ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Semester Thesis

Cloud-based Spatial Understanding for Improved VIO State Estimation and Human-Robot Interaction

VIO drift compensation & Mixed Reality drone control using Azure Spatial Anchors

Spring Term 2021

Supervised by:
Michael Pantic
Daniel Dugas

Author:
Eric Vollenweider

Contents

Abstract

Robot state estimation and human-robot interaction are two essential concepts in robotics, highlighted by recent advancements in the research and development of omnidirectional drones. The first is often implemented using Visual Inertial Odometry (VIO) pipelines, which achieve high update rates and have low latencies but suffer from drift. Fusing GPS or other absolute localization data into the VIO pipeline can compensate for the drift. Especially in the case of omnidirectional drones, the human-robot interaction requires more research to allow non-domain experts to intuitively control these robots since conventional controllers are too simple for these drones. Both of these challenges require spatial understanding to be solved.

This project investigates whether cloud-based Structure from Motion (SfM) pipelines such as Azure Spatial Anchors can be used to improve an omnidirectional drone's state estimation and to allow human operators to control the robot more intuitively using Mixed Reality (MR) interfaces. Furthermore, it asks whether individual devices' capabilities can be leveraged to improve the system's overall performance by smartly enabling MR devices and drones to collaborate on building a cloud-based point cloud.

The proposed concept of connecting the drone and a MR device to the Azure Spatial Anchor (ASA) cloud service was implemented and tested on hardware utilizing an Extended Kalman Filter-based VIO state estimator and Microsoft's HoloLens 2 as a MR device and the Robot Operating System (ROS) as a communication platform between the two. Over 15 different configurations and approaches were implemented.

A thorough analysis shows that drift can indeed be successfully compensated. However, currently available cloud-based SfM pipelines with localization standard deviations of 5 to 10 cm are not precise enough to replace GPS data fusion. Furthermore, the achievable update frequency of about 1Hz is not sufficient for the tested filter-based state estimator. Finally, the ASA cloud service's robustness was tested in four different scenarios, including a precision and internet bandwidth test. It was shown that the drone can localize itself more precisely using a point cloud generated by the HoloLens 2 compared to drone-generated point clouds.

While the developed MR interface was not evaluated with statistical significance, it was intuitive to use and straightforward to implement since the drone already utilized the same SfM platform as the MR device needed for the co-localization of the drone and the HoloLens 2.

Symbols

Symbols

ϕ, θ, ψ	roll, pitch and yaw angle
b	gyroscope bias
Ω_m	3-axis gyroscope measurement
T_{AB}	Homogeneous Transformation transform vectors expressed in B to A

Acronyms and Abbreviations

ASA	Azure Spatial Anchors
ETH	Eidgenössische Technische Hochschule
EKF	Extended Kalman Filter
IMU	Inertial Measurement Unit
IOT	Internet of Things
UAV	Unmanned Aerial Vehicle
SfM	Structure from Motion
VIO	Visual Intertial Odometry

Chapter 1

Introduction

The robotics field has a few important key concepts most mobile robotic systems depend on to work flawlessly. Two of which are robot-state estimation and human-robot interaction. Practically every mobile robot needs to be aware of its current pose relative to its environment to fulfil its purpose. Similarly, many robots interact with humans directly or indirectly by having a human control/teleoperate it. The two challenges (state estimation and human-robot interaction) are often treated as separate and independent problems, although both require spatial intelligence to be solved.

Spatial intelligence is a term, which summarizes a class of methods and systems, that help devices such as smartphones and robots to understand their physical surroundings. With recent advancements in consumer-grade Mixed Reality (MR), more companies are beginning to offer cloud services around spatial intelligence.

One cloud-based services class is called Spatial Anchors. It allows MR devices to place anchors in their physical surroundings before other devices can query the anchors' pose. These Spatial Anchor platforms perform some variant of Structure from Motion (SfM) and build online point-clouds. This allows devices to localize themselves and each other in their environment.

A mobile robot's state estimation precision strongly impacts its ability to execute its mission. Many robots are limited by their onboard computational power, which is especially true for drones. These robots can benefit from cloud-based solutions by offloading work involved in state estimation onto servers. These are not restricted by weight and power consumption, while possibly allowing the robots to improve their state estimation accuracy.

Cloud-based spatial intelligence services are also routinely used in MR applications to localize themselves and other devices in their environment. MR devices such as the HoloLens 2 can be regarded as advanced human-computer interfaces, allowing humans to interact with computers intuitively and more human-like using 3D holograms instead of 2D screens or hardware controllers.

With recent advancements in the development of omnidirectional drones and their methods to plan flights while interacting with surfaces [1], controller inputs do not translate to intuitive behaviors of the drone anymore. This is because the

trajectory generators operate in a non-angle preserving 2D subspace. The robotic community could use MR apps to develop more intuitive high-level interfaces for complex drones using the same cloud-based spatial intelligence services to localize them.

That prompts the question of whether cloud-based SfM pipelines can be used in robotic systems to enhance the performance of the robot's state estimation and the human-robot interaction by enabling multiple IoT devices to collaborate towards a common goal, allowing each device to contribute and benefit.

This can be achieved by connecting the robot and the MR device to these cloud-based spatial intelligence services. Then all devices can contribute to the online point clouds to jointly create the basis to solve the state estimation and human-robot interaction problem. Robots can query the point cloud to achieve a more precise state estimation. At the same time, the MR device can co-localize itself with the drone, which allows the human to control the robot more intuitively. In other words, the HoloLens 2 can provide its map to the drone, which uses it to improve its state estimation, localize itself in the environment, and allows the MR device to control the drone.

In conclusion, this research tries to answer the following questions:

1. Is it possible to improve a Visual Inertial Odometry (VIO) state estimator's performance using cloud-based SfM pipelines, and if so, by how much?
2. Can other devices contribute their spatial understanding of the environment and improve the state estimation further?
3. Which limitations do cloud-connected computer vision systems have in robotics, what are their implications, and how can they be handled?
4. Can the same platform be used to develop intuitive controllers for complex scenarios?

This report continues with a description of the State of the Art in chapter 2, followed by a description of the concept chapter 3, explains the implemented architecture and some interesting details in chapter 4, goes into depth with analysing the results of this work in chapter 5 and finally draws a conclusion in chapter 6.

Chapter 2

State of the Art

2.1 Robot State Estimation

While other methods are available, many mobile robots use Visual Inertial Odometry (VIO) pipelines to estimate their current state. Filter-based VIO pipelines have the benefit of achieving high frequencies for relatively low computational costs, which makes them suitable for flying robots (drones), which often have strict limitations in terms of computational power and latency. These pipelines do not build maps of their surroundings, which makes them vulnerable to drift [2]. A drifting state is a severe problem since the robot assumes it is at a different location than it actually is, the effects of which are likely to hinder the robot from fulfilling its purpose. Furthermore, estimating the yaw angle using VIO is hard since the yaw angle relative to gravity-aligned coordinate frames is unobservable using the IMU [2].

Absolute pose inputs from GPS can compensate for the drift in VIO systems, for example, which is often used in combination with VIO pipelines to achieve a robust state estimation [3, 4, 5]. GPS has two drawbacks: It only provides position, and GPS connection is not given inside buildings or underground, making the combination fragile concerning the yaw angle estimation and whenever GPS is not available.

An alternative solution to compensate the drift of VIO state estimators is to use a spatial anchor and repeatedly query the anchor during the drone's operation. An anchor query returns the relative transform from the camera to the virtual anchor. If the robot knows that the virtual anchor should be at a given location, the drift can be worked out easily and thus compensated. Unfortunately, the precision of specific anchor query operations is unknown. This means that a proposed solution has to estimate the anchors' covariance and handle the uncertainties correspondingly.

2.2 Visual Intertial Odometry (VIO)

Visual Intertial Odometry (VIO) systems use both visual and inertial sensors (one or more cameras and Internal Measurement Units (IMU)) to estimate the robot's pose and speed. In VIO pipelines, cameras and IMUs complement each other. While cameras are precise during slow motions, they become less useful during quick movements due to motion blur. Furthermore, cameras are negatively influenced by lighting changes, over-exposure and even suffer from scale ambiguities in monocular setups. IMUs, on the other hand, are scene-agnostic and are not influenced by the environment's state. They do have a low signal-to-noise ratio during lower accelerations, though. Cameras have typical frame rates of about 100Hz, while IMUs can operate at about 1 kHz. All in all, cameras and IMUs complement each other and can jointly perform relatively precise state estimation [2].

Rovio ("Robust Visual Inertial Odometry") [6], developed by Bloesch et al., is one example of a well-known implementation of VIO pipelines. It uses an Extended Kalman-Filter, one of the three major VIO paradigms (Filtering, Fixed-lag-smoothing, and Full-smoothing). Rovio directly uses pixel intensity errors as innovation terms during the EKF's update step. Contrary to other VIO implementations, Rovio uses a robot-centric approach. The 3D position of tracked batches is estimated in the current camera frame, eliminating the need for prior calibration.

Rovio supports external pose inputs relative to other coordinate frames than Rovio's origin, which results in the filter trying to co-estimate the drift of its origin relative to a static inertia frame. This feature can be used to mitigate drift effects using external measurements from GPS, anchors, or other systems.

2.3 Motion Planning for Omnidirectional Micro Aerial Vehicles (MAV)

Omnidirectional drones have the ability to control their rotation around all axes statically. This extends their ability to observe their environment and interact with objects massively since they are no longer constrained to hovering in an upright attitude.

Pantic et al. worked on novel methods to plan trajectories for omnidirectional drones supposed to fly while interacting with surfaces in their recent work [1]. Their framework also supports the approach of the surface and in-contact-flight with & without applying forces onto the wall. Their method allows flexible resolutions in the mesh representation, making this method suitable for large-scale applications. Their approach is to map a given 3D mesh the drone should interact with to a 2D mesh while maintaining the topological equivalence. This allows them to generate acceleration fields as Riemannian Motion Policies that, if followed, lead to the drone following the desired trajectory.

Their approach has the disadvantage of the joystick inputs being interpreted in the 2D subspace. Due to the mesh's homeomorphic mapping from two to three dimensions, a straight line in the 2D mesh does not have to translate to a straight line in the 3D mesh. However, the joystick input exerts commands in the 2D mesh, making it hard for humans to use since pushing the joystick forward on the controller might result in a complex path along the 3D mesh.

2.4 Spatial Anchors

Services such as Microsoft's Azure Spatial Anchors [7] or Google's Cloud Anchors [8] allow devices to create anchors coupled to visual features in the device's environment. An anchor is only an abstract, named object accompanied by data, which describes the anchor's environment and a defined origin in this spatial data. The same (and other) devices can query the anchors' pose by providing information about the environment and can retrieve the relative transform between the camera and the anchor's pose. Multiple devices can share coordinate systems by querying the same anchor.

The precision of an anchor depends on the quality of the device's pose estimation and feature extraction. Devices such as the Microsoft HoloLens 2 are equipped with high-quality depth cameras and IMUs, which enable it to track the device's pose very accurately, which in turn allows high-quality anchors with a low covariance to be created. High-quality anchors mean that the point cloud accurately and robustly represents the real-world environment. Other devices can benefit from these good anchors by receiving precise and accurate poses.

In the context of robot-human interaction, this means that a device such as the HoloLens 2 can help the drone by providing precise virtual anchors, which the drone can use to estimate its own state accurately.

Azure Spatial Anchors (ASA) is a Cloud Service offered by Microsoft within the MR Cloud Services. ASA can create Spatial Anchors, which are defined relative to real-world structures as conceptually described in section 2.4. Strictly speaking, they are defined relative to a point cloud derived from visual features from multiple monochromatic images and the poses of the camera when the pictures were taken. Once an anchor is created, it can be queried from other devices optionally with large time differences. This effectively allows MR devices to co-locate in a shared environment and build time-persistent MR experiences.

2.4.1 Background

ASA uses a Structure from Motion [9] pipeline to create sparse point clouds with additional feature descriptors. The major difference between ASA and typical SfM pipelines is that only the feature extraction happens locally on the users' devices while finding correspondences and jointly estimating their 3D position happens in the cloud. This has the benefit of anonymizing data before it is sent to the cloud. However, it has been shown that through applying an inverse SfM process, the original images can be generated from the sparse point clouds [10].

2.4.2 ASA Ros Wrapper & ASA Linux SDK

Microsoft Research released an ASA Linux SDK [11] in 2020, which targets Ubuntu 18.04 and 20.04. The Linux SDK allows developers to use the ASA service from ubuntu computers, similarly to Android and iOS phones can do that already through their respective SDKs. Additionally, Microsoft Research released a ROS wrapper for this Linux SDK, making it easy to integrate the service in robotic projects [12]. The ROS wrapper starts a ROS node, which requires three things: Undistorted camera images, corresponding camera intrinsics, and the camera's pose. It accesses

the first two by subscribing to the configured ROS topics, while the camera pose is retrieved using ROS' TF and a configured camera-frame-name.

After starting the ROS node, an ASA session is created. The node synchronizes the images and camera intrinsics messages using an approximate time synchronization policy, fetches the camera's pose when the camera took the images and submits the frame to the ASA Linux SDK. The Linux SDK is currently not open source and has to be viewed as a black box. The Linux SDK performs feature extraction and sends the camera's pose and the feature descriptors to the cloud, accumulating the data into a local point cloud and returns status updates. The session can create anchors as soon as it collected sufficient data. The ASA wrapper offers services to create and query anchors. It publishes messages whenever anchors were created or queried with additional information such as the request's result or the anchor's pose relative to the camera.

Anchors can be created and queried from different devices with different operating systems running their ASA SDK versions. This is important because it allows robots to localize anchors known to MR devices, effectively sharing coordinate systems. This allows robots and MR devices to communicate spatial data and for a more intuitive human-machine interaction. Since ASA is a cloud-based service, all of this comes with relatively affordable computational costs.

Lastly, every successful anchor query process improves the sparse point cloud in the cloud, making anchors more stable the more they are used.

2.5 MR-based Human-Robot Interaction

MR (MR) is the concept of merging the real and virtual world. MR devices can project realistic 3D holograms into the real world, which are perceived as (more or less) real objects by humans. Many MR devices allow humans to intuitively interact with holograms by utilizing hand tracking or other input methods. The technology allows humans to interact with data and machines the same way humans interact with each other in their daily lives: much more intuitive and human-like. Furthermore, using MR interfaces in robot-human collaboration use-cases results in higher performance than 2D screens, as Gadre et al. found in their study [13]. Benedict et al. showed that MR gesture control has a very steep learning curve as a very brief training duration suffices to see rapid performance improvements [14].

A repeatedly upcoming issue in MR-based robot control is the co-location between the robots and the MR devices. The MR device has to know where the robot is relative to itself to overlay the robot or its environment with helpful holograms. This problem can be re-formulated to the requirement that the MR device and the robot know of a shared coordinate system to share spatial information. Without that, spatial information is almost useless. This goal can be achieved in many different ways. Amongst them are marker-based solutions, external tracking cameras, direct pose estimation, pre-shared map-based localization, or spatial anchors.

While marker-based solutions tend to be very precise, they have the disadvantage of distributing markers before the operations can start. Depending on the use case, even multiple markers might be required to cover the area sufficiently. Furthermore, marker-based solutions need the marker to be visible at least once to all devices. Time-persistent applications can only be built with makers if the markers are persistent as well.

External tracking systems have the obvious disadvantage of not being mobile. The preparation time and coverable area are even worse compared to marker-based solutions. But if they are correctly configured, their precision is unmatched. Another alternative to anchors is direct pose estimation, where the video feed from one MR device is used to estimate the drone's pose in its own frame directly. While the research community is extremely active in this area [15, 16, 17, 18], this approach might not work robustly enough with non-static objects, such as robots that change their form factor by moving joints.

A broadly applicable solution is spatial anchors, which benefit from not requiring any preparation while being less precise compared to the alternatives discussed above (see section 5). By letting all involved MR devices and robots localize the same anchors, they can find common coordinate frames and then communicate information accordingly. A robot can send its pose relative to a shared anchor to the HoloLens 2, which can visualize the drone's state using immersive holograms and command the robot. Instead of using 2D screens, human operators can use more intuitive gestures such as pointing and pulling to command the robot. This is especially useful if the drone's planner and controller operate in mapped subspaces, where a subspace-line does not necessarily correspond to a line in the 3D space [1].

Chapter 3

Concept

The goal is to utilize a cloud-based spatial anchor service as an integral part of both state estimation and human-robot collaboration. This can be achieved using a Spatial Anchors platform as a central architecture component of the robotic system.

Fig. 3.1 is an abstract depiction of this concept. All devices of the robotic system, such as the Drone (3) and the HoloLens 2 (2), are connected to the Spatial Anchor Cloud (1). This connection means that these devices can create and query anchors and use this data however they want to. The HoloLens 2 primarily contributes by uploading high-quality spatial data, while the drone mostly queries the cloud service to retrieve these spatially static anchors to improve its VIO state-estimation. The HoloLens 2 also knows the drone's relative pose by localizing the same anchors, which allows the HoloLens 2 to project holograms (visualized in orange) directly onto the drone. The shared coordinate system allows that and enables the human to control the drone's actions using holograms, improving the human-robot interaction. To do that, the drone and the HoloLens 2 additionally communicate via the Robot Operating System (ROS), to manage the drone's interaction with the real world structure (4), which may also be overlayed with holograms.

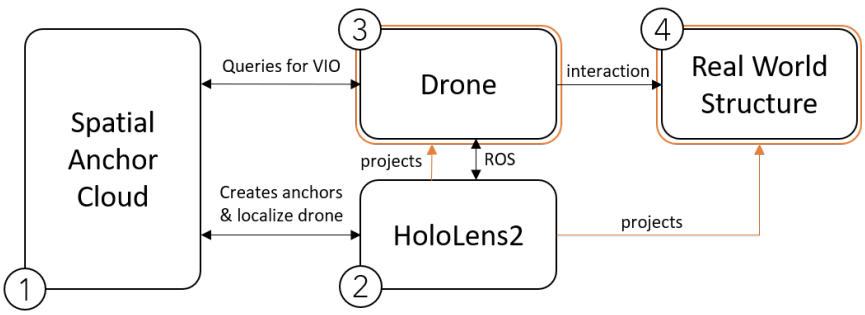

Figure 3.1: Concept

This concept allows for a more intuitive human-robot interaction while possibly increasing the drone's state-estimation precision by utilizing a cloud-based service and thus minimizing the drone's onboard computational costs.

Three processes are running in parallel:

1. All devices: Joint anchor creation (filling the Spatial Anchor Cloud)

2. Robot: VIO improvements using the Spatial Anchor Cloud

3. Robot & HoloLens 2: Communicating their spatial states relative to anchors

3.1 Anchor Creation

Before the other processes can start, the spatial data has to be created first. While all devices can contribute to the point cloud, some excel more than others. The HoloLens 2 has high-quality tracking capabilities and can create high-quality point clouds in the spatial anchor cloud service. Thus, the device mainly uploads data to the cloud. Robots can also create anchors, but in the case of drones, they cannot do that at the same level of precision as the HoloLens 2 does.

Nevertheless, all devices create detailed point clouds and spatial anchors together. All connected devices can use this data for their own purposes.

3.2 VIO Improvements

As discussed in section 2.2, VIO pipelines suffer from drift, which lies within the magnitude of about 1% of the trajectory length. Clearly, an error of 1m after a trajectory length of 100m is far too large for many use-cases.

Estimating the state estimators drift between the initial world frame and the frame at time t allows it to be compensated, which requires some external pose update. This can be achieved using spatial anchors by creating an anchor (A) at an arbitrary known location at the beginning of a trajectory, where the drift is still neglectable.

Later in the trajectory, when the state-estimator drifted by an unknown translation and rotation, the robot can query the previously created anchor and compare the assumed, known homogeneous transformation between the initial and anchor query result's transformation between the two frames. Using these two homogeneous transformations, the transform between the world frames at these two time-stamps can be derived. Expressing the drone's pose in the drift compensated world frame represents the pose update sent to the state-estimator, which can fix its current estimate. This process can be repeated over and over again to compensate for the drift error continuously. For a more thorough mathematical description, see section 4.1.

3.3 Relative Odometry Sharing

The third and last concept is straightforward. The drone sends its current pose relative to an anchor and the ID of the used anchor to the MR device. The only requirement is that both the MR device and drone must know the same anchor. If this assumption is met (which usually is the case), the MR drone can interpret the received odometry however it fits the use case.

This might include visualizing the drone's state using holograms to help the human operator efficiently and allowing the human to control the drone using other holograms, eliminating the need for conventional controllers. Additionally, the HoloLens 2 can show information such as the drone's planned trajectory, the 3D mesh of the surface it is interacting with, interaction parameters (force, etc.), and so forth.

This makes a Mixed Reality app a good solution to the human-robot-interaction problem introduced in Sect. 1, where modern trajectory planners are very advanced and possibly unintuitive to use. By having a shared coordinate system between the devices, the HoloLens 2 allows the human to intuitively control the drone by allowing them to input their commands with gestures in three dimensions instead of 2D joysticks.

Chapter 4

Implementation

This chapter describes the architecture applied to reach the outlined goal. The implementation can be separated into two parts. The first is performing drift compensation for the VIO state-estimator. The second is the MR-based control for an omnidirectional drone. Since this research is about a unified platform for both of these problems, the architecture will first be explained on a high level showing how all components come together.

Fig. 4.1 shows a high-level overview of the architecture. First of all, there is a separation between the real and the "virtual" world. In the real world, there are the drone with its cameras, IMU & possibly other sensors (I), the human (VI), holograms (V) created by the HoloLens 2 (IV), and also Spatial Anchors indicated by blue crosses. The holograms (orange overlays) are projected onto the drone and other real-world objects that the drone is attempting to interact with. The HoloLens 2 uses Spatial Anchors created by the Azure Cloud Service (III) to co-localize itself with the drone. At the same time, the drone uses these anchors to improve its own state estimation. It does that by sensing the real world using its cameras and IMU, sending the data to the state-estimator (II), which queries the Azure Cloud Service

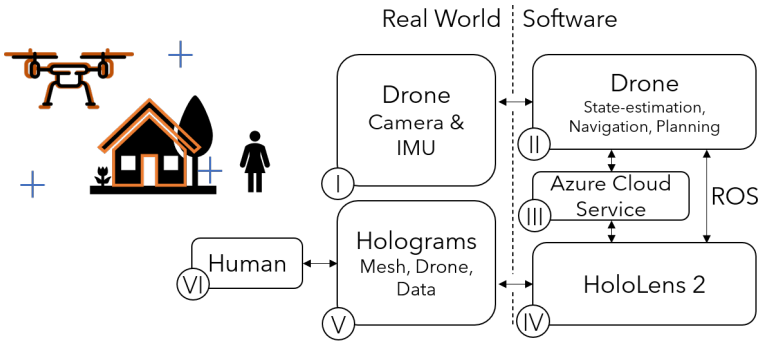

Figure 4.1: System overview

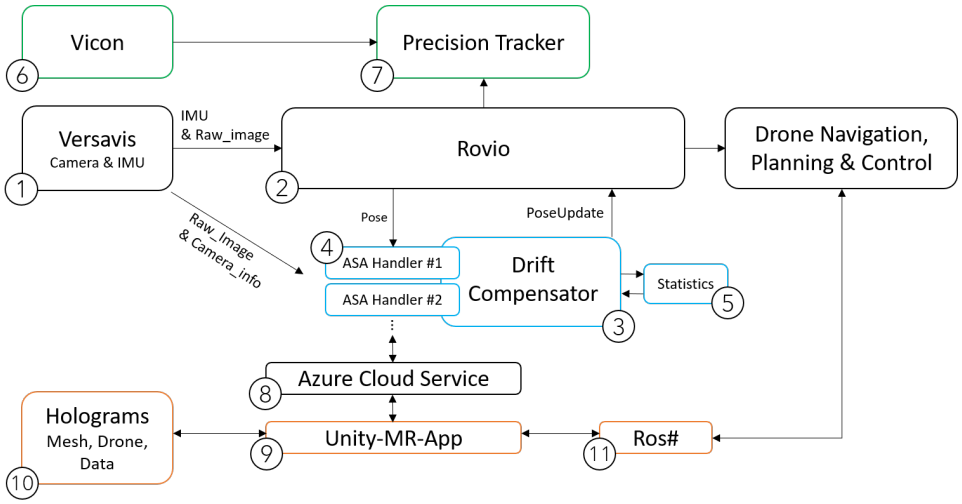

Figure 4.2: System architecture

for an improved state-estimation.

The following two sections describe the detailed architecture separately. First, the implementation of the VIO state-estimation drift compensation is explained in section 4.1, and then the MR side of the system is explained in section 4.2.

4.1 VIO Drift Compensation Using Spatial Anchors

The proposed system was implemented [1] using the Robot Operating System (ROS) [19]. More information and tutorials about ROS can be found here [20]. Figure 4.2 shows a simplified version of the implemented architecture. The black nodes (1 & 2) represent the bare VIO system similarly to its current use. The blue nodes (3-5) are the nodes responsible for executing the drift compensation, and finally, the green nodes (6 & 7) are only there for the performance analysis. Nodes 1, 2, and 6 are out-of-the-box components from previous work, and the rest was developed for this research. Nodes 8-11 are part of the MR implementation and are described in section 4.2.

The individual nodes have the following responsibilities:

1. VersaVIS [21]: VersaVIS is a hardware board capable of synchronizing IMU measurements and camera images very precisely.
2. Rovio [6]: A VIO library based on an Extended Kalman Filter. It takes IMU and Images as an input and outputs a 6 DOF pose estimate.
3. Drift compensator: The node responsible for compensating the drift of Rovio (2).

[1] OpenSource: `https://github.com/EricVoll/vio_drift_comp_using_ASA`

4. ASA Handler: These nodes (multiple possible) handle the Azure Spatial Anchors sessions and communicate with the drift compensator (3).
5. Statistics: Provides Co-variance estimates for every ASA anchor.
6. Vicon: A tracking system providing high precision ground truth information.
7. Precision Tracker: A data collection node for performance analysis post-flight.

VersaVIS (1) outputs synchronized images and IMU messages and publishes them via ROS. Rovio (2) uses the IMU messages and camera images to estimate the camera's current 6 DOF pose. This process suffers from drift due to many different reasons, for example, but not limited to, the low signal-to-noise ratio in the IMU measurements for Yaw rotations, impacts resulting in large IMU spikes, strong lighting changes or movement in front of the camera.

The drift compensator (3) subscribes to Rovio's odometry output and starts to estimate Rovio's drift. It does that by starting one or more instances of the ASA handler nodes (4). Every ASA handler operates independently. After an ASA Handler (4) started, it subscribes to Rovio's Odometry output and to VersaVIS' images and uses that to start an ASA session as described in section 2.4.2. As soon as the ASA sessions are ready to create anchors (which usually takes about 15 to 20 seconds), they create anchors at known locations (usually the origin of the current world coordinate frame) and immediately start to query the same anchors, after resetting their ASA session to clear the cache. For the rest of the mission, the ASA handler nodes continue to track their anchors, report the anchors' locations to the drift compensator (3), reset themselves, and start the loop again.

The drift compensator (3) also creates a covariance estimator (5) instance for every ASA handler created. The drift compensator forwards the covariance of Rovio's last odometry message and queried anchor-locations to the covariance estimator (5), which can calculate a covariance matrix for every pose update calculated by the drift compensator. Since both the ASA handler and Rovio contain uncertainties, it is essential to respect the uncertainty in the pose update's calculations.

Following some rules depending on the configuration (described in section **??**), the drift compensator estimates Rovio's drift, calculates its estimated drift compensated pose, and sends a pose update to Rovio, which uses the update in its next filter update to estimate its world frame drift. It is important to note that this whole process repeats and only stops when the feed of input images and IMU messages stops.

Finally, the precision tracker subscribes to Rovio's odometry and the Vicon tracker's output and logs time-synchronized 6 DOF poses. This data will later be used to analyze the performance of individual configurations and can be found in section 5.

4.1.1 Coordinate Frames, Assumptions, and Notation

A homogeneous transformation $T_{AB} \in \mathbb{R}^{4x4}$ describes a transformation of a vector from frame B to frame A. All numbers used in superscripts specify the timestamp of the symbol (e.g. W^1, $t = 1$) and could alternatively be written as $W^{t=1}$. If a symbol has a bar over it (e.g., $\bar{D}^1$), it is drift compensated. In figure 4.3 dotted lines are used for frames of the previous time-step. W^1 is the drifted version of W^0 at time $t = 1$. This drift may include rotations and translations around all

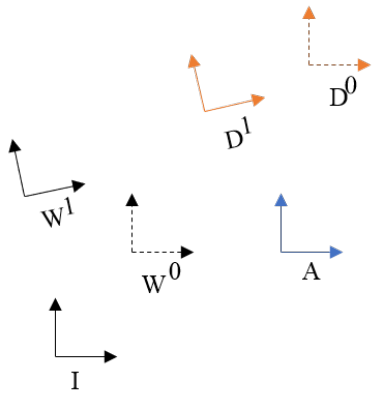

Figure 4.3: Basic coordinate frames

axes. The frame A is the spatial anchor, which is assumed to not suffer from drift if handled correctly. The re-querying process does suffer from imprecision and thus has a certain variance attributed to it. A per frame description can be found in table 4.1.

It is important to state that in this thought model, the frames D^t do not drift relative to their corresponding world frame W^T, but the world frames W^T themselves drift relative to I. The frame A is assumed to be stationary since it is defined by visual features of the real world, which is assumed to be stationary.

Table 4.1: Frame description

Frame name	Description	Drift-assumption
W^t	Rovio's world frame at time t	Drifts
D^t	The drone's frame at time t	Drifts
A	The ASA frame	Stationary, with inprecisions
I	Fixed frame	No drift

4.1.2 Derivation of the Pose Update

The drift can be compensated by placing an Azure Spatial Anchor at a known location and utilizing this knowledge about the anchor's location during the system's run-time by re-querying the anchor constantly and using the difference between the queried position to the initial anchor position.

Mathematically, ASA returns the homogeneous transformation T_{W^tA}. While creating the anchor, we can define the anchor's location and orientation, which will be called T_{W^0A}. The frame W^0 is the frame where the Rovio state estimator is

initialized and will be handled as the original undrifted world frame. The world frame suffers from drift due to Rovio's inner workings.

If the anchor is queried at $t = 1$, ASA returns a transformation T_{W^1A} (the querying process is assumed to finished instantaneously). We know that the transformation T_{W^1A} should be equal to T_{W^0A}.

$$T_{W^0A} = \underbrace{T_{W^0W^1}}_{\text{drift from } 0 \to 1} * T_{W^1A}$$

$$\begin{aligned} T_{W^0W^1} &= T_{W^0A} * T_{W^1A}^{-1} \\ &= T_{W^0A} * T_{AW^1} \\ &= T_{drift^{0\to1}} \end{aligned}$$

This means that we can use the transformation $T_{W^0W^1}$ to transform frames/vectors defined in W^1 to W^0 which we can use to calculate the drone's position in the drift-compensated frame by calculating:

$$\begin{aligned} \bar{T}^1_{W^0D^1} &= T_{W^0W^1} * T_{W^1D^1} \\ {}_{W^0}\bar{r}^1_{W^0D^1} &= pos(\bar{T}^1_{W^0D^1}) \\ \bar{R}^1_{W^0D^1} &= rot(\bar{T}^1_{W^0D^1}) \\ \bar{q}^1_{W^0D^1} &= quat(\bar{R}^1_{W^0D^1}) \end{aligned}$$

The pose $\begin{pmatrix} {}_{W^0}\bar{r}^1_{W^0D^1} \\ \bar{q}^1_{W^0D^1} \end{pmatrix} \in \mathbb{R}^7$ is used as a pose update for rovio. The functions $pos(T)$ and $rot(T)$ extract the position and rotation parts of the homogeneous transformation T and $quat(R)$ converts the rotation matrix R into a quaternion.

4.1.3 Derivation of the Pose Update's Covariance

For simplicity, the derivation of the covariance of pose updates is calculated in the translation-only case.

Step 1: Estimate the drift vector

$$\boldsymbol{r}_{W^0W^t} = \boldsymbol{r}_{IW^t} - \boldsymbol{r}_{IW^0} \tag{4.1}$$

$$\begin{aligned} \boldsymbol{r}_{IA^0} &= \boldsymbol{r}_{IW^0} + \boldsymbol{r}_{W^0A^0} \\ \boldsymbol{r}_{IA^t} &= \boldsymbol{r}_{IW^t} + \boldsymbol{r}_{W^tA^t} \\ \boldsymbol{r}_{IA^0} &= \boldsymbol{r}_{IA^t} \end{aligned}$$

$$\boldsymbol{r}_{IW^t} = \boldsymbol{r}_{IW^0} + \boldsymbol{r}_{W^0A^0} - \boldsymbol{r}_{W^tA^t} \tag{4.2}$$

Using 4.2 $\to$ 4.1

$$\boldsymbol{r}_{W^0W^t} = \cancel{\boldsymbol{r}_{IW^0}} + \boldsymbol{r}_{W^0A^0} - \boldsymbol{r}_{W^tA^t} - \cancel{\boldsymbol{r}_{IW^0}}$$

$$\boxed{\boldsymbol{r}_{W^0W^t} = \boldsymbol{r}_{W^0A^0} - \boldsymbol{r}_{W^tA^t}} \tag{4.3}$$

Step 2: Update the pose

$$\boldsymbol{r}_{W^tD^t} = \boldsymbol{r}_{W^{t-1}D^{t-1}} + \boldsymbol{r}_{D^{t-1}D^t} + \boldsymbol{r}_{W^{t-1}W^t}$$

Where $\boldsymbol{r}_{W^tD^t}$ is the position of the drone reported by Rovio at time t, which suffers from drift, $\boldsymbol{r}_{D^{t-1}D^t}$ is the movement the drone really made between timesteps $t-1$ and t and $\boldsymbol{r}_{W^{t-1}W^t}$ is the drift of the world frame between $t-1$ and t. $\boldsymbol{r}_{W^{t-1}D^{t-1}}$ is the position Rovio reported at time $t-1$. To get the drift-compensated position $\bar{\boldsymbol{r}}_{W^0D^t}$, the drift has to be subtracted.

$\bar{\boldsymbol{r}}_{W^0D^t} = \boldsymbol{r}_{W^tD^t} - \boldsymbol{r}_{W^0W^t}$

Step 3: Derive the variance of our estimate

$$Var(\bar{\boldsymbol{r}}_{W^0D^t}) = Var(\boldsymbol{r}_{W^tD^t} - \boldsymbol{r}_{W^0W^t})$$
$$Using: Var(X-Y) = Var(Z) = Cov(Z,Z) = Var(X) + Var(Y) - 2 * Cov(X,Y)$$
$$Var(\bar{\boldsymbol{r}}_{W^0D^t}) = \underbrace{Var(\boldsymbol{r}_{W^tD^t})}_{\text{from Rovio}} + \underbrace{Var(\boldsymbol{r}_{W^0W^t})}_{\text{Drift estimate cov.}} - 2 * \underbrace{Cov(\boldsymbol{r}_{W^0W^t}, \boldsymbol{r}_{W^tD^t})}_{?}$$

$$Cov(\boldsymbol{r}_{W^0W^t}, \boldsymbol{r}_{W^tD^t}) = E[\boldsymbol{r}_{W^tD^t} * \boldsymbol{r}_{W^0W^t}] - \mu_{\boldsymbol{r}_{W^0W^t}} * \mu_{\boldsymbol{r}_{W^tD^t}}$$
$$= E[(\boldsymbol{r}_{W^{t-1}D^{t-1}} + \boldsymbol{r}_{D^{t-1}D^t} + \boldsymbol{r}_{W^{t-1}W^t}) * \boldsymbol{r}_{W^0W^t}] - \mu_{\boldsymbol{r}_{W^0W^t}} * \mu_{\boldsymbol{r}_{W^tD^t}}$$
$$= \underbrace{E[\boldsymbol{r}_{W^{t-1}D^{t-1}} * \boldsymbol{r}_{W^0W^t}]}_{=0 \text{ (uncorrelated)}} + \underbrace{E[\boldsymbol{r}_{D^{t-1}D^t} * \boldsymbol{r}_{W^0W^t}]}_{\approx 0 \text{ (maybe small corr.)}} + \underbrace{E[\boldsymbol{r}_{W^{t-1}W^t} * \boldsymbol{r}_{W^0W^t}]}_{=0 \text{ (unbiased drift assumed)}} - \underbrace{\mu_{\boldsymbol{r}_{W^0W^t}} * \mu_{\boldsymbol{r}_{W^tD^t}}}_{\text{small}}$$

$$\implies Cov(\boldsymbol{r}_{W^0W^t}, \boldsymbol{r}_{W^tD^t}) \approx 0$$

The drone's position at $t-1$ ($\boldsymbol{r}_{W^{t-1}D^{t-1}}$) does not correlate with the drift vector, because the drift between time $t=0$ and $t=t$ happens after Rovio estimates the position at $t-1$ and is therefore not influenced by the absolute position. The movement delta $\boldsymbol{r}_{D^{t-1}D^t}$, on the other hand, might have a small influence on the drift amount, which I cannot specify further at the moment. The estimated value of the two drifts between different timestamps is zero because we assume an unbiased drift direction at every timestamp, which means that the expected value should be 0.

Since the anchor querying process's uncertainty is relatively high (about a magnitude larger), we can safely ignore the covariance between the drone's reported pose and the drift between two timesteps. Thus, the variance of the pose update sent to Rovio will be implemented as: $Var(\bar{\boldsymbol{r}}_{W^0D^t}) \approx Var(\boldsymbol{r}_{W^tD^t}) + Var(\boldsymbol{r}_{W^0W^t})$.

The last unknown is the variance of the drift $\boldsymbol{r}_{W^0W^t}$, which I will derive from the anchor observations.

$$\begin{aligned} Var(\boldsymbol{r}_{W^0W^t}) &= Var(\boldsymbol{r}_{W^0A^0} - \boldsymbol{r}_{W^tA^t}) \\ &= Var(\boldsymbol{r}_{W^0A^0}) + Var(\boldsymbol{r}_{W^tA^t}) - 2 * \underbrace{Cov(\boldsymbol{r}_{W^0A^0}, \boldsymbol{r}_{W^tA^t})}_{=0 \text{ (uncorrelated)}} \end{aligned}$$

The query-operations take place in de-coupled sessions are thus uncorrelated to each other. Thus the resulting variance calculation for our pose-update $\bar{\boldsymbol{r}}_{W^0W^t}$ is:

$$Var(\bar{\boldsymbol{r}}_{W^0D^t}) = Var(\boldsymbol{r}_{W^tD^t}) + \underbrace{Var(\boldsymbol{r}_{W^0A^0})}_{=0 \text{ if anchor created by drone}} + Var(\boldsymbol{r}_{W^tA^t}) \tag{4.4}$$

Where $\boldsymbol{r}_{W^tD^t}$ is the last pose reported by Rovio, $\boldsymbol{r}_{W^0A^0}$ is the anchor's position in the undrifted world frame at $t = 0$ and $\boldsymbol{r}_{W^tA^t}$ is the anchor's position in the drifted world frame W^t. If the system created the anchor at time $t = 0$ itself, its attributed variance is 0 because its precise location is known. If the first anchor observation at $t = 0$ is not precisely known (e.g., if another device created the anchor and forwarded its anchor id), this variance is not known and hard to estimate since an empirical covariance estimate cannot be calculated from one sample. In this case a static variance from empirical tests can be used for the value $Var(\boldsymbol{r}_{W^0A^0})$.

4.2 Mixed Reality Implementation

The MR side of this system [2,3] is also shown in Fig. 4.2 (orange nodes 9 - 11). Their responsibilities are:

9. Unity-MR-App: An application developed in Unity. Contains the application logic on the MR side of things.
10. Holograms: Holograms created by the HoloLens 2 should help humans interact with the drone more intuitively.
11. Ros#: A C# client for ROS, which allows a Unity application to start Ros-nodes and exchange messages with nodes hosted on the drone.

When starting up the Unit MR App, it immediately starts an ASA session and creates a spatial anchor. It also connects to the drone's Rosmaster by using Ros# [22], specifically a fork of the repository adjusted to be executable on Universal Windows Platforms (UWP) [23], which is the build target for Unity HoloLens 2 Apps.

As soon as the anchor is successfully created, the Unity App commands the drone to use the anchor for its localization. As soon as the drone successfully queried the anchor, it publishes its odometry relative to it. The Unity app subscribes to this topic and can then place a GameObject (An abstract object used in Unity for objects in the app, which may or may not have physical appearances) relative to its known anchor at the pose the drone published. Since these anchors represent shared coordinate systems, the virtual GameObject (Hologram) will overlay the real world's drone.

It is important to note that Unity works with left handed coordinate systems where the y-axis points up and therefore also creates anchors this way. This has to be respected in the query-process of the drone, which queries the anchor using right handed coordinate frames. While the fact that the anchor was created in a left handed coordinate frame gets lost somewhere in ASA's process, the y-axis is still pointing upwards, such that the anchor frame has to be rotated for the drone to use it correctly.

[2] OpenSource implementation: `https://github.com/EricVoll/ethz_asl_semester_project`

[3] Video: `https://youtu.be/SxmkRreG5j8`

The HoloLens 2 can also subscribe to all other topics, allowing it to visualize the drone's planned trajectory, state (such as battery level, interaction forces, pose, etc.), and much more. It also allows the human to place an object in MR and then publish this object's pose relative to an anchor. The drone's controller listens to this topic and can then execute the command. This allows the human to control the drone intuitively.

The current implementation allows the drone to publish its known 3D mesh, which it will use to plan its motions. The Unity App receives the mesh via ROS and visualizes it. It then allows the human to add waypoints on the mesh by tapping the mesh, the coordinates of which can be sent to the drone via ROS, which follows generates a trajectory visiting all the waypoints. Sadly the available time for this project ran out, such that the commanding of the real drone could not be tested anymore and was limited to simulations.

Chapter 5

Analysis

This chapter summarizes the analysis performed on the developed methods. First, the general method and the used data sets are detailed (Sect. 5.1), followed by precision and robustness tests for the cloud-based SfM pipeline (Sect. 5.2 and 5.3). The performed robustness tests include scenarios such as limited internet connection and artificial drift. In the end, the performances of different configurations are analyzed (Sect. 5.4), before a conclusion is drawn.

5.1 Data, Trajectory Alignment and Error Metrics

To create repeatable results, nine different rosbags were recorded. Table 5.1 describes the content, characteristics, and length of each rosbag. All configurations were tested on all rosbags. Since each rosbag recorded a different kind of motion, analyzing the performance across all rosbags might reveal certain types of movements the default state-estimator cannot handle well or scenarios where the drift compensation can provide great support.

During all tests, the ground truth data was compared to the state estimator's output. Before calculating error metrics between the ground truth and the estimate, the trajectories were aligned following the tutorial of Zhang et al [24]. The alignment was performed by applying a transformation $T \in SE(3)$ to one of the trajectories. This means that no scaling or morphing was performed. Two methods were used to find the transformation T:

1. Position error minimization over the whole trajectory
2. Alignment using the first frame of both trajectories

While the first alignment strategy will return smaller error metrics (by definition), its interpretation is less intuitive. The second method directly highlights the effects of drift during the runtime of the system. In the following analysis, both the Absolute Trajectory Error (ATE) and Relative Error (RE) will be used (see [24]).

Table 5.1: Data description

Nr.	Size	Characteristics
1	180s, 108m	Smooth movement and rotations around all axes.
2	128s, 79m	Translation only along x,y,z & camera facing in x direction.
3	132s, 88m	Translation only along x,y,z & camera facing in downwards.
4	138s, 51m	Bag1 + Shaking in increasing strength
5	121s, 58m	Bag1 + Impact with surfaces for IMU spikes
6	140s, 50m	Bag1 + Lighting changes in environment
7	121s, 44m	Bag1 + Person walking through the scene twice
8	139s, 141m	Bag1 + Facing towards wall, then pointing back to anchor
9	99s, 51m	Flat viewing angles to the surface of the anchor
10	217s, 61m	Similar to Bag1 but with an HoloLens 2 anchor
11	202s, 128m	Rough version of Bag10 with an HoloLens 2 anchor

Table 5.2: Rosbag content

Name	Frequ.	Description
IMU	400Hz	Typical IMU measurements (device: ADIS16448)
Image	50 Hz	Distorted images. Resolution: 540 x 720
Groundtruth	100 Hz	6 DOF pose. Precision of around 1 mm

Table 5.3: ASA precision Rosbag 1 (n = 158, values rounded to 4 decimal places)

Statistic	Value
$\bar{\pi}_{t\ query}$	1.20 *sec*
$\bar{\pi}_{drone}$ [m^{1x3}, rad^{1x3}]	[0.0277 −0.0751 −0.0029 −0.0027 0.0015 −0.0164]
σ_{drone} [m^{1x3}, rad^{1x3}]	[0.2085 0.4038 0.0778 0.0104 0.0106 0.0760]

5.2 ASA Precision Analysis

First, the general precision under optimal circumstances was analyzed. This was done by using the ground truth Vicon data as pose inputs for ASA, synchronized with images from the real drone setup. By using Vicon data, it was made sure that no drift errors negatively influence the performance of ASA. An anchor was created at Vicon's frame origin and then repeatedly queried while feeding the data from the rosbags into the system. The returned anchor poses were collected and stored after converting rotations from quaternions to Euler angles. The average query location and standard deviation for all axes were then calculated while also tracking the time between individual queries. The results of which can be found in table 5.3.

The average anchor location was very precise in the z direction. The position's standard deviation was surprisingly large, though, especially along the x *and* y axes with standard deviations of 0.2m and 0.4m, respectively. The different values for x and y are probably the result of a slightly unbalanced view direction in the used dataset, where the camera primarily pointed towards positive x. The angles with a maximum standard deviation of 0.01 rad around the yaw axes, on the other hand, were very precise. The average anchor query operation took 1.2 seconds with a standard deviation of 1.4 seconds and about 1 second as a minimum duration. It is important to note that the duration measurement included the time it took to reset the ASA session and to clear the cache. Furthermore, the ASA node setup was the same as in the VIO drift compensator setup. Namely, it auto-restarted the ASA handler if the ASA session crashed and also restarted the ASA node if a query operation was unsuccessful for more than 8 seconds. Slightly better performance values were achieved with the Rosbags 2, 3, 10 and 11.

Anchors that were created using the HoloLens 2 are slightly better, as shown in table 5.4. An anchor was created using the HoloLens 2 in the same room with the same setup as Rosbag 10 was recorded. Then the same test as previously was started. Two anchors (one created from the HoloLens 2 and one from the Rosbag itself) were continuously queried.

Querying anchors using the ASA ROS node can be done in two ways. A system can either restart the ASA node completely after every successful query or only clear the cache. The average query time $\bar{\pi}_{t\ query}$ is 1.2 *sec* when only clearing the cache, while it is 5.2 *sec* when restarting the ASA node completely. Interestingly the precision advantage of the HoloLens 2 anchor almost disappears when only clearing the cache.

Table 5.4: ASA precision HoloLens2 vs Drone anchor with Rosbag 10

Standard deviation	Value [m^{1x3}, rad^{1x3}]
$\sigma_{drone\ anchor}$	[0.121 0.09 0.062 0.035 0.008 0.077]
$\sigma_{HoloLens2\ anchor}$	[0.051 0.089 0.0370 0.008 0.008 0.05]

5.3 ASA Robustness Analysis

In the Robustness Analysis, the question of whether or not certain movements have large (possibly negative) influences on the performance of the anchor querying process of the ASA pipeline should be answered. During the system's development, it was observed that sometimes an ASA session would be incapable of finding an anchor. In the implemented system, this was treated by resetting the ASA session if no anchor was found for more than 8 seconds. This means that there must be some trajectory error, which results in ASA reaching an unrecoverable state. Three different scenarios were designed to isolate certain performance characteristics of the ASA platform in a controlled manner:

1. Linear velocity drift
2. Angular velocity drift
3. Internet bandwidth tests

1. Linear Velocity Drift
A linear velocity along the x-axis was added to the Vicon ground truth data to simulate a drifting state in a very controlled and repeatable way. The drift velocity started at $0\frac{m}{s}$ and was increased in steps of $0.2\frac{m}{s}$ every time an anchor was successfully found. Each query operation started with the test frame located at the Vicon frame and then drifting away with its respective velocity. The tests were stopped at a drift velocity of $16.0\frac{m}{s}$ because the linear velocity did not seem to impact ASA's ability to find an anchor. Notably, the average anchor query duration was about 2.0 seconds, with a standard deviation of 2.5 seconds, therefore likely taking twice as long as in the optimal case.

2. Angular Velocity Drift
Similar to the previous case, an increasing angular velocity around the world's yaw frame starting at $0\frac{rad}{s}$ was added to the Vicon ground truth frame. After ASA was successfully querying anchors up to an angular drift velocity of $18.4\frac{rad}{s}$ the test was stopped, as similarly to linear velocities, the angular velocity does not seem to influence the ability of ASA to query anchors. Similarly to the linear velocity case, the average query operation duration increased to 1.9 seconds with a standard deviation of 2.6 seconds. Interestingly the average anchor location error in (x, y, z) direction was about ($0.38m, 0.34m, 0.005m$) and the average yaw error was about 0.28 radians, considerably higher than in the optimal case. The standard deviation along the (x, y, z) axes was ($1.12m, 1.08m, 0.11m$) respectively, and around (roll, pitch, yaw) a standard deviation of ($0.01\ rad, 0.01\ rad, 1.65\ rad$) was observed. This means that the orientation of the queried anchor was not usable anymore. These numbers are the result of the complete test, where the same anchor was queried once with each angular velocity drift from $0\frac{m}{s}$ to $18.4\frac{rad}{s}$.

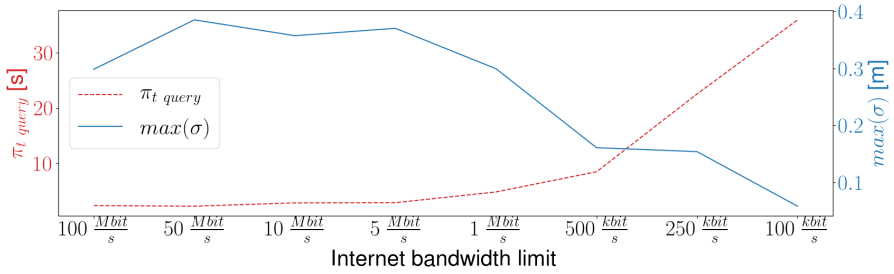

Figure 5.1: Reduced internet bandwidth

It is important to note that these high drift velocities ($16.0\frac{m}{s}$ & $18.4\frac{rad}{s}$) are unlikely to occur in real state-estimators and were merely used as an attempt to isolate possible issues.

3. Internet bandwidth
Since ASA is a cloud service, it is important to ask how reliable the service is in varying internet connection scenarios. By using the tool `wondershaper` the up and download bandwidth was limited to values ranging from 100 $\frac{Mbit}{s}$ down to 100 $\frac{kbit}{s}$. The bandwidth was throttled more and more while analyzing ASA's ability to query anchors, the precision, and duration metrics.

Figure 5.1 shows the results of this analysis. This test indicates that bandwidths down to 5 $\frac{Mbit}{s}$ do not really negatively influence the frequency of anchor queries. Bandwidths lower than 1 $\frac{Mbit}{s}$ result in a massive increase in average query duration and decrease in query frequency. Interestingly the precision does not seem to be influenced by the bandwidth limitations. The dropping maximum standard deviation entries could be explained in two ways: First, the increased query duration means fewer samples to calculate the empirical standard deviation, which could lead to a low standard deviation by luck, although the clear trend somewhat speaks against that. The second answer could be that the ASA Linux SDK reacts to the low bandwidth by prioritizing images with very well-defined visual features, which return more precise results. Interestingly, the average anchor position ("accuracy", not plotted in the figure) also improves with lower bandwidths.

Conclusion on ASA robustness
It seems that linear and angular drift velocities do not change ASA's ability to query anchors but do affect the precision. It is hard to argue why ASA can still find anchors since it is a black box and exact details are not known. A good guess is that ASA is often able to query poses using only one or two frames. Thus, the actual pose of the camera passed to it only influences the anchor's returned pose, but not the ability to find the anchor itself. As far as the tests show, ASA does not seem to use the full trajectory uploaded to the server for the querying process.

With internet bandwidths of 1 $\frac{Mbit}{s}$, very reasonable results can be achieved. In most countries, the internet infrastructure is strong enough to deliver these required speeds. Currently, ASA requires a connection to the public internet, meaning that the bandwidth issue cannot be solved by having a local server. After talking to Microsoft Employees about these findings, another suggestion for the reason why ASA would enter this unrecoverable state was that the service simply failed and either the request never reached the server or the server failed silently. According

to Microsoft this might very well be the case. This investigation nicely highlights the issue of closed source commercial services, where consumers do not have any chance of understanding what is going wrong, which is a huge downside for robotic projects.

Finally, there seems to be high potential in making the precise tracking data of devices such as the HoloLens 2 available to robots for their state estimation when using cloud-based systems. This makes sense given that the HoloLens 2 has depth cameras, and the used drone does not.

5.4 VIO Drift Compensation Analysis

In this section, the proposed drift compensation's performance using cloud-based SfM pipelines is analyzed using the data described in section 5.1. In a system like this, many parameters can be tuned to find a well generalizing optimal configuration. The following list details the tested configurations, briefly explains them, and gives each configuration a name.

1. Default: No external drift compensation applied—only Rovio performing state estimation.

2. asa_{alt}: Calculate the updates at every frame (see sect. 4.1.2) and track the frame of the update instead of Rovio's output for the performance measurements. This configuration bypasses Rovio's filter with the update.

3. $asa_{update\ 1}$: Send pose updates with covariance estimates to Rovio using the `/odometry` topic until Rovio's `world` frame starts to drift. If it drifts, stop and wait for a new anchor to be found. This configuration forces Rovio to accept the pose updates and was added because Rovio seems to have a hard time accepting low-frequency updates. Sending updates has to be stopped as soon as Rovio drifts because otherwise, it would create a positive feedback loop since the updates start to drift as well due to their dependency on Rovio's state.

4. $asa_{update\ 2}$: Send single pose updates with covariance every time an anchor was found using the `/odometry` topic.

5. $asa_{multiple\ alt}$: Method 2 (asa_{alt}) but with multiple anchors being tracked at the same time and then using the anchor with the highest score for the updates. $S(\sigma^2, t_{age}) = \frac{1}{max(\sigma^2)} + e^{-t_{age}} - e^{t_{age}-10}$. This score function values anchors with low covariances and discourages old anchors, since they might have been queried when the drift was different. Anchors older than 10 seconds receive very low scores.

6. $asa_{multiple\ update\ 1}$: Method 3 ($asa_{update\ 1}$) with multiple anchors. The anchor with the highest score is used for the updates to Rovio.

7. $asa_{multiple\ update\ 2}$: Method 4 ($asa_{update\ 2}$) with multiple anchors. All anchors are used for the updates with their respective covariances.

8. $asa_{average}$: A weighted average (weighted by the score function $S(\sigma^2, t_{age})$) of the position of all anchors is used as a basis for the drift estimation.

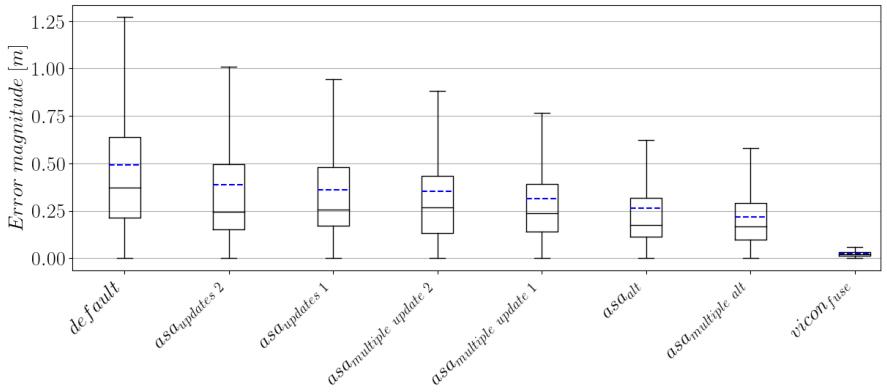

Figure 5.2: Accuracy overview of all configurations [1]

9. $vicon_{fuse}$: Ground truth data was provided to Rovio as odometry updates as a benchmark as the best achievable result. A standard deviation of 1mm was used.

All of these configurations were tested in three modes:

1. Create and use their own spatial anchors
2. Provide previously created anchors
3. Provide an anchor that was created using Vicon ground truth data

Due to time issues, the methods could not be tested with anchors created by the HoloLens 2 yet.

5.4.1 Overall Configuration Performance

Figure 5.2 visualizes an overview of the trajectory analysis result sorted from worst to best. As expected, the *default* configuration performed worst, while the $vicon_{fuse}$ configuration performed best. The default configuration had an average position error of about 1% across all rosbags. The two configurations $asa_{update\ 1}$ and $asa_{update\ 2}$ performed relatively badly due to Rovio not really adjusting its state when receiving updates with such high covariances. While $asa_{multiple\ update\ 1}$ and $asa_{multiple\ update\ 2}$ performed better than their counterparts which only utilized one anchor, their performance is still not as good as the "alt" configurations. $asa_{multiple\ alt}$ superseding asa_{alt} makes sense, since the former can use multiple anchor observations and select the best one, eliminating the damage created by outliers in the anchor detection.

[1] Diagram created with the position error magnitudes measured every 0.1 seconds, aligned using a transform $T \in SE(3)$ on the first frame, with the error data of all rosbags concatenated. The blue line is the arithmetic mean. Diverging trajectories were excluded, and outliers are not visualized.

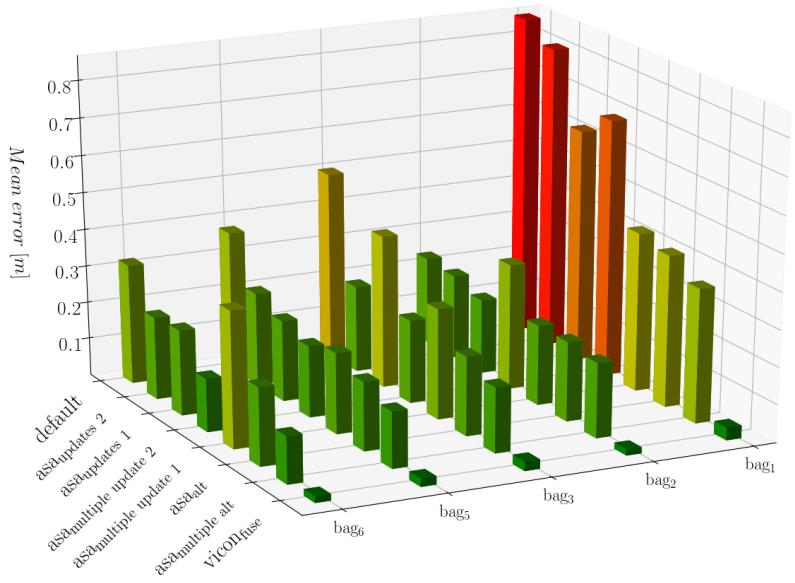

Figure 5.3: Mean error magnitude across the whole rosbag

asa_{alt} performed relatively well but suffered from outliers. Sometimes ASA would return anchors with an error of more than 2m, which results in asa_{alt} over-compensating the drift by a huge amount, which increases the average error. This can be seen in the Fig. 5.2 by the high arithmetic mean value indicated by the blue line. The $asa_{multiple\ alt}$ configuration mostly does not suffer from this issue since multiple anchors were available, and the best anchor was chosen. An obvious outlier would result in a high empiric covariance, which leads to a low score, which results in the anchor not being used.

Overall $asa_{multiple\ alt}$ performed best in the shown error metrics. The used graph is a bit misleading though since all nine rosbags are combined in one box-plot diagram. Fig.,5.3 on the other hand, visualizes the average error of every configuration on every rosbag. Here it is more clear that rosbag 1 was the toughest data-set for all non-diverging configurations. This makes sense since it is also the longest data set with rotations and translations along all axes. All configurations (even $vicon_{fuse}$) failed not to diverge when applied to rosbags 4,7,8, and 9. Interestingly $asa_{multiple\ update\ 2}$ managed to stabilize during rosbag 4 (which included shaking the drone) for a brief duration, while the others diverged during the first wave of shaking.

5.4.2 Characteristics of the Best Configurations

In the following analysis, only the best performing configurations of the two categories (sending updates to Rovio VS bypassing Rovio & directly tracking the update frame) $asa_{multiple\ update\ 2}$ and $asa_{multiple\ alt}$ will be compared to each other while using the default and $vicon_{fuse}$ configurations as baselines.

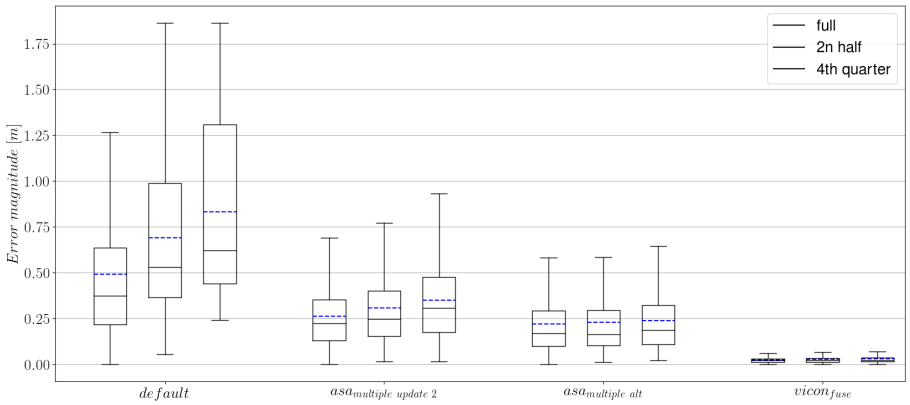

Figure 5.4: Error Magnitude Boxplots of parts of the trajectory across all rosbags

Fig. 5.4 shows three box plots per configuration, where the first one shows an error magnitude summary statistics across all trajectories of all nine rosbags, the second one only includes data from the second halves of the trajectories. Finally, the third only includes the fourth quarter of all trajectories. The reason why this plot is valuable is that it clearly shows the effects of drift. Keep in mind that the trajectory alignment between ground truth data and the trajectory only uses the first frames of both. If an alignment across the whole trajectory were performed, the drift would not be visible as it is now.

Both $asa_{multiple\ update\ 2}$ and $asa_{multiple\ alt}$ are seemingly able to compensate the drift down to an almost constant average error. It is important to keep in mind that the standard deviation of the ASA querying process is about 15cm in x *and* y direction. According to the derivation of the pose update's covariance, the resulting variance should be the sum of Rovio's covariance (which is relatively small) and the empirical covariance estimation. In Fig. 5.4 it can be seen that the error's standard deviation is in that range, indicating that the calculations are correct since the standard deviation of the ground truth data is neglectable.

Inspecting one rosbag on its own delivers further interesting insights. Fig. 5.5 shows the position error magnitude of four configurations applied to the first data set. The best trajectory estimate not using ground truth data (e.g. the best but not $vicon_{fuse}$ since this one is cheating) is drawn in green. Three learnings can be extracted from this figure. First, the drift error in Rovio seems to be periodic. Upon further investigation, the periodicity seems to correlate with the circular movements made during the data recording. It seems that Rovio outputs a slightly scaled position compared to the ground truth Vicon data. This might be why the configurations that send updates to Rovio have a hard time compensating the drift because this (apparent) scale issue mathematically looks like a world drift to the drift compensator, drifting from one side to the other.

The second learning is that the "alt" methods (asa_{alt}, $asa_{multiple\ alt}$) have large jumps in the position estimate since they immediately switch to new anchors. $asa_{multiple\ update\ 2}$ on the other hand, has about the same frequency as the default configuration has since the updates are much smoother due to Rovio's filter. Large jumps could easily be fixed by inserting a PID controller, which controls the state from the current to the desired value or by removing outliers before using the anchor observations. If not taken care of, these large jumps could cause serious issues in

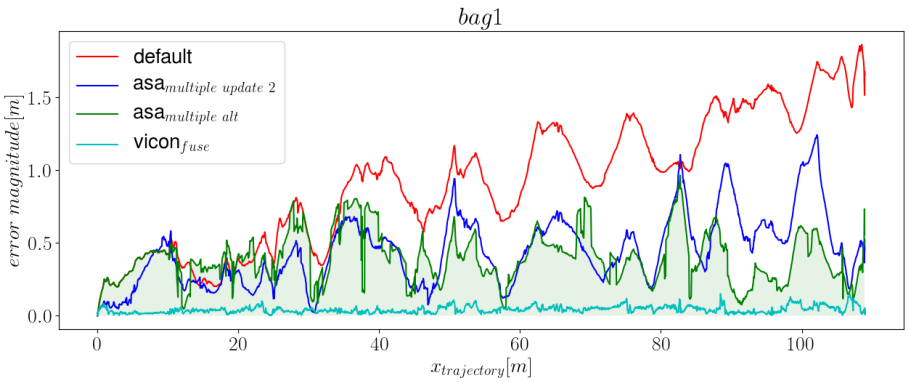

Figure 5.5: Position Error during rosbag 1

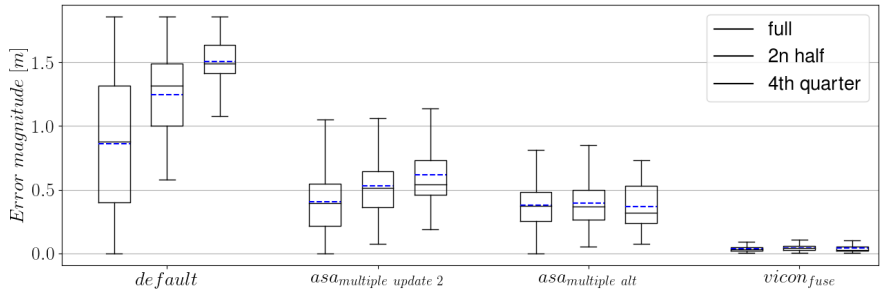

Figure 5.6: Position Error statistics during rosbag 1

the drone's controllers.

The third learning is that even the "alt" methods suffer from Rovio's short-term imprecision and that the frequency of the pose updates is not high enough to compensate the error further. Since the implemented drift compensator always uses Rovio's current pose in one way or another, Rovio's pose estimation will dictate how precise the state estimation is between two updates. This means that increasing the update frequency (frequency of anchor query operations) is an absolute key to lower the trajectory errors further. In the displayed run, the $asa_{multiple\ alt}$ configuration performed 224 successful anchor queries (1.18 Hz), while the $vicon_{fuse}$ configuration updated Rovio with a frequency of about 100Hz and with much higher precision and accuracy.

Fig. 5.6 shows summary statistics over different parts of the trajectory estimate of the four configurations. This plot makes it clear that the majority of the error comes from the later parts in the trajectory and that $asa_{multiple\ alt}$ seems to be able to compensate drift better than $asa_{multiple\ update\ 2}$.

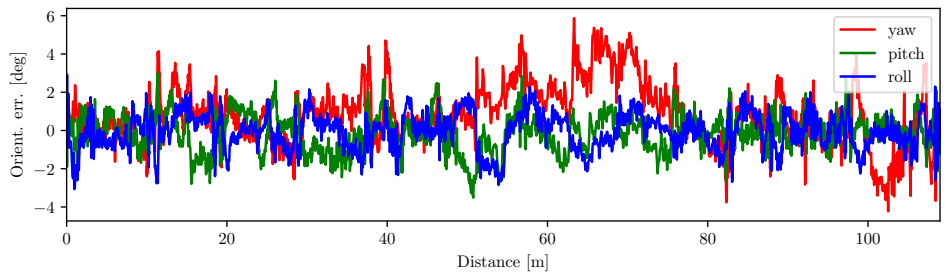

Figure 5.7: Rotation Error $vicon_{fuse}$for bag 1

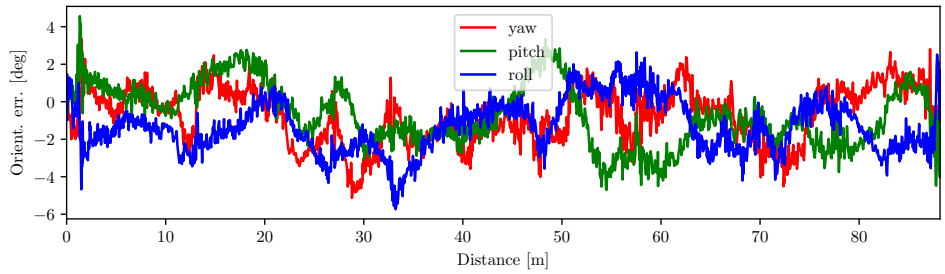

Figure 5.8: Rotation Error $vicon_{fuse}$for bag 3

5.4.3 VIO Drift Compensation Rotation Error

Fig. 5.7 and Fig. 5.8 show the rotation error in roll, pitch and yaw plotted over the trajectory length for two of the recorded rosbags for the $vicon_{fuse}$ configuration. Clearly the error values are far too high in amplitude and frequency, especially considering that the plotted configuration uses ground truth data for the estimation and should therefore be almost perfect, as it was with the position error case. While this is an extremely unsatisfying result, countless hours of trying to find the error in the evaluation process were not enough to fix the issue.

It is likely that the error has its source in one or both of two components: First, the tf reading and recording could be implemented poorly towards time-stamping, although a lot of effort and care was invested here. Second, the trajectory alignment and evaluation might be configured poorly. All configurations appear to have very similar rotation error plots, further indicating that not the estimation but evaluation is wrong.

ASA's rotation precision is relatively good as shown in Sect. 5.2. While the configurations bypassing Rovio (asa_{alt} & $asa_{multiple\ alt}$) also appear to have jumps in the rotation error due to anchor observation outliers, the overall error amplitude should be much lower. Sadly there is no data to back this statement up due to the issues explained above.

5.4.4 The Ambiguity Between Scale and Drift in the Data

Analyzing the raw data coming from Rovio indicates that the used Rovio configuration might have a small-scale issue. All trajectories seem to overshoot the ground truth position data by a factor of about 1.2. The problem with that is that the drift compensator cannot differentiate between a drifted world coordinate frame and a scaled trajectory. From the perspective of the drift compensator, the drone's frame is off by a certain delta ($\Delta p \approx (x, y, z) * 0.2$), which it will compensate, since the theory behind the drift compensator only looks at the pose of the camera and moves it to where it should be. It does and cannot differentiate where the camera's pose error came from. As soon as the drone moves to the opposite side, the scale issue results in an inverted delta, tricking the drift compensator into compensating non-existing drift. Due to the low query frequency and relatively low ASA precision, it cannot keep up with these large deltas looking like drift.

Chapter 6

Conclusion

Cloud-based solutions are likely to significantly transform the field of robotics in the near future. Advancements in the public 5G internet infrastructure will accelerate the research and development of approaches employing cloud computing for challenges currently solved with onboard computers. This not only solves the issue of limited computational power on robots but is also inherently makes the robotic system more accessible to other IoT devices, allowing a better user experience with Mixed Reality interfaces.

This project utilizes a cloud-based Structure from Motion pipeline to perform drift compensation in the drone's state estimator and co-localize a Mixed Reality device with the drone to build an intuitive interface to control the omnidirectional drone along complex 3D meshes. The cloud service used is called Azure Spatial Anchors and allows multiple devices to contribute to one point cloud, thereby leveraging individual devices' capabilities in a system.

The system works as conceptualized in Sect. 3 and indeed does improve the state estimator's performance as shown in Sect. 5. The precision and query frequency of ASA seem to be too low to replace other ground truth data sources such as GPS or external tracking systems if there is dynamic drift in different directions.

In this system, three sequential sources of error add up to substantial amounts, requiring multiple queries to gain trustful data. First, the state estimator's uncertainty, second the variance of querying anchors, and third, the estimation of the queries' variance.

Future cloud-based SLAM pipelines will have to:

1. Be more precise in real-world applications and have a standard deviation of less than a centimeter.
2. Should be transparent about the quality of the localization uncertainty for robotic systems to process the data correctly.
3. Should allow query frequencies much higher than 1Hz.
4. Should be platform-independent and accessible by a variety of devices with different sensors.

The Mixed Reality interface for the flying omnidirectional drone interacting with structures was easy to use and worked very well. While there was no formal evaluation about the usability of the Mixed Reality application developed for this project, I strongly believe that MR interfaces have a strong future in robotics. Implementing a Mixed Reality interface for a robotic system already utilizing cloud-based spatial understanding platforms was straightforward. Intuitive MR interfaces especially make sense in the case of omnidirectional drones since conventional controllers are too simple for them. There are too many controllable degrees of freedom, such that joysticks do not offer the required input agility.

To specifically answer the questions posed in the introduction of this report:

1. *Is it possible to improve a Visual Inertial Odometry (VIO) state estimators performance using cloud-based SfM pipelines, and if so, by how much?*
   Yes, it is possible, but current SfM pipelines are not precise and fast enough to replace GPS.

2. *Can other devices contribute their spatial understanding of the environment and improve the state estimation further?*
   Yes, devices such as the HoloLens 2 can successfully improve the drone's state estimation compared to drone-only configurations.

3. *Which limitations do cloud-connected computer vision systems have in robotics, what are their implications, and how can they be handled?*
   Cloud-supported pipelines must have backup solutions available if a flawless internet connectivity is not guaranteed. Furthermore, pipelines should be capable of dynamically prioritizing data in the case of limited bandwidths. Hybrid solutions, such as this project researched, might be a good approach going forward.

4. *Can the same platform be used to develop intuitive controllers for complex scenarios?*
   Yes. The implementation very straightforward, if the chosen platform provides interfaces for the desired target device.

6.1 Future Work

This project has a few parts where additional work and more thorough investigations could be done. The hardware in this project was optimized for onboard VIO pipelines and not for cloud-based SfM. Hence, valuable research would be to identify setups that are optimal for cloud-based localization in terms of image resolution, internet bandwidths, and VIO precision. Furthermore, it would be interesting to analyze the trade-off between which calculations to perform locally and which in the cloud.

A very natural extension of this work would be a complete SLAM (Simultaneous Localization and Mapping) system in the cloud. Interesting aspects are latency, performance, other requirements, and robustness.

Finally, a proper evaluation of Mixed Reality interfaces for robotic systems would be of great value. Identifying and defining future projects' guidelines to allow them to optimally control omnidirectional (and other types of robots) even more intuitively.

Bibliography

[1] M. Pantic, L. Ott, C. C. Lerma, R. Siegwart, and J. Nieto, "Mesh manifold based riemannian motion planning for omnidirectional micro aerial vehicles," *IEEE Robotics and Automation Letters*, 2021.

[2] D. Scaramuzza and Z. Zhang, *Springer Encyclopedia of Robotics, 2019.*

[3] W. Lee, K. Eckenhoff, P. Geneva, and G. Huang, "Intermittent gps-aided vio: Online initialization and calibration," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 5724–5731.

[4] T. Qin, S. Cao, J. Pan, and S. Shen, "A general optimization-based framework for global pose estimation with multiple sensors," 2019.

[5] Y. Yu, W. Gao, C. Liu, S. Shen, and M. Liu, "A gps-aided omnidirectional visual-inertial state estimator in ubiquitous environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7750–7755.

[6] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," pp. 298–304, 2015.

[7] M. Inc., "Microsoft azure spatial anchors," 2020.

[8] "Google cloud anchors," 2020.

[9] S. Ullman, "The interpretation of structure from motion," 1976.

[10] P. Speciale, S. B. Kang, M. Pollefeys, J. Schönberger, and S. Sinha, "Privacy preserving image-based localization," in *2019 Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2019.

[11] J. Delmerico, J. Nieto, and M. Pollefeys, "Uniting humans, robots via mixed reality with cloud-based localization," Oct 2020.

[12] H. Oleynikova and J. Delmerico, "Azure spatial anchors linux sdk ros wrapper," 2020.

[13] S. Y. Gadre, E. Rosen, G. Chien, E. Phillips, S. Tellex, and G. Konidaris, "End-user robot programming using mixed reality," in *2019 International conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 2707–2713.

[14] J. D. Benedict, J. D. Guliuzo, and B. S. Chaparro, "The intuitiveness of gesture control with a mixed reality device," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 63, no. 1. SAGE Publications Sage CA: Los Angeles, CA, 2019, pp. 1435–1439.

[15] A. Aldoma, M. Vincze, N. Blodow, D. Gossow, S. Gedikli, R. B. Rusu, and G. Bradski, "Cad-model recognition and 6dof pose estimation using 3d cues," in *2011 IEEE international conference on computer vision workshops (ICCV workshops)*. IEEE, 2011, pp. 585–592.

[16] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "Pvnet: Pixel-wise voting network for 6dof pose estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4561–4570.

[17] Y. He, W. Sun, H. Huang, J. Liu, H. Fan, and J. Sun, "Pvn3d: A deep point-wise 3d keypoints voting network for 6dof pose estimation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 11 632–11 641.

[18] R. König and B. Drost, "A hybrid approach for 6dof pose estimation," in *European Conference on Computer Vision*. Springer, 2020, pp. 700–706.

[19] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[20] "Robot operating system (ros), http://wiki.ros.org/."

[21] F. Tschopp, M. Riner, M. Fehr, L. Bernreiter, F. Furrer, T. Novkovic, A. Pfrunder, C. Cadena, R. Siegwart, and J. Nieto, "VersaVIS—An Open Versatile Multi-Camera Visual-Inertial Sensor Suite," *Sensors*, vol. 20, no. 5, p. 1439, 2020.

[22] M. Bischoff, "Ros# github," 2021.

[23] M. Bischoff, D. Whitney, and E. Vollenweider, "Ros# for uwp and mixed reality," 2021.

[24] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.