|  |
| --- |
| Hololens Tour |
| Abschlussbericht |
|  |
| **Datum** 11. Dezember 2017  **Autor(en)** Markus Kiser, Moritz Scherer, Selim Naji |

Inhaltsverzeichnis

[Inhaltsverzeichnis ii](#_Toc500776125)

[1.1. Abstract 3](#_Toc500776126)

[1.2. Use Case 3](#_Toc500776127)

[1.3. Systemarchitektur 4](#_Toc500776128)

[1.3.1. Actors 4](#_Toc500776129)

[1.3.2. Nachrichten 4](#_Toc500776130)

[1.3.3. Initialisierung 4](#_Toc500776131)

[1.3.4. ControlActor 5](#_Toc500776132)

[1.3.5. RecognitionManager 5](#_Toc500776133)

[1.3.6. SyncActor 5](#_Toc500776134)

[1.3.7. CameraFeedActor 5](#_Toc500776135)

[1.3.8. Erkennungsablauf 5](#_Toc500776136)

[1.4. Übersicht über Bibliotheken und Tools 6](#_Toc500776137)

[1.4.1. Bilderkennung 6](#_Toc500776138)

[1.4.2. Actorimplementierung 7](#_Toc500776139)

[1.4.3. Darstellung 7](#_Toc500776140)

[1.5. Probleme in der Entwicklung 7](#_Toc500776141)

[1.6. Ergebnisse der Arbeit 8](#_Toc500776142)

[1.7. Ausblick 8](#_Toc500776143)

## 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

Die Idee war eine Applikation zu kreieren, die auf der HoloLens läuft. Dabei sollte es gelingen, mit der AR - Unterstützung der HoloLens eine «Guided Tour through SCS» zu erleben. Dabei werden reale, ausgedruckte Aruco Code Marker in den Räumlichkeiten der SCS an spezifischen Orten platziert. Der User sollte dann mit der HoloLens auf dem Kopf herumspazieren und jedes Mal, wenn ein Marker im Blickfeld der HoloLens auftaucht, sollte eine von den Entwicklern definierte Aktion eintreten. Dies könnte z.B. beim Firmeneingang ein Pop – Up Begrüssungsvideo von Herrn Gunzinger sein. Ein mögliches Szenario von solch einer «Guided Tour through SCS» könnte wie folgt aussehen:

* Ein Kunde steht im Eingangsbereich und meldet sich bei der Rezeption. Diese empfängt ihn herzlich und übergibt ihm die HoloLens mit der Bitte sich diese doch aufzusetzen.
* Sobald der Kunde die HoloLens auf dem Kopf sitzend hat, sollte die App bereits laufen.
* Als erstes soll ein kleines Tutorial für die Gestensteuerung laufen. Danach soll ein kurzes Infovideo folgen, dass den nachstehenden Ablauf kurz erläutert.
* Danach sollte eine warmherzige Willkommensrede von Herrn Gunzinger per Video folgen, die den Kunden via virtuelle Wegweiser zum Christof Bühler navigieren sollte. Auf dem Weg sollten verschiedenen Zusatzinformationen, wie etwa die Benennung der verschiedenen Departments der SCS, die Beschriftung der Büros des Chefs und der Finanzabteilung oder die Anzahl der momentanen Mitarbeiter, um nur ein paar mögliche aufzuzählen, aufpoppen.
* Beim Departement Life Science & Physics angelangt, taucht ein blinkender Pfeil über dem Pult vom Christof Bühler auf, sodass gerade klar wird, wen der Kunde ansprechen sollte.
* Danach geht’s zurück zur Rezeption mit der Begleitung von Christof Bühler. Die eigentliche Tour beginnt nun. Christof erzählt über die Geschichte der SCS und nimmt Bezug auf das Mobiliar des SCS – Museums. Dabei sollten immer wieder unterstützende Informationen zu den angesprochenen Geschichten eingeblendet werden.
* Danach geht die Führung wieder den Wegweisern nach und wird im gleichen Stil gehalten, das heisst, dass zu den erwähnten Produkten Zusatzinformationen eingeblendet werden. Es können dies Bilder, Ton, Videos, oder 3D – Modelle sein. Der Fantasie sollte dabei keine Grenzen gesetzt sein.
* Die Verabschiedung erfolgt wieder bei der Rezeption mit einer kurz und knackigen Abschlussmessage vom Chef persönlich. Danach wird die HoloLens abgegeben und der Kunde wird dann schlussendlich noch real und persönlich von Christof verabschiedet.

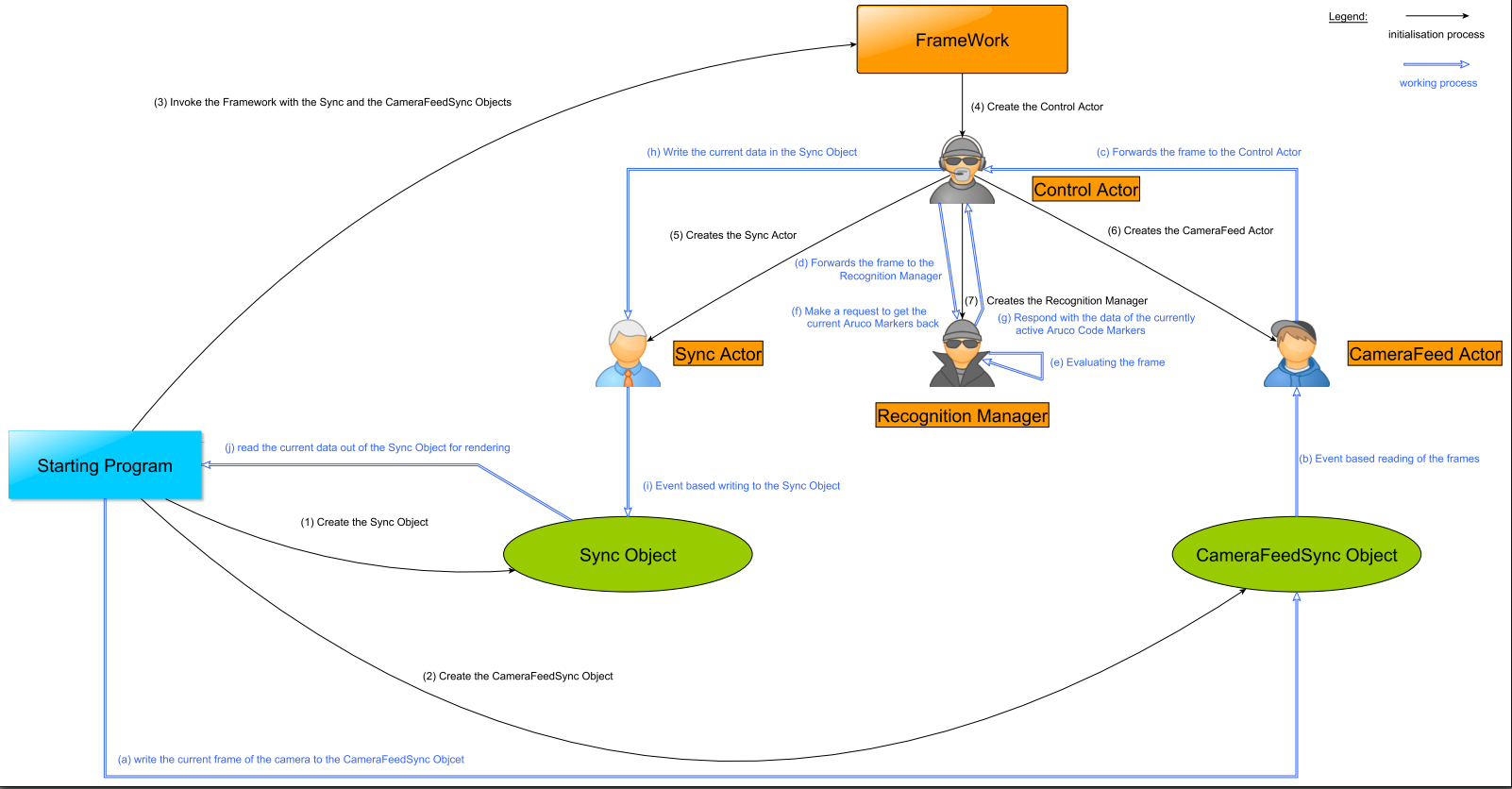
## Systemarchitektur

### Actors

Actors sind das Kernstück unseres Systems. Im Wesentlichen sind Actors einzelne «Threads», die untereinander Nachrichten verschicken können. Dadurch lassen sich einzelne Aufgaben oder Module als Akteure modellieren. Zum Beispiel haben wir den RecognitionManager der Aufgabe der Erkennung von Markern auf einem Bild nach modelliert. Der RecognitionManager wird über gesteuert, indem er die Nachrichten NewFrameArrived oder RequestAllVirtualObjects erhält.

### Nachrichten

In unserer Implementierung existiert für jede Nachrichtenart, zum Beispiel NewFrameArrived, eine eigene Klasse. Damit ist sichergestellt, dass die Daten jedes Mal im gleichen Format übertragen werden, vorausgesetzt, dass der Nachrichtenkonstruktor korrekt verwendet wurde.



Systemarchitektur

### Initialisierung

Zur Initialisierung erstellt der Enduser jeweils ein SyncObject und ein CameraFeedSyncObject und gibt ein Dictionary gemäss Emgu.CV an, das die zu erkennenden Marker beinhaltet.

Dann erstellt der Enduser ein neues Objekt der Klasse Framework mit den erstellten Objekten auf und ruft darauf die Funktion Initialize auf, was das Framework startet. Danach «füttert» er die einzelnen Frames in des CameraFeedSyncObject und erhält die Daten aus dem SyncObject.

### ControlActor

Der ControlActor kümmert sich um den generellen Datenfluss. Er erhält von angebundenen Modulen Nachrichten, die es ihm ermöglichen den Kontroll- und Datenfluss zu steuern. Die Anbindung von neuen Modulen mittels Actors ist möglich, indem man in der asynchronen Funktion ReceiveAsync eine neue Nachricht erwartet.

### RecognitionManager

Der RecognitionManager kümmert sich um das Erkennen von ArucoCodes mittels EmguCV. Er verwaltet dabei die erkannten Marker in einem Dictionary. Auf Anfrage liefert er dieses Dictionary zurück.

### SyncActor

Der SyncActor stellt die Daten aus der Auswertung eines Frames bereit. Auf dem von ihm verwalteten SyncObject kann dann der Enduser die Daten einsehen/kopieren.

### CameraFeedActor

Der CameraFeedActor ist dafür verantwortlich neue Frames zu registrieren und dem ControlActor über deren Ankunft Auskunft zu erteilen. Dem Actormodell entsprechend sendet er dem ControlActor dafür eine Nachricht.

### Erkennungsablauf

Der grundsätzliche Ablauf ist folgender:

Der Enduser erstellt ein SyncObject und ein CameraFeedSyncObject und definiert die zu erkennenden Marker. Daraufhin erstellt er mit diesen Daten eine Instanz der Klasse Framework und ruft die Initialize-Funktion auf.

Ein neuer Frame wird auf das CameraFeedSyncObject geladen. Daraufhin wird die Funktion UpdateFrame vom Enduser aufgerufen. Das löst beim CameraFeedActor das Event OnFrameUpdated aus.

Der CameraFeedActor löst bei diesem Event eine Nachricht NewFrameArrived beim ControlActor aus, der beim Erhalt der Nachricht einen Request mit einem neuen NewFrameArrived an den RecognitionManager startet.

Der RecognitionManager startet die FrameEvaluation beim Erhalt von NewFrameArrived und updated sein Dictionary den neuen Daten entsprechend und informiert den ControlActor mit der Nachricht RespondNewFrameArrived, dass er die Auswertung vorgenommen hat.

Der ControlActor fragt daraufhin mit RequestAllVirtualObjects das Dictionary des RecognitonManagers ab, der mit RespondRequestAllVirtualObjects eben dieses zurückgibt.

Der ControlActor sendet beim Erhalt von RespondRequestAllVirtualObjects eine neue Nachricht WriteCurrentTourState an den SyncActor.

Der SyncActor updated das SyncObject beim Erhalt der Nachricht WriteCurrentTourState und antwortet dem ControlActor mit RespondWriteCurrentTourState. Damit ist die Verarbeitung eines Frames abgeschlossen.

## 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**

Weitere Probleme hatten wir mit OpenCvSharp. Die Bilderkennungsbibliothek hatte zwar ein Aruco Code Erkennungsmodul, doch war dieses nicht vollständig. Die Funktion DetectMarkers() konnte man verwenden, wenn man das NuGet package ins Projekt eingebunden hatte, doch die Funktion EstimatePoseSingleMarkers() war nicht aufzufinden. Denn in der CvAruco Klasse war nur die DetectMarkers() Funktion drin, siehe <https://github.com/shimat/opencvsharp/blob/master/src/OpenCvSharp/Modules/aruco/CvAruco.cs>, nicht aber die EstimatePoseSingleMarkers(), die für unser Use Case zentral und unerlässlich ist. In einer neueren Version von OpenCvSharp als die 3.3.1 könnte sie jedoch nachgeführt sein.

Am Ende hatten wir noch ein paar Hürden mit EmguCV zu überwinden. Angefangen hat es mit den Warnungen der Nichtverwendbarkeit von EmguCV nach Download des NuGets Packets. Dies umgingen wir mit einem direkten Verweis auf die EmguCV - dll, die man auf Github finden und downloaden kann. Weiter waren dann zwar beide Funktionen (siehe Abschnitt oben), die wir brauchten, da und funktionsfähig, aber die Funktionen und ihre Argumente mit den Datentypen von EmguCv sind sehr schlecht im Netz dokumentiert. Besonders der Typ Mat bringt für Neulinge Schwierigkeiten mit sich. In unserem Framework sind die Schritte so genau wie möglich beschrieben, wie man z.B. eine Bitmap in ein Image umschreiben kann (dafür mussten wir noch selbst einige Hilfsfunktionen schreiben, siehe Utils Funktionen) oder wie man Daten aus einem Mat herausliest. Dafür suche man doch im Code des Recognition Managers und in den Utils Funktionen und deren Tests nach den genauen von uns gefundenen Lösungen zu diesen Problemen.

## Ergebnisse der Arbeit

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

## Ausblick

Mit dem Update 2018.1 von Unity sollte .NET Standard 2.0 unterstützt werden und somit unser Framework lauffähig werden. Aufgrund der Struktur unseres Framework sollte es problemlos möglich sein, weitere Module wie Positionserkennung oder Userinputverarbeitung als Actors einzufügen.

TODO: Selim/Kusi

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