B-BKSPP-PR (WS 2013/2014)
Einführung in die Systemprogrammierung mit C
Teil 3: Projekte

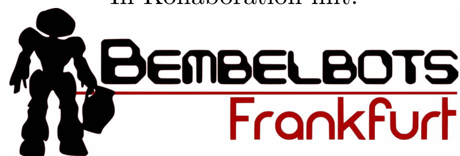
6. Dezember 2013

Im Folgenden sind mehrere Projekte aufgeführt, die Sie im letzten Teil des Praktikums durchführen können. Falls Sie noch kein eigenes Projektthema vereinbart haben, suchen Sie sich bitte für Ihre Gruppe aus den unteren Themen mehrere aus und senden Sie die Liste in absteigender Reihenfolge Ihrer Präferenz aus, also mit dem bevorzugten Projekt zuerst. Die Projektzuweisung erfolgt dann über einen randomisiert-fairen Zuweisungsmechanismus.

Sie haben weiterhin bis zur Deadline zur Einsendung von Präferenzen Zeit, ein eigenes Projekt zu vereinbaren[1].

Beachten Sie: Alle Projekte erfordern, daß Sie sich zusätzliches Wissen *selbstständig* aneignen, z.B. betreffs Programmschnittstellen. Bei Fragen könnnen Sie sich selbstverständlich an den Veranstalter richten.

In Kollaboration mit:



[1] Beachten Sie, daß *Vereinbarung* bedeutet, daß Sie Zustimmung erhalten haben; eine Einsendung eines Projektvorschlages ist nicht ausreichend.

Inhaltsverzeichnis

1 Projekt: Erweiterung des SPIM-Simulators für den MIPS-Prozessor **3**

2 Projekt: Automatische Optimierung von `malloc` **4**

3 Projekt: Optimierte nebenläufige Speicherabbildungen **5**

4 Projekt: C++: Ein Refactoring mit *Clang* **6**

5 Projekt: Eine kleine Datenbank **7**

6 Projekt: Ein optimierter Klassifizierungsmechanismus zum *machine learning* **9**

7 Projekt: Chat mit Hut **10**

8 Projekt: Vektorisiertes 2D-Shadow Mapping mit OpenGL **11**

9 Projekt: Eine Linux-Kernelerweiterung zur Sammlung der Netzwerkaktivität **15**

10 Projekt: Ein Linux-Kernelmodul für Zeitbegrenzung **16**

11 Projekt: Ein kleiner Interpreter **17**

11.1 Sprachgrammatik 17
11.2 Typen 19
11.3 Sprachsemantik 19
11.4 Erweiterungen 20
11.5 Validierung 20

12 Projekt: Bembelbots/C++: Deutender Roboter **21**

13 Projekt: Bembelbots/C++: Schnelle Bilderkennung **22**

1 Projekt: Erweiterung des SPIM-Simulators für den MIPS-Prozessor

Diese Aufgabe ist besonders geeignet für Studierende, die an der Veranstaltung B-SYSP teilgenommen haben.

Der SPIM-Simulator ist ein Programm, das den MIPS32-Prozessor simuliert. Wir verwenden dieses Programm in der Lehre als Teil der Veranstaltung "Einführung in die Systemprogrammierung". SPIM ist im Wesentlichen ein C-Programm.

Ihre Aufgabe in diesem Projekt ist es, eine spezielle Variante von SPIM, die wir für die Lehre verwenden, um drei Komponenten zu erweitern:

- Einen *Zyklenzähler*, der zählt, wieviele Programmzyklen vergangen sind, seit das Programm gestartet wurde
- Einen *Dynamischen Verzweigungsvorhersage-Simulator*, der die dynamische Verzweigungsvorhersage des MIPS32-Prozessors simuliert. Aufgabe dieser Komponente ist, zu berechnen, wieviele Zyklen eine Verzweigung kostet, abhängig von einer Verzweigungsvorhersagetabelle. Diese soll sowohl 1-Bit als auch 2-Bit-Sprungvorhersage implementieren.
- Einen *Cache-Simulator*, der Speicherzugriffskosten simuliert, indem er den Zyklenzähler bei Speicherzugriffen entsprechend erhöht. Dieser Simulator soll konfigurierbar sein:
  - Konfigurierbare Hauptspeicherzugriffskosten
  - Konfigurierbare Cache-Zugriffskosten
  - Konfigurierbare und erweiterbare Cache-Implementierungen (über geeignete Funktionszeiger o.ä.)
  - Folgende Cache-Implementierungen sollten Sie mitliefern, wobei als Cache-Größen nur Zweierpotenzen unterstützt werden müssen:
    - Direkt abgebildeter Cache
    - k-fach satzassoziativer Cache (Beliebige Zweierpotenzen für k)
    - Vollassoziativer Caches
- Vom Benutzer aktivierbare Textausgabe zu allen eingebauten Komponenten, die Aktualisierungen des Zyklenzählers, der Verzweigungsvorhersagetabelle, und des Caches an die Standardausgabe ausschreibt.

2 Projekt: Automatische Optimierung von `malloc`

Die Linux-Implementierung von `malloc()`, `free()`, und deren Freunde erlaubt es Entwicklern, diese Funktionen zu beobachten und durch andere `malloc`-Implemntierungen auszutauschen. Da größere Programme `malloc` oft sehr aggressiv verwenden, kann das Austauschen dieser Operationen einen großen Performanzunterschied bewirken, da die (de)allozierung von Ablagespeicher in C (im Gegensatz zu automatisch speicherverwalteten Sprachen wie Java) intrinsisch recht teuer ist.

In diesem Projekt entwickeln Sie eine alternative `malloc`-Implementierung, die das `malloc`/`free`-Verhalten ausgewählter Programme vorhersagt[2]. Ihre Implementierung kann dadurch `malloc` beschleunigen. Wenn Ihre Implementierung z.B. lernt, daß das Programm

```
node_t *alloc_node()
{
  node_t *n = malloc(sizeof(node_t));
  n->data = malloc(20);
}

void free_node(node_t *node)
{
  free(node->data);
  free(node);
}
```

immer zwei Daten zusammen alloziert und dealloziert, kann es diese in einem Block allozieren und freigeben.

Ihr Programm besteht also aus drei Komponenten:

1. Eine *Beobachtungskomponente*, die Programmdurchläufe beobachtet und sich die aufgerufene Funktion (`malloc`, `calloc`, ...) zusammen mit der Menge an beantragtem Speicher und dem Programmzähler des Aufrufers (aus dem Aufrufstapel extrahiert) merkt und diese als *Programmdurchläufe* ausschreibt.

2. Eine *Auswertungskomponente*, die in mehreren Programmdurchläufen nach Mustern sucht, die Blockallozierung oder Vorallozierung erlauben (dieser Teil muß nicht in C implementiert werden.)

   Die genauen Muster, die Sie erkennen können sind Ihnen überlassen, ebenso die exakten Mechanismen zur Erkennung.

3. Eine *Ausführungskomponente*, die die gelernten Muster umsetzt und somit alternative `malloc`/`free`-Implementierungen anbietet.

Ihr System sollte seinen Nutzen dadurch demonstrieren, daß es mindestens zwei unterschiedliche Programme meßbar beschleunigt.

[2] Sie können dazu **__malloc_hook(3)** verwenden.

3 Projekt: Optimierte nebenläufige Speicherabbildungen

Dieses Projekt ist besonders geeignet für Studierende, die an Veranstaltungen zur parallelen Programmierung oder an B-SYSP teilgenommen haben.

Ein großer Vorteil der von Ihnen in Teil 1b implementierten Hashtabellen mit offener Adressierung ist, daß sie für viele parallele Algorithmen sehr angenehme Eigenschaften haben. In diesem Projekt erweitern Sie diese Tabellen so, daß sie zum Einen besser skalieren und zum Anderen in der Gegenwart von nebenläufigen Prozessen effizient funktionieren.

- Stellen Sie Ihre Implementierung so um, daß Sie korrekt mit nebenläufigen Zugriffen umgehen kann. Verwenden Sie dazu idealerweise *compare-and-swap*-Operationen (`cmpxchg`). (Ihre Implementierung muß nur auf neueren 64-Bit-Intel-Systemen korrekt funktionieren.)

- Behandeln Sie das Vergrößern der Tabelle auch im nebenläufigen Fall. Dazu merkt sich die Tabelle, wieviele Threads gerade auf sie zugreifen (per `cmpxchg` aktualisiert). Der vergrößernde Thread wartet, bis diese Zahl gleich 1 ist und setzt sie dann auf einen designierten Wert, der andere Threads dazu zwingt, auf den Abschluß der Vergrößerung zu warten.

- Verwenden Sie einen besseren Hash-Mechanismus als bisher. Die offene Adressierung (‘open addressing’) hat bei einer einfachen Implementierung das Problem, daß eine Tabelle mit Einträgen mit mehreren nahe beieinanderliegenden Hashcodes oft auch bei mehrfacher Vergrößerung die Daten nicht halten kann.

  Diese Art von Tabelle ist zwar prinzipiell nicht in der Lage, mit ‘worst case’-Fällen umzugehen (Hashfunktionen, die alles auf auf einen konstanten Wert ‘hashen’), aber in weniger dramatischen Fällen ist Besserung möglich, z.B. durch dynamische Erhöhung des open access-Intervalls, unbeschränkte Intervallgröße, und re-hashing.

- Erlauben Sie das Löschen von Einträgen.

- Konstruieren Sie eine nebenläufige Testsuite, die demonstriert, daß Ihre Tabelle korrekt funktioniert. Da das Scheduling von nebenläufigen Prozessen nichtdeterministisch ist, ist ein rein testbasierter Nachweis nicht vollständig möglich; hierbei handelt es sich also um eine ‘best effort’-Aufgabe.

4 Projekt: C++: Ein Refactoring mit *Clang*

Diese Aufgabe ist besonders für Studierende mit Erfahrung mit C++ und Interesse an Softwarewerkzeugen geeignet.

Verwenden Sie das C++-Compiler-Frontend Clang[3] um eine verhaltenserhaltende Programmtransformation (*refactoring*) zu implementieren. Ihr Refactoring soll anhand einer Position im Programm (z.B. 'das 2000ste Zeichen im Programm') eine C++-Wert-Template finden und expandieren. Beispielsweise soll bei Definition von

```
template <int v>
int add(int c)
{
  return v + c;
}
```

und Verwendung von

```
  int i = add<3>(2);
```

das Programmstück

```
int add__3(int c)
{
  return 3 + c;
}
```

und an den Aufrufstellen

```
  int i = add__3(2);
```

generiert werden können[4].

Dabei müssen nicht alle Sprachkonstrukte von C++ korrekt behandelt werden, aber es sollte klar definiert sein, welche Sprachkonstrukte wo erlaubt sind. Die Transformation sollte auch auf einfache Klassen und Methoden anwendbar sein. Verwenden Sie dazu die Clang-Namensanalyse und das existierende Rahmenwerk zu Refactoring in Clang[5].

[3] http://clang.llvm.org/
[4] Diese Transformation ist eine spezielle Form von *inlining*.
[5] http://clang.llvm.org/doxygen/Refactoring_8h.html

5 Projekt: Eine kleine Datenbank

Bauen Sie den Backend-Teil einer kleinen Datenbank. Ihre Implementierung soll sich auf die Repräsentierung von Datenbanktabellen in B+-Bäumen und die Ausführung einer einfachen Abfragesprache beschränken.

Eine Datenbanktabelle ist dabei so definiert, daß sie eine *Typspezifikation* mit einer unbegrenzten Anzahl von *Einträgen* ist, die der Typspezifikation entsprechen. Eine Typspezifikation ist eine Folge mit Länge k und k Paaren $\langle n_k, \tau_k \rangle$, wobei n_k der *Name* einer Tabellenspalte und τ_k der *Typ* der Tabellenspalte ist. Beispiel Tablle t_0:

Vorname : `string`	**Nachname** : `string`	**Alter** : `int`
"Hans"	"Wurst"	17
"Klara"	"Wurst"	15
"Elise"	"Kartoffel"	23

Die Typspezifikation dieser Tabelle gibt uns in diesem Fall drei Namen für die Spalten der Tabelle („Vorname“, „Nachname“, und „Alter“), wobei die ersten beiden Spalten den Typ `string` (Zeichenkette) und die letzte Spalte den Typ `int` (Zahl) haben. Ihre Implementierung muß nur die Typen string und int unterstützen.

Ihre Aufgabe besteht also aus drei Teilen:

- Implementierung von B+-Bäumen: Erzeugung, Eintrag (mit Schlüssel), und Löschen von Einträgen.
- Implementierung von Tabellen, die nichts anderes als B+-Bäume plus Typspezifikation für die Tabelleneinträge sind. Die erste Spalte (Index 0) dient dabei als der Schlüssel in die B+-Bäume.
- Implementierung eines Interpreters für eine Abfragesprache. Gehen Sie dabei davon aus, daß die Sprache bereits in der Form von C-Daten vorliegt (also nicht aus Zeichenketten eingelesen, analysiert, oder optimiert werden muß). Sie können Sich die Repräsentierung der Sprache so heraussuchen, daß sie Ihren Arbeitsaufwand minimiert.

  Eine Abfrage in dieser Sprache besteht aus einer Zahl `#v` (*Anzahl der Variablen*) und einer Folge von Operationen. Die Operationen können sich auf Variablen v_i beziehen, wobei $0 \leq i <$ `#v`.

  Jede Operation kann beliebig viele Ergebnisse produzieren, daher müssen die Operationen in rekursiv verschachtelten Schleifen implementiert werden. Ein ‘Ergebnis’ bedeutet dabei, daß die Operation sich selbst als erfolgreich betrachtet; oft (aber nicht immer) wird dabei auch ein Wert in eine oder mehrere Variablen geschrieben.

  Die Operationen sind:

  1. STRING s, v_i: Setzt Variable v_i auf Zeichenkette s. Hat immer ein Ergebnis und ist immer erfolgreich.
  2. INT n, v_i: Setzt Variable v_i auf Zahl n. Hat immer ein Ergebnis und ist immer erfolgreich.
  3. <=v_i, v_j: Vergleicht v_i mit v_j. Beide müssen `int`-Werte gespeichert haben. Hat maximal ein Ergebnis und schlägt fehl (hat kein Ergebnis) gdw $v_i > v_j$.
  4. LIES $t_i\langle v_{l_0}, \ldots, v_{l_k}\rangle$: Diese Operation liest alle Spalten aus einer Tabelle t_i und schreibt diese in Variablen (oder vergleicht, s.u.). Die Anzahl der Parameter-Variablen muß also der Anzahl der Spalten der Tabelle entsprechen.

     Jeder Variablenparameter v_{l_i} kann für diese Abfrage eine *Vergleichsmarkierung* $m(v_{l_i})$ haben. Wenn diese Vergleichsmarkierung gesetzt ist, wird die Spalte nicht in die Variabe geschrieben, sondern stattdessen mit deren Inhalt verglichen. Wenn der Vergleich fehlschlägt, wird der aktuelle Tabelleneintrag und die Operation schlägt für diesen Tabelleneintrag fehl. Sie kann aber bei den folgenden Tabelleneinträgen wieder erfolgreich sein.

5. LIES_DIREKT $t_i\langle v_{l_0}, \ldots, v_{l_k}\rangle$: Ähnlich LIES, aber $m(v_{l_0})$ wird fest angenommen. Statt des Durchlaufs durch alle Tabelleneinträge wird direkt nach dem in Variable v_{l_0} gesetzten Primärschlüssel gesucht.
6. AUSGABE$\langle v_{l_0}, \ldots, v_{l_k}\rangle$: Normalerweise der letzte Befehl, aber kann auch zum Debuggen verwendet werden. Dieser Befehl gibt die Inhalte der angegebenen Variablen aus.

Beispiel:

STRING "Wurst", v_1
INT 16, v_3
LIES t_0, $v_0, m(v_1), v_2$
<= v_2, v_3
AUSGABE v_0, v_2

Dieses programm setzt zunächst (in den ersten beiden Befehlen) Variablen v_1 und v_3 auf die Werte "Wurst" und 16. Dann schreitet es durch Tabelle t_0. Jedes Mal, wenn es einen Eintrag findet, der die Zeichenkette „Wurst“ (aus Variable v_1) an zweiter Stelle („Nachname“) hält, schreitet es erfolgeich weter– in Falle unseres Beispiels also zwei Mal, für „Hans“ und „Klara“. Die anderen Einträge werden übersprungen. Für jeden dieser Einträge vergleicht das Programm nun v_2 („Alter“) mit dem in v_3 gespeicherten Wert 16. Im Erfolgsfall gibt es dann v_0 und v_2 aus. Hier haben wir also nur einen Erfolgsfall– es wird ausgegeben:

```
Klara 15
```

Wenn „Hans“ ein Alter von 14 hätte, wären zwei Zeilen ausgegeben worden:

```
Hans 14
Klara 15
```

Beachten Sie, daß es normalerweise effizienter ist, rekursiv durch die Liste der Operationen zu schreiten, als für jede Operation eine Menge aller Zwischenergebnisse zu erzeugen.

6 Projekt: Ein optimierter Klassifizierungsmechanismus zum *machine learning*

Diese Aufgabe ist besonders geeignet für Studierende, die Interesse an den Methoden der künstlichen Intelligenz haben.

Support Vector Machines (SVMs) sind eine Klasse von Algorithmen, die anhand einer Menge von positiven und negativen Daten eine mathematische Funktion 'lernen', die neue Daten positiv oder negativ klassifiziert. Eine SVM wird also in zwei Phasen verwendet:

1. Lernen, anhand von positiven und negativen Datensätzen, und
2. Klassifizieren neuer Datensätze, von denen wir vorher nicht wissen, ob sie positiv oder negativ sind.

Implementieren Sie den Machine Learning-Algorithmus *HULLER*[6].

Vergleichen Sie die Effizienz Ihres Algorithmus mit einem Algorithmus der Bibliothek libsvm[7]. Die Bibliothek ist mit mehreren Benchmark-Datensätzen ausgestattet, die Sie unmittelbar zum Vergleich verwenden können. Dabei können Sie zwei Formen der Performanz vergleichen:

- Laufzeit-Performanz (wie lange dauert das Lernen bzw. die Klassifizierung?)
- Klassifizierungs-Performanz (wie oft wird korrekt klassifiziert?)

Untersuchen Sie hier gezielt die Laufzeit-Performanz Ihrer Implementierung mit den vorgestellten Mitteln und versuchen Sie, eine ähnliche oder bessere Laufzeit wie die von libsvm zu erreichen. Sie können ggf. Klassifizierungs-Performanz für Laufzeit-Performanz opfern.

[6] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.105.8314
[7] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

7 Projekt: Chat mit Hut

Diese Aufgabe ist besonders geeignet für Studierende, die Interesse an Netzwerkprogrammierung haben.

SDL[8] ist eine Bibliothek zur Darstellung von Multimedia-Informationen (Video, Audio, ...) auf verschiedenen Plattformen. In diesem Projekt verwenden Sie SDL zur Erweiterung des Chat-Servers aus Praktikumsteil 2.

Speziell sind Ihre Aufgaben wie folgt:

- Erweitern Sie den Chat-client so, daß eingeloggte Benutzer in einem neuen Fenster einen graphischen *avatar* erhalten, den der Benutzer gezielt „über den Bildschirm laufen lassen“ kann. Dabei soll die Bewegung des Avatars animiert sein.

  Eingabe und Empfang von Nachrichten soll ebenfalls in dem neuen graphischen Fenster durchgeführt werden.

- Erweitern Sie den Server so, daß er die Position und Bewegungen aller Chatteilnehmer an alle Klienten weiterleitet, so daß jeder sehen kann, wo der Avatar jedes anderen steht.

  Dabei soll jeder Avatar sich visuell von anderen solchen unterscheiden. Verwenden Sie verschiedene optische Effekte zur Abgrenzung (z.B. verschiedene Hüte).

- Bei Verbindungen über große Distanzen gehen oft Netzwerkpakete verloren. TCP sendet in diesem Fall die verlorenen Pakete automatisch neu und verzögert dabei die Versendung nachfolgender Pakete. Das kann ineffizient sein: wenn der Avatar bereits eine neue Position erreicht hat, können die unterwegs verlorenen alten Positionen getrost ignoriert werden.

  Verwenden Sie statt TCP zur Informationsübertragung das UDP-Protokoll. Beachten Sie dabei, daß UDP keine Garantien zur Reihenfolge des Empfangs oder zum tatsächlich erfolgten Empfang macht. Behandeln Sie dies wie folgt:

  - Senden Sie eine ständig erhöhende Laufvariable mit, so daß der Server alte Daten leicht erkennen und verwerfen kann.
  - Senden Sie für Textnachrichten (die nie gehen werden dürfen) Bestätigungen vom Server an den Klienten, so daß der Klient nicht empfangene Nachrichten bemerken und neu senden kann.

Sie müssen für dieses Projekt keine eigenen Graphiken erstellen. Sie können sich stattdessen auf existierende Graphikquellen beziehen:

- `https://github.com/themanaworld/tmwa-client-data/blob/master/graphics/sprites/`
- `http://opengameart.org`

[8] http://www.libsdl.org

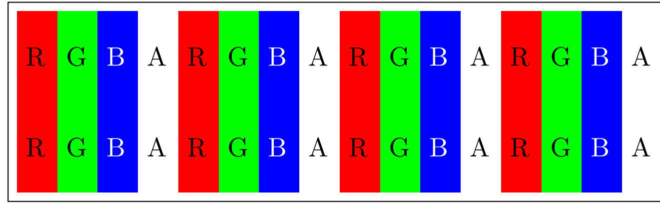


Abbildung 1: Komponenten eines kleinen Bildes mit 4x2 Bildpunkten

8 Projekt: Vektorisiertes 2D-Shadow Mapping mit OpenGL

Diese Aufgabe ist besonders geeignet für Studierende, die Interesse an Bilddarstellung mit OpenGL und vektorisierter Berechnung (optional mit OpenCL) haben.

OpenGL, die *Open Graphics Library*, ist eine Schnittstelle zur Darstellung zweidimensionaler und dreidimensionaler Szenen aus abstrakten geometrischen Beschreibungen wie z.B. Polygonen im zwei- oder dreidimensionalen Raum. Eine der Fähigkeiten von OpenGL ist die Darstellung von sogenannten *Texturen*, die Polygone mit zweidimensionalen Bitmap-Bildern wie eine Haut überziehen.

Dabei ist jeder Bildpunkt (Pixel) in der Textur mit den Attributen R (rot), G, (grün), B (blau) und A (alpha) versehen (Abbildung 1), wobei *alpha* ausdrückt, wie durchsichtig die Textur ist. Wir werden diesen Wert hier im Wesentlichen ignorieren. Um so höher ein R, G, oder B-Wert ist, um so mehr rot, grün, oder blau ist in dem entsprechenden Bildpunkt sichtbar. Wenn R, G, B auf dem Maximum sind, erkennen wir einen solchen Bildpunkt als weiß; wenn die Werte auf 0 sind, erkennen wir den Bildpunkt als schwarz. Wir nehmen die Abbildung als ungefähr linear wahr, so daß unser Auge z.B. die gleichmäßige Multiplikation von R, G, B mit einem Wert kleiner 1 (und größer oder gleich 0) als Abdunklung des Farbwertes wahrnimmt.

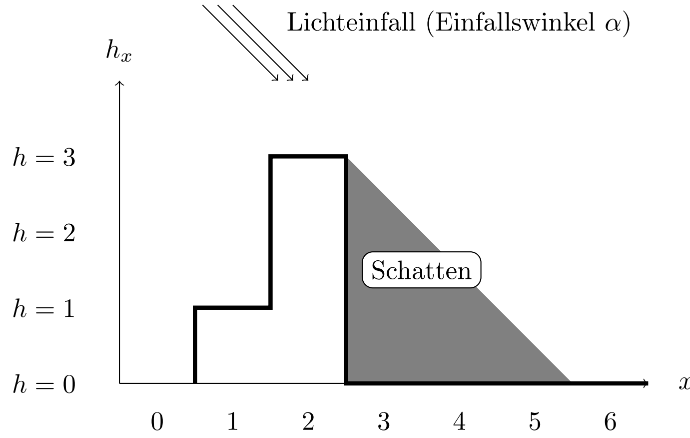
Wenn OpenGL im zweidimensionalen Raum eingesetzt wird, können die Texturabbildungsfunktionen von OpenGL sehr effektiv zum Skalieren (vergrößern/verkleinern) und Rotieren von zweidimensionalen Bildern eingesetzt werden.

Wir fügen nun einen Zwischenschritt in die OpenGL-Darstellung ein, indem wir Licht und Schatten auf die Textur werfen.

Die zugrundeliegende Idee ist, daß wir jedem Bildpunkt der 2D-Textur eine zusätzliche *Höhe* h zuordnen, die in einer *Höhenkarte* gespeichert wird. Wir simulieren nun Licht, das von der Seite in einem bestimmten Winkel auf die Textur einfällt. Wenn ein Bildpunkt komplett im Schatten liegt, soll er die Intensität $R{,}G{,}B = \langle 0, 0, 0\rangle$ haben, ansonsten eine Helligkeit, die der Menge an absorbiertem Licht entspricht.

Wir müssen also mit zwei neuen Konzepten arbeiten: Der Menge an absorbiertem Licht (λ) und der Höhe des einfallenden Schattens (η^s).

Betrachten wir dies anhand einer Reihe von nebeneinanderliegenden Bildpunkten mit verschiedenen Höhen:



Dieses Diagramm ist ein Querschnitt durch eine Textur mit Höhenkarte. Wir halten die y-Koordinate der Höhenkarte konstant (sie ist in dem Diagramm nicht ersichtlich). Die x-Achse der Höhenkarte wird auf der x-Achse des Diagrammes dargestellt, und die y-Achse des Diagrammes gibt die korrespondierende Höhe wieder.

Hier hat der Bildpunkt auf $x = 1$ die Höhe $h_1 = 1$, und der Bildpunkt auf $x = 2$ die Höhe $h_2 = 3$. Alle anderen Bildpunkte haben die Höhe 0. Wir lassen nun Licht von links auf das Bild einfallen. Dabei können wir zwei Effekte beobachten:

1. Da der Bildpunkt auf $x = 2$ weiter 'herausragt' als der Bildpunkt auf $x = 1$, kann er mehr Licht absorbieren.

2. Die Bildpunkte mit $x \in \{3, 4, 5\}$ sind im Schatten des Bildpunktes mit $x = 2$.

Wir simulieren zunächst den ersteren Effekt, indem wir dem Bildpunkt an Koordinate x einen *seitlichen Lichtabsorptionsfaktor* λ^a_x zuordnen:

$$\lambda^a_x = (h_x - h_{x-1})c_{\lambda^a}$$

wobei c_{λ^a} eine Konstante ist, die wir hier auf $\frac{\cos(\alpha)}{c}$ (Kosinus des Lichteinfallswinkels α geteilt durch eine Konstante c) festlegen. Dies kodiert zwei Intuitionen: (1) Desto flacher der Lichteinfall (kleineres α, höherer Kosinus), desto mehr Licht kann von der Seite absorbiert werden, und (2) Desto größer wir die Konfigurationskonstante c halten, desto detaillierter können wir die Helligkeitseffekte der Höhentexturen einstellen.

Um den zweiten Effekt, also Schatten, zu simulieren, verwenden wir ein Konzept, das wir *Schattenhöhe* nennen und als η^s notieren. Für eine bestimmte Koordinate x ist die Schattenhöhe η^s_x rekursiv definiert als

$$\eta^s_x = \max(h_{x-1}, \eta^s_{x-1}) - c_{\eta^s}$$

wobei $c_{\eta^s} = \frac{\sin(\alpha)}{c'}$ (Sinus des Lichteinfallswinkels α geteilt durch eine Konstante c'). Die Intuition ist dabei, daß ein höherer Lichteinfallswinkel α zu einem schnelleren Absinken des Schattens führt. Wir setzen $\eta^s_{-1} = -\infty$ (kein Schatten vor Beginn der Textur); in der Implementierung wählen wir einen hinreichend niedrigen negativen Wert, um den gleichen Effekt zu erzielen.

Wir können die Helligkeit λ_x eines Punktes nun aus diesen beiden Komponenten zusammensetzen:

$$\lambda_x = \begin{cases} 0 & \iff \eta^s_x > h_x \\ \lambda^a_x + \frac{\sin\alpha}{c''} & \iff \eta^s_x \leq h_x \end{cases}$$

Hierbei drückt $\frac{\sin\alpha}{c''}$ die Lichtabsorption von oben aus. Falls λ einen Wert größer 1 annehmen sollte, wird er automatisch auf 1 zurückgekürzt.

Die Implementierung der obigen Definitionen ist im einfachen hier angegebenen Fall (Lichteinfall von links, feste y-Koordinate) recht einfach: wir müssen nur von links nach rechts durch die Bildpunkte laufen, und uns dabei die Schattenhöhe η^s merken. Bei jedem Bildpunkt berechnen wir die lokale Helligkeit λ_x wie oben angegeben und aktualisieren η^s.

Eine Implementierung dieser Idee in voller Allgemeinheit kann in drei Schritten erfolgen:

1. Lichteinfall immer 'von links' aus Sicht der Textur, also über die x-Achse. Implementierung mit zwei verschachtelten Schleifen (eine für y, eine für x) mit genau einer Schattenhöhenvariable. Diese Variante ermöglicht einen schnellen Start und einfaches Testen. Der Lichteinfallwinkel α soll beliebig sein.

2. Vektorisierte Implementierung. Hierbei werden die Schattenhöhen mehrerer Bildpunkte in einem Array als *Schattenfront* gespeichert. Die Schattenfront können wir uns also als eine vertikale Linie vorstellen, die von links nach rechts über die Textur wandert. Dazu müssen wir die Verschachtelung der Iteration umkehren.

3. Lichteinfall mit beliebiger Rotation β. In dieser Variante gehen die Lichtstrahlen nicht mehr von links nach rechts über die Textur, sondern können auch von oben, unten, oder einem schrägen Winkel einfallen. Das heißt, daß die Schattenfront nun in einem beliebigen Winkel über die Textur wandern kann.

   Zur Vereinfachung verwenden wir folgenden Ansatz: Wir rotieren die Schattenfront um den Winkel β, lassen aber nun diese rotierte Schattenfront von oben nach unten, links nach rechts etc. über die Textur laufen. Da diese Durchlaufsform nicht zum Winkel β paßt, gleichen wir bei der Berechnung der neuen η^s-Werte aus: diese kopieren wir nun nicht mehr aus dem jeweils vorhergehenden η^s-Wert, sondern interpolieren zwischen diesem und einer der beiden Nachbar-Schattenhöhen. Sie können dabei mit Interpolierungsverfahren experimentieren, bis das Ergebnis Ihrer optischen Intuition entpsricht.

Sie haben zur Implementierung der Vektorisierung zwei Optionen:

1. Sie können die Berechnungen in vektorisiertem C-Code implementieren und dann in den Texturspeicher der Graphikkarte übertragen. In diesem Fall müssen Sie kein OpenCL verwenden.

2. Sie können die Berechnungen in OpenCL durchführen, wodurch sie ohne Datentransfer von der Grpahikkarte aus gelesen werden können.

OpenCL (mit 'C') ist eine hochgradig parallele Programmiersprache, die zur Programmierung von Graphikprozessoren (GPUs) verwendet wird. OpenCL basiert auf C99. OpenCL-Programme werden zur Laufzeit eines 'normalen' Programmes über bestimmte API-Funktionen übersetzt und in die Vektorprozessoren der Graphikkarte geladen, auf denen sie nebenläufig ausgeführt werden können.

Die Rechenergebnisse von OpenCL können als Texturen an OpenGL gesendet werden, über ein Feature namens cl_khr_gl_sharing[9].

Ressourcen:

- Einführung in OpenGL für 2D-Bilder:
  `http://www.ntu.edu.sg/home/ehchua/programming/opengl/CG_Introduction.html`

- Texturen in OpenGL:
  `http://www.opengl.org/wiki/Texture`
  `http://www.nullterminator.net/gltexture.html`

- Beispiel: OpenGL-Texturen aus OpenCL übertragen
  http://virtrev.blogspot.de/2010/08/ill-present-sample-that-create-opengl.html

[9] `http://www.khronos.org/registry/cl/sdk/1.2/docs/man/xhtml/cl_khr_gl_sharing.html`

- Einführung in OpenCL: http://www.drdobbs.com/parallel/a-gentle-introduction-to-opencl/231002854?pgno=1
  Phttp://www.desy.de/~mhohmann/gpu/gpu.pdf

9 Projekt: Eine Linux-Kernelerweiterung zur Sammlung der Netzwerkaktivität

Die sognenannte 'NSA-Affäre' hat in Erinnerung gerufen, daß die Ausspähung von Rechnern weiterhin ein sehr reales Problem darstellt. Eine Methode, um solche Ausspähung zu entdecken, ist, unerwartete Netzwerkkommunikation zu identifizieren (*anomaly detection*) und dem Benutzer zu melden. Idealerweise wird dies duch spezialisierte Hardware durchgeführt, die nicht oder nur schwer durch Software manipuliert werden kann.

In dieser Aufgabe implementieren Sie eine einfachere Variante dieses Konzeptes.

Schreiben Sie eine Linux-Kernelerweiterung, das mitprotokolliert, wieviele Datenpakete über das Netzwerk gesendet wurden. Verwenden Sie ein sysfs-Interface[10], um diese Informationen darzustellen (mit einem globalen Zähler).

Ihr System soll zusätzlich mitprotokollieren, wieviele dieser Pakete von welchem Prozeß geschrieben wurden, so daß wir die Prozeß-IDs (PIDs) von fragwürdigen Prozessen identifizieren können. Sendeoperationen, die ohne laufenden Prozeß direkt vom Kernel gestartet werden, sind besonders suspekt und sollten ebenfalls protokolliert werden.

Entwickeln Sie Ihre Lösung als eine Linux-Kernelerweiterung.

Ressourcen:

- `http://lxr.free-electrons.com/` (Nachschlageseite für Definitionen im Kernel)
- `http://wiki.openwrt.org/doc/networking/praxis` (Übersicht des Linux 2.6-Netzwerkstacks)

Zur Entwicklung wird Ihnen ein geeigneter Linux-Rechner zur Verfügung gestellt. Sie können u.U. schneller arbeiten, wenn Sie Zugang zu einem eigenen solchen Rechner haben.

Stellen Sie IMMER sicher, daß Sie eine alternative Möglichkeit haben, Ihren Rechner zu booten (ältere, funktionierende Kernel). Sichern Sie Backups Ihrer wichtigen Daten, bevor Sie mit dem Kernel experimentieren.

[10] `https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt`

10 Projekt: Ein Linux-Kernelmodul für Zeitbegrenzung

Erweitern Sie den Linux-Kernel so, daß ein spezieller Systemaufruf existiert, der einen Prozeß mit einer bestimmten Prozeß-ID automatisch nach Ablauf einer benutzerdefinierten Anzahl von Sekunden terminiert.

Sie können dabei auf Sicherheitskontrollen verzichten. Implementieren Sie den Systemaufruf als Linux-Blockgerät, das über einen ioctl (ein generisches Systemaufruf-Interface) den gewünschten Effekt verursachen kann. Der ioctl soll als Parameter die Prozeß-ID (PID) des zu terminierenden Prozesses und die Anzahl an Sekunden, nach denen terminiert wird, nehmen. Diese Zeitbegrenzung gilt als 'Sekunden seit Aufruf des ioctls' und ist unabhängig davon, wieviel Zeit der Prozeß tatsächlich gelaufen ist.

Ihre Implementierung soll mindestens vier solche Terminierungen gleichzeitig laufen lassen können. Wenn der laufende Prozeß bereits terminiert wurde, soll Ihr Programm den betreffenden Terminierungseintrag automatisch entfernen.

Entwickeln Sie Ihre Lösung als ein Linux-Kernelmodul[11]. Geben Sie ein kleines C-Programm an, mit dem ein normaler Benutzer mit Zugriffsrechten auf Ihr Blockgerät das Interface verwenden kann.

Ressourcen:

- `http://lxr.free-electrons.com/` (Nachschlageseite für Definitionen im Kernel)
- `https://www.kernel.org/doc/htmldocs/kernel-api/blkdev.html` (API für Blockgeräte im Kernel)

Stellen Sie IMMER sicher, daß Sie eine alternative Möglichkeit haben, Ihren Rechner zu booten (ältere, funktionierende Kernel). Sichern Sie Backups Ihrer wichtigen Daten, bevor Sie mit dem Kernel experimentieren.

[11] http://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html

11 Projekt: Ein kleiner Interpreter

Diese Aufgabe ist besonders geeignet für Studierende, die Interesse an Programmiersprachen und Übersetzern haben. Kenntnisse über kontextfreie Grammatiken sind hierbei sehr hilfreich.

Bauen Sie einen Interpreter für eine kleine Programmiersprache. Ihr Programm soll in der Lage sein, Programme der gegebenen Sprache einzulesen und auszuführen.

Ihr Programm besteht aus drei Teilen:

- Lexer/Scanner/Tokeniser (lexikalische Analyse): Dieser Programmteil zerlegt Zeichenketten in eine Folge von *Tokens*, die die eingelesenen Objekte abstrakt repräsentieren. Tokens sind z.B. INT für Ganzzahlen, REAL für Fließkommazahlen, und ID für Bezeichner.

  Ihnen wird ein fertiger Lexer zur Verfügung gestellt, den Sie nur noch nach Bedarf erweitern müssen. Ihr Lexer wird dabei von dem Programm **flex(1)** automatisch aus einer abstrakten Spezifikation erzeugt.

- Parser: Dieser Programmteil kombiniert die Tokens der lexikalischen Analyse, um die Programmstruktur in einer Baumstruktur abzubilden. Schreiben Sie einen sogenannten *rekursiv absteigenden Parser* (recursive descent parser), um die Eingabesprache einzulesen[12].

  Sie können auch **bison(1)**, **yacc(1)**, oder **antlr** verwenden, wenn Sie mit diesen Werkzeugen vertraut sind.

  Die Ausgabe Ihres Parsers ist ein sogenannter *abstrakter Syntaxbaum* (abstract syntax tree, AST), also eine Baumstruktur, die das Programm repräsentiert. Dabei sind die Baumknoten Operationen und die Kinder Parameter für die Baumknoten. Ihr AST kann der Syntax der BNF-Spezifikation entsprechen, aber Sie können ihn auch nach Ihren Bedürfnissen vereinfachen oder erweitern.

- Interpreter: Dieser Programmteil führt den gegebenen abstrakten Syntaxbaum aus.

Die von Ihnen zu implementierende Sprache ist im Folgenden definiert.

11.1 Sprachgrammatik

Die Grammatik wird Ihnen in BNF-Form angegeben. BNF-Grammatiken sind kontextfreie Grammatiken, die aus einer Menge von Regeln bestehen. Jede Regel besteht aus einem Nichtterminal und mehreren Regelkörpern, Notation

```
Nichtterminal ::= Regelkoerper_1
                  ...
                  | Regelkoerper_n
                  ;
```

Ein Regelkörper wiederum besteht aus einer Folge der folgenden zwei Dinge:

- Nichtterminale, die rekursiv weitere Bedeutungen haben
- Terminale, die den Ausgaben des vorgegebenen Lexers entsprechen. Terminale wiederum fallen in zwei Klassen:
  - Schlüsselwörter wie `'while'` oder `'{'` (durch einfache Anführungszeichen markiert)
  - Nicht-Schlüsselwörter, wie z.B. Zahlen oder Zeichenketten, denen eine besondere Bedeutung zukommt.

Diese Regeln beschreiben die kontextfreie Grammatik der Sprache, und auch deren Struktur:

[12] `http://www.fh-wedel.de/~si/vorlesungen/cb/SyntaxAnalyse/RecursiveDescent.html`

```
Program ::= Block
          ;

Block ::=   /* kann leer sein */
        | Statement ';' Block
        ;

Statement ::=  ID ':=' Expression
            | 'while' Expression 'do' Statement
            | '{' Block '}'
            | 'print' Expression
            ;

Expression ::= IfExpr
             ;

IfExpr ::= 'if' Expression 'then' Expression 'else' Expression
         | CmpExpr
         ;

CmpExpr ::= AddExpr '<' AddExpr
          | AddExpr '<=' AddExpr
          | AddExpr '=' AddExpr
          | AddExpr
          ;

AddExpr ::= AddExpr '+' MulExpr
          | AddExpr '-' MulExpr
          | MulExpr
          ;

MulExpr ::= MulExpr '*' BaseExpr
          | MulExpr '/' BaseExpr
          | MulExpr '%' BaseExpr
          | BaseExpr
          ;

BaseEpxr ::= '(' Expression ')'
           | NUM
           | STRING
           | ID
           | 'true'
           | 'false'
           ;
```

Wir verwenden genau drei nicht-Schlüsselwörter:

- `ID`: Bezeichner (Variablen)
- `NUM`: Zahlen
- `STRING`: Zeichenketten

11.2 Typen

Die Sprache verwendet drei Typen:

- Boolean (mit den Werten true und false)
- Number (entweder eine IEEE-754-Fließkommazahl oder ein 64-Bit-Integerwert. Wenn ein Integer-Wert mit einer Fließkommazahl verknüpft wird, ist das Ergebnis eine Fließkommazahl.)
- String (eine Zeichenkette)

Jeder Wert hat genau einen dieser Typen. Die Typen werden allerdings *dynamisch* vergeben, so daß die gleiche Variable im Verlaufe des Programmes mehrere unterschiedliche Typen annehmen kann.

11.3 Sprachsemantik

Die Semantik der Sprache definiert sich durch die Semantik der einzelnen Sprachkonstrukte. Für Ausdrücke (`Expression`, `IfExpr`, `CmpExpr`, `AddExpr`, `MulExpr`, `BaseExpr`) ist die Semantik ein Wert in einem der Typen. Für Befehle (`Statement`, `Block`, `Program`) ist die Semantik eine Menge von Variablenbindungen, ggf. zusammen mit Seiteneffekten. Aus `Statement`-Sicht ist die Semantik eines Ausdruckes das Ergebnis einer *Auswertung* des Ausdruckes.

Zur Berechnung der Semantik benötigen wir eine *Umgebung*, die Variablen Werte zuweist. Wir können sie uns als partielle Abbildung von Bezeichnernamen (`ID`) auf Werte beliebigen Typs vorstellen. Aus der Umgebung können Variablen gelesen werden; die Zuweisungsoperation kann die Umgebung auch aktualisieren. Ein Versuch, eine nicht zugewiesene Variable zu lesen, ist ein Programmfehler.

Ausdrücke. Die Semantik eines `BaseExpr` ist ein entsprechender String, Number, oder Boolean-Wert. Im Falle einer `ID` ist die Semantik gleich der aktuellen Variablenbindung in der *Umgebung*, sofern eine Binding vorliegt, sonst ein Programmfehler.

Die Semantik einer `MulExpr` mit `'*'`, `'/'`, oder `'/'` ist das Ergebnis der Multiplikation, Division, oder Modulo-Berechnung der Operanden. Wenn die Operanden keinen Number-Typ haben, ist dies ein Programmfehler.

Die Semantik einer `AddExpr` mit `'+'` oder `'-'`, ist das Ergebnis der Addition oder Subtraktion der Operanden, sofern die Operanden den Typ Number haben. Für `'+'` mit zwei Werten vom Typ String ist die Semantik gleich dem Ergebnis der Stringkonkatenierung der beiden Operanden. In allen anderen Typkombinationen liegt ein Programmfehler vor.

Die Semantik einer `CmpExpr` mit `'<'` oder `'<='` ist nur für Operanden mit dem Typ Number definiert, als numerische Vergleich ('kleiner' bzw. 'kleiner oder gleich'). Alle anderen Fälle sind Programmfehler. Der `'='`-Fall ist für alle Typen definiert als exakte Wert-Gleichheit.

Die Semantik einer `IfExpr` der Form **if** e_0 **then** e_1 **else** e_2 definiert sich wie folgt: Wenn e_0 die Semantik true hat, ist die Semantik der `IfExpr` gleich der Semantik von e_1. Wenn e_0 die Semantik false hat, ist die Semantik der `IfExpr` gleich der Semantik von e_2. Ansonsten liegt ein Programmfehler vor.

Die Semantiken der `Statement`-Konstrukte sind wie folgt:

- `ID ':=' Expression`: Die angegebene Variable `ID` wird an das Ergebnis der Auswertung von `Expression` gebunden. Diese Variablenbindung ist in der Umgebung ab dem nächsten Befehl sichtbar.
- `'while' Expression 'do' Statement`: Der Ausdruck `Expression` wird ausgewertet. Wenn der Ausdruck false ist, hat das Konstrukt eine leere Semantik (keine Wirkung). Wenn der Ausdruck true ist, wird das innere `Statement` ausgeführt, gefolgt von einer neuerlichen Auswertung des äußeren `'while'`-Konstruktes (unbegrenzt oft, wenn nötig). Wenn der Ausdruck einen anderen Wert hat, liegt ein Programmfehler vor.

- `'print' Expression`: Der Ausdruck `Expression` wird ausgewertet und sein Ergebnis an die Standardausgabe ausgegeben.

Die Semantik eines `Block`s ist die nacheinanderfolgende Ausführung der enthaltenen `Statement`s.

Alle verbleibenden Regeln haben die Form 'N_0 `::=` N_1'. Die Semantik von N_0 ist in diesem Fall gleich der von N_1.

11.4 Erweiterungen

Sie dürfen die obige Sprache erweitern.

11.5 Validierung

Demonstrieren Sie als Teil des Projektes die Fähigkeit Ihres Interpreters, einfache Berechnungen (größter gemeinsamer Teiler, Fibonacci-Zahlen) durchführen zu können.

12 Projekt: Bembelbots/C++: Deutender Roboter

(In Zusammenarbeit mit dem Bembelbots-Team der Universität)

In diesem Projekt entwickeln Sie ein Frontend für einen Nao-Roboter des Bembelbots-Teams. Dazu lernen und verwenden Sie die von diesem Team entwickelten Schnittstellen. Um Schäden an der Roboter- Hardware zu vermeiden, entwickeln Sie Ihre Lösungen zuerst für einen Roboter-Simulator, bevor Sie diese dann am tatsächlichen Gerät ausprobieren.

Die zugrundeliegende Software-Plattform **JrlSoccer** verbindet zahlreiche Sensoren und Aktuatoren über *shared memory*, also einen Speicherbereich, auf den die verschiedenen Komponenten abwechelnd zugreifen können, um zu lesen oder zu schreiben. Das Auslesen und Setzen der Sensoren und Aktuatoren wird zur Verfügung gestellt, ebenso eine angemessene Dokumentation.

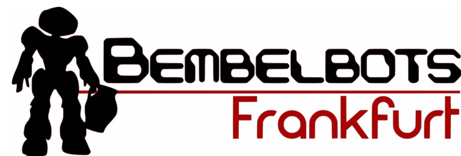
Ihre Aufgaben in diesem Projekt sind wie folgt:

1. Der humanoide Roboter Nao[13] verfügt über 21 Motoren zur Bewegungssteuerung. Jeder einzelne Motor lässt sich direkt ansteuern. Setzen Sie die Positionen der drei Motoren in einem der Arme derart, das dieser in eine festgelegte Richtung deutet.

2. Bereits vorhandene Bibliotheken zur Objekterkennung erlauben es, verschiedene Objekte auf dem Spielfeld zu detektieren. Basierend auf den aktuellen Motorstellungen kann die 2D-Position des Objektes im Bild genutzt werden, um die Entfernung zu einem Objekt auch ohne Stereo-Vision zu bestimmen (unter der Annahme, das sich das Objekt auf dem Boden befindet). Berechnen Sie die Position eines Objektes im Raum.

3. Korrigieren Sie die Roboterarm-Steuerung so, daß dieser immer in Richtung eines ausgewählten Objektes, z.B. eines Robocup-Balls, zu deuten versucht.

Sie können die genauen Ziele nach Absprache mit Herrn Fürtig (s.u.). Ihrer persönlichen Interessenslage anpassen.



Kontakt: Andreas Fürtig (andieh@bembelbots.de)

[13] Ein humanoider Roboter der Firma Aldebaran Robotics mit Sitz in Paris, Frankreich http://www.aldebaran-robotics.com

13 Projekt: Bembelbots/C++: Schnelle Bilderkennung

(In Zusammenarbeit mit dem Bembelbots-Team der Universität)

Eine der wichigsten Komponenten eines guten (robotischen) Fußballspielers ist schnelle Bilderkennung: Wo ist der Ball, wo das Tor, wo sind die Spielder des anderen Teams?

Die Nao-Roboter der Bembelbots verwenden zur Bilderkennung einen Algorithmus, der jedem eingelesenen Kamera-Bildpunkt eine Bedeutung gemäß einer Kategorie zuordnet. Die aktuelle Implementierung verwendet dazu eine komprimierte Nachschlagtabelle (*look-up table*, LUT), also ein Array im Speicher, das mit Farbwerten indiziert wird und Kategorien als Werte beinhaltet. Es gibt allerdings Anzeichen dafür, daß diese Implementierung keine ausreichend gute Performanz liefert.

In diesem Projekt analysieren Sie die Performanz der aktuellen Implementierung mit der erwähnten Nachschlagtabelle und entwickeln eine Strategie zur Optimierung dieses Problems.

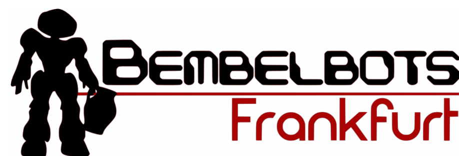
Der Algorithmus bearbeitet aktuell 30 Bilder pro Sekunde mit einer Auflösung von 640 mal 480 Bildpunkten. Die beiden Kameras des Roboters liefern jedoch 60 Bilder pro Sekunde. Eine erfolgreiche Durchführung der Aufgabe könnte die Reaktionsfähigkeit der Roboter also fast verdoppeln.

Die Bilder liegen im YUV Format vor, wobei die Werte jedes Kanals $0 - 255$ annehmen können. Die Nachschlagtabelle besteht demnach aus 256^3 Werten. Aktuell gibt es 6 Kategorien, die als Werte in der Tabelle abgespeichert werden. Weitere Details können bei Herrn Fürtig (s.u.) erfragt werden.

Zur erfolgreichen Teilnahme am Praktikum ist es nicht nötig, daß Ihre Optimierungsstrategie erfolgreich ist, aber Sie sollten plausibel erklären, warum die Strategie erfolgreich sein könnte.

Genauer sind Ihre Aufgaben wie folgt:

- Messen Sie mit PAPI die CPI (Prozessorzyklen pro Maschineninstruktion), Cache-Fehlzugriffsrate[14], und Sprungfehlvorhersagerate [15] des Prozessors bei der Ausführung mit Nachschlagtabelle.
- Untersuchen Sie, welche Teile der Nachschlagtabelle wie häufig benötigt werden, indem Sie für jeden Eintrag die Anzahl der Überprüfungen mitzählen. Untersuchen Sie diese Daten darauf, ob es bestimmte ‘dunkle Flecken’ (Zonen mit selten verwendeten Einträge) gibt.
- Schlagen Sie eine alternative Implementierung oder Repräsentierung der Nachschlagtabelle vor, und begründen Sie diese mit Ihrem Wissen der Systemprogrammierung und den Meßergebnissen.
- Untersuchen Sie die Performanz der alternativen Implementierung.



Kontakt: Andreas Fürtig (andieh@bembelbots.de)

[14] Foliensatz 3, Folien 3–34 (http://sepl.cs.uni-frankfurt.de/2013-ss/b-sysp/folien-03.pdf)
[15] Foliensatz 2, Folien 48–59 (http://sepl.cs.uni-frankfurt.de/2013-ss/b-sysp/folien-02.pdf)