Erstellen einer Kompasskarte

Hannes Seidel (992075),
Jan Tobias Mühlberg (992024)

24. Januar 2003

Inhaltsverzeichnis

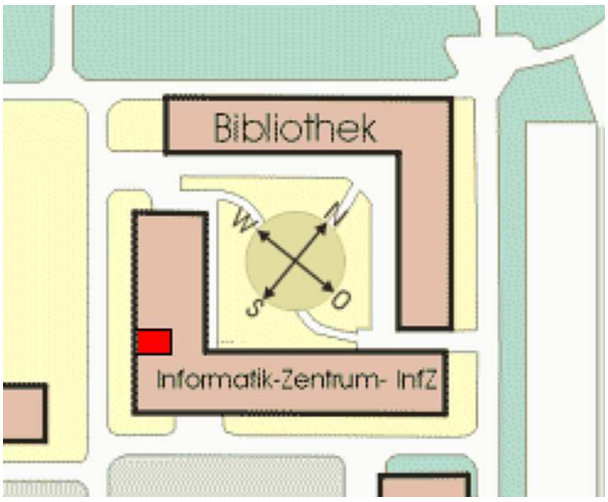

Abbildung 1: Lage des KI-Labors (rote Fläche) und tatsächlicher magnetischer Nordpol (entnommen aus Brandschutzplan der FHB)

1 Aufgabenstellung

Erstellen Sie beginnend vom Punkt E eine Karte von der Freifläche vor den Schränken mit den Richtungen des Magnetfeldes (Magnetfeldkarte). Die Kachelgrösse der Karte sollte mindestens 10cm x 10cm betragen. Visualisieren Sie die gefundenen Richtungsvektoren in einer graphischen Darstellung (z.B. mittels gnuplot). Klären Sie folgende Fragen:

- Ist das Magnetfeld auf der Testfläche ortsabhängig?
- Wie beeinflusst der Roboter das Magnetfeld?
- Ist das Magnetfeld zeitlich stabil?
- Wie sollte der Kompass angebracht werden?

Wiederholen Sie den Versuch mehrmals um ein stabiles Messergebnis zu erhalten.

2 Das Testszenarium

Die Testfläche, die Freifläche vor den Schränken, befindet sich im KI-Labor des Informatik Zentrums und ist ca. 4m (Schrankseite) x 2m groß. Die Schrankseite ist die, dem Nordwesten zugewandte Wand (siehe Abb. 1).

3 Vorüberlegung

Bevor wir mit der endgültigen Programmierung begannen, mußten wir einige Fragen klären und gegeneinander abwägen. So zum Beispiel, ob wir senkrecht oder parallel zum Schrank fahren wuerden. Vorteil beim parallelem Kurs wäre zum Beispiel die geringere Anzahl der Drehungen, welche sich weniger stark auf Richtungsänderungen auswirken wuerden. Als Nachteil des parallelen Kurses ist aber die Ungenauigkeit der Sonare auf großen Entfernungen und das ständige Ausrichten parallel zur Wand, welches eine nicht zu unterschätzende Fälschung der Kompasswerte zur Folge hätte, zu erwähnen. Da die Nachteile bei einem senkrechtem Abfahren der Fläche uns weniger verheerend erschienen, entschlossen wir uns für diese Variante. Wir überlegten uns, wie wir die Kompasswerte messen. Entweder wir lassen den den Roboter 10cm fahren, anhalten, messen, und weitere 10cm fahren, oder wir bestimmen anhand der Sonare, welche ja den Abstand zum Schrank anzeigen sollten, wann eine Messung erfolgt. Dabei kann der Roboter in einem gleichmässigem Tempo eine Wegstrecke zurücklegen, ohne mit den Problemen eines eventuell auftretenden Schlupfes beschäftigt zu werden. Wir entschieden uns fuer die letztere Möglichkeit. Eine weitere Frage war, ob wir beim Vorwärts- oder beim Rückwärtsfahren Messwerte aufnehmen. Wir entschieden uns, aus Zeitgründen, sowohl beim Vorwärts- also auch beim Rückwärtsfahren Messwerte aufzunehmen. Die Genauigkeit der Messpunkte ist dabei zwar nicht mehr so hoch, sind aber durchaus noch im Bereich des Akzeptablen.

4 Die Navigation des Roboters

Zur Realisierung des korrekten Abfahrens der zu kartierenden Fläche erstellten wir eine Colbert Activity für die Navigation und eine DLL Datei für die Schreiboperationen in eine Log- Datei. Die Activity lädt standardmässig die DLL Datei mit, so daß man sich darum nicht mehr kümmern muss.
Mit dem Starten der geladenen Activity beginnt der Roboter, sich zur Schrankwand auszurichten. Dazu dreht er sich solange gegen den Uhrzeigersinn, bis sich die Sonare 3 und 4 einigermaßen gleichen. Dabei berücksichtigen wir nur Objekte, die sich in einem Radius von 1,1m (Punkt E ist 90 cm vom Schrank entfernt) befinden. Damit wird vermieden, daß eventuell sich in Sensorreichweite befindende Personen, welche mit offenem Mund dem Geschehen folgen, als Schrankwand erkannt werden. Danach beginnt die Feinausrichtung, so dass der Roboter eine Position genau 90 Grad zur Schrankwand einnimmt. Ist die richtige Position gefunden, nähert sich der Roboter bis auf 25 cm der Wand. An dieser Stelle beginnt das Abfahren des Parcours. Der Roboter fährt nun solange rückwärts, wie die Werte der Sonare 3 und 4 innerhalb der definierten Länge liegen. In unserem Fall sind dies, wegen der Anord-

nung der Tische, ca. 1,6m. Während des Rückwärtsfahrens werden ständig die Sonare 3 und 4 kontrolliert. Ist der Abstand zur Wand um 10 cm gestiegen, so wird die Funktion „LOG_print()“, die über die geladene „task_8.dll“ bereitgestellt wird, aufgerufen. Dabei werden die Parameter Anzahl der Bahnen *turn_counter*, Fahrtrichtung *turn_direction* und Kompasszahl *sfRobot.compass* übergeben und in die Datei c:\compass.log geschrieben. Die Genauigkeit der gemessenen Kompasswerte hängt dabei sehr stark von den Messungen der Sonare, aber auch von der Geschwindigkeit des Roboters ab. Bei der voreingestellten Geschwindigkeit von 100 mm/sek ergeben sich bei den einzelnen Messungen Sonarentfernungen mit einem Fehler ± 10mm. Um der Abdrift vorzubeugen, richten wir den Roboter auf dieser Strecke neu aus. Da das passive Spornrad sich in der richtigen Fahrtrichtung befindet, wird der Roboter auf der Hälfte der Strecke ausgerichtet.
Ist der Roboter mit seiner Rückwärtsfahrt fertig, d.h. wenn die Sonare 3 und 4 mehr als 1,6m zurückliefern, wird die nächste Bahn angepeilt. Dazu dreht sich der Roboter um 100°, fährt 10 cm weit, und dreht sich wieder um 100° zurück. Die 100° deswegen, weil der Roboter sich nach diesen zwei Drehungen immer noch auf der gleichen Höhe befindet, d.h. die Sonare vor und nach der Drehung einigermaßen gleiche Werte anzeigen, was bei zwei 90° Drehungen nicht der Fall wäre. Hier wird nun die Richtungsvariable gesetzt, um später im Logfile herausbekommen zu können, wann die neü Bahn beginnt.
Nun beginnt der Roboter, sich wieder auf die Schrankwand zuzubewegen. Auf ein Ausrichten beim Vorwärtsfahren haben wir verzichtet, da der Roboter noch gerade genug steht. Sollte er ein wenig abgedriftet sein, wird er beim nächsten Rückwärtsfahren wieder ausgerichtet. Auch bei der Vorwärtsfahrt wird alle 10cm ein Messwert an die „LOG_print()“ Funktion übergeben. Bei einem Sonarabstand von 25cm hält der Roboter und beginnt die nächste 100° Drehnung. Danach 10cm Fahrt, und die 100° Drehung zurück. An dieser Stelle würde der Greifer, würden nur eine 90° Drehung ausgeführt werden, mit grosser Sicherheit mit dem Schrank kollidieren. Die Richtungsvariable wird erneut gesetzt, die Anzahl der Bahnen um eins erhöht, und die ganze Prozedur beginnt von vorn, solange, bis alle Bahnen abgefahren sind. Das Ende wird, falls alles glatt gegangen ist, mit einer Erfolgsmeldung quittiert.
Bei den derzeitigen Einstellungen benötigt der Roboter für das Abfahren der gesamten Teststrecke rund 11 Minuten.

5 Die Datenerfassung

Um die Auswertung der während der Fahrt vom Roboter erfassten Daten möglichst einfach und kompfortabel zu gestalten, sollten diese aus der Colbert Activity heraus, direkt in eine Datei auf der Festplatte des Laptops

geschrieben werden. Hierzu wurde eine DLL mit der entsprechenden Funktionalität entwickelt. Der Einfachheit halber wird auf diese DLL hier nicht weiter eingegangen, der dokumentierte Quelltext findet sich im Anhang dieser Arbeit bzw. auf dem beigefügten Datenträger.
Bei jeder Messung werden folgende Werte in die Logdatei geschrieben: Datum und Uhrzeit, die Nummer der gerade aktuellen Runde, die aktuelle Bewegungsrichtung des Roboters und der von Kompass zurueckgegebene Wert, der die Abweichung der Kompasstellung zum Nordpol angibt. Ein solcher Logeintrag könnte also beispielsweise folgendes Aussehen haben, wobei das „ d:“ für „data“ steht und lediglich zum die nachträgliche automatische Verarbeitung der Logdatei vereinfachen soll:

```
01-22-2003 15:21:35 d: turn 1, direction 0, value 4
```

Darüber hinaus werden noch weitere Ereignisse wie z.B. der Anfang und das Ende einer Messreihe protokolliert.

6 Probleme bei der Versuchsdurchführung

Eines der grössten Problem stellen die Sonare dar. Während man mit den Schwankungen der Sonarwerte noch einigermaßen arbeiten kann, passiert es auch manchmal, dass Aussetzer oder nicht wieder eingefangene Echos den Roboter zu Tätigkeiten zwingen, die noch nicht gewollt sind. So passierte es mehrmals, daß der Roboter beim Rückwärtsfahren, obwohl er sich noch innerhalb des definierten Bereiches befand, meinte, er wäre bereits über sein Ziel hinausgeschossen.
Darüber hinaus hinaus scheint es im Informatik Zentrum auch kaum möglich zu sein, eine korrekte Messung mit einem Kompass durchzuführen. Während die Wand mit den Schränken Abbildung 1 zufolge eher dem Nordwesten zugewandt sein sollte, zeigten alle unsere im innern des KI-Labors durchgeführten Messungen mit einem handelsüblichen Magnetkompass, daß sich die Wand ziemlich genau von West nach Ost erstreckt und dementsprechend dem Norden zugewandt ist. Da auch der Kompass des Roboters nichts anderes als das in dem Raum vorhandene Magnetfeld messen kann, wird im weiteren Versuchsverlauf davon ausgegangen, daß sich die Wand im Norden befindet. Eine Messung, bei der der Kompass des Roboters auf die Wand gerichtet ist und eine Abweichung von etwa 0° zurückgibt, wird dementsprechend als korrekt gewertet.

7 Messergebnisse und Bewertung

Die Abbildungen 2 und 3 stellen das mittels des Kompasses gemessene Magnetfeld dar. Die Pfeile bezeichnen hierbei die gemessene Nordrichtung. Die oberen Kanten der Abbildungen stellen die Wand mit den Schränken dar,

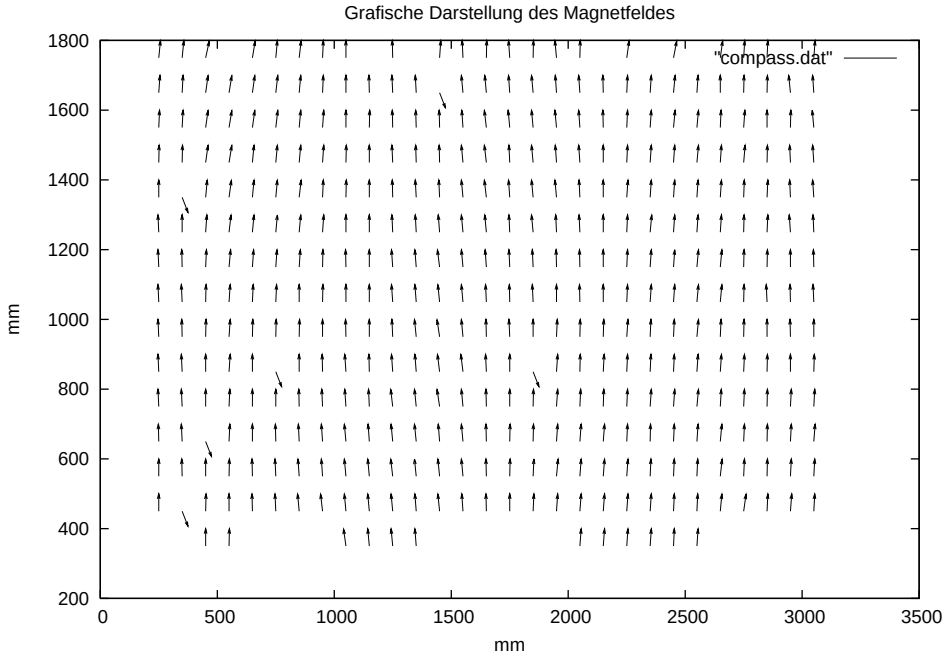

Abbildung 2: Gemessenes Magnetfelt bei Positionierung der Kompasses auf einer Kiste etwa 30cm über dem Roboters

die linke Seite repräsentiert dementsprechend die den Fenstern zugewandte Seite der Testfläche.

7.1 Ortsunabhängigkeit des Magnetfeldes

Im Verlauf des Versuches konte eine Ortsabhängigkeit des Magnetfeldes beobachtet werden. Wie in Abb. 2 erkennbar, weichen auf der den Fenstern zugewandten Seite der zu kartierenden Fläche viele Richtungsvektoren um 5° bis 15° in negativer Drehrichtung, auf der der Tür zugewandten Seite um einen ähnlichen Betrag in positiver Drehrichtung von der im Raum vorherrschenden magnetischen Nrdrichtung ab. Daß dieses Phänomän bei allen durchgeführten Messungen beobachtet werden konnte, legt die Vermutung nahe, daß die Ursache hierfür in der Gebäudekonstruktion und in der Wahl der im Informatik Zentrum verbauten Materialien zu finden ist.

7.2 Beeinflussung des Magnetfeldes durch den Roboter

Auch konnte eine direkte Beeinflussung des Magnetfeldes durch den Roboter festgestellt werden: Befindet sich der Kompass in seiner ursprünglichen Lage auf der rechten Seite des Roboters, zeigen alle Richtungsvektoren um

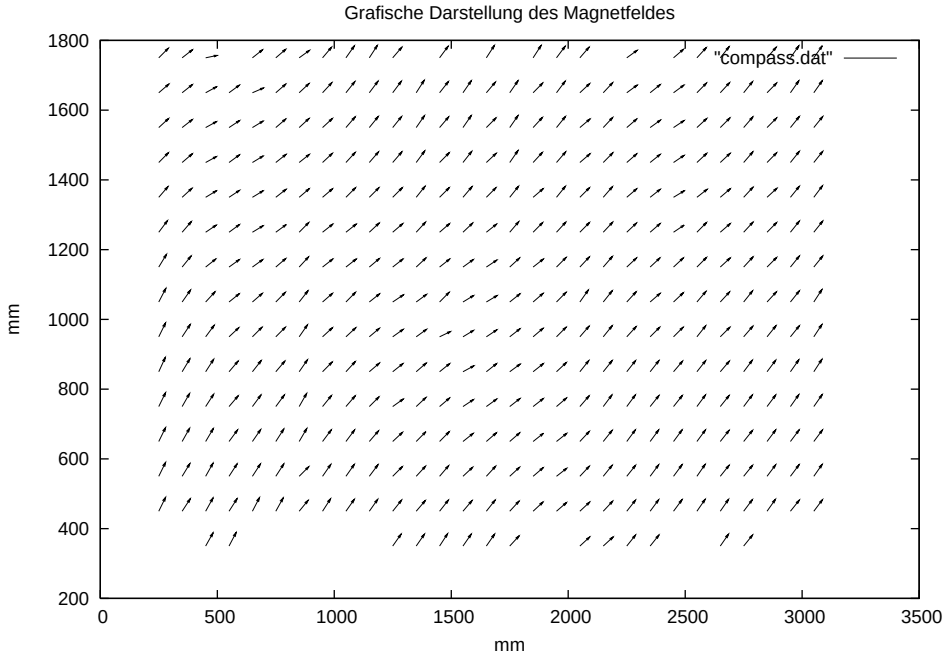

Abbildung 3: Gemessenes Magnetfelt bei unveränderter Positionierung des Kompasses rechts neben der Kammera

durchschnittlich 45° in positiver Drehrichtung von der tatsächlichen Nordrichtung weg (siehe Abb. 3), wird der Kompass auf der linken Seite des Roboters montiert, sind es ca. 45° in negativer Drehrichtung - offensichtlich ist die Robotermitte ein ganz hervorragender Südpol.

7.3 Zeitliche Stabilität des Magnetfeldes

Das gemessene Magnetfeld wies leichte zeitlichen Instabilitäten auf. Es war jedoch nicht möglich, deren genaue Ursachen zu ergründen. Ein Hauptfaktor hierfür ist zweifellos der Roboter selbst. Eine Beinflussung des Magnetfeldes durch die vielen weiteren elektrichen Geräte im KI-Labor ist ebenfalls nicht auszuschliessen, jedoch ohne genauere Informationen über den Kompass bzw. ohne einen genaueren Kompass und eine veränderte, nach Möglichkeit stationäre Versuchsanordnung nicht nachzuweisen.

7.4 Positionierung des Kompasses am Roboter

Nach Durchführung mehrerer Versuche erwies sich eine Positionierung des Kompasses etwa 30cm über dem Roboter als die sinnvollste. Auf diese Weise wurde die, vorher mittels eines konventionelen Handheld-Magnetkompasses gemessene vorherrschende Magnetfeldrichtung im KI-Labor mit einer Ge-

nauigkeit von durchschnittlich $\pm 10^{\circ}$ korrekt ermittelt, was für eine Navigation im Gelände evtl. ausreichend ist. Die Werte blieben auch bei einem auf der Kiste verschobenem Kompass im Rahmen dieser $\pm 10^{\circ}$ konstant, so daß eine Beeinflußung des Magnetfeldes durch den Roboter in dieser Höhe nur eine untergeordnete Rolle zu spielen scheint. Wegen des zu kurzen Verbindungskabels zwischen Roboter und Kompass konte leider nicht mit groesseren Abständen experimentiert werden.

8 Anhang: Quellcodes und Skripte

8.1 task_8.act - Die Activity

```
/*############################################################################*/
/* AIS - WS2002/2003 - Aufgabe 8 - Erstellen einer Kompasskarte im KI Lab
/* Jan Tobias Muehlberg/ 992024 - Hannes Seidel/ 992075
/*############################################################################*/

load c:\muehlber_seidelh\task_8.dll;    /* DLL fuer die Dateiausgabe laden */

act task_8
 {

  int tester;       /* Testvariable */
  int turn_counter;                  /* Anzahl der Bahnen */
  int turn_direction;                  /* Richtungsanzeiger */
  int last_grid;       /* Hilfsvariable Kompassmessung */
  int sonar_value;         /* EntfernungsWerte vom Sonar */
  int length; /* Laenge des Spielfeldes/ Abstand v. Schrank */
  int width;     /* Breite des Spielfeldes */
  int grid_width;   /* Rasterweite der Kompassmessungen */
  int velocity;    /* Geschwindigkeit */

/*############################################################################*/

  length = 1600;      /* Laenge der Karte in mm */
  width = 3000;         /* Breite der Karte in mm */
  grid_width = 100;        /* Rastergrsse */
  velocity = 100;  /* Geschwindigkeit in mm/sek */

/*############################################################################*/

  width = width / 200;          /* Umrechnung der Breite in Anzahl der Bahnen */


/*############################################################################*/
/* Vom Startpunkt E senkrecht zum Schrank ausrichten */
/*############################################################################*/

  tester = 1;
  rotate(10);

  while (tester == 1)
```

```
   {
    if ( (((sfSonarRange(3) + 50) > sfSonarRange(4)) &&
 ((sfSonarRange(3) - 50) < sfSonarRange(4))) &&
         ((sfSonarRange(3) < 1100) && (sfSonarRange(4) < 1100)) )
{
tester = 0;
sfSMessage("Something found!!");

while (tester == 0)
{
        if (((sfSonarRange(3) + 5) < sfSonarRange(4)) &&
            ((sfSonarRange(3) - 5) < sfSonarRange(4)))
          rotate (1);

        if (((sfSonarRange(3) + 5) > sfSonarRange(4)) &&
            ((sfSonarRange(3) - 5) > sfSonarRange(4)))
          \\rotate (-1);

if (  ((sfSonarRange(3) + 2) > sfSonarRange(4)) &&
  ((sfSonarRange(3) - 2) < sfSonarRange(4)) )
  {
rotate(0);
halt;
tester = 2;
sfSMessage("90 Degree found!!");
sfMoveRobotPos(0,0,0);
turnto(0);    /*vermutlich ueberfluessig!*/
  }
}

}

   }


/*############################################################################*/
/* Abfahren des Parcours senkrecht zum Schrank im 10cm Raster */
/*############################################################################*/

turn_counter = 0;
while (turn_counter != width)       /* Fahre alle Runden */
{

   turn_direction = 0;  /* Fahren wir hin(0) oder zurueck(1) */
```

```
   last_grid = length;
   sfSMessage("------ Far --------");
   speed(velocity);

   while ((sfSonarRange(3) > 270) && (sfSonarRange(4) > 270))
    {
/*#####################################################################*/
/*Kompass alle ~10 cm auslesen (vorwaerts)*/
   /*#####################################################################*/
        sonar_value = ((sfSonarRange(3) + sfSonarRange(4)) /2 );
if ((sonar_value <= (last_grid - (grid_width-10))) &&
    (turn_counter != 0))
{
sfSMessage("%d %d ",sfRobot.compass,sonar_value);
LOG_print(3,"turn %d, direction %d, value %d",turn_counter,
\\turn_direction, sfRobot.compass);
last_grid = sonar_value;
}

    }

   if (turn_counter > 0)
    {

    /* Wir sind am Schrank */
     sfSMessage("%d %d ",sfRobot.compass,sonar_value);
     LOG_print(3,"turn %d, direction %d, value %d",turn_counter, turn_direction,
     \\ sfRobot.compass);

     turn(-100); /* 90Grad Drehung, +Minimierung des Versatzes */
     move(grid_width);    /* zur naechsten Bahn fahren (10cm weiter) */
     turn(100);                      /* wieder zur Wand drehen */
     sfMoveRobotPos(0,0,0);              /* fuer Winkel spaeter benoetigt */
     sfSMessage("------ Near --------");
    }

   turn_direction = 1;

   speed(-velocity);
   tester = 0;
   last_grid = 0;

   while ((sfSonarRange(3) < length) && (sfSonarRange(4) < length))
    {
```

```
/*#####################################################################*/
/*Kompass alle ~10 cm auslesen (rueckwaerts)*/
   /*#################################################################*/
        sonar_value = ((sfSonarRange(3) + sfSonarRange(4)) /2 );
if (sonar_value >= (last_grid + (grid_width-10)))
{
sfSMessage("%d %d ",sfRobot.compass,sonar_value);
LOG_print(3,"turn %d, direction %d, value %d",turn_counter,
\\turn_direction, sfRobot.compass);

last_grid = sonar_value;
}

      /*#################################################################*/
      /* bei halbem Weg anhalten und neu ausrichten*/
      /*#################################################################*/

      if ((sfSonarRange(3) < ((length/2) + 15)) && (sfSonarRange(3) >
         ((length/2) - 15)))

{
  speed(0);

        while (tester == 0)
{
        if (((sfSonarRange(3) + 5) < sfSonarRange(4)) &&
            ((sfSonarRange(3) - 5) < sfSonarRange(4)))
          rotate (2);

        if (((sfSonarRange(3) + 5) > sfSonarRange(4)) &&
            ((sfSonarRange(3) - 5) > sfSonarRange(4)))
          rotate (-2);

if (  ((sfSonarRange(3) + 2) > sfSonarRange(4)) &&
  ((sfSonarRange(3) - 2) < sfSonarRange(4)) )
  {
rotate(0);
halt;
tester = 2;
sfSMessage("Direction refreshed");
sfMoveRobotPos(0,0,0);
turnto(0);  /* vermutlich ueberfluessig! */
  }
}
```

```
  speed(-velocity);

}
      /*#########################################################################*/
    }

   /*Wir sind jetzt nicht am Schrank*/
   speed(0);
   turn(-100);
   move(grid_width);
   turnto(0);
   turn_counter = turn_counter + 1;     /* eine Runde geschafft, naechste */
   sfSMessage("Round number %d finished", turn_counter);
  }
 sfSMessage("Successfully finished after %d rounds", turn_counter);

/*#########################################################################*/

}

/*EOF*/
```

8.2 task_8.dll - Die Quellen zur Bibliothek

Die im folgenden aufgeführten C-Quellen implementieren eine DLL zur Protokollierung der vom Magnetkompass auf dem Roboter gelieferten Messdaten in eine Datei. Diese wird in der „config.h“ spezifiziert und ist derzeit auf c:\compass.log gesetzt. Zweifellos ist der hier betriebene Aufwand and Quellcode verglichen mit der tatsächlich genutzten Funktionalität unangemessen hoch. Dies rühr u.A. daher, daß unserer anfänglichen Planung zufolge vieles von dem, was im nachhinein denn doch in der Activity untergebracht wurde, eigentlich in C, also in der DLL implementiert werden sollte. Zu dieser Entscheidung kam es vorallem weil die Activity viel einfacher und schneller getestet werden konnte - das ständige neukompilieren entfällt hier.

compass.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "config.h"
#include "typedefs.h"
#include "log.h"

#include <saphira.h>

/* saphira.h seems to be somewhat bugy */
#define sfINT       1
#define sfFLOAT     2
#define sfVOID      3
#define sfVAR       4
#define sfFUNCTION  5
#define sfEND       6
#define sfACTIVITY  7
#define sfSTRING    8


static struct settings_t settings;


/* Two constants indicating success and failure. They are returned by
 * non-void functions in case of success and failure. */
#define COM_SUCCESS  0
#define COM_FAILURE -1
```

```
/* This function initializes COMPASS. */
void COM_init (settings_t *settings)
 {
#ifdef DEBUG
  time_t ltime;

  ltime = time(NULL);
  srand ((int)ltime);
#endif

  settings->logfilename = LOGFILE;
  settings->loglevel = LOGLEVEL;

  LOG_set_loglevel (settings->loglevel);
  LOG_open (settings->logfilename);

#ifdef DEBUG
  LOG_print (LOG_LEV_WARNING, "This distribution of %s has been build " \
                              "against", PROGNAME);
  LOG_print (LOG_LEV_WARNING, "a Saphira - dummy. Therefore all values " \
                              "returned by");
  LOG_print (LOG_LEV_WARNING, "%s are random values and are useful " \
                              "for testing", PROGNAME);
  LOG_print (LOG_LEV_WARNING, "purpose only. The PRNG has been seeded with " \
                              "the value");
  LOG_print (LOG_LEV_WARNING, "%i.", (int)ltime);
#endif

  LOG_print (LOG_LEV_INFO,"Starting %s v. %s.", PROGNAME, VERSION);
 }

/* When COMPASS finishes, this function should be called. */
void COM_final (settings_t *settings)
 {
  LOG_print(LOG_LEV_INFO, "%s complete.", PROGNAME);
  LOG_close ();
 }


#ifdef DEBUG
/* If we are building with DEBUG we are going to build an executabe for
 * testing purpose only.
```

```
 */
 int main ()
 {
  struct settings_t settings;

  COM_init (&settings);

  COM_final (&settings);

  return (COM_SUCCESS);
 }
#else
/* If not we want to link a DLL for Saphira. Therefore the function
 * name should not be main(). In fact we do not need any functions
 * but we have to initialize the DLL.
 */

 __declspec(dllexport) void sfLoadInit(){

     COM_init(&settings);
     sfAddEvalFn("LOG_print", LOG_print, sfVOID, 5, sfINT, sfSTRING,
          sfINT, sfINT, sfINT);
     sfSMessage("Dll geladen");
 }

__declspec(dllexport) void sfLoadExit(){

 COM_final(&settings);
 sfSMessage("Dll entladen");
 }

#endif
```

log.c

```
#include "stdint.h"
#include <time.h>
#include <errno.h>
#include <string.h>

#include <fcntl.h>
#include <sys/types.h>
/*#include <unistd.h>*/
#include <stdarg.h>
```

```
#include <io.h>


#include "log.h"
#include "config.h"

#define LOG_NOTOPEN -1

#define LOG_STDOUT   1
#define LOG_STDERR   2

/* a handle to the logfile initialized by ::LOG_open */
static int LOG_logfile = LOG_NOTOPEN;

/* the "maximal logging-level". Indicates that there are logged only messages
with a logging level less or equal the maximal logging level. */
static uint8_t LOG_loglevel = LOG_LEV_ERROR;



void LOG_open(char *filename)
 {
  if (LOG_logfile == LOG_NOTOPEN)
   {
    LOG_logfile = open(filename, O_CREAT|O_APPEND|O_WRONLY, 0600);
    if (LOG_logfile < 0)
     {
      LOG_logfile = LOG_NOTOPEN;
      LOG_print(LOG_LEV_ERROR,"Couldn't append to logfile %s.", filename);
      LOG_print(LOG_LEV_WARNING,"Logging will be done to STDERR and STDOUT.");
     }
     else
      LOG_print(LOG_LEV_DEBUG,"Appending to logfile %s.", filename);
   }
 }


void LOG_set_loglevel(uint8_t loglevel)
 {
  if (loglevel < (LOG_LEV_ALL + 1))
   {
    LOG_loglevel = loglevel;
   }
   else
```

```
   {
    LOG_print(LOG_LEV_ERROR,"Unknown loglevel: %d", loglevel);
    LOG_loglevel = LOG_LEV_ERROR;
   }
 }


void LOG_print(uint8_t loglevel, char *format, ...)
 {
  int logfile;
  time_t ltime;
  struct tm *now;
  char ltimestr[40];
  char buf[LOG_MAXLOGBUF + 1];

  va_list ap;
  va_start(ap, format);

  if (loglevel <= LOG_loglevel)
   {
     /* if LOG_file is not open, switch to STDERR & STDOUT*/
     if (!(LOG_logfile != LOG_NOTOPEN))
      {
       if (!(loglevel != LOG_LEV_ERROR))
         logfile = LOG_STDERR;
        else
         logfile = LOG_STDOUT;
      }
     else
      logfile = LOG_logfile;


     ltime = time(NULL);
     now = localtime(&ltime);

     /* Display operating system-style date and time. */
     strftime(ltimestr, 40, "%m-%d-%Y %H:%M:%S", now);
     write(logfile, ltimestr, strlen(ltimestr));

     switch (loglevel)
      {
       case LOG_LEV_FATAL:
        write(logfile, " F: ", 4);
        break;
```

```
       case LOG_LEV_ERROR:
        write(logfile, " E: ", 4);
        break;
       case LOG_LEV_WARNING:
        write(logfile, " W: ", 4);
        break;
       case LOG_LEV_DATA:
        write(logfile, " d: ", 4);
        break;
       case LOG_LEV_INFO:
        write(logfile, " I: ", 4);
        break;
       case LOG_LEV_DEBUG:
        write(logfile, " D: ", 4);
        break;
       default:
        write(logfile, " o: ", 4);
        break;
      }
     buf[0] = '\0';
     vsprintf((char *) buf, (const char *) format, (char *) ap);
     write(logfile, (const void *) buf, strlen((const char *) buf));
     write(logfile, (const void *) "\n", 1);
   }
  va_end(ap);
 }

void LOG_close(void)
 {
  LOG_print(LOG_LEV_DEBUG, "Closing logfile.");
  close(LOG_logfile);
  LOG_logfile = LOG_NOTOPEN;
  LOG_loglevel = LOG_LEV_ERROR;
 }
```

log.h

```
#ifndef __LOG_H
#define __LOG_H


#include "stdint.h"
#include <stdio.h>
```

```
/* The following LOGLEVELS (== output verbosities) are used by
 * COMPASS:
 */
/* constant indicating a fatal error to be logged. Occurrences of
 * a fatal error usually result in program termination */
#define LOG_LEV_FATAL   0

/* constant indicating an error to be logged */
#define LOG_LEV_ERROR   1

/* constant indicating some warning to be logged */
#define LOG_LEV_WARNING 2

/* constant indicating that the following message contains some kind
 * of important data. You should always specify at least this verbosity,
 * otherwise no acqusition of data will be done. */
#define LOG_LEV_DATA    3

/* constant indicating some "Info" to be logged */
#define LOG_LEV_INFO    4

/* constant indicating "Debug Information" to be logged */
#define LOG_LEV_DEBUG   5

/* constant indicating "logging of a not further specified message"
    (any of the messages above) */
#define LOG_LEV_ALL     6




/* Constant indicating the maximal number of bytes allowed for a
   logging message. This number should not be greater than 1024 -
   we always want to be able to write the message withhin a single
   call. */
#define LOG_MAXLOGBUF 1024

/* constant indicating a "not open" logfile */
#define LOG_NOTOPEN             -1

/* file handle for "Standarderror" */
#define LOG_STDERR              2
```

```
void LOG_open(char *filename);

void LOG_set_loglevel(uint8_t loglevel);

void LOG_print(uint8_t loglevel, char *format, ...);

void LOG_close(void);


#endif
```

config.h

```
#ifndef __CONFIG_H
#define __CONFIG_H

#include "log.h"


/* The name of the game. */
#define PROGNAME "COMPASS"

/* The version - it should be increased according to the CVS-tag used.*/
#define VERSION "0.0.2"

/* The default logfile we are using */
#define LOGFILE "c:\\compass.log"

/* And the default logging verbosity. */
#define LOGLEVEL LOG_LEV_ALL


#endif
```

typedefs.h

```
#ifndef __TYPEDEFS_H
#define __TYPEDEFS_H


#include "config.h"
```

```
typedef struct settings_t
 {
  char      *logfilename;    /* logfile to use */
  uint8_t   loglevel;        /* Default debugging level */
 } settings_t;



#endif
```

stdint.h

```
#ifndef __STDINT_H
#define __STDINT_H

typedef signed char             int8_t;
typedef short int               int16_t;
typedef int                     int32_t;


typedef unsigned char           uint8_t;
typedef unsigned short int      uint16_t;
typedef unsigned int            uint32_t;


#endif
```

8.3 Skripte zur graphischen Darstellung

Zum Parsen der Logdateien bzw. zur graphischen Darstellung der aufgezeichneten Messwerte werden die folgenden zwei Skripte genutzt. Bei „plot.sh“ handelt es sich um ein Shellskript, zugegeben, ein nicht sehr performantes, daß die erzeugten Logdateien parst und die Punkte und Koordinaten für die Richtungsvektoren berechnet, in eine Datei schreibt und am Ende *gnuplot* mit „compass.plot“ als Parameterdatei zur Visuaisierung aufruft.

plot.sh

```
#!/bin/bash

#
# This script intends to draw a map from the data measured during one
# of the last runs of COMPASS.
# It is fairly undocumented because the author does not want to touch
# this code again.
#

STEPSIZE=100
XSTART=250
YSTART=250

OLDIFS=$IFS

DATFILE="compass.dat"
TMPFILE1="compass.tmp1"
TMPFILE2="compass.tmp2"

if [ -z $1 ];
 then
  echo "No logfile given - using default (compass.log).";
  LOGFILE="compass.log";
 else
  LOGFILE="$1";
 fi
if [ ! -r "$LOGFILE" ];
 then
  echo "Can't stat $LOGFILE. Exiting";
  exit -1;
 fi

# first we are going to clean up with all the old stuff
```

```
touch $DATFILE
if [ $? -eq 0 ];
 then
  echo "Starting acquisition of data. Writing output to $DATFILE.";
 else
  echo "Can't stat $DATFILE. Exiting.";
  exit -1;
 fi

# let's parse the LOGFILE
rm -f $DATFILE $TMPFILE $TMPFILE2
touch $DATFILE $TMPFILE1 $TMPFILE2
ROWCOUNT=`cat $LOGFILE | grep "I: Starting COMPASS" | wc -l`
 if [ $ROWCOUNT -ge 2 ];
  then
   echo "The logfile contains data of more than one run of COMPASS - ";
   echo "I am evaluating only the first set of data.";
  fi
echo "Parsing file $LOGFILE..."
sed 's/
$//g' $LOGFILE >$TMPFILE1
mv -f $TMPFILE1 $LOGFILE
sed '1,/I: Starting COMPASS/d' $LOGFILE >$TMPFILE1
#how to remove leading direction-0 lines?
#ti=`sed '15,$d' compass.log | grep "direction 0" | wc -l`
sed '/I\:\ COMPASS\ complete./,/\n/d' $TMPFILE1 | grep "\ d:\ " >$TMPFILE2
sed 's/...................\ d:\ //g' $TMPFILE2 >$TMPFILE1
sed 's/turn //g;s/ direction //g;s/ value //g' $TMPFILE1 >$TMPFILE2
valcount=0;
prevdir=1;
pi=$(echo "scale=10; 4*a(1)" | bc -l)
for i in `cat $TMPFILE2`;
 do
  IFS=",";
  valexportcount=0;
  for values in $i;
   do
    case $valexportcount in
     0)
       turn=$values;
       ;;
     1)
       direction=$values;
       ;;
```

```
     2)
       value=$values;
       ;;
     *)
       echo "I don't understand this logfile: $LOGFILE";
       exit -1;
       ;;
    esac;
    valexportcount=$(($valexportcount + 1));
   done
  IFS=$OLDIFS;
# using perl seems to be somewhat useless...
#  turn=`echo "$i" | perl -e 'while(<>){split/,/;print@_[0]."\n";}'`;
#  direction=`echo "$i" | perl -e 'while(<>){split/,/;print@_[1]."\n";}'`;
#  value=`echo "$i" | perl -e 'while(<>){split/,/;print@_[2]."\n";}'`;
  echo "turn: $turn, direction: $direction, value: $value";
  if [ ! $prevdir -eq $direction ];
   then
    x1=$(($x1 + $STEPSIZE));
   fi

  if [ $valcount -eq 0 ];
   then
    x1=$XSTART;
    y1r=$YSTART;
   else
    if [ $direction -eq 1 ];
     then
      if [ $prevdir -eq $direction ];
       then
        y1r=$(($y1r + $STEPSIZE));
       fi
     else
      if [ $prevdir -eq $direction ];
       then
        y1r=$(($y1r - $STEPSIZE));
       fi
     fi
   fi

  if [ ! $prevdir -eq $direction ];
   then
    if [ $direction -eq 1 ];
     then
```

```
      y1r=$YSTART;
     fi
   fi

  y1=$((2000 - $y1r));

  value=$(echo "scale=10; ($value * $pi) / 180" | bc -l);
  x2=$(echo "scale=10; ((s ($value)) * 50)" | bc -l);
  y2=$(echo "scale=10; ((c ($value)) * 50)" | bc -l);

  echo "  -> $x1, $y1, $x2, $y2";
  echo "$x1 $y1 $x2 $y2" >>$DATFILE;
  valcount=$(($valcount + 1));
  prevdir=$direction;
 done

 rm -f $TMPFILE1 $TMPFILE2

 gnuplot compass.plot
```

compass.plot

```
set xlabel "mm"
set ylabel "mm"
set title "Grafische Darstellung des Magnetfeldes"
plot "compass.dat" w vec
pause -1 "<CR> to continue"
set terminal postscript
set output "compass.ps"
replot
pause -1 "<CR> to exit"
```