Werte an den Fototransistoren wird nun eine Variable als Array deklariert, d. h. es werden in dieser Variable zwei Speicherplätze bereitgestellt, welche in den eckigen Klammern einzeln angesprochen werden können. Außerdem wird eine zusätzliche Variable `abgrund` deklariert, welche zunächst einmal `FALSE` ist. Außerdem wird die FrontLED durch den Funktionsaufruf `FrontLED(ON);` eingeschaltet.

In der darauf folgenden while-Schleife, welche eine Endlosschleife darstellt, da die Bedingung **1** immer wahr ist, wird die Richtung des Motors auf FWD, also „vorwärts“ gesetzt. In einer if-Abfrage innerhalb dieser Endlosschleife wird überprüft, ob die Variable `abgrund` gerade `FALSE` ist. Ist dies der Fall, sollen die Fototransistoren über die Funktion `LineData(Ftrans);` die aktuellen Helligkeitswerte ermitteln und der Variable `Ftrans` übergeben. Mit dem sich daraus ergebenden Array kann man nun arbeiten.

In einer weiteren if-Abfrage, die sich immer noch im if-Teil der äußeren if-Abfrage befindet, wird überprüft, ob der Wert des rechten Fototransistors größer ist, als der des linken Fototransistors. Ist dies so, führt der ASURO eine Kurve nach links durch und die Variable `abgrund` bleibt auf `FALSE`.

Im else-Teil wird eine einfache Rechtskurve durchgeführt, die ausgeführt wird, wenn der rechte Fototransistor weder einen höheren Wert als der linke hat, noch einen Abgrund erkennt.

Diese beiden Vorgänge bewirken, dass der ASURO einer schwarzen Linie folgt.

Falls beide Fototransistoren einen kleineren Wert erhalten soll der ASURO abrupt anhalten. Dies wird mittels `MotorDir(BREAK, BREAK);` erreicht. Dieser besondere Befehl sorgt für ein Kurzschließen beider Motoren, wodurch ein sofortiges Anhalten ermöglicht wird. Zusätzlich wird `abgrund` auf `TRUE` gesetzt, damit nun der else-Teil der äußeren if-Abfrage ausgeführt werden kann. Somit hält der ASURO an einem Abgrund oder einer über einer schwarzen Linie, die quer zu seiner Bewegungsrichtung verläuft, an.

Ist nun `abgrund TRUE`, so wird beim linken ASURO eine Kurve rückwärts und nach rechts durchgeführt, beim rechten ASURO nach links. Mit dem Funktionsaufruf `Msleep(1200);` aus der Bibliothek wird dafür gesorgt, dass die Kurve 1200 ms [Millisekunden] andauert, bevor der ASURO dann 100 ms geradeaus fährt, um danach die Variable `abgrund` wieder auf `FALSE` zu setzen.

Nachdem dies geschehen ist, wird wieder der if-Teil ausgeführt, d. h. der ASURO folgt wieder der Linie.

5.3.2 Das Programm des dritten ASUROS⁷

Im Programm des ASURO, welcher die Gegenstände trennen soll, müssen im Grunde wieder die selben Vorgänge ablaufen, mit dem Unterschied, dass der ASURO nachdem die Taster gedrückt wurden und er also vor einem Gegenstand steht, die Odometriesensoren ansprechen muss, um zu ermitteln, ob er sich gerade vor einem schwarzen oder weißen Gegenstand befindet.

5.3.2.1 Header-Abschnitt

Zu Beginn des Programms wird wieder auf die ASURO-Bibliothek `asuro.h` bezogen, ein Haltezeitwert und zusätzlich noch ein Wert für die Odometrie gesetzt. Anhand dieser Konstante kann man später einen schwarzen von einem weißen Gegenstand unterscheiden.

Zur besseren Übersicht werden `0` und `1` durch `LINKS` und `RECHTS` ersetzt.

5.3.2.2 Eigene Funktionen

Vor dem eigentlichen Programm werden nun noch zwei Funktionen für zwei unterschiedlich lange Kurven deklariert. Anhand zweier übergebener Argumente `LinksRechts` und `VorZurueck` wird bestimmt, in welcher Richtung die Kurve durchgeführt werden soll.

Eine der beiden Kurven soll 0,800, die andere 0,620 Sekunden lang dauern.

5.3.2.3 Das Hauptprogramm

Im Hauptprogramm werden zuerst wieder die Bauelemente auf der ASURO-Platine angesprochen und dann ein paar Variablen bereitgestellt. Dies sind wieder das Array `Ftrans[2]` und nun zusätzlich noch `Otrans[2]`, welches zum Zwischenspeichern der Werte der Odometrietransistoren dient. Außerdem werden die Variablen `Richtung`, welche `LINKS` ist, und `Status`, welcher `0` ist, bereitgestellt. Die Variable `i` wird für die for-Schleife im Programm benötigt.

Im Programm wird nun eine switch-Abfrage innerhalb einer while-Schleife endlos wiederholt.

Da `Status` zunächst `0` ist, wird der Abschnitt nach `case 0`: durchgeführt. Hier findet wieder eine gewöhnliche Linienverfolgung statt, wie wir sie schon aus dem ersten Programm kennen. Zusätzlich wird aber noch überprüft, ob Taster K2 oder K5 gedrückt wird. Wenn dies der Fall ist, wird die Variable `Status` auf `1` gesetzt und der Abschnitt nach `case 1`: wird ausgeführt.

Hier wird mittels for-Schleife zehnmal abgefragt, ob der Odometrietransistor einen niedrigeren oder einen höheren Wert als `ODOMETRIE` erkennt und entsprechend wird die Variable `Richtung` bei einem niedrigen, also hellen, Wert auf `LINKS`, bei einem hohen auf `RECHTS` gesetzt. Die BackLEDs werden ebenfalls je nach Richtung an bzw. ausgeschaltet.

Nach dieser Abfrage soll der ASURO über die `Msleep()`-Funktion eine Sekunde lang warten und danach `Status` auf `2` stellen.

⁷ siehe im Anhang 7.2.2 ASURO drei

Im case 2: wird über den Funktionsaufruf Kurve(Richtung, FWD); eine Vorwärtskurve in der vorher bestimmten Richtung vollzogen und danach die Motorenrichtung bei beiden wieder auf FWD gestellt. Nun wird Status auf 3 gestellt.

In case 3: folgt nun wieder eine Linienverfolgung, ohne die Abfrage der Taster, sondern mit einer Abfrage nach einem Abgrund. In diesem Fall soll der ASURO wieder abrupt anhalten und dann auf Status = 4 wechseln.

In diesem Abschnitt fährt der ASURO für 0,55 Sekunden rückwärts gerade aus, um danach wieder eine Vorwärtskurve in entsprechender Richtung durchzuführen und danach wieder auf den ersten Status zu wechseln, damit er wieder der Linie folgen kann.

6 RFID – eine zukünftige Einsatzmöglichkeit

Transponder ist ein zusammengesetzter Begriff aus *Transmitter* und *Responder*. Er bildet das Herzstück des RFID -Systems. Der Transponder muss an dem zu identifizierenden Objekt angebracht werden. Das können Container sein, aber auch CD-Hüllen oder Getränkedosen. Der Transponder kann unterschiedliche Formen und Größen haben, an Containern ist er normalerweise etwa so groß wie eine Zigarettenschachtel, an CD-Hüllen ist er jedoch kaum von einem Etikett zu unterscheiden. Im Allgemeinen gilt, je größer ein Transponder, desto leistungsfähiger ist er. Ein Transponder besteht aus einer Antenne, die gleichzeitig eine Spule zur Induktion von Spannung sein kann, und einem beschreibbaren Mikrochip, der Daten speichern und auch wieder über die Antenne an ein Lesegerät senden kann.

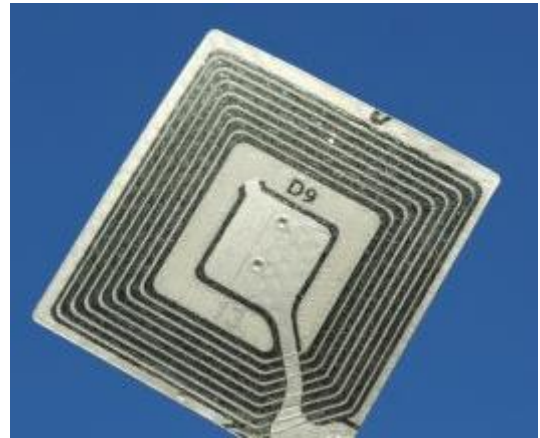


Abb. 6.1: RFID-Chip

6.1 Funktion

Das Lesegerät erzeugt ein elektromagnetisches Feld. Wird nun der Transponder in das Feld hinein bewegt so wird in der Antenne eine Spannung induziert, mit der ein Kondensator aufgeladen wird, der den Mikrochip dann mit Strom versorgt. Der Transponder sendet seine Antwort, in dem er das elektromagnetische Feld leicht verändert (mit Hilfe von Funkwellen), was vom Lesegerät registriert wird und entsprechend ausgewertet wird. Dieser Ablauf geschieht in der Regel in Bruchteilen von Sekunden und kann theoretisch über eine Entfernung von bis zu einem Kilometer stattfinden. Diese Entfernung wird jedoch nur von sehr leistungsstarken, teuren Systemen unter Idealbedingungen, dass heißt ohne Störfaktoren, wie das Vorhandensein von Metall im Feld, erreicht. Diese großen Entfernungen setzen aber auch eine eigene Stromversorgung des Transponders und ein sehr starkes elektromagnetisches Feld voraus. Meist bleibt die Übertragungsentfernung, aus Kostengründen und Verwendungszweck des Systems, bei wenigen Metern. Bei solchen geringen Leistungsanforderungen kann der Transponder, wie oben genannt die notwendige Betriebsenergie durch Induktion aus dem Feld entnehmen.

6.2 Verwendung der RFID beim ASURO

Es ist theoretisch auch möglich einen ASURO mit der RFID -Technik auszurüsten, und so unser Projekt noch weiter zu verfeinern. Jedoch ist eine solche Umrüstung sehr

kostenintensiv und würde neben einer Datenbank zum Vergleichen von empfangenen Daten und der Hardware auch sehr wahrscheinlich eine zusätzliche Energiequelle benötigen. Deshalb bleibt die Anwendung auf dem ASURO vorerst eine Theorie, die wie folgt aussieht: Am ASURO würde ein Lesegerät angebracht werden, am besten in Form der Greifer. Das elektromagnetische Feld könnte sich dann wie bei einem Plattenkondensator zwischen den Greifarmen ausbilden. Die zu sortierenden Gegenstände würden jeweils mit einem Transponder ausgerüstet werden, die vermutlich Etikettenform haben würden, da diese Transponder an nahezu jedem Objekt angebracht werden können. Wenn der ASURO nun mit dem Lesegerät einen solchen Gegenstand erfasst, so kann der Transponder Informationen über den Gegenstand an den ASURO senden.

Die praktische Anwendung könnte dann wie folgt aussehen:

Man könnte Lebensmittelverpackungen wie Aluminiumdosen oder Kunststofftüten mit Transpondern ausrüsten. Der ASURO könnte dann an einer Sammelstelle für recycelbaren Müll eingesetzt werden, um dort eine Feintrennung des Mülls, in diesem Fall von Kunststoff und Aluminium, vorzunehmen. Die Transponder würden bei Kontakt mit dem Feld des Lesegeräts, das auf dem ASURO montiert ist, senden aus welchem Material der Gegenstand besteht auf dem sie montiert sind. Dem ASURO wäre es so möglich Aluminium und Kunststoffe zu trennen. Das zugrunde liegende Programm wäre das selbe, wie das zur Trennung der verschiedenfarbigen Holzklötze verwendete Programm, mit dem Unterschied, dass der ASURO zur Unterscheidung der Gegenstände die Informationen aus dem Lesegerät verwenden würde, anstatt die aus der Odometrie.

7 Anhang

7.1 Quelltext-Legende

Sämtliche Quelltext-Beispiele in dieser Dokumentation werden nach demselben Schema dargestellt:

```
int i;  
for(i=0; i==20; i++)  
{  
    printf(i.". Durchgang\n\r"); //Kommentar  
}
```

for(...): Quelltext wird in Monospace-Schrift dargestellt.

32: Zahlen werden rot dargestellt.

"abcd": Zeichenketten werden ebenfalls rot dargestellt.

int: Variablentypen werden blau dargestellt.

printf: Funktionen, die in der Standardbibliothek von C deklariert sind, werden ebenfalls blau dargestellt.

//abcd: Kommentare werden hinter doppelte Schrägstriche geschrieben und grün dargestellt.

7.2 Quelltext

7.2.1 ASURO eins und zwei

```
#include "asuro.h"
#define HALT 80

int main(void)
{
    Init();

    unsigned int Ftrans[2];
    unsigned char abgrund = FALSE;

    FrontLED(ON);
    MotorDir(FWD,FWD);
    while(1)
    {
        MotorDir(FWD,FWD);
        if(!(abgrund))
        {
            LineData(Ftrans);
            if (Ftrans[1] > Ftrans[0])
            {
                MotorSpeed(90,230);
                BackLED(OFF,ON);
                StatusLED(GREEN);
                abgrund = FALSE;
            }
            else if((Ftrans[0] < HALT) && (Ftrans[1] < HALT))
            {
                MotorDir(BREAK,BREAK);
                MotorSpeed(0,0);
                StatusLED(RED);
                abgrund=TRUE;
            }
        }
    }
}
```

```

        else
        {
            MotorSpeed(230,90);
            BackLED(ON,OFF);
            StatusLED(YELLOW);
            abgrund = FALSE;
        }
    }
    else if(abgrund)
    {
        MotorDir(RWD,RWD);
        MotorSpeed(0,140);
        Msleep(1200);
        MotorDir(FWD,FWD);
        MotorSpeed(150,150);
        Msleep(100);
        abgrund = FALSE;
    }
}

return 0;
}

```

7.2.2 ASURO drei

```
#include "asuro.h"
#define HALT 100
#define ODOMETRIE 995

#define LINKS 0
#define RECHTS 1

void Kurve(int LinksRechts, int VorZurueck)
{
    MotorDir(VorZurueck,VorZurueck);
    if (LinksRechts == RECHTS)
    {
        MotorSpeed(195,0);
    }
    else if (LinksRechts == LINKS)
    {
        MotorSpeed(0,230);
    }
    Msleep(800);
}

void Kurve2(int LinksRechts, int VorZurueck)
{
    MotorDir(VorZurueck,VorZurueck);
    if (LinksRechts == RECHTS)
    {
        MotorSpeed(200,60);
    }
    else if (LinksRechts == LINKS)
    {
        MotorSpeed(60,200);
    }
    Msleep(620);
}
```

```

int main(void)
{
    Init();

    unsigned int Ftrans[2];
    unsigned int Otrans[2];

    int Richtung = LINKS;
    int Status = 0;

    int i;

    FrontLED(ON);

    while(1){
        switch(Status)
        {
            case 0:
                LineData(Ftrans);
                StatusLED(RED);
                if(Ftrans[LINKS] < Ftrans[RECHTS])
                {
                    MotorSpeed(80,190);
                }
                else if(Ftrans[LINKS] > Ftrans[RECHTS])
                {
                    MotorSpeed(190,80);
                }
                if((PollSwitch() == 2) || (PollSwitch() == 16))
                {
                    Status = 1;
                }
                break;
            case 1:
                MotorSpeed(0,0);

```

```

for(i=1;i<=10;i++)
{
    OdometrieData(Otrans);
    if(Otrans[LINKS] > ODOMETRIE)
    {
        Richtung = RECHTS;
        BackLED(OFF,ON);
    }
    else if(Otrans[LINKS] < ODOMETRIE)
    {
        Richtung = LINKS;
        BackLED(ON,OFF);
    }
}
Msleep(1000);
Status = 2;
break;
case 2:
    Kurve(Richtung,FWD);
    MotorDir(FWD,FWD);
    Status = 3;
    break;
case 3:
    LineData(Ftrans);
    StatusLED(RED);
    if(Ftrans[LINKS] < Ftrans[RECHTS])
    {
        MotorSpeed(80,190);
    }
    else if(Ftrans[LINKS] > Ftrans[RECHTS])
    {
        MotorSpeed(190,80);
    }
    if((Ftrans[RECHTS] < HALT) && (Ftrans[LINKS] < HALT))
    {
        StatusLED(YELLOW);
        MotorDir(BREAK,BREAK);
        MotorSpeed(0,0);
        Status = 4;
    }
}
break;

```

```
        case 4:
            MotorDir(RWD,RWD);
            MotorSpeed(200,200);
            Msleep(550);
            Kurve2(Richtung,FWD);
            Status = 0;
            break;
    }
}
return 0;
}
```

7.3 Schaltplan

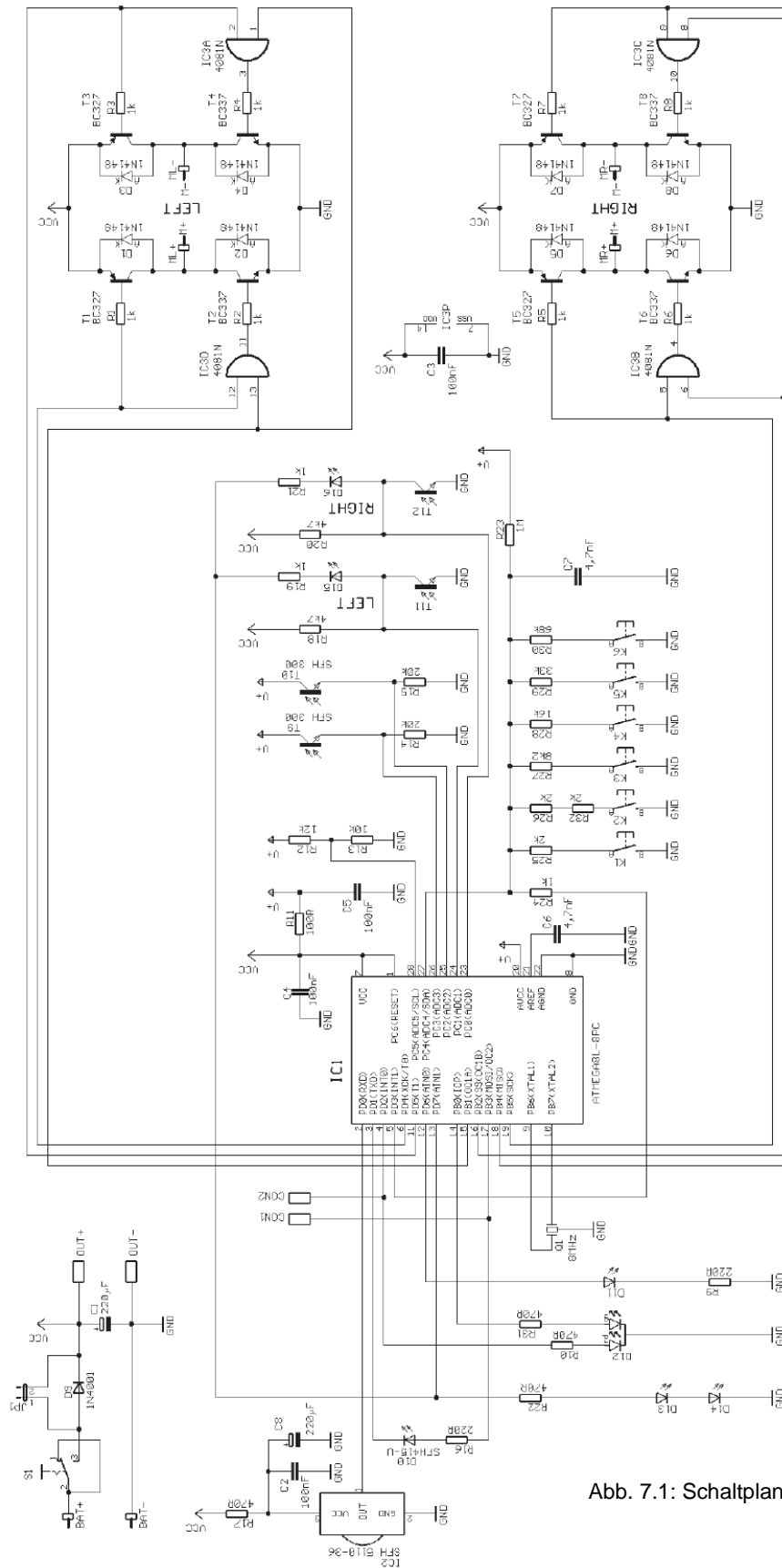


Abb. 7.1: Schaltplan

8 Quellenverzeichnis

8.1 Literatur

- Mehr Spaß mit ASURO Band 1
- ASURO Bau- und Bedienungsanleitung
- Unterlagen, welche wir während des Projekts an Treffen erhalten haben
- Erhaltene Unterlagen von Herrn Stern

8.2 Internet

- <http://arexx.com>
- <http://sander-electronic.de/be00001.html>
- http://itwissen.info/definition/lexikon/__diode_diode.html
- <http://roboternetz.de>
- <http://mikrocontroller.net>
- <http://asurowiki.de>

8.3 Bildquellen

- Abb. 2.1: http://leds.de/images/product_images/info_images/8000330560.jpg
- Abb. 2.2: http://leifi.physik.uni-muenchen.de/web_ph10/grundwissen/04_schichtwiderst/ph-06b1.gif
- Abb. 2.3: <http://electronix.com/catalog/images/diode.jpg>
- Abb. 2.4: Physikbuch 12/13
- Abb. 2.5: Skript von Markus Walter
- Abb. 2.6: Unterrichtsaufschrieb Physik 12
- Abb. 2.7: <http://davr.org/elec/ATMega8-28.JPG>
- Abb. 2.8: <http://asuro.pytalhost.de/pmwiki/uploads/Main/processor.jpg>
- Abb. 6.1: <http://www.rfid-journal.de/rfid.jpg>
- Abb. 1.1, Abb. 3.1, Abb. 3.2, Abb. 3.3, Abb. 3.4, Abb. 3.5, Abb. 7.1: AsuroManual_deu.pdf von der ASURO-CD
- Abb. 5.1, Abb. 5.2, Abb. 5.3: ©2007 Andreas Hasenfuß, Dominik Walter, Paul Vorbach