Bachelorarbeit

Florian Brüll

Konzeptionierung und Implementierung eines interaktiven Bildmesssystems

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Florian Brüll

Konzeptionierung und Implementierung eines interaktiven Bildmesssystems

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. -Ing. Andreas Meisel
Zweitgutachter : Prof. Dr.Ing. Baran

Abgegeben am 18. Oktober 2007

Florian Brüll

Thema der Bachelorarbeit

Konzeptionierung und Implementierung eines interaktiven Bildmesssystems

Stichworte

LTI-Bibliothek, C++ MFC Projekt, Imaging Source, DLL, Shared Memory, Bildbearbeitung, MFC GUI, Kamera Steuerung

Kurzzusammenfassung

Um Bildaufnahme und Bildbearbeitung separat voneinander entwickeln zu können, ist es sinnvoll, ein einheitliches Programierkonzept zu entwerfen. Das in dieser Arbeit umgesetzte Konzept umfasst eine einfach zu handhabende Schnittstelle zwischen einer Kamera Applikation(GUI) und den Bildmessroutinen in Form einer Mess-DLL. Hierbei werden die aufgenommenen Bild der Kamera in einen gemeinsamen Speicherbereich geschrieben, worauf auch andere Komponenten zugreifen können. Auf diesen gemeinsamen Speicherbereich, der als „Shared Memory“ bezeichnet wird, können somit GUI als auch die als DLL geschriebenen LTI-Bildbearbeitungsroutinen Zugriff nehmen. Durch dieses Konzept ist eine unabhängige Bearbeitung von GUI und Bildbearbeitung gewährleistet. Diese Arbeit beschreibt die Arbeitsweise und Umsetzung des Shared Memory Konzeptes sowie das Zusammenspiel zwischen Oberflächenprogrammierung und DLL-Aufrufen.

Florian Bruell

Title of the paper

design and development of a concept for an interactive image measuring system

Keywords

LTI-library, C++, camera GUI, shared memory, image editing, C++, Imaging Source camera control

Abstract

to develop an image capturing software and and image editing program separately from each other, you have to find a way to communicate between these both processes. Therefore a so called "‘shared memory"’ pattern was used. In this particular memory it is possible that mutual processes on the same system have read or write access on the same part of memory. The image editing process was implemented with a dynamic link library (DLL). This paper is supposed to illustrate an impression how to deal with that subject.

Danksagung

An dieser Stelle möchte ich allen Danken, die direkt oder indirekt durch ihre Hilfe und Unterstützung zu dieser Bachelorarbeit beigetragen haben. Meinen besonderen Dank gilt Prof. Dr.-Ing. Andreas Meisel für die Betreuung dieser Arbeit. Durch seine freundliche Hilfsbereitschaft auf allen Bereichen des Themengebietes wurden viele aufkommende Fragen beantwortet und Probleme selbstständig gelöst werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Ziel der Arbeit

Im Rahmen dieser Bachelorarbeit soll ein Workframe für ein Bildmesssystem konzeptioniert und implementiert werden, bei dem eine einfache und einheitliche Kommunikationsplattform zwischen Kamera-Interface-Oberfläche (im weiterem Verlauf nur noch Kamera-GUI genannt) und Bildbearbeitungsroutinen (Mess-DLL) ermöglicht wird. Dieses Workframe soll als eine Art Handbuch zur Erstellung für ein zukünftiges Bildmesssystem dienen, bei dem Kamera-GUI und Messroutinen strikt getrennt voneinander arbeiten. Um dies zu realisieren, wird zur Steuerung der Kamera ein Visual Studio C++ Projekt verwendet. Dies beinhaltet eine GUI, in der Bilder aufgenommen und in einen gemeinsamen Speicherbereich abgelegt werden. Auf diesen gemeinsamen Speicherbereich können sowohl Kamera-GUI als auch Bildbearbeitungsroutinen Zugriff nehmen.

In vielen Anwendungsbereichen der Bildverarbeitung werden oft komplexe und zeitintensive Algorithmen eingesetzt. Um nicht für jeden neuen Algorithmus oder Filter das ganze Projekt umschreiben zu müssen, ist es sinnvoll, die Kamera GUI Steuerung von der eigentlichen Bildnachbearbeitung zu trennen. Somit erreicht man ein Höchstmaß an unabhängigen Komponenten, die jederzeit unabhängig voneinander erweitert und bearbeitet werden können, ohne dass andere Komponenten beeinflusst werden.

In dieser Bachelorarbeit werden folgende Themenbereiche erläutert:

- Ansteuerung einer Imaging Source Kamera (Bild-Grabbing) in einer C++ Anwendung
- Integration von LTI-Bibliotheks Methoden zur Bildbearbeitung
- Erstellung und Aufruf von einer Dynamic Link Library (DLL), zur Compile-und Laufzeit
- Nutzung eines gemeinsamen Speicherbereiches durch mehrere Prozesse mit Hilfe des Shared Memory Verfahrens

Im Detail wird beschrieben, wie die Kommunikation der verschiedensten Komponenten untereinander funktionieren und auf welche Art die verschiedenen Prozesse auf einen gemeinsamen Speicherbereich zugreifen können.
Im Gegensatz zu der Arbeit von (Harbili 2006) steht hier das Zusammenspiel der verschiedensten Betriebssystem-Mechanismen im Vordergrund, welche die Windows-API dem Benutzer zur Verfügung stellt.

1.2. Aufbau der Arbeit

Diese Bachelorarbeit umfasst folgende Kapitel:

Kapitel 2: Problemfeldvorstellung

In diesem Kapitel wird kurz erläutert, weshalb man überhaupt Bildmesssysteme einsetzt.

Kapitel 3: Realisierungsansätze und Lösungsvorschläge

Die verschiedensten Lösungsmöglichkeiten werden gegeneinander abgewogen und ausgewertet.

Kapitel 4: Design und Aufbau

Hier werden die angewandten Betriebssystem-Mechanismen anhand von Beispielen erklärt.

Kapitel 5: Konzept und Implementierung

Hier wird der Programmablauf, von der Bildaufnahme durch die Kamera bis zur Bearbeitung in der Mess-DLL ausführlich beschrieben.

Kapitel 6: Zusammenfassung, Bemerkungen und Ausblicke

Dieses Kapitel fasst die Bachelorarbeit nochmals kurz zusammen und gibt einen abschließenden Überblick und Fazit.

2. Vorstellung des Problemfeldes

2.1. Bedeutung in der industriellen Bildverarbeitung

Bildmesssysteme sind in der heutigen industriellen Produktion unerläßlich. Wie beispielsweise in ein Bericht des (Materialprüfungsamt-NRW 2006) zu entnehmen ist, werden vor allem kleine in großen Stückzahlen gefertigte Werkstücke durch optische Messkontrollen in wenigen Sekunden überprüft und selektiert.
Nach einen Bericht der (Gesellschaft 2007), wäre eine handverlesene Auswertung in der Produktion von Werkstücken, wie beispielsweise bei der Firma (CIM-Aachen 1998), welche Kettenräder, wie in Abbildung 2.1b herstellt, nicht möglich gewesen. Aus diesem Grund werden in vielen Industrieproduktionen Bildmesssysteme zur optischen Qualitätssicherung eingesetzt.

Eine Industriekamera, wie die in dieser Bachelorarbeit eingesetzte Imaging Source DMK 21F04 (2.1c) ist für diese Aufgabe Bestens geeignet. Mit ihrem hochwertigem Objektiv ist die zu erreichende Messgenauigkeit sehr hoch.

(a) Materialprüfer

(b) Kettenrad

(c) DMK 21F04 Kamera

Abbildung 2.1.: Messobjekte

Ziel dieser Arbeit ist es, mit Hilfe eines optischen Messsystems (s. Bild 2.3) millimetergenaue Vermessungen von Werkstücken vorzunehmen, die auf einen Glasträger (s. Bild 2.3c) liegen. Um die Messung von äußeren Faktoren unabhängig zu machen, ist der Messschrank mit Metallplatte verkleidet. Zur optimalen Beleuchtung dient ein Lichttisch (s. Bild 2.3d)von oben zur Ausleutung.
Die nachfolgenden Bilder zeigen die Planungsskizze des Messschrankes 2.2 und Bilder vom fertigem Messschrank Aufbau 2.3.

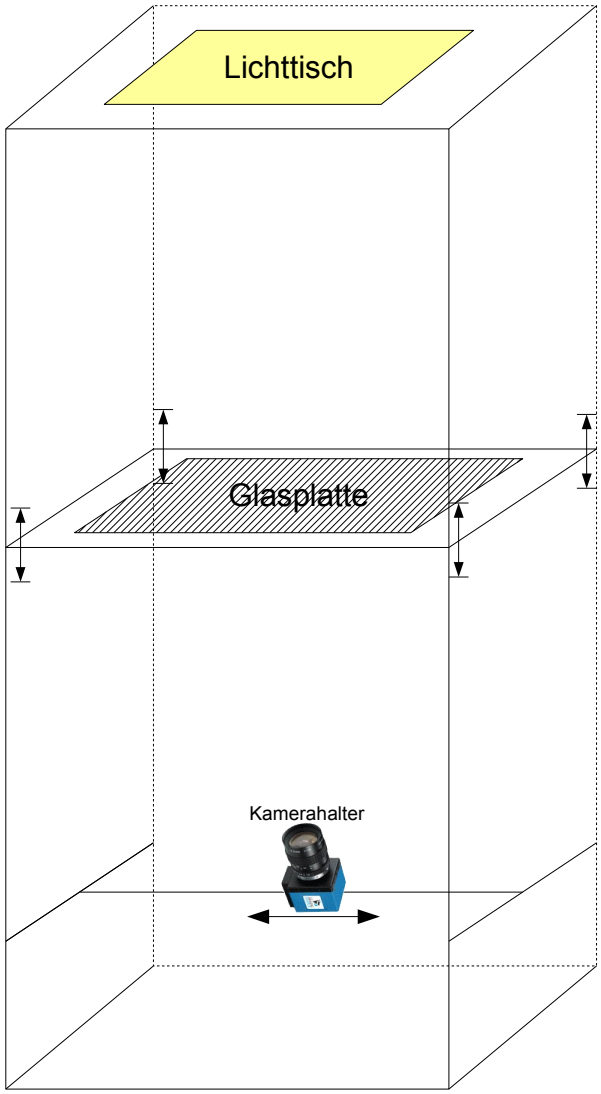


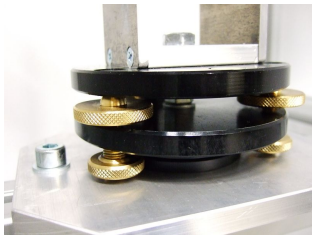
Abbildung 2.2.: Messschrank Skizze

(a) Messschrank Frontalansicht

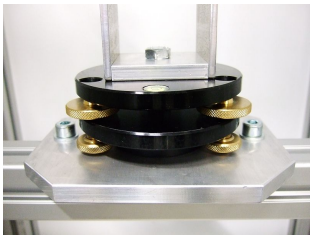
(b) Kamera Nahaufnahme

(c) Glasträgerplatte

(d) Lichttisch



(e) Kamerahalterung seitlich



(f) Kamerahalterung frontal

Abbildung 2.3.: Bilder vom Messsystem

2.2. Programmablaufübersicht

Der Ablauf von der Bildaufnahme bis zu Messverarbeitung besteht aus mehreren Einzelschritten, welche in den folgenden Kapiteln detailliert erklärt werden. Der Arbeitsablauf besteht aus folgenden Schritten:

1. **Aufnahme des zu vermessenden Werkstückes als Schwarz/Weiss Bild**
   Das zu vermessende Werkstück wird von der Messschrank Kamera in der GUI angezeigt.
2. **Abspeichern des aufgenommenes Bildes**
   Das Bild wird von der Kamera-Software in ein „Shared Memory“ Bereich abgelegt, worauf die Messroutine(der Mess-DLL) ebenfalls Zugriff haben.
3. **Aufrufen der DLL**
   Als nächster Schritt stößt der GUI Benutzer die Mess-DLL an, die auf das gespeicherte Bild im Shared Memory zugreift.
4. **Verarbeitung des Bildes in der Mess-DLL**
   Das geladene Bild wird durch die Mess-DLL verarbeitet (hier nur Negierung der Grauwerte) und in einem neuem Shared Memory Bereich abgelegt.
5. **Laden und Anzeigen des Bildes**
   Das von der Mess-DLL bearbeitete Bild wird in der GUI angezeigt und kann bei Bedarf auf der Festplatte abgespeichert werden.

siehe dazu auch Abbildung 2.4

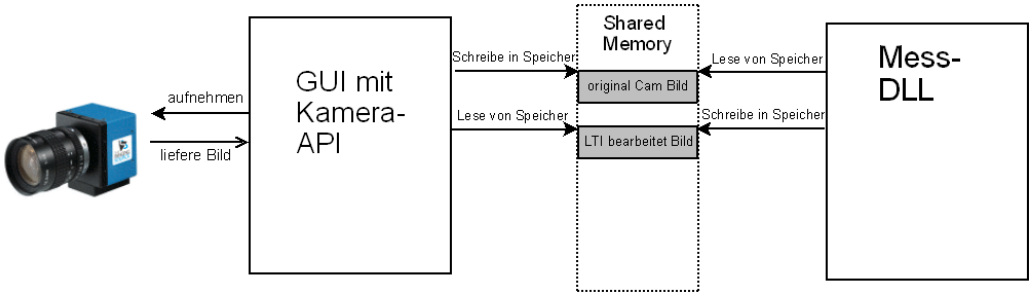


Abbildung 2.4.: Programmablaufsequenz

2.3. Funktionen der grafischen Benutzeröberfläche(GUI)

Die GUI dient zur Steuerung aller beteiligten Softwarekomponenten. Hier bestehen folgende Möglichkeiten:

- Kamera Optionen (Auflösung, Farbwerte, Kontraste) einstellen
- die Kamera in den live Video-Modus versetzen
- ein oder mehrere Bilder auf der Festplatte speichern
- das aktuell angezeigte Bild in den Bildpuffer schreiben
- Bilder in den Shared Memory Bereich ablegen
- Mess-DLL aufrufen, mit den Parametern des Bildes und der gewünschten Bearbeitungsmethode

Das folgende Bild zeigt ein Screenshot der GUI 2.5

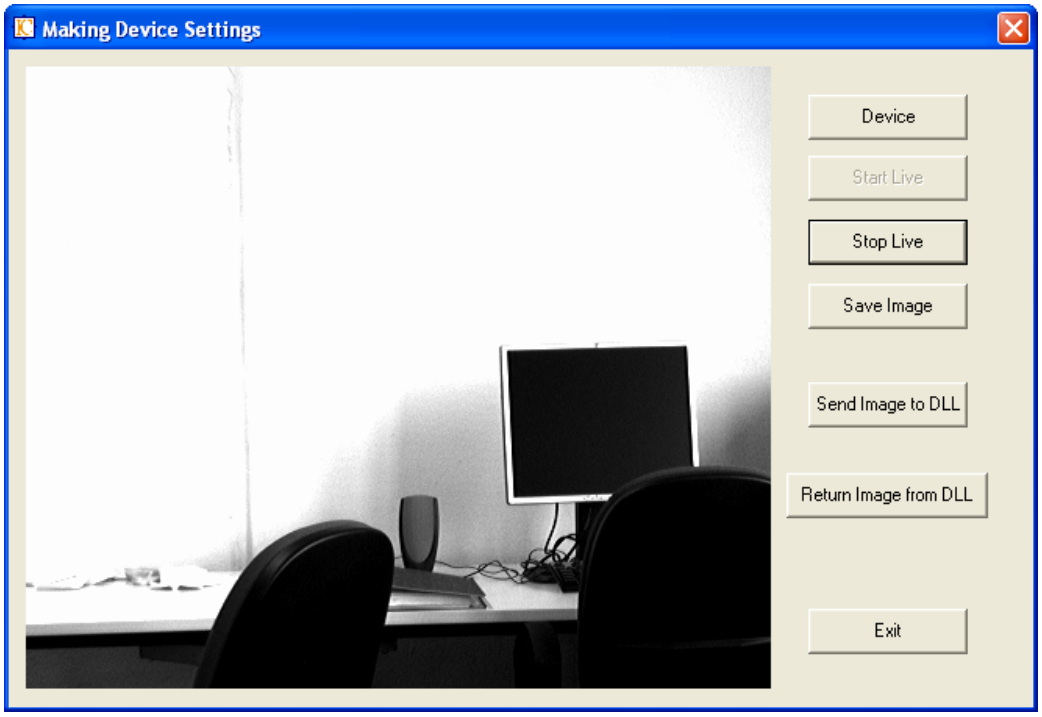


Abbildung 2.5.: Screenshot der GUI Steuerung

3. Realisierungsansätze und Lösungsvorschläge

In diesem Kapitel werden die verschiedensten Lösungsmöglichkeiten für die Kommunikation zwischen einer C++-Kamera-Anwendung und der LTI-Bibliothek beschrieben. Sowie Techniken die Zugriff auf das aufgenommene Bild der Kamera gewähren.

3.1. Gesamtprojektlösung

Die wohl naheliegendenste Lösung wäre es, die Kamera-Anwendungs-GUI und die dazu gehörigen Bildbearbeitungsalgorithmen in ein gemeinsames Projekt zu schreiben. Hierbei würden die Bilddaten, welche die Kamera speichert, als globale Variablen abgelegt werden, auf diese wiederum die Bildbearbeitungsmethoden zugreifen können.

Die Vorteile dieser Lösung sind:

- kein Bedarf an einem „Shared Memory“, auf das mehrere Pozesse zugreifen können
- einfache Bilddatenablage als globale public Variablen im Stack oder Heap
- schnellerer Zugriff auf die Bilddaten, Schreib und Lesezugriffe müssen nicht explizit geschützt werden

Dem stehen folgende Nachteile gegenüber:

- die GUI Anwendung unter Visual Studio 2005 läuft nur bedingt mit der LTI-Lib, laut Angaben der (LTILIB 2007)
- das Projekt wird mit zunehmender Größe und Komplexität schnell unübersichtlich
- bei Erweiterungen muss das gesamte Projekt bearbeitet und neu kompiliert werden
- schlechte Austauschbarkeit des Bedienkonzeptes bei sonst gleichbleibender Funktionalität

Aufgrund der hier aufgezählten Nachteile wurde diese Realisierungsmöglichkeit verworfen.

3.2. Kommunikation über das Dateisystem

Hierbei wird klar zwischen Kamera Software Steuerung und Bildbearbeitungsroutinen getrennt. Die von der Kamera aufgenommenen Bilder werden als image Dateien auf der Festplatte abgelegt und beim Aufruf der Mess-DLL von der Festplatte eingelesen. Nach der Bearbeitung auf Seiten der Mess-DLL findet eine erneute Speicherung auf der Festplatte statt. Das Anzeigen des Bildes in der GUI geschieht durch das Betätigen eines Lade-Buttons, wodurch ein Lesezugriff auf die Bilddatei von der Festplatte ausgeführt wird.

Die Vorteile dieser Lösung sind:

- kein Bedarf an einem echtem „Shared Memory“, auf dem mehrere Pozesse zugreifen müssen
- die GUI Anwendung ist unter Visual Studio 2005 programmierbar und die Mess-DLL (welche LTI-LIB Methoden verwendet) unter Visual Studio 2003

- durch die Speicherung als jpg/bmp Bilddatei können auch andere Programme Zugriff auf die aufgenommenen und bearbeiteten Bilder nehmen

Dem stehen folgende Nachteile gegenüber:

- Schreib und Lesezugriffe auf die Bilddateien müssen geschützt werden (Lese-& Schreib-Zugriffs-Kontrolle)
- langsamerer Zugriff auf die Bilddaten, da immer auf die Festplatte zugegriffen werden muss

Diese Realisierungsmöglichkeit wurde aufgrund ihrer Vor & Nachteile realisiert.

3.3. Shared-Memory-Lösung

Hierbei wird ebenfalls zwischen Kamera Software Steuerung und Bildbearbeitungsroutinen getrennt. Die von der Kamera GUI aufgenommenen Bilder werden in einen gemeinsamen Speicherbereich abgelegt, auf diesen sowohl GUI-Prozess als auch der Mess-DLL-Prozess Zugriff nehmen kann. Nach der Bearbeitung auf Seiten der Mess-DLL findet eine erneute Speicherung in dem Shared Memory Bereich statt. Das Anzeigen des Bildes in der GUI geschieht durch ein Lesezugriff auf den Shared Memory Bereich. Eine vereinfachte Darstellung ist unter (Abbildung3.1) zu sehen.

Die Vorteile dieser Lösung sind:

- die GUI Anwendung ist ebenfalls unter Visual Studio 2005 programmierbar und die Mess-DLL (welche LTI-LIB Methoden verwendet) unter Visual Studio 2003
- direkter Zugriff auf die Bilddaten, da der gemeinsame Speicherbereich von beiden Prozessen gelesen und beschrieben werden kann

- Schreib und Lesezugriffe müssen nicht besonders geschützt werden, diese Funktion übernimmt die API des Betriebssystems
- Kamera Steuerung und Bildbearbeitung können unabhängig voneinander entwickelt werden, lediglich ein Austausch der überarbeiten DLL Dateien ist notwendig

Dieser Lösung stehen folgende Nachteile gegenüber:

- beide Prozesse müssen aktiv sein, damit der Shared Memory Bereich nicht verloren geht

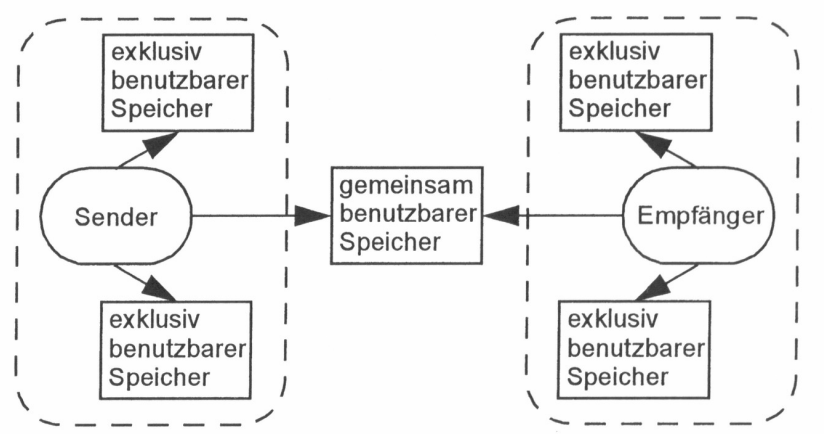


Abbildung 3.1.: Darstellung eines Shared Memory

3.4. Vergleich der Messzeiten

Auf einem Testsystem mit folgender Hardware:

CPU	RAM
2 x Intel Pentium D 3.0 GHz	2GB RAM

auf dem 50mal ein Bild der Größe 100x100 Bildpunkte angelegt und einmal als bmp-Datei und einmal ins Shared Memory Bereich abgelegt wurde, ergab folgendes Messergebnis 3.2:

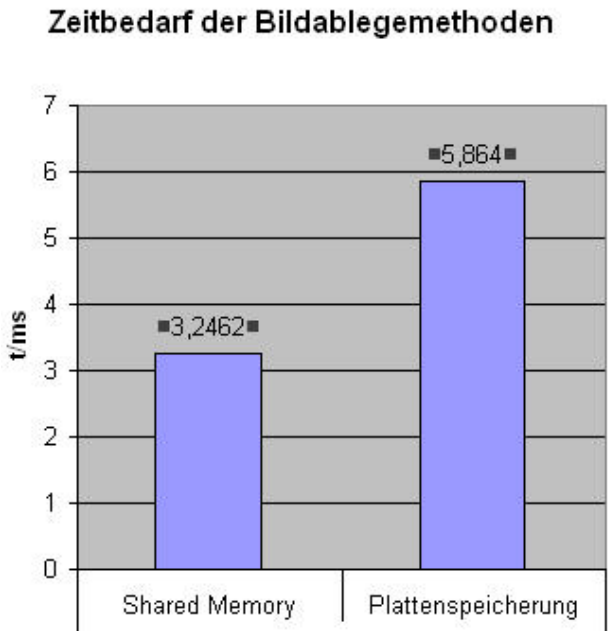


Abbildung 3.2.: Zeitmessung bei der Kommunikation über das Dateisystem und der Shared Memory Lösung

Auswertung

Der Vergleich zeigt einen klaren Vorteil zugunsten der Shared Memory Lösung. Insbesondere bei vielen hochauflösenden Bildern spielt der direkte Schreibzugriff in den Hauptspeicher seine Vorteile gegenüber dem Festplattenzugriff aus. (s. Bild 3.2)
Die Shared Memory Lösung wird aufgrund des Zeitvorteils favorisiert und dementsprechend auch implementiert.

4. Design und Aufbau

In diesem Kapitel werden Details zum Aufbau und Design des Projekts erläutert.

4.1. Auswahl der Entwicklungsumgebung für die GUI

Die Entwicklungsumgebung für die Steuerung der Imaging Source Kamera ist Visual Studio 2005. Die Kamera GUI ist in einem Unmanaged/Native[1] Code Projekt(C++-MFC-Anwendung) geschrieben und basiert auf einem Beispielprogramm, der Firma (imaging source 2007), welches um die Anforderungen der Aufgabenstellung erweitert wurde.
Auf Rücksprache mit einem Mitarbeiter der Firma Imaging Source ist nur ein MFC Projekt mit den Dateien „tisudshl.h“ und „CmdHelper.h“ funktionstüchtig. Aus diesem Grund war es nicht möglich, die Kamera unter einem CLR-Projekt[2] zum Laufen zu bringen.

Dadurch mussten auf die Vorteile, die es unter einem CLR-Projekt gibt(komfortableren GUI-Builder, Garbage Collection), verzichtet werden.

[1] Unmanaged Code ist das Gegenstück zu Managed Code, also Programmcode der nicht der Kontrolle der CLR unterliegt.

[2] Die Common Language Runtime (CLR) ist die virtuelle Maschine (VM) von .NET und stellt somit die Laufzeitumgebung für verschiedene an .NET angepasste Hochsprachen zur Verfügung. Neu hinzugekommen ist unter anderem eine Garbage Collection, wie sie z.B. im Java standardmäßig vorzufinden ist

4.2. Entwicklungsumgebung der Mess-DLL mit LTI-LIB-Funktionen

Die Entwicklung der Mess-DLL erfolgte unter Visual Studio 2003. Nur unter Visual Studio 2003 ist nach Angaben der (LTILIB 2007) eine problemlose Implementierung eines LTI Projektes gewährleistet.

Die Mess-DLL Bearbeitungsroutinen sind als Win32-DLL-Aufrufe implementiert. Wenn die DLL von der GUI angesteuert wird, greift die Mess-DLL auf den Shared Memory Bereich zu und ladet das zu verarbeitende Bild in seine Funktionen ein. Nach der erfolgreicher Bearbeitung des Bildes speichert die Mess-DLL das Bild in einem neuem Shared Memory Bereich. Auf diesem die GUI wieder Zugriff hat. siehe dazu auch Bild 2.4

4.3. Vorstellung der verwendeteten Werkzeuge und Betriebssystem-Mechanismen

4.3.1. Shared Memory

Shared Memory bezeichnet eine bestimmte Art der Interprozesskommunikation (IPC). Bei dieser Art nutzen zwei oder mehrere Prozesse einen bestimmten Teil des Arbeitsspeichers gemeinsam. Für alle beteiligten Prozesse liegt dieser gemeinsam genutzte Speicherbereich in deren Adressraum und kann mit normalen Speicherzugriffsoperationen ausgelesen und verändert werden. Weitere Informationen sind unter (Stegemann 2007) zu finden.

4.3.2. LTI-LIB

Die LTI-Bibliothek ist eine objektorientierte Bibliothek, die in C++ für Windows/MS-VC++ und Linux/GCC geschrieben wurde. Die LTI-Bibliothek wurde am Lehrstuhl der technischen Informatik an der RWTH Aachen Universität entwickelt und umfasst unter anderem:

- Bildverarbeitungsalgorithmen
- Klassifizierungsmethoden (Merkmalsextraktionen, Segmentierungen, etc)
- Hough-Transformation
- mathematische Operationen (Vektoren- und Matrizenrechnung)
- geometrische Bildtranformationen

Eine ausführliche Beschreibung findet sich unter (LTILIB 2007)

4.3.3. Aufgaben einer Dynamic Link Library (DLL)

Dynamic Link Library (DLL) bezeichnet allgemein eine dynamische Bibliothek, meist bezieht sich der Begriff jedoch auf die unter dem Betriebssystem Microsoft Windows verwendete Variante.

Eine DLL ist eine Datei mit Programmcode, der nicht eigenständig ausgeführt werden kann, sondern von einer Anwendung (.exe) oder einem Skript aus, aufgerufen werden muss.

Zweck

Der Zweck von DLL-Dateien ist, den von einer Anwendung auf der Festplatte und im Hauptspeicher benötigten Speicherplatz zu reduzieren. Jeglicher Programmcode, der von mehr als einer Anwendung benötigt werden könnte, wird deshalb in einer einzelnen Datei auf der Festplatte gespeichert und nur einmal in den Hauptspeicher geladen, wenn mehrere Programme dieselbe Programmbibliothek benötigen.

Vorteile des DLL-Konzepts

Wird ein Stück Programmcode verbessert, müssen nicht alle Programme geändert werden, die diesen Code nutzen, sondern es genügt, ihn in der DLL zu aktualisieren. Alle Programme können in diesem Fall auf die aktualisierte Fassung zugreifen. Dadurch ist es Softwareentwicklern möglich, relativ kleine Patches für größere Softwarepakete herauszugeben, beispielsweise auch für ganze Betriebssysteme. Ein ganzes Paket kann so durch die Aktualisierung einzelner DLLs auf den neuesten Stand gebracht werden.

In Form von Plug-ins können mit DLLs neue Programmteile für ein bereits bestehendes Programm erstellt und darin nahtlos integriert werden, ohne dass am schon existierenden Programm Veränderungen vorgenommen werden müssten.

Beim DLL-Konzept findet die Direktive „Trennung von Aufrufenden und Aufgerufenden Methoden“ aus der Softwareentwicklung strikte Anwendung. Ein unkomplizierter Austausch der DLL-Datei erfordert keine Neu-Kompilierung des Gesamtprojektes.

Nachteile des DLL-Konzepts

Ein Nachteil, der durch die Verwendung von DLLs auftreten könnte, ist das bei einem Überschreiben oder Austauschen einer älteren DLL-Datei durch

eine aktualisierte Version Einfluss auf anderen Progamme nehmen könnte. Die auf die ursprüngliche DLL-Datei angewiesen Programme könnten mit einer geänderten DLL-Version in Konflikt geraten.

Vergleich von statischer und dynamischer DLL

Es gibt zwei verschiedene Arten, wie DLLs vom Betriebssystem in den Speicher geladen werden. Es gibt statische DLLs, die nur einmal geladen werden. Alle Programme greifen dann auf diese eine Instanz der DLL zu. Diese DLL hat dann nur einen globalen Speicherbereich. Die Windows-Kernel-DLLs sind solche statischen DLLs, was ihnen erlaubt, das gesamte System zu verwalten (z. B. alle offenen Dateien zu überwachen).

Die andere Art, wie eine DLL im Speicher verwaltet werden kann, ist die dynamische DLL, dass jedes mal wenn ein neues Programm sie benötigt, auch eine neue Instanz von ihr in den Speicher geladen wird. Ob eine DLL statisch oder nicht ist legt ein weiteres Flag im Header der DLL fest. Weitere Informationen sind in den Büchern (Stevens 2004) und (Richard 2001) zu finden.

4.3.4. Aufruf und Einbindung einer Dynamic Link Library (DLL)

Damit eine Programmthread eine Funktion in einem DLL-Modul 4.1(hier aus der Mess-DLL) aufrufen kann, muss die DLL im Adressraum des aufrufenden Threads eingebettet werden.

Abbildung 4.1.: Verarbeitungssequenz eines typischen DLL-Aufrufes

Erstellen einer DLL

Die DLL-Schnittstelle wird mit Hilfe der Export-Funktion declspec(dllexport) definiert. Dies wird im folgenden Beispiel demonstriert:

```
#include <windows.h>
#if defined(_MSC_VER)
#define DLL extern "C" __declspec(dllexport)
#else
#define DLL
#endif
// Die Funktion, die anderen Programmen zur Verfügung gestellt werden soll
// (in diesem Beispiel: Addieren zweier Zahlen)
DLL double AddNumbers (double a, double b) {
        return a + b;
}
```

Dieses Beispiel erzeugt beim Compilieren sowohl eine DLL als auch eine LIB-Datei.

Einbindung einer DLL, zur Compilzeit

DLL-Funktionen können einfach aufgerufen werden, nachdem man sie mit der Funktion declspec(dllimport) importiert hat.

```
#include <windows.h>
#include <stdio.h>
// Importieren der Funktion aus der oben erstellten DLL
extern "C" __declspec(dllimport)double AddNumbers (double a, double b);

int main () {
        // Aufrufen der externen Funktion
        double result = AddNumbers(1, 2);
        printf("Das Ergebnis ist: %f\n", result);
        return 0;
}
```

Zu beachten ist, dass der Linker die LIB-Datei benötigt und dass sich die DLL-Datei im selben Ordner wie das Programm, das sie aufrufen soll, befinden sollte. Die LIB-Datei wird vom Linker benötigt, da diese (ähnlich einer Header-Datei) die Funktionsprototypen enthält.

Einbindung einer DLL, zur Laufzeit

DLL-Bibliotheken können auf zwei verschiedene Weisen in eine Anwendung geladen werden, entweder mit dem Starten des Programmes (so wie in den obigen Beispielen beschrieben) oder während der Laufzeit, indem man die API-Funktionen LoadLibrary, GetProcAddress und FreeLibrary verwendet.
Die Art und Weise, wie DLLs während der Laufzeit einzubinden sind, ist in jeder Programmiersprache gleich, solange man eine Windows-API-Funktion importieren möchte.
Der folgende Code demonstriert das anhand eines VC++-Beispieles:

```
#include <windows.h>
#include <stdio.h>

// Definition der DLL-Funktion, die verwendet werden soll
typedef double (*AddNumbers)(double, double);

int main () {
        AddNumbers function;
  double result;

  // DLL Datei laden
  HINSTANCE hinstLib = LoadLibrary("MyDll.dll");

  if (hinstLib) {
        // Die Einsprungadresse abfragen
    function = (AddNumbers) GetProcAddress(hinstLib, "AddNumbers");

    // Die Funktion aufrufen
    if (function)
        result = (function) (1,2);
```

```
    // Die DLL-Datei wieder entladen
        BOOL fFreeResult = FreeLibrary(hinstLib);
    }

    // Das Ergebnis anzeigen
    if (!hinstLib  || !function)
        printf("Fehler: Konnte die Funktion nicht aufrufen\n");
    else
        printf("Das Ergebnis ist: %f\n", result);
    return 0;
}
```

Die LIB-Datei wird in diesem Fall nicht benötigt. Die DLL-Datei muss aber immer noch in einem Ordner liegen, der dem Programm zugänglich ist.

Zu beachten ist außerdem, dass beim Versuch, eine nicht vorhandene DLL direkt beim Programmstart automatisch mitladen zu lassen, vom Betriebssystem eine Fehlermeldung angezeigt und das Programm beendet wird, ohne dass der Programmierer eine Möglichkeit hat, diesen Fehler abzufangen.
Beim Einbinden von DLLs während der Laufzeit können Fehler beim Laden hingegen abgefangen werden.
Weitere Informationen zu dynamischen Bibliotheken findet man unter (Wikipedia 2007a), (Soltendick 1998) und (Richter 1997, 569ff) sowie (Petzold 2000, 1145ff)

Invertierungs-Beispiel

Ein leicht verständliches Beispiel wie man in der LTI Lib ein Bild in ein Shared Memory Bereich ablegen und Einlesen kann, befindet sich im Anhang.

5. Konzept und Implementierung

In diesem Kapitel wird die Implementierung des Projektes anhand eines Beispiels erläutert.
Dieses Projekt besteht aus 2 Komponenten. Der GUI-Komponente, zur grafischen Steuerung des Projektes, und der Mess-DLL, zum Bearbeiten der aufgenommenen Bilder.

5.1. Übersicht des Programmablaufs

1. Einstellung der Kameraparameter in der GUI,
2. LiveBild anzeigen (StartLive),
3. gewünschtes LiveBild grabben[1] und in den Shared Memory Bereich schreiben,
4. Bild in der Mess-DLL bearbeiten(hier nur Bildinvertierung),
5. das bearbeitet Bild, welches die Mess-DLL in ein neues Shared Memory Bereich geschrieben hat, einlesen und auf der GUI anzeigen lassen (hier nur das zuvor aufgenommene Bild invertiert darstellen),
6. bei Bedarf das Bild als .bmp Datei auf der Festplatte speichern.

[1] live Bilder von der Kamera in den Speicher laden

5.2. Image Capturing

Im folgendem werden die wichtigsten Funktionen erklärt, die nötig sind, um Bilder von einer Imaging Source Kamera aufzunehmen und ein Verweis auf das gespeicherte Bild zu bekommen.

```
//erstmal Senke erstellen
pSink = FrameHandlerSink::create( eY800, 5 );
//FrameHandlerSink wird benutzt um ein Frames
//vom Image Stream der Kamera zu Kopieren

// Snapmode aktivieren
pSink->setSnapMode(true);
//Im Snap Modus werden die aufgenommen Bilder in der Reihenfolge
//ihrer Erstellungszeit gespeichert
//Im Grap Modus werden alle Frames der Senke übergeben und in die
//MemBufferCollection übergeben

// Senke setzen
m_Grabber.setSinkType( pSink );
//dem Grabber wird die Senke übergeben

// Grabber starten
m_Grabber.startLive(true);
//Setzt den Grabber in den live Modus

// Image snappen
Error e = pSink->snapImages( 1 );
//Nimmt ein Bild auf und kopiert es in die MemBufferCollection

int size_buffer = m_Grabber.getAcqSizeMaxX() *m_Grabber.getAcqSizeMaxY();
//Berechnet die Größe des aufgenommenen Bildes (Breite*Höhe)

BYTE* pPic = pSink->getLastAcqMemBuffer()->getPtr();
//setzt den pPic Pointer auf das in die MemBufferCollection
//gespeicherte Bild
```

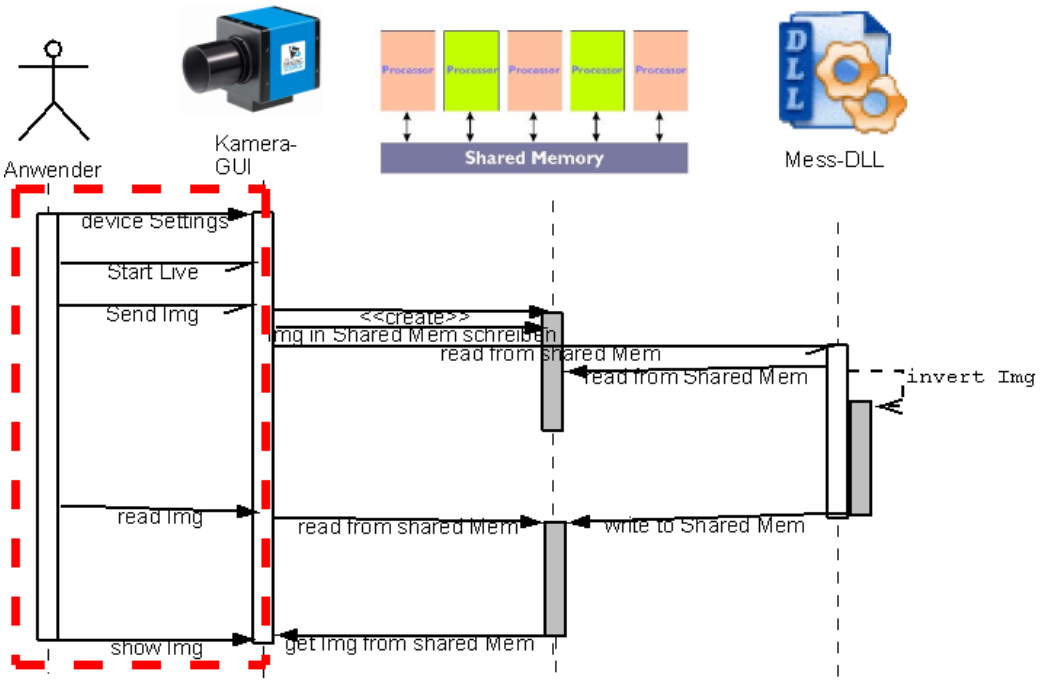


Abbildung 5.1.: Übersicht der GUI Methoden

5.3. Shared Memory Methoden der Win32-API

Die Windows-API[2] bei modernen Windowssystemen ist Win32-API[3]. Um eine Kommunikation zwischen 2 oder mehr Prozessen unter einer Win32 Oberfläche zu erreichen, ist die Nutzung eines Shared Memory die eleganteste Lösung.

[2]Windows Application Programming Interface zu dt. etwa: Windows-Anwendungs-Programmierungs-Schnittstelle ist eine Programmierschnittstelle und Laufzeitumgebung, welche Windows-Programmierern bereitsteht, um Anwendungsprogramme für Windows-Betriebssysteme zu erstellen

[3]Win32 ist die 32-Bit API für moderne Versionen von Windows. Die API besteht aus Funktionen die, wie bei Win16, in Programmbibliotheken implementiert sind, wurde mit Windows NT eingeführt.(Wikipedia 2007b)

WIN32 bietet die Möglichkeit, Dateien in den Adreßraum eines Prozesses einzublenden. Der physikalische Speicher wird dabei nicht in der Auslagerungsdatei reserviert, sondern liegt direkt in der in den Adreßraum abgebildeten Datei (oder in einem entsprechenden Cache Bereich im Hauptspeicher).

Dieser Abschnitt zeigt, wie Daten zwischen simultan ausgeführten Prozessen ausgetauscht werden können. Das folgende Beispiel gibt eine Einblick, welche 3 Schritte notwendig sind, um ein „Shared Memory“ zu implementieren:

1. Erzeugung einer mapped file zum Programmbeginn
2. Gewährung von Lese & Schreibzugriffen auf die File Mapping Structure
3. Schließung und Freigabe der mapped file zum Programmende

Die folgende Abbildung 5.2 zeigt die Beziehung zwischen der File on disk, dem Filemapping Object und der File View.

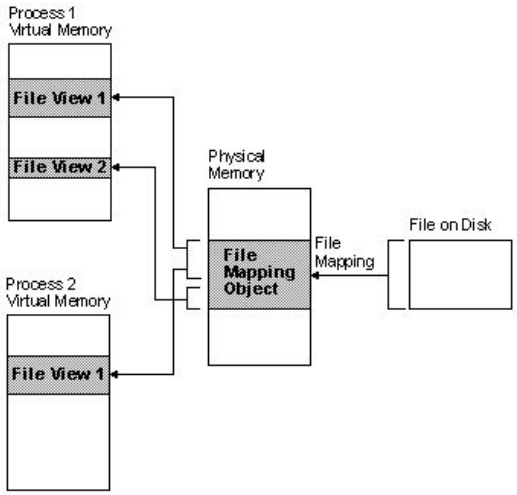


Abbildung 5.2.: FileMapping

Prozesse, welche Lese oder Schreibzugriffe auf die File-View ausüben, tun dies mit Hilfe von Pointern, so als wäre es Zugriff auf dynamisch allokierter Speicher. Das File-on-Disk kann jede beliebige Datei sein, die in den gemeinsamen Speicher abgelegt werden soll. Ein File View kann aus dem gesamten File Mapping Object bestehen oder nur Teile des File Mapping Objects.

Die Methoden, welche die Win32 API bereit stellt, allokieren direkt physikalischen Speicher als wäre es virtueller Speicher, welcher vom Betriebssystem für den Prozess zur Verfügung gestellt wird. Diese Funktionen reservieren Speicher in speziell dafür vorgesehene Speicherbereiche.

Übersicht der Shared Memory Funktionen

Die Wichtigsten Memory Mapped File Funktionen sind:

1. Erzeugen bzw. Öffnen der Datei mit Hilfe von **CreateFile**
2. Zugriff auf das Dateiabbildungsobjektes mit Hilfe der Funktonen:
   a) Erzeugen eines neuen Dateiabbildungsobjektes (falls noch nicht vorhanden) mit **CreateFileMapping**
   b) Öffnen eines bereits erzeugten Dateiabbildungsobjektes (falls bereits vorhanden) mit Hilfe von **OpenFileMapping**

   Das Dateiabbildungsobjekt enthält für das System wichtige Informationen zur Verwaltung einer speicherbasierten Datei. Beide Funktionen liefern im Erfolgsfall ein Handle zurück, das ungleich NULL ist, und mit dem im dritten Schritt schließlich ein Zeiger auf die Abbildung der Datei im Prozeßaddressraum ermittelt wird.
3. Die Funktion **MapViewOfFile** liefert schließlich einen void Zeiger zurück, mit dessen Hilfe der Zugriff auf die Daten der Datei auf die gleiche Weise erfolgen kann, wie auf einen ganz normalen Speicherbereich.

4. Freigabe alle reservierten Handles & Zeiger, sobald die speicherbasierte Datei nicht mehr benötigt wird:
   a) Durch **CloseHandle** wird das reservierten Filemapping-Handles wieder freigegeben
   b) Durch **UnmapViewOfFile** wird der reservierte void Zeigers wieder freigegeben

Im folgendem werden die verwendeten Win32 API Methoden zum Zugriff auf das Shared Memory anhand von Beispielquellcode erklärt.

CreateFileMapping

Diese Methode der Win32 API erzeugt ein neues file mapping Objekt für die angegebene Datei. Der Aufruf der Funktion mit dem „INVALID_HANDLE_VALUE“ erzeugt ein file mapping Object mit der Größe der buffer size Angabe(BUF_SIZE). Wenn die CreateFileMapping-Funktion erfolgreich ausgeführt wurde, ist der Rückgabewert das Handle auf das Filemapping Object. Ist der Rückgabewert Null, ist die ein Fehler im Funktionsaufruf aufgetreten.

Beschreibung der CreateFileMapping API Methode:

```
HANDLE CreateFileMapping(
  HANDLE hFile,                       // handle to file
  LPSECURITY_ATTRIBUTES lpAttributes, // security
  DWORD flProtect,                    // protection
  DWORD dwMaximumSizeHigh,            // high-order DWORD of size
  DWORD dwMaximumSizeLow,             // low-order DWORD of size
  LPCTSTR lpName                      // object name
);
```

Verwendete Parameter:

```
hMapFile = CreateFileMapping(
                  INVALID_HANDLE_VALUE,    // use paging file
                  NULL,                    // default security
                  PAGE_READWRITE,          // read/write access
                  0,                       // max. object size
                  BUF_SIZE,                // buffer size
                  szName);                 // name of mapping object

if (hMapFile != NULL && GetLastError() == ERROR_ALREADY_EXISTS)
{
            CloseHandle(hMapFile);
            hMapFile = NULL;
}
return hMapFile;
```

OpenFileMapping

```
HANDLE OpenFileMapping(
  DWORD dwDesiredAccess,  // access mode
  BOOL bInheritHandle,    // inherit flag
  LPCTSTR lpName          // object name
);
```

Die OpenFileMapping Methode öffnet ein file mapping Object, welches im lpName-Parameter angeben wird.

MapViewOfFile

Die MapViewOfFile Funktion bildet die Sicht auf das file mapping object des aufrufenden Prozesses in den Adressraum ab.

```
LPVOID MapViewOfFile(
  HANDLE hFileMappingObject,   // handle to file-mapping object
  DWORD dwDesiredAccess,       // access mode
  DWORD dwFileOffsetHigh,      // high-order DWORD of offset
  DWORD dwFileOffsetLow,       // low-order DWORD of offset
  SIZE_T dwNumberOfBytesToMap  // number of bytes to map
);
```

UnmapViewOfFile

Die UnmapViewOfFile Methode entfernt die Sicht auf das file mapping Object des aufrufenden Prozesses aus seinem Adressraum.

CloseHandle

Schließt das Handle auf ein zuvor geöffnetes file mapping Object.

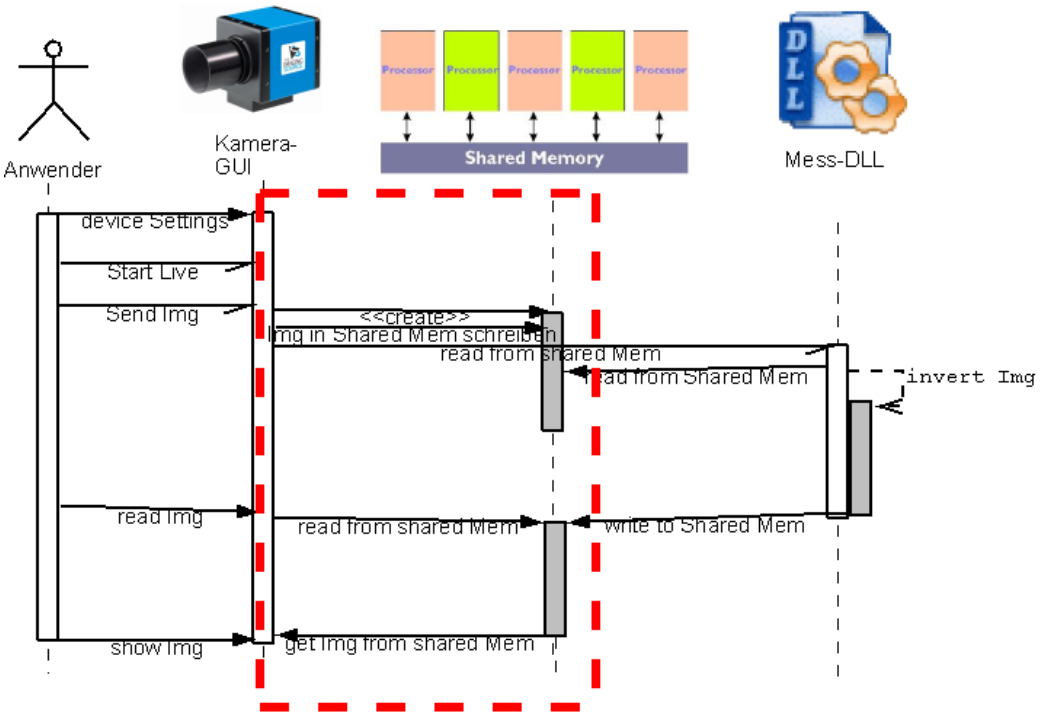


Abbildung 5.3.: Übersicht der Shared Memory Methoden

5.4. Aufruf der Mess-DLL

Nachdem nun das aufgenommene Bild im Shared Memory Bereich abgelegt worden ist, muss die Mess-DLL aufgerufen werden. Hierzu sind folgende Schritte notwendig:

1. Parameterdefinition der DLL

```
typedef int (* LPFNDLLFUNC1)(int, int, int);
```

2. Anlegung des DLL-Handle

```
HINSTANCE hDLL;                 // Handle to DLL
LPFNDLLFUNC1 lpfnDllFunc1;      // Function pointer

hDLL = LoadLibrary("Win32Dll.dll");
lpfnDllFunc1 = (LPFNDLLFUNC1)GetProcAddress(hDLL,"Fktname");
```

3. Funktionsaufruf

```
int intRes = lpfnDllFunc1(height, width, size_buffer);
```

   Der Funktionsaufruf lpfnDllFunc1 wird mit den entsprechenden Parametern des Bildes aufgerufen und liefert als Rückgabewert einen Integer Wert, der den Erfolg bzw. Misserfolg des Funktionsaufrufes quitiert.

5.5. Zugriff der Mess-DLL auf den Shared Memory Bereich

Nachdem der OpenFileMapping-Aufruf in der Mess-DLL einen ubyte-Pointer auf das Bild liefert, muss der Pointer mit Hilfe der LTI-LIB bearbeitet werden. Hierzu sind folgende Schritte notwendig:

1. Anlegung eines channel8 Bild unbekannter Größe

```
lti::channel8 myImg(0,0);
//kein Speicherallocierung
```

2. Speicherzuweisung mit Hilfe der attach Methode der LTI-Lib (Abbildung 5.4)

```
myImg.attach(height,width,pBuf);
//Pointer aus dem Shared Memory Bereich
//auf das channel8 Bild zuweisen
```

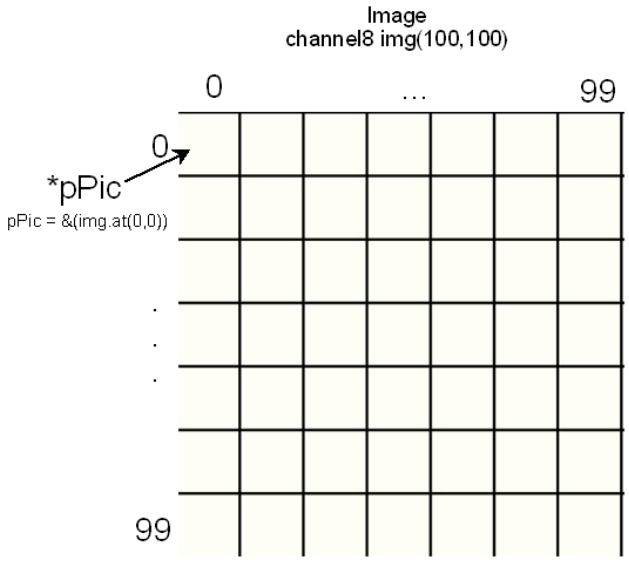


Abbildung 5.4.: Zeigerverweis auf das im Speicher abgelegte Bild

3. Anlegen eines Zielbildes mit entsprechender Größe

```
lti::channel8 dst(0,0);
dst.resize(height,width);
//set destination size to source size
```

4. Aufruf der Bearbeitungsroutine

```
doSomething(myImg,dst);
```

Das bearbeitete Bild wird in einem neuem Shared Memory Bereich abgelegt, sodass die GUI das bearbeitet Bild anzeigen kann.

Eine Übersicht vom Programmstart bis zur Verarbeitung zeigt das folgende Bild 5.5.

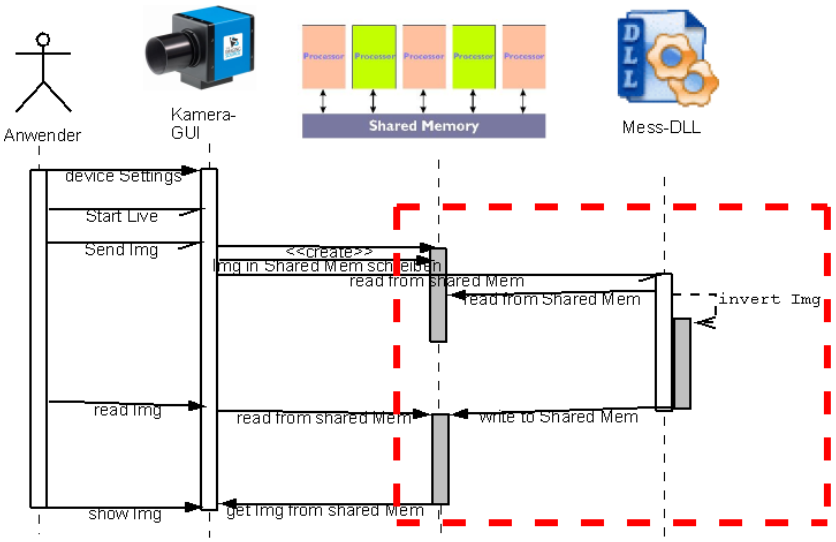


Abbildung 5.5.: Übersicht der DLL Methoden

6. Zusammenfassung

6.1. Abschließende Bemerkung und Bewertung der Arbeit

Diese Bachelorarbeit hatte zum Ziel, die Konzeptionierung und Implementierung eines interaktiven Bildmesssystems, das die Kommunikation zwischen einer dynamischen C++ Bildverarbeitungsbibliothek (wurde als Mess-DLL umgesetzt) und einer in C++ GUI-Anwendung, sicherstellt. Hintergrund ist die Trennung von grafischer Benutzer-Oberfläche(GUI) und verarbeitende Bildbearbeitungsroutinen(Mess-DLL). Der Vorteil, welcher durch diese Trennung erreicht wurde, ist die einfache Austauschbarkeit der DLL-Komponente ohne das gesamte Projekt bearbeiten oder neu kompilieren zu müssen.
Die Problemstellung wurde in zwei Teilfelder aufgeteilt:

- Zum Einem die Erstellung der graphischen Oberfläche zur Steuerung der Kamera und des Gesamtprojektes
- Zum Anderen die Entwicklung des Kommunikationsinterfaces zwischen GUI und Mess-DLL

Für die Realisierung des Kommunikationsinterfaces kamen verschiedene Techniken in Frage, von denen 2 auch implementiert und verglichen wurden. Eine Messung der Zugriffszeiten zwischen einer Bildspeicherung auf der Festplatte und in einem Shared Memory Bereich ergab einen klaren Vorteil zugunsten der Shared Memory Technik. Die ursprünglich gestellten Anforderungen, konnten mit Hilfe dieser Art der Datenspeicherung erreicht werden.

6.2. Ausblick

Mit diesem entwickeltem Konzept, Datenspeicherung in einem Shared Memory und Bearbeitung in einer unabhängigen DLL, kann jetzt das angestrebte Messsystem entwickelt werden. Durch den Einsatz dieser Basismechanismen ist der Grundstein für das Messsystem gelegt. Im Folgendem sind noch diese Punkte zu klären:

- Auswahl der Kamera (Auflösung, S/W oder Farbe, Position im Messsystem)
- Einsatz der Bildbearbeitungsroutinen (Filterung der aufgenommenen Bilder)
- Größenberechnung der Messobjekte

Literaturverzeichnis

(CIM-Aachen 1998) CIM-AACHEN: *Produktivität und Prozessketten im Industriebetrieb*. 1998. – URL http://www.cim-aachen.de/index.php

(Gesellschaft 2007) GESELLSCHAFT, Fraunhofer: *Kamera und Auswertungssysteme für industrielle Prüfaufgaben*. 2007. – URL http://www.vision.fraunhofer.de/de/2/projekte/6.html

(Harbili 2006) HARBILI, Zakaria E.: *Kommunikationsinterface zwischen einer dynamischen C++-Bildverarbeitungsbibliothek und Java-basierten Anwendungen*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2006. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/diplom/harbili.pdf

(LTILIB 2007) LTILIB: *LTI-Introduction*. 2007. – URL http://ltilib.sourceforge.net/doc/homepage/index.shtml

(Materialprüfungsamt-NRW 2006) MATERIALPRÜFUNGSAMT-NRW: *Prüfung und Kalibrierung von Werkstoffprüfmaschinen*. 2006. – URL http://www.mpanrw.de/start.html

(Petzold 2000) PETZOLD, Charles: *Windows Programmierung : [das Entwicklerhandbuch zur WIN32-API]*. Microsoft Press Deutschland, 2000. – ISBN 3-86063-487-9

(Richard 2001) RICHARD, J. S.: *Windows-2000-API-Referenz*. Markt und Technik-Verl., 2001. – ISBN 3-82725999-1

(Richter 1997) RICHTER, Jeffrey: *Microsoft Windows : Programmierung für Experten [der Entwicklerleitfaden zur Win32-API für Windows NT 4.0 und Windows 95]; Bd. 3*. Microsoft Press, 1997. – ISBN 3-86063-389-9

(Soltendick 1998) SOLTENDICK, Wolfgang: *Das Win32-API; Bd. 2*. C und L Computer und Literaturverlag, 1998. – ISBN 3-932311-36-1

(imaging source 2007) SOURCE imaging: *imagingsource Manuel*. 2007. – URL http://www.theimagingsource.com/

(Stegemann 2007) STEGEMANN, Fred: *Shared Memory unter WIN32*. 2007. – URL http://www.fstsoft.de/shared.htm

(Stevens 2004) STEVENS, W. R.: *The Sockets Networking API*. Addison-Wesley, 2004. – ISBN 0-13-14155-1

(Wikipedia 2007a) WIKIPEDIA: *Dynamic Link Library*. 2007. – URL http://de.wikipedia.org/wiki/Dynamic_Link_Library

(Wikipedia 2007b) WIKIPEDIA: *Windows Application Programming Interface*. 2007. – URL http://de.wikipedia.org/wiki/Win32#Win32

A. LTI-Lib Beispiel

Ein Beispielprogramm bei dem 2 Programme auf einen gemeinsamen Speicherbereich mit Hilfe des Shared Memory Verfahrens zugreifen können.

Beispiel Programm 1 legt ein 100x100 Muster-Bild an und legt es in dein Shared Memory Bereich ab.

RV02.cpp

```
/*
InvertImage CreateFileMapping Bild anlegen
*/
#include "RV02.h"

using std::cout;
using std::endl;

TCHAR szName[]=TEXT("Global\\MyFileMappingObject");
//Bezeichnung des Shared Memory Bereich

namespace lti {

  void RV02::operator()(int argc,char *argv[]) {

    /*********************************************/
    /**** has always to be started (AMei)     ****/
    /**** if program is without gtk-Widgets   ****/
    /*********************************************/
```

```
gtkServer server;
server.start();

/*****************************************/
/**** instantiation of used components ****/
/*****************************************/

/*----------------------*/
/* viewers and savers*/
/*----------------------*/
saveBMP saver;
viewer view("Original");
viewer viewTransformed("Filtered");

/*----------------------*/
/* images & channels   */
/*----------------------*/

HANDLE hMapFile;

ubyte *pBuf;
ubyte *pPic;

const int height = 100, width = 100;
//Groesse des Testbildes: 100x100 Pixel

lti::channel8 img(height, width);

for(int y = 0; y < img.size().y; y++){
    for(int x = 0; x < img.size().x; x++){
        img.at(y,x) = 255;
    }
}
for(int y = 0; y < img.size().y/2; y++){
    for(int x = 0; x < img.size().x/2; x++){
        img.at(y,x) = 128;
    }
}
```

```
pPic = (ubyte *)(&(img.at(0,0)));

/*****************************************/
/*    the program                        */
/*****************************************/

const int rowSize    = img.rows();
const int columnSize = img.columns();
const int myTotalSize = rowSize * columnSize;

printf("myTotalSize: %d\n", myTotalSize);

hMapFile = CreateFileMapping(
    INVALID_HANDLE_VALUE,    // use paging file
    NULL,                    // default security
    PAGE_READWRITE,          // read/write access
    0,                       // max. object size
    myTotalSize,             // buffer size
    szName);                 // name of mapping object

if (hMapFile == NULL)
{
    printf("Could not create file mapping object (%d).\n",
    GetLastError());
}
pBuf = (ubyte *)MapViewOfFile(hMapFile,// handle to map object
    FILE_MAP_ALL_ACCESS, // read/write permission
    0,
    0,
    myTotalSize);

if (pBuf == NULL)
{
    printf("Could not map view of file (%d).\n",
    GetLastError());
}

CopyMemory((PVOID)pBuf, (PVOID)pPic, myTotalSize);
printf("CopyMemory, bitte Prozess 2 Aufrufen! \n");
saver.save("test.bmp",img);
```

```
    printf("Bild abgespeichert\n");
    getchar();
    printf("getChar() nach UnmapViewOfFile");
    UnmapViewOfFile(pBuf);
    CloseHandle(hMapFile);
  }
};
```

Beispiel Programm 2 liest das Bild aus dem Shared Memory Bereich und invertiert es anschließend.

RV02load.cpp

```
/*
InvertImage und OpenFileMapping, Bild aus Shared Memory einlesen
*/
#include "RV02.h"

using std::cout;
using std::endl;

TCHAR szName[]=TEXT("Global\\MyFileMappingObject");

namespace lti {
    void RV02::operator()(int argc,char *argv[]) {
        /*********************************************/
        /**** has always to be started (AMei)     ****/
        /**** if program is without gtk-Widgets   ****/
        /*********************************************/
        gtkServer server;
        server.start();

        /*******************************************/
        /**** instantiation of used components ****/
        /*******************************************/

        /*----------------------*/
        /* viewers and savers   */
        /*----------------------*/

        saveBMP saver;
        viewer view("Original");
        viewer viewTransformed("Invertiert");
        /*----------------------*/
        /* images & channels    */
        /*----------------------*/
```

```
HANDLE hMapFile;

ubyte *pBuf;
ubyte *pPic1;

const int height = 100, width = 100;
const int myTotalSize = height*width;

/*****************************************/
/*    the program                        */
/*****************************************/

printf("myTotalSize: %d", myTotalSize);

hMapFile = OpenFileMapping(
    FILE_MAP_ALL_ACCESS,   // read/write access
    FALSE,                 // do not inherit the name
    szName);               // name of mapping object

if (hMapFile == NULL)
{
    printf("Could not open file mapping object(%d)\n",
    GetLastError());
}

pBuf=(ubyte *) MapViewOfFile(hMapFile,//handle to map object
    FILE_MAP_ALL_ACCESS,//read/write permission
    0,
    0,
    myTotalSize);

if (pBuf == NULL)
{
    printf("Could not map view of file(%d)\n",
    GetLastError());
}

lti::channel8 myImg(0,0);//keine Alocierung von Speicher
```

```
        myImg.attach(height,width,pBuf);
        //vorhandenen Speicher von pPuf zuweisen
        //LTI attach Methode weist dem channel8 Bild
        //den Bildspeicher aus dem Shared Memory Bereich zu

        view.show(myImg);

        saver.save("original.bmp",myImg);

        lti::channel8 dst(0,0);
        // set destination size to source size
        dst.resize(height,width,0,false,true);

        InvertImage(myImg,dst);
        //Aufruf der Bearbeitungsmethode
        saver.save("dst.bmp",dst);
        viewTransformed.show(dst);
        printf("Show and Save Pics");
        getchar();

        UnmapViewOfFile(pBuf);

        CloseHandle(hMapFile);
    }
    /*******************************************************/
    /* Function definition: ----- InvertImage-operator----*/
    /*******************************************************/
    void RV02::InvertImage(const channel8& sPic,// source picture
        channel8& dPic// destination picture
        )
    {
        const int PicSizeY = sPic.rows();
        const int PicSizeX = sPic.columns();

        int x,y;
        for(y=0; y<PicSizeY; y++)
            for(x=0; x<PicSizeX; x++)
                dPic[y][x] = 256 - sPic[y][x];
    }
};
```

B. Inhalt der CD

Verzeichnis und Datei	Beschreibung
Bachelorarbeit.pdf	Dieses Dokument im pdf Format
Quellcode/GUI	Quellcode zur GUI
Quellcode/DLL	Quellcode zur Mess-DLL
Quellcode/BSP	Quellcode zum Beispielprogramm
Literatur	Kopie verwendeter Internetseiten

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 18. Oktober 2007	
Ort, Datum	Unterschrift