Projektarbeit

Fahrsimulator

Sandro Ropelato (ropelsan)
Christof Würmli (wurmlchr)

18. Dezember 2011

Studiengang: Systeminformatik SI
Betreuender Dozent: Peter Früh (frup)

Inhaltsverzeichnis

1. Einleitung

1.1. Ausgangslage

Im Gebiet der Fahrsimulatoren gibt es bereits eine Vielzahl von verschiedenen Lösungen. Einige davon bestehen aus Filmmaterial, das abgespielt wird und der Fahrer muss auf die Bremse drücken sobald ein bestimmtes Ereigniss eintritt. Andere Fahrsimulationen bringen bereits eine virtuelle Welt mit, in der man sich mehr oder weniger frei bewegen bzw. frei fahren kann. Jedoch sind bei den meisten von diesen Fahrsimulatoren bereits feste Szenarien implementiert die nicht geändert werden können.

Die Grenzen eines Fahrsimulators liegen vor allem in der Leistungsfähigkeit des Rechners auf dem die Simulation installiert werden soll.

Das Projekt wird in Zusammenarbeit mit der ETH Zürich durchgeführt. Es ist bereits eine LabView Schnittstelle für das Steuerrad vorhanden.

1.2. Aufgabenstellung

1.2.1. Formulierung

Das Ziel der Arbeit besteht darin, einen Fahrsimulator für die bestehende Simulatinsumgebung zu erstellen. Diese besteht aus einem Fahrercockpit, einer Leinwand, einem Projektor und einem Computerterminal, von dem aus die Simulation gesteuert werden kann. Das Fahrercockpit enthält ein Steuerrad, drei Pedalen, einen Schaltknüppel und einen Autositz mit Sicherheitsgurt.

Die Fahrsimulation sollte dem Benutzer die Illusion des Autofahren möglichts realistisch vermitteln. Die soll durch einen Einsatz von Karteninfomationen von Google Maps oder Google Street View unterstützt werden.

Zudem sollen alle Betriebszustände und Benutzereingaben registriert und aufgezeichnet werden um eine genaue Analyse zu ermöglichen.

1.2.2. Aufteilung der Arbeit

Wir teilten die Arbeit im Wesentlichen in zwei Teile auf. Im ersten Teil legten wir den Fokus auf die korrekte Ansteuerung des Cockpits. Um dies zu testen, setzten wir uns das Ziel, ein Video abzuspielen und mit Gas- und Bremspedal die Geschwindigkeit kontrollieren zu können. Im zweiten Teil folgte die Implementation und Installation der Fahrsimulation.

1.3. Zeitplan und Arbeitsteilung

2. Aufbau des Systems

2.1. Systembeschreibung

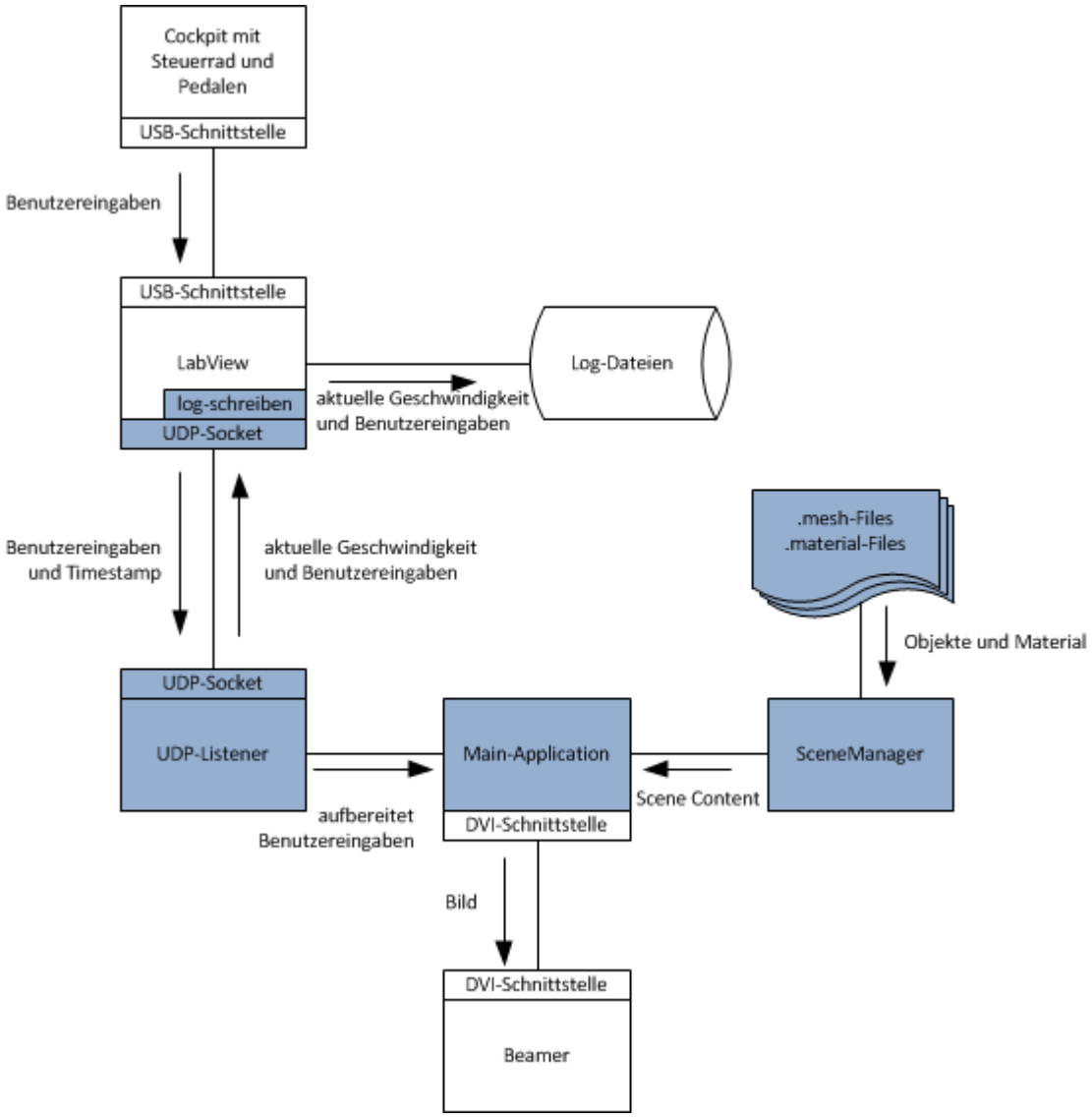

Abbildung 1: Systembeschreibung

Der Aufbau des Systems für den Fahrsimulator wird anhand der Abbildung 1 ilustriert. Die blau markierten Komponenten werden im Rahmen dieser Projekt Arbeit entwickelt. Alle übrigen sind bereits vorbestehend. Benutzereingaben, die im Cockpit gemacht werden, werden von einem LabView Programm eingelesen. Nun benötigt es einen UDP-Port, über den verschiedenen Eingaben an das Programm weitergeleitet werden. Es handelt

sich hierbei um Werte, die das Drehen des Steuerrades und den Druck auf Gas- oder Bremspedal quantifizieren. Zusätzlich wird der UDP-Port auch für das Empfangen diverser Log-Daten, die von unserem Programm gesendet werden, verwendet. Damit die Empfangenen Daten sauber in ein Log-File geschrieben werden, wird das LabVIEW Programm erweitert. Weiter muss in Programmiersprache C einen UDP-Socket mit entsprechendem UDP-Listener implementieren werden, um die Benutzereingaben zu empfangen. Gleichzeitig wird der UDP-Listener dazu verwendet die Geschwindigkeit des Fahrzeugs sowie Timestamps und weitere Daten an das LabVIEW Programm zurück zu schicken. Damit können die Daten gespeichert und später ausgewertet werden.
Diese Aufteilung durch eine Netzwerkschnittstelle ermöglicht es, das System, wenn notwendig, zu dezentralisieren. Einfachheitshalber wurde der UDP-Listener erst in einem Video-Beispiel implementier und getestet (Siehe Anhang B). Nachfolgend wird dieser UDP-Listener auch in das Programm des Fahrsimulator transferiert. Sind die Daten vom UDP-Listener empfangen und aufbereitet, werden sie im Hauptprogramm (Main Application) weiter verwendet. Während die Position des Steuerrades, des Gas und Bremspedales vom UDP-Listener permanent an das Hauptprogramm übertragen werden, wertet dieses die Positionen aus und veranlass die entspechenden Aktionen in der geladenen Szene. Die Szene selbst wird von einem Szenen-Manager geladen. Dieser benötig für die zu ladenden Objekte ein Meshfiles und mindestens ein Materialfile. Die Form jedes Objektes in der Szene wird durch ein separates Meshfile definiert. Texturen und Materialien werden durch ein oder mehrere Materialfiles beschrieben. Ein Materialfile kann zur Beschreibung unterschiedlicher Objekte verwendet werden. Die berechnete Szene wird schlussendlich in einem Fenster von Hauptprogramm angezeigt und über eine DVI-Schnittstelle an den Beamer übertragen. Der Beamer projeziert das Bild an die Wand, die sich direkt vor dem Cockpit befindet.

2.2. Anforderungen

2.2.1. Funktionale Anforderungen

- Der Proband kann das Fahrzeug im Fahrsimulator durch manipulation am Steuerrad und der Pedalen im Cockpit steuern.
- Die aktuelle Geschwindigkeit der Fahzeuges wird dem Probanden angezeigt.
- Es sollen zwei unterschiedliche Szenen zur verfügung gestellt werden. Die eine Szene sollte eine Stadt darstellen und die andere Szene eine Landschaft mit Tunnels.
- Die manipulationen des Benutzers und wichtige Parameter wie z.B. Geschwindigkeit sollen in einer Datei aufgezeichnet werden um diese auszuwerten.
- Alle ein- und ausgehenden Parameter des System sollen in LabVIEW verfügbar sein um diese auswerten und kontrollieren zu können.

2.2.2. Nicht Funktionale Anforderungen

- Das System soll robust sein.
- Das Starten des Fahrsimulator sollte möglichst einfach gehalten werden. Das Steuer des Fahrzeuges soll möglichst intuitiev sein wie man es sich von einem richtigen Fahrzeug gewöhnt ist.
- Der Fahrsimulator soll dem Probanden eine möglichst realistische Fahrsimulation bieten in der sich Strassen und verschiedene Objekte befinden.
- Die Raktionszeit des Systems soll möglichst klein sein. Die Verzögerung des Systems soll mess- und kalkulierbar sein.
- Das System soll möglichst Modular aufgebaut sein um später einfach erweitert werden zu können.
- Das System soll auf der existierender Hardware funktionieren, soll aber auch noch lauffähig sein wenn Teile des Fahrsimulators ausgetauscht werden.

3. Vorgehen und Methoden

Wie bereits in der Systembeschreibung beschrieben, wird der Fahrsimulator in 3 Hauptkomponenten unterteilt. Zum Fahrsimulator gehört ein UDP-Listener, ein Szenen-Manager und das Hauptprogramm. Da die Verbindung zwischen dem Cockpit und dem Fahrsimulator mit einer Netzwerkschnittstelle realisiert wird, ist es möglich die Anbindung an das Cockpit und den Fahrsimulator phisikalisch auf zwei Rechnern zu betreiben. Eine weiter Lösung hätte mit dem Ogre-Framework (Siehe Anhang D?) realisiert werden können. Dieses Framework bietet umfassende Lösungen um Steuerradär und Pedalen, wie sie in userem Cockpit vorhanden sind, anzusteueren.
Da die 3D-Umgebungen mit fortscheitendem Projekt zunimmt, könnte die Rechenleistung der momentan verwendeten Maschiene zu einem späteren Zeitpunkt nicht mehr ausreichen. Die Folgen von ungenügender Leistung können unregelmässige Bewegungen (Lags) in der virtuellen Umgebung sein oder es kann sogar bis zum Absturz des gesamten Programms führen.
Dieses Problem hat zur Folge, dass der Fahrsimulator einer hohen Portierbarkeitanforderung genügen muss. Damit ist die Variante mit der Netzwerkschnittstelle für das Fahrsimulatorprojekt geeignet. Die andere Variante wird verworfen.

3.1. LabView modifizierung

Es existiert bereits ein LabVIEW-Programm das die Eingaben im Cockpit einliest. Dieses Programm wird so erweitert das diese Eingaben in Form von definierten Parameter als UDP-Packet permanent gesendet werden. Zusätzlich wird ein zweiter UDP-Socket eingerichtet um Packete die vom Fahrsimulator gesendet werden zu empfangen. Die Daten der empfangenen Packete werden von LabVIEW in ein Log-File gespeichert.

3.2. UDP-Socket

Die Paramter werden vom LabVIEW-Programm permanent gelesuen und mindestens 50 mal pro Sekunde wird ein UDP-Packet gesendet. Um die Zeit zwischen der Eingabe des Probanden und dem ragierne des Fahrzeuges im Simulator möglichst tief zu halten, müssen ankommende Packete sofort gelesen werden. Unwichtig ist jedoch das jedes Packet empfange wird oder das die Packete in der gleichen Reihenfolge ankommen wie sie gesendet wurden. Das UDP-Protokoll gewährleistet genau diese Eigenschaften.

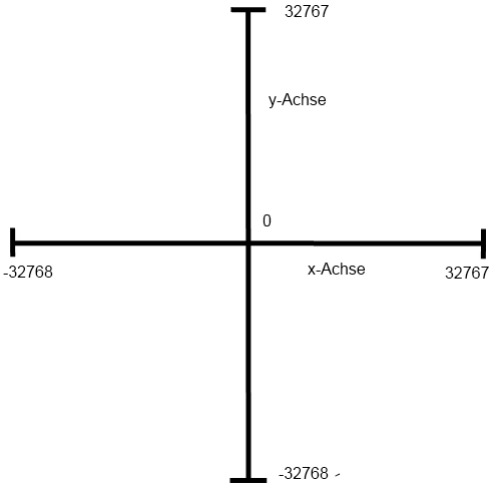

Abbildung 2: Koordinatensystem

Die Daten die vom LabVIEW empfangen werden, sind in erster Linie Parameter die den Zustand der Pedalen und des Steuerrades representieren. Die beiden Pedalen werden in einem Koordinatensystem wie in Abbildung 2 auf der y-Achse abgebildet. Die positive y-Richtung quantifiziert das Gas und die negative y-Richtung das Bremspedal. Die intensität beider Pedalen wird durch 32767 bzw. 32768 ganzzahlige Werte identifiziert. Wenn also keins der beiden Pedalen gedrückt ist, wird dies durch den y-Wert 0 dargestellt. Ein voll gedrücktes Gaspedal entspricht dem y-Wert 32767 und dementsprechend ein voll gedrücktes Bremspedal dem Wert -32768. Der x-Wert im Koordinatensystem quantifiziert den Einschlagswinkel des Steuerrades. Ist das Steuerrad in der neutralen Stellung, entspricht dies dem x-Wert 0. Wenn das Steuerrad vollständig nach rechts eingeschlagen ist, wird dies durch den positiven x-Wert 32767 identifiziert. Umgekehrt wird ein vollständiger Einschlag nach links durch den negiativen x-Wert -32768 identifiziert. Es können zusätzlich weitere Parameter, wie Tastendrücke am Steuerrad, übermittlet werden, wenn dies erforderlich sein sollte. Zusätzlich zu den Parametern des Cockpits wird von LabVIEW auch noch ein Timestamp gesendet. Der Timestamp wird von UDP-Listener empfangen und nach dem abspeichern der Parameter wieder zurückgeschikt. Somit kann festgestellt werden wieviel Verzögerung die Netzwerkschnittstelle verursacht.
Da das LabVIEW-Programm und der Simulator nicht synchronisiert sind, wird der DP-Listener dazu verwendet dies zu überbrücken. Der UDP-Listener speicher alle Parameter in Variabeln ab. Diese Variabeln werden vom Hauptprogramm ausgelesen und interpretiert. Dieses auslesen der Variabeln erfolgt nun synchron, da das Hauptprogramm nur dann Daten vom UDP-Listener liest wenn dieses dazu bereit ist.

3.3. Virtual Reality

3.4. Hautpprogramm

```
import java.io.Serializable;
public class Karte implements Serializable {

    // Instanzvariablen
    public Farbe farbe;
    public byte zahl;

    // Die vier Farben als Enumeration
    public enum Farbe {
        Rot, Gelb, Grün, Blau;
    }
}
```

listings/Karte.java

4. Resultate und Tests

4.1. Erreichtes

4.2. Zeitliches Verhalten des Systems

4.3. Testfälle

5. Nächste Schritte

5.1. Offene Punkte

5.2. Zusätzliche Funktionen

5.3. Ausblick auf Bachelor Arbeit 2012

6. Nachwort

6.1. Danksagung

7. Verzeichnisse

8. Abbildungsverzeichnis

Anhang

A. Aufgabenstellung

Aufgabenstellung der Projektarbeit PA11_frup_2

Herbstsemester 2011

Students: Sandro Ropelato (ropelsan@students.zhaw.ch), Christof Würmli (wurmlchr@students.zhaw.ch)
Industriepartner: Frau Rudy Ying-Yin Huang (yingyinhuang@ethz.ch, 044 63 22823)
ETH Zürich - MTEC - TIM - Research - Ergonomie der Informationsmedien
Scheuchzerstrasse 7, 8092 Zürich

Betreuer: Dr. Peter Früh (frup@zhaw.ch), Martin Schlup (spma@zhaw.ch)

Titel der Arbeit

Fahrsimulator mit realistischer virtuellen Umgebung

Ausgangslage

Im Rahmen einer grösseren Studie, soll das Verhalten diverser Autofahrer unter bestimmten reproduzierbaren Bedingungen, wie verschiedene Geschwindigkeiten oder Strassenverhältnisse, untersucht werden. Dazu steht an der ETH Zürich ein Fahrsimulator zur Verfügung, für den eine interaktiv steuerbare virtuelle Umgebung entwickelt werden soll. Die benötigten Strassenszenen sollen durch "Google Street View" (Google Earth) geliefert und durch diverse in die Landschaft eingefügte Objekte wie Fahrzeuge, Fussgänger oder Signalisationsschilder ergänzt werden, z.B. mit Hilfe der Google SketchUp-Software. Die Steuerelemente bestehen aus den üblichen Bedienelementen eines PWs (Lenkrad, Gas- und Bremspedale, usw.) welche über LabVIEW den Szenenablauf in "Echtzeit" steuern sollen. Die Nahtstellen zwischen den Bedienelementen und LabVIEW sind bereits vorhanden. Mit LabVIEW sollen auch die Betriebszustände und -abläufe des Simulators, sowie die eingegebenen Steuerbefehle registriert werden.

Zielsetzungen

1. Erzeugen einer virtuellen Umgebung (virtual reality) für den Fahrsimulator mit
   - interaktiv steuerbaren Strassenlandschaft basierend auf Google Street View
   - Synthetisieren und Einfügen von diversen Objekten in die „VR-Landschaft“, z.B. mittels Google SketchUp
2. Registrieren der Betriebszustände und -abläufe des Simulators, sowie der eingegebenen Steuerbefehle mit LabVIEW.

Beschreibung der Arbeit

- Situationsanalyse: Aufstellen der Anforderungen an die Computer-Graphik, Beschreibung der Nahtstellen und Abläufe, Auflistung der Steuersignale und Registrierdaten
- Realiserungsvorschläge, Übersicht über mögliche Verfahren
- Erarbeiten eines Lösungs- und Vorgehenskonzepts (die Reihenfolge der einzelnen Entwicklungsschritte ist frei wählbar)
- Realisierung der gewählten Lösung
- Analyse, Tests und Dokumentation
- gegebenenfalls Verbesserungen, Vorschläge

Weitere Einzelheiten werden in regelmässigen Besprechungen festgelegt.

Aufwand: entsprechend 2x6 ECTS, 300 bis 360 Mann-Stunden

Material

Ex: Instrument, Messgerät, Steuerung, Antrieb, usw.

- Software: LabVIEW, Google Tools
- PC
- Fahrsimulator

Bericht und weitere Dokumente

Im Rahmen der Arbeit sollen in einem Bericht folgende Punkte dokumentiert werden:

- Pflichtenheft und Spezifikation der VR
- Beschrieb der untersuchten Lösungsvarianten und Begründung der Wahl der Realisierungsvariante
- Detailbeschreibung der Implementierung mit Testergebnissen
- Bedienanleitung und Kurzbeschrieb der Programme
- offene Punkte und Ausblick

Termine

Kick-off Meeting: ab 19. Sept. 2011, nach Absprache in der KW 38
Besprechung: in der Regel einmal wöchentlich, nach Absprache
Präsentation der Arbeit: in der KW50 oder 51
Abgabe des Berichtes: 23. Dez. 2011

B. Video Player

B.1. Ziel

In einem ersten Schritt, noch vor der Realisierung des Fahrsimulator wird ein Video-Player ralisiert. Das Ziel des Video-Players ist es eine Verbindung zwischen dem Cockpit und unserem Programm herzustellen und diese zu Testen. Bei der Betätigung des Gaspedals im Cockpit soll das aufgenommene Video schneller abgespielt werden. Bei der Betätigung des Bremspedals dementsprechend langsamer. Der Video-Player soll so gegliedert werden, damit Komponenten davon auch im Fahrsimulator wiederverwendet werden können. Zudem ist die Darstellung einer Tunneleinfahrt mit den richtigen Lichtverhältnissen in einer 3D-Umgebung sehr schwierig. Eine gute Alternative bietet ein aufgenommenes Video einer Tunneleinfahrt. Die ETH-Zürich besitzt bereits Videos die sich dafür eigenen. Um Experimente mit diesen Videos durchführen zu können, müssen Manipulationen die der Proband im Cockpit mach mit dem entsprechenden Zeitpunkt im Video abgespeichert werden. Dies dient zur späteren Auswertung der Experimente.

B.2. Systembeschreibung

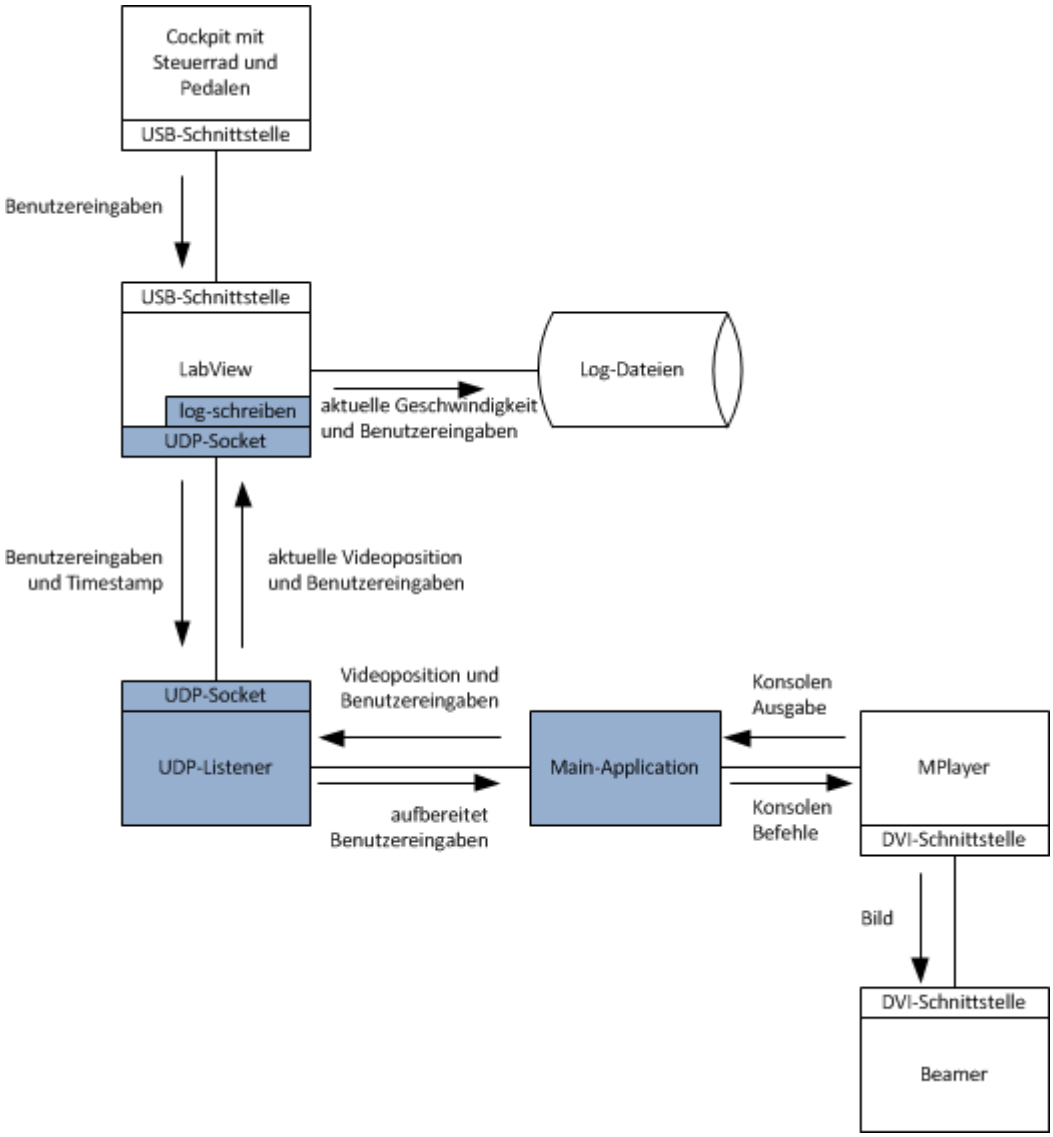

Abbildung 3: Systembeschreibung Video-Player

Um einzelne Komponenten des Video-Player wiederverwenden zu können, wird das System möglichst gleich wie das des Fahrsimulators aufgebaut. Dieser Aufbau wird anhand der Abbildung 3 illustriert. Die blau markierten Komponenten werden im Rahmen des Video-Players entwickelt. Alle übrigen sind bereits vorbestehend.
Die LabVIEW-Komponente in der Abbildung 3 liest bereits die Eingaben die der Proband im Cockpit mach ein. Diese wird mit einem UDP-Socket erweitert um die Eingaben des Probanden an den Video-Player weiterzuleiten. Die UPD-Packete sollen vom UDP-

Listener gelesen werden und die in ihnen enthaltenen Parameter gespeichert werden. Das Hauptprogramm soll auf die gespeicherten Parameter Zugriff haben. Aufgrund der Parameter soll das Hauptprogramm die Abspielgeschwindigkeit des Videos manipulieren, das mit einem Programm abgespielt wird. Durch mehere Versuche hat sich gezeigt das sich der MPlayer am besten für solch eine Aufgabe eignet. Dies wird in der Ralisierung ausführlich dokumentiert. Der MPlayer soll die Position im Video an das Hauptprogramm übergeben können. Das Hauptprogramm soll diese Information der Video-Position mit den getätigten Eingaben des Probanden ergänzen und dies an den UDP-Listener übergeben. Über den UDP-Sockes sollen die Daten an das LabVIEW-Programm gesendet werden. Dort muss zusätzlich ein Port eingerichtet werden um UDP-Packete zu empfangen.

Eine wird erweitert wie in Kapitel 3.1 dokumentiert.

Die LabVIEW-Komponente in der Abbildung 3 liest bereits die Eingaben die der Proband im Cockpit mach ein. Dieses muss nun so erweitert werden, dass diese über einen UDP-Socket an das Programm übertragen werden. Zusätzlich muss im LabVIEW ein UDP-Port eingerichtet werden, der verwendet werden kann, um UDP-Packete zu empfangen. Dieser wird benötigt um Daten die von unserem Programm gesendet werden in ein Log-File zu schreiben. Die Parameter werden vom vom UDP-Listener gespeichert und vom Hauptprogramm abgefragt. Aufgrund der Parameter wird dann die Geschwindigkeit des Videos manipuliert. Für das Abspielen des Videos wird der mPlayer verwendet. Die Befehle für den mPlayer können von unserem Programm über die Komandozeile abgesetzt werden. Ausgaben vom mPlayer werden ebenfalls über die Komandozeile den Standartausgang der Konsole übergeben.

B.3. Realisierung

In einem bestehendem LabVIEW-Programm werden bereits alle Eingaben, die im Cockpit gemacht werden können, eingelesen. Nun muss dieses Programm nur noch mit einem UDP-Port erweitert werden, damit es die Parameter, die unser Video-Player benötigt, senden kann. Diese Daten sind im Video-Player vor allem die betätigung von Gas- und Bremspedal. Diese beiden Pedalen werden in einem Koordinatensystem wie in Abbildung 4 auf der y-Achse abgebildet. Die positive y-Richtung verifiziert das Gas und die negative y-Richtung das Bremspedal. Die intensität beider Pedalen wird durch 32767 bzw. 32768 ganzzahlige Werte identifiziert. Wenn also keins der beiden Pedalen gedrückt ist, wird dies durch den y-Wert 0 verifiziert. Ein voll gedrücktes Gaspedal entspricht dem y-Wert 32767 und dementsprechend ein voll gedrücktes Bremspedal dem Wert -32768. Der x-Wert im Koordinatensystem verifiziert den Einschlagswinkel des Steuerrades. Ist das Steuerrad in der neutralen Stellung, entspricht dies dem x-Wert null. Wenn das Steuerrad vollständig nach rechts eingeschlagen ist, wird dies durch den positiven x-Wert 32767 identifiziert. Dementsprchend wird ein vollständiger Einschlag nach links durch den negiativen x-Wert -32768 identifiziert.

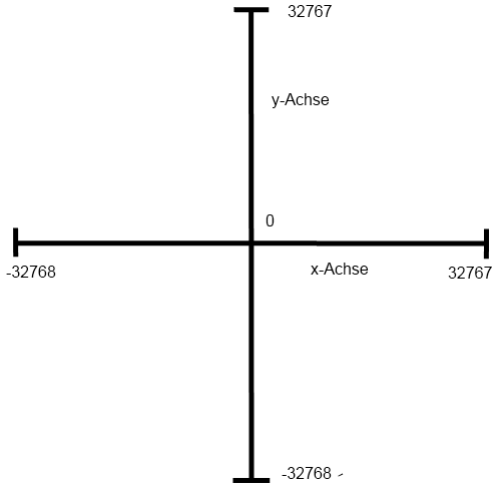

Abbildung 4: Koordinatensystem

Diese beiden Werte werden im LabVIEW-Programm in ein UDP-Packet verpackt und über das Netzwerk gesendet. Der UDP-Listener empfängt diese Packete und speichert die x und y-Werte separat ab. Das Hauptprogramm greift auf diese Werte zu, interpretiert diese und Steuert das Video.
Der Video Player wird in einem eigenen Prozess gestartet. Die erste Zeile speichert alle Argumente in der Variabel ärguments". Durch die mitgegebenen Argumente wird der Video Player mit einem extra Telnetinterface gestartet. Das Telnetinterface hört auf den angegebenen Port. In der zweiten Zeile wird der Prozess erstellt und in diesem Prozess wird der Video Player ausgeführt. Nun können Manipulationsbefehle über den ßend"Befehl an den mPlayer geschickt werden. Hier ein Beispiel eines solchen Befehls. Der send-Befehl verlangt eine Socket-Id, die Nachricht, in unserem Fall den Befehl, und die Länge der Nachricht. Das Hauptprogramm liest nun aus dem UDP-Listener den y-Wert der empfangenen Packete aus. Liegt dieser Wert über 20000 ist das Gaspedal mindestens zu zwei drittel gedrückt. Dann sendet das Hauptprogramm über die Telnetverbindung den Befehl an den Video Player die Geschwindigketi des Videos zu erhöhen. Fall der Wert unter -20000 liegt, ist das Bremspedal mindestens zu zwei drittel gedrückt. Das Hauptprogramm veranlasst den Videoplayer die Geschwindigkeit des Videos u reduzieren.

C. Das OGRE-Framework

C.1. Was ist das OGRE-Framework

OGRE (Object-Oriented Graphics Rendering Engine) ist eine Grafikengine zur Echtzeitdarstellung von dreidimensionalen Szenen. Sie ist in C++ geschrieben und ihre Verwendung unterliegt der MIT-Lizenz. Durch den modularen Aufbau und die Unterstützung auf verschiedenen Plattformen erweist sich OGRE als sehr flexibel und mächtig. OGRE selbst benutzt die Grafikbibliotheken OpenGL und DirectX zum hardwarebeschleunigten Rendern der Szenen und setzt automatisierte Optimierungsalgorithmen zur Geschwindigkeitsgewinnung ein. Mittlerweile existiert eine grosse und aktive Community, welche das OGRE-Framework ständig weiter entwickelt und verbessert.

Abbildung 5: OGRE Logo

C.2. Weshalb setzen wir OGRE ein?

Im Vergleich zur direkten Verwendung von OpenGL oder DirectX bringt der Einsatz einer Grafikengine wie OGRE eine Vielzahl von Vorteilen mit sich:

Abstraktion
OpenGL und DirectX sind zwei komplett verschiedene Bibliotheken. Hat man sich für eine von beiden entschieden, so bedeutet das Umsteigen auf eine andere unter Umständen das Neuschreiben des kompletten Codes. Ogre abstrahiert die Verwendung dieser Grafikbibliotheken und ermöglicht den selben Code entweder mit OpenGL oder DirectX laufen zu lassen.

Geschwindigkeit
text

D. Modelling Tools

E. Setup

F. Listings

G. Detailierter Zeitplan

H. Glossar

Lab-View Lab-View Programm Log-Datei Cockpit