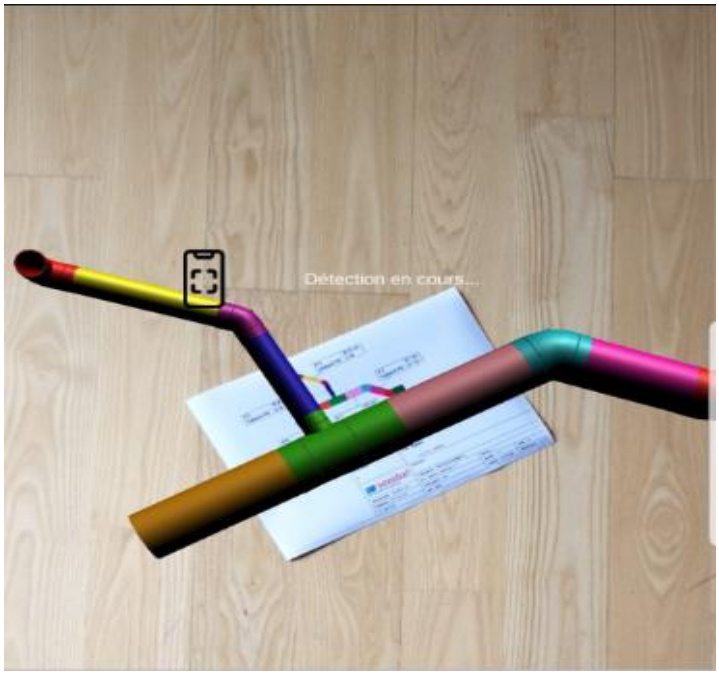
Hes·so VALAIS WALLIS

Haute Ecole de Gestion
Hochschule für Wirtschaft

Filière Informatique de gestion
Studiengang Wirtschaftsinformatik

Travail de Bachelor 2021

Appréhender la représentation spatiale grâces à la réalité augmentée



Etudiant-e : Sarah Morard

Professeur : Antoine Widmer

Déposé le : 04 août 2021

Site web : www.hevs.ch

RÉSUMÉ

Ce projet s'inscrit dans le contexte d'un travail de Bachelor à la HES-SO Valais-Wallis. Il a été proposé par le professeur Antoine Widmer en collaboration l'Organisation Romande d'Intégration et de Formation [ORIF].

Le but de ce travail de Bachelor est d'ajouter deux fonctionnalités à une application de réalité augmentée qui a été développée dans le contexte d'un précédent travail de Bachelor. Cette application a pour but d'aider les apprentis de l'ORIF qui travaillent dans le domaine des installations sanitaires et qui souffrent de troubles de l'apprentissage.

Cette aide vient du fait que l'application transforme un plan de tuyauterie en deux dimensions en un modèle en trois dimensions sur l'écran d'un téléphone ou d'une tablette. Cela permet aux apprentis en difficulté de mieux se représenter ce qu'ils doivent construire.

Afin d'apporter une aide supplémentaire aux apprentis de ORIF, nous devons faire en sorte que cette application puisse reconnaître chaque élément d'un plan mis en couleur et les colorer de manière identique sur un modèle en trois dimensions qui sera affiché sur l'écran d'un téléphone ou d'une tablette.

Cela facilitera la tâche des apprentis qui éprouvent des difficultés à distinguer une pièce d'une autre sur un plan et sur son modèle.

Mots-clés : réalité augmentée, apprentissage, application

AVANT-PROPOS

L'ORIF s'occupe de l'orientation, de l'intégration socioprofessionnelle et de la formation de personnes atteintes dans leur santé ou en difficulté en Suisse Romande. La mission de cette organisation est de permettre une intégration professionnelle durable sur le marché du travail.

Dans le canton du Valais, des formations dans les domaines du bâtiment, de la restauration et des services sont données à Sion.

La réalité augmentée est de plus en plus utilisée grâce à l'expansion des téléphones portables. Elle est utilisée autant pour les loisirs sous la forme de jeux vidéo que pour dans les domaines professionnels sous forme d'applications. Les personnes travaillant dans le domaine de l'éducation s'y intéressent également car son utilisation peut apporter une aide ludique.

REMERCIEMENTS

Je remercie toutes les personnes qui m'ont aidé à réaliser ce travail de Bachelor, en particulier :

- Le professeur Antoine Widmer pour son accompagnement durant la réalisation de ce projet.
- Mme Michèle Constantin pour la relecture de ce rapport.

TABLE DES MATIÈRES

TABLE DES ILLUSTRATIONS

LISTE DES ABRÉVIATIONS

ORIF	Organisation Romande d'Intégration et de Formation
RGB	Red, Green, Blue – Rouge, Vert, Bleu
SDK	Software Development Kit – Kit de développement

GLOSSAIRE

Wrapper	Programme dont la fonction est d'appeler une autre fonction
Framework	Environnement de travail
Open Source	Logiciel dont le code est ouvert à son utilisateur
Plugin	Outils qui permettent d'ajouter des fonctions à un programme
Product Backlog	Outils de travail qui contient une liste de tâches à faire durant un projet
Script	Type de fichiers utilisé par Unity
.Net	Framework créé par Microsoft pour le développement d'applications
Vecteur	Tableau à une dimension
Matrice	Tableau à deux dimensions

1. Introduction

1.1. Sujet

Ce travail de Bachelor a pour but l'ajout de deux fonctionnalités à une application de réalité augmentée. L'application en question a fait l'objet d'un ancien travail de Bachelor. Celle-ci reconnait un plan en deux dimensions préalablement défini et l'affiche en trois dimensions. Notre travail de Bachelor détecte les couleurs des différents éléments d'un plan en deux dimensions qui est filmé par un appareil et les ajoute sur le plan en trois dimensions.

1.2. Problématique

Certains apprentis de l'ORIF souffrent de troubles de l'apprentissage, ce qui les empêche de pouvoir visualiser correctement des concepts abstraits. Il leur est donc compliqué de mettre en pratique leurs connaissances théoriques. Dans le domaine des installations sanitaires, par exemple, les apprentis sont confrontés à des plans abstraits en deux dimensions. Ils doivent pouvoir visualiser en trois dimensions des tuyaux dessinés sur un plan afin de pouvoir les assembler en situation réel. Ces tâches sont très compliquées pour certains apprentis.

L'ORIF souhaite ajouter des fonctionnalités à un ancien travail de Bachelor réalisé en 2020. Ce sont la reconnaissance de couleurs sur un plan en deux dimensions et la mise en couleurs des différentes pièces d'un modèle en trois dimensions. Les couleurs des pièces du modèle en trois dimensions doivent correspondre aux couleurs des pièces du plan.

1.3. La réalité augmentée

La réalité augmentée est un mélange entre le monde digital et le monde réel. Elle utilise plusieurs de nos sens afin de nous donner l'impression qu'une chose irréelle se passe dans un univers réel. (DORLAC, 2014)

Cette technologie n'est pas nouvelle car des recherches à son sujet ont commencé dans les années 90, mais sa popularité a vraiment pris de l'ampleur grâce à la prolifération des téléphone portables et des tablettes numériques. (Linowes & Babilinski, 2017)

La réalité augmentée met à contribution plusieurs de nos sens. Nous avons la vue, mais aussi l'audition et parfois même le toucher. (Jesse, 2018)

Plusieurs composants physiques sont nécessaires à l'utilisation de la réalité augmentée. Il y a tout d'abord la caméra qui va servir à renvoyer une ou plusieurs images à l'application de réalité augmentée. Ces images vont pouvoir être analysées par l'application qui va créer une carte de l'environnement filmé et garder une trace de la position de l'utilisateur. L'écran, qui est un autre composant nécessaire pour employer la réalité augmentée, va servir de support d'affichage et d'interaction avec l'utilisateur. (Arnaldi, Guitton, & Moreau, 2018)

Imaginons que nous désirons construire une maison. Nous disposons des plans d'un architecte mais ils sont pour nous, simple client, difficiles à comprendre. C'est là qu'intervient la réalité augmentée. Grâce à une application dédiée, nous pouvons filmer le terrain de notre future maison et la voir apparaître de manière virtuelle dans la scène que nous sommes en train de filmer. De cette manière, nous pouvons avoir une idée plus concrète de sa taille, sa structure et ce à quoi elle ressemblera une fois sa construction terminée. (Arnaldi, Guitton, & Moreau, 2018)

Mais il n'y a pas qu'à l'extérieur que nous pouvons utiliser des applications de réalité augmentée. De grandes entreprises spécialisées dans l'ameublement comme IKEA ont lancé leur propre application de réalité augmentée, IKEA Place, en automne 2017. Celle-ci permet à leurs clients de placer virtuellement des meubles dans leur maison. De cette façon, un client peut déterminer chez lui si un meuble s'intégrera bien dans son salon ou non avant de l'acheter. (IKEA, 1999)

Figure 1 - Démonstration de l'application IKEA Place - Site web IKEA, https://www.ikea.com/ch/fr/customer-service/mobile-apps/.

La figure 1 est une démonstration de l'application de réalité augmentée appelée « IKEA Place » créée par l'entreprise IKEA. L'utilisateur place ici un fauteuil virtuel dans son salon. Il peut faire de même avec d'autre meubles IKEA qui sont disponibles dans IKEA Place.

1.4. L'application de base

Dans le contexte de ce travail de Bachelor, nous devons ajouter deux fonctionnalités à une application de réalité augmenté. Cette application est un ancien travail de Bachelor réalisé par Nelson David Ribeiro Teixeira et proposé par le professeur Antoine Widmer en collaboration l'ORIF.

Cette application permet de reconnaître des images lorsque nous les filmons et d'afficher des formes en trois dimensions sur l'écran d'un téléphone. Ces images sont des plans de tuyaux données aux apprentis de l'ORIF et les formes en trois dimensions affichées sur l'écran sont ces tuyaux.

Grâce à cette application, les apprentis de l'ORIF peuvent avoir l'image de la forme en trois dimensions correspondantes sur le plan affiché sur leur écran. Ils peuvent ainsi mieux se représenter l'objet qu'ils doivent concevoir.

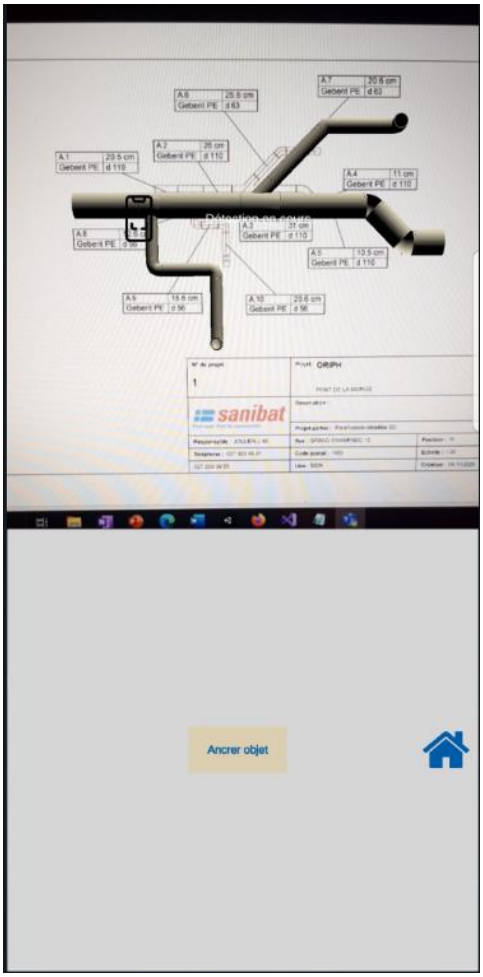


Figure 2 - Application du travail de Bachelor précédent - Image de l'auteure.

La figure 2 représente une capture d'écran de l'application. Nous pouvons y voir l'un des plans de l'ORIF et son tuyau affiché en trois dimensions par-dessus le plan.

2. État de l'art

Afin d'aider les apprentis de l'ORIF à mieux se représenter un plan en trois dimensions, nous allons devoir analyser les différentes technologies existantes qui nous aideront à intégrer une reconnaissance de couleurs sur un plan en deux dimensions puis de l'appliquer sur les pièces correspondantes sur le plan en trois dimensions.

Cette analyse va nous permettre de définir une série de critères. Ils vont nous aider à choisir la technologie qui sera la plus à même de répondre aux besoins du développement des nouvelles fonctionnalités qui doivent être apportées à l'application de réalité augmentée qui a été développée lors d'un ancien travail de Bachelor.

2.1. Vuforia

Vuforia Engine est un kit de développement [SDK]. Il est utilisé pour la création d'applications de réalité augmentée, spécifiquement pour la reconnaissance de formes ou d'images. Les avantages de ce SDK sont qu'il est compatible avec plusieurs plateformes telles qu'Android, iOS et Windows. (PTC Inc., 2011b)

Il est facilement intégrable à Unity (voir chapitre 4.2.1) via le gestionnaire de paquets et il supporte le langage de programmation C# utilisé par Unity. (PTC Inc., 2011c)

Vuforia permet de stocker plusieurs types d'objets dans une base de données comme des images, des modèles en trois dimensions, des formes cubiques et des cylindres. Nous pouvons ensuite récupérer ces objets pour les comparer à de vrais objets ou images dans la réalité. (PTC Inc., 2011b)

Ce SDK est gratuit et offre la possibilité de récupérer les trames d'une vidéo. Cela veut dire que nous pouvons travailler sur une image spécifique et donc trouver les couleurs qui la composent. Le point négatif de Vuforia est qu'il fait très mal la différence entre deux images identiques possédant des couleurs différentes. En effet, lorsque ce SDK analyse une image, les couleurs de celle-ci sont transformées en nuances de gris. Il est donc compliqué de l'utiliser pour la reconnaissance de couleurs. (PTC Inc., 2011e)

2.1.1. Les marqueurs

Il est important pour nous de comprendre comment Vuforia détecte une image. Lorsque nous filmons un environnement avec une application de réalité augmentée, ce SDK se mettra à chercher des marqueurs. Ceux-ci sont représentés par de petites étoiles jaunes sur les images insérées dans la base de donnée de Vuforia. Ce sont ces marqueurs qui vont servire de repères à l'application afin que Vuforia puisse être en mesure de reconnaître une image filmée par la caméra d'un appareil. (Cardinale, 2017)

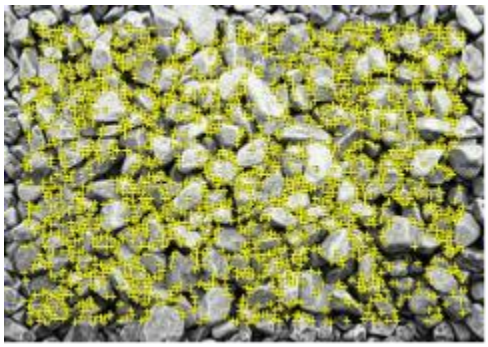


Figure 3 - Marqueurs placés sur une image par Vuforia - Documentation Vuforia, https://library.vuforia.com/features/images/image-targets/best-practices-for-designing-and-developing-image-based-targets.html.

La figure 3 représente une photo de cailloux qui a été examinée par Vuforia. Des marqueurs représentés par des étoiles jaunes ont été placées un peu partout sur l'image. (PTC Inc., 2011a)

2.1.2. La qualité des images

Les marqueurs qu'utilise Vuforia ne servent pas seulement à détecter une image, mais aussi à déterminer sa qualité. Cette qualité représente la facilité avec laquelle Vuforia pourra détecter une image. Les algorithmes de ce SDK se basent sur plusieurs critères afin de reconnaître une image. Lorsqu'une image est ajoutée dans la base de données de Vuforia, celle-ci est analysée et une note, allant de zéro à cinq étoiles, lui est attribuée afin d'informer l'utilisateur de sa qualité. Le premier critère analysé est la richesse des détails d'une image. En effet, plus elle possède de détails, plus Vuforia sera à même de la reconnaître. Il est également important que ces détails soient pourvus d'un certain nombre d'angles pointus comme ceux d'une forme cubique. Vuforia peine à reconnaître les formes arrondies. (PTC Inc., 2011a)

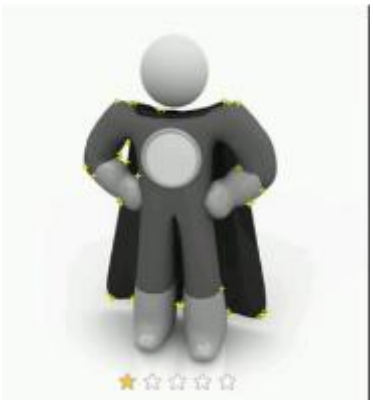


Figure 4 - Image considérée comme étant de mauvaise qualité par Vuforia - Documentation Vuforia, https://library.vuforia.com/features/images/image-targets/best-practices-for-designing-and-developing-image-based-targets.html.

La figure 4 est considérée par Vuforia comme étant de mauvaise qualité car les angles du personnage représenté sont trop arrondis. (PTC Inc., 2011a)

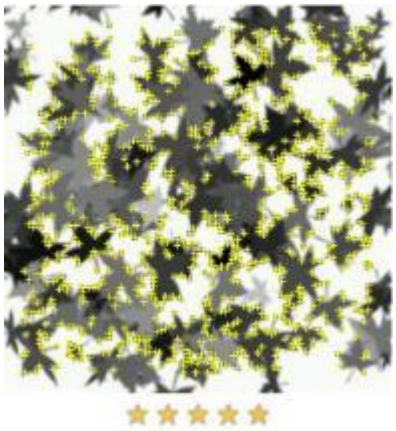


Figure 5 - Image considérée comme étant de bonne qualité par Vuforia - Documentation Vuforia, https://library.vuforia.com/features/images/image-targets/best-practices-for-designing-and-developing-image-based-targets.html.

La figure 5 est considérée par Vuforia comme étant de bonne qualité car les angles des feuilles sont pointus. Elle possède également une grande quantité de détails. (PTC Inc., 2011a)

La répartition des marqueurs sur une image est aussi importante pour déterminer sa qualité. En effet, si elles sont toutes concentrées à un même endroit, cela donnera une image de mauvaise qualité. (PTC Inc., 2011a)

Figure 6 - Image considérée comme étant de mauvaise qualité par Vuforia - Documentation Vuforia, https://library.vuforia.com/features/images/image-targets/best-practices-for-designing-and-developing-image-based-targets.html.

Les marqueurs jaunes sur la figure 6 sont principalement concentrés à gauche, ce qui en fait une image de mauvaise qualité pour Vuforia. (PTC Inc., 2011a)

Un autre critère que nous devons prendre en compte est le contraste des couleurs sur une image. Vuforia ne reconnaît pas les différentes teintes d'une image. Celle-ci est passée en monochrome avant d'être analysée. (PTC Inc., 2011a)

Figure 7 - Image considérée comme étant de mauvaise qualité par Vuforia - Documentation Vuforia, https://library.vuforia.com/features/images/image-targets/best-practices-for-designing-and-developing-image-based-targets.html.

La figure 7 est considérée par Vuforia comme étant de mauvaise qualité car ses différentes couleurs sont très similaires avec peu de contraste. (PTC Inc., 2011a)

Figure 8 - Image considérée comme étant de bonne qualité par Vuforia - Documentation Vuforia, https://library.vuforia.com/features/images/image-targets/best-practices-for-designing-and-developing-image-based-targets.html.

La figure 8 est considérée par Vuforia comme étant de bonne qualité car le contraste entre le fond et les feuilles est bien visible. (PTC Inc., 2011a)

Le dernier critère est la répétition de motifs sur une image. Si une répétition de motifs similaires est détectée sur une photo, l'image sera considérée comme étant de mauvaise qualité par Vuforia. (PTC Inc., 2011a)

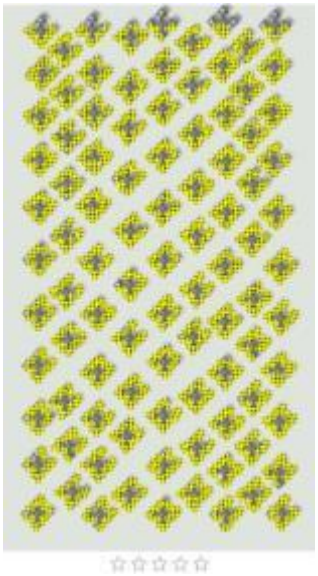


Figure 9 - Image considérée comme étant de mauvaise qualité par Vuforia - Documentation Vuforia, https://library.vuforia.com/features/images/image-targets/best-practices-for-designing-and-developing-image-based-targets.html.

La figure 9 est considérée par Vuforia comme étant de mauvaise qualité car il y a sur celle-ci une répétition claire des modèles affichés sur l'image. (PTC Inc., 2011a)

Il est important de comprendre que si nous voulons que Vuforia considère nos images comme étant de bonne qualité et donc qu'elles soient correctement exploitables, il faut qu'elles remplissent tous ces critères. Ils ne peuvent pas être indépendants les uns des autres. (PTC Inc., 2011a)

2.2. OpenCV for Unity

OpenCV for Unity est un plugin qui permet d'intégrer la librairie open source OpenCV (Open Source Computer Vision Library) à Unity. Cette librairie comporte une grande quantité d'algorithmes de reconnaissance d'images et de modules prévus à cet effet. Par exemple, OpenCV permet le traitement d'images ou d'objets ainsi que l'analyse de vidéos. Ce plugin est compatible avec les plateformes Android, iOS et Windows. Sa grande force pour la reconnaissance de couleurs réside dans sa classe « Mat ». Cette classe nous donne la possibilité de transformer une image en un tableau de pixels. Il est ensuite possible de parcourir ce tableau à l'aide d'une boucle et nous pouvons récupérer les couleurs de chaque pixel ainsi que leur emplacement sur l'image. (Intel, 2000b)

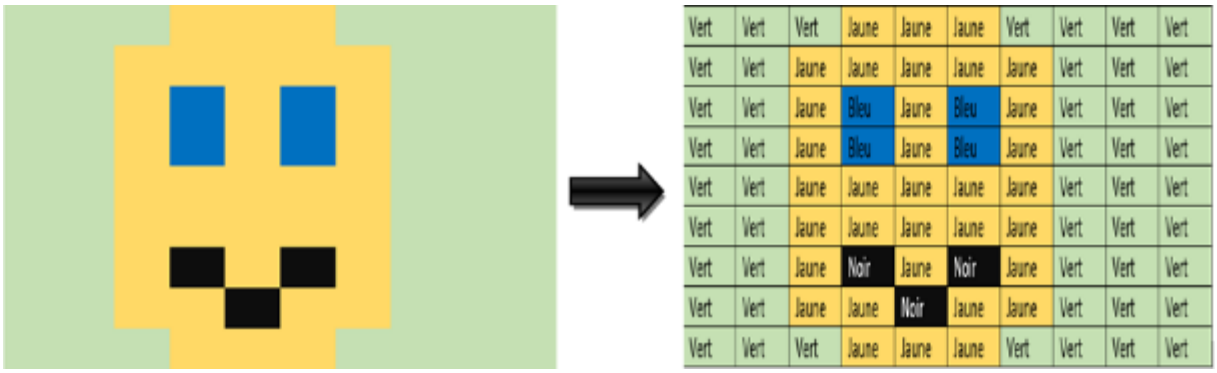


Figure 10 - Image passée dans une matrice - Image de l'auteure.

La figure 10 représente une image classique à gauche qui a été passée dans une matrice à droite. Sur cette image, nous pouvons voir les couleurs que possède celle-ci ainsi que leur emplacement.

OpenCV possède également une méthode appelée « inRange ». Celle-ci permet de trouver les nuances d'une couleur sur une image. Cela demande tout de même de traiter l'image sur laquelle une couleur doit être extraite. Cela se fait à l'aide d'un code qui utilise d'autres méthodes propres à OpenCV. (Intel, 2000c)

Le principal défaut de cette librairie est que le plugin Unity est payant. Lorsque nous cherchons à développer une application de réalité augmentée, il est courant d'utiliser plusieurs technologies. Dans un cas comme celui-là, si nous souhaitons utiliser Vuforia avec OpenCV for Unity, ces deux technologies doivent pouvoir accéder à la caméra de notre appareil, mais elles ne peuvent pas le faire en même temps. Nous devrons adapter le code de notre projet afin de gérer ce conflit. (Intel, 2000a)

2.3. EmguCV

EmguCV est un Wrapper gratuit de OpenCV pour .Net utilisable par les technologies Visual Studio et Unity. Il est compatible avec les plateformes Android, iOS et Windows. Le principal avantage qu'offre ce Wrapper est l'utilisation gratuite des modules d'OpenCV for Unity, mais en contrepartie, son installation sur Unity est beaucoup plus complexe. Nous devrons créer un dossier spécifique dans le projet et y ajouter manuellement certains fichiers du Wrapper EmguCV. (Emgu Corporation, 2008a)

Des parties de sa documentation ne sont pas à jour, et afin de pouvoir utiliser EmguCV avec d'autres technologie comme Vuforia, nous devrons adapter le code qui donne accès à la caméra. (Emgu Corporation, 2008b)

2.4. AR Foundation

AR Foundation est un Framework développé par Unity. Il est utilisé pour le développement d'application de réalité augmentée. Il est facilement implémentable dans Unity via le gestionnaire

de paquets. Il est également compatible avec les plateformes mobiles Android et iOS, mais également pour des lunettes de réalité augmentée comme Magic Leap et HoloLens. Il est important de comprendre que AR Foundation ne possède pas de fonctions de réalité augmentée en lui-même. Il utilise des plugins différents pour chaque plateforme. (Unity Technologies, 2018a)

AR Foundation est également utilisable avec le kit de développement Vuforia dans un même projet. Cependant, nous devons faire attention à la raison pour laquelle nous utilisons cette combinaison car elle peut baisser les performances de l'application. (PTC Inc., 2011d)

2.4.1. AR Core

Parmi les plugins utilisés par AR Foundation, il y a AR Core. C'est une plateforme créée par Google pour la création d'applications de réalité augmentée sur Android. Elle utilise la détection des mouvements afin garder en mémoire la position d'un téléphone après avoir fait une analyse de l'environnement filmé par la caméra. Il utilise égalament de la compréhension d'environnement afin de détecter les différentes surfaces filmées par la caméra d'un téléphone. Enfin, AR Core va également calculer la luminosité d'une scène qui est filmée par un téléphone ou une tablette afin d'évaluer la luminosité de l'environnement. (Nguyen, 2020)

2.4.2. AR Kit

Un autre plugin utilisé par Unity est AR Kit. C'est un Framework créé par Apple. Il permet le développement d'application de réalité augmentée sur iOS ou iPadOS. Tout comme AR Core, AR Kit utilise la détection des mouvements de notre téléphone, la compréhension de l'environnement et l'estimation de la luminosité. En plus de cela, AR Kit possède également une fonctionnalité de reconnaissance faciale. (Nguyen, 2020)

3. Organisation du projet

Cette partie traitera de l'organisation de ce travail de Bachelor. Nous avons séparé notre travail en trois parties. Une première partie documentation, une seconde partie rédactionnelle et une troisième partie pratique. Chacune de ces parties possède un product backlog. Ce product backlog contient le nom de la tâche à faire, son statut avec des indicateurs qui permettent de savoir si une tâche est en cours (orange), terminée (vert) ou n'a pas encore été commencée (rouge), une date à laquelle la tâche a été terminée et enfin, des notes sur la tâche.

3.1. Partie documentation

Dans cette partie, nous allons commencer par prendre connaissances des travaux de Bachelors qui ont été effectués précédemment. Cela va nous donner une idée de la manière de rédiger ce travail de Bachelor et ce qu'il doit contenir. Ensuite, nous allons nous documenter sur la réalité augmentée, cela va nous aider à savoir vers quelles technologies nous orienter pour rédiger l'état de l'art. Pour finir, nous irons lire la documentation des technologies que nous aurons jugées comme étant pertinentes pour ce travail de Bachelor ainsi que divers tutoriels sur la réalité augmentée, la détection de couleurs et la détection d'images de manière générale.

Partie Documentation			
Tâches	Statut	Fini le...	Notes sur la tâche
Lire de précédent TB		10.05.2021	
Se renseigner sur les technologies qui font de l'AR		13.05.2021	
Lire des livres, documents, thèses traitant de la réalité augmentée			
Lire documentation Unity			
Lire documentation AR Foundation			
Lire documenation OpenCV			
Lire documentation EmguCV			
Lire documentation Vuforia			
Lire des tutoriels sur la reconnaissance de couleurs par des applications			
Lire tutoriels sur la récupération des pixels sur une image			

Figure 11 - Product backlog de la partie documentation - Image de l'auteure.

La figure 11 représente les tâches que nous devons faire pour la partie documentation.

Les tâches du product backlog sur la figure 11 qui viennent après la deuxième tâche ont été ajoutées après que nous nous sommes renseignés sur l'existence de celles-ci. C'est pour cela qu'il y a déjà une date sur les deux premières tâches.

3.2. Partie rédactionnelle

Pour la partie rédactionnelle, nous commencerons par la tâche « Organisation du projet » afin d'avoir une idée d'ensemble claire de ce que nous devons faire. Nous ferons ensuite la tâche « Etat de l'art ». Le reste, jusqu'à la tâche « Résultats », se fera après le développement des nouvelles fonctionnalités de l'application. Nous écrirons ensuite les parties « Discussions » et « Conclusions » puis nous enchaînerons avec une relecture du document et des corrections. La partie « Introduction » se fera en dernier.

Partie Rédactionnelle			
Tâches	Statut	Fini le...	Notes sur la tâche
Organisation des différentes partie du TB		10.05.2021	
Introduction			
Etat de l'art			
Organisation du projet			
Choix technologique			
Résultat			
Discussion			
Conclusion			
Relectures			
Corrections			
Impression papier du TB			

Figure 12 - Product backlog de la partie rédactionnelle - Image de l'auteure.

La figure 12 représente les tâches que nous devons faire pour la partie rédactionnelle. Nous avons déjà la partie « Organisation des différentes parties du TB » du travail de Bachelor qui est terminée car elle a été faite avant le début de la rédaction de ce rapport. Quant à la partie « Organisation du projet » elle est en cours au moment où nous écrivons ces lignes.

3.3. Partie pratique

Pour cette partie, nous commencerons par suivre des tutoriels afin de que nous puissions nous familiariser avec la réalité augmentée. Nous créerons également de petites applications qui ne sont pas en lien avec ce travail de Bachelor. Par la suite, nous suivrons les étapes qui sont nécessaires aux fonctionnalités que nous devons ajouter à l'application.

Partie Pratique			
Tâches	Statut	Fini le...	Notes sur la tâche
Tutoriels AR capture et image tracking avec Unity			•Tutoriels hors TB pour comprendre l'AR
Tutoriels Vuforia avec Unity			•Tutoriels hors TB pour comprendre l'AR
Mise en couleurs des plans			
Division d'un plan en zones			
Création d'un algorithme de détection des couleurs			
Trouver la position d'une couleur sur un plan			
Lier les pièces d'un plan en trois dimensions avec les pièces déssinées sur un plan			
Mise en couleurs des pièces en trois dimensions			
Mise en couleurs des pièces en trois dimensions selon les couleurs affichées sur un plan filmé			

Figure 13 - Product backlog de la partie pratique - Image de l'auteure.

La figure 13 représente les tâches que nous devons faire pour la partie pratique. Elles ont été définies après avoir pris connaissance des technologies existantes dans le domaine de la réalité augmentée ainsi que les différents tutoriels en matière de détection de couleurs et de traitement d'images. Sans cela, nous n'aurions pas pu définir à l'avance quelles technologies utiliser pour cette partie.

4. Choix technologique

Afin de choisir la ou les technologies qui vont être utilisé pour la réalisation de ce travail de Bachelor, nous nous somme basés sur plusieurs critères. Ces critères sont la compatibilité avec Windows, Android et iOS, la facilité d'intégration de la technologie dans Unity, la possibilité que la technologie possède des modules capables de faire de la reconnaissance de couleurs telle qu'une méthode ou une classe, la possibilité que la technologie possède d'autres modules qui pourraient nous aider à faire de la reconnaissance de couleurs, la possibilité que la technologie soit bien documentée et enfin, la possibilité que la technologie soit gratuite.

Nous avons tout d'abord regroupé les technologies qui nous intéressaient puis nous les avons comparées sur la base de ces critères.

4.1. Matrice de comparaisons

Technologies	Compatibilité avec Windows	Compatibilité avec Android	Compatibilité avec iOS	Facilité d'intégration avec Unity	Modules de détection des couleurs	Autres modules pouvant aider à la création d'un algorithme de détection des couleurs	Bonne documentation	Produit gratuit
vuforia	✓	✓	✓	✓	✗	✓	✓	✓
OpenCV	✓	✓	✓	✓	✓	✓	✓	✗
EMGU	✓	✓	✓	✗	✓	✓	✗	✓
ARKit	✗	✗	✓	✓	✗	✗	✓	✓
ARCore	✗	✓	✗	✓	✗	✗	✓	✓

Figure 14 - Matrice de comparaison des technologies - Image de l'auteure.

La figure 14 représente une matrice de comparaison des technologies qui ont été citées dans la partie 2 « Etat de l'art ».

4.2. Le moteur de jeu

4.2.1. Unity

Unity est un moteur de jeu créé par la société américaine Unity Technologie. Il est principalement utilisé pour la création de jeux vidéo en deux ou trois dimensions, mais il offre également la possibilité de concevoir des applications qui utilisent la réalité augmentée. Il utilise principalement le langage de programmation C#. (Unity Technologies, 2004b)

Nous avons choisi Unity car il a été utilisé lors de la réalisation du travail de Bachelor précédent. Afin de pouvoir ajouter la fonctionnalité de mise en couleurs des modèles de tuyaux en trois dimensions, nous devons posséder une technologie capable de gérer ce type d'objets. Si nous ne nous basons pas sur la technologie utilisée lors du développement du précédent projet, nous risquons de perdre les fonctionnalités déjà implantées dans l'application. Celles-ci sont nécessaires à la réalisation de ce travail de Bachelor puisque nous devons y ajouter une fonctionnalité qui les impacte directement.

4.2.2. Unreal Engine

Unreal Engine est un moteur de jeu créé par la société Epic Game. Il est principalement utilisé pour la création de jeux vidéo, mais il offre également la possibilité de créer des applications de réalité augmentée. Il utilise principalement le langage de programmation C++. (Epic Game, 2004)

Si nous le comparons à Unity, il est son principal concurrent. Cependant, Unreal Engine ne supporte pas le langage de programmation utilisé pour réaliser le précédant travail de Bachelor. Nous devrions donc le modifier.

4.3. Le langage de programmation

4.3.1. C#

C# est un langage de programmation principalement utilisé pour le développement d'applications. (Microsoft, 1975)

Nous avons choisi C# comme langage de programmation pour la réalisation de ce travail de Bachelor car il est le principal langage utilisé par Unity. Il a également été utilisé pour la réalisation du travail de Bachelor précédent sur lequel nous devons ajouter des fonctionnalités.

4.4. Technologies

Lorsque nous analysons la matrice de comparaison que nous avons créée, nous constatons que deux technologies obtiennent le même score, Vuforia et le plugin OpenCV for Unity. Parmi celles-ci, nous avons choisi Vuforia pour les raisons suivantes.

Tout d'abord ce SDK est nécessaire à l'application car il a été utilisé dans le précédent projet pour la reconnaissance des plans de l'ORIF sans leurs couleurs. Il a également été employé afin de réaliser l'affichage des modèles de tuyaux en trois dimensions au moment où l'application reconnait un plan. Nous ne pouvons pas nous passer de cette technologie car nous avons besoin des modèles de tuyaux en trois dimensions si nous voulons leur ajouter des couleurs.

Ensuite, il y a l'utilisation de la caméra. En effet, Vuforia et OpenCV for Unity sont deux technologies qui utilisent la caméra de l'appareil sur lequel notre application est installée.

Cependant, ces deux technologies ne peuvent pas l'utiliser en même temps. Nous devrions désactiver Vuforia pour utiliser OpenCV for Unity et vis-versa. Malheureusement, nous avons constamment besoin que Vuforia soit actif pour la détection d'un plan et l'affichage de formes en trois dimensions. C'est pourquoi nous n'utiliserons pas OpenCV pour la réalisation de ce travail.

Les technologies EmguCV, AR Kit et AR Core ne correspondent pas suffisamment à nos besoins, c'est pourquoi elles ne seront pas utilisées lors de ce travail.

5. Résultats

Cette partie de ce travail de Bachelor traitera de la programmation des fonctionnalités que nous devons ajouter à un précédent travail de Bachelor, mais également des problèmes rencontrés lors de son développement ainsi que les tests effectués après la mise en place des nouvelles fonctionnalités.

5.1. Images de la caméra

Lorsque nous filmons une scène avec la caméra d'un téléphone portable, celui-ci nous renvoie en réalité une série d'images (figure 15).

Figure 15 - Décomposition en images d'un film - Image du cours de java avancé, par David Russo.

Dans l'une des étapes de ce travail de Bachelor, nous devons filmer un plan et faire en sorte que l'application puisse trouver les couleurs qui se trouvent sur ce plan. En réalité, l'algorithme de détection des couleurs ne va pas analyser la vidéo en elle-même mais les images qui ressortent de cette vidéo.

La première étape du développement de nos fonctionnalités consiste à récupérer chaque image de la caméra. Comme nous l'avons mentionné dans la partie « Etat de l'art » pour la technologie Vuforia, celle-ci possède une fonctionnalité qui permet de récupérer chaque image que nous renvoie la caméra. Cette fonctionnalité est une méthode appelée « GetCameraImage ».

```
Vuforia.Image image = CameraDevice.Instance.GetCameraImage(mPixelFormat);
```

Figure 16 - Méthode GetCameraImage - Image de l'auteure.

La figure 16 représente la méthode appelée « GetCameraImage » qui permet de récupérer chaque image de la caméra.

Une fois les image récupérées, l'algorithme de détection des couleurs peut désormais travailler sur chacune d'elles indépendamment des autres.

5.2. La position des éléments sur une image

Puisque dans ce travail de Bachelor des pièces en trois dimensions doivent posséder les mêmes couleurs que celles qui sont affichées sur le plan en trois dimensions correspondant à ces pièces, il est important que l'application connaisse l'endroit où se trouvent les couleurs lorsque la caméra du téléphone filme le plan.

Nous avons fait face à plusieurs problèmes durant la réalisation de cette tâche. Pour commencer, il existe plusieurs moyens de déterminer la position d'un élément sur une image. Vuforia se charge tout seul de trouver la position de l'image sans ses couleurs dans la scène que filme un téléphone.

5.2.1. La matrice

En programmation, une matrice est de façon conceptuelle une sorte de tableau en deux dimensions. Dans le langage C# qui est utilisé dans ce travail de Bachelor, une matrice n'a pas de nom, mais un type. (Microsoft, 1975)

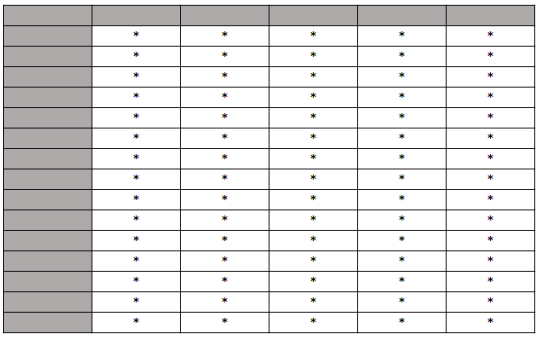


Figure 17 - Exemple conceptuel de tableau - Image de l'auteure.

La figure 17 représente un tableau à double entrées qui peut être interprété comme une image conceptuelle d'une matrice en deux dimensions.

Dans ce projet, le type de matrice appelé « Color32 » est utilisé (figure 18).

```
Color32[,] imageMat;
```

Figure 18 - Exemple de matrice dans le langage C# - Image de l'auteure.

L'image de la caméra d'un téléphone ou d'une tablette va pouvoir être passée à l'intérieur de cette matrice afin que l'algorithme de détection des couleurs puisse parcourir tous les pixels qui sont contenus dans l'image.

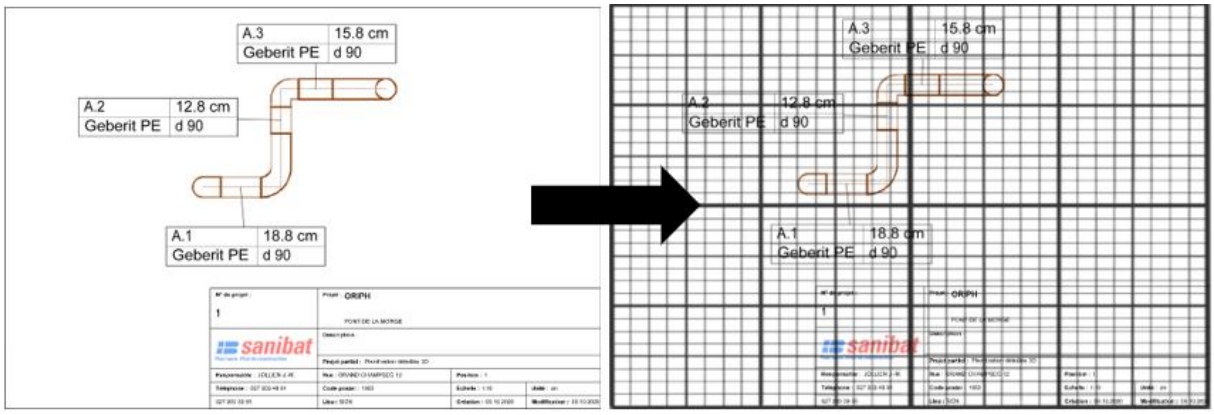


Figure 19 - Exemple conceptuel d'un plan de l'ORIF passé dans une matrice - Image de l'auteure.

Chaque case de la figure 19 représente de manière conceptuelle les pixels de l'image. L'algorithme de détection des couleurs va désormais pouvoir passer sur chaque pixel du plan et déterminer sa couleur.

L'algorithme de détection des couleurs va parcourir ce tableau grâce à deux indicateurs dans notre code. Ceux-ci représenteront la hauteur et la largeur du tableau, un peu à la manière d'un axe des x et d'un axe des y. Les carrés de ce tableau, représenteront la position en x et en y d'un pixel sur l'image.

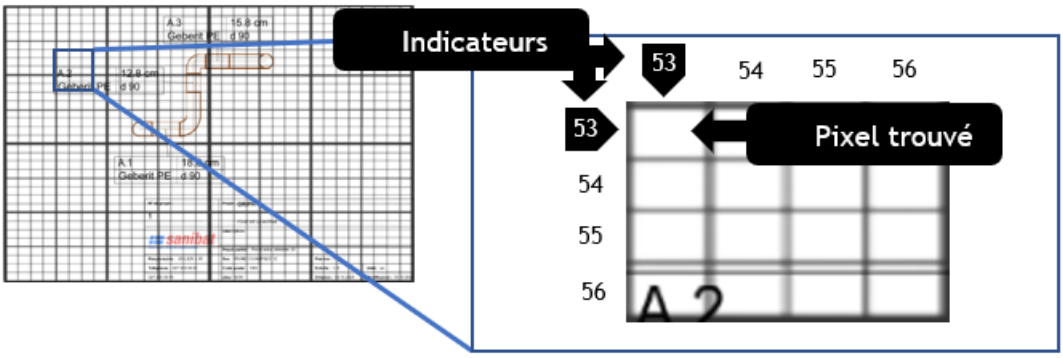


Figure 20 - Recherche d'un pixel sur une image d'un plan de l'ORIF passé dans une matrice - Image de l'auteure.

La figure 20 représente un schéma conceptuel de la recherche d'un pixel dans une matrice.

Après avoir effectué plusieurs tests avec cette méthode, nous avons constaté qu'il y avait des problèmes de performances avec l'application. En effet, une image contient un grand nombre de pixels. En prenant une image prise par la caméra de notre ordinateur, nous avons pu déterminer qu'elle contenait 307'200 pixels. L'algorithme de détection des couleurs doit parcourir chacun de ces pixels et analyser leur couleur, ce qui demande beaucoup de ressources au programme.

Sur l'image qui est renvoyée par la caméra de notre appareil, ce qui nous intéresse est l'image du tuyau qui se trouve sur le plan de l'ORIF. Avec la méthode qui consiste à parcourir l'entièreté de l'image qui est prise par la caméra de notre appareil, nous avons constaté que beaucoup de zones sur l'image en question ne servent à rien pour l'algorithme de détection des couleurs puisqu'elles ne contiennent pas l'image du tuyau. Cela veut dire que le programme va effectuer un grand nombre de recherches inutiles.

Afin d'améliorer la performance de l'application, nous devons réduire la zone de recherche. Ce qui nous intéresse réellement, c'est le plan de l'ORIF. Nous avons donc cherché à savoir s'il était possible de trouver et d'isoler ce plan afin de réduire la zone de recherche pour l'algorithme de détection des couleurs.

5.2.2. L'image cible

Les images que nous avons stockées dans la base de données de Vuforia on comme nom « image target » ou image cible en français. Ce sont ces images que Vuforia va pouvoir reconnaître lorsque nous filmons une scène qui contient un plan.

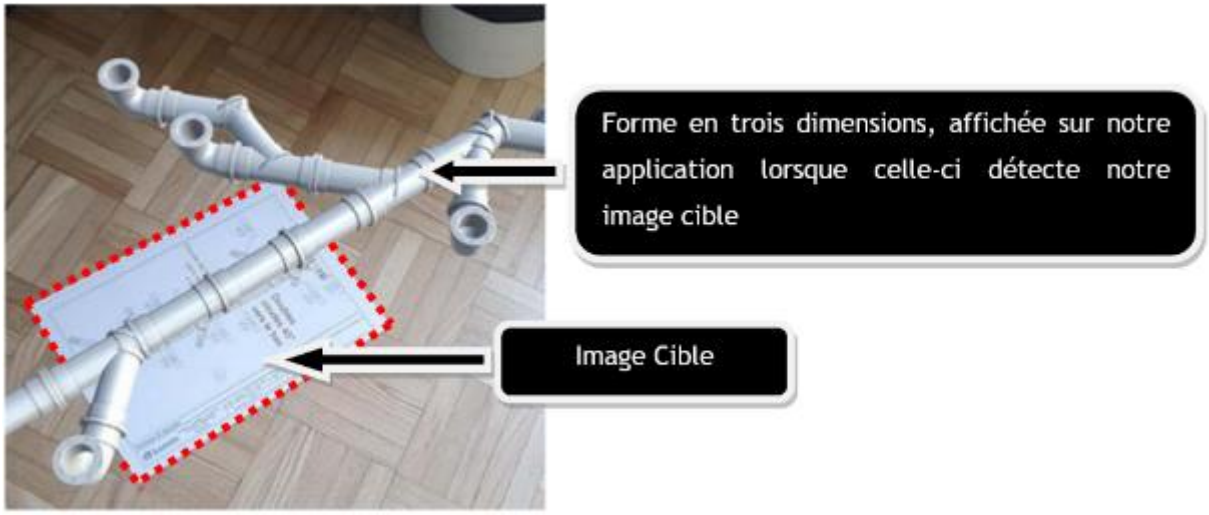


Figure 21 - Illustration de la fiche d'information du travail de Bachelor - Image modifiée par l'auteure.

La figure 21 représente une démonstration de l'application avec une image cible et sa forme en trois dimensions qui est affichée par-dessus le plan.

Comme nous l'avons mentionné dans la partie « Etat de l'art » de Vuforia, ce SDK ne reconnaît pas les couleurs, mais nous savons que celles-ci se trouvent sur nos images cibles. Nous savons également que notre image cible se trouve dans la scène que filme notre appareil. Avec ces informations, nous avons conclu que nous devions effectuer notre recherche dans une zone spécifique de la scène, celle où se trouve l'image cible. Unity possède des méthodes qui peuvent permettre à l'application de

trouver la position d'une image cible lorsque celle-ci est détectée. L'une de ces méthodes se nomme « transform.position ».

```
float positionX = imageCible.transform.position.x;
float positionY = imageCible.transform.position.y;
float positionZ = imageCible.transform.position.z;
```

Figure 22 - Méthodes « transform.position » - Image de l'auteure.

La figure 22 représente trois méthodes utilisées par Unity pour définir la position en x, y et z d'un objet.

Le problème avec cette méthode est le type de la valeur que nous donne Unity lorsque l'application trouve l'image cible. Afin de comprendre pourquoi le type de la valeur pose un problème à l'algorithme de détection des couleurs, nous devons tout d'abord comprendre la différence entre la position d'un élément sur une image et la position d'un élément dans une scène.

5.2.3. L'objet Vector3

L'objet appelé « Vector3 » (figure 23) est dans Unity un objet qui représente une structure en trois dimensions avec ses axes x, y et z ainsi que tous les points compris sur ces axes. Cette structure sert à définir la position d'un objet en trois dimensions dans un espace qui est, dans notre cas, numérique. (Unity Technologies, 2020e)

```
Vector3 planPosition = new Vector3(positionX, positionY, positionZ);
```

Figure 23 - Objet Vector3 - Image de l'auteure.

Comme nous l'avons expliqué un peu plus haut dans la partie qui traite des images de la caméra, Vuforia récupère une image en particulier sur laquelle l'algorithme de détection des couleurs va pouvoir travailler. Une image est en elle-même une série de pixels regroupés. Notre but est que l'algorithme de détection des couleurs puisse parcourir cette série de pixels et analyser les couleurs afin de définir quelle pièce d'un plan possède quelle teinte. Avec les méthodes « transform.position », l'application récupère en réalité la position d'un objet dans un espace en trois dimensions. Ce n'est pas la même chose que de récupérer un pixel qui se trouve sur une image en deux dimensions.

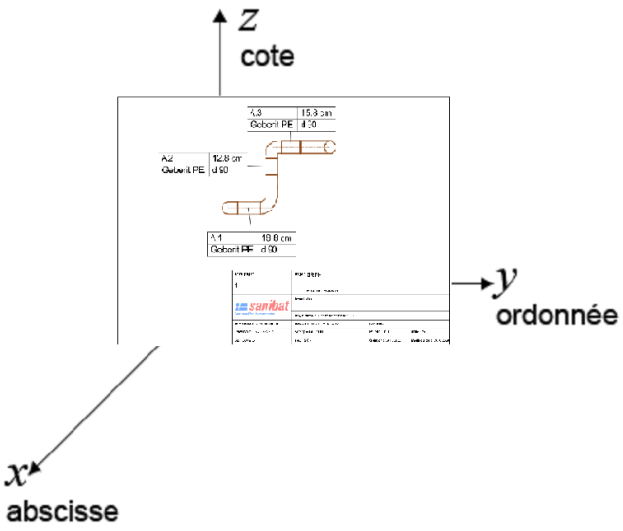


Figure 24 - Position d'un plan dans l'espace sur les axes x, y et z - Image de l'auteure.

La figure 24 représente un plan dans un espace en trois dimensions. Cet espace est composé de trois axes définis par les lettres x, y et z. Une position qui est récupérée dans un espace peut être soi négative, soi positive selon si l'objet ou la partie de l'objet se trouve sur ou sous un axe. La position d'un pixel dans une matrice est toujours positive car une matrice commence à zéro.

Nous comprenons désormais que la position du plan dans l'espace en trois dimensions n'est pas la même que la position d'un pixel sur une image en deux dimensions. Puisque nous cherchons à faire en sorte que l'algorithme de détection des couleurs puisse travailler sur un plan en deux dimensions, nous devrions plutôt nous tourner vers l'objet prévu à cet effet qui est dans Unity l'objet appelé « Vector2 ».

5.2.4. L'objet Vector2

L'objet appelé « Vector2 » est dans Unity un objet qui représente une structure en deux dimensions avec ses axes x et y ainsi que tous les points compris sur ces axes (figure 25). Cette structure sert à définir la position d'un objet en deux dimensions dans un espace qui est dans notre cas, numérique. (Unity Technologies, 2020d)

```
Vector2 planPosition = new Vector2(positionX, positionY);
```

Figure 25 - Objet Vector2 - Image de l'auteure.

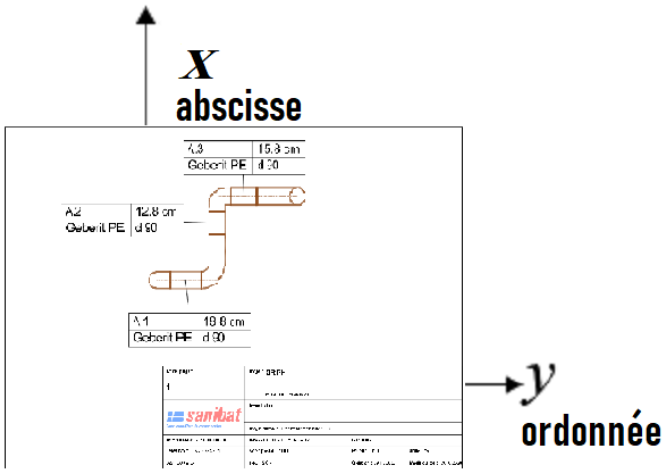


Figure 26 - Position d'un plan dans l'espace sur les axes x et y - Image de l'auteure.

La figure 26 représente un plan dans un espace en deux dimensions. Cet espace est composé de deux axes définis par les lettres x et y.

Cependant, avec cette solution, le programme rencontre un problème similaire à l'objet « Vector3 ». Même si désormais l'objet ne possède plus que deux axes, l'image cible se trouve toujours dans un espace. Les coordonnées de l'image peuvent être soi négative, soi positive. Or, cet espace ne possède pas les mêmes coordonnées qu'une matrice. Le programme ne peut donc pas combiner cette matrice avec l'objet « Vector2 ».

5.2.5. Le découpage de l'image en zones

Le programme n'a pas pu combiner la détection de l'image cible avec la solution de recherche de pixels dans une matrice. Nous avons donc décidé de définir des zones de recherche dans le tableau de pixels afin de réduire la quantité de recherche que doit effectuer le programme.

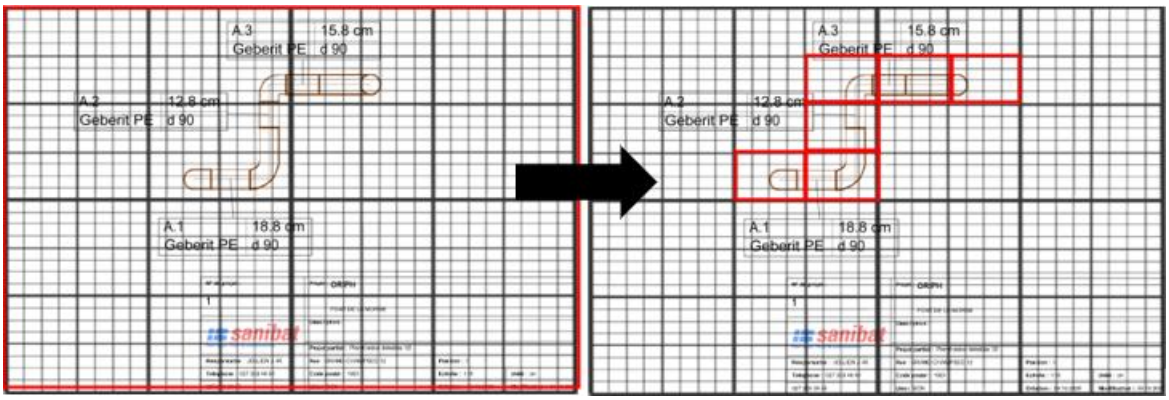


Figure 27 - Plan découpé en zones - Image de l'auteure.

La figure 27 représente le découpage d'une image d'un plan de l'ORIF en zones de recherches.

Cependant, lorsque l'appareil filme un plan avec cette méthode, l'application ne sait pas où se trouve l'image cible, elle n'a qu'un tableau de pixels à sa disposition. C'est pourquoi nous devons faire attention à bien centrer notre plan lorsque nous le filmons tout en faisant en sorte que celui-ci couvre l'entièreté de l'écran de l'appareil. De cette façon, nous pouvons faire en sorte que les zones de recherches définies dans notre code correspondent au mieux aux endroits où se trouvent les pièces du tuyau.

5.3. Extraction des couleurs d'un pixel

Maintenant que le programme possède un tableau de pixels et des zones de recherche, l'algorithme de détection des couleurs peut enfin analyser leurs couleurs. Tout d'abord, lorsque cet algorithme parcourt un tableau et s'arrête sur un pixel, il tombe sur un nombre qui désigne sa couleur de ce pixel. L'algorithme de détection des couleurs peut désormais extraire le codage des couleurs de notre pixel.

5.3.1. Le codage des couleurs

Sur un ordinateur, les couleurs ont un système de codage appelé RGB (Red, Green, Blue). Une image est un assemblage de pixels et chacun de ces pixels possède trois valeurs numériques qui, lorsque nous les mettons ensemble, forment une couleur.

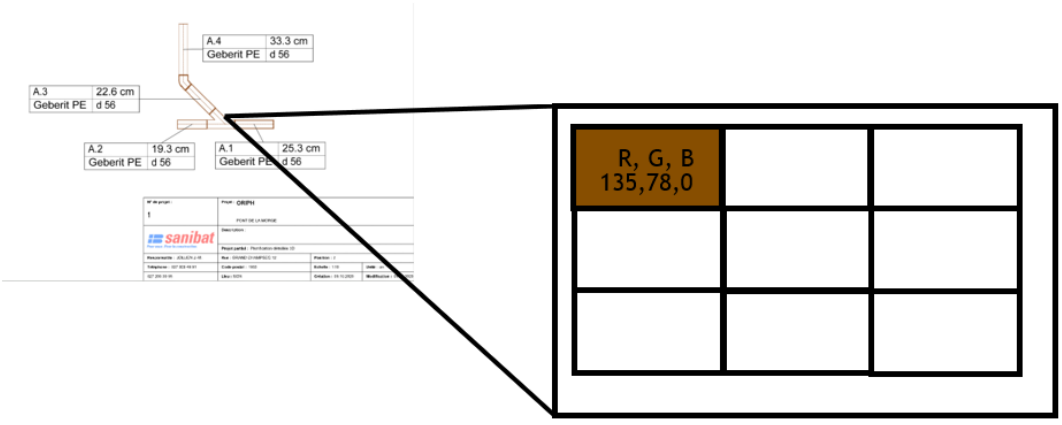


Figure 28 - Pixel avec son code couleur RGB - Image de l'auteure.

La figure 28 représente un pixel avec son code couleur RGB sur un des plans de l'ORIF.

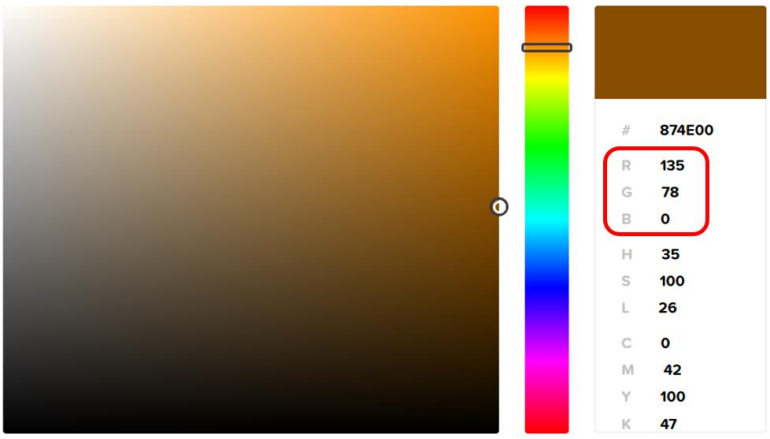


Figure 29 - Sélecteur de couleurs - Capture d'écran prise sur https://htmlcolorcodes.com/fr/.

La figure 29 représente un sélecteur de couleurs avec les valeurs RGB mises en évidence.

Afin que l'application puisse connaître la couleur d'un pixel grâce à l'algorithme de détection des couleurs, une méthode doit extraire les valeurs RGB d'un pixel.

5.3.2. L'algorithme d'extraction des couleurs

Afin d'extraire les trois valeurs d'un pixel, le programme utilise une méthode. Le but de cette méthode est de convertir un vecteur de bytes qui est un tableau en une dimension, en un vecteur de type « Color32 ». Le type « Color32 » va permette à l'algorithme de détection des couleurs de récupérer les valeurs RGB d'une image.

Pour commencer, tous les pixels de l'image d'une caméra sont passés dans un vecteur. Cela se fait grâce à une méthode qui est fournie par Vuforia et qui a pour nom « Image.Pixels » (figure 30).

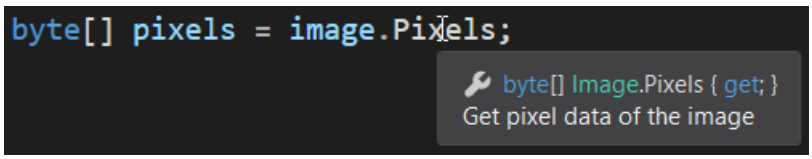


Figure 30 - Méthode "Image.Pixels" - Image de l'auteure.

Cette méthode possède le type « byte[] » qui va permettre à l'algorithme de détection des couleurs de récupérer la valeur numérique des pixels.

Ensuite, une méthode qui va prendre en paramètre le vecteur de bytes est utilisée.

Figure 31 - Méthode « GetColorArray » - Image de l'auteure.

La figure 31 représente la méthode pour extraire les valeurs RBG des pixels d'une image qui a été passée dans un vecteur de type "byte[]".

Cette méthode va commencer par créer un vecteur de type « Color32 ». Ce type va permettre à l'algorithme de détection des couleurs de récupérer les valeurs RGB des pixels de notre image.

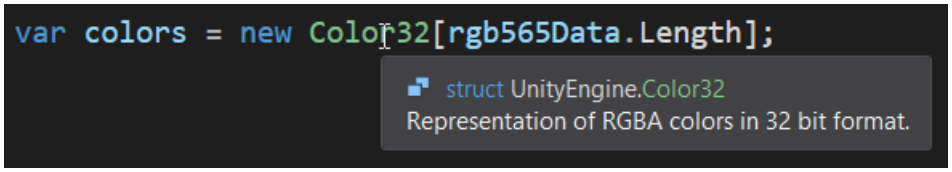


Figure 32 - Vecteur de type "Color32" - Image de l'auteure.

La figure 32 représente une variable qui prendra le type « Color32[] ». Ce type est un vecteur.

Puis, ce vecteur va être parcouru grâce à une boucle, byte par byte et les valeurs RGB de ces bytes vont être extraites.

```
for (var i = 0; i < rgb565Data.Length; i += 2)
{
    colors[i / 2] = new Color32((byte)(rgb565Data[i] & 0xF8),
      (byte)(((rgb565Data[i] & 7) << 5) | ((rgb565Data[i + 1] & 0xE0) >> 3)),
      (byte)((rgb565Data[i + 1] & 0x1F) << 3),
      (byte)1);
}
```

Figure 33 - Boucle pour extraire les valeurs RGB - Image de l'auteure.

La figure 33 représente la boucle qui va extraire les valeurs RGB de chaque pixel du vecteur appelé "colors" qui a été défini précédemment dans le code

Pour finir, le résultat est un vecteur de type « Color32 ». Par la suite, ce tableau, qui possède une seule dimension, sera retransformé en une matrice afin d'avoir à nouveau une image en deux dimensions (figure 34).

```
int k = 0;

imageMat = new Color32[image.Width, image.Height / 2];


for (int i = 0; i < image.Height / 2; i++)
{
    for (int j = 0; j < image.Width; j++)
    {
        imageMat[j, i] = colorArray[k];

        k++;
    }
}
```

Figure 34 - Transformation d'un vecteur en une dimension en une matrice en deux dimensions - Image de l'auteure.

5.4. Mise en couleurs des formes en trois dimensions

Cette partie traitera de la mise en couleurs des formes en trois dimensions ainsi que de la façon dont les couleurs sont détectées sur une image.

5.4.1. L'algorithme de recherche d'une couleur

Maintenant que le programme a la possibilité d'accéder aux valeur RGB des pixels et que l'image que renvoie la caméra de l'appareil se trouve dans une matrice en deux dimensions, l'algorithme de détection des couleurs peut parcourir les pixels de l'image et analyser leurs couleurs.

Pour commencer, un intervalle pour chaque couleur que nous souhaitons détecter a été défini à l'intérieur du code. Comme nous l'avons vu précédemment, pour un ordinateur, une couleur est le mélange de trois valeurs numériques. Si nous voulons que l'algorithme puisse détecter la couleur rouge par exemple, nous allons devoir définir quelle sorte de rouge nous souhaitons que l'application détecte. Pour cela, un rouge maximal et un rouge minimal vont devoir être définis.

Pour définir un rouge maximal, l'objet appelé « Color » doit être créé dans le code. Cet objet est tout simplement une couleur. Une valeur RGB doit ensuite être donnée comme paramètre à cet objet. Ce processus est le même pour définir un rouge minimal.

```
Color higher_red = new Color(255, 100, 100);
Color lower_red = new Color(100, 0, 0);
```

Figure 35 - Objets de type « Color » avec leurs valeurs RGB - Image de l'auteure.

La figure 35 représente deux objets de type "Color" avec leurs valeurs RGB qui correspondent à du rouge.

Ensuite, l'algorithme de détection des couleurs doit vérifier si les valeurs RGB du pixel qu'il doit analyser se trouvent bien dans l'intervalle entre les valeurs du rouge maximal et du rouge minimal.

```
if (imageMat[j, i].r <= higher_red.r && imageMat[j, i].r >= lower_red.r
    && imageMat[j, i].g <= higher_red.g && imageMat[j, i].g >= lower_red.g
    && imageMat[j, i].b <= higher_red.b && imageMat[j, i].b >= lower_red.b)
```

Figure 36 - Condition de vérification pour la couleur rouge - Image de l'auteure.

La figure 36 représente la condition de vérification des valeurs RGB d'un pixel pour la couleur rouge.

Un intervalle de couleurs sur une image doit être défini, car le programme travaille avec un haut niveau de précisions. Nos yeux d'humain voient du rouge de manière générale, mais un ordinateur fera la différence entre deux pixels rouges dont l'un, par exemple, qui possède les valeurs 255,100,100 et un autre pixel qui possède les valeurs 255,100,101, même si la variation entre ces deux pixels n'est que de 1 sur la valeur du bleu. Cela veut dire que tous les pixels rouges que l'on peut trouver sur une image n'ont pas tous la même valeur RGB.

Une fois qu'un pixel a été trouvé, l'algorithme de détection des couleurs retient sa position et il termine sa première recherche.

```
positionI = i;
positionJ = j;
break;
```

Figure 37 - Position d'un premier pixel qui a été trouvé - Image de l'auteure.

La figure 37 représente la position d'un premier pixel qui a été trouvé

En partant de cette position, l'algorithme de détection des couleurs va regarder s'il trouve d'autres pixels qui viennent à la suite de celui qu'il vient de trouver.

```
for (int i = positionI; i < 120; i++)
{
    for (int j = positionJ; j < 480; j++)
    {
```

Figure 38 - Boucles de recherche de pixels en partant d'un pixel qui a été trouvé - Image de l'auteure.

La figure 38 représente deux boucles imbriquées qui servent à la recherche de pixels en partant d'un pixel qui a été trouvé.

Dès lors que l'algorithme de détection des couleurs a fait cette comparaison et que celle-ci ressort positive, il va devoir vérifier qu'une certaine quantité de pixels se trouve bel et bien entre le rouge maximal et le rouge minimal. Cela sert à éviter que le programme ne trouve un pixel rouge isolé dans une zone de recherche et qu'il interprète cela comme étant une pièce du plan qui a été coloré en rouge.

```
if (cptRed >= 40)
```

Figure 39 - Condition qui permet de vérifier si une certaine quantité de pixels rouge a été trouvée sur une image - Image de l'auteure.

La figure 39 représente la condition qui permet de vérifier si une certaine quantité de pixels rouge a été trouvée sur une image.

Pour finir, lorsque ces deux conditions sont remplies, la forme de notre tuyau en trois dimensions peut être affichée avec la couleur indiquée sur le plan que filme l'appareil.

5.4.2. Les textures d'une forme en trois dimensions

Dans Unity, un objet en trois dimensions possède un « Renderer ». C'est ce qui permet à un objet d'être visible sur une scène filmée. (Unity Technologies, 2018c)

C'est également avec cela que le programme va pouvoir accéder au matériel de la forme en trois dimensions. Ce matériel est une class appelé « Material » dans Unity. Cette classe va nous servir à changer la couleur d'une pièce d'un tuyau en trois dimensions. (Unity Technologies, 2018)

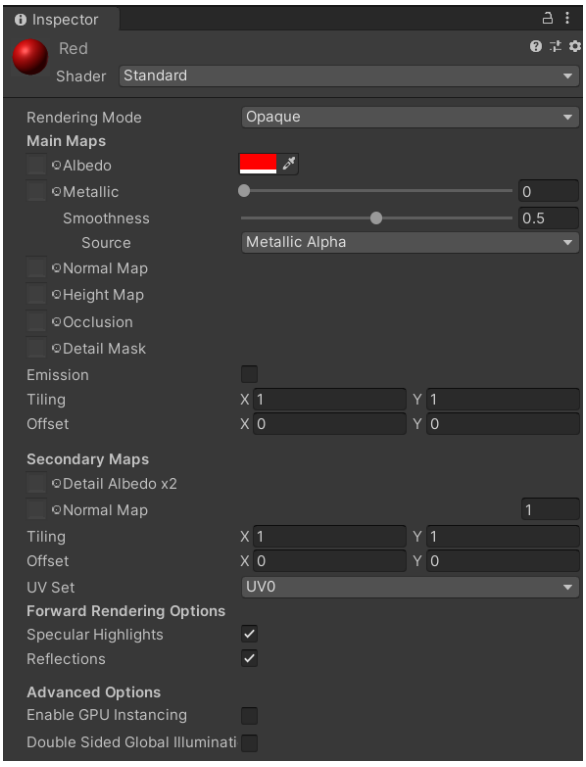


Figure 40 - Matériel rouge dans Unity - Image de l'auteure

La figure 40 représente un matériel rouge dans Unity dans l'inspecteur de Unity.

```
piece.GetComponent<Renderer>().material = redMaterial;
```

Figure 41 - Methode « GetComponent » - Image de l'auteure.

La figure 41 représente la méthode qui sert à récupérer le "Renderer" et le "Material" d'une pièce de tuyau en trois dimensions ainsi que son changement de couleur en rouge grâce à la variable "redMaterial"

5.5. Les tests

Dans cette partie, nous allons tester l'application sur trois plans différents de l'ORIF dont les pièces ont été mises en couleur. Nous allons regarder l'efficacité de notre application ainsi que la vitesse à laquelle celle-ci détecte les couleurs sur une forme.

Ces tests vont prendre en compte la quantité de pièces qu'un plan possède, leur taille, la quantité de pixels que le programme doit trouver afin de définir si une couleur a été trouvée ou non, et enfin, la plage de couleur qui va être comparée à la couleur trouvée sur un plan. L'application sera testée sur un Samsung Galaxy S8.

5.5.1. Plan numéro un

Le premier plan est composé de sept pièce, toutes colorées.

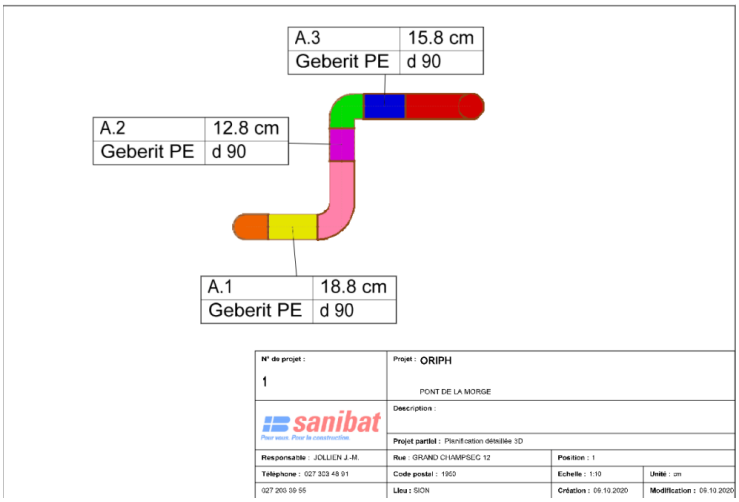


Figure 42 - Plan numéro un de l'ORIF mis en couleur - Image de l'auteure.

La figure 42 représente un des nombreux plans de tuyauterie de l'ORIF. Celui-ci a été mis en couleurs.

Nous avons effectué un premier teste en demandant au programme de vérifier qu'une quantité d'au moins dix pixels d'une même couleur ait été trouvée dans une zone spécifique de l'image renvoyé par la caméra de l'appareil.

Après avoir effectué une vingtaine de tests sur ce plan, nous avons constaté que l'application trouvait rapidement les couleurs affichées. Par rapidement, nous avons une mesure qui se situe entre 1 et 3 secondes en moyenne. Nous avons pris en compte le temps de cadrer correctement la caméra afin que le plan prenne la totalité de l'écran. Cependant, nous avons remarqué que le programme prenait presque toujours plus de temps à trouver la couleur jaune et la couleur verte sur le plan. Nous avons un nombre de secondes plus élevé que 3 dans ce cas-là. Entre ces deux couleurs, nous avons constaté que la couleur verte était très souvent la dernière à être trouvée.

Nous avons cherché à comprendre pourquoi l'application rencontrait plus de difficultés à reconnaître ces deux couleurs. Nous avons commencé par le jaune. Nous avons constaté que l'intervalle de couleur que cherche l'algorithme de détection des couleurs n'était pas très bon. Nous l'avons donc changé et nous avons recommencé une série de tests. Avec ce nouveau réglage sur la couleur jaune, nous avons remarqué que l'application arrivait, cette-fois ci, à trouver beaucoup plus rapidement la couleur jaune.

Jaune			
	R	G	B
Max	255	255	0
Min	165	165	0

Jaune			
	R	G	B
Max	255	255	100
Min	200	200	0

Figure 43 - Changement des valeurs RGB pour la couleur jaune - Image de l'auteure.

La figure 43 représente les changements des valeurs RGB pour la couleur jaune représenté par des tableaux.

Pour la pièce de couleur verte, nous avons vérifié l'intervalle comme pour le jaune. Nous l'avons légèrement changé car cet intervalle ne nous semblait pas très bon. Après ce changement, nous avons à nouveau effectué une série de tests. Cette fois, la couleur verte était trouvée autant rapidement que les autres couleurs par l'application.

Vert			
	R	G	B
Max	55	255	55
Min	0	165	0

Vert			
	R	G	B
Max	100	255	100
Min	0	200	0

Figure 44 - Capture d'écran du changement des valeurs RGB pour la couleur verte représenté par des tableaux - Image de l'auteure.

La figure 44 représente les changements des valeurs RGB pour la couleur verte représenté par des tableaux.

Pour la suite de nos tests sur ce plan, nous avons augmenté la quantité de pixels que le programme doit trouver afin que celui-ci puisse définir qu'une couleur a été trouvée ou non. Nous avons fait passer cette valeur de 10 à 30.

Nous avons pu constater que l'application avait de la peine à reconnaître les couleurs orange, jaune et à deux occasions le violet sur une série de vingt tests. Nous avons donc commencé par regarder l'intervalle de ces deux couleurs tout comme nous l'avions fait pour le test précédent.

Nous avons commencé par le jaune puisque dans le plan testé la plus grande des pièces qui pose problème était de cette couleur. Nous avons changé ses valeurs RGB afin de recalibrer l'intervalle de la couleur jaune. Nous avons constaté un résultat positif, et l'application a été capable de détecter plus rapidement la couleur jaune.

Jaune			
	R	G	B
Max	255	255	100
Min	200	200	0

Jaune			
	R	G	B
Max	255	255	100
Min	220	220	0

Figure 45 - Changement supplémentaire des valeurs RGB pour la couleur - Image de l'auteure.

La figure 45 représente un changement supplémentaire des valeurs RGB pour la couleur jaune représenté par des tableaux.

Nous avons fait de même pour la couleur orange. Cette couleur nous a demandé plus de recalibrage sur l'application pour la détecter que la couleur jaune.

Orange			
	R	G	B
Max	255	137	75
Min	200	82	0

Orange			
	R	G	B
Max	255	160	111
Min	201	105	0

Figure 46 - Changement des valeurs RGB pour la couleur orange - Image de l'auteure.

La figure 46 représente un changement des valeurs RGB pour la couleur orange représenté par des tableaux.

Nous n'avons pas touché au violet puisque les tests que nous avons effectué avec l'application ne nous ont pas donné un résultat significatif pour cette couleur. De manière générale, nous avons remarqué que le programme prenait quelques secondes de plus, une à deux en moyenne, à trouver les couleurs sur le plan avec cette nouvelle configuration.

Dans notre troisième test, nous avons demandé au programme de vérifier 50 pixels d'une couleur dont les valeurs RGB sont comprises dans un intervalle que nous avons défini. Nous avons fait une vingtaine de détections sur le plan numéro un. Nous avons cette fois-ci constaté que les couleurs orange, violet, verte et bleue posaient des problèmes à notre application.

Cela fait beaucoup de pièce par rapport à nos tests précédents. Nous avons remarqué que ces pièces sont toutes plus petites que celles que l'application arrive facilement à détecter. De plus, la plus grande des pièces, celle qui possède la couleur rose, est celle qui était toujours détectée en premier, suivie des pièces rouge et jaune.

Nous avons fait un quatrième test où nous avons demandé au programme de vérifier 70 pixels. Pour qu'une couleur soit trouvée, ses valeurs RGB doivent être comprises dans un intervalle défini.

Nous sommes parvenus à la même conclusion que lors du test précédent. Ce sont les plus petites pièces qui prennent le plus de temps à être détectée. De plus, en général, l'application met un peu plus de temps à reconnaître toutes les couleurs. Cette fois, nous avons un temps moyen qui dépasse les cinq secondes de manière générale.

Au-delà d'un nombre de pixels à rechercher supérieur à 100 pour une couleur, l'application prend nettement plus de temps à détecter une couleur, en moyenne, entre huit et dix secondes. Nous devons bouger notre téléphone et recadrer la caméra sur le plan à de nombreuses reprises si nous voulons que l'application détecte toutes les couleurs. Les grandes pièces telles que celle qui est coloré en rose, restent celles qui sont détectées le plus rapidement par notre application. Les pièces de plus petite taille mettent, quant à elle, de plus en plus de temps à être détectées par l'application à mesure que nous augmentons le nombre de pixels qui doivent être recherchés afin qu'une couleur soit défini comme ayant été trouvée.

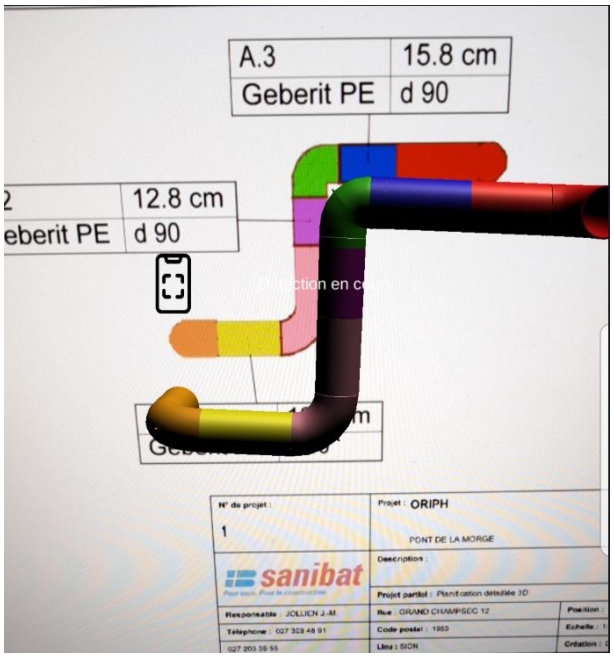


Figure 47 - Détection des couleurs sur le plan numéro un - Image de l'auteure.

La figure 47 représente le résultat de l'application avec l'implémentation de la détection des couleurs pour le plan numéro un.

5.5.2. Plan numéro deux

Le deuxième plan est composé de six pièces, toutes colorées.

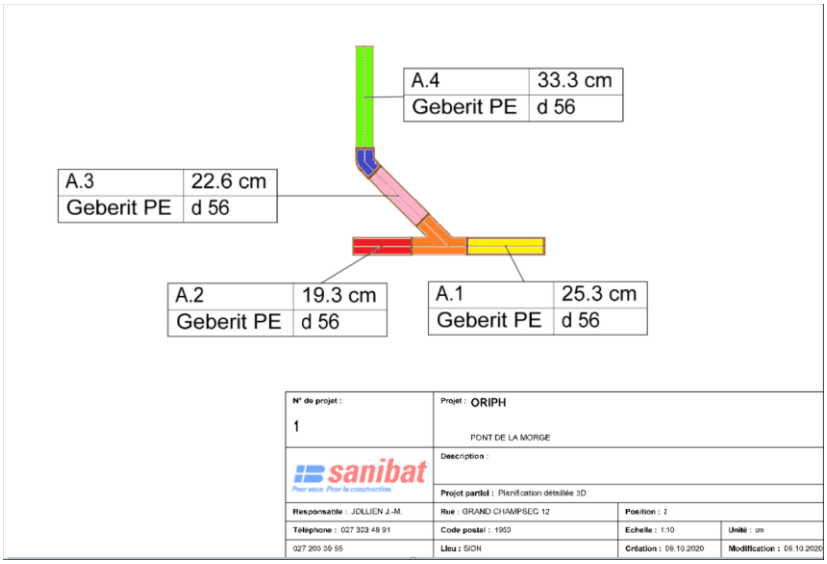


Figure 48 - Plan numéro deux de l'ORIF mis en couleur - Image de l'auteure.

La figure 48 représente un des nombreux plans de tuyauterie de l'ORIF. Celui-ci a été mis en couleurs.

Les valeurs RGB que nous avons modifiées durant le test précédent ont été reportées dans le code qui gère le plan numéro deux. Nous avons fait cela car nous avons estimé que ces corrections ont apporté une meilleure détection des couleurs par l'application.

Nous avons effectué les mêmes testes que sur le plan numéro 1. Nous avons commencé par demander au programme qu'il effectue une recherche de dix pixels pour chaque couleur de ce plan.

Après une vingtaine de tests, nous avons pu observer que l'application trouvait systématiquement toutes les couleurs du plan, en une seconde ou moins. Nous avons observé qu'à trois reprises, la couleur bleue qui se trouve sur la plus petite forme, n'a pas été trouvée aussi rapidement que les autres. 2 secondes en moyenne. Nous avons conclu que ce nombre n'était pas suffisamment significatif pour apporter une correction au code qui gère le deuxième plan.

Nous avons effectué une série d'autre tests en augmentant à chaque fois le nombre de pixels que l'application doit trouver pour une couleur afin que celle-ci soit considérée comme ayant été détectée.

Nous avons observé que l'application ne commençait à prendre un peu plus de temps à détecter les couleurs qu'à partir d'un nombre supérieur à 150 pixels. La moyenne de ce temps est assez similaire au plan numéro une, en six à dix secondes en moyenne. En comparaison, le plan numéro un ne commençait à prendre plus de temps qu'à partir d'un nombre supérieur à 100. La difficulté du calcul de ce temps vient du fait que même avec un taux de recherche élevé, les plus grandes pièces sont trouvées beaucoup plus rapidement que les petites.

Nous avons également pu constater que lorsque qu'une recherche dépasse 150 pixels, la plus petite des pièces, celle qui contient la couleur bleue, était celle qui était très souvent détectée en dernier. Nous avons observé le même phénomène lors de nos tests sur le plan numéro un. L'application semble avoir plus de mal à détecter les petites pièces à partir d'un certain nombre de pixels qu'elle doit trouver pour chaque couleur.

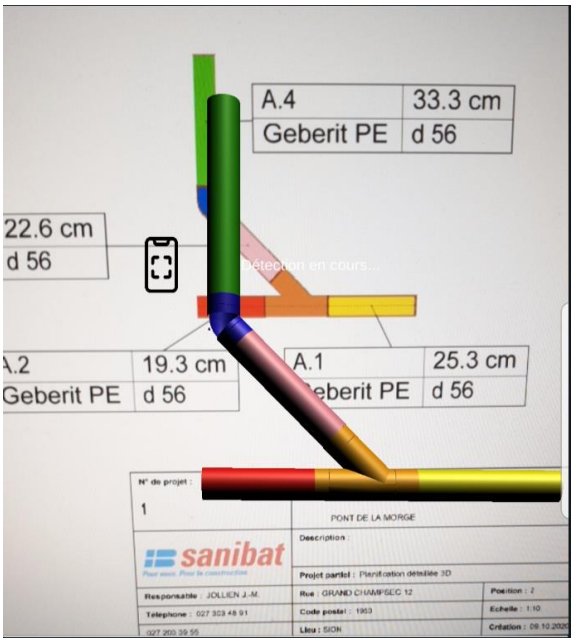


Figure 49 – Détection des couleurs avec le plan numéro deux – Image de l'auteure.

La figure 49 représente l'application avec l'implémentation de la détection des couleurs pour le plan numéro deux.

5.5.3. Plan numéro trois

Le troisième plan est constitué d'onze pièces, toutes colorées, avec la couleur rouge reportée deux fois.

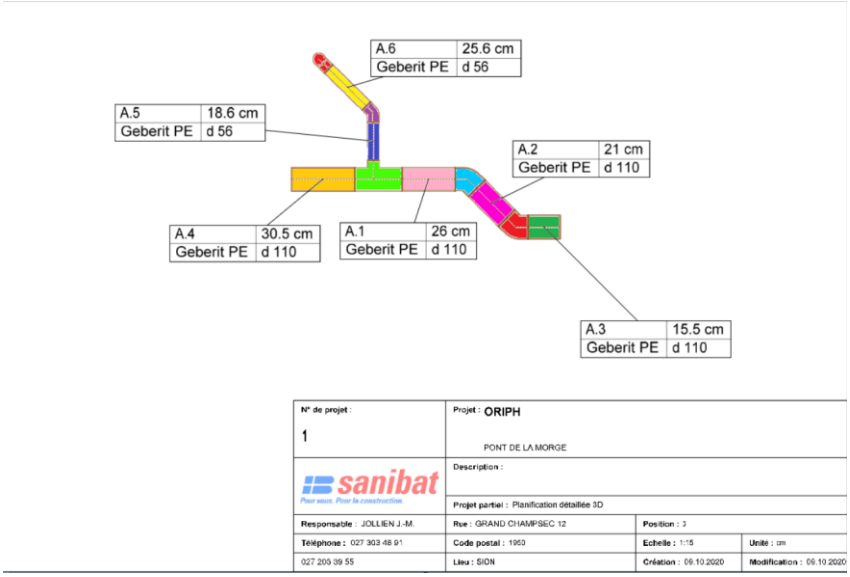


Figure 50 - Plan numéro trois de l'ORIF mis en couleur – Image de l'auteure.

La figure 50 représente un des nombreux plans de tuyauterie de l'ORIF. Celui-ci a été mis en couleurs.

Les valeurs RGB que nous avons modifiées durant le premier test ont été reportées dans le code qui gère le plan numéro trois. Nous avons fait cela car nous avons estimé qu'après ces corrections, l'application détectait mieux les couleurs.

Nous avons effectué les mêmes tests que sur le plan numéro un et deux. Nous avons commencé par demander au programme qu'il effectue une recherche de dix pixels pour chaque couleur que possède le plan.

Durant ce premier test, le programme a été capable de détecter immédiatement toutes les couleurs qui sont affichées sur ce plan. Il lui a fallu en moyenne une seconde, voire moins, pour trouver toutes les couleurs.

Cependant, lorsque nous avons effectué notre deuxième test durant lequel nous avons demandé au programme de trouver 30 pixels d'une même couleur afin que celle-ci soit considérée comme étant été détectée, nous avons remarqué que le programme avait plus de difficultés à trouver la couleur rouge qui se trouve sur la plus petite pièce du plan.

Lorsque nous demandons au programme de trouver 50 pixels pour chaque couleur, nous avons observé que l'application avait cette-fois ci beaucoup de peine à trouver les plus petites pièces du plan qui sont teintes en rouge et en violet. Ces couleurs étaient trouvées trois à cinq secondes après les autres en moyenne. Nous avons également remarqué que le programme commençait à prendre un petit plus de temps à trouver les couleurs du plan de manière général, trois secondes de plus en moyenne. Nous devons davantage bouger le téléphone afin de recadrer le plan pour permettre à l'application de trouver toutes les couleurs.

A partir d'une recherche de 100 pixels, l'application met un certain temps pour trouver de manière générale, toutes les couleurs du plan, huit à 11 secondes en moyenne si nous prenons en compte les plus petites pièces qui prennent plus de temps à être trouvées que les grandes.

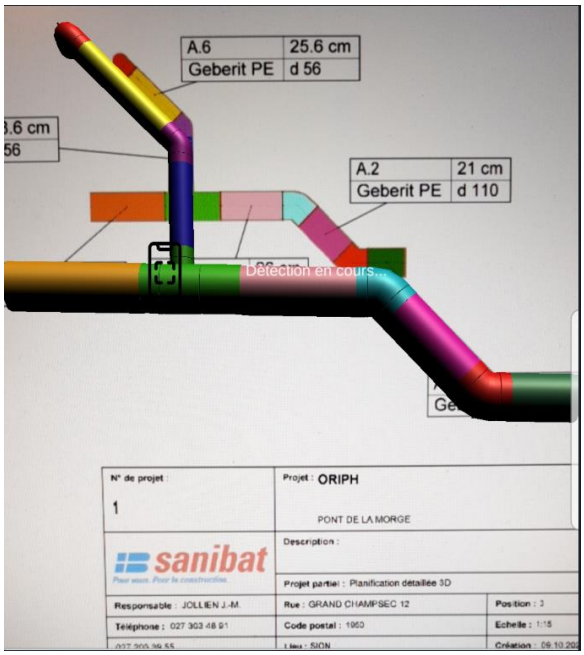


Figure 51 - Détection des couleurs avec le plan numéro trois - Image de l'auteure.

La figure 51 représente l'application avec l'implémentation de la détection des couleurs pour le plan numéro trois.

5.5.4. Synthèse des tests

Après avoir effectué ces tests, nous pouvons conclure que plusieurs critères affectent la détection des couleurs sur un plan. Pour commencer, nous avons le nombre de pièces que possède un plan.

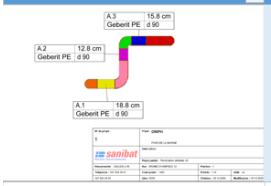
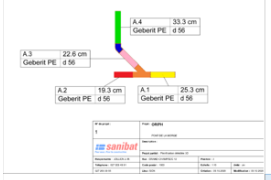
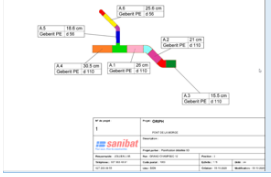
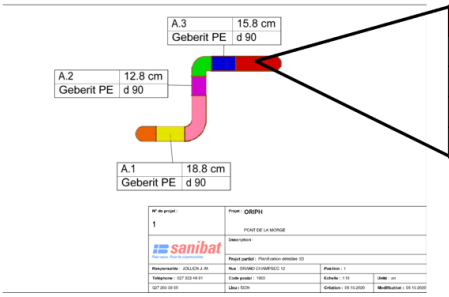
Plan	Nombre de pièces	Nombre de recherches d'un nombre de pixels limite avant que l'application commence à prendre du temp	Temps approximatif (secondes)
	7	100	8 - 10
	6	150	6 - 10
	11	100	8 - 11

Figure 52 - Tableau comparatif des plans - Image de l'auteure.

La figure 52 représente un tableau comparatif des plans qui ont été testés avec leur nombre de pièces et le nombre de recherches de pixels que l'application doit atteindre avant que celle-ci ne commence à prendre du temps à détecter les couleurs.

De plus, nous devons prendre en compte les valeurs RGB maximales et minimales. Ce critère est délicat à déterminer car ce genre de valeur peut varier. Comme nous l'avons mentionné précédemment, deux couleurs rouges qui semblent être les mêmes pour nos yeux humains ne sont pas du tout semblable pour un ordinateur car leur valeurs RGB sont différentes. Nous devons donc faire attention à ce que le code soit adapter pour rechercher les valeurs affichées sur un plan.



Rouge			
	R	G	B
Max	255	80	80
Plan	210	35	35
Min	200	0	0

Figure 53 - Valeur RGB pour la couleur rouge qui aurait pu être trouvée sur un plan de l'ORIF - Image de l'auteure.

La figure 53 représente une valeur RGB pour la couleur rouge qui aurait pu être trouvée sur un plan de l'ORIF. Dans le tableau, nous avons les valeurs exactes du pixel sur la ligne du plan. Celles-ci se trouvent entre les valeurs maximales et minimales qui ont été définies dans le tableau.

Finalement, nous devons tenir compte de la taille des pièces. Puisque que l'algorithme de détection des couleurs se base sur une quantité de pixels adjacents trouvés pour déterminer si une couleur a été décelée, plus une pièce est petite, moins celle-ci contiendra de pixels. Cela réduit la quantité de pixels qui peuvent être recherchée dans une zone.

5.6. Ajouter un nouveau plan

Cette partie est un tutoriel qui va décrire comment ajouter un nouveau plan à l'application. Comme nous l'avons expliqué précédemment, plusieurs facteurs vont influencer la capacité de l'algorithme à détecter une couleur. Ce sont la zone de recherche, les valeurs RGB, la quantité de pièces et enfin, la taille des pièces d'un plan. Cela veut dire que chaque nouveau plan est différent et a besoin d'utiliser des paramètres personnalisés.

Nous allons prendre en compte tous ces facteurs lors de l'ajout d'un plan et regarder quels sont les paramètres à configurer pour un nouveau plan.

5.6.1. Choisir un nouveau plan

La première étape est de choisir un nouveau plan. Pour notre exemple, nous avons pris un des plans de l'ORIF parmi les nombreux que cette organisation possède.

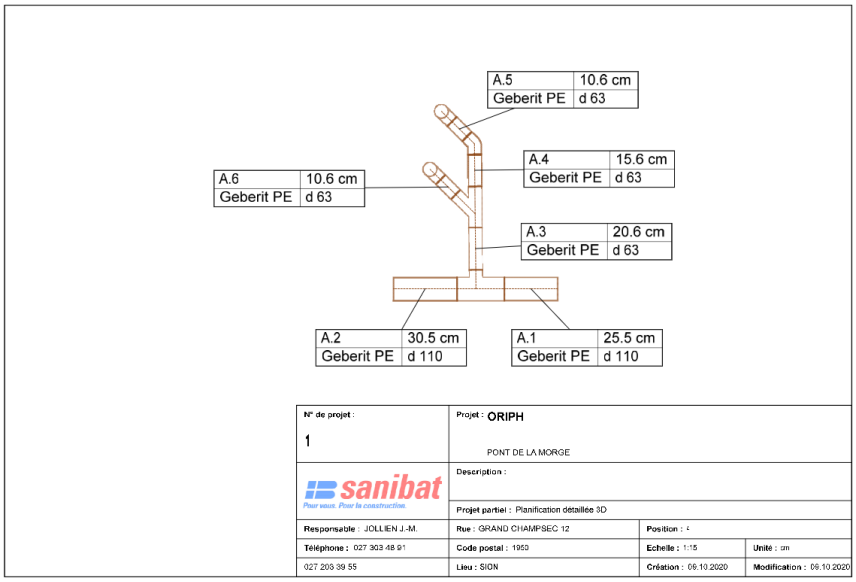


Figure 54 - Plan de l'ORIF - Image de l'auteure.

La figure 54 représente un des nombreux plans de l'ORIF. Lorsque nous l'observons, nous pouvons déjà constater que celle-ci possède un grand nombre de pièces, onze au total, et que ce tuyau est composé à la fois de grandes et de petites pièces. Ces informations sont importantes pour le réglage des paramètres de l'algorithme de détection des couleurs.

Ce plan est déjà présent dans la base de données de Vuforia et dans l'application.

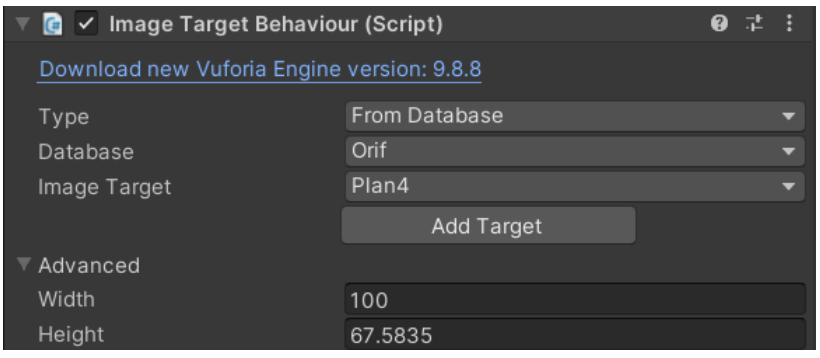


Figure 55 - Capture d'écran de l'inspector de Unity pour le plan n°4 - Image de l'auteure.

La figure 55 représente le script appelé « Image Target Behaviour » dans l'inspecteur de Unity. Nous pouvons voir que le plan numéro quatre est sélectionné dans la base de données de Vuforia.

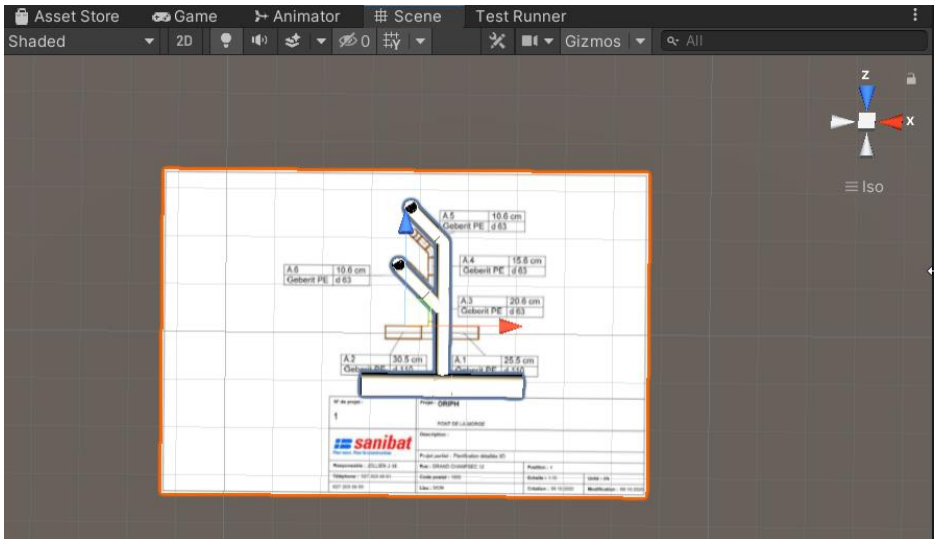


Figure 56 - Scène dans Unity avec le plan numéro quatre - Image de l'auteure.

Sur la figure 56 représente le plan numéro quatre dans la scène de Unity. Nous pouvons y voir sa forme en trois dimensions.

5.6.2. Définir les zones de recherche

La deuxième partie consiste à définir des zones de recherche pour chacune des pièces du plan qui a été choisi. Grâce au programme, nous connaissons la taille de l'image que renvoie la caméra de l'appareil et qui est récupérée par Vuforia. Sa hauteur est de 480 pixels et sa largeur est de 640. Lorsque nous filmons la scène où se trouve le plan, nous savons que nous devons faire en sorte que le plan couvre la totalité de l'écran du téléphone afin que les zones de recherches soient justes.

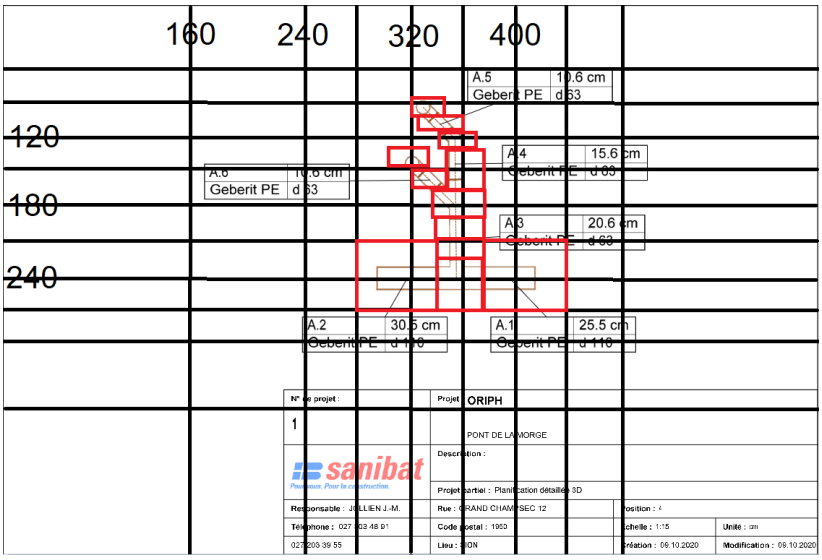


Figure 57 - Capture d'écran du plan n°4 de l'ORIF qui a été divisé - Image de l'auteure.

Sur la figure 57, nous pouvons voir les différentes divisions que nous avons faites sur le plan pris en exemple ainsi que les zones en rouge qui nous intéressent. Ces zones sont toujours tracées en carré ou en rectangle car l'algorithme de détection des couleurs fait une recherche sur une matrice qui possède ces formes.

Si nous observons bien les zones qui délimitent les pièces du plan, nous pouvons voir qu'il y a des détails auxquels nous devons faire attention. En effet, certaines pièces, à cause de leur forme, ne permettent pas à ces zones de les délimiter entièrement. Des bouts de ces pièces se retrouvent en dehors de leur zone principale tandis que d'autres se trouvent dans une zone adjacente. A cause de cela, nous devons faire attention à plusieurs choses. Tout d'abord, il est important d'éviter de donner des couleurs similaires aux pièces qui se touchent et dont les zones inclusent d'autre pièces. Ensuite, nous devons regarder combien de pixels nous voulons que le programme recherche dans une zone pour qu'une couleur soit définie par l'algorithme de détection des couleurs comme étant été détectée. Le plan possède des zones de différentes tailles, ce qui veut dire que la valeur de recherche doit être différente pour chaque zone du plan.

5.6.3. Définir les couleurs

La troisième partie est la définition des couleurs sur le plan. Comme nous l'avons mentionné précédemment, nous devons faire attention aux pièces dont certains bouts se retrouve dans des zones adjacentes.

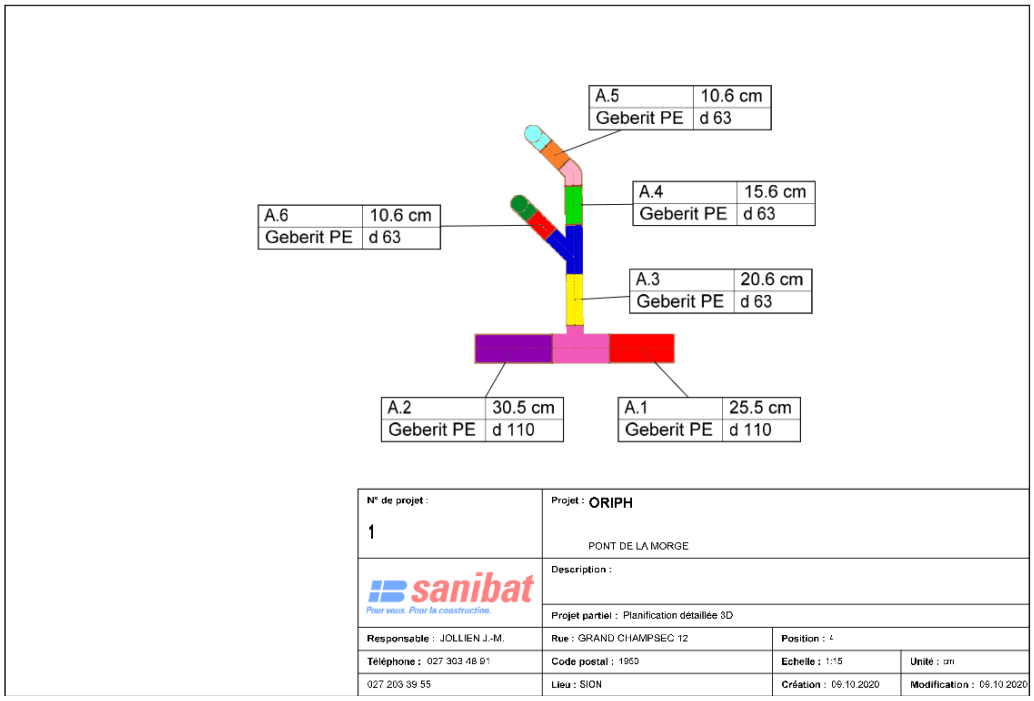


Figure 58 - Plan n°4 de l'ORIF mis en couleurs - Image de l'auteure.

La figure 58 représente le plan numéro quatre mis en couleurs.

Par la suite, nous devons créer les matériaux de couleurs dans Unity afin de pouvoir mettre en couleur la forme en trois dimensions qui est associée au plan pris en exemple. Une série de couleurs se trouve déjà dans Unity à l'intérieur du dossier appelé « Ressources ».

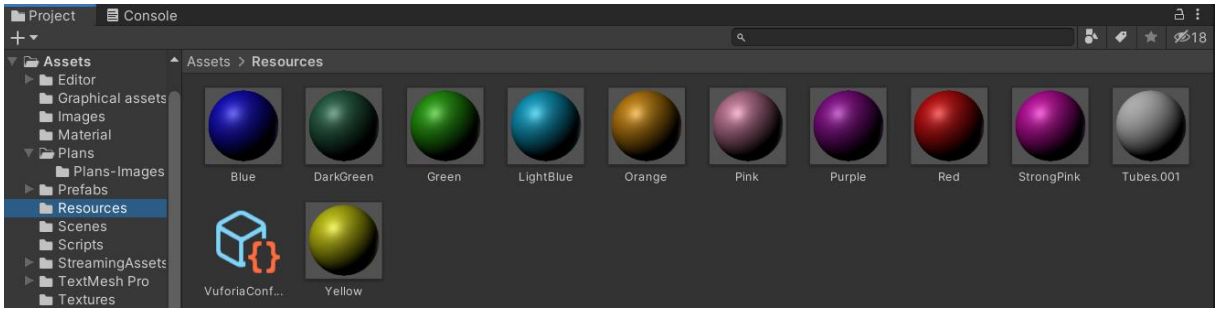


Figure 59 - Contenu du dossier « Resources » - Image de l'auteure.

La figure 59 représente les matériaux de couleur qui sont contenus dans le dossier appelé « Resources ».

Si nous voulons ajouter une nouvelle couleur dans l'application, nous devons faire un clic droit à l'intérieur du dossier « Ressources », puis cliquer sur « Create » et « Material » (figure 60).

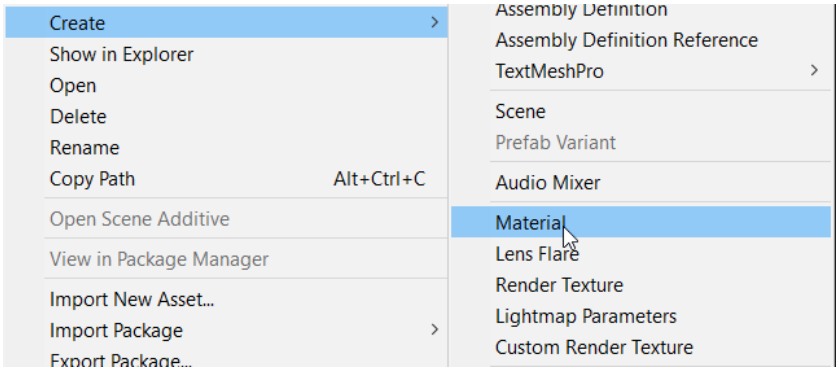


Figure 60 - Chemin pour créer un objet "Material" - Image de l'auteure.

Il faudra ensuite ajouter une couleur à ce nouveau matériel.

Figure 61 - Choix des couleurs pour l'objet "Material" nouvellement créé - Image de l'auteure.

La figure 61 représente le choix des couleurs pour un matériel dans Unity.

La mise en couleurs d'une forme en trois dimensions se fait au moyen d'un code. Nous devons tout d'abord aller dans le script appelé « ShapeManager ». Nous devons créer l'objet qui sera notre nouvelle couleur.

```
private Material blackMaterial;
```

Figure 62 - Objet de type « Material » - Image de l'auteure.

La figure 62 représente un objet de type « Material » en C# dans Unity.

Ensuite, nous devons appeler cet objet dans la méthode appelée « Start ».

```
blackMaterial = (Material)Resources.Load("Black", typeof(Material));
```

Figure 63 - Méthode pour l'appel du matériel de couleur noire - Image de l'auteure.

Sur la figure 63, nous pouvons voir que le code va chercher le nouveau matériel via un chemin. Celui-ci est un dossier appelé « Ressources ». Il est important de toujours placer les matériaux créés dans un dossier qui possède ce nom si nous voulons que le code puisse les trouver. Il est aussi important de mettre le nom du nouveau matériel créé en paramètre.

Ceci fait, nous devons créer les méthodes pour changer la couleur d'une pièce avec la nouvelle couleur noire que nous avons créée. La première méthode va récupérer une pièce, peu importe laquelle pour l'instant, et changer sa couleur à l'aide du matériel créé précédemment.

```
public void ChangeBlack(GameObject piece)
{
    piece.GetComponent<Renderer>().material = blackMaterial;

}
```

Figure 64 - Méthode pour changer la couleur d'une pièce - Image de l'auteure.

La figure 64 représente la méthode qui sert à changer la couleur d'une pièce en trois dimensions.

Enfin, une méthode qui va récupérer la bonne pièce sur un plan doit être créée, elle va se servir de la méthode précédente pour changer sa couleur.

```
public void ManageShapeBlack(int nbChild)
{
    //For yellow
    piece = childPlan.transform.GetChild(nbChild).gameObject;
    ChangeBlack(piece);
}
```

Figure 65 - Méthode pour changer la couleur d'une pièce spécifique - Image de l'auteure.

La figure 65 représente la méthode qui permet de changer la couleur d'une pièce spécifique en trois dimensions. Celle-ci se sert de la méthode précédente.

5.6.4. Création du script pour gérer un plan

La quatrième consiste à créer le code qui va gérer la détection des couleurs et la colorisation des pièces en trois dimensions pour le plan pris en exemple.

Pour commencer, nous allons créer un script et l'assigner au plan. Pour créer un script, nous allons dans le dossier appelé « Scripts », faire un clic droit, aller sur « Create » et « C# Script » (figure 66).

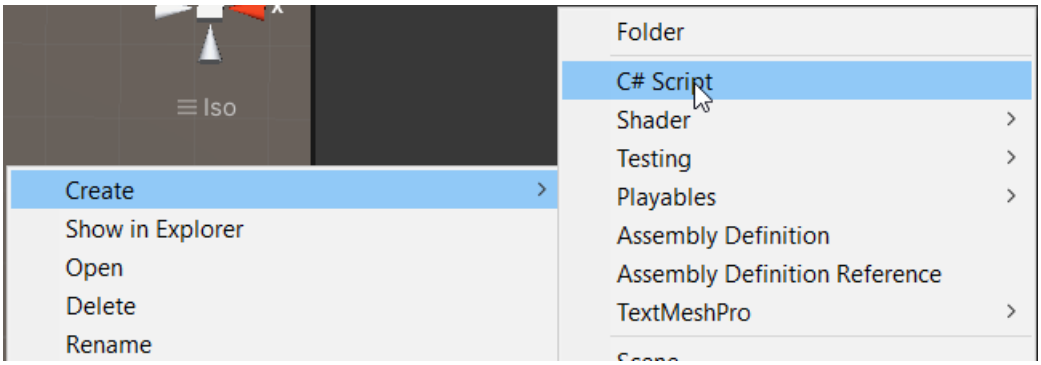


Figure 66 - Création d'un script dans Unity - Image de l'auteure.

Pour assigner un script à un plan, nous devons cliquer sur le plan en question qui se trouve dans la scène nommée « ImageTrackingScene ».

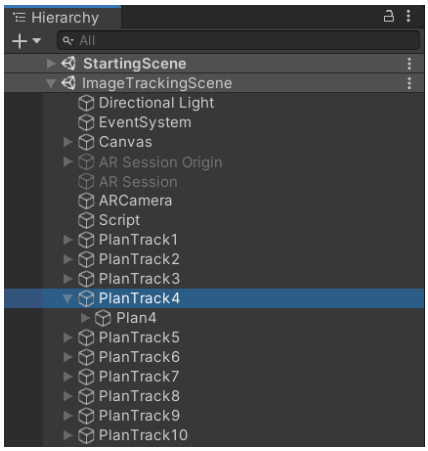


Figure 67 - Scène "ImageTrackingScene" - Image de l'auteure.

La figure 67 représente la hiérarchie de la scène nommée « ImageTrackingScene ».

Puis, à droite dans l'inspecteur, nous devons cliquer sur « Add Component » (figure 68) et sélectionner le script (figure 69).

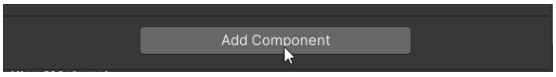


Figure 68 - Bouton « Add Component » - Image de l'auteure.

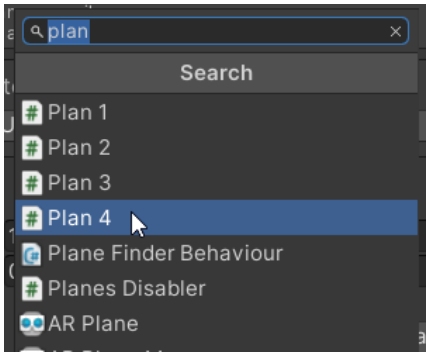


Figure 69 - Capture d'écran de l'ajout du script "Plan 4" à notre plan n°4 dans Unity - Image de l'auteure.

5.6.5. Création du code pour gérer un plan

La cinquième étape est d'ajouter du code à l'intérieur du nouveau script afin de pouvoir gérer le plan. Il est important de noter que le code est le même pour chaque plan et que seuls certains paramètres peuvent changer comme les valeurs RGB, la zone de recherche et la quantité de couleurs avec leur méthode. Il faut aussi savoir que chaque couleur possède sa propre méthode pour être détectée. Les changements faits dans le script propre au plan sont principalement faits dans ces méthodes.

Nous allons commencer par définir les valeurs RGB maximales et minimales que nous voulons rechercher.

```
//RED
Color higher_red = new Color(255, 80, 80);
Color lower_red = new Color(200, 0, 0);

//GREEN
Color higher_green = new Color(100, 255, 100);
Color lower_green = new Color(0, 200, 0);

//BLUE
Color higher_blue = new Color(100, 100, 255);
Color lower_blue = new Color(0, 0, 200);
```

Figure 70 - Capture d'écran d'une partie des valeur RGB pour la détection des couleurs - Image de l'auteure.

La figure 70 montre les valeurs RGB pour les couleurs rouge, vert et bleu pour le plan numéro quatre que nous avons pris en exemple. Ces valeurs peuvent varier selon la teinte de la couleur que nous voulons détecter.

Ensuite, nous devons adapter les limites de chaque zone pour chacune de ces couleurs. Ces limites sont définies par des boucles.

```
for (int i = 320; i < 360; i++)
{
    for (int j = 150; j < 170; j++)
    {
```

Figure 71 - Capture d'écran des boucles qui définissent la zone dans laquelle chercher une couleur - Image de l'auteure.

La figure 71 représente les boucles qui définissent la zone dans laquelle l'algorithme doit chercher une couleur.

Nous devons également changer le numéro de la pièce que nous voulons colorer dans la méthode prévue à cet effet.

```
shapeManager.ManageShapeRedPlan(5);
```

Figure 72 - Méthode pour donner la couleur rouge à une pièce - Image de l'auteure.

La figure 72 représente la méthode qui donne la couleur rouge à une pièce. Le nombre cinq qui est passé en paramètre indique le numéro de la pièce qui doit prendre la couleur rouge.

Le script possède également une méthode qui permet d'effacer les couleurs du modèle en trois dimensions. Nous devons donner à cette méthode le nombre de pièces que possède un plan.

```
public void planIsLost()
{
    isDetected = false;
    shapeManager.RemoveAllMaterialPlan(10);
}
```

Figure 73 - Méthode pour effacer les couleurs d'un modèle en trois dimensions - Image de l'auteure.

Sur la figure 73 nous avons la méthode appelée « RemoveAllMaterialPlan » qui prend en paramètre le nombre 10 qui correspond au nombre de pièces moins une du plan numéro quatre qui a été pris en exemple.

5.6.6. Autres modifications

La sixième étape consiste à faire les changements finaux pour faire fonctionner la détection de couleur sur un nouveau plan. Les scripts nécessaires à ces changements sont déjà implémentées dans le projet.

Tout d'abord, dans chaque script qui gère les plans, il existe une méthode appelée « planIsDetected » qui va permettre de lancer la détection des couleurs et une méthode appelée « planIsLost » qui permet de l'arrêter.

```
public void planIsDetected()
{
    isDetected = true;
}

0 références
public void planIsLost()
{
    isDetected = false;
    shapeManager.RemoveAllMaterialPlan(10);
}
```

Figure 74 - Capture d'écran des méthodes qui permettent de lancer et d'arrêter la détection des couleurs - Image de l'auteure.

La figure 74 représente les méthodes de lancement et d'arrêt de l'algorithme de détection des couleurs pour le plan numéro quatre que nous avons pris en exemple.

Ces méthodes ont été créées pour des raisons de performance. La détection des couleurs est lancée pour un plan spécifique au moment où celui-ci a été détecté. Pour utiliser ces deux méthodes, le programme utilise un script appelé « DefaultTrackableEventHandler » qui est déjà implémenté par Vuforia. A l'intérieur de ce script, il y a deux méthodes qui permettent de savoir si un plan a été trouvé ou non par l'application. Nous devons cliquer tout d'abord sur le plan que nous voulons sélectionner puis dans l'inspecteur, et finalement nous devons nous rendre sur le script en question.

Figure 75 - Script "Default Trackable Event Handler" - Image de l'auteure.

La figure 75 représente le script appelé "Default Trackable Event Handler" dans l'éditeur de Unity.

En restant dans l'inspecteur de Unity sur le plan que nous avons choisi, nous pouvons voir qu'il y a deux méthodes à l'intérieur de ce script. La première est appelée « On Target Found ». Elle sert à définir si le plan en question a été trouvé ou non. La deuxième s'appelle « On Target Lost » qui sert à définir si un plan n'est pas ou plus détecté. Sur la méthode appelée « On Target Found », nous devons cliquer sur l'icône plus. Cela va ajouter un champ vide.

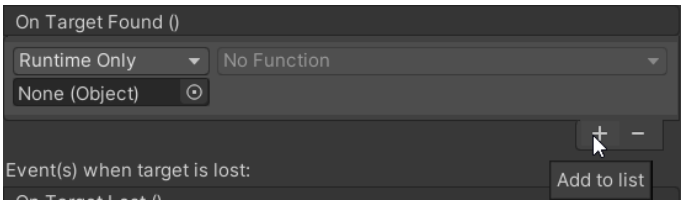


Figure 76 - Méthode "On Target Found" - Image de l'auteure.

La figure 76 représente l'endroit où la méthode nommé « planIsDetected » doit être ajouté.

A l'intérieur du champ appelé « Objet », nous devons ajouter le plan numéro quatre car la méthode nécessaire se trouve à l'intérieur du script, lui-même compris dans le plan numéro quatre.

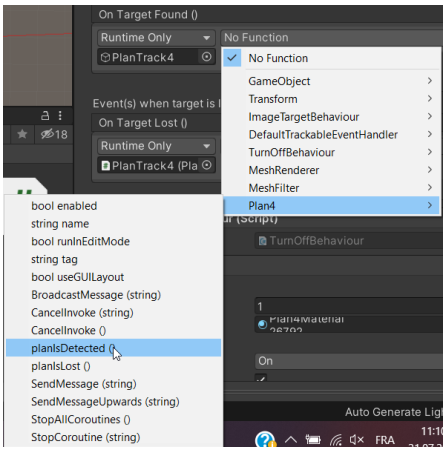


Figure 77 - Chemin pour appeler la méthode "planIsDetected" - Image de l'auteure.

La figure 77 représente le chemin pour l'ajout de la méthode nommée « planIsDetected ».

Nous devons faire la même chose pour la méthode appelée « planIsLost » lorsque notre application ne détecte plus le plan.

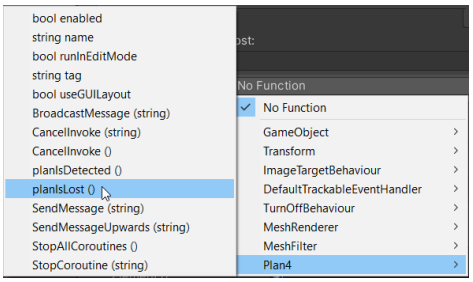


Figure 78 - Capture d'écran de l'ajout de la méthode appelé "planIsLost" - Image de l'auteure.

La figure 78 représente le chemin pour ajouter la méthode nommée « planIsLost ».

Nous devons également ajouter une méthode d'appel de plan lorsque celui-ci est détecté. Cette méthode va permettre à l'application de savoir quelle forme en trois dimensions doit être mise en couleur. Tout d'abord, nous devons aller dans le script appelé « ShapeManager ». Nous devons créer l'objet qui sera associé au plan numéro quatre que nous avons pris en exemple.

```
GameObject plan1;
GameObject plan2;
GameObject plan3;
GameObject plan4;
```

Figure 79 - Objet "plan4" - Image de l'auteure.

La figure 79 représente l'objet appelé « plan4 » en langage C#. Cet objet est de type « GameObject ».

Nous devons ensuite appeler cet objet à l'intérieur de la méthode nommée « Start ».

```
plan1 = GameObject.Find("PlanTrack1");
plan2 = GameObject.Find("PlanTrack2");
plan3 = GameObject.Find("PlanTrack3");
plan4 = GameObject.Find("PlanTrack4");
```

Figure 80 - Appel du plan n°4 - Image de l'auteure.

La figure 80 représente les différents plans qui ont été appelé dans la méthode « Start ». L'appel du plan numéro quatre est mis en évidence.

Pour finir, nous devons définir la méthode qui va appeler le plan numéro quatre (figure 81).

```
public void CallPlan4()
{
    plan = plan4;
    childPlan = plan.transform.GetChild(0).gameObject;

}
```

Figure 81 - Méthode qui appelle le plan n°4 - Image de l'auteure.

Comme nous pouvons le voir sur la figure 80, les objets sont les mêmes pour tous les plans, seule leur dénomination change. Il en va de même pour la méthode d'appel d'un plan. Il y en a une par plan. Afin de l'appeler, nous devons retourner dans l'éditeur du plan numéro quatre dans Unity, à l'intérieur du script appelé « DefaultTrackableEventHandler ». Cette méthode doit être lancée lorsque le plan numéro quatre a été trouvé.

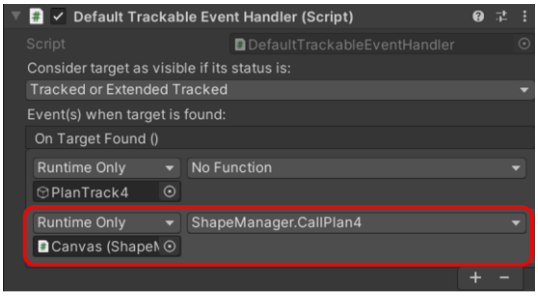


Figure 82 - Ajout de la méthode "CallPlan4" - Image de l'auteure.

La figure 82 représente la méthode nommée « CallPlan4 » qui a été ajouté à la méthode appelée « OntTargetFound ».

Enfin, nous devons appeler les méthodes de détection des couleurs. Pour cela, nous devons aller dans le script appelé « CameraImageAccess ». Tout d'abord, nous devons créer un objet qui va être le script de notre plan numéro quatre.

```
Plan1 plan1;
Plan2 plan2;
Plan3 plan3;
Plan4 plan4;
```

Figure 83 - Déclaration du script pour le plan numéro quatre - Image de l'auteure.

La figure 83 représente la déclaration du script de chaque plan qui sont utilisés dans l'application. Le plan numéro quatre est mis en valeur.

Nous devons ensuite appeler ce script dans la méthode appelée « Start ».

```
void Start()
{
    //Call the different plan scripts
    plan1 = GameObject.Find("PlanTrack1").GetComponent<Plan1>();
    plan2 = GameObject.Find("PlanTrack2").GetComponent<Plan2>();
    plan3 = GameObject.Find("PlanTrack3").GetComponent<Plan3>();
    plan4 = GameObject.Find("PlanTrack4").GetComponent<Plan4>();
```

Figure 84 - Appel du script pour le plan numéro quatre - Image de l'auteure.

La figure 84 représente les appels de différents scripts. Le script pour le plan numéro quatre est mis en évidence.

Finalement, nous devons appeler les différentes méthodes de détection des couleurs du plan numéro quatre dans la méthode appelée « OnTrackablesUpdated » avec les bons paramètres.

```
plan4.FindRed1ColorPositionForPlan4(image, imageMat);
plan4.FindBlueColorPositionForPlan4(image, imageMat);
plan4.FindGreenColorPositionForPlan4(image, imageMat);
plan4.FindDarkGreenColorPositionForPlan4(image, imageMat);
plan4.FindLightBlueColorPositionForPlan4(image, imageMat);
plan4.FindPurpleColorPositionForPlan4(image, imageMat);
plan4.FindOrangeColorPositionForPlan4(image, imageMat);
plan4.FindPinkColorPositionForPlan4(image, imageMat);
plan4.FindRed2ColorPositionForPlan4(image, imageMat);
plan4.FindStrongPinkColorPositionForPlan4(image, imageMat);
plan4.FindYellowColorPositionForPlan4(image, imageMat);
```

Figure 85 - Appel des méthodes pour le plan numéro 4 - Image de l'auteure.

La figure 85 représente l'appel des méthodes de détection des couleurs pour le plan numéro quatre.

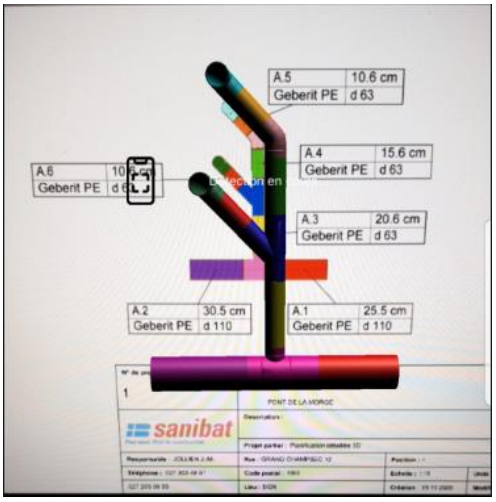


Figure 86 - Résultat de l'ajout d'un nouveau plan de l'ORIF dans l'application - Image de l'auteure.

La figure 86 représente le résultat final de la partie « Ajout d'un nouveau plan ».

6. Discussion

Dans cette partie, nous allons regarder les améliorations qui pourraient être apportées à ce projet dans le futur.

6.1. Améliorations possibles

6.1.1. Qualité du code

Tout d'abord, nous pourrions améliorer le code. Nous utilisons beaucoup de méthodes similaires et de répétitions à l'intérieur du code, particulièrement au niveau des scripts qui sont propres à chaque plan. Nous pourrions prendre du temps pour restructurer le code de manière à réduire la quantité de morceaux de code qui se répète.

6.1.2. Algorithme de détection des couleurs

Nous pourrions également améliorer l'algorithme de détection des couleurs. Celui-ci contient une grande quantité de boucles, ce qui rend le programme peu optimisé.

Cet algorithme ne peut pas fonctionner correctement sans qu'un plan ne recouvre tout l'espace de la caméra. Dans le futur, ce problème pourrait être corrigé. Nous pourrions faire en sorte que les couleurs soient détectables par l'application, peu importe où se trouve un plan dans une scène lorsque celle-ci est filmée.

Nous aurions pu tester plus efficacement l'algorithme de détection des couleurs. Il aurait été possible de prendre une image de notre scène filmée par l'appareil et lancer l'algorithme de détection des couleurs sur cette image, par exemple pour la couleur bleue. Ensuite, un masque aurait pu être créé avec les nuances de bleu déterminées par l'algorithme de détection des couleurs et aurait pu être appliqué sur l'image de la scène. De cette façon, nous aurions pu voir plus précisément quelles sont les sortes de nuances de bleu qui ont été détectées sur l'image de la scène.

Figure 87 - Application d'un masque fait en langage Python avec OpenCV pour la couleur bleue, - Tutoriel Python avec OpenCV, https://www.etutorialspoint.com/index.php/329-detect-specific-color-from-image-using-python-opencv.

La figure 87 représente une image de fleurs sur laquelle un masque a été appliqué. La couleur bleue est la seule qui a été gardé.

Lors des recherches de technologies qui permettent de faire de la détection de couleurs nous avons trouvé que la librairie OpenCV possédait des modules capables de le faire. Cependant, nous avons trouvé beaucoup de documentation pour utiliser OpenCV avec le langage Python et très peu avec le langage C# sur Unity. Nous aurions pu tenter de créer notre propre module de détection des couleurs en langage Python et essayer de l'appeler dans Unity. Lorsque Vuforia détecte une image, nous aurions pu appeler notre module qui aurait simplement pris l'image de la caméra. Ce module aurait effectué une détection des couleurs, par exemple le rouge, en dehors de l'application. Ensuite, il aurait rendu l'image de la scène avec un masque sur lequel seules les nuances de rouge auraient été affichées. Nous aurions donc pu dire au programme principal de travailler avec cette nouvelle image dont la couleur rouge avait été extraite.

7. Conclusion

En conclusion, nous avons effectué des recherches sur la réalité augmentée ainsi que sur les technologies qui sont en lien avec celle-ci. Nous avons analysé ces technologies afin de savoir lesquelles nous seraient utiles pour le développement des fonctionnalités de détection des couleurs et de mise en couleur de formes en trois dimensions.

Grâce à nos recherches, nous avons pu développer ces deux fonctionnalités et nous les avons ajoutées à l'application.

Nous avons commencé par réfléchir à une solution pour la création d'un algorithme de détection des couleurs. Pour cela, nous nous sommes intéressés à la façon de récupérer les images d'une caméra qui filme une scène avec Vuforia. Ensuite, nous avons cherché comment effectuer une récupération des données d'un pixel sur une image. Nous avons finalement décidé de créer un algorithme qui cherche une certaine quantité de pixels comprise entre deux valeur RGB dans une zone définie.

Les tests que nous avons effectués avec notre application nous ont montré que plusieurs facteurs pouvaient influencer l'algorithme de détection des couleurs. Ils nous ont aussi montré qu'il était plus efficace de créer des scripts pour chaque plan avec des paramètres propres à chaque plan.

Finalement, nous avons réfléchis aux améliorations possibles pour notre code dans le futur.

RÉFÉRENCES

Arnaldi, B., Guitton, P., & Moreau, G. (2018). *Réalité virtuelle et réalité augmentée*. Londre: ISTE Editions Ltd. Consulté le Juillet 06, 2021, sur https://books.google.ch/books?hl=fr&lr=&id=u6xqDwAAQBAJ&oi=fnd&pg=PP1&dq=R%C3%A9alit%C3%A9+virtuelle+et+r%C3%A9alit%C3%A9+augment%C3%A9e&ots=I84ziLuupd&sig=YdtFcvmzFlS-ib3CfTzQhIvH-Bw&redir_esc=y#v=onepage&q=R%C3%A9alit%C3%A9%20virtuelle%20et%20r%C3%A9ali

Cardinale, A. (2017). *Développer des applis innovantes avec unity*. (D.-B. éditions, Éd.) Consulté le Juillet 6, 2021, sur http://www.scholarvox.com/catalog/book/88840592#

DORLAC, S. (2014). *La réalité augmentée avec Unity Guide du développeur C#*. Eni Editions. Consulté le Juillet 02, 2021, sur http://www.eni-training.com/portal/client/mediabook/home

Emgu Corporation. (2008a, Février). *Main Page*. Consulté le Mai 17, 2021, sur EMGU: https://www.emgu.com/wiki/index.php/Main_Page

Emgu Corporation. (2008b). *Working with Vuforia*. Consulté le Mai 14, 2021, sur EMGU: https://www.emgu.com/wiki/index.php/Working_with_Vuforia

Epic Game. (2004). *Features*. Consulté le Juillet 02, 2021, sur unrealengine: https://www.unrealengine.com/en-US/features

IKEA. (1999). *Applis IKEA*. Consulté le Juillet 6, 2021, sur ikea: https://www.ikea.com/ch/fr/customer-service/mobile-apps/

Intel. (2000a, Juin). *cv::Mat Class Reference*. Consulté le Mai 14, 2021, sur OpenCV: https://docs.opencv.org/4.5.2/d3/d63/classcv_1_1Mat.html

Intel. (2000b, Juin). *Introduction*. Consulté le Mai 11, 2021, sur OpenCV: https://docs.opencv.org/master/d1/dfb/intro.html

Intel. (2000c). *Thresholding Operations using inRange*. Consulté le Juin 12, 2021, sur OpenCV: https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html

Jesse, G. (2018). *Unity 2018 Augmented Reality Projects: Build four immersive and fun AR applications using ARKit, ARCore, and Vuforia*. Consulté le Juillet 14, 2021, sur https://books.google.ch/books?hl=fr&lr=&id=aO1mDwAAQBAJ&oi=fnd&pg=PP1&dq=vuforia+advantages&ots=c5pVdh6SSX&sig=DXExmlVmnKdANXwsHh0C9fd02h4&redir_esc=y#v=onepage&q=vuforia%20advantages&f=false

Linowes, J., & Babilinski, K. (2017). *Augmented Reality for Developpers*. Consulté le Juillet 15, 2021, sur http://univ.scholarvox.com/catalog/book/docid/88855189?searchterm=unity%20ar

Microsoft. (1975, Avril 4). *A tour of the C# language*. Consulté le Juillet 2, 2021, sur microsoft: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

Microsoft. (1975, Février 4). *Multidimensional Arrays (C# Programming Guide)*. Récupéré sur microsoft: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/multidimensional-arrays

Nguyen, N. (2020, Juin 6). *Developping a multilayer AR game using AR Foundation and Unity*. Consulté le Juillet 7, 2021, sur Theseus: https://www.theseus.fi/handle/10024/355400

PTC Inc. (2011a). *Best Practices for Designing and Developing Image-Based Targets*. Consulté le Mai 11, 2021, sur Vuforia: https://library.vuforia.com/features/images/image-targets/best-practices-for-designing-and-developing-image-based-targets.html

PTC Inc. (2011b). *Getting Started*. Consulté le Mai 11, 2021, sur Vuforia: https://library.vuforia.com/

PTC Inc. (2011c). *Getting Started with Vuforia Engine in Unity*. Consulté le Mai 11, 2021, sur Vuforia: https://library.vuforia.com/articles/Training/getting-started-with-vuforia-in-unity.html

PTC Inc. (2011d). *Vuforia Engine and AR Foundation*. Consulté le 2021, sur Vuforia: https://library.vuforia.com/articles/Solution/ar-foundation.html

PTC Inc. (2011e). *Working with the Camera*. Consulté le Mai 11, 2021, sur Vuforia: https://library.vuforia.com/articles/Solution/Working-with-the-Camera.html

Unity Technologies. (2004b, Août 2). *La plateforme leader pour la création de contenu interactif en temps réel*. Consulté le Juillet 2, 2021, sur unity: https://unity.com/fr

Unity Technologies. (2018). *Material*. Consulté le Juillet 14, 2021, sur Unity: https://docs.unity3d.com/ScriptReference/Material.html

Unity Technologies. (2018a). *About AR Foundation*. Consulté le Mai 10, 2021, sur Unity: https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/manual/index.html

Unity Technologies. (2018c). *Renderer*. Consulté le Juillet 14, 2021, sur Unity: https://docs.unity3d.com/ScriptReference/Renderer.html

Unity Technologies. (2020d). *Vector2*. Consulté le Juillet 12, 2021, sur Unity: https://docs.unity3d.com/ScriptReference/Vector2.html

Unity Technologies. (2020e). *Vector3*. Consulté le Juillet 12, 2021, sur Unity: https://docs.unity3d.com/ScriptReference/Vector3.html

ANNEXE I : Journal de bord

mai.21	lundi, 10 mai 2021	08:00:00	•Lires documents •Suivre tuto Unity AR Fundation •Organisation générale •Lecture code du TB précédent
	mardi, 11 mai 2021	08:00:00	•Tuto capture et image tracking •Application test pour changement de couleur d'un model en 3D •Compréhension et utilisation de la librairie Vuforia
	mercredi, 12 mai 2021		
	jeudi, 13 mai 2021	05:00:00	•Tuto reconnaissance de couleurs
	vendredi, 14 mai 2021	05:00:00	•Documentation OpenCV
	samedi, 15 mai 2021		
	dimanche, 16 mai 2021		
	lundi, 17 mai 2021	06:00:00	•Tutos OpenCV EmguCV
	mardi, 18 mai 2021	06:00:00	•Début de l'organisation du rapport •Comprendre la class Mat dans OpenCV/EmguCV
	mercredi, 19 mai 2021		
	jeudi, 20 mai 2021	03:00:00	•Résolution de problèmes
	vendredi, 21 mai 2021		
	samedi, 22 mai 2021		
	dimanche, 23 mai 2021		
	lundi, 24 mai 2021	08:00:00	•Tuto OpenCV for Unity •Gérer la compatibilité avec Vuforia •Comprendre la récupération des pixels dans une image
	mardi, 25 mai 2021	07:00:00	•Accéder aux pixel d'une image récupérée avec Vuforia •Passer une image dans la class Mat de OpenCV for Unity
	mercredi, 26 mai 2021		
	jeudi, 27 mai 2021		
	vendredi, 28 mai 2021		
	samedi, 29 mai 2021		
	dimanche, 30 mai 2021		
	lundi, 31 mai 2021	08:00:00	•Remettre notes en ordre pour la reconnaissance de couleurs •Analyse de l'application afin de trouver comment faire le lien entre la forme sur le plan et la forme en 3d •Trouver une solution pour lié une forme sur un plan et

juin.21	mardi, 1 juin 2021	04:00:00	•Analyse de la faisabilité des solutions trouvé pour la reconnaissance des formes sur un plan en 2D et la liaison avec une forme en 3D
	mercredi, 2 juin 2021		
	jeudi, 3 juin 2021		
	vendredi, 4 juin 2021		
	samedi, 5 juin 2021		
	dimanche, 6 juin 2021		
	lundi, 7 juin 2021	08:00:00	•Détection des différent plan coloré par Vuforia •Vérification si Vuforia peine à détecter des formes très similaires même si elles ont des couleurs différents •Implémentation de la solution orienté utilisateur
	mardi, 8 juin 2021	07:00:00	•Implémentation de la solution orienté utilisateur •Rechercher d'un moyen de récupérer l'objet en 3D affiché sur l'écran selon le plan scanné
	mercredi, 9 juin 2021	07:00:00	•State of Art
	jeudi, 10 juin 2021		
	vendredi, 11 juin 2021	02:00:00	•Récupérer la position de l'image target
	samedi, 12 juin 2021		
	dimanche, 13 juin 2021		
	lundi, 14 juin 2021	07:00:00	•Implémentation de la solution Vuforia
	mardi, 15 juin 2021	06:00:00	•Créer une matrice de couleurs •Récupérer la position de la couleur trouvé dans la matrice •Définir un range de pixels pour les couleurs
	mercredi, 16 juin 2021	06:00:00	•Avancement du rapport •Recherche de documentation officiel
	jeudi, 17 juin 2021	02:00:00	•Avancement du rapport
	vendredi, 18 juin 2021	02:00:00	•Avancement du rapport
	samedi, 19 juin 2021		
	dimanche, 20 juin 2021		
	lundi, 21 juin 2021	02:00:00	•Avancement du rapport
	mardi, 22 juin 2021		
	mercredi, 23 juin 2021	08:00:00	•Récupération du nom de chaque plan scanné grâce à la méthode OnTargetFound •Assigner le nom du plan dans le script de changement de couleurs pour la forme en 3D •Avancement du rapport
	jeudi, 24 juin 2021		
	vendredi, 25 juin 2021		
	samedi, 26 juin 2021		
	dimanche, 27 juin 2021		
	lundi, 28 juin 2021	08:00:00	•Trouver la position de l'image target dans la scène •Trouver le coin suppérieur gauche de l'image target •Insérer l'image target dans un tableau
	mardi, 29 juin 2021	05:00:00	•Segmentation d'un plan
	mercredi, 30 juin 2021	08:00:00	•Compréhension des positions de l'image target par rapport à la scène et à la caméra •Découpage de l'image de la caméra afin de pouvoir faire de la reconnaissance de couleur dans une zone spécifique

juil.21	jeudi, 1 juillet 2021		
	vendredi, 2 juillet 2021	05:00:00	•Avancement du rapport •Documentation
	samedi, 3 juillet 2021	08:00:00	•Définition des zones pour l'algo de détection des couleurs
	dimanche, 4 juillet 2021		
	lundi, 5 juillet 2021		
	mardi, 6 juillet 2021	07:00:00	•Avancement du rapport •Documentation •Réglage de la quantité de pixels d'une seule couleur pour valider la détection d'une couleur •Testes de l'application avec caméra du pc et caméra du smartphone (différence de qualité)
	mercredi, 7 juillet 2021	05:00:00	•Avancement du rapport
	jeudi, 8 juillet 2021	05:00:00	•Avancement du rapport •Documentation •Ajout des méthode sur les formes 2 et 3
	vendredi, 9 juillet 2021	05:00:00	•Ajout des méthode sur les formes 2 et 3 •Avancement du rapport
	samedi, 10 juillet 2021		
	dimanche, 11 juillet 2021		
	lundi, 12 juillet 2021	08:00:00	•Avancement du rapport
	mardi, 13 juillet 2021	05:00:00	•Avancement du rapport
	mercredi, 14 juillet 2021	08:00:00	•Avancement du rapport •Documentation
	jeudi, 15 juillet 2021	08:00:00	•Avancement du rapport •Documentation
	vendredi, 16 juillet 2021	08:00:00	•Avancement du rapport
	samedi, 17 juillet 2021		
	dimanche, 18 juillet 2021		
	lundi, 19 juillet 2021	08:00:00	•Avancement du rapport
	mardi, 20 juillet 2021	08:00:00	•Avancement du rapport
	mercredi, 21 juillet 2021	08:00:00	•Avancement du rapport
	jeudi, 22 juillet 2021	08:00:00	•Avancement du rapport
	vendredi, 23 juillet 2021	08:00:00	•Correction du rapport
	samedi, 24 juillet 2021		
	dimanche, 25 juillet 2021		
	lundi, 26 juillet 2021	05:00:00	•Correction du rapport
	mardi, 27 juillet 2021	05:00:00	•Correction du rapport
	mercredi, 28 juillet 2021	08:00:00	•Correction du rapport
	mercredi, 28 juillet 2021	08:00:00	•Correction du rapport
	jeudi, 29 juillet 2021	11:00:00	•Correction du rapport
	vendredi, 30 juillet 2021	08:00:00	•Correction du rapport
	samedi, 31 juillet 2021		
août.21	dimanche, 1 août 2021	07:00:00	•Correction du rapport
	lundi, 2 août 2021		
	mardi, 3 août 2021		
	mercredi, 4 août 2021		
Temps total passé sur le TB	**302:00:00**		

Source : images de l'auteure

ANNEXE II : Product Backlog

Partie Documentation			
Tâches	**Status**	**Fini le...**	**Notes sur la tâche**
Lecture de précédent TB		10.05.2021	
Renseignement sur les technologies qui font de l'AR		13.05.2021	
Lire des livres, documents, thèses traitant de la réalité augmentée		15.07.2021	•Documentation tout au long de l'écriture du TB selon besoins
Lire documentation Unity		11.05.2021	•Avec Tuto
Lire documentation AR Foundation		11.05.2021	•Avec Tuto
Lire documenation OpenCV		14.05.2021	
Lire documentation EmguCV		17.05.2021	
Lire documentation Vuforia		24.05.2021	•Avec Tuto
Lire des tutoriel sur la reconnaissance de couleur par des applications		31.05.2021	
Lire des tutoriel sur la récupération des pixels sur une image		31.05.2021	

Partie pratique			
Tâches	Status	Fini le...	Notes sur la tâche
Lecture des documents		10.05.2021	
Préparation de l'organisation		10.05.2021	
Lecture et analyse du code		10.05.2021	
Tuto sur l'AR		11.05.2021	•ARTrackedImageManager •vuforia library
Tuto image tracking avec		11.05.2022	
Changer la couleur d'une forme via le code		12.05.2021	
Tuto reconnaissance des couleurs		25.05.2021	•OpenCV dans Unity •Comprendre la class Mat •Emgu CV
Créer les différents matériaux dans l'application DE TESTE, chaque matériaux d'une couleur différente qu'il faudra appeler selon la couleur de la pièce sur le plan		25.05.2021	•Pas besoin de Emgu CV ou OpenCV
Changer la couleur d'une forme par rapport à la couleur détectée		28.05.2021	•Optimiser la détection des couleurs qui est basé sur un pixel uniquement
Lier une forme sur un plan à la frome en 3d		03.07.2021	•Pas explicitement faisable à 100%
Changer la couleur de la forme en 3d selon la couleur de la forme sur le plan		03.07.2021	•Solution Vuforia avec beaucoup de plan et de forme 3D pour avoir toutes les combinaisons de couleur •Solution user ou l'utilisateur choisis la couleur qu'il veut donner à la pièce

Partie Rédactionnelle			
Tâches	Status	Fini le...	Notes sur la tâche
Organisation des différentes partie du TB		10.05.2021	
Introduction			
Etat de l'Art		02.07.2021	
Organisation du projet		22.07.2021	
Choix Technologique		15.07.2021	
Résulat		16.07.2021	
Discussion		23.07.2021	
Conclusion		23.07.2021	
Relectures		01.08.2021	
Corrections		01.08.2021	
Impression papier du TB		02.08.2021	

Sources : images de l'auteure

ANNEXE III : Versions des technologies

Vuforia : Version 9.6.3

Unity : Version 2019.4.1f1

DÉCLARATION DE L'AUTEUR

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du responsable de filière et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- Antoine Widmer

Sierre, le 1 août 2021

Sarah Morard