|  |
| --- |
| Hololens Tour |
| Abschlussbericht |
|  |
| **Datum** 15. Dezember 2017  **Autor(en)** Markus Kiser, Moritz Scherer, Selim Naji |

Inhaltsverzeichnis

[Inhaltsverzeichnis ii](#_Toc501114018)

[1.1. Abstract 3](#_Toc501114019)

[1.2. Use Case 3](#_Toc501114020)

[1.3. Systemarchitektur 4](#_Toc501114021)

[1.3.1. .NET Implementationen 4](#_Toc501114022)

[1.3.2. Actors 5](#_Toc501114023)

[1.3.3. Nachrichten 5](#_Toc501114024)

[1.3.4. Initialisierung 5](#_Toc501114025)

[1.3.5. ControlActor 6](#_Toc501114026)

[1.3.6. RecognitionManager 6](#_Toc501114027)

[1.3.7. SyncActor 6](#_Toc501114028)

[1.3.8. CameraFeedActor 7](#_Toc501114029)

[1.3.9. Erkennungsablauf 7](#_Toc501114030)

[1.4. Übersicht über Bibliotheken und Tools 8](#_Toc501114031)

[1.4.1. Bilderkennung 8](#_Toc501114032)

[1.4.2. Actorimplementierung 9](#_Toc501114033)

[1.4.3. Darstellung 10](#_Toc501114034)

[1.5. Probleme in der Entwicklung 10](#_Toc501114035)

[1.6. Ergebnisse der Arbeit 12](#_Toc501114036)

[1.7. Ausblick 12](#_Toc501114037)

## 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 Lösungen 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 uns 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 vom Chef der Firma 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 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 verabschiedet.

## Systemarchitektur

### .NET Implementationen

Bei Beginn unseres Projekt stellte sich für uns zuerst die Frage, welches Framework auf der Hololens zum Einsatz kommt. Nach einiger Recherche sind wir darauf gestossen, dass es zur Zeit 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 der .NET Standard Version 1.4 implementiert, die im Vergleich zum .NET Framework sehr mager ausfällt. Auf dieser Basis war es für uns nicht möglich unser Framework zu implementieren. Die aktuelle Version .NET Standard 2.0 ist allerdings wesentlich weiter und 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 Funktiionalität zu Verfügung wie .NET Standard.

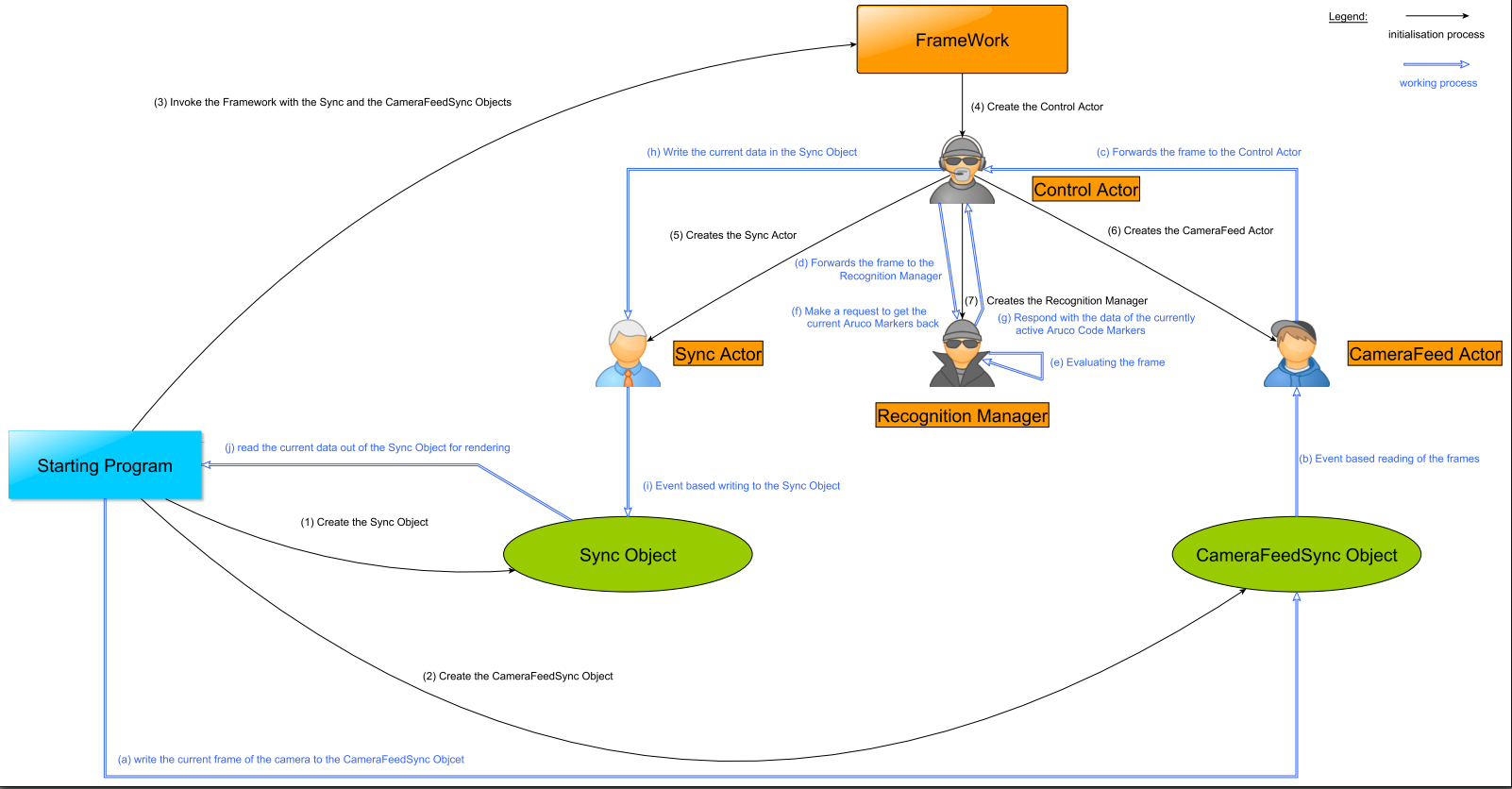
#### UWP

Als Universal Windows Platform 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 SoftwareBitmap die Pixeldaten auszulesen.

### 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 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 Control Actor 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 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 Control Actor steuert.



Systemarchitektur

### 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 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, in dem man einen neuen Actor vom Control Actor 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 ArucoCodes 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 ArucoCode 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 werfen. Da sind zum Beispiel noch die Funktionen:

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

Diese sind jedoch noch nicht implementiert! Man kann, je nach dem was man noch mit den angezeigten Objekten tun will, so viele Fancy Funktionalitäten hinzufügen, wie man möchte.

### 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/kopieren.

### CameraFeedActor

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

### Erkennungsablauf

Der grundsätzliche Ablauf ist folgender:

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

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 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-Markererkennung 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 für die wir uns für die Implementierung des Durchstichs entschieden 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. 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, 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[[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 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 für AR Anwendungen auf der HoloLens praktischer, bietet mehr 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 den Anderen weit voraus. Vorteile von Unity sind, dass es sehr benutzerfreundlich ist. Das Erstellen von Objekten in einer AR Szene und das Kreieren eines Builds für UWP bzw. die HoloLens sind relativ unkompliziert machbar, hätte es bei uns nicht einige Kompatibilitätsprobleme gegeben. 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.

Wir starteten unser Projekt mit der gratis Unity Personal 2017.1 und upgradeten danach im Verlauf auf 2017.2.

## 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 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.

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 noch spätere Versionen verschoben. Das hat dazu geführt hat, dass wir für den „Proof of Concept“ auf die WPF GUI umgestiegen sind.

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. Unity initialisiert das Framework nur, sodass das Actor Framework so wie geplant funktioniert. Damit ist gemeint, dass jeder Actor auf seinem eigenen Thread arbeitet.

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[[2]](#footnote-2), 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 [[3]](#footnote-3)) oder wie man Daten aus einem Mat herausliest. Dafür suche man doch im Code des Recognition Managers und in den Utils Funktionen[[4]](#footnote-4) und deren Tests[[5]](#footnote-5) nach den genauen von uns gefundenen Lösungen zu diesen Problemen.

## Ergebnisse der Arbeit

Als Endprodukt haben wir das funktionierende Actor Framework, welches vordefinierte Aruco-Marker 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 auch 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.

## Ausblick

Nach dem Update der HoloLens[[6]](#footnote-6) im Frühling 2018 und dem bis dahin wohl auch erschienen Update 2018.1 von Unity[[7]](#footnote-7) sollte .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 entsprechend gut. Die verwendeten Bibliotheken benötigen einen Mindeststandard von .NET Core 2.0 und die UI basiert auf UWP und sollte damit leicht auf Unity portierbar sein. Weitere Module sind bereits jetzt mehr oder minder leicht zu implementieren, da die Struktur des Actor Frameworks eine hohe Parallelität zulässt. Die Frage, ob man mit der momentanen Bilderkennungssoftware die Marke von 24 Frames pro Sekunde ohne weiteres erreichen kann, bleibt allerdings offen.

1. https://github.com/akkadotnet/akka.net/issues/2153 [↑](#footnote-ref-1)
2. <https://github.com/shimat/opencvsharp/blob/master/src/OpenCvSharp/Modules/aruco/CvAruco.cs> [↑](#footnote-ref-2)
3. Framework, Tourbackend . Net Standard Projekt, Datei Utils/UseCustomVideoFrameSource.cs, Klasse Utils [↑](#footnote-ref-3)
4. Framework, Tourbackend . Net Standard Projekt, Datei Utils/UseCustomVideoFrameSource.cs, Klasse Utils [↑](#footnote-ref-4)
5. Framework, UnitTestProject1, Datei UtilsTest, TestEmguCV\_DetectMarkers\_and\_EstimatePoseSingleMarkers() [↑](#footnote-ref-5)
6. https://windowsarea.de/2017/11/hololens-expandiert-in-europa-und-bekommt-anfang-2018-neue-features/ [↑](#footnote-ref-6)
7. https://unity3d.com/unity/roadmap [↑](#footnote-ref-7)