Die Khepera C++ Bibliothek

Thomas Pantzer, Februar 1997

Inhaltsverzeichnis

1 Einleitung

1.1 Was ist khepera

Khepera ist

1. der Name eines alten ägyptischen Gottes mit einem Kopf in Form eines Käfers (siehe *http://www.khepera.com/xpr/story.html*).

2. ein Miniaturroboter mit 8 Infrarot- und Umgebungslichtsensoren und zwei, von je einem Motor getriebenen, Rädern. Weiterhin kann man ihn mit einer 64-Pixel CCD-Kamera und/oder einem Greifer ausstatten. Dieser Roboter wird von der Firma **Applied AI Systems, Inc.** vertrieben. Genauere technische Daten kann man auch über deren Webseite *http://fox.nstn.ca/~aai/khepera.htm* erfahren.

3. der Name (m)einer C++Bibliothek zur Kommunikation und Steuerung des unter Punkt 2 genannten Miniaturroboters.

Der weitere Text behandelt ausschließlich um die im Punkt 3 genannte C++Bibliothek, deren Kernstück die Objektklasse *RobotContext* ist. Die Implementation der Fast Fourier Transformation wurde von Philip Van Baren vorgenommen. Die Routinen zur Behandlung der seriellen Schnittstelle entstammen dem Programm **sim** von Oliver Michel und wurden von M. von Holzen und L. Tettoni programmiert. Man beachte auch deren Copyrights in den entsprechenden Quelltexten und Headerdateien.

1.2 Motivation

Im Rahmen des Praktikums Autonome Roboter an der Universität Leipzig kam ich das erste Mal mit dem khepera-Roboter in Berührung. Da diese Roboter mit zusätzlichen Komponenten (Greifer und Kamera) ausgestattet waren, und noch keine Beispielsoftware für diese Komponenten aufzutreiben war, faßte ich den Entschluß, selbst ein Steuerprogramm für diesen Roboter zu schreiben. Die Bibliothek wurde später aus verschiedenen Modulen dieses Steuerprogramms zusammengesetzt.

2 Die Objektklassen

2.1 class RobotContext

Diese Struktur dient hauptsächlich dazu, die internen Statusvariablen im Kontroller des Khepera - Roboters im Hostrechner abzubilden. Die Methoden dieser Objektklasse sind zum Teil Funktionen die "hardwareseitig" im Roboter implementiert sind, und zum anderen sind es aus mehreren einzelnen Funktionen kombinierte Methoden oder Algorithmen zur Vorverarbeitung und Aufbereitung der gelieferten Daten.

2.1.1 Definition

```
class RobotContext {
   public:
      t_vertex_2d pos;              // Position
      double alpha;                 // Orientierung, Winkel
      double aspect, delta;         // polare Koordinaten, Winkel/Abstand
      char With_Camera,With_Gripper,socket_mode,used;
      t_world World;      // ''Eck''-daten des Koordinatensystems
```

```
// Statusvariablen,
   double gone;          // zurückgelegter Weg
   int arm_pos, gr_width, ohm;
   char catched;
   int speed_left,speed_right,darkness, motion_flags;
   int gpl,gpr;         // [g]lobal_[p]osition_[l]eft enthält Summe
                        // Radimpulszähler links/rechts
   int ID;              // Identifizierungsnummer
   int fd;              // filedescriptor für IO
   int SensorValue[2][8];  // Array für Sensordaten
   int SensorMax[3];       // maxima der versch.  Sensoren
   short int Image[256];   // Array für Kamera Pixel
   char device[256];       // Dateiname oder ( hostname + portnummer )

   void Init(char *);
   void Done(void);
   void Reset(void);
   void Turn(double degrees);
   void Move(double millimeter);
   void LookHome(void);
   void SetOrientation(double deg);
   void Lift(void);
   void WaitStopped(void);
   void Stop(void);
   void SolveCoords(void);
   void SetMotorSpeed(double left_speed, double right_speed);
   void SetMotorSpeed(int motor_num, double speed);
   double GetMotorSpeed(int motor_num);
// BASIC-Turret,  low level functions of the robot
   void SetWheelCounter(int left, int right);
   void TurnWheels(int left, int right);
   void ReadWheelCounter(int *left, int *right);
   void SetMotorSpeed(int left_speed, int right_speed);
   void GetMotorSpeed(int &left, int &right);
   double ReadSensor(int sensor_class, int sensor_num);
   void UpdateSensors(void);
   void SetSpeedProfile(int left_max, int right_max, int left_acc, int right_acc);
   void LED_Ctrl(int led_num, LED_Behavior);
   int GetMotionCtrlFlags(void);
// GRIP-Turret
   void SetArmPos(int);
   int ReadGripOhmMeter(void);
   char CatchedObject(void);
   int GrippedWidth(void);
   int GetArmPos(void);
   void Grip(void);
   void Release(void);
// VISION-Turret
   char ImageRead(void);
   float GetIntensity(void);
   int GetPixelMaxLight(void);
   int GetPixelMinLight(void);
   void SetScanningPeriod(int);
};
```

2.1.2 Erläuterung der Methoden

1. `void Init(char *);`
   stellt die Verbindung zum Roboter her und initialisiert alle Variablen auf null, der Übergabe wert ist eine Zeichenkette und wird als Dateiname einer Gerätedatei interpretiert. Die Zeichenkette muß mit Ziffern enden. Falls ein Leerzeichen in der Zeichenkette enthalten ist, wird der erste Teil der Zeichenkette als Hostname und der zweite Teil als Portnummer interpretiert. In diesem Fall wird versucht, anstelle einer Gerätedatei(serielle Schnittstelle) einen Socket zu öffnen. Damit besteht die Möglichkeit auch mit einem Robotersimulator zu arbeiten.

2. `void Done(void);`
   gibt allozierten Speicher wieder frei

3. `void Reset(void);`
   setzt alle Varablen auf null

4. `void Turn(double degrees);`
   dreht den Roboter um einen bestimmten Winkel

5. `void Move(double millimeter);`
   bewegt den Roboter vorwärts

6. `void SetOrientation(double deg);`
   dreht den Roboter in seine Ursprungslage und dann um den angegebenen Winkel

7. `void LookHome(void);`
   dreht den Roboter in Richtung seines Ursprungspunktes

8. `void Lift(void);`
   hebt den Greifer und stellt ihn auf ca. 78° ein

9. `void WaitStopped(void);`
   wartet bis sich kein Sensor des Roboters mehr verändert. Diese Funktion sollte nur mit sehr viel Vorsicht eingesetzt werden, da unter ungünstigen Umständen das ganze Program zum stehen kommen kann. Sie ist nur für kurze (einzeilige) Demoprogramme geeignet und gehöhrt nicht in eine echtzeitfähige Robotersteuerung.

10. `void Stop(void);`
    schaltet die Motoren des Roboters ab und ruft die Funktion *WaitStopped()* auf. diese Funktion sollte genau wie die unter Punkt 9 gemieden werden.

11. `void SolveCoords(void);`
    berechnet die X/Y-Koordinaten des Roboters nur mit Hilfe der Radumdrehungen. Diese Methode wird auch aus den Funktionen *Move()*, *Turn()* und *SetMotorSpeed()* aufgerufen.

12. `void SetWheelCounter(int left, int right);`
    setzt die Werte für den den linken und rechten Radzähler

13. `void TurnWheels(int left, int right);`
    die Räder werden in Bewegung gesetzt bis die Werte für den linken und rechten Radzähler mit den übergebenen Werten übereinstimmen. Die erreichbare Maximalgeschwindigkeit und Beschleunigung kann vorher mit der Funktion *SetSpeedProfile(...)* eingestellt werden.

14. `void ReadWheelCounter(int *left, int *right);`
Mit dieser Funktion können die Radzähler ausgelesen werden. Ein Wert von 600 entspricht einer Radumdrehung

15. `void SetMotorSpeed(int left_speed, int right_speed);`
Schaltet beide Motoren gleichzeitig an. Die Geschwindigkeit der Motoren kann links und rechts unterschiedlich sein. Das Wert kann zwischen -128 und 127 variieren. Ein Wert von 127 entspricht einem Meter pro Sekunde.

16. `void SetMotorSpeed(double left_speed, double right_speed);`
wie im Punkt 15 aber mit Werten zwischen -1.0 und +1.0 . Ein Wert von 1.0 entspricht einer Geschwindigkeit von 0.2 m/s. Dieser Wert kann zum Zeitpunkt der Kompilation anders festgelegt werden und wurde mit Absicht so gewählt, damit der Roboter nicht in einem unbeobachteten Moment vom Tisch fällt.

17. `void SetMotorSpeed(int motor_num, double speed);`
wie in Punkt 16 aber es wird nur ein Motor angesprochen. Der Index 0 steht für den linken, der Index 1 für den rechten Motor.

18. `double GetMotorSpeed(int motor_num);`
liest den aktuellen Geschwindigkeitswert eines Motors. Die Rückgabewerte sind in der selben Größenordnung wie die in der Funktion 16 übergebenen Werte.

19. `double ReadSensor(int sensor_class, int sensor_num);`
gibt den skalierten Wert eines Sensors zurück. Das Ergebnis ist immer im Bereich [0,1.0]. Ein Wert 1.0 entspricht dem Maximum aller bis dahin eingelesenen Werte. Als Argument wird die Art des Sensors (0 für Infrarot oder 1 für Umgebungslicht) und die Nummer bzw. Position des Sensors angegeben.

20. `void UpdateSensors(void);`
Diese Funktion liest nacheinander alle Infrarot- Entfernungssensoren und danach alle Umgebungslichtsensoren und speichert die Werte in einem Array ab. Mit der Funktion ReadSensor(..) können die einzelnen Werte abgefragt werden.

21. `void SetSpeedProfile(int left_max, int right_max,`
`int left_acc, int right_acc);`
Mit dieser Funktion kann der roboterinterne Positionierungskontroller konfiguriert werden. Die Einheiten sind $x * 8\frac{mm}{s}$ für die Geschwindigkeit und $x * 3.125\frac{mm}{s^2}$für die Beschleunigung. Der Roboter benutzt dieses Profil um die Motorgeschwindigkeit während der Ausführung der Funktion *TurnWheels(..)* zu regeln.

22. `void LED_Ctrl(int led_num, LED_Behavior);`
schaltet die LEDs an, aus oder um

23. `int GetMotionCtrlFlags(void);`
gibt die Statusvariablen des Bewgungskontrollers im Roboter zurück. Für jeden Radzähler gibt es die drei Zustände "am Ziel", "bewegt durch Geschwindigkeitsmodus" und "Positionierfehler". Die entsprechenden Bitmasken kann man der Headerdatei entnehmen.

24. `void SetArmPos(int degree);`
schwenkt dem Greifarm in die angegebene Position. Das Argument wird in Grad angegeben.

25. `int ReadGripOhmMeter(void);`
mißt den elektrischen Widerstand des Objektes in der Greifzange. Der Wert 255 entspricht dem eines Null-Ohm-Widerstandes, der Wert 50 entspricht ungefähr 450 kOhm. Eine Kennlinie kann man aus [?] entnehmen.

26. `char CatchedObject(void);`
gibt den Wert *true* zurück, wenn die Lichtschranke in der Greifzange durch ein Objekt unterbrochen ist

27. `int GrippedWidth(void);`
gibt an, wie weit die Greifzange geschlossen ist

28. `int GetArmPos(void);`

29. `void Grip(void);`
schließt die Greifzange

30. `void Release(void);`
öffnet die Greifzange

31. `char ReadImage(void);`
Diese Funktion gibt den Wert *true* zurück, wenn ein Bild von der CCD-Kamera komplett und korrekt eingelesen wurde. Die Pixelwerte befinden sich in dem Array *Image*. Dieses Array enthält 256 Felder obwohl die Kamera nur 64 Pixel einlesen kann, d.h. es können in dem Array 4 Bilder abgespeichert werden. Im ersten Bild (Indizes 0..63) werden die Kamerawerte abgespeichert. In der momentanen Implementation wird das zweite und dritte Bild (Indizes 64..127 und 128..191) benutzt, um die Ergebnisse einer Kantenfilterung bzw. einer Fourier-Transformation des ersten Bildes anzuzeigen. Das vierte Bild (Indizes 192..255) ist temporärer Speicher für die Bildverarbeitungsalgorithmen

32. `float GetIntensity(void);`
gibt die durchschnittliche Lichtintensität im Sichtbereich aus

33. `int GetPixelMaxLight(void);`
gibt die Nummer des Pixels mit der größten Lichtintensität aus

34. `int GetPixelMinLight(void);`
gibt die Nummer des Pixels mit der kleinsten Lichtintensität aus

35. `void SetScanningPeriod(int);`
Diese Funktion setzt das Intervall mit dem der Bildspeicher der Kamera aktualisiert wird. Die zulässigen Werte sind in der nachfolgenden Tabelle aufgelistet.

Eingabewert	Intervall in msec
0	20
1	50
2	100
3	150
4	250
5	500
6	1000
7	5000

2.2 class t_World

Dieses Objekt besitzt drei 2-dimensionale Vektoren in denen die maximalen(minimalen) auftretenden Werte der Koordinaten des Roboters abgespeichert werden. Der Vektor *min* markiert somit die linke untere Begrenzung des Koordinatensystems auf der internen Karte des Roboters, der Vektor *max* hält entsprechend die rechte obere Begrenzung. Erreicht der Roboter einen Punkt hinter diesen Begrenzungen werden die neuen Werte übernommen und die Variable *grow* auf den Wert *true* gesetzt. Im nächsten Schritt wird die Variable *grow* wieder auf *false* zurückgesetzt.

2.2.1 Definition

```
typedef struct { double x,y; } t_vertex_2d;
class t_world {
   public:
     t_vertex_2d max,min,size;
     char grow;
     void Update(t_vertex_2d);
     void Init(void);
   };
```

2.2.2 Erläuterung der Methoden

1. `void Init(void);`
   Die Methode Init setzt die Variablen min(max) auf besonders hohe(niedrige) Werte, damit beim allerersten Aufruf der Methode Update alle Variablen auf korrekte Werte gesetzt werden können.

2. `void Update(t_vertex_2d P);`
   Hier wird geprüft ob der übergebene Punkt P innerhalb des von *min* und *max* aufgespannten Rechtecks liegt. Ist das nicht der Fall, werden die Vektoren *min* bzw. *max* so verändert, daß die obengenannte Bedingung wieder erfüllt ist. Dabei wird die Variable *grow* auf den Wert *true* gesetzt. Zurückgesetzt wird sie erst, wenn der übergebene Punkt P von vornherein innerhalb des Rechtecks liegt.

3 Verschiedenes

3.1 Hilfsfunktionen

```
char FD_Ready(int fd);
void InitTimer(void);
long int GetTime(void);
void DoneTimer(void);
void usleep(long int µsec);
```

FD_Ready ist eine Funktion mit der man ohne Verzögerung prüfen kann, ob eine Eingabedatei Zeichen bereithält. Ist das nicht der Fall wird *false,* ansonsten wird *true* zurückgegeben. Als Argument wird der Dateideskriptor (Integerzahl) fuer die zu prüfende Datei erwartet. Es werden die Funktion *select()* sowie die Makros *FD_SET*, *FD_ISSET* und *FD_CLEAR* benutzt. Beispiel: *FD_Ready(0)* gibt aus, ob der Nutzer eine Taste gedrückt hatte. (ähnlich wie die aus Pascal bekannte Funktion *keypressed*), Der Wert 0 (Null) steht hier für den Dateideskriptor der Standardeingabe.

InitTimer initialisiert und startet einen Zeitzähler.

GetTime gibt die vom Start an verstrichene Zeit in Millisekunden zurück.

DoneTimer stellt den Zustand von vor dem Aufruf von *InitTimer()* wieder her.

usleep läßt den aktuellen Prozess für eine bestimmte Mindestzeit warten. Das Argument ist ein integer-Wert und gibt die Wartezeit in Mikrosekunden an. Da alle UNIX Betriebssysteme multiuser- und multitaskingfähig sind und über einen Scheduler verfügen, kann diese Routine auch zu einer längeren Wartezeit, als angegeben, führen. Diese Funktion steht nur auf manchen Plattformen nicht zur Verfuegung (z.B. HPUX) und ist durch eine Kompileranweisung noch ausgeschlossen.

3.2 Tips zur Installation

Zur Installation benötigt man die GNU-Programme *tar*(*gtar*), *install*(*ginstall*), *gcc* und *make*(*gmake*). Als erstes packt man das Archiv in einem dafür vorgesehenen Verzeichnis mit *tar* aus. Danach wählt man, der Architektur des Rechners entsprechend, ein Makefile aus (z.B. ln -s Makefile Makefile.sun). Leider habe ich an der Universität nur Zugang zu Rechnern mit 3 verschiedenen Plattformen, sodaß ich nur für diese Plattformen ein Makefile bereitstellen kann. Es sollte allerdings nicht schwer sein die Bibliothek auf eine andere Plattform zu portieren. Das Makefile enthält viele Variablen, sodaß nur an wenigen Stellen editiert werden muß. Die Pfadvariable PREFIX sollte allerdings in jedem Fall editiert werden.Für eine systemweite Installation setzt man sie i.A. auf *"/usr/local"*. Wenn man nicht über *root*-Rechte verfügt, bleibt nur noch die Installation im eigenen Homedirectory, die Variable PREFIX sollte dann diesen Wert enthalten.

Dynamische Bibliotheken lassen sich auch dann noch benutzen, wenn man sie nicht mit den Systemverwalterrechten installieren kann. Vor der Ausführung von Programmen die solche dynamischen Bibliotheken benutzen, muß dann allerdings die Umgebungsvariable LD_LIBRARY_PATH auf das Verzeichnis mit den Bibliotheken setzen. Im nachfolgenden Beispiel hat der Nutzer *tom* das Homedirectory */home/tom*. Seine dynamischen Bibliotheken hat er im Verzeichnis */home/tom/lib* abgespeichert und er benutzt die *bash* als Shell.

```
(sun):/home/tom/>  export LD_LIBRARY_PATH=/home/tom/lib [ENTER]

(sun):/home/tom/>  ldd demoprog [ENTER]

          libkhepera.so  =>        /home/tom/lib/libkhepera.so
          libm.so.1  =>     /usr/lib/libm.so.1
          libnsl.so.1  =>   /usr/lib/libnsl.so.1
          libsocket.so.1  =>        /usr/lib/libsocket.so.1
          libc.so.1  =>     /usr/lib/libc.so.1
          libdl.so.1  =>    /usr/lib/libdl.so.1
          libintl.so.1  =>  /usr/lib/libintl.so.1
          libmp.so.1  =>    /usr/lib/libmp.so.1
          libw.so.1  =>     /usr/lib/libw.so.1

(sun):/home/tom/>  demoprog [ENTER]
```

3.3 Ein kurzes Demo

Wenn die Bibliothek fertig kompiliert ist, kann das Demo mit der Kommandozeile

```
gcc -o demo -I./../include demo.cc -L./..  -lkhepera -lm
```

übersetzt werden. Gegebenenfalls müssen noch die Bibliotheken libsocket.a und libnsl.a mit dazugelinkt werden. Die Praktikumsteilnehmer im Milab können die Optionen `-I/home/public/kurs207/include` und `-L/home/public/kurs207/lib` benutzen, wenn sie an der Bibliothek selbst keine Änderungen vornehmen möchten.

```
#include "khepera.h"
int main(void)
{
// Variablendeklaration
   RobotContext robot;
// Initialisierung auf COM2 unter dem Betriebssystem Linux
   robot.Init("/dev/ttyS1");
// fahre 30 Zentimeter vorwärts
   robot.Move(300);
// senke den Greifarm
   robot.SetArmPos(-20);
// schließe den Greifer
   robot.Grip();
// hebe das Objekt an
   robot.Lift();
// drehe dich um 90 Grad nach rechts
   robot.Turn(-90);
// fahre einen Bogen vorwärts nach rechts
   robot.SetMotorSpeed((int)10,(int)8);
// lese den linken vorderen Entfernungssensor
   robot.UpdateSensors(300);
   double d = robot.ReadSensor(IRED,FRONT_LEFT);
...
   robot.Done();
}
```

Literatur

[1] Khepera User Manual, K-Team SA, 1995

[2] Khepera Gripper User Manual, K-Team SA, 1995.

[3] Khepera K213 Vision Turet User Manual, K-Team SA, 1995