# Abschlussbericht zur Gruppenarbeit Hololens-Tour

## Abstract

Unsere Gruppenarbeit begann mit dem Ziel, für die Hololens ein Framework zu programmieren, welches es ermöglichen sollte, eine geführte Tour mittels Bilderkennung zu implementieren.

In diesem Bericht fassen wir unsere Ergebnisse zusammen, beleuchten welche Probleme und Schwierigkeiten auftraten und welche Lösung wir für diese fanden.

Der Ablauf unserer Arbeit lässt sich folgendermassen zusammenfassen: Zuerst haben wir die nötige Programmierumgebung installiert, die Ziele konkretisiert und einen Plan erstellt. Als zweites haben wir Recherche betrieben, welche Bibliotheken/Tools bereits vorhanden waren und welche davon für unseren Use-Case in Frage kamen. Als dritten Schritt haben wir unser für das Actor-Modell entschieden und eine entsprechende Framework-Struktur entworfen. Im letzten Schritt haben wir unser Framework implementiert.

## Use Case

TODO:Kusi

## Übersicht über Bibliotheken und Tools

Für unser Projekt haben wir folgende Tools näher in Betracht gezogen:

* Bilderkennung
  + Vuforia
  + EmguCV
  + OpenCVSharp
  + ARToolkit
* Actorimplementierung
  + Akka.NET
  + Proto
* Darstellung
  + Unity
  + Xamarin

Im Folgenden gehen wir näher auf unsere Ergebnisse mit den einzelnen Tools ein.

### Bilderkennung

Für den ersten Mini Use Case war es nötig, dass wir uns für eine Bibliothek entscheiden, die die Erkennung von Aruco-Markern effizient und zuverlässig implementiert. Dazu sollten die Tools möglichst offenen Quellcode haben und frei verfügbar sein.

#### ARToolkit

Als erstes haben wir versucht, basierend auf einem bereits existierenden Projekt, das ARToolkit, eine speziell für die Hololens angepasste Version eines OpenCV-Wrappers, zu verwenden. Die dabei entstandenen Probleme bezogen sich primär auf das Testing: Da wir unser Framework nicht auf der Hololens testeten und die Bibliothek erst zur Laufzeit eingebunden wurde, wäre es für uns nicht ohne grossen Aufwand möglich gewesen, mit dieser Bibliothek testorientiert zu arbeiten.

#### Vuforia

Zu Beginn unserer Gruppenarbeit gab es noch keine native Unity-Unterstützung von Vuforia. Nachdem im Unity 2017.2 Update Vuforia standardmässig und ohne grosse Umstände mit Unity verwendbar wurde, haben wir die Bibliothek ebenfalls in Betracht gezogen. Leider stellte sich heraus, dass Vuforia in seiner kostenfreien Version nicht von Unity getrennt funktioniert. Da unser Ziel aber war, ein plattformunabhängiges, sprich nicht an Unity gebundenes, Framework zu entwickeln entschieden wir uns gegen weitergehende Recherche zu Vuforia.

#### OpenCVSharp

OpenCVSharp war unsere vorletzte Anlaufstelle, nachdem alle vorherigen Bibliotheken sich aus verschiedenen Gründen als unzulänglich erwiesen hatten.

OpenCVSharp stellt eine Portierung der bekannten OpenCV-Bibliothek auf C# zur Verfügung. Leider fiel uns früh auf, dass OpenCVSharp die Funktionalität der Aruco-Markererkennung nur teilweise implementiert hat. Da uns das implementieren eines Wrappers für ebendiese Funktionen viel Zeit gekostet hätte, entschieden wir uns gegen OpenCVSharp. Vorteil von OpenCVSharp ist, dass es eine vergleichsweise günstige Unity-Version gibt.

#### EmguCV

EmguCV ist die Bibliothek für die wir uns für die Implementierung des Durchstichs entschieden haben. EmguCV bietet die volle Funktionalität von OpenCV.Aruco und deckt damit den Bedarf von unserem Use Case ab. Des Weiteren ist OpenCV im Bereich Computer Vision eines der verbreitetsten und meist genutzten Tools.

### Actorimplementierung

Das Schwergewicht unserer Entwicklungsarbeit lag auf dem Aufbau eines Actor-Frameworks, welches um weitere Module wie Positionserkennung und ähnlichem ergänzt werden kann. Dafür war es nötig, dass wir eine Bibliothek wählen, welche das Actor-Modell implementiert und dazu noch möglichst wenig Overhead produziert.

#### Akka.NET

Unsere erste Recherche ergab, dass Akka.NET, eine .NET kompatible Implementierung des Actor Modells, sich eignen könnte. Nach langwierigem Einarbeiten und testen der Bibliothek fiel uns auf, dass Akka.NET aufgrund seiner Vielzahl von Abhängigkeiten (System.Threading.Thread ,…) nicht mit der aktuellen Version von Unity (2017.2) und dem Framework der Hololens kompatibel ist. Gemäss Entwicklerforen[[1]](#footnote-1) ist eine .NET Core Portierung allerdings in Entwicklung. Grosser Vorteil von Akka ist das mitgelieferte TestKit, dass die Entwicklung von Tests stark vereinfacht.

#### Proto

Proto ist eine Actor Bibliothek von jenem Entwickler, der auch Akka.NET initiiert hat. Proto liefert mehr Funktionalität als Akka bei gleichzeitigem Verzicht auf viele der Dependencies, die Akka für uns unbrauchbar machten. Einziges Manko bei Proto ist, dass kein TestKit implementiert ist, was das Testing schwierig gestaltete, da der interne Zustand von Actors von aussen nicht eingesehen werden kann.

### Darstellung

Die Darstellung der den Markern zugeordneten Objekten wollten wir mit Hilfe eines Programms verwalten. Als erste und zu unserem Zeitpunkt einzige brauchbare Umgebung stand uns dafür Unity zur Verfügung.

#### Unity

TODO: Selim

#### Xamarin

TODO: Selim

## Probleme in der Entwicklung

Die Entwicklung unsere Frameworks war von Beginn an von Inkompatibilitäten geprägt. Unsere erste Wahl für die Implementierung des Actor Frameworks, Akka.NET, war nicht mit den Anforderungen der Hololens an das Framework, UWP bzw. .NET Core 2.0, kompatibel. Entsprechend mussten wir uns umorientieren und sind nach kurzer Zeit auf Proto gestossen. Proto war zwar zu dem Zeitpunkt noch in einer Betaphase, allerdings war die Funktionalität, die bereits implementiert war, gemäss Entwicklern stabil. Protos Abhängigkeiten sind so gestaltet, dass Proto auch mit .NET Core 1.6 kompatibel ist und entsprechend auf der Hololens lauffähig sein sollte.

Bei der Implementierung der Frameworks sind wir auf ein weiteres Problem gestossen. Die aktuelle Version, .NET Standard 2.0, hat aufgrund von GDI+ Abhängigkeiten keine Implementation von System.Drawing. Da man allerdings Bilddateien zur Bilderkennung benötigt, mussten wir um diese Beschränkung herum Konvertierungsfunktionen schreiben um dennoch weiterarbeiten zu können.

Als zweite grosse Problemquelle trat Unity auf. Da Unity zum Zeitpunkt unserer Arbeit mit dem Mono-Framework arbeitete, waren gewisse Schwierigkeiten bereits vorprogrammiert. Leider ergaben sich auch nicht nachvollziehbare Kompatibilitätsprobleme: So konnten wir zum Beispiel die Funktion String.Copy nicht mit Unity verwenden, da Unity der Meinung war, dass String.Copy nicht auf der Hololens lauffähig sein würde. Es bleibt daher abzuwarten, ob in künftigen Versionen von Unity solche Kompatibilitätsprobleme gelöst werden können.

Probleme mit Unity TODO: Selim

OpenCVSharp & EmguCV TODO: Kusi

## Ergebnisse der Arbeit

Als Endprodukt haben wir das funktionierende Framework, Ansätze für die Portierung auf Unity und eine WPF-GUI zur Darstellung der Ergebnisse, sowie die zugehörige Dokumentation. Leider ist uns die vollständige Implementierung der Tour aufgrund der Probleme mit Unity nicht gelungen.

1. https://github.com/akkadotnet/akka.net/issues/2153 [↑](#footnote-ref-1)