|  |
| --- |
| Hololens Tour |
| Abschlussbericht |
|  |
| **Datum** 22. Dezember 2017  **Autor(en)** Markus Kiser, Moritz Scherer, Selim Naji |

**Inhaltsverzeichnis**

[1. Einleitung 1](#_Toc502763211)

[1.1. Abstract 1](#_Toc502763212)

[1.2. Use Case 2](#_Toc502763213)

[1.3. Test-Driven-Design 3](#_Toc502763214)

[2. Systemarchitektur 4](#_Toc502763215)

[2.1. .NET Implementationen 4](#_Toc502763216)

[2.1.1. .NET Framework 4](#_Toc502763217)

[2.1.2. .NET Standard 4](#_Toc502763218)

[2.1.3. UWP 4](#_Toc502763219)

[2.2. Systemdesign 5](#_Toc502763220)

[2.3. Actors 5](#_Toc502763221)

[2.3.1. Nachrichten 5](#_Toc502763222)

[2.3.2. Initialisierung 5](#_Toc502763223)

[2.3.3. ControlActor 5](#_Toc502763224)

[2.3.4. RecognitionManager 6](#_Toc502763225)

[2.3.5. SyncActor 6](#_Toc502763226)

[2.3.6. CameraFeedActor 6](#_Toc502763227)

[2.3.7. Erkennungsablauf 6](#_Toc502763228)

[2.4. Übersicht über Bibliotheken und Tools 7](#_Toc502763229)

[2.5. Markers 8](#_Toc502763230)

[2.5.1. VuMark 8](#_Toc502763231)

[2.5.2. Apriltag 8](#_Toc502763232)

[2.5.3. Barcode 8](#_Toc502763233)

[2.5.4. QR Codes 8](#_Toc502763234)

[2.5.5. ARToolKit Marker 9](#_Toc502763235)

[2.5.6. Aruco Marker 9](#_Toc502763236)

[2.6. Bilderkennung 9](#_Toc502763237)

[2.6.1. ARToolkit 9](#_Toc502763238)

[2.6.2. Vuforia 9](#_Toc502763239)

[2.6.3. OpenCVSharp 10](#_Toc502763240)

[2.6.4. EmguCV 10](#_Toc502763241)

[2.7. Actor - Implementierung 10](#_Toc502763242)

[2.7.1. Akka.NET 10](#_Toc502763243)

[2.7.2. Proto 11](#_Toc502763244)

[2.8. Darstellung 11](#_Toc502763245)

[2.8.1. Xamarin 11](#_Toc502763246)

[2.8.2. Unity 11](#_Toc502763247)

[2.8.3. UWP UI 11](#_Toc502763248)

[2.9. Probleme in der Entwicklung 12](#_Toc502763249)

[2.9.1. Probleme mit dem Framework 12](#_Toc502763250)

[2.9.2. Probleme mit Unity 12](#_Toc502763251)

[2.9.3. Probleme mit den Bilderkennungsbibliotheken 13](#_Toc502763252)

[3. Ergebnisse der Arbeit 14](#_Toc502763253)

[3.1. Ausblick 15](#_Toc502763254)

[4. Anhang 1](#_Toc502763255)

[4.1. Abbildungsverzeichnis 1](#_Toc502763256)

# Einleitung

## Abstract

Im Rahmen unserer Gruppenarbeit haben wir den Auftrag bekommen, für die SCS ein Framework für eine virtuelle Tour durch die Büroräumlichkeiten zu programmieren und uns mit den technischen Hürden der Augmented Reality vertraut zu machen.

In diesem Bericht fassen wir unsere Ergebnisse zusammen, beleuchten die aufgetretenen Schwierigkeiten und beschreiben die Lösungen, die wir dafür fanden.   
Zuerst haben wir die nötige Programmierumgebung installiert, die Ziele konkretisiert und einen Plan erstellt. Als zweites haben wir Recherche betrieben, welche Bibliotheken bzw. Tools bereits vorhanden waren und welche davon für unseren Use-Case in Frage kamen. Als dritten Schritt haben wir uns für das Actor-Modell entschieden und eine entsprechende Framework-Struktur entworfen. Im letzten Schritt haben wir unser Framework implementiert.

Das Framework lief zum Abgabezeitpunkt noch nicht auf der Microsoft Hololens, da die dort verwendete Version von .Net Core noch nicht fortgeschritten genug war.   
Dennoch ist es uns gelungen ein Proof of Concept zu erstellen, dass mit wenig Mehraufwand auf die Hololens portierbar sein wird, sobald das Update eintrifft.

## Use Case

Die Idee war eine Applikation zu kreieren, die auf der HoloLens[[1]](#footnote-1) 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 vom Firmenchef 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 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 vom Chef der Firma per Video folgen, die den Kunden via virtuelle Wegweiser zur gesuchten Person 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.
* Wenn die zu suchende Person zum Beispiel im Departement «Life Science & Physics» arbeitet, sollte eine virtuelle Navigation mit Pfeilen den Kunden dorthin führen. Dort angelangt, taucht ein blinkender Pfeil über dem Pult der zu suchenden Person auf, sodass gerade klar wird, wen der Kunde ansprechen sollte.
* Danach geht’s zurück zur Rezeption mit der Begleitung von der nun gefundenen Person. Die eigentliche Tour beginnt nun. Die Person erzählt über die Geschichte der SCS und nimmt Bezug auf das Mobiliar des SCS – Museums. Dabei sollten am Kunden 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 Projekten Zusatzinformationen eingeblendet werden. Es können dies Bilder, Ton, Videos, oder 3D – Modelle sein. Der Fantasie sollen 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 verabschiedet.

Uns ist klar, dass dieser Use-Case etwas konstruiert daherkommt, da man natürlich Kunden nach Möglichkeit persönlich begrüssen und herumführen möchte. Die Tour sollte eher zur Überbrückung der Wartezeit dienen und dabei ein spannendes Technikprojekt der SCS präsentieren.

## Test-Driven-Design

Unser Framework programmierten wir mit der TDD Methode. Die Abkürzung TDD, die für Test Driven Development steht, wird oft in der agilen Entwicklung von Computerprogrammen verwendet. Dabei werden die Software-Tests immer vor den zu testenden Modulen implementiert. Man programmiert nur das Nötigste, sodass der geschriebene Test grün aufleuchtet. Dadurch ist die zu erfüllende Funktion garantiert und redundanten Code gibt es nicht mehr. Dies führt zu einer höheren Codequalität und zu einer besseren gedanklichen Strukturierung des Problems[[2]](#footnote-2).

Getestet wird jedes Modul einzeln in sogenannten Unit Tests. Dabei wird jede noch so kleine Funktionalität für sich alleine getestet und dann implementiert bis der Test bestanden ist. Dies wird für alle Module durchexerziert. Sind alle Unit Tests bestanden, so werden die Integrationstests geschrieben. Dabei werden mehrere Module in einem Test geprüft. Da man ja weiss, dass jedes Modul einzeln für sich funktioniert, wie es sollte, existieren die Integrationstests zur Prüfung des Datenflusses und der Kommunikation zwischen den Modulen. Am Ende folgt der System-Test, der das gesamte Programm und somit den gesamten Datenfluss prüft.

Bereits jetzt kann die Einfachheit und Praktikabilität dieses Modells erkannt werden. Sobald es um die Wartung und Weiterentwicklung des zu schreibenden Programms geht, kann der Software-Entwickler auf intuitive und zeitsparende Weise den Code analysieren. Vor allem bei der Fehlersuche bringt das TDD enorme Vorteile mit sich, da wirklich alles was geschrieben wurde, auch getestet ist. Dadurch wird auf einen Blick erkannt auf welcher Ebene etwas schiefgelaufen ist und kann dementsprechend schnell und effizient behoben werden.

# Systemarchitektur

## .NET Implementationen

Zu Beginn unseres Projektes stellte sich für uns zuerst die Frage, welches Framework auf der HoloLens zum Einsatz kommt. Nach einiger Zeit an Recherche sind wir darauf gestossen, dass es zurzeit vier wichtige unterschiedliche .Net Implementationen gab.

### .NET Framework

Das .Net Framework ist die bekannteste Implementation von .NET. Aus diesem Grund setzen auch die meisten Bibliotheken und Programme auf .NET als Zielplattform, so zum Beispiel auch Akka oder System.Drawing.Bitmap.

Das .NET Framework kam für uns leider nicht in Frage, da die HoloLens aufgrund noch fehlender Softwareupdates keine Rückwärtskompatibilität für das .NET Framework aufweist.

### .NET Standard

Der .NET Standard ist der Versuch von Microsoft für alle .NET Implementationen einen kleinsten gemeinsamen Nenner zu definieren. So sollen alle Bibliotheken, die im .NET Standard definiert sind, auf allen aktuellen Windows Plattformen verfügbar sein. Entsprechend stellt auch die HoloLens eine .NET Standard Version zur Verfügung.

Leider stellte sich heraus, dass die HoloLens den .NET Standard Version 1.4 implementiert. Dieser fällt im Vergleich zum .NET Framework sehr mager aus. Deswegen war es für uns nicht möglich mit diesem Standard unser Framework zu implementieren. Die aktuelle Version .NET Standard 2.0 ist allerdings wesentlich umfangreicher und ermöglichte uns auch das implementieren der gewünschten Funktionalität.

Die Bezeichnung .NET Core steht für die tatsächliche Implementation des .NET Standards. Entsprechend stellt .NET Core dieselbe Funktionalität zur Verfügung wie .NET Standard.

### UWP

Als Universal Windows Platform (UWP) bezeichnet Microsoft jene Laufzeitumgebung, die auf allen Geräten zur Verfügung steht, die Windows 10 unterstützen, zum Beispiel die HoloLens oder die neue Xbox. Entsprechend implementiert UWP einen Teil der aus den .NET Framework bekannten Bibliotheken und stellt auch eigene zur Verfügung. Leider sind die Klassen, die UWP verwendet, nicht sehr gut dokumentiert und auch nicht sehr durchsichtig. So ist es zum Beispiel eine echte Herausforderung aus einer Software Bitmap die Pixeldaten auszulesen.

## Systemdesign

## 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. Als Beispiel soll hier der RecognitionManager[[3]](#footnote-3) genommen werden. Er hat die Aufgabe die Frames auszuwerten, in dem er die Marker erkennt und die Position und Rotation der gesehenen Marker berechnet und speichert. Der RecognitionManager wird über den ControlActor gesteuert, indem er zum Beispiel die Nachrichten NewFrameArrived oder RequestAllVirtualObjects erhält.

Wir entschieden uns für das Actor Modell, da die Implementierung der Actors intuitiv, verständlich und dazu noch einfach zu erweitern ist. Das Ziel war, dass man sich schnell im Programmcode zurechtfinden und somit zügig den Ablauf verstehen sollte. Zusätzliches Feature ist die Erweiterbarkeit. Als Beispiel könnte man ein Positionsmodul, das die Auswertung der Daten von Bluetooth Beacons implementiert, in das System einfügen, in dem man nur einen neuen «Position Actor» erstellt und diesen über den ControlActor steuert.

### 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 Nachrichten Konstruktor korrekt verwendet wurde.

### Initialisierung

Zur Initialisierung erstellt der End-User jeweils ein SyncObject, in dem die Daten von der Auswertung der Frames, d.h. die einzelnen CodeObjects mit der aktuellen Position und Rotation vom Framework, reingeschrieben werden und ein CameraFeedSyncObject, auf das die neuen Frames von aussen geschrieben und anschliessend vom Framework gelesen werden. Weiter erstellt und übergibt der End-User ein Array mit den zu erkennenden CodeObjects mitsamt deren eventuell angegeben Startpositionen.

Dann erstellt der End-User ein neues Objekt der Klasse Framework mit den erstellten Objekten als Konstruktor Argumente. Anschliessend ruft er von diesem die Funktion Initialize() auf, was das Framework startet. Danach «füttert» er die einzelnen Frames in das CameraFeedSyncObject und erhält die Daten aus dem SyncObject.

### ControlActor

Der ControlActor kümmert sich um den generellen Datenfluss. Er erhält von den angebundenen Modulen Nachrichten, die es ihm ermöglichen den Kontroll- und Datenfluss zu steuern. Die Anbindung von neuen Modulen mittels Actors ist möglich, in dem man einen neuen Actor vom ControlActor aus kreiert. Die neu gewonnene Funktionalität setzt man um, in dem man in der asynchronen Funktion ReceiveAsync eine neu definierte Nachricht erwartet.

### RecognitionManager

Der RecognitionManager kümmert sich um das Erkennen von Aruco Codes in den Frames mittels der Bilderkennungsbibliothek EmguCV. Er verwaltet dabei die erkannten Marker in einem Dictionary. Auf Anfrage liefert er dieses Dictionary zurück. Weiter kann er einzelne Aruco Codes aktivieren oder deaktivieren. Diese Funktionalitäten kann man noch erweitern. Um ein paar mögliche Ideen zu sehen, was der Recognition Manager auch noch tun könnte, kann man ein Blick in das Protokoll «ControlToRecognitionManager» im Ordner «Protocols» im Framework werfen. Eine kleine Übersicht sei hier gezeigt:

* Start a virtual Object – kann so z.B. ein Video abspielen lassen
* Stop a virtual Object – kann so z.B. ein Video stoppen
* Create a virtual Object – kann so z.B. einen zusätzlichen Marker hinzufügen, der nicht im Initialisierung - Dictionary drinstand, aber den man in Zukunft trotzdem erkennen will.
* Kill a virtual Object – kann so z.B. einen Marker, den man momentan erkennt, aus dem Initialisierung – Dictionary löschen und damit wird man ihn in Zukunft nicht mehr erkennen können.

All diese Nachrichten sind jedoch noch nicht implementiert! Weiter kann man, je nach dem was man mit den angezeigten Objekten noch tun will, so viele Funktionalitäten hinzufügen, wie man möchte. Das Ziel war es, ein Framework als Grundbaustein für spätere Module zu entwerfen.

### SyncActor

Der SyncActor stellt die Daten aus der Auswertung eines Frames für die Verarbeitung ausserhalb des Frameworks bereit. Auf dem von ihm verwalteten SyncObject kann dann der End-User die Daten einsehen bzw. kopieren.

### CameraFeedActor

Der CameraFeedActor ist dafür verantwortlich neue Frames zu registrieren und dem Control Actor über deren Ankunft Auskunft zu geben. Dem Actor Modell entsprechend sendet er dem Control Actor dafür eine Nachricht.

### Erkennungsablauf

Der grundsätzliche Ablauf ist:

* Der End-User 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 darauf die Initialize-Funktion auf.
* Ein neuer Frame wird auf das CameraFeedSyncObject geladen. Daraufhin wird die Funktion UpdateFrame vom End-User aufgerufen. Das löst beim CameraFeedActor das Event «OnFrameUpdated» aus.
* Der CameraFeedActor schickt bei diesem Event eine Nachricht NewFrameArrived an den ControlActor, der beim Erhalt der Nachricht einen Request mit einem neuen NewFrameArrived an den RecognitionManager startet.
* Der RecognitionManager startet die FrameEvaluation beim Erhalt von NewFrameArrived. Er 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 RecognitonManager’s 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:

* Markers
  + VuMarker
  + Apriltag
  + Barcode
  + QR - Code
  + ARToolKit Marker
  + Aruco Code
* Bilderkennung
  + ARToolKit
  + Vuforia
  + OpenCVSharp
  + EmguCV
* Actor Implementierung
  + Akka.NET
  + Proto
* Darstellung
  + Xamarin
  + Unity

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

## Markers

Die Wahl eines geeigneten Markers stand am Anfang unseres Projekts, um die dementsprechenden Tools dafür zu finden. Wir informierten uns über die folgenden Marker.

### VuMark

Vuforia bietet ihre eigenen Marker, die sogenannten VuMarks[[4]](#footnote-4). Da eine Bedingung unseres Projekts war, nur OpenSource Software zu benutzen, waren die Marker kaum eine Option. Zwar hätte man in der gratis Version von Unity 2017.2, die erst während des Projekts veröffentlich wurde, die Marker verwenden können, doch in der Stand-Alone Version von Vuforia war dies nur mit der PRO Lizenz möglich.

Abbildung 1: VuMark

### Apriltag

Apriltags[[5]](#footnote-5) sind ähnlich aufgebaut wie Aruco Codes und werden generell für Aufgabenstellungen im Bereich der AR, Robotics und der Kamerakalibration verwendet. Diese Codes wären auch eine gute Alternative gewesen, jedoch schienen uns die Bibliotheken für Aruco Markers für unser Projekt ausgereifter zu sein, weshalb wir uns gegen Apriltags entschieden haben.

Abbildung 2: Apriltag

### Datei:Barcode EAN8.svgBarcode

Es wurden nicht so viele Codierungsmöglichkeiten, wie uns ein Barcode[[6]](#footnote-6) bereitstellen könnte, benötigt. Weiter sind feine Linien schwer detektierbar aus grosser Distanz, was für unser Use Case problematisch geworden wäre.

Abbildung 3: Barcode

### https://upload.wikimedia.org/wikipedia/commons/8/87/QRCode.pngQR Codes

QR Codes[[7]](#footnote-7) bieten viele verschiedene ID’s. Jedoch brauchen Sie dazu mehr Details für die Auswertung, was wiederum die maximale Distanz für die Erkennung mindert. Da für eine realistische Tour kaum mehr als 1024 IDs benötigt werden, haben wir uns gegen die QR Codes entschieden.

Abbildung 4: QR-Code

### ARToolKit Marker

Das ARToolKit[[8]](#footnote-8) verwendet sogenannte „square Markers“, die ein paar Bedingungen erfüllen müssen. Offensichtlich müssen diese quadratisch sein, zusätzlich müssen sie einen ununterbrochenen Rand haben und das Muster soll nicht rotationssymmetrisch sein. Anfangs unseres Projekts haben wir mit dem Hiro Marker aus dem ARToolKit gearbeitet und mit ihm das Sample vom ARToolKit auf der HoloLens erfolgreich getestet.

Abbildung 5: Hiro Marker

### Aruco Marker

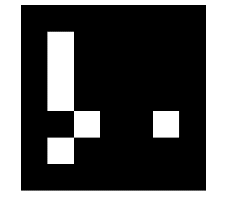
Aruco Marker[[9]](#footnote-9) werden häufig in Augmented Reality Anwendungen verwendet, da sie trotz ihrer minimalen und gröberen Muster ausreichend Codierungsmöglichkeiten bieten. Die 1’024 verschiedenen IDs können dank dem Hamming Code auch auf grössere Distanz beinahe fehlerfrei detektiert werden. Dies war einer der Gründe, weshalb wir uns schlussendlich für Aruco Marker entschieden haben. Weitere Gründe waren, dass es bereits einige Libraries für die Aruco Marker Erkennung gab und einer unserer Betreuer, sich damit schon befasst hatte.

Abbildung 6: Aruco Marker

## Bilderkennung

Für den ersten Mini Use Case war es nötig, dass wir uns für eine Bibliothek entscheiden, die die Erkennung von Aruco-Markern bereits effizient und vor allem zuverlässig implementiert hat. 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, das eine speziell für die HoloLens angepasste Version eines OpenCV-Wrappers darstellt, 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 Recherchen 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 Erkennung nur teilweise implementiert hat. Da uns das implementieren eines Wrappers für eben diese 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, die wir für unsere Implementierung des Durchstichs gewählt haben. EmguCV stellt ebenfalls eine Portierung der bekannten OpenCV-Bibliothek auf C# zur Verfügung. EmguCV bietet dabei die volle Funktionalität von OpenCV.Aruco und deckt damit den Bedarf von unserem Use Case ab. Desweiteren ist OpenCV im Bereich Computer Vision eines der verbreitetsten und meist genutzten Tools.

## Actor - Implementierung

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

### Akka.NET

Unsere erste Recherche ergab, dass Akka.NET[[10]](#footnote-10), was eine .NET kompatible Implementierung des Actor Modells darstellt, sich für unser Projekt eignen könnte. Nach langwierigem Einarbeiten und testen der Bibliothek fiel uns auf, dass Akka.NET aufgrund seiner Vielzahl von Abhängigkeiten (z.B. System.Threading.Thread) nicht mit der aktuellen Version von Unity (2017.2) und dem Framework der HoloLens kompatibel ist. Gemäss Entwicklerforen[[11]](#footnote-11) 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[[12]](#footnote-12) 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 Abhängigkeiten, die Akka für uns unbrauchbar machten. Einziges Manko bei Proto ist, dass kein TestKit implementiert ist, was das Testing schwieriger 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.

### Xamarin

Eine Alternative zu Unity wäre Xamarin gewesen. Obwohl Microsoft Xamarin aufgekauft hat, ist Unity[[13]](#footnote-13) für AR Anwendungen auf der HoloLens praktischer, bietet mehr Funktionsumfang und wird deshalb auch von den HoloLens Entwicklern als Darstellungsprogramm empfohlen. Microsoft kollaboriert auch mit Unity und somit gab es für uns keinen Grund Xamarin für dieses Projekt zu benutzten.

### Unity

Unity ist eine Entwicklungsumgebung für Spiele und sonstige 3D-Anwendungen. Im Vergleich mit den Alternativen ist Unity im Bereich VR und AR weit voraus. Vorteile von Unity sind, dass es sehr benutzerfreundlich ist und das Erstellen von Objekten in einer AR Szene unkompliziert ablauft. Wir starteten unser Projekt mit der gratis Unity Personal 2017.1 und upgradeten danach im Verlauf auf 2017.2. Allerdings gab es mit diesen Versionen einige Kompatibilitätsprobleme, deshalb konnten wir das Anzeigen mit Unity noch nicht umsetzen. Dies sollte aber mit den folgenden Versionen wie 2018.1 hoffentlich drin liegen. Das Testing in Unity verhält sich etwas anders als in Visual Studio, da hier der integrierte Test Runner geeigneter ist, um im Edit Mode zu testen.

### UWP UI

Das UI für UWP Apps basiert auch auf XAML und bietet daher relative ähnliche Darstellungsoptionen wie eine normale WPF GUI. In unserem Projekt reichte ein simples Fenster mit Webcam Übertragung und der Anzeige von Listbox-Elementen aus, um zu zeigen, dass unser Framework funktioniert.

Die Funktion des UWP UI’s ist bei Beginn des Programms, die Kamera, das SyncObject, das CameraFeedSyncObject und das Framework zu initialisieren und ab dann sendet es die Frames in das Framework. Danach aktualisiert es die angezeigte Listbox, sobald das SyncObject vom Framework neu beschrieben wird.

## Probleme in der Entwicklung

### Probleme mit dem Framework

Die Entwicklung unseres 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 des 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.

### Probleme mit Unity

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. Das erhoffte Update von Unity, um unser Actor Framework mit der .Net Core 2.0 Kompatibilität einzubinden, wurde auf 2018.1 oder auf noch spätere Versionen verschoben. Das hat dazu geführt hat, dass wir für den „Proof of Concept“ zuerst auf die WPF GUI umgestiegen sind und schlussendlich eine UWP UI mit Webcam programmiert haben.

Erst später haben wir realisiert, dass Unity nicht das Hauptproblem war, sondern die HoloLens, welche noch auf der UWP Version Build 14393 war, welches zu diesem Zeitpunkt für die HoloLens die aktuellste Version war und somit maximal .Net Standard 1.4 oder .Net Core 1.0 unterstützte. Das nächste Update für die HoloLens sollte im Frühling 2018 kommen.

Eine weitere Einschränkung kam von Unity, dass die Darstellung nicht parallelisierbar ist, da Unity nur auf einem Main Thread läuft. Aber da Unity das Framework nur initialisiert, müsste das Actor Framework so wie geplant funktionieren. Damit ist gemeint, dass jeder Actor auf seinem eigenen Thread arbeitet.

### Probleme mit den Bilderkennungsbibliotheken

Weitere Probleme hatten wir mit der Bilderkennungsblibliothek «OpenCVSharp». Sie hatte zwar ein Aruco Code Erkennungsmodul, doch war dieses nicht vollständig. Die Funktion DetectMarkers() konnte man verwenden, wenn man das NuGet-Packet ins Projekt eingebunden hatte, doch die Funktion EstimatePoseSingleMarkers() war nicht aufzufinden. Denn in der CvAruco Klasse war nur die DetectMarkers() Funktion drin[[14]](#footnote-14), 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.

Weiter hatten wir noch ein paar Hürden mit EmguCV zu überwinden. Angefangen hat es mit den Warnungen der Nichtverwendbarkeit von EmguCV nach Download des NuGet-Pakets. 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 in den Kommentaren beschrieben, wie man z.B. eine Bitmap in ein Image umschreiben kann (dafür mussten wir noch selbst einige Hilfsfunktionen schreiben, siehe [[15]](#footnote-15)) oder wie man Daten aus einem Mat herausliest. Dafür suche man doch im Code des Recognition Managers und in den Utils Funktionen[[16]](#footnote-16) und deren Tests[[17]](#footnote-17) nach den genauen von uns gefundenen Lösungen zu diesen Problemen.

# Ergebnisse der Arbeit

Im unteren Bild sieht man das funktionierende Actor Framework zusammen mit der UI, das vordefinierte Aruco Marker ID’s relativ robust erkennt und Positions- sowie Rotationsdaten liefern kann. Ansätze für die Portierung auf Unity und eine GUI zur Darstellung der Ergebnisse, sowie die zugehörige Dokumentation haben wir ebenfalls erarbeitet. Leider ist uns die vollständige Implementierung der Tour auf der HoloLens aufgrund der Probleme mit Unity und der fehlenden Unterstützung von .NET Standard 2.0 seitens der HoloLens nicht gelungen.

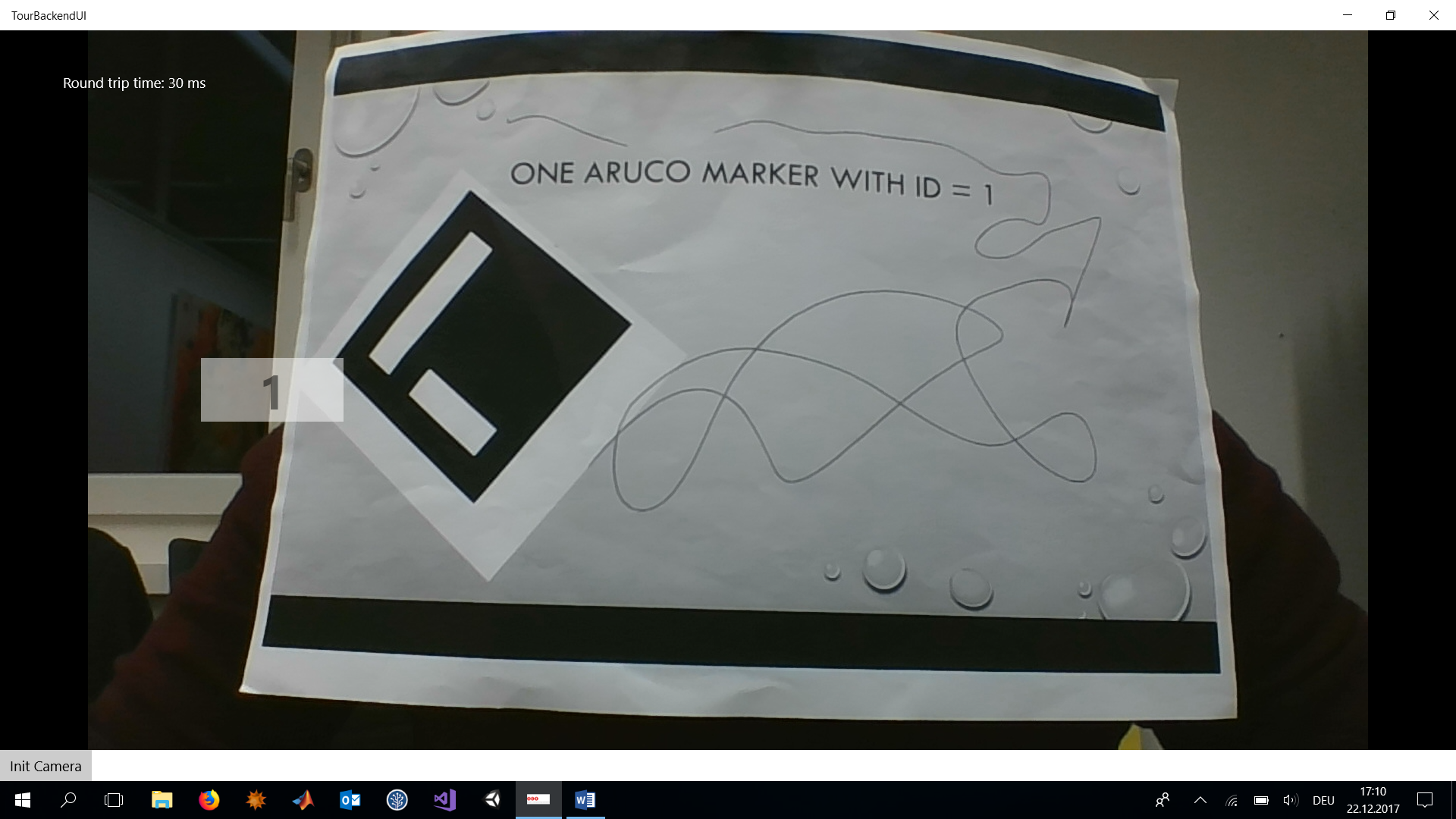


Abbildung 7: erstelltes UWP UI mit der Laptop - Webcam als Datenquelle

## Ausblick

Nach dem Update der HoloLens[[18]](#footnote-18) im Frühling 2018 und dem bis dahin wohl auch erschienen Update 2018.1 von Unity[[19]](#footnote-19) sollte der .NET Standard 2.0 komplett unterstützt werden und somit unser Framework auf der HoloLens lauffähig sein. Fraglich ist lediglich ob der Kompilierungsvorgang von Unity ebenfalls verbessert wird, denn momentan benötigt man einige teils unsaubere Hacks um Projekte auf Unity zu portieren.

Der Ausblick für unser Framework ist dementsprechend gut. Die verwendeten Bibliotheken benötigen einen Mindeststandard von .NET Core 2.0 und das UI basiert auf UWP und sollte damit leicht auf Unity portierbar sein. Weitere Module sind bereits jetzt leicht zu implementieren, da die Struktur des Actor Frameworks eine hohe Erweiterbarkeit zulässt. Die Frage, ob man mit der momentanen Bilderkennungssoftware die magische Grenze von 24 Frames pro Sekunde, ohne weiteres erreichen kann, bleibt allerdings offen. Es ist dies die Schwelle für das menschliche Auge die Bilder als flüssige Bewegungen wahrzunehmen.

# Anhang

## Abbildungsverzeichnis

[Abbildung 1: VuMark 8](file:///C:\Users\Moritz%20Scherer\Documents\Holotrio\HololensTour\Organisatorisches\Dokumentation\Bericht.docx#_Toc502763383)

[Abbildung 2: Apriltag 8](file:///C:\Users\Moritz%20Scherer\Documents\Holotrio\HololensTour\Organisatorisches\Dokumentation\Bericht.docx#_Toc502763384)

[Abbildung 3: Barcode 8](file:///C:\Users\Moritz%20Scherer\Documents\Holotrio\HololensTour\Organisatorisches\Dokumentation\Bericht.docx#_Toc502763385)

[Abbildung 4: QR-Code 8](file:///C:\Users\Moritz%20Scherer\Documents\Holotrio\HololensTour\Organisatorisches\Dokumentation\Bericht.docx#_Toc502763386)

[Abbildung 5: Hiro Marker 9](file:///C:\Users\Moritz%20Scherer\Documents\Holotrio\HololensTour\Organisatorisches\Dokumentation\Bericht.docx#_Toc502763387)

[Abbildung 6: Aruco Marker 9](file:///C:\Users\Moritz%20Scherer\Documents\Holotrio\HololensTour\Organisatorisches\Dokumentation\Bericht.docx#_Toc502763388)

[Abbildung 7: erstelltes UWP UI mit der Laptop - Webcam als Datenquelle 14](#_Toc502763389)

1. Übersicht über die Hololens: https://www.microsoft.com/de-ch/hololens [↑](#footnote-ref-1)
2. https://de.wikipedia.org/wiki/Testgetriebene\_Entwicklung [↑](#footnote-ref-2)
3. Siehe auch Kapitel 2.3.4 Recognition Manager [↑](#footnote-ref-3)
4. https://library.vuforia.com/articles/Training/VuMark [↑](#footnote-ref-4)
5. https://april.eecs.umich.edu/software/apriltag.html [↑](#footnote-ref-5)
6. https://de.wikipedia.org/wiki/Strichcode [↑](#footnote-ref-6)
7. https://de.wikipedia.org/wiki/QR-Code [↑](#footnote-ref-7)
8. https://archive.artoolkit.org/documentation/doku.php?id=3\_Marker\_Training:marker\_about [↑](#footnote-ref-8)
9. https://docs.opencv.org/3.1.0/d5/dae/tutorial\_aruco\_detection.html [↑](#footnote-ref-9)
10. Übersicht über Akka.NET: https://getakka.net/ [↑](#footnote-ref-10)
11. https://github.com/akkadotnet/akka.net/issues/2153 [↑](#footnote-ref-11)
12. Übersicht über Proto: http://proto.actor/ [↑](#footnote-ref-12)
13. Siehe auch Kapitel [↑](#footnote-ref-13)
14. <https://github.com/shimat/opencvsharp/blob/master/src/OpenCvSharp/Modules/aruco/CvAruco.cs> [↑](#footnote-ref-14)
15. Framework, Tourbackend .Net Standard Projekt, Datei Utils/UseCustomVideoFrameSource.cs, Klasse Utils [↑](#footnote-ref-15)
16. Framework, Tourbackend .Net Standard Projekt, Datei Utils/UseCustomVideoFrameSource.cs, Klasse Utils [↑](#footnote-ref-16)
17. Framework, UnitTestProject1, Datei UtilsTest, TestEmguCV\_DetectMarkers\_and\_EstimatePoseSingleMarkers() [↑](#footnote-ref-17)
18. https://windowsarea.de/2017/11/hololens-expandiert-in-europa-und-bekommt-anfang-2018-neue-features/ [↑](#footnote-ref-18)
19. https://unity3d.com/unity/roadmap [↑](#footnote-ref-19)