Projektdokumentation für den Fußballroboter „Jaqueline“

Rene Peschmann und Ronny Gorzelitz

Inhaltsverzeichnis

Entwicklung von Jaqueline

Das Getriebe und Gehäuse

Das Getriebe wird mit jeweils 2 Modellbaumotoren für jede Seite angetrieben. Dabei wurde eine Übersetzung von 1:125 angestrebt. Nach mehreren Versuchen wurden die Zahnräder horizontal in Übersetzung gebracht. Dabei wurde immer eine Übersetzung von 1:5 beachtet.

Durch den Versuch den Schwerpunkt so niedrig wie möglich zu halten, wurde der ganze Roboter im Verhältnis zu seiner Größe eher flach.

Die Schussvorrichtung und der Käfig

Der Käfig wurde sehr einfach gehalten durch 2 Stangen. Diese sollten den Ball nur für eine Drehung des Roboters festhalten. Der Käfig selber wurde durch einen Servomotor mit einer Kette geöffnet bzw. geschlossen.

Mit der Schussvorrichtung wurde der Ball noch mal in der Vorwärtsrichtung beschleunigt. Auch die Schussvorrichtung wurde mit einem Servomotor betrieben.

Die Ballsensoren

Die Erkennung des Balles erfolgte über einen IR-Empfänger oberhalb und 3 IR-Empfänger an der Front des Roboters. Der IR-Empfänger oberhalb wird benutzt um die Umgebungshelligkeit zur erkennen. Davon wird jeweils die erkannte Helligkeit der 3 IR-Empfänger an der Front abgezogen. Ist der dabei entstandene Wert größer als ein Schwellwert, so ist in dieser Richtung der Ball.

Die Torsensoren

Zum erkennen der Tore wurden alle 4 Sensoren im 90° Winkel angebrachte. Dadurch konnte ohne Ausrichtung der Sensoren eine grobe Richtung zum Tor erkannt werden. Damit die Tore auch über das ganze Spielfeld erkannt wurden, musste die Fehlerrate größer gewählt werden.

Software für Jaqueline

Definitionen

Bei den Definitionen wurde darauf geachtet einige Details nicht immer wieder im Programm zu verwenden, um dann bei kleinen Änderungen den ganzen Quelltext zu ändern.

Typendefinitionen für Bewegung

```
typedef enum
{     LINKS,
      RECHTS,
      VORWAERTS,
      RUECKWAERTS,
      STOP,
} RICHTUNG;
```

Mit der Enumeration wurden alle möglichen Fahrtrichtungen aufgezählt. Dadurch konnte eine Funktion **`setMotor(RICHTUNG r)`** festgelegt werden und dort wurden dann die entsprechenden Motoren gesetzt.

Definitionen der Winkel für die Servomotoren

```
#define KAEFIG_AUF 45
#define KAEFIG_ZU 20

#define SCHUSS_VOR 35
#define SCHUSS_ZURUECK 85
```

Hier wurden die Winkel festgelegt um dann bei einer mechanischen Änderung die Winkel nur neu zu definieren.

Definitionen der Anschlüsse für die einzelne Peripherie

```
#define SERVO_SCHUSS          1
#define SERVO_KAEFIG          2

#define BALL_VORNE_RECHTS     0
#define BALL_HINTEN_RECHTS    1
#define BALL_HINTEN_LINKS     2
#define BALL_VORNE_LINKS      3
#define BALL_VORNE_VORNE      4
#define BALL_VORNE_SONNE      5
#define BALL_DETECTION        6
```

Durch eine Definition der Anschlüsse konnte im Programm nur noch diese Definition verwendet werden. Bei einer Umgestaltung des Roboters müssen nur die Definition für die Anschlüsse entsprechend neu angepasst werden.

Funktionen von Jaqueline

Setzen der PWM für die Motoren

```
void setMotorSpeed()
{     motor_pwm(0, 10);
      motor_pwm(1, 10);
      motor_pwm(2, 9);
      motor_pwm(3, 9);
}
```

Hier wurde die Pulsweitenmodulation der einzelnen Motoren festgelegt. Die eine Seite erzeugte eine etwas größere Drehzahl. Dadurch bewegte sich der Roboter in einem größeren Kreis. Durch unterschiedliche festgelegte PWM für beide Seiten konnte sie der Roboter wieder halbwegs geradeaus bewegen.

Erzeugen einer Zufallszahl

```
int random() {
      long f;
      f = akt_time();
      f = f * 11213; f = f % 61 + f % 521; f = f % 255;
      return f;
}
```

Berechnen einer einfachen Zufallszahl zwischen 0 und 255. In einem Testlauf mit 150.000 Zahlen kam ein Durchschnitt von etwas unter 127 raus.

Setzen der Motorrichtung

```
void setMotor(RICHTUNG direct)
{
      static RICHTUNG lastDirect = STOP;

      if (lastDirect == direct) return;
      if ((lastDirect == STOP) && (direct != STOP))
            setMotorSpeed();

      // Motoren setzen
      switch(direct)
      {
      case LINKS:             motor_richtung(0, 0);
                              motor_richtung(1, 0);
                              motor_richtung(2, 1);
                              motor_richtung(3, 1);
                              break;

      case RECHTS:            motor_richtung(0, 1);
                              motor_richtung(1, 1);
                              motor_richtung(2, 0);
                              motor_richtung(3, 0);
                              break;

      case VORWAERTS:         motor_richtung(0, 0);
                              motor_richtung(1, 0);
                              motor_richtung(2, 0);
                              motor_richtung(3, 0);
                              break;

      case RUECKWAERTS:       motor_richtung(0, 1);
                              motor_richtung(1, 1);
                              motor_richtung(2, 1);
                              motor_richtung(3, 1);
                              break;

      case STOP:              motor_pwm(0, 0);
                              motor_pwm(1, 0);
                              motor_pwm(2, 0);
                              motor_pwm(3, 0);
                              break;
      }
      lastDirect = direct;
}
```

Das setzen einer neuen Motorrichtung erfolgte über die zuvor definierten Enumerations. Ein alte Richtung wird gemerkt und bei einem erneuten setzen gleich „ignoriert“.

Setzen des Tores

```
void setGoal(int goal)  // 0 für 125 MHz
{
      // Empfänger setzten ... dabei zeigt der erste Sensor
      // in Fahrtrichtung ... anschließend geht es im
      // Uhrzeigersinn weiter
      mod_ir0_takt(4 + goal);
      mod_ir0_maxfehler(4 + goal);
      mod_ir1_takt(4 + goal);
      mod_ir1_maxfehler(4 + goal);
      mod_ir2_takt(4 + goal);
      mod_ir2_maxfehler(4 + goal);
      mod_ir3_takt(4 + goal);
      mod_ir3_maxfehler(4 + goal);
}
```

Setzt die IR-Sensoren auf 100 MHz bzw. 125 MHz. Um die IR-Sensoren auf 125 MHz zu setzen musste der Funktion der Wert 0 übergeben werden. Bei einer anderen Zahl wurde entsprechend auf eine andere Frequenz gesetzt.

Entgegen der empfohlen Fehlerrate in der Anleitung, wurde ein erhöhte Fehlerrate zugelassen. Dadurch konnten die Tore fast über das komplette Spielfeld erkannt werden.

Initialisierung des Systemes

```
void initSystem()
{
      setMotor(STOP);
      servo_arc(SERVO_KAEFIG, KAEFIG_ZU);
      sleep(200);
      servo_arc(SERVO_KAEFIG, KAEFIG_AUF);
      servo_arc(SERVO_SCHUSS, SCHUSS_ZURUECK);
      setGoal(0);
}
```

Bei der Initialisierung des Systems werden die Winkel der Servomotoren getestet. Dadurch lies sich die Mechanik des Käfigs und der Schußvorrichtung auf Funktion überprüfen. Anschließend werden die IR-Sensoren auf das 125 MHz Tor gesetzt.

Initialisierung der Wanderkennung

```
void wandinit()
{
      led(0,1);   led(1,1);
      led(2,1);   led(3,1);
}
```

IR-Sender auf senden schalten um ein Hindernis zu erkennen.

Initialisierung der Ballkennung

```
void ballinit()
{
      led(0,0);   led(1,0);
      led(2,0);   led(3,0);
}
```

IR-Sender ausschalten um den Ball zu erkennen.

Schuss des Balles

```
void schuss()
{
      int lasttime = 0;

      // mein Ball !!!
      servo_arc(SERVO_KAEFIG, KAEFIG_ZU);
      sleep(300);

      lasttime = akt_time();
      while ((lasttime + 2000) < akt_time()) fahreTor();

      // dann vorwärts, Käfig auf und Ball schießen
      setMotor(VORWAERTS);
      sleep(500);
      servo_arc(SERVO_KAEFIG, KAEFIG_AUF);
      sleep(100);
      servo_arc(SERVO_SCHUSS, SCHUSS_VOR);

      // jetzt wegen Spielregeln anhalten, zurück wäre sinnlos
      //  da  wir  den  Ball  eh  nicht  in  1  Sekunde  wieder
einfangen
      setMotor(STOP);
      sleep(400);
      servo_arc(SERVO_SCHUSS, SCHUSS_ZURUECK);
      sleep(500);
}
```

In der Funktion **`schuss()`** wird der Ball zielstrebig zum Tor gebracht. Dabei wird zuerst der Käfig geschlossen, um den Ball bei einer Rotation nicht wieder zu verlieren. Anschließen wird etwa 2 Sekunden lang das Tor gesucht bzw. in dessen Richtung zu fahren. Nach etwa 2 Sekunden wird der Käfig wieder geöffnet und der Ball wird zum Tor geschossen. Nach dem Schuss bleibt der Roboter wegen der Spielregeln 1 Sekunde stehen. Innerhalb dieser Funktion wird auf kein Hindernis geachtet.

Tor suchen

```
void fahreTor()
{
      static int timeout = 0;

      int vorne  = mod_ir0_status();
      int links  = mod_ir1_status();
      int hinten = mod_ir2_status();
      int rechts = mod_ir3_status();
      const int SCHWELLE = 5;

      if (rechts > SCHWELLE)
      {
      setMotor(LINKS); timeout = 0; return;
      }

      if  (links > SCHWELLE)
      {
      setMotor(RECHTS); timeout = 0; return;
      }

      if (vorne > SCHWELLE)
      {
      setMotor(VORWAERTS); timeout = 0; return;
      }

      if (hinten > SCHWELLE)
      {
      setMotor(RECHTS); timeout = 0; return;
      }

      if (!timeout)
      {
      timeout = akt_time();
      } else
      {
      if (akt_time() - timeout > 1000)
      {
            timeout = 0;
            setMotor(RECHTS);
      }
      }
}
```

Die Richtung in die gefahren wird um zum Tor zu gelangen ist Priorisiert. Dabei wird erst Links vor Rechts angefahren, dann folgt Vorwärts und zum Schluss zurück. Sollte der Kontakt zum Tor verloren gehen, so wird noch eine 1 Sekunde vorwärts gefahren. Nach dieser Sekunde dreht sich der Roboter automatisch rechts herum.

Suchen des Balls

```
RICHTUNG ballsuchen()
{
      int rechts, mitte, links, sonne;
      ballinit();

      sonne = analog(BALL_VORNE_SONNE);
      rechts = sonne - analog(BALL_VORNE_RECHTS);
      mitte = sonne - analog(BALL_VORNE_VORNE);
      links = sonne - analog(BALL_VORNE_LINKS);

      // kleine Signalunterschiede ausblenden
      if (rechts > 15)  rechts = 1; else rechts = 0;
      if (mitte  > 15)  mitte = 1;  else mitte = 0;
      if (links  > 15)  links = 1;  else links = 0;

      if ((rechts + links + mitte) == 1)
      {
      if (rechts) return RECHTS;
      if (links)  return LINKS;
      if (mitte)  return VORWAERTS;
      }

      // kein Ball gefunden
      return STOP;
}
```

Beim Ball suchen wird einmal die Umgebungshelligkeit erkannt und davon wird dann jeweils der Sensor für rechts, Links und Vorne subtrahiert. Anschließend wird dieser Wert in einen boolschen Wert umgerechnet. Dabei wird einfach nur geprüft ob der Wert über 15 ist und somit auf true (bzw. 1), sonst auf false gesetzt. Anschließend werden alle Richtungen addiert und wenn dann eine 1 rauskommt, wird die entsprechende Richtung von der Funktion zurückgegeben. Wird kein Ball erkannt, so wird ein STOP-Signal zurückgegeben.

Hauptfunktion

```
void AksenMain(void)
{
      RICHTUNG wish; // diese Richtung kommt von ballsuchen()
      initSystem();
      while(1)
      {
            if (analog(BALL_DETECTION) < 100) schuss();
            wish = ballsuchen();
            if (wish == STOP)
            {
                  // kein Ball
                  wish = RECHTS;
            }
            setMotor(wish);
      }
}
```

Hier wird in einer Endlosschleife überprüft ob der Ball eingefangen wurde, dann wird die Funktion **`schuss()`** aufgerufen. Oder es wird der Ball gesucht, dabei gibt die Funktion eine Empfehlung zurück, also in welche Richtung der Ball liegt bzw. in welche Richtung gefahren werden sollte. Anschließend wird diese Richtung gesetzt. Geplant war hier noch eine Funktion aufzurufen die überprüft ob in die gewünschte Richtung gefahren werden kann, also eine Hindernis Erkennung.

Fazit

Zu wenig Zeit. Es ist viel Zeit verloren gegangen bei der Entwicklung des Getriebes. Des Weiteren fehlte eine genauere Technische Planung des Roboters am Anfang. Eine genauere Umsetzung der Softwarelösung haperte ebenfalls, da es viele Möglichkeiten für eine Lösung gab.

Insgesamt war der Roboter durch die 2 Servomotoren viel zu schwer. Eine Lösung mit leichteren Servormotoren wäre für die Belastung des Vorderrates besser gewesen. Gleichzeitig würde der Stromverbrauch durch eine geringere Belastung des Antriebes sinken. Alternativ wäre eine gleichmäßigere Verteilung des Gewichtes auf alle Achsen besser gewesen. Dafür hätte man aber das gesamte Gehäuse neu entwickeln müssen.