Université Gustave Eiffel

ENSG Géomatique

ÉCOLE NATIONALE DES SCIENCES GÉOGRAPHIQUES

Rapport de projet

Master 2 : Technologies des systèmes d'informations

Projet commun : API pour le positionnement d'image

Étudiants :

Vincent Miras
Ibrahim Sall
Loïs Bilheran
Hugo Geslin

Avril 2025

Commanditaires :

Mathieu Bredif
Ewelina Rupnik

Table des matières

Glossaire & Acronymes

- **API** : Application Programming Interface, ensemble normalisé qui sert de façade à un logiciel pour offrir ses services à un autre.

- **Benchmark (paragon)** : Suite de tâches conçue pour mesurer et comparer la capacité de raisonnement d'un modèle de langage (LLMs).

- **ETH, École polytechnique fédérale de Zurich** : Université technique et réputé, ayant grandement contribuée à la production du lamar-benchmark.

- **IGN** : Institut National de l'information Géographique et forestière.

- **LaMAR** : Localisation and Mapping for Augmented Reality.

- **LLMs, Language Learning Models** : (définition à compléter si nécessaire).

- **OGC, Open Geospatial Consortium** : Organisation internationale qui développe et promeut des standards.

- **Toolbox** : Ensemble d'outils et de traitements prédéfinis.

Introduction

Cette décennie voit l'avènement d'un projet majeur pour l'IGN : la création d'un jumeau numérique à l'échelle nationale. Ce projet ambitieux vise à produire un modèle 3D complet du territoire, tout en fournissant un ensemble de services associés pour en faciliter l'utilisation. Les données nécessaires à la constitution de ce jumeau numérique proviendront des référentiels géographiques de l'IGN, mais aussi des collectivités et de divers acteurs du territoire, afin de garantir un socle de données le plus précis et robuste possible.

L'un des usages envisagés de ce jumeau numérique concerne les utilisateurs se déplaçant sur le territoire français. Grâce à une surcouche numérique intégrée à la caméra d'un smartphone, il serait possible d'afficher en temps réel des informations supplémentaires sur les bâtiments ou l'environnement alentour. Pour permettre cette réalité augmentée (RA), il est impératif de pouvoir localiser précisément l'utilisateur et d'estimer l'orientation de sa caméra dans l'espace. La RA repose en effet sur la capacité à superposer de manière cohérente des informations numériques dans un environnement physique réel, ce qui nécessite une connaissance fine de la position et de l'orientation d'un dispositif de capture d'images, généralement une caméra, dans un référentiel global.

Le projet présenté dans ce rapport propose ainsi la mise en place d'un service de Visual Positioning System (VPS), fondé sur des standards ouverts et des outils existants. En combinant des images, des données GPS et divers capteurs intégrés, ce service vise à déterminer une GeoPose, c'est-à-dire la position et la rotation absolues de la caméra dans l'espace. Cette approche s'appuie notamment sur les travaux de la communauté Open AR Cloud, ainsi que sur le benchmark de localisation et de cartographie visuelle LaMAR, développé par Microsoft et l'ETH Zurich.

Chapter 1 | Technologies utilisées

1.1 Lamar-benchmark

1.1.1 Un jeu de données de référence

Le benchmark **LaMAR** (Localization and Mapping for Augmented Reality) est avant tout un jeu de données conçu pour s'adapter aux exigences de la localisation visuelle dans des environnements complexes et variés.

La conception de ce jeu de données a été guidée par le besoin de représenter fidèlement la complexité du monde réel. En effet, le jeu de données est annoncé comme résistant aux conditions météo, aux changements de luminosité dus à l'heure, aux déplacements de mobiliers, etc. Les acquisitions d'images s'étendent sur une durée d'un an, ce qui permet de prendre en compte à la fois des changements de courte et de longue durée.

Un aspect fondamental de ce jeu de données est son caractère multimodal. Il intègre, au-delà des images, une diversité de signaux provenant de capteurs variés :

- ❖ capteurs de profondeur (lidar, infrarouge),
- ❖ unités de mesure inertielle (IMU),
- ❖ signaux radio (WiFi, Bluetooth),
- ❖ dispositifs d'acquisitions variés : NavVis, HoloLens 2, iPhone/iPad.

Cette diversité rend le jeu de données particulièrement tolérant à la variabilité des conditions de capture.

1.1.2 Un algorithme de localisation par image

LaMAR est également un VPS (Visual Position System). Le principe d'un VPS est de pouvoir replacer la position à laquelle une image a été prise. Pour cela, un VPS analyse une image d'entrée afin de trouver des points d'intérêt. Il va ensuite comparer ces points d'intérêt avec ceux d'autres images qu'il connaît. En mettant en évidence les relations entre les points de plusieurs images, le modèle choisit ensuite la position la plus probable parmi celles qu'il a obtenues. Ce genre d'algorithme donne la position mais aussi la rotation de l'image dans le repère du jeu de données. Un VPS est donc souvent utilisé dans des cas où la localisation par GPS est impossible ou bien dans des cas où la précision est plus importante que la vitesse d'exécution. Le cœur de l'estimation de pose repose sur la toolbox **HLoc** (Sarlin et al. (2019)),

qui combine plusieurs étapes :

1. récupération d'images candidates par descripteurs globaux ;
2. mise en correspondance de points d'intérêt avec des descripteurs locaux (ex. SuperPoint) ;
3. estimation de la pose par PnP et triangulation sur un modèle SfM.

Cependant, l'approche LaMAR s'enrichit de l'intégration de capteurs complémentaires. Les signaux radio, notamment, permettent de contraindre spatialement la recherche d'images de référence.

Cette approche hybride permet d'atteindre une précision centimétrique, même dans des environnements dynamiques.

1.1.3 Algorithme et jeu de données liés par un format

Le format *Capture* est une extension du format *Kapture* développé par Naver Labs.

Les objectifs principaux du format sont les suivants :

- ❖ Offrir une structure modulaire pour l'organisation de données spatiales hétérogènes.
- ❖ Permettre l'alignement automatique de plusieurs sessions de capture au sein d'un même site.
- ❖ Fournir une API simple pour la manipulation des données (via `scantools.capture`).
- ❖ Faciliter la visualisation et la validation du processus de traitement.

La structure du format est centrée autour d'un répertoire principal, représentant un lieu de capture. Celui-ci contient l'ensemble des sessions effectuées sur ce site, ainsi que les résultats de leur alignement et des données de visualisation.

Structure détaillée des sessions

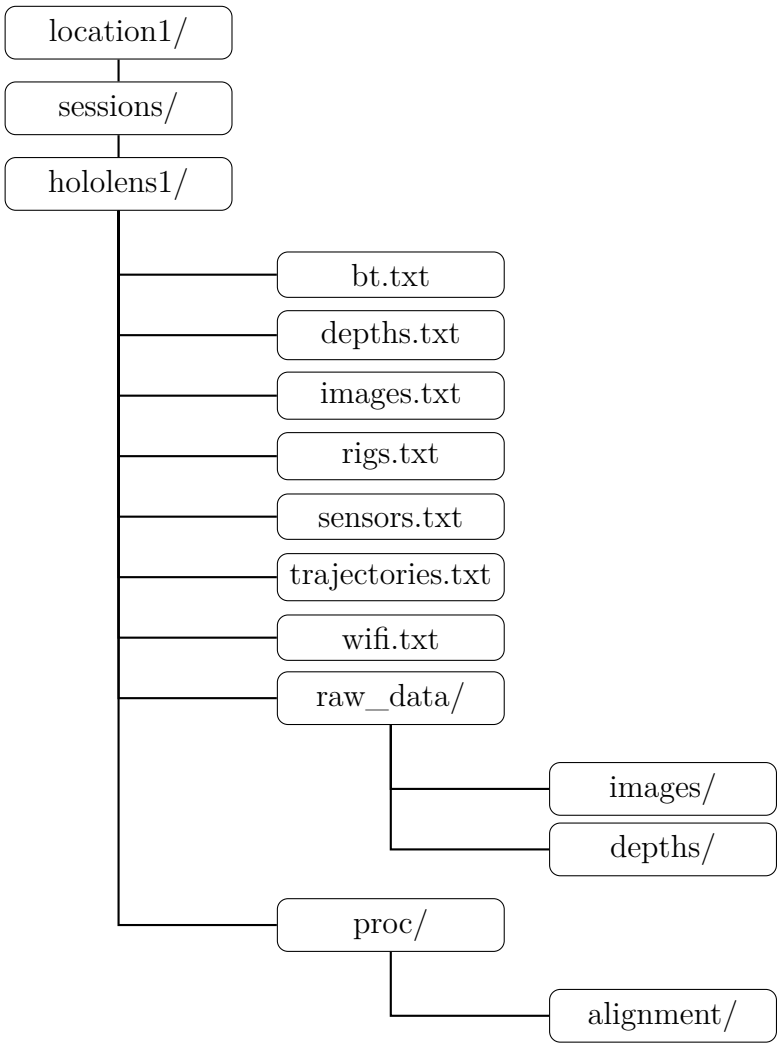

FIGURE 1.1 – Structure d'une session

Chaque session de capture comprend les fichiers suivants :

- **sensors.txt** : spécification des capteurs utilisés.
- **rigs.txt** : transformations caméra → rig.
- **trajectories.txt** : poses capteur → monde.
- **images.txt**, **depths.txt** : index des données visuelles.
- **wifi.txt**, **bt.txt** : mesures de contexte.
- **origins.txt** : pose initiale de la session.

- ❖ **raw_data/** : données brutes.
- ❖ **proc/** : données traitées (maillages, alignements, etc.)

Les fichiers utilisés dans ce projet se présentent sous plusieurs formats spécifiques. Le format **.txt** correspond à des fichiers texte encodés en UTF-8, où les données sont séparées par des virgules.
Le format **.ply** est utilisé pour les nuages de points et les maillages, avec une préférence pour le format binaire.
Les fichiers d'images sont généralement en **.png** ou **.jpg**, et peuvent être soit des images RGB, soit des cartes de profondeur, ces dernières étant en **.png** avec une profondeur de 16 bits.
Enfin, les fichiers **.h5** sont des fichiers HDF5 contenant les caractéristiques (features) et les appariements d'images, qui sont gérés par l'outil *hloc*.

1.2 GeoPose

GeoPose est un standard OGC. L'objectif de ce standard est d'être un vecteur d'échange pour la localisation et l'orientation d'objets géométriques réels ou virtuels (« Poses ») dans des cadres de référence ancrés à la surface de la Terre (« Geo ») ou dans d'autres systèmes de coordonnées.

GeoPose est structuré en 8 cibles de standardisation indépendantes entre elles. Ces cibles sont regroupées en 3 niveaux d'implémentations technologiques (Basic, Advanced et Composite) correspondant à différents cas d'usages.

Niveaux d'implémentations	**Types de cibles**
Basique	Basic-YPR (Yaw, Pitch, and Roll) Basic-Quaternion
Avancé	Advanced
Composite	Chain Graph Regular Time Series Irregular Time Series Stream

TABLE 1.1 – Classification des types de cibles GeoPose

Dans le cadre de notre projet, le niveau de standardisation à atteindre pour le cas d'usage souhaité est celui qui est préconisé par le Standard OGC GeoPose, à savoir le niveau 'Advanced'.

```
{
  "frameSpecification": {
    "authority": "EPSG",
    "id": "4979",
```

```
    "parameters": "longitude=-76.3000&latitude=47.7000&
      height=11.000"
  },
  "quaternion": {
    "x": 0.20056154657066608,
    "y": -0.08111602541464237,
    "z": 0.36606032744426537,
    "w": -0.9050939692261301
  },
  "validTime": 1630560671227
}
```

Le format avancé de GeoPose contient plusieurs paramètres clés. La section `frameSpecification` précise le système de référence utilisé, incluant une autorité (par exemple, EPSG), un identifiant de référence géodésique tridimensionnelle, ainsi que des paramètres de position tels que la longitude, la latitude et l'altitude.
L'orientation est décrite à l'aide d'un quaternion, représenté par quatre composantes (x, y, z, w), permettant de définir une rotation dans l'espace 3D de manière efficace et sans ambiguïté.
Enfin, le champ `validTime` fournit un horodatage en millisecondes depuis l'époque Unix.

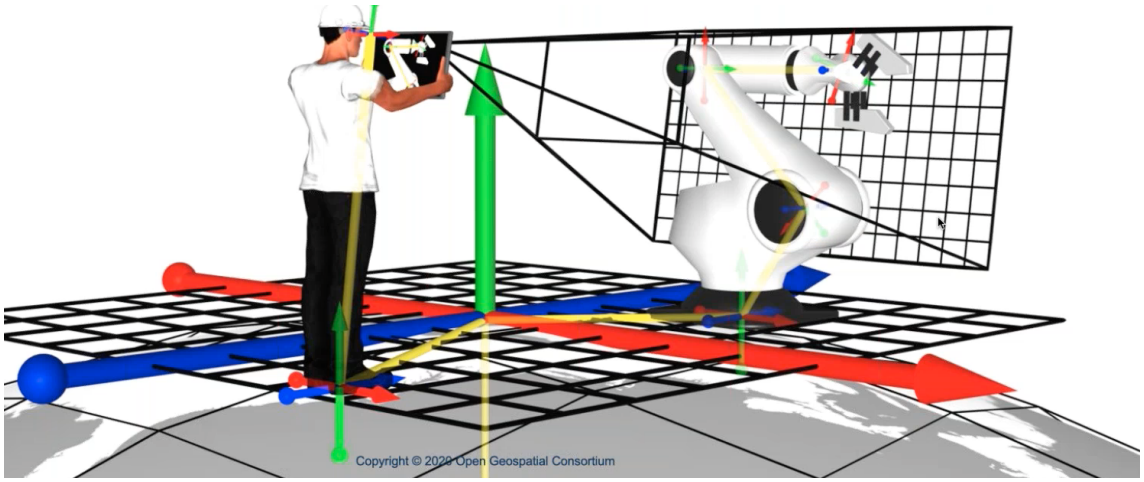

FIGURE 1.2 – Exemple de représentation d'une GeoPose

Pour faciliter l'utilisation de ce standard, un répertoire Github (Open AR Cloud (2021)) propose un exemple d'implémentation du protocole. Ce code permet, à partir de fichiers contenant les informations de la caméra, de renvoyer la GeoPose en respectant le niveau "Advanced". L'API proposée utilise donc cette architecture pour simuler une requête contenant des informations de caméra à un serveur. Ce dernier va donc traiter la requête qui, par défaut, va être d'écrire des valeurs par défaut dans une GeoPose. À la suite de ça, le serveur renvoie la GeoPose vers le client qui va pouvoir l'afficher dans la console.

L'avantage de ce répertoire est que le code est extrêmement modulable et permet donc de lier avec celui-ci des fonctionnalités supplémentaires. De plus, il est documenté de manière à être rempli de manière claire pour les fonctions primaires et suffisamment explicite pour les fonctionnalités que l'on souhaite ajouter.

Chapter 2 | Architecture

2.1 Mise en place du service Web

2.1.1 Partie Serveur

Le projet est basé sur un serveur Flask composé de trois routes principales :

La route `/process` correspond à une route de type POST permettant de soumettre l'image dont on souhaite la localisation, accompagnée d'un ensemble de fichiers de données capteurs. Le serveur enregistre localement ces fichiers dans un répertoire, puis déclenche un processus de localisation via l'appel à la route `/GeoPose`.

La route `/GeoPose` également de type POST traite des requêtes structurées au format GeoPoseRequest, contenant des données de capteurs et une image encodée en base 64. Après avoir converti GeoPoseRequest au format Capture, le serveur lance l'algorithme Lamar via un conteneur Docker pour estimer la GeoPose de l'appareil ayant capturé l'image. Les résultats sont extraits d'un fichier texte généré (`poses.txt`), puis convertis au format de coordonnées géographiques standard WGS84. La réponse finale, sous forme de GeoPoseResponse est renvoyée à l'utilisateur.

Enfin, la route `/swagger` offre une interface web interactive pour l'exploration et les tests de l'API. Cette interface repose sur un fichier de spécification Swagger situé localement (`static/swagger.json`) et permet à l'utilisateur de tester les différentes routes en temps réel, en modifiant les paramètres et les entrées.

2.1.2 Partie Client

L'interface (fig : 2.1) a pour objectif de fournir à l'utilisateur une plateforme simplifiée permettant :

- ❖ d'importer une image,
- ❖ de sélectionner un dossier contenant des fichiers de géolocalisation,
- ❖ de lancer un traitement côté serveur,
- ❖ puis de visualiser les résultats de manière textuelle et cartographique.

L'interface est structurée suivant une logique d'usage en trois étapes principales :

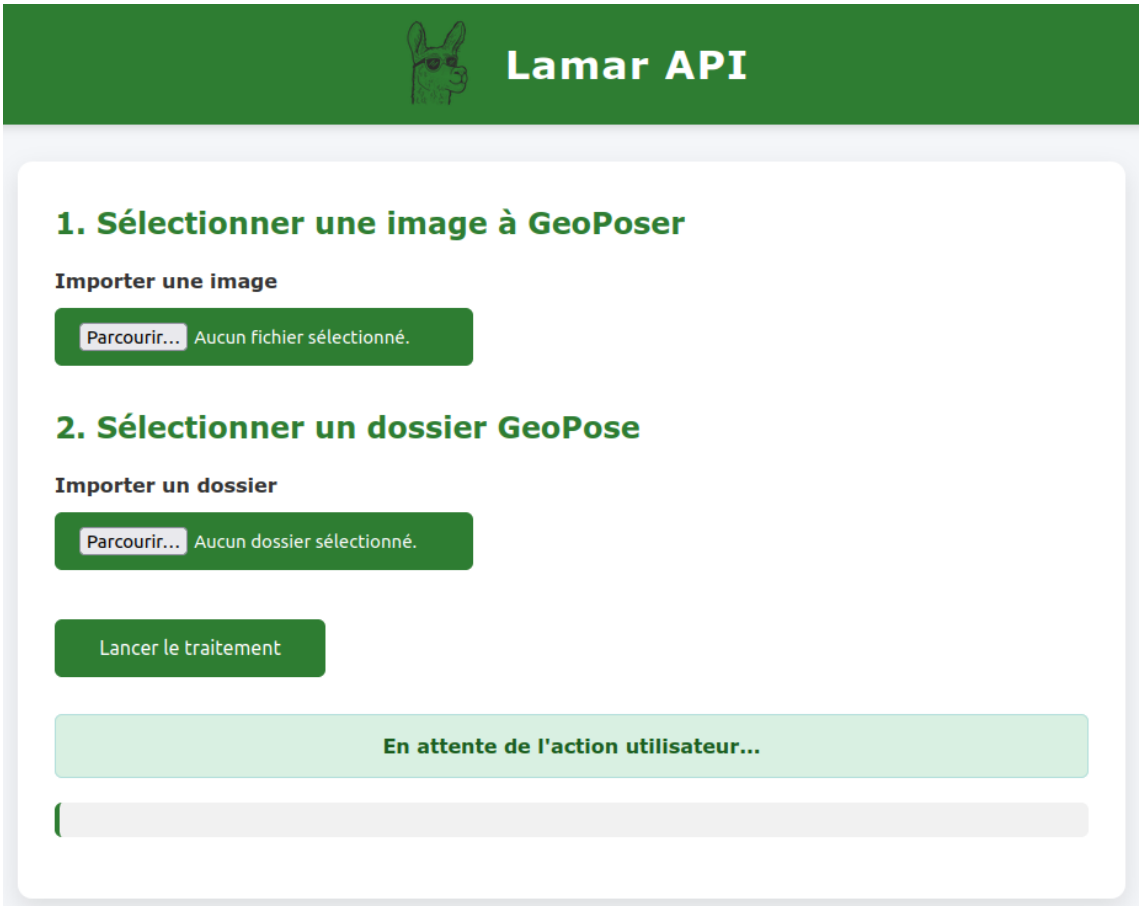

FIGURE 2.1 – Visuel du site web en arrivant.

1. **Importation de l'image** : L'utilisateur sélectionne une image locale. Un aperçu de celle-ci est automatiquement généré afin de confirmer la sélection (fig : 2.2).

2. **Importation des données capteurs** : Un dossier contenant les fichiers de données capteur est sélectionné via un champ adapté à la navigation dans les répertoires.

3. **Lancement du traitement** : Un bouton unique permet de transmettre l'ensemble des fichiers au serveur pour analyse et traitement.

Des zones d'affichage dynamiques (fig : 2.3) permettent de suivre le déroulement du processus :

- ❖ Un indicateur de statut informe l'utilisateur du retour du serveur (code HTTP) avec une mise en forme visuelle appropriée.

- ❖ Une zone de sortie textuelle présente les résultats structurés du traitement (type, identifiant, timestamp, position géographique).

- ❖ Une carte interactive affiche la position géographique si celle-ci est disponible.

- ❖ Un lien automatique vers Google Street View permet de visualiser l'emplacement dans son environnement réel.

Cette interface repose sur des technologies web standard : `HTML`, `CSS` pour la mise en forme, et `JavaScript` pour la logique de traitement côté client. L'utilisation de `Leaflet.js` permet l'intégration d'une carte interactive utilisant les données d'OpenStreetMap.

2.2 Choix techniques

L'un des principaux défis de ce projet a été la contrainte de temps. Les premières semaines ont été consacrées à l'exploration et à la compréhension de la documentation associée aux différentes briques techniques que nous envisagions d'utiliser. Un léger retard a été accumulé notamment lors de l'intégration et des tests de l'algorithme LaMAR, en raison de sa complexité et de son environnement de déploiement spécifique.

Afin de maximiser notre efficacité, nous avons opté pour une architecture modulaire, en connectant les composants existants à l'aide d'adaptateurs, sans modifier leur code source. Ce choix nous a permis de tirer parti des outils déjà éprouvés tout en limitant les risques liés à une modification en profondeur de leurs mécanismes internes.

Les différents modules étant majoritairement développés en Python, nous avons naturellement conservé ce langage. Il est à la fois bien maîtrisé par l'équipe, largement documenté et adapté aux besoins de prototypage rapide, tout en restant robuste.

Les données d'entrée nécessaires au calcul de la GeoPose sont volumineuses et hétérogènes. Pour en faciliter le traitement, nous avons développé une interface de prétraitement permettant de convertir et structurer les fichiers bruts dans un format adapté à l'API.

Enfin, dans un souci de portabilité et de reproductibilité, l'ensemble du système a été conteneurisé via Docker. Ce choix garantit une cohérence des environnements de développement, de test et de déploiement, tout en simplifiant la maintenance et le partage du projet.

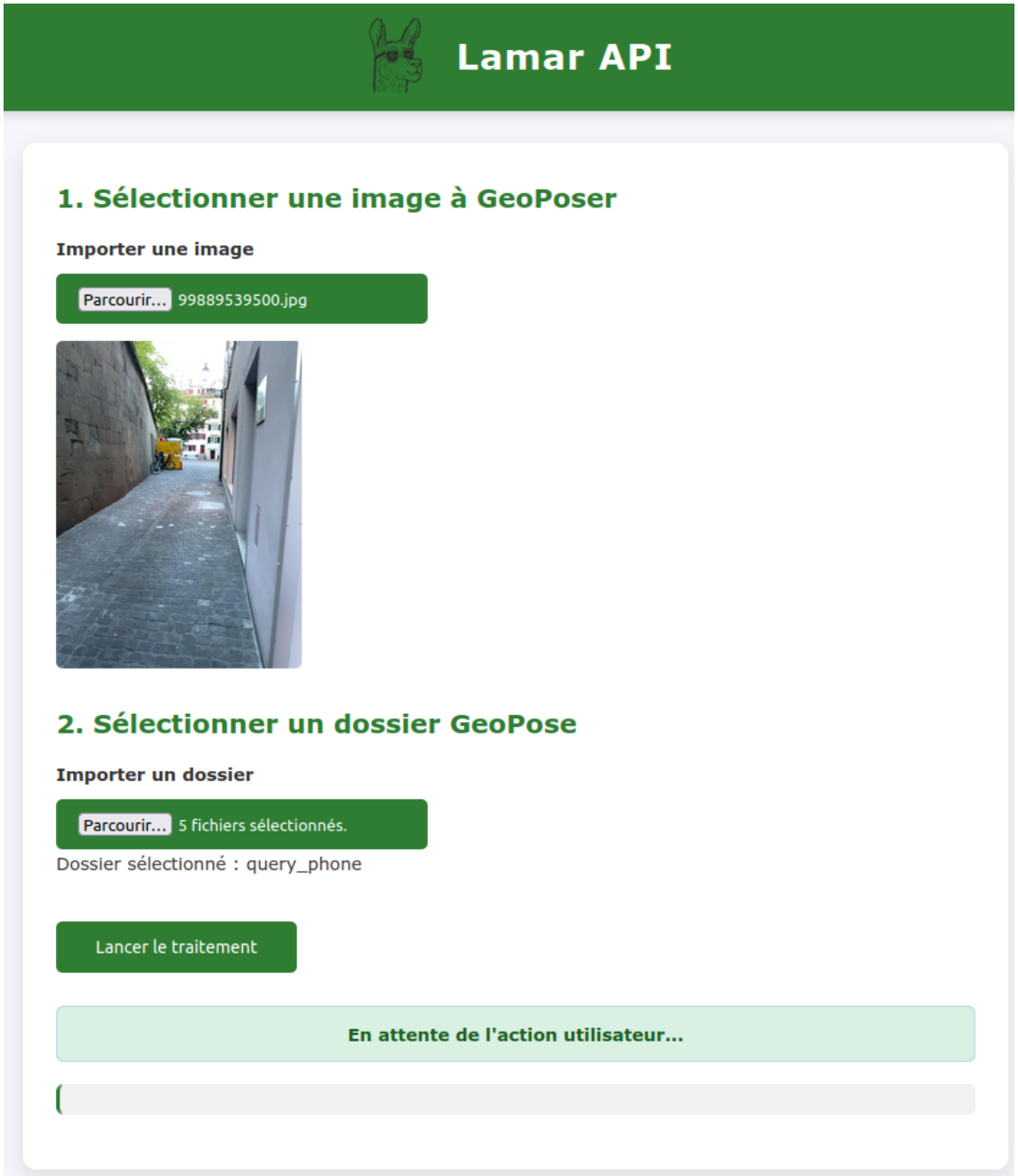

FIGURE 2.2 – Visuel du site web avec des données rentrées.

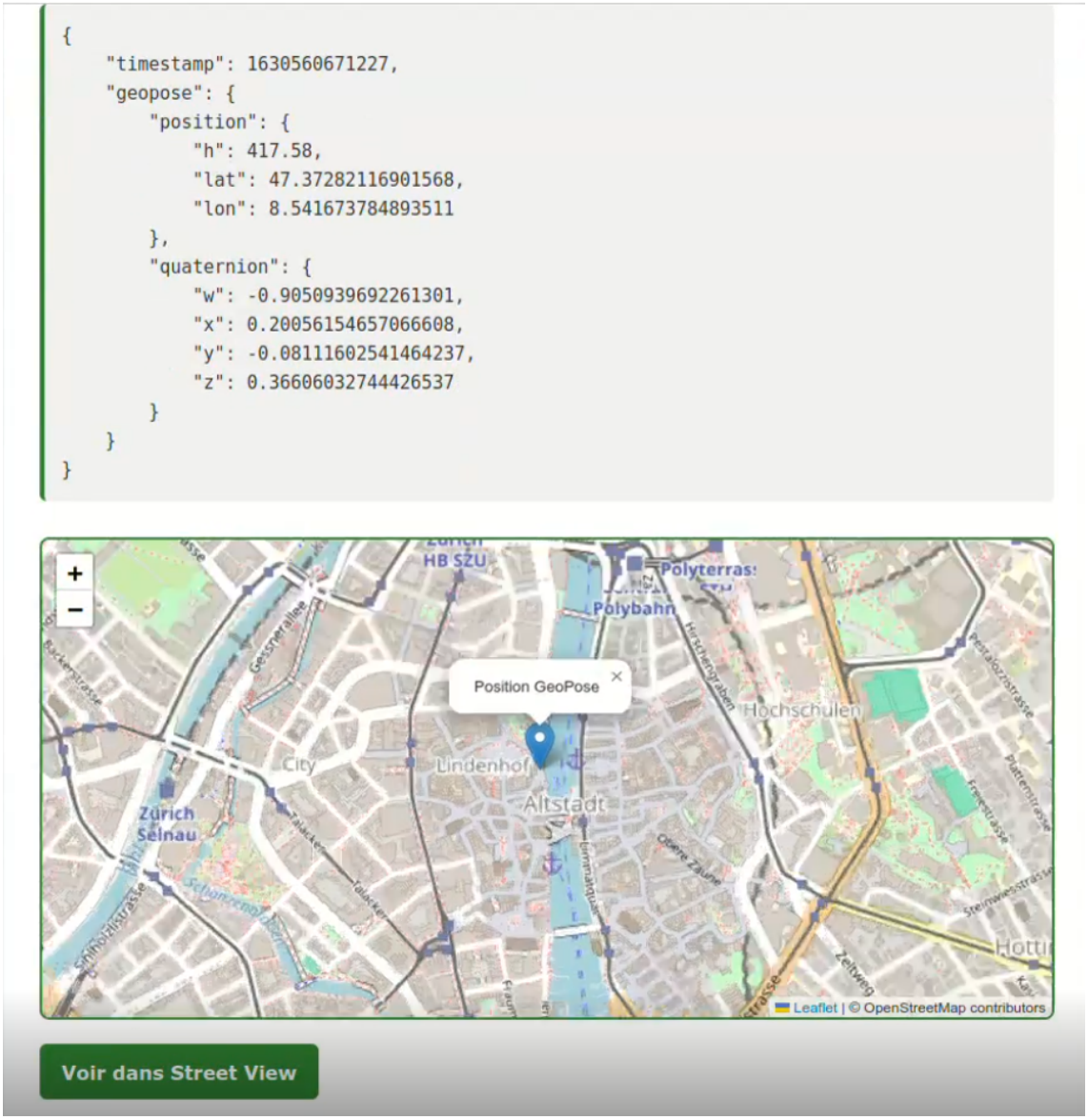

FIGURE 2.3 – La GeoPose et la carte Leaflet à la fin des traitements.

2.3 Trajet de la donnée

Le schéma ci-dessous illustre le flux de données au sein de notre serveur, depuis l'utilisateur jusqu'à la génération de la géoposition globale. Ce processus suit une séquence d'étapes cohérentes qui impliquent plusieurs modules interconnectés.

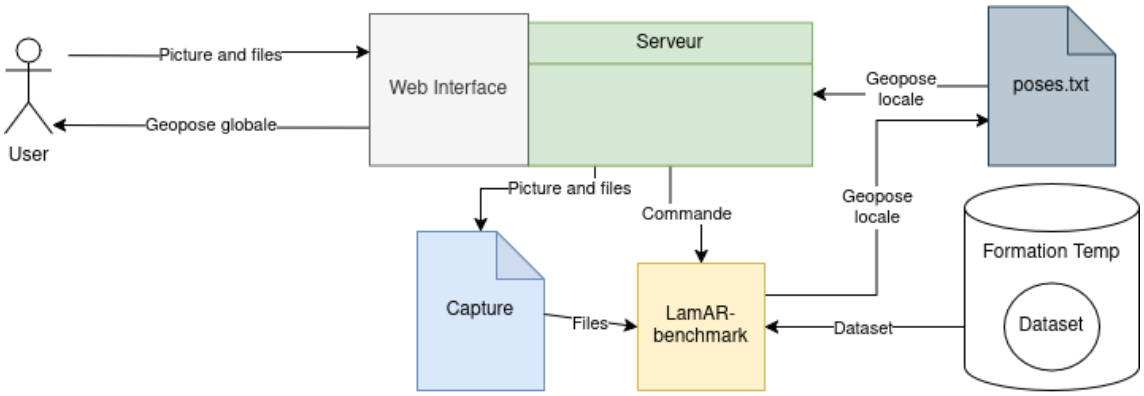

FIGURE 2.4 – Architecture du trajet de la donnée

1. **Passage des données par l'utilisateur**
   L'utilisateur interagit via une interface web dédiée. Il y soumet l'image qu'il souhaite géoréférencer ainsi que les fichiers txt associés s'il en a (*sensors.txt, trajectories.txt, images.txt, wifi.txt* et *bt.txt*). Il recevra en sortie une géoposition globale estimée à partir des données transmises.

2. **Stockage des données d'entrée**
   L'interface web transmet les fichiers au serveur qui enregistre l'ensemble des données dans un fichier *Capture*.

3. **Lancement du traitement `LamAR-benchmark`**
   Une fois les fichiers enregistrés, le serveur déclenche automatiquement l'exécution du programme `LamAR-benchmark`. C'est la pipeline qui prend en entrée les *données de Capture* ainsi que *dataset de référence* d'où provient l'image. De par sa lourdeur, le dataset est déposé sur le stockage interne de l'école « Formation Temp ».

4. **Génération de la GeoPose locale**
   Le programme `LamAR-benchmark` traite les entrées et génère une GeoPose locale, qui est ensuite écrite à la fin du fichier `poses.txt`, constituant une nouvelle ligne du fichier.

5. **Récupération et traitement par le serveur**
   Le serveur récupère la GeoPose locale dans le fichier `poses.txt` une fois le traitement terminé. Il effectue ensuite une transformation pour produire une géoposition globale dans le repère WGS84.

6. **Transmission des résultats à l'utilisateur**
   Enfin, la géoposition globale est renvoyée à l'utilisateur via l'interface web. Ce retour constitue la fin du trajet de la donnée dans notre architecture. La GeoPose est affichée sur l'interface Web, et une carte permet de la situer.

2.4 Diagramme de composants

2.4.1 Structure Générale

L'architecture du système (fig : 2.5) est composée de plusieurs modules qui permettent de séparer les responsabilités (upload, traitement, serveur, client) tout en favorisant l'extensibilité du système. L'intégration de Docker garantit la reproductibilité des traitements, et Flask assure la souplesse côté serveur.

2.4.2 Flux d'exécution

1. Le script `server.sh` est exécuté par le fichier `docker-compose.yaml` et démarre le serveur `demo_server.py`.

2. Lorsque l'utilisateur ajoute une image et des fichiers via le client web, le serveur appelle les fonctions de `to_capture.py` pour enregistrer les fichiers.

3. Le client (`demo_client.py`) est ensuite exécuté, et envoie les informations de l'utilisateur via une requête POST à l'endpoint `/geopose`.

4. Le serveur sauvegarde alors l'image et les données du côté du docker de LamAR.

5. Le traitement est réalisé via Docker par `demo_docker.py`.

6. La sortie est parsée et une réponse GeoPose est retournée au client.

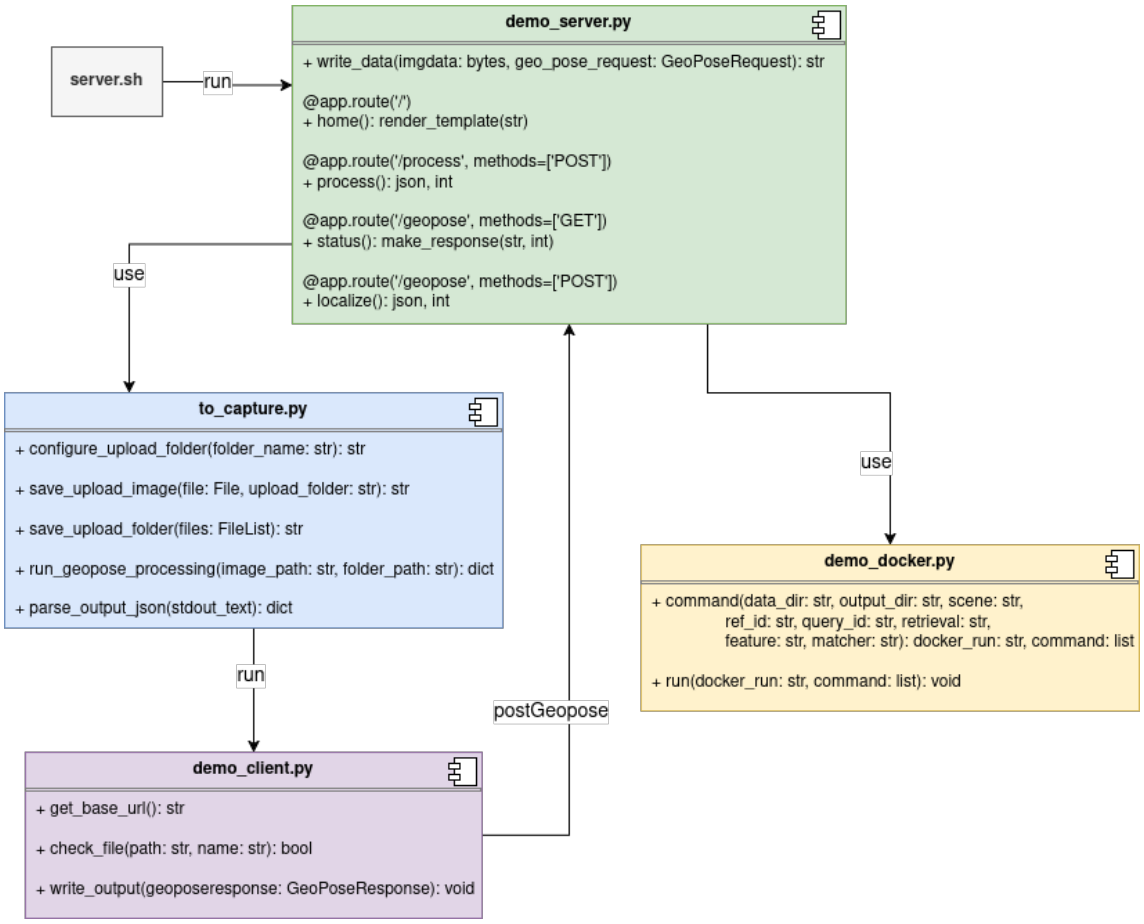

FIGURE 2.5 – Diagramme de composants

Chapter 3 | Évaluation du positionnement

La précision du positionnement est une partie importante du projet. En effet, ayant pour but de repositionner les appareils de réalité augmentée, il faut être plus précis que ce que le casque et autres dispositifs peuvent faire actuellement. Ainsi, la chaîne de traitement LaMAR a déjà une précision de 5 cm pour une dérive de 10 cm, soit 50 %. Cependant, on ajoute sur cette couche LaMAR un passage des coordonnées locales aux coordonnées globales. Ce passage nécessite de faire un géo-positionnement du modèle dans lequel se trouvent les points, puis d'interpoler les coordonnées de sortie afin d'obtenir la position globale de notre image.

Pour avoir une notion de précision, il faut donc commencer par la précision du positionnement du modèle. Pour cela, trois points ont été pris sur Google Maps. Ces trois points correspondent à des centres optiques de photos du jeu de données. En replaçant ces trois points sur Google Maps, on obtient une « vérité terrain » de ces points en WGS84. Après cela, un script Python permet de faire une conversion de tous les autres points appartenant au jeu de données associé.

— Évaluation de la précision —	
Point 1 - Erreur	2.7348 m
Point 2 - Erreur	8.0381 m
Point 3 - Erreur	5.3033 m
Résumé des erreurs	
Erreur moyenne	5.3588 m
Erreur max	8.0381 m
Écart-type	2.1654 m
Erreur RMS	5.7797 m

Table 3.1 – Évaluation de la précision et résumé des erreurs

Ainsi, la précision de notre résultat dépend de plusieurs facteurs. Certes, la précision de l'interpolation est mesurable ; cependant, la plus grande marge d'erreur vient de notre vérité terrain, qui ne peut être qu'une simple estimation visuelle d'une photo. L'estimation de la précision de notre interpolation est donc possible, mais limitée. Pour cela, on compare les coordonnées de notre vérité terrain avec celles qui sont interpolées. Cette comparaison donne un décalage maximal de 8 m (voir Table 3.1), ce qui peut être dû à notre script, mais aussi à la précision du MNT utilisé pour l'interpolation de la hauteur.

Ainsi, il faut mettre en évidence ce grand manque de précision dans le cas d'un dataset en coordonnées locales à estimer. Cependant, ce passage est obligatoire car le jeu de données fourni nécessite cette transformation. La solution à ce problème est l'utilisation d'un dataset en coordonnées globales ou bien de s'assurer d'une transformation du modèle suffisamment précise afin de ne pas avoir d'estimation à effectuer. Or, dans la réalité et le contexte de ce projet, l'IGN souhaite exploiter son propre jumeau numérique. Les positions et orientations des caméras sont alors déjà référencées avec une grande précision. On n'a de fait plus besoin de passer par un repère local dans ces conditions, et les GeoPoses fournies ne présentent plus que l'incertitude et les erreurs du *lamar-benchmark*.

Chapter 4 | Limites rencontrées et axes d'amélioration

4.1 Utilisation d'un autre jeu de données

Tout au long de notre projet, nous avons rencontré des difficultés en tentant d'utiliser des jeux de données autres que celui de *LIN*. Nous avons essayé de nous baser sur :

- ❖ Un jeu de données produit au sein de notre établissement via l'application iOS **ScanCapture**, elle aussi disponible sur le Git de LaMAR.

- ❖ Les données de **Panoramax**, réalisées par l'IGN aux alentours de l'école. Pannoramax est l'alternative libre et souveraine au plus réputé *StreetView* de Google.

Mais dans les deux cas, nous nous sommes heurtés à l'absence du fichier *"trajectories"*. Celui-ci est produit par les capteurs NavVis et contient les positions du capteur dans l'espace grâce à une centrale inertielle très aboutie. Dès lors que ce fichier est manquant, il devient impossible d'entraîner le modèle avec le benchmark. Or, ces jeux de données sur lesquels nous avons un contrôle total nous auraient permis d'éviter l'anonymisation et d'utiliser des points de liaison, qui auraient permis un géoréférencement absolu avec une erreur bien moindre que la solution actuellement utilisée.

4.2 Gain en précision

La robustesse de l'algorithme de LaMAR provient de son format **Capture** et de ses sessions. Comme l'expliquent les développeurs du benchmark dans leur rapport Peretroukhin et al. (2023) : « L'évaluation d'une seule image est trop limitée et n'est pas représentative des performances de l'AR. » Il faudrait alors modifier le benchmark en conséquence, afin qu'il puisse s'appuyer sur les images précédentes pour estimer la position actuelle de la caméra. Il est, en l'état, seulement possible de recalculer toutes les positions de la session, ce qui améliore la précision mais allonge grandement le temps de calcul.

4.3 Scalabilité et mise en production

Actuellement, notre API et les conteneurs qu'elle emploie sont hébergés sur un seul nœud. Il faudrait, pour accepter plus de requêtes et permettre la résilience :

- ❖ Multiplier le nombre de nœuds,

- ❖ Répliquer les conteneurs et leurs états en cas d'interruption d'un nœud,

❖ Assurer la précision des résultats et détecter les fautes.

4.4 Temps de réponse

De plus, dans l'état actuel de notre architecture, les données du jeu de données LIN sont stockées sur *formationtemp*, un serveur Samba de l'école. Le temps de réponse pour un gros volume de données comme le nôtre est conséquent et ne permet pas à notre API d'être uniquement dépendante des temps de calcul. Le temps de réponse dépend également du transfert des données. Il faudrait donc rapprocher au maximum les conteneurs de calcul des données essentielles à ceux-ci. On imagine alors la mise en place directe d'un serveur de données et de calcul.

4.5 Anonymisation

Dans le cadre de notre projet, nous nous sommes appuyés sur des jeux de données anonymisés grâce à *Brighter AI* ; les images et données de validation l'étaient elles aussi. De fait, pour respecter le RGPD, il faudra que les données du jumeau numérique soient anonymisées (par exemple, flouter les visages). Il faudra également prendre en compte la possibilité d'anonymiser les requêtes, puisque des données

Conclusion

Pour conclure, ce projet représente une première étape vers l'aboutissement de l'ambition finale. En effet, cette première phase met déjà en évidence certains problèmes qui pourraient ne faire qu'être amplifiés à plus grande échelle. Cependant, le retour d'expérience donne déjà des signes positifs.

Premièrement, la faisabilité de ce type de prototype est avérée. Même si cette étape n'a pas été conclue dans le temps imparti, il ne reste plus grand-chose à faire pour que cette phase devienne fonctionnelle. Les différentes briques composant le projet, telles qu'elles sont actuellement, semblent adaptées et respectent des standards qui faciliteront les transferts de données.

Ensuite, en ce qui concerne les performances, on rencontre la première grande problématique. Le prototype actuel met environ 3 minutes pour s'exécuter de bout en bout. Ce délai semble anodin au premier abord, mais il aurait d'énormes conséquences sur l'avenir de ce projet. En effet, l'objectif à long terme est de permettre le traitement en temps réel et de corriger les erreurs de positionnement des casques ou des téléphones en réalité augmentée. Un délai aussi long entraînerait plus d'erreurs que de réels avantages dans l'état actuel des choses.

D'un point de vue architectural, le prototype est adapté pour un utilisateur unique, mais il n'est pas conçu pour être utilisé simultanément par plusieurs utilisateurs. Ce point, qui est crucial pour le projet dans son ensemble, doit être étudié, et l'architecture devrait être modifiée en fonction des solutions de scalabilité choisies.

Enfin, ce projet constitue une avancée significative dans la direction finale et représente un grand pas en avant dans la première analyse de ce qui est réalisable. De plus, le retour d'expérience offre une analyse critique des composants utilisés, ce qui permettra de guider l'évolution à long terme dans les meilleures conditions.

Bibliographie

Open AR Cloud. OSCP GeoPose Protocol. https://github.com/OpenArCloud/oscp-geopose-protocol, 2021. Accessed : 2025-04-15.

V. Peretroukhin, Y. Zang, S. Zhang, J. Engel, C. Cadena, V. Koltun, and D. Scaramuzza. LaMAR : Benchmarking Localization and Mapping for Augmented Reality. Lamar-poster.pdf, 2023. Poster presented at CVPR 2023.

P.-E. Sarlin, C. Cadena, R. Siegwart, and M. Pollefeys. From coarse to fine : Robust hierarchical localization at large scale. https://github.com/cvg/Hierarchical-Localization, 2019. Accessed : 2025-04-15.